DISEÑO E IMPLEMENTACIÓN DE UN ANALIZADOR DE ESPECTRO EN LAS BANDAS VHF Y UHF

JUAN PABLO MAYORAL ARTEAGA
LISSETH SAAVEDRA PATIÑO
ROBERTO CARLOS PEJENDINO JOJOA

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE ELECTRÓNICA
SAN JUAN DE PASTO
2013

DISEÑO E IMPLEMENTACIÓN DE UN ANALIZADOR DE ESPECTRO EN LAS BANDAS VHF Y UHF

JUAN PABLO MAYORAL ARTEAGA
LISSETH SAAVEDRA PATIÑO
ROBERTO CARLOS PEJENDINO JOJOA

Trabajo de grado modalidad investigación presentado para optar por el título de Ingeniero Electrónico

DIRECTOR
WILSON OLMEDO ACHICANOY MARTÍNEZ
PROFESOR ASISTENTE DEL DEPARTAMENTO DE ELECTRÓNICA DELA UNIVERSIDAD DE NARIÑO

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
SAN JUAN DE PASTO
2013

“Las ideas y conclusiones aportadas en este trabajo de grado son responsabilidad exclusiva de sus autores”.

Articulo1º del Acuerdo No 324 de octubre 11 de 1966, emanado del honorable Consejo Directivo de la Universidad de Nariño.

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

_____
JURADO

_____
JURADO

San Juan de Pasto, Agosto de 2013

AGRADECIMIENTOS

Mg. CARLOS ANDRÉS VITERI MERA asesor del trabajo de grado durante gran parte de la ejecución del proyecto, por el apoyo y tiempo dedicado en las diferentes etapas de este proyecto.

Mg. WILSON OLMEDO ACHICANOY MARTÍNEZ director del trabajo de grado, por el tiempo dedicado y correcciones pertinentes.

Programa de Ingeniería Electrónica de la Universidad de Nariño, a los docentes, al personal administrativo y de laboratorio por su contribución en nuestra formación profesional.

DEDICATORIA

‘A mi madre, gracias por el apoyo incondicional y la confianza brindada a lo largo de toda mi vida. A toda mi familia y amigos’

R.C.P.J

‘A mi hermosa familia’

Liss

‘Dedicar este trabajo a la fuerza superior que mueve el universo es importante, pero puede hacerse cada día de hoy en adelante; por la vida por las ganas de continuar, por esa persona importante que puso en mi camino…

Así que dedico este trabajo principalmente a mi familia, el pilar fundamental en este camino hacia mi formación profesional. A mi madre, por demostrarme siempre su cariño y apoyo incondicional sin importar nada más. A mi padre, por brindarme su apoyo, sus ideas y su tiempo. A mi hermano por ser esa buena energía en mi vida.

Este trabajo lo dedico a mis compañeros de la U, a los que les suelo llamar los “truchas” en especial a mis “hermanos” de tesis y a sus familias porque creo que nos hemos convertido en grandes amigos…’

Juancho

RESUMEN

Con el fin de proveer a los estudiantes de ingeniería electrónica con equipamiento en el área de telecomunicaciones, se decide implantar un analizador de espectro modular para realizar prácticas de laboratorio y promover la investigación en el área de instrumentación electrónica.

En este documento se presenta el informe final del trabajo de tesis, en modalidad proyecto de investigación, titulado “Diseño e implementación de un analizador de espectro en las bandas VHF y UHF”, adscrito al Grupo de Investigación en Ingeniería Eléctrica y Electrónica (GIIEE) de la Universidad de Nariño. En él se describe la metodología seguida durante el proceso de desarrollo del proyecto y los principales resultados obtenidos.

El desarrollo del proyecto se hace en cuatro etapas consecutivas: (i) Revisión bibliográfica y de antecedentes, (ii) Diseño del analizador de espectro, (iii) Implementación del prototipo y (iv) Resultados. Finalmente se realizan conclusiones y recomendaciones para trabajos futuros relacionados con el tema de investigación.

Al culminar el proyecto se da cumplimiento a los objetivos, obteniendo como producto final un prototipo modular de un analizador de espectro en las bandas VHF y UHF, con un diseño flexible y escalable para su utilización en proyectos de investigación futuros.

ABSTRACT

To provide electronics engineering students with equipment in the telecommunications area, we decided to implement a modular spectrum analyzer for laboratory practices and promote research in the electronic instrumentation area.

This document presents the thesis work final report, under research modality, entitled “Diseño e implementación de un analizador de espectro en las bandas VHF y UHF” attached to the Universidad de Nariño Research Group in Electrical and Electronic Engineering (GIIEE). It describes the methodology followed during the project development process and the main results obtained.

The project development is done in four consecutive stages: (i) Literature review, (ii) spectrum analyzer design, (iii) prototype implementation and (iv) Results. Finally, some conclusions and recommendations for future work related to the research topic.

Upon completion of the project, we fulfills the objectives, obtaining as a final product a modular spectrum analyzer prototype in the VHF and UHF, we get a flexible and scalable design for use in future research projects.

CONTENIDO

Pág.

LISTA DE FIGURAS

Pág

LISTA DE TABLAS

Pág

LISTA DE ANEXOS

Pág

GLOSARIO

Aliasing: es el efecto que causa que señales continuas distintas se tornen indistinguibles cuando se muestrean digitalmente.

Analizador vectorial: es un instrumento de medición de señales electrónicas usualmente de radio frecuencia, que reemplaza el analizador de espectro como instrumento de medición. Ideal para las medidas de señales rápidas de gran ancho banda o espectro extendido.

Balun: del inglés *balanced-unbalanced lines transformer*, es un dispositivo adaptador de impedancias que convierte líneas de transmisión simétricas en asimétricas.

Bits de precisión: se refiere al número de bits contenidos en una palabra en informática, es importante al momento de diseñar una arquitectura de computadores, los ordenadores modernos normalmente tienen un tamaño de palabra de 16, 32 o 64 bits. Muchos otros tamaños se han utilizado en el pasado, como 8, 9, 12, 18, 24, 36, 39, 40, 48 y 60 bits.

Bluetooth: es una especificación industrial para redes Inalámbricas de área Personal, que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda de 2.4 GHz.

Circuito de tanque resonante: es un circuito formado por un inductor y un capacitor, con el fin de aprovechar las propiedades de resonancia eléctrica en una frecuencia específica denominada de resonancia para crear un oscilador.

Conversión pipeline: se refiere a la acción de llevar una señal análoga al dominio digital con el uso de la tecnología pipelining que es un método por el cual se consigue aumentar el rendimiento de algunos sistemas electrónicos digitales. En programación se aplica incrementando el número de caminos en que los cálculos deben ser registrados o sincronizados con el reloj para que el retardo computacional entre dos registros de reloj se reduzca.

Piezoelectricidad: es un fenómeno presentado por determinados cristales que al ser sometidos a tensiones mecánicas adquieren una polarización eléctrica en su masa, apareciendo una diferencia de potencial y cargas eléctricas en su superficie. Este fenómeno también se presenta a la inversa, esto es, se deforman bajo la acción de fuerzas internas al ser sometidos a un campo eléctrico. El efecto piezoeléctrico es normalmente reversible: al dejar de someter los cristales a un voltaje externo o campo eléctrico, recuperan su forma.

DBR: del inglés, *Dual Behavior Resonator,* resonadores de comportamiento dual, su estructura básica es creada usando dos elementos resonantes, con impedancias Z_1 y Z_2 en configuración paralelo.

DEP: densidad espectral de potencia, es una función matemática que nos informa de cómo está distribuida la potencia de una señal sobre las distintas frecuencias de las que está formada, es decir, su espectro.

DFT: del inglés, *Discrete Fourier Transform,* transformada discreta de Fourier, es un tipo de transformada discreta utilizada en el análisis de Fourier. Transforma una función matemática en otra, obteniendo una representación en el dominio de la frecuencia, siendo la función original una función en el dominio del tiempo.

DSP: del inglés, *Digital Signal Processing,* procesamiento digital de señales, es la manipulación matemática de una señal de información para modificarla o mejorarla en algún sentido. Este está caracterizado por la representación en el dominio del tiempo discreto, en el dominio frecuencia discreta, u otro dominio discreto de señales por medio de una secuencia de números o símbolos y el procesado de esas señales.

Sistema embebido: es un sistema de computación diseñado para realizar una o algunas funciones dedicadas, frecuentemente en un sistema de computación en tiempo real. Al contrario de lo que ocurre con los ordenadores de propósito general que están diseñados para cubrir un amplio rango de necesidades, los sistemas embebidos se diseñan para cubrir necesidades específicas.

Factor de ruido: es la magnitud del ruido generado por un dispositivo electrónico

FFT: del inglés, *Fast Fourier Transform,* transformada rápida de Fourier, permite calcular la transformada de Fourier discreta y su inversa.

FFTSA: del inglés, *FFT-based spectral analysis,* analizador de espectro basado en el algoritmo de la transformada rápida de Fourier.

Fotolitografía: es un proceso empleado en la fabricación de dispositivos semiconductores o circuitos integrados. El proceso consiste en transferir un patrón a una oblea. El silicio, en forma cristalina, se procesa en la industria en forma de obleas. Las obleas se emplean como sustrato litográfico, no obstante existen otras opciones como el vidrio, zafiro, e incluso metales.

FPGA: del inglés *Field Programmable Gate Array,* arreglo de compuertas lógicas programables en campo, es un dispositivo lógico, que contiene dos arreglos dimensionales de celdas lógicas e interruptores programables.

IBW: del inglés, *Instantaneous BandWidth,* ancho de banda instantáneo, se refiere a la anchura de la banda en la que todos los componentes de frecuencia pueden ser capturados y analizados simultáneamente.

IFFT: del inglés, *Inverse Fast Fourier Transform,* transformada inversa de Fourier, se usa para convertir una señal del dominio de la frecuencia al dominio del tiempo.

Interferencias de co-canal: La reutilización de frecuencias implica que en un área de cobertura dada haya varias celdas que usen el mismo conjunto de frecuencias. Estas celdas son llamadas celdas co-canales, y la interferencia entre las señales de estas celdas se le llama interferencia co-canal.

Admitancia: admitancia (Y) de un circuito es la facilidad que este ofrece al paso de la corriente. De acuerdo con su definición, la admitancia es la inversa de la impedancia, (Z) : $\mathrm{Y} = 1/\mathrm{Z}$. En el sistema internacional, su unidad es el Siemens.

Circuitos monolíticos: están fabricados en un cristal de una sola pieza, habitualmente de silicio, pero también existen en germanio, arseniuro de galio, silicio-germanio, etc.

OCXO: es un oscilador que utiliza una cámara de temperatura controlada para prevenir cambios en la frecuencia, debido a variaciones ambientales de temperatura, ya que mantiene el oscilador de cristal a una temperatura constante.

PLL: del inglés, *Phase-locked loops*, se trata de un sistema en el que la frecuencia y la fase son realimentados para generar relojes digitales programados.

Script: en informática un guión, archivo de órdenes o archivo de procesamiento por lotes, es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano. El uso habitual de los guiones es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario.

SSA: del inglés, *Swept Spectrum Analyzer,* analizador de espectro análogo de barrido.

Tecnología determinística: Se refiere al modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores.

TEM: transmisión electromagnética

UMTS: del inglés, *Universal Mobile Telecommunications System,* sistema universal de telecomunicaciones móviles, es una de las tecnologías usadas por los móviles de tercera generación.

VHDL: del inglés, *VHSIC Hardware Description Language,* es un lenguaje de programación que es usado para la descripción de hardware, las descripciones VHDL

pueden ser sintetizadas por FPGAs y por circuitos integrados de aplicación específica.

VHSIC: del inglés**,** del inglés, *very-high-speed integrated circuits,* Circuito Integrado de Muy Alta Velocidad.

Wi-Fi: es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica. Los dispositivos habilitados con Wi-Fi, tales como: un ordenador personal, una consola de videojuegos, un smartphone o un reproductor de audio digital, pueden conectarse a Internet a través de un punto de acceso de red inalámbrica. Dicho punto de acceso tiene un alcance de unos 20 metros en interiores y al aire libre una distancia mayor. Pueden cubrir grandes áreas la superposición de múltiples puntos de acceso.

INTRODUCCIÓN

En el año 2011 el departamento de electrónica ofreció la primera convocatoria de investigación estudiantil “José Dolores Rodríguez”, cuyo objetivo principal es dotar los laboratorios del programa de Ingeniería Electrónica con algunos equipos necesarios para el desarrollo de actividades académicas y de investigación. Estos equipos son diseñados e implementados por los estudiantes a través de proyectos de investigación en las áreas críticas del programa. Se observa que una de estas áreas es la de comunicaciones y que a la fecha no se cuenta aún con los equipos suficientes de laboratorio para el desarrollo de prácticas. La falta de estos equipos se debe principalmente a que son de difícil adquisición por parte de la facultad, principalmente por su costo.

Como parte fundamental de la formación de estudiantes en el área de comunicaciones, el programa requiere de dispositivos que permitan realizar el análisis de señales radioeléctricas en campo y en tiempo real, y de aquellas de frecuencias altas, que se generan por enlaces de comunicaciones de telefonía celular, redes de área local inalámbrica y dispositivos Bluetooth®, principalmente. Este análisis, basado principalmente en el análisis de Fourier, requiere la visualización del contenido de frecuencia de las señales y la medida de su densidad espectral de potencia (DEP).

En este proyecto se propone el diseño e implementación de un prototipo de analizador de espectro que permite calcular la DEP y visualizar el contenido de frecuencias de señales en las bandas de muy alta frecuencia (*Very High Frequency*, VHF) y ultra alta frecuencia (*Ultra High Frequency*, UHF). Por medio de este proyecto se pretende mejorar el nivel de aprendizaje experimental de los estudiantes del programa y fomentar la investigación en las áreas de señales y comunicaciones en el departamento de electrónica. Se elige este prototipo porque permite hacer prácticas académicas e investigativas en torno al análisis de modulaciones, radiodifusión, propagación, interferencia, análisis de redes inalámbricas y prueba de antenas, entre otras.

En los sectores académico y productivo mundiales se pueden encontrar distintos tipos de analizadores de espectro, basados en distintas tecnologías, y que funcionan para una gama amplia de frecuencias. Es en el sector académico donde mayormente se ha investigado y desarrollado distintos tipos de algoritmos, teóricos y basados en computadora y en sistemas embebidos, para el cálculo del espectro y de la DEP de las señales. Estos algoritmos son la base de varios equipos que se encuentran en el mercado.

1. PLANTEAMIENTO DEL PROBLEMA

1.1 DESCRIPCIÓN DEL PROBLEMA

La formación integral de ingenieros electrónicos en la universidad de Nariño implica la evolución del aprendizaje tanto a nivel teorico, como a nivel práctico. A través de los años, el departamento de electrónica se ha preocupado por la contínua adecuación de intalaciones, formación avanzada de docentes y mejoramiento de los equipos de laboratorio, todo dentro de las posibilidades que los recursos existentes le permiten.

Es claro que dichos recursos son en gran medida insuficientes y el ritmo de evolución de la tecnología en el ámbito de la ingeniería tiene un crecimiento acelerado. El acceso a dichas actualizaciones implica una inyección cosiderable de capital, sí lo que se desea es comprar dichas tecnologías, pero es importante tanto para la utilización optima de recursos, como para el fomento de la investigación, aprenidizaje y formación integral del recurso humano del departamento; el diseño y construcción de dispositivos de tecnología avanzada, mediante la explotación del conocimiento teorico que se tiene y que constantemente se renueva.

1.2 ANTECEDENTES

Algunos de los trabajos más relevantes, identificados en la etapa de revisión bibliográfica y antecedentes, en el área de diseño e implementación de analizadores de espectro se citan a continuación.

El trabajo presentado en [1] es la implementación de un analizador de espectro de ancho de banda reducido, se exploran los conceptos matemáticos, de programación y el hardware asociado al procesamiento digital de señales (*Digital Signal Processing*, DSP). Con relación al área matemática, el trabajo describe el algoritmo para el cálculo de la transformada rápida de Fourier (*Fast Fourier Transform*, FFT), que es esencial, para el análisis espectral con dispositivos de tecnología digital. También, describe ampliamente los dispositivos DSP y sus características internas, proporcionando los fundamentos para la selección apropiada de componentes que permiten alcanzar los objetivos de diseño trazados. Finalmente, detalla el proceso necesario para la programación tanto del dispositivo DSP, como de una computadora, que realiza las funciones de control y visualización.

El trabajo [2] el diseño de un analizador de espectro de baja frecuencia y ancho de banda reducido, enfocado al trabajo con redes de voltaje monofásicas de 120 V a 60 Hz. Este analizador de espectro brinda a la Universidad de San Buenaventura seccional Medellín una herramienta de trabajo con posibilidades de implementarse en sus laboratorios para el estudio y comprensión de temas de procesamiento de señales analógicas y digitales, especialmente en lo relacionado con el análisis del comportamiento de señales en función de la frecuencia. Como en este caso muchos

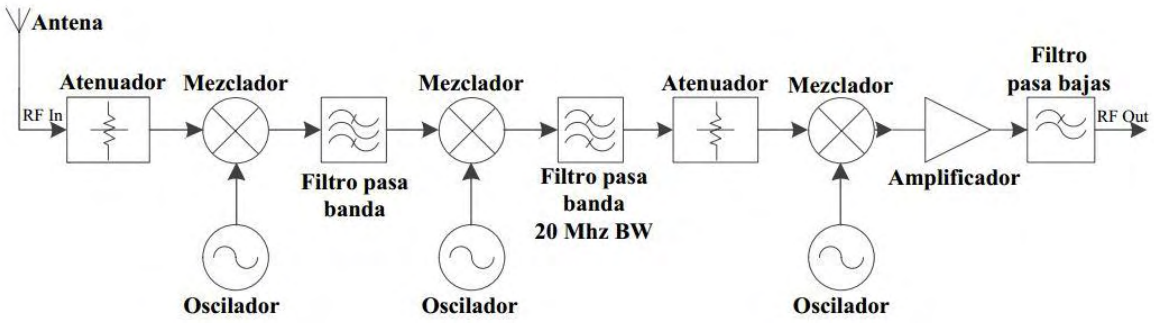
de los analizadores de espectro implementan un algoritmo de FFT para llevar a cabo sus mediciones, las cuales se visualizan en una pantalla y por lo general pueden ser descargadas en una computadora para generar registros, tablas, gráficas y otras clases de análisis. Para desarrollar el analizador se diseñó una tarjeta de adquisición de datos para la digitalización de la señal por medio de un micro controlador, el cual utiliza una interfaz serial RS-232 para transmitir la señal a una computadora donde se realiza el análisis de Fourier para finalmente visualizar la señal y los resultados del análisis en un entorno implementado en LabView.

El uso de sistemas de desarrollo embebidos, basados en *Field Programmable Gateway Array* (FPGA), al diseño y construcción de analizadores de espectro se describe en [3]. En este caso, el diseño y construcción del analizador se hace para el espectro de baja frecuencia y para el análisis de mediciones de distorsión armónica total en líneas monofásicas. El desarrollo de analizadores basados en FPGA requiere el desarrollo y la verificación de un algoritmo de transformada de Fourier, con programación parametrizable en el lenguaje VHDL, tal y como se hace en [4].

El documento presentado en [5] nos indica especificaciones del analizador vectorial de señales de características modulares PXI-5661 de National Instruments, puede ser utilizado como analizador de espectro, trabaja en frecuencias desde 9 KHz a 2.7 GHz, con ancho de banda de 20 MHz muestra señales en tiempo real, el puerto de conexión al computador es una mejora al puerto de interconexión periférico (*Peripheral Component Interconnect,* PCI), dicho puerto se denomina PXI (*PCI Extensions For Instrumentation*) controlado por el software LabView de la misma compañía. Algunas de las características que posee este dispositivo son compatibles con el prototipo del proyecto que está en desarrollo, por ende este documento se constituye en una guía.

El analizador vectorial PXI-5661 hace uso de mezcladores que tienen la función de correr el espectro hasta frecuencias que puedan ser muestreadas con un conversor análogo digital (*Analog to Digital Converter,* ADC). Por factores de costo, el proyecto está diseñado para hacer este proceso en una sola etapa, pero se puede hacer en varias, como se muestra en la Figura 1. El diagrama plantea hacer un filtrado con un ancho de banda de 20 MHz, esto con el fin de que en la implementación final se muestre en pantalla porciones del espectro con el mismo ancho de banda, lo que es suficiente para mostrar las características de las señales a estudiar.

Figura 1. Diagrama simplificado del convertidor en frecuencia que forma parte del analizador vectorial de señales.



Fuente: Basado en INTRODUCTION TO RF RECORD AND PLAYBACK, National Instruments, 16 de agosto de 2011 Pag. 2

Otro analizador es el MS2723B de la marca Anritsu™ perteneciente al departamento de electrónica de la Universidad de Nariño. [6].

El documento presentado en [46] muestra tipos, características y conceptos básicos de los analizadores de espectro de una manera clara, lo cual se tiene en cuenta para el diseño del proyecto, específicamente se indican las características de los Hameg HM5006 que se utilizan en la Universidad de Cantabria en el departamento de ingeniería de comunicaciones.

Teniendo en cuenta los requerimientos del prototipo de analizador de espectro que se quiere obtener, y los prototipos analizados en la etapa de revisión bibliográfica y referencias, en este proyecto se decide diseñar y construir un prototipo de analizador de espectro basado en un receptor superheterodino y un sistema embebido que utiliza una FPGA. La visualización de los resultados se hace a través de una computadora portátil.

1.3 JUSTIFICACIÓN

El análisis espectral de potencia es una herramienta fundamental para comprender parámetros de las comunicaciones y el procesamiento de señales, por ello se requiere la implementación de un analizador de espectro en la Universidad de Nariño, de gran utilidad para el desarrollo de prácticas académicas e investigativas. El desarrollo de este proyecto se concentra en dar solución a la problemática relacionada con la dificultad de observar el espectro de frecuencias de señales electromagnéticas en el espacio libre. Se busca un análisis de alta frecuencia, que incluya idealmente las bandas de uso libre de hasta 2.7 GHz y un ancho de banda al menos 20 MHz, que nos permita analizar todo tipo de comunicaciones inalámbricas que funcionen en este rango: radiodifusión, televisión, telefonía celular, telemetría, redes de área local inalámbricas (*Wireless Local Area Network*, WLAN), etc.

El analizador de espectro permite realizar gran variedad de medidas con un único instrumento, por lo que es indispensable para cualquier laboratorio de comunicaciones. Su presencia, en muchos casos, evita la adquisición de un completo y costoso equipamiento específico para cada tipo de medida, es posible realizar la medición de parámetros tales como: potencia de las señales, potencia de ruido, medidas de relación señal a ruido, frecuencia, distorsión, medidas relativas a modulaciones de todo tipo, pureza espectral e intensidad de campo electromagnético.

Al tener contacto con equipos de alta tecnología desarrollados en la Universidad, los estudiantes pueden comprender el funcionamiento y diseño de la instrumentación para ingeniería de una manera más sencilla, además de resolver sus necesidades académicas en el laboratorio. Asimismo, los equipos pueden ser analizados y mejorados por medio de trabajos de la comunidad estudiantil, lo que motiva a los estudiantes y a los interesados en esta área del conocimiento de la ingeniería de telecomunicaciones.

El analizador de espectro puede ser utilizado para una gran cantidad de aplicaciones específicas, entre las cuales está el diseño de redes de radiofrecuencia que respeten los lineamientos de seguridad y salud pública, así como el análisis de interferencias de co-canal, canal adyacente y análisis de cobertura. También puede analizar diferentes métodos de modulación digital, especialmente con los estándares comerciales tales como el Sistema Universal de Telecomunicaciones Móviles (*Universal Mobile Telecommunications System*, UMTS) y su extensión de Acceso de Paquetes de Alta Velocidad (*High-Speed Packet Access*, HSPA), las redes de área local inalámbrica Wi-Fi y las conexiones inalámbricas Bluetooth. Los analizadores de espectro están siendo utilizados para identificar y medir la interferencia en las señales, a menudo son necesarios para el monitoreo de operaciones en las torres de telecomunicaciones, estaciones de televisión, los sistemas de guía de los aeropuertos. etc. Una aplicación específica que concentra los esfuerzos del equipo de trabajo de este proyecto es la apropiación de los conocimientos prácticos que pueden llegar a obtener los estudiantes de ingeniería electrónica en las áreas afines a las telecomunicaciones de la Universidad de Nariño.

1.4 OBJETIVOS

1.4.1 Objetivo general

Diseñar e implementar un prototipo de un analizador de espectro mediante el uso de tecnologías digitales reconfigurables, que permita obtener la densidad espectral de potencia de señales con frecuencia central en el rango VHF y UHF.

1.4.2 Objetivos específicos

- Analizar las soluciones existentes para el cálculo y la visualización de la densidad espectral de potencia de señales en diferentes bandas del espectro electromagnético.
- Analizar las tecnologías digitales reconfigurables aplicables a la implementación de un analizador de espectro y elegir la más viable para el proyecto.
- Diseñar un analizador de espectro que permita obtener la densidad espectral de potencia de señales en las bandas VHF y UHF.
- Simular los diferentes bloques del diseño con el fin de estimar sus parámetros de funcionamiento, teniendo en cuenta las posibles soluciones que se tengan en cada etapa.
- Implementar un prototipo de analizador de espectro.
- Probar y validar el funcionamiento del prototipo por medio de comparaciones con las simulaciones y con el funcionamiento de un analizador de espectro comercial.

1.5 ALCANCES

Se diseña e implementa un analizador de espectro haciendo uso del arreglo de compuertas lógicas programables en campo (*Field Programmable Gate Array*, FPGA) de la serie Cyclone II EP2C35F672C6 [8], [9] de Altera como núcleo de procesamiento, los resultados se pueden visualizar en tiempo real, posee las capacidades de desplegar en pantalla la densidad espectral de potencia de una señal. Esta función tiene una resolución suficiente para observar con claridad las componentes espectrales de potencia y permite manipulación digital de la visualización; por ejemplo hacer acercamientos y hacer captura de pantalla en un instante de tiempo determinado.

El analizador de espectro trabaja en la banda de frecuencias hasta 2.7 GHz que incluye la banda Industrial, Científica y Médica (*Industrial, Scientific and Medical, ISM*) de 2.4 GHz, permite medir un ancho de banda de 20MHz. Se realizan los códigos de programación que calculan la FFT utilizando la herramienta de software Quartus II provista por Altera, se utiliza el lenguaje de programación VHDL (*VHSIC* [1] *Hardware Description Language*), además se implementan métodos para mejorar la relación señal a ruido de las mediciones que permitan mejor visualización.

El producto del trabajo es un prototipo modular de un analizador de espectro.

[1] *very-high-speed integrated circuits.*

2. MARCO TEÓRICO

2.1 ANALIZADOR DE ESPECTRO

En su forma más básica, la tarea fundamental de un analizador de espectro es medir la potencia de señal frente a la frecuencia. En tiempos pasados, esto se logró casi exclusivamente con el analizador de espectro análogo de barrido (*Swept Spectrum Analyzer*, SSA). Sin embargo, la disponibilidad de alta velocidad y alto rango dinámico del ADC, junto con una alta velocidad de DSP ha provocado cambios dramáticos en la arquitectura del analizador de espectro. La mayor parte del procesamiento de señal a cargo del SSA como la resolución de ancho de banda (*Resolution BandWidth*, RBW), ahora se puede hacer de forma digital, mejorando el rendimiento y reduciendo los requerimientos de calibración. El crecimiento de la tecnología DSP no sólo ha mejorado las características técnicas del SSA, sino también ha conducido al desarrollo y la prevalencia del analizador de espectro basado en la FFT (*FFT-based spectral analysis,* FFTSA). Por estas razones FFTSA se ha convertido en el método de elección para la mayoría de las implementaciones.

2.1.1 Arquitectura

Ambas arquitecturas SSA y FFTSA pueden ser descritas por el diagrama de bloques de la Figura 2. La señal de entrada se aplica primero al módulo del convertidor-reductor de frecuencia analógica, cuya función principal es traducir la señal de entrada a una frecuencia intermedia (*Intermediate Frequency*, IF) a un nivel de potencia adecuado para el procesamiento posterior. El modulo del oscilador local (*Local Oscillator*, LO), proporciona una señal sinusoidal al módulo convertidor-reductor, que se mezcla con la señal de entrada produciendo el desplazamiento de frecuencia deseado a la IF. En una SSA, la señal sinusoidal es barrida linealmente sobre la banda de frecuencia o intervalo a medir. En una FFTSA, el módulo LO proporciona una frecuencia de paso de la señal sinusoidal, cuyo tamaño de paso se determina por la cobertura de frecuencia de la FFT.

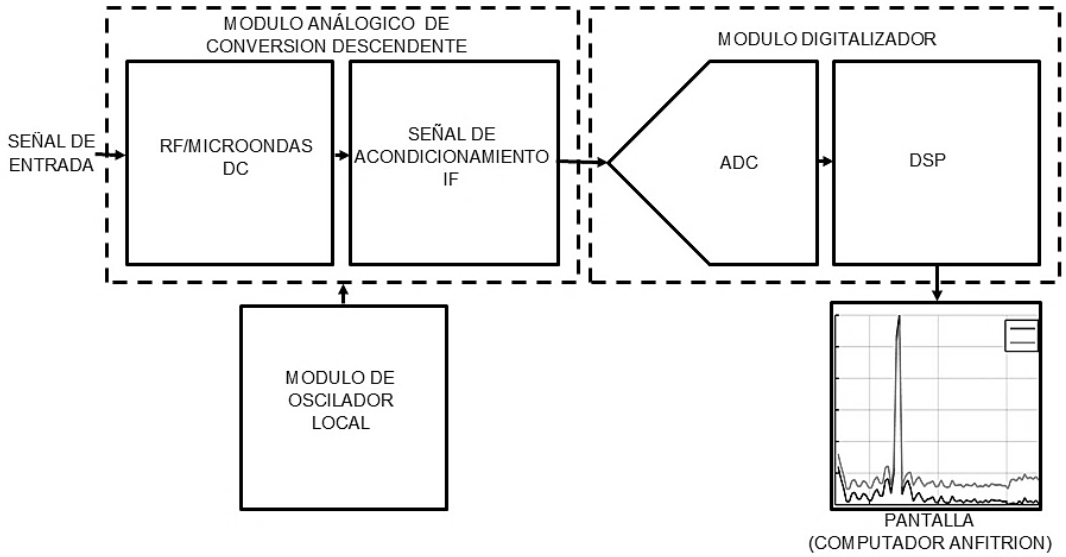
El módulo digitalizador convierte la salida analógica del módulo convertidor-reductor de frecuencia al dominio digital. También puede contener los algoritmos DSP complejos utilizados para producir una medición. Esto incluye operaciones tales como la conversión digital descendente (*Digital DownConversion*, DDC) en banda base, FFT, promedio y disparo. El ordenador central puede realizar algunos, todos o ninguno de los algoritmos necesarios en función de la capacidad del digitalizador. Los resultados de medición de datos en bruto se transfieren a un ordenador central, para su posterior procesamiento y visualización. El digitalizador desempeña un papel clave en la determinación de la velocidad del sistema. Si el digitalizador contiene un DDC y un remuestreador, la cantidad de datos a transferir al sistema principal puede ser minimizada, lo cual resulta en rapidez de operación. Cuando se implementa con

FPGA, se procesan señales de transmisión desde el convertidor-reductor de frecuencia en tiempo real.

2.1.2 Rango dinámico

El rango dinámico es la relación, expresada en decibeles (dB), de las señales más grandes a las más pequeñas, presentes simultáneamente en la entrada del analizador de espectro, permite la medición de la señal menor a un determinado grado de incertidumbre. Se considera el efecto del rango dinámico del ADC en todo el sistema.

Figura 2. Analizador de espectro, diagrama de bloques simplificado

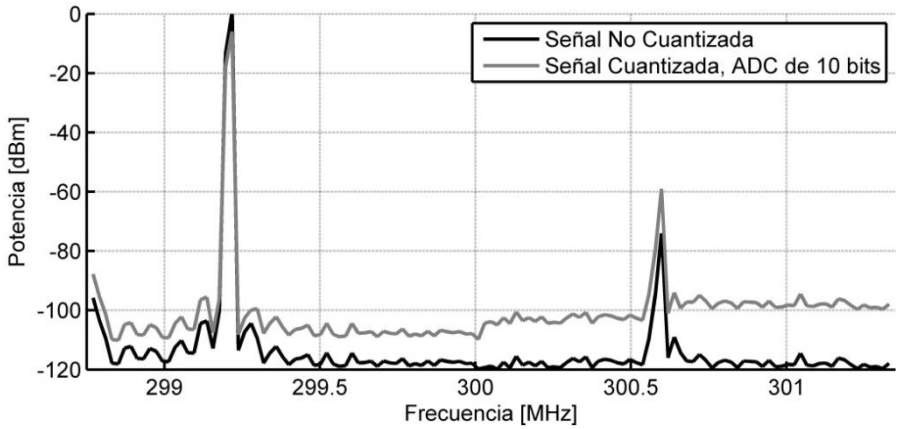


Fuente: *basado en FUNDAMENTALS OF MODERN SPECTRAL ANALYSIS, Matthew T. Hunter, Achilleas G. Kourtellis†, Christopher D. Ziomek‡ and Wasfy B. Mikhael, 2010. Pag 17*

Se ha afirmado que la implementación digital de las SSAs proporciona un mejor rendimiento de rango dinámico que las FFTSAs. El argumento es el siguiente. Un analizador de espectro de barrido puede tener al final un ancho de banda de IF muy estrecho, por lo tanto se puede utilizar un control automático de ganancia (*Automatic Gain Control*, AGC) para ajustar el nivel de señal en el ADC tal que la señal es cuantizada con el número máximo de bits. Este ajuste de nivel se realiza también en las FFTSAs, sin embargo, ya que el ancho de banda es típicamente más amplio, la probabilidad de que una señal muy fuerte pueda encontrarse con una señal débil se incrementa. En este caso, los bloques de señal fuerte o máscaras de la señal débil se pueden cuantificar con sólo unos pocos bits y terminan en el ruido. Esto se muestra en la Figura 3 para un ADC de 10-bit.

Para atenuar este problema, se pueden tomar enfoques diferentes. La primera técnica es el uso de AGC como se ha descrito anteriormente, un método típico de la SSA. Cabe señalar que también se puede utilizar esta técnica con una FFTSA,

Figura 3. Señal fuerte bloqueando la señal más débil debido a la cuantización, ADC de 10 bits.



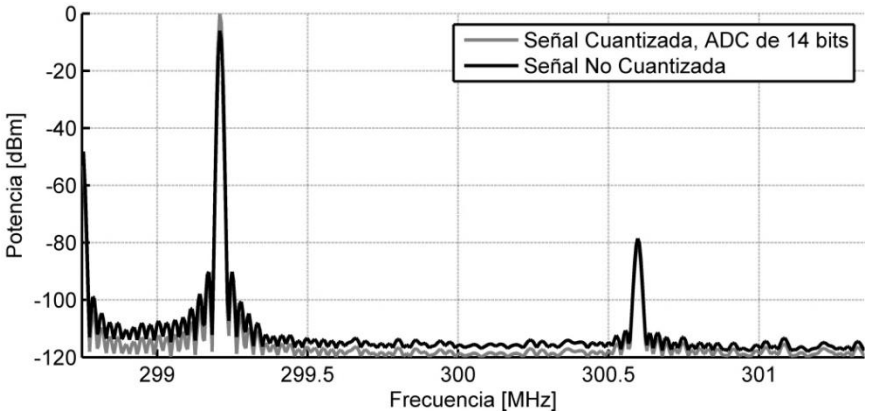
Fuente: *basado en FUNDAMENTALS OF MODERN SPECTRAL ANALYSIS, Matthew T. Hunter, Achilleas G. Kourtellis†, Christopher D. Ziomek‡ and Wasfy B. Mikhael, 2010. Pag 18*.

no hay ninguna limitación inherente que lo impida. Sin embargo, no es necesario y de hecho se suma a la complejidad del sistema; alternativamente, la IF puede ser dividida en una ruta de acceso de banda estrecha conectada a un ADC de alta resolución y una ruta de acceso de banda ancha conectada a un ADC de alta velocidad. No obstante, esto también complica el sistema mediante la adición de una ruta de señal adicional y sólo se debe utilizar en las aplicaciones más exigentes.

Puesto que la tecnología ADC ha mejorado drásticamente en la última década, es posible la digitalización de anchos de banda del orden de cientos de Mega hercios con 14-bits de precisión con una única ruta que pueda ofrecer el ancho de banda y rango dinámico adecuados. Si las dos señales anteriores se comparan de nuevo, esta vez usando 14-bits en lugar de 10, no se producen bloqueos. Esto se puede observar en la Figura 4.

Como se puede deducir del análisis anterior, no hay necesidad de utilizar SSA para tener un excelente rango dinámico. La FFTSA proporciona los mismos o mejores resultados usando hardware flexible más genérico. Por lo tanto, nos centramos en la FFTSA.

Figura 4. Señal fuerte bloqueando la señal más débil debido a la cuantización, ADC 14 bits.



Fuente: *basado en FUNDAMENTALS OF MODERN SPECTRAL ANALYSIS, Matthew T. Hunter, Achilleas G. Kourtellis, Christopher D. Ziomek and Wasfy B. Mikhael, 2010. Pag 18.*

2.1.3 Ancho de banda instantáneo

Ancho de banda instantáneo (*Instantaneous BandWidth,* IBW) se refiere a la anchura de la banda en la que todos los componentes de frecuencia pueden ser capturados y analizados simultáneamente. IBW también se llama análisis IF, modulación IF, o ancho de banda IF. Un IBW más grande tiene muchos beneficios, dos de los cuales son un aumento en la velocidad de medición para grandes lapsos y un aumento en la flexibilidad del sistema.

2.2 TRANSFORMADA RÁPIDA DE FOURIER

El análisis de Fourier, nombrado en honor a Jean Baptiste Joseph Fourier, es un proceso en el cual una señal es descompuesta en ondas sinusoidales con variaciones en frecuencia y amplitud. Esto se ejecuta para facilitar operaciones matemáticas a una señal. Existen dos tipos de señales que establecen el tipo de análisis. Si una señal es periódica, se emplea la serie de Fourier. Si la señal es continua y no periódica, se aplica la transformada de Fourier. Para señales muestreadas periódicas y no periódicas se usa la transformada discreta de Fourier (*Discrete Fourier Transform*, DFT).

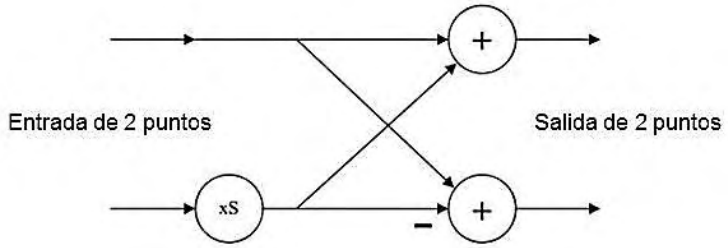
La DFT puede ser expresada como una ecuación simple. En esta ecuación k es evaluada de 0 a $N/2$. Esta forma de DFT solamente muestra la mitad positiva del rango de frecuencias, pero comúnmente es representado desde $-N/2$ hasta $N/2$. Donde N es el número de muestras.

$$X[k] = \sum_{i=0}^{N-1} x[i] \cos\left(\frac{2\pi k i}{N}\right) - \sum_{i=0}^{N-1} x[i] sen\left(\frac{2\pi k i}{N}\right) \quad (1)$$

La implementación directa de la anterior fórmula resulta en una ejecución en tiempo que es proporcional al número de muestras elevadas al cuadrado (N^2).

La FFT es un algoritmo que simplifica la DFT, mediante el procesamiento de xS en un orden diferente y direccionando la operación conforme a la estructura que se muestra en la Figura 5.

Figura 5. Bloque de construcción básico de algoritmos de FFT



Fuente: *Basado en DEVELOPMENT AND VERIFICATION OF PARAMETERIZED DIGITAL SIGNAL PROCESSING MACROS FOR MICROELECTRONIC SYSTEMS, Miller Adam Robert. Thesis presented for the Master of Science, University of Tennessee, Knoxville. Agosto 2003, pag.10*

Es posible calcular la FFT con una ejecución en tiempo proporcional a $Nlog_2(N)$ la estructura mostrada en la Figura 5 es llamada mariposa y es el bloque de construcción básico de la mayoría de algoritmos FFT. Por su forma, puede ser usada para convertir una señal del dominio de la frecuencia al dominio del tiempo, lo que se denomina transformada rápida de Fourier inversa (*Inverse Fast Fourier Transform*, IFFT).

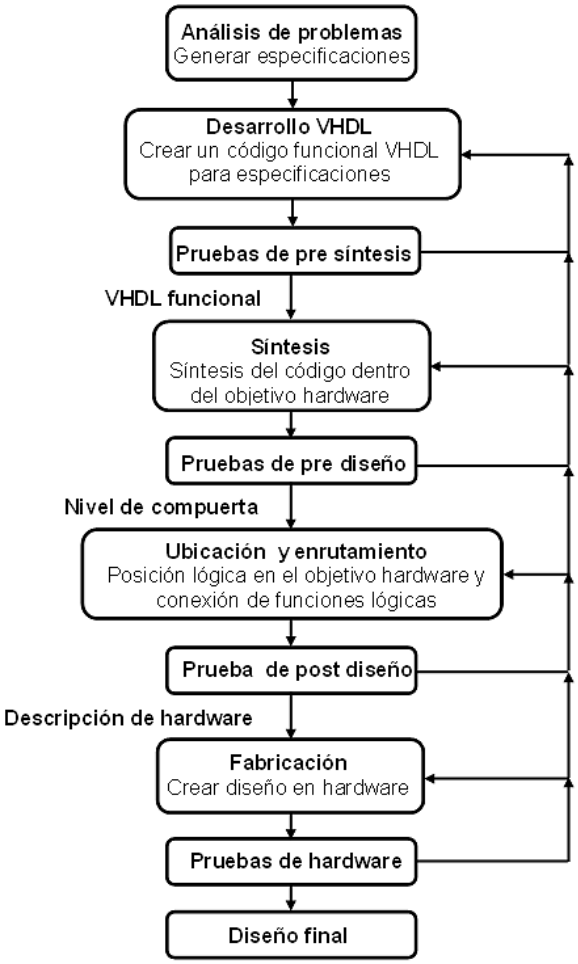
2.3 VHDL Y FLUJO DE DISEÑO

VHDL es un lenguaje de programación que es usado para la descripción de hardware. Está diseñado para VHSIC. Descripciones VHDL pueden ser sintetizadas por FPGAs y por circuitos integrados de aplicación específica (*Application Specific Integrated Circuits, ASICs*). Las FPGAs son comúnmente diseñadas para ser reprogramables, así que usualmente son usadas para probar algoritmos. ASICs son chips que son diseñados para un propósito específico y no son reprogramables, pero son más rápidos que las FPGAs.

La Figura 6 muestra un diagrama de flujo usado para el diseño de circuitos digitales. En primer lugar, las especificaciones para el diseño son creadas analizando los requerimientos del problema, luego el código es creado para cumplir las especificaciones. En este ejemplo, el código está escrito en VHDL y probado para verificar si es funcionalmente correcto. Una vez el código es correcto es sintetizado. La síntesis es el proceso de trasladar el código VHDL a compuertas y funciones lógicas de una tecnología específica.

Si el código es implementado en una FPGA, los recursos de hardware son ubicados en las diferentes funciones lógicas del código, sin embargo la asignación de recursos de hardware no son especificados. Por ejemplo una FPGA puede tener 2000 compuertas NAND disponibles y cuando un diseño es sintetizado, este puede requerir 50 de ellas, las herramientas de síntesis prueban si las 50 compuertas NAND están disponibles y si han sido usadas pero no cuales específicamente. Si se utiliza tecnología ASIC la síntesis toma diferentes funciones lógicas de una librería de celdas estándar. Esta librería es creada con celdas que tienen diferentes funciones lógicas y algunas veces variantes de la misma función lógica que son diseñadas para manejar mayores cargas en las salidas. La herramienta de síntesis mantiene la trayectoria de cuales compuertas son usadas para sintetizar las diferentes funciones del código.

Figura 6. Diagrama de flujo para diseños VHDL



Fuente: *basado en DEVELOPMENT AND VERIFICATION OF PARAMETERIZED DIGITAL SIGNAL PROCESSING MACROS FOR MICROELECTRONIC SYSTEMS. Miller Adam Robert, Thesis presented for the Master of Science, university of Tennessee, Knoxville. Agosto 2003, pág. 4.*

El diseño es probado una vez más después de que este ha sido sintetizado y si esto es correcto, una lista de funciones lógicas, sus conexiones y la lista de compuertas de nivel son transferidas al paso de ubicación y enrutamiento.

Una vez el diseño haya sido sintetizado, los recursos de hardware de los anteriores pasos necesitan ser ubicados y las conexiones entre los recursos enrutados. La ubicación y enrutamiento intenta incrementar la frecuencia a la que el diseño puede correr, disminuyendo el retardo. Algunos otros objetivos durante la ubicación y el enrutamiento puede ser disminuir el hardware total de recursos usados, disminuyendo el uso de potencia o alguna combinación de los tres, después de que este proceso finaliza, se genera el plano, es verificado y el diseño puede ser realizado en hardware. Para FPGAs este es el proceso de configuración con el diseño, lo cual usualmente toma menos de un minuto. Solamente después que el hardware es probado en el diseño el proceso termina.

2.3.1 Diseño para el reúso

Es un método para el desarrollo de algoritmos que pueden ser reusados para otros proyectos. Esto tiene la ventaja de ahorrar tiempo de diseño en proyectos posteriores pero puede costar más tiempo en el desarrollo inicial del algoritmo. Además implica una reducción del número de personas involucradas por proyecto, velocidad y costos, una de las maneras para diseñar algoritmos reutilizables es crear código reconfigurable o macros. El término ‘macro’ es algunas veces aplicado a un bloque de código que es reconfigurable.

La flexibilidad del código permite mayor reutilización, sin embargo puede provocar mayor complicación de uso. Esta relación permite balancear la flexibilidad y la facilidad de manejo, una manera de mantener el código fácil de usar es proveer las características por defecto para algunos de los parámetros que permiten configuración. Un usuario que está más familiarizado con los algoritmos puede entrar y modificar estas características para la aplicación requerida.

En VHDL existen dos formas de crear código reconfigurable, el primero es usar la declaración generic. La sentencia generic puede ser descrita como una constante que puede cambiar con diferentes instanciaciones del mismo bloque de código. En otras palabras generics es la manera de especificar parámetros para una instanciación dada de un bloque de código incluso el parámetro del ancho en bit de un vector de datos. Un ejemplo del uso de generics es un contador que puede ser creado usando un generic para que se detenga en un valor determinado, cada vez que el bloque contador es llamado, un valor diferente de parada puede ser establecido al bloque.

La segunda manera para crear código reconfigurable es usar la sentencia generate, esta es manejada durante la síntesis y puede simplificar la lógica necesaria para un algoritmo. Existen dos clases de generate que pueden ser usadas para crear un código flexible, el primero es llamado generate condicional, este tipo de sentencias

permite a la lógica ser creada como síntesis si una condición existe. La ventaja principal de usar generate condicional es que permite opciones diferentes para hardware cuando se diseña el algoritmo. Las opciones no utilizadas no son sintetizadas disminuyendo la cantidad de recursos requeridos comparados a una versión que incluye lógica redundante.

El segundo tipo de sentencias generate incluye un lazo. Este tipo de generate es útil cuando una operación necesita ser separada en paralelo o en serie, ejemplos de esto incluye sumadores y multiplicadores. El lazo generate puede también ser utilizado dentro de generate condicional y viceversa. Combinado esto con generics, es posible crear código flexible y es más probable el reúso en futuros diseños.

2.4 FPGA

FPGA es un dispositivo lógico, que contiene dos arreglos dimensionales de celdas lógicas e interruptores programables. La estructura conceptual de una FPGA se muestra en la Figura 7. Una celda lógica puede ser personalizada para proveer interconexiones entre celdas lógicas. Un diseño personalizado puede ser implementado especificando la función de cada celda lógica y configurando selectivamente la conexión de cada interruptor programable. Una vez el diseño y la síntesis termina, se puede usar un simple cable adaptador para descargar el diseño.

2.4.1 Look up table (LUT)

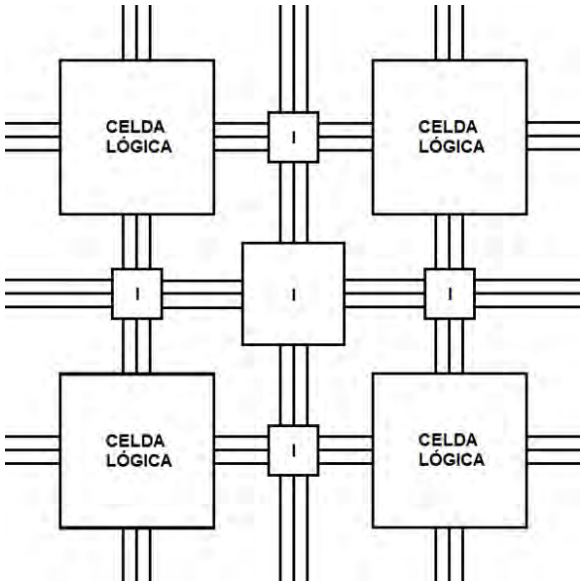
Lógica basada en celdas. Una celda lógica usualmente contiene un circuito lógico combinacional usando un flip-flop tipo D (D FF), el método más común para implementar un circuito combinacional es una LUT. Una LUT de n entradas puede ser considerada como una pequeña memoria de 2^n bits de almacenamiento, así que para escribir contenido en una memoria se puede usar LUTs e implementar una función combinacional de n entradas.

2.4.2 FPGA Cyclone II

Los dispositivos Cyclone II están compuestos por una matriz bidimensional de filas y columnas con arquitectura variable para implementar con lógica personalizada. Las interconexiones entre columnas y filas de velocidad variable proveen señales de interconexión entre bloques de arreglos lógicos (*Logic Arrange Blocks,* LABs), bloques de memorias embebidas y multiplicadores embebidos.

Los arreglos lógicos consisten de LABs, con 16 elementos lógicos (*Logic Elements,* LEs) en cada LAB. Una LE es una pequeña unidad de lógica la cual provee una implementación eficiente de las funciones lógicas de usuario. LABs son agrupadas dentro de filas y columnas a través del dispositivo. Los dispositivos Cyclone II poseen desde 4,608 a 68,412 LEs.

Figura 7. Estructura conceptual de una FPGA



Fuente: *basada en PROTOTYPING BY VERILOG EXAMPLES, Pong P. Chu, FPGA, XILINX Spartan™ -3 version, Cleveland State University. John Wiley & Sons, INC. Publication. Pag 12.*

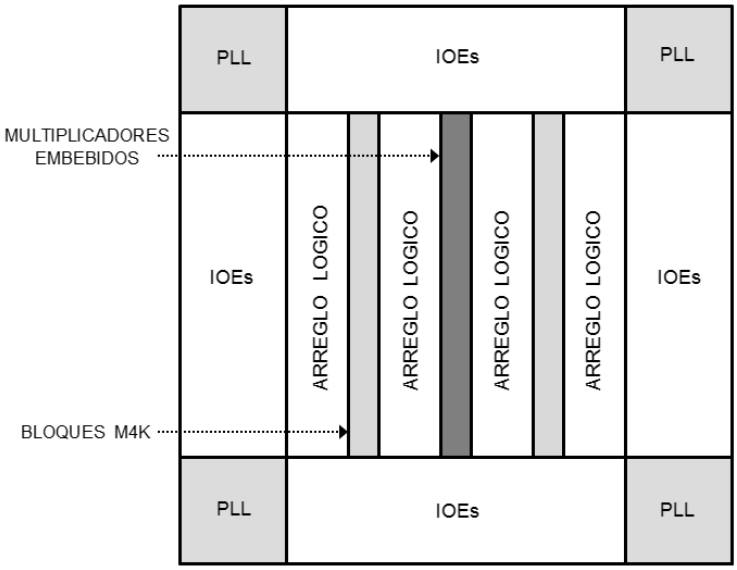
Los dispositivos Cyclone II, proveen una red global de reloj y hasta *cuatro phase-locked loops* (PLLs). La red global de reloj consiste de hasta 16 relojes globales que están dispuestos en todo el dispositivo. Dicha red puede proveer relojes para todos los elementos dentro del dispositivo, como elementos de entrada/salida (*Input-Output Elements,* IOEs), LEs, multiplicadores embebidos y bloques de memoria embebidos.

Cada pin de entrada-salida es alimentado por un IOE localizado al final de las filas y columnas LAB alrededor del dispositivo. Los pines I/O soportan varias terminaciones simples y estándares diferenciales, como el 66 y 33 MHz, 64 y 32 bits estándar PCI, puerto de interconexión periférica extendida (*Peripheral Component Interconnect eXtended,* PCI-X) y el estándar de señal diferencial de bajo voltaje (*Low-voltage differential signaling,* LVDS), a una máxima tasa de muestreo de 805 megabits por segundo (Mbps) para entradas y 640 Mbps para salidas. Cada IOE contiene un buffer I/O bidireccional y tres registros para almacenar entradas, salidas y señales de habilitación de salida.

Las interfaces de doble propósito DQS, DQ y pines de máscara de datos (*Data Mask,* DM) son usadas para alineación de fase de señales de doble rata (*Double data rate,* DDR) y proveen soporte para dispositivos de memoria externa tales como DDR, DDR2, datos de rata simple (*Simple Data Rate,* SDR), memorias dinámicas, síncronas de acceso aleatorio (*Synchronous dynamic random-access memory,* SDRAM) y dispositivos de cuádruple rata SRAM (*Quad Data Rate SRAM,* QDRII SRAM) de hasta 167 MHz.

El número de bloques de memoria M4K, bloques de multiplicadores embebidos, PLLs, filas y columnas varían según el dispositivo.

Figura 8. Arquitectura de la FPGA Cyclone II



Fuente: *basado en CYCLONE II DEVICE HANDBOOK, Altera Corporation, Volume 1. Copyright © 2008.pag 2-2.*

2.4.2.1 Elementos lógicos, LEs

Es la unidad lógica más pequeña de la arquitectura de la Cyclone II, un LE, es compacto y provee características avanzadas con una eficiente utilización lógica. Cada LE posee las siguientes características:

- Una LUT de cuatro entradas, la cual es un generador, en el que se puede implementar cualquier función de cuatro variables.
- Un registro programable.
- Una cadena de conexiones.
- Una conexión de registros en cadena.
- La habilidad de manejar todos los tipos de interconexiones: locales, filas, columnas, cadenas de registros e interconexiones directas.
- Soporte para registros embebidos.
- Soporte para registros de realimentación.

Cada registro programable puede ser configurado para operaciones D, T, JK o SR [18]. Los registros tienen datos, reloj, habilitador de reloj y reset para las entradas, las señales que se usan en la red global de reloj, pines de propósito general I/O o cualquier lógica interna puede conducir los registros de reloj y las señales de control *reset*. Bien sea los pines de propósito general I/O o la lógica interna pueden conducir

a la habilitación de reloj. Para funciones combinacionales las salidas LUT no pasan por los registros y son conducidos directamente a las salidas LE.

Cada LE tiene tres salidas que son conducidas localmente, por filas, columnas enrutándolas a los recursos. La LUT o una salida de registro pueden ser llevadas a esas tres salidas independientemente, dos salidas LE son conducidas bien sea a filas o columnas y son directamente acopladas hacia conexiones de enrutamiento, una se interconecta a recursos locales, siguiendo la LUT para conducir una salida mientras el registro conduce otra salida. Esta característica, mejora la utilización del dispositivo puesto que este puede usar el registro y la LUT para funciones no relacionadas entre sí. Cuando se usa registros embebidos, no está disponible la señal de control de carga síncrona LAB.

Además de las tres salidas de enrutamiento generales, las LEs dentro de un LAB tienen salidas de registro en cadena, que permiten que los registros se enlacen dentro de la misma LAB y que un LAB use LUTs para funciones combinacionales únicas y los registros sean usados para una aplicación de registro de desplazamiento no relacionado.

2.4.2.2 Bloques de arreglo lógico (LAB)

Cada LAB consiste de:

- 16 LEs
- Señales de control LAB
- Cadenas de registros
- Interconexiones locales

Las interconexiones locales transfieren señales entre LEs en el mismo LAB. Las conexiones de cadenas de registros transfieren las salidas de un registro del LE al adyacente dentro de una LAB. El compilador QUARTUS II sitúa la lógica asociada dentro de una LAB o LABs adyacentes, siguiendo el uso de conexiones locales y de cadenas de registros para la mayor eficiencia y rendimiento en el área de diseño.

2.4.2.3 Conexión multitrack

En la arquitectura de la Cyclone II, las conexiones entre LEs, bloques de memoria M4K, multiplicadores embebidos y pines I/O son provistos por la estructura de interconexión *multitrack* con tecnología *DirectDrive*. Las conexiones *multitrack* consisten de continuas líneas de enrutamiento optimizadas de diferentes velocidades usadas para la conectividad interior y exterior de bloques lógicos. El compilador Quartus II automáticamente ubica rutas críticas sobre las interconexiones más rápidas para mejorar el rendimiento del diseño.

La *DirectDrive* es una tecnología determinística de enrutamiento que asegura un uso de recursos de enrutamiento idéntico para cualquier función sin tener en cuenta el lugar de ubicación en el dispositivo. El *multitrack* interconecta y la tecnología *DirectDrive* simplifica la integración de las etapas de diseño basadas en bloques para eliminar ciclos de re optimización que típicamente implican cambios en el diseño y adiciones.

La interconexión *multitrack* consiste de interconexiones de filas y columnas que abarcan distancias fijas. Una estructura de enrutamiento con fuentes de longitud fija para todos los dispositivos permite rendimiento predecible y repetible cuando migra a través de diferentes densidades del dispositivo.

2.4.2.4 Multiplicadores embebidos

Los dispositivos Cyclone II tienen bloques multiplicadores embebidos optimizados para funciones de procesamiento digital, tales como filtros de respuesta finita al impulso (*Finite Impulse Response, FIR*), FFT, transformada de coseno discreto entre otras. Se puede usar multiplicadores embebidos en uno de dos modos operacionales básicos dependiendo de la aplicación requerida:

- Un multiplicador de 18 bits.
- Dos multiplicadores independientes de 9 bits.

Los multiplicadores embebidos pueden operar hasta 250 MHz para multiplicaciones de 18 x 18 y 9 x 9 cuando son usados registros de entrada y salida.

Cada dispositivo Cyclone II tiene de una a tres columnas de multiplicadores embebidos que implementan eficientemente dichas funciones. Un multiplicador embebido abarca la longitud de una fila LAB. Los multiplicadores embebidos constan de los siguientes elementos:

- Un bloque multiplicador.
- Registros de entrada y salida.
- Interfaces de entrada y salida.

2.4.2.5 Estructura y características de I/O

IOEs incluyen las siguientes características:

- Estándares diferenciales y de terminación simple I/O.
- 3.3 V, 64 y 32 bits, 66 y 33 MHz PCI.
- Soporte para *Joint Test Action Group* (JTAG), *boundary-scan test* (BST).
- Unidad de control de salida.
- *Buffers* triestado.
- Circuitería de retención de bus.

- Resistores *pull-up* programables en modo usuario.
- Retrasos programables de entrada y salida.
- Salidas de drenaje abierto.
- Pines DQ y DQS I/O.
- Pines de voltaje de referencia (V_{REF}).

Las IOEs contienen un *buffer* bidireccional I/O y tres registros para completar la transferencia de datos a una tasa de transmisión simple.

La IOE contiene un registro de entrada, un registro de salida y un registro habilitador de salida. Se pueden usar los registros de entrada para configuraciones veloces y registros de salida para mostrar relojes de alta velocidad, adicionalmente se puede usar los registros habilitadores de salida (*Output Enable,* OE) para habilitar funciones de alta velocidad. El software Quartus II automáticamente duplica un registro simple OE que controla múltiples pines de salida o bidireccionales. Se puede usar IOEs como entradas, salidas o pines direccionales.

Las IOEs están localizadas en bloques I/O alrededor de los periféricos del dispositivo, existen hasta cinco IOEs por fila y hasta cuatro IOEs por columna en cada bloque.

2.4.2.6 Quartus II

Es un software de alto rendimiento y productividad para dispositivos de lógica programable compleja (*Complex programmable logic device,* CPLD), FPGA, sistemas en chip (*Sistem on CHIP,* SoC FPGA) y diseño ASIC, Quartus II puede ser fácilmente adaptado a necesidades específicas en todas las fases del diseño en FPGA y CPLD. Las herramientas de simulación y de nivel de sistema integradas en Quartus II son las siguientes:

- ModelSim® (incluido libre o en paquetes de suscripción)
- DSP Builder (requiere licencia adicional)
- Qsys (parte de Quartus II)
- Otras herramientas de diseño electrónico automatizado (*Electronic Design Automation,* EDA)

2.4.2.6.1 **Características**:

- Compilación incremental: compila solamente los cambios del diseño, para reducir el tiempo de compilación hasta 70%.
- Analizador de potencia *PowerPlay*: existe un botón para la optimización de potencia, consigue un 10% de reducción en el consumo de energía gracias a la optimización automática.
- Analizador de tiempos TimeQuest: se trata de un analizador de tiempos de segunda generación, ofrece un ambiente de interfaz gráfica de usuario (*Graphical*

User Interface, GUI) y soporte para *scripts*, brinda relativa facilidad para modificar las restricciones de diseño (*Synopsys Design Constraints,* SDC).
- Herramienta integrada Qsys: agiliza tiempo y esfuerzo en el proceso de diseño en FPGA puesto que automáticamente genera interconexiones lógicas para comunicar subsistemas y funciones de propiedad intelectual (*Intellectual Property,* IP), entrega un alto rendimiento, mejora el diseño al aplicar reúso y una verificación más rápida comparada con SOPC Buider (*System on a Programmable Chip Builder*).

2.5 CONVERSIÓN ANALÓGA A DIGITAL

Un ADC toma un voltaje de entrada analógico y después de cierto tiempo produce un código de salida digital que representa la entrada analógica. Existen diferentes tipos de conversión A/D, una de las tecnologías más utilizadas y reconocidas para aplicaciones de alta velocidad en la conversión pipeline.

Las aplicaciones para ADCs pipeline incluyen procesamiento de imagen y video, comunicaciones, entre muchas otras, esta arquitectura ofrece costos relativamente bajos para diferentes procesos de circuitos integrados, tales como: CMOS (*Complementary Metal–Oxide–Semiconductor*) y BiCMOS (*Bipolar-CMOS*) siendo este último de gran popularidad para ADCs de alta resolución.

Estos dispositivos se basan en bloques que contienen componentes tales como ADC Flash y conversores digital-análogos (*Digital-to-Analog Converter,* DAC), se logra sintetizar una conversión de alta velocidad por adquisiciones sucesivas, En cada ciclo de reloj se tiene el resultado de una muestra, pero existe una latencia o retardo que hay que considerar.

2.5.1 Conversor análogo digital AD9481

El AD9481 [10] es un conversor análogo a digital monolítico de 8 bits de resolución, optimizado para altas velocidades y bajo consumo de potencia, posee una velocidad de muestreo de hasta 250 mega muestras por segundo (*Mega Samples Per Second,* MSPS), con excelente linealidad.

Las salidas digitales son compatibles tanto con la tecnología de transistor a transistor (*Transistor–Transistor Logic*, TTL) como CMOS, con opción para salida en complemento a dos o formato binario, los bits de datos de salida son provistos de forma intercalada junto con la salida de relojes, lo cual simplifica la captura de datos. Este circuito integrado puede ser utilizado de forma segura a temperaturas industriales desde -40 °C hasta 85 °C.

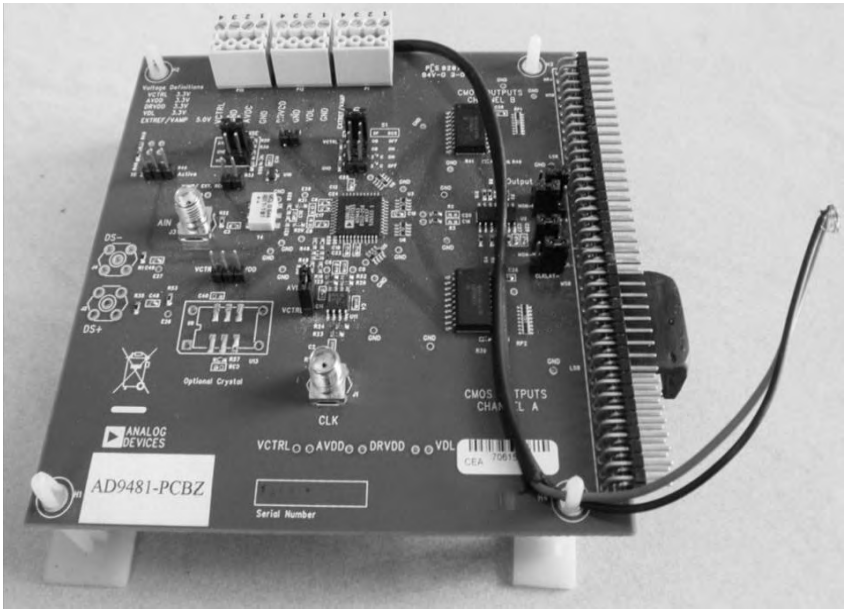
2.5.1.1 Características destacadas:

- Linealidad superior. Una no linealidad diferencial (*Differential nonlinearity,* DNL) de ± 0.35 lo hace adecuado para aplicaciones de medición e instrumentación.
- Modo de bajo consumo de energía (Power-Down). La función Power- Down puede ser ejecutada para ofrecer un consumo total de 15 mW.
- Salidas CMOS demultiplexadas que permiten una fácil interfaz con FPGAs de bajo costo y lógica convencional.
- Entradas analógicas que tienen un buffer diferencial para un mejor rendimiento dinámico, las impedancias en V_{IN+} y V_{IN-} deben estar acopladas.
- Circuito de rastreo y retención que es esencialmente un mezclador y cualquier ruido, distorsión o saltos temporales en el reloj se combinan con la señal deseada a la salida, de gran utilidad debido a que los ADCs de alta velocidad son muy sensibles a la calidad de muestreo del reloj.

2.5.1.2 Placa de evaluación PCBZ-9481

La board de evaluación provista por Analog Devices ofrece una forma fácil de probar las características del AD9481, requiere únicamente de una fuente de reloj, una señal analógica de entrada y una fuente de alimentación de 3.3 V. las salidas digitales y las salidas de reloj están disponibles en dos conectores de 40 pines implementados como uno de 80. La board tiene diferentes modos de operación configurables mediante *jumpers*. La placa de evaluación mostrada en la Figura 9 acepta señales analógicas de entrada de hasta 700 mV pico a pico centrado a tierra, con acople de impedancia de 50 Ω y puerto de entrada SMA (*SubMiniature version A*) para conexión de reloj externo.

Figura 9. Imagen placa de evacuación AD9481-PCBZ



2.6 MEZCLADOR DE FRECUENCIA

Los mezcladores son dispositivos activos o pasivos de tres puertos, están diseñados para entregar tanto una suma como una diferencia de frecuencias en un único puerto de salida cuando dos frecuencias de entrada se insertan en los otros dos puertos. Este proceso, llamado mezcla de frecuencias o *heterodino*, se encuentra en la mayoría de equipos de comunicación y se utiliza de modo que se puede aumentar o disminuir la frecuencia de una señal.

Una de las dos frecuencias de entrada será normalmente una onda continua (*Continuous Wave,* CW), producida por un oscilador local, mientras que la otra entrada será la señal recibida de una fuente de radiofrecuencia como antena o un generador.

Si se desea producir una frecuencia de salida menor que la señal RF de entrada, entonces esto se llama conversión descendente. Por el contrario, si se desea producir una señal de salida que está a una frecuencia más alta que la señal de entrada, se le conoce como conversión ascendente. La frecuencia en el puerto de salida del mezclador, se denomina frecuencia intermedia.

2.6.1 Descripción matemática:

Las señales de entrada son, en el caso más simple, ondas de tensión sinusoidales, que se pueden representar como:

$$v_s(t) = A_s x(t) \cos\big(w_s t + \varphi(t)\big) \tag{2}$$

$$v_o(t) = A_o \cos(w_o t) \tag{3}$$

Donde A_i y w_i son respectivamente la amplitud y frecuencia angular de cada señal de entrada, t el tiempo y $\varphi(t)$ la fase.

Una forma común para sumar y restar frecuencias es multiplicar las dos señales, utilizando la siguiente identidad trigonométrica :

$$\cos(A)\cos(B) = \frac{1}{2}[cos|A+B| + cos|A-B|] \tag{4}$$

Aplicándo (4) a las señales (2) y (3) se obtiene la salida del mezclador.

$$v_i(t) = v_s(t) v_o(t) \tag{5}$$

$$v_i(t) = \frac{K}{2} A_s A_o x(t)\big[\cos\big(|w_s + w_o| + \varphi(t)\big) + \cos\big(|w_s - w_o| + \varphi(t)\big)\big] \tag{6}$$

Se puede ver en (6) la señal suma $|w_s + w_o|$ y la señal resta $|w_s - w_o|$ de las frecuencias de las señales de entrada.

La translación de la señal de entrada $f_s \rightarrow v_s(t)$ a la frecuencia deseada $f_s \rightarrow y(t)$, se realiza seleccionando el valor adecuado de $f_s \rightarrow v_o(t)$ y filtrando la señal deseada $|w_s + w_o|$ o $|w_s - w_o|$.

Hay dos tipos de clasificaciones para los mezcladores, dependiendo de la ganancia o pérdida de conversión y dependiendo de la estructura utilizada para la implementación:

2.6.2 Clasificación según la ganancia o pérdidas de conversión

- Mezcladores pasivos: generalmente utilizan diodos como elementos no lineales, no tienen ganancia si no que tienen pérdidas de conversión (la potencia de salida es inferior a la de entrada) y tienen la relación de factor de ruido igual a las pérdidas de conversión.
- Mezcladores activos: están basados en transistores los cuales requieren una polarización, tienen ganancia de conversión (la potencia de salida es superior a la de entrada), requieren un menor nivel de señal del LO y el factor de ruido es independiente de las ganancias de conversión, este parámetro lo proporciona el fabricante.

Para el desarrollo del proyecto se utiliza un mezclador pasivo.

2.6.3 Clasificación según la estructura utilizada en la implementación, para mezcladores pasivos.

- Mezclador de balanceo simple (*Single-Balanced Mixers*, SBM): se componen de dos diodos, un *balun* y generalmente dos filtros. El *balun* convierte la salida no balanceada del LO a una entrada balanceada del mezclador, iguala los diodos a la impedancia del puerto, ayuda en el aislamiento de puerto a puerto y al equilibrio de los diodos. Los filtros, uno en cada uno de los puertos de RF e IF, son para mejorar el aislamiento del mezclador.
- Mezcladores de doble balanceo (*Double-Balanced Mixers,* DBM): se llaman así por la misma razón, se emplean dos *baluns*. La calidad de los mezcladores de doble balanceo es proporcional a la salida del producto de intermodulación (*Intermodulation Distortion,* IMD) que se construyen de armónicos impares RF y LO. Esta acción disminuye la salida total de los DBMs del producto del mezclador un cuarto de la cantidad generada dentro de cualquier mezclador simple. Sin embargo, el producto del mezclador se elimina a niveles variables, depende en gran medida de la calidad del diodo y la exactitud del *balun*. Por lo tanto, un DBM a menudo puede requerir el doble de potencia del LO, así como el doble del número de diodos internos equilibrados y baluns que un mezclador de simple balanceo. Un DBM tiene mejor supresión IMD, un ancho de banda más amplio y un punto de intersección más elevado.
- Mezcladores de Triple-balanceo (*Triple-Balanced Mixers,* TBM): tienen *baluns* ubicados en los tres puertos, junto con dos anillos de diodos. Se incrementa los puntos de intersección, para disminuir la generación del producto del mezclador, además se incrementa dos tonos los niveles de distorsión de intermodulación, se mejora el aislamiento puerto a puerto y se obtiene un mayor ancho de banda IF. Sin embargo, los TBMs necesitan mayor potencia del LO, otro anillo de diodo

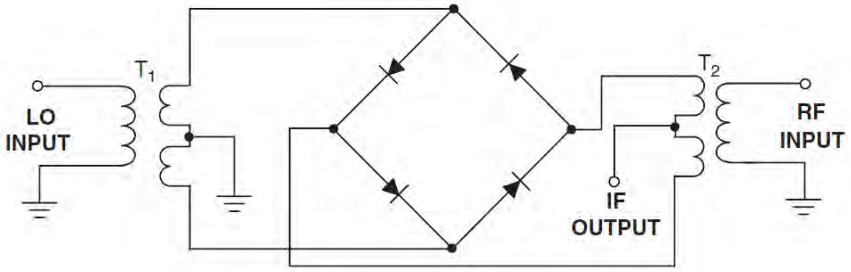
balanceado y un balun más equilibrado, por encima de lo exigido en el tipo DBM además el precio es más alto.

Si se requiere aumentar las especificaciones y la calidad general de este dispositivo, se debe aumentar el número de diodos. Esto permite una mayor amplitud del LO.

2.6.4 Mezclador SYM-63LH+

Se utiliza el mezclador de doble balanceo de referencia SYM-63LH+ nivel 10 (LO Power +10 dBm) y ancho de banda de 1 a 6000 MHz de la empresa Mini-Circuits, cuyo diagrama eléctrico se muestra en la Figura 10, junto con su respectiva board de evaluación TB_12 de la misma empresa como se mira en la figura 11 a) y b).

Figura 10. Diagrama eléctrico del mezclador SYM-63LH+

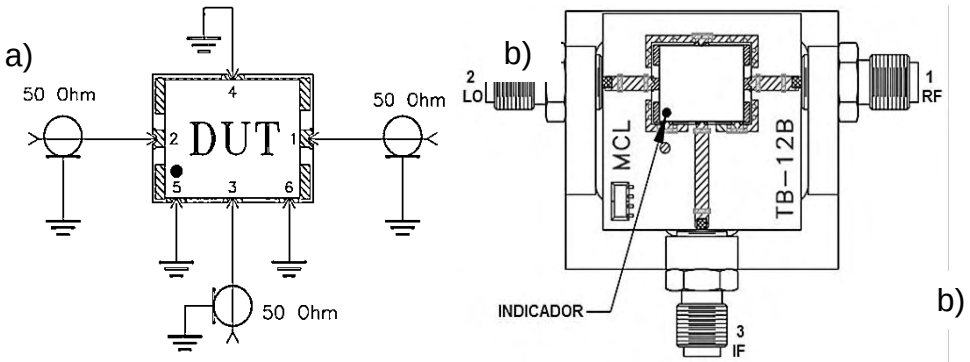


Fuente: *Tomada de COMPLETE WIRELESS DESING pag.380*

2.7 CIRCUITOS OSCILADORES

Un circuito oscilador es un sistema capaz de crear perturbaciones, cambios periódicos o cuasi periódicos en un medio, ya sea un medio material o un campo electromagnético como ondas de radio, microondas, infrarrojo, luz visible, rayos X, rayos gamma, rayos cósmicos, etc. Estos dispositivos ofrecen una señal cambiante en la salida.

Figura 11. Diagramas de la board de evaluación TB-12, para el mezclador SYM-63LH+ a) esquema eléctrico b) diagrama físico.



Fuente: *Basado en EVALUATION BOARD AND CIRCUIT FOR PIN CONNECTIONS, Mini-Circuits, Pag 1.*

Si la señal varía conforme a la onda seno, el circuito se denomina oscilador sinusoidal, si el voltaje de salida se eleva rápidamente hacia otro nivel de voltaje, el circuito por lo general se denomina oscilador de onda cuadrada o de pulso.

Un oscilador electrónico es fundamentalmente un amplificador cuya señal de entrada se toma de su propia salida a través de un circuito de realimentación. Se puede considerar que está compuesto por un circuito cuyo desfase depende de la frecuencia. Por ejemplo: un oscilador capacitivo-inductivo o un cristal de cuarzo.

2.7.1 Oscilador de cristal

Un oscilador de cristal es un oscilador de circuito sintonizado que utiliza un cristal piezoeléctrico como circuito de tanque resonante. El cristal, por lo general de cuarzo, posee mayor estabilidad para mantener constante la frecuencia de operación a la que originalmente se haya cortado. Los osciladores de cristal se utilizan donde sea que se requiera una alta estabilidad, como en el caso de transmisores y receptores de comunicación.

Un oscilador de cristal de cuarzo presenta la propiedad de que cuando se le aplica una tensión mecánica a través de sus caras, se desarrolla una diferencia de potencial eléctrico través de sus caras opuestas. A esta propiedad se la denomina efecto piezoeléctrico.

2.7.1.1 Oscilador de cristal controlado por horno OCXO

Un oscilador de este tipo utiliza una cámara de temperatura controlada para prevenir cambios en la frecuencia, debido a variaciones ambientales de temperatura, ya que mantiene el oscilador de cristal a una temperatura constante. Este oscilador logra la más alta estabilidad de frecuencia posible con un cristal, son usados típicamente para el control de frecuencia de radio-transmisores, estaciones base de celular, equipamiento en comunicaciones militares y para hacer mediciones de frecuencia con una alta precisión.

La frecuencia a la cual el cristal de cuarzo vibra depende de sus dimensiones físicas, un cambio en la temperatura causa que el cuarzo se expanda o contraiga debido a la expansión térmica, cambiando así la frecuencia de la señal producida por el oscilador, aunque el cuarzo tiene un bajo coeficiente térmico de expansión, los cambios en temperatura son la mayor causa de variaciones de frecuencia en osciladores de cristal.

El horno está térmicamente aislado y dentro de él está el cristal y uno o más elementos eléctricos de calentamiento, el termistor tiene un control de realimentación negativa para controlar la potencia del calentamiento y asegurar que el horno se mantenga a la temperatura deseada. Los hornos normalmente funcionan a una temperatura mayor a la ambiente, por lo tanto el oscilador usualmente requiere un

tiempo de calentamiento, durante este tiempo, la frecuencia puede no ser totalmente estable, una temperatura común para un OCXO es 75°C pero puede variar desde 30°C hasta 80°C. Los OCXOs requieren más potencia que los osciladores que trabajan a una temperatura ambiente, los requerimientos para la calefacción, la masa térmica, el aislamiento térmico significa que son físicamente más grandes.

2.7.2 Oscilador AOCJY1

Para obtener una excelente exactitud y precisión se adquiere un OCXO de referencia AOCJY1 100MHz de ABRACON [11], el cual genera ondas RF de forma sinusoidal o CMOS, frecuentemente utilizado en aplicaciones de infraestructura celular, sistemas de radar, equipamiento de medidas, rastreo GPS, WiMax/WLAN

2.8 FILTROS

Un filtro electrónico es un elemento que discrimina una determinada frecuencia o gama de frecuencias de una señal eléctrica que pasa a través de él, consiguiendo modificar tanto su amplitud como su fase. Los filtros pasivos son aquellos circuitos que utilizan capacitores, inductores y resistencias, son de baja sensitividad y poca disipación de energía. Por otra parte los filtros activos son los diseños en los cuales se usa amplificadores operacionales u otros elementos activos, la ventaja de diseñar este tipo de filtros es la eliminación de inductancias, ya que el circuito se reduce tanto en tamaño como en costo.

Con independencia de la realización del filtro (analógico, digital o mecánico) el comportamiento de un filtro se describe por su función de transferencia. Ésta determina la forma en que la señal aplicada cambia en amplitud y en fase al atravesar el filtro. La función de transferencia elegida tipifica el filtro. Algunos tipos de filtros son:

- Filtro de Butterworth, con una banda de paso suave y un corte agudo.
- Filtro de Chebyshev, con un corte agudo pero con una banda de paso con ondulaciones.
- Filtros elípticos o filtro de Cauer, que consiguen una zona de transición más abrupta que los anteriores a costa de oscilaciones en todas sus bandas.
- Filtro de Bessel, que, en el caso de ser analógico, aseguran una variación de fase constante.

Una señal $x(t)$ periódica, de periodo $2\pi/w$, se puede representar por su desarrollo en serie de Fourier como:

$$x(t) = a_0 + \sum_{k=1}^{\infty} (a_k cos kw_0 t + b_k sen kw_0 t) = A_0 + \sum_{k=1}^{\infty} A_k \cos(kw_0 t + \varphi_k) \qquad (7)$$

Donde a_k, b_k, A_k y φ_k son reales. De la misma forma, si $x(t)$ es una señal no periódica, se puede expresar como (integral de Fourier):

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(w)e^{jwt}dw \qquad (8)$$

El filtrado se considera, en general, como un proceso de cambio de espectro de la señal de entrada, es decir, una modificación de los valores X_k o de la función $X(w)$ para conseguir la señal de salida deseada. Más concretamente se puede entender como filtrado la eliminación de determinadas componentes de frecuencia de una señal (bandas atenuadas o eliminadas) dejando pasar las demás (bandas de paso).

Entre sus diversas aplicaciones se puede mencionar:
- Demodular señales.
- Ecualizar y obtener una calidad de audio de mayor fidelidad.
- Eliminar ruidos en los diferentes sistemas de comunicación.
- Convertir señales muestreadas en señales continuas.
- Detectar señales, como de la T.V. o radio.

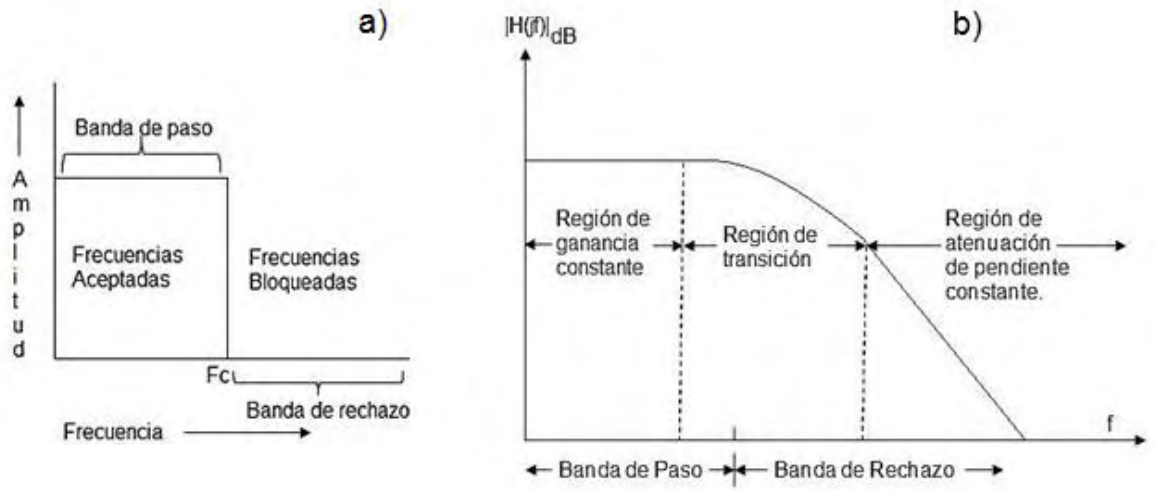
En resumen, los filtros son sistemas de dos puertos, uno de entrada y otro de salida, que funcionan en el dominio de la frecuencia. El espectro de frecuencia de la señal de salida tiene relación directa con respecto a la señal de entrada.

2.8.1 Características de los filtros

- La naturaleza de la respuesta en magnitud y fase en circuitos de pasa-bajas, pasa-banda, banda de paso y pasa-todo, describe la magnitud de la ganancia de un circuito como una función de frecuencia bajo condiciones de excitación sinusoidal. Un filtro típico, tiene una respuesta en magnitud casi constante sobre un cierto rango de frecuencias y atenúa la misma fuera de este rango.
- Banda de paso, banda de rechazo y región de transición: La banda de paso se define como el rango de frecuencias que un filtro permite pasar, con una mínima atenuación o alguna amplificación. La banda de rechazo, son todas las frecuencias no contempladas en la banda de paso. La región de transición es una zona ubicada entre la relativa porción plana de la banda de paso y la región constante de *rolloff* en la banda de rechazo. La Figura 12 a) y b) ilustra las regiones.
- Frecuencia de corte (w_c) o frecuencia critica. Es la frecuencia donde la respuesta de amplitud esta 3dB por debajo del valor de la banda de paso.
- Ganancia de la banda de paso (H_0). Es la ganancia que se obtiene por la amplificación de la banda de paso. Está presente en filtros que contienen dispositivos activos, ya que estos pueden amplificar la señal de entrada en la banda de paso.
- Atenuación de la pendiente (*rolloff*) u orden. Describe la proporción en que se disminuye la ganancia de un filtro fuera de la banda de paso. Se mide normalmente en octavas de dB (dB/octavas) o décadas de dB (dB/décadas).
- Características de frecuencia (w_0). Es la frecuencia central en los filtros con respuesta de pasa banda y rechaza banda. Mientras que en los filtros pasa baja y pasa alta representa el pico de una figura de mérito alta.

- Figura de mérito (Q). tiene muchas interpretaciones, depende de la respuesta del filtro. Q en conjugación con w_0, especifica la localización de cada par de polos complejos de la función de transferencia de un filtro. También se define como la distancia del polo al eje jw. En la banda de paso y en los filtros rechaza banda, Q mide la relación de la frecuencia central w_0 al ancho de banda $(w_{cu} - w_{cl})$. En filtros de segundo orden y filtros con respuestas pasa altas, mide el grado máximo de las características a la frecuencia w_0.

Figura 12. a) Banda de paso y banda de rechazo b) Regiones de la señal filtrada.



Fuente: *Tomada de LABORATORIO ANALÓGICO, Gabriela Quiroz Córdova, Universidad de las Américas Puebla, Tesis profesional para obtener el título Licenciatura en Ingeniería en Electrónica y Comunicaciones, 2007 Pag 5*

Las características secundarias incluyen la sensitividad a los cambios de parámetros y retraso de grupo. Generalmente la frecuencia de corte (w_c o f_c), se calcula a los 3dB. La razón de escoger este punto es porque a 3dB el decremento en la ganancia de voltaje equivale a una reducción del 50% de la potencia entregada a la carga que está siendo alimentada por el filtro. Por esta razón, f_c es conocida como el punto medio de potencia.

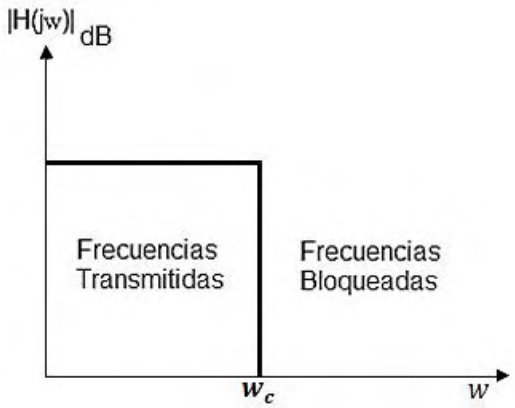
2.8.2 Tipos de filtros

Existen diferentes tipos de filtros, cada filtro contiene una respuesta específica de su aplicación. Hay cinco tipos básicos de filtros: pasa bajas, pasa altas, pasa banda, rechaza banda y pasa todo (o retardador de tiempo). Sus nombres describen el comportamiento del filtro.

2.8.2.1 Filtro pasa-bajas:

Tiene la propiedad de transmitir componentes de señales de excitación de bajas frecuencias, incluyendo las señales de corriente directa, mientras que las componentes de altas frecuencias, incluyendo las infinitas, son bloqueadas. La magnitud de la función pasa-bajas tiene la respuesta ideal que se ilustra en la Figura 13. Cualquier señal que sea mayor a la frecuencia de corte (w_c) del filtro es rechazada, mientras que las señales menores a w_c son transmitidas. La banda de paso se extiende desde DC hasta la frecuencia de corte w_c.

Figura 13. Respuesta ideal de un filtro pasa bajas.



Fuente: *Tomada de LABORATORIO ANALÓGICO, Gabriela Quiroz Córdova, Universidad de las Américas Puebla, Tesis profesional para obtener el título Licenciatura en Ingeniería en Electrónica y Comunicaciones, 2007 Pag 10*

Normalmente una función de red pasa bajas con una respuesta similar a la mostrada en la Figura 13, tiene sus ceros localizados en el infinito, por lo tanto los polinomios del numerador son de grado cero. Su forma general es:

$$T(s) = \frac{H}{D(s)} \tag{9}$$

Donde H, es una constante y no una función de s; la forma del polinomio $D(s)$ depende de los elementos de la red.

2.8.2.2 Filtro pasa-altas:

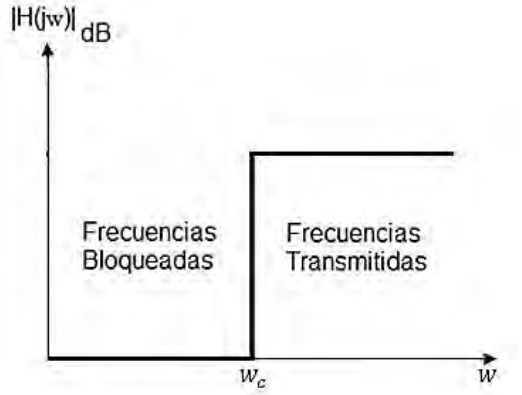
Tiene como prioridad bloquear las frecuencias que se encuentran por debajo de la frecuencia de corte w_c, y transmitir todas aquellas componentes de frecuencia que sean mayores, la banda de rechazo se extiende desde DC hasta w_c y la banda de paso, en teoría, se extiende desde w_c hasta la frecuencia infinita. La Figura 14 ilustra la respuesta ideal de un filtro pasa altas.

Las funciones pasa altas con características en magnitud comúnmente tienen sus ceros localizados en el origen del plano de la frecuencia compleja. Por lo tanto, las funciones racionales tienen la forma.

$$T(s) = \frac{Hs^n}{D(s)} \tag{10}$$

Donde H es una constante y n es el grado del denominador polinomial $D(s)$.

Figura 14. Respuesta ideal de un filtro pasa altas.



Fuente: *Tomada de LABORATORIO ANALÓGICO, Gabriela Quiroz Córdova, Universidad de las Américas Puebla, Tesis profesional para obtener el título Licenciatura en Ingeniería en Electrónica y Comunicaciones, 2007. Pag 11.*

2.8.2.3 Filtro pasa-banda:

Tiene la característica de transmitir un rango de frecuencias o banda de paso y rechazar dos bandas de frecuencias o bandas de rechazo, una de las bandas de rechazo contiene frecuencias menores que las frecuencias de la banda de paso (w_1), mientras que la otra contiene frecuencias mayores (w_2). La Figura 15 muestra un filtro pasa banda ideal.

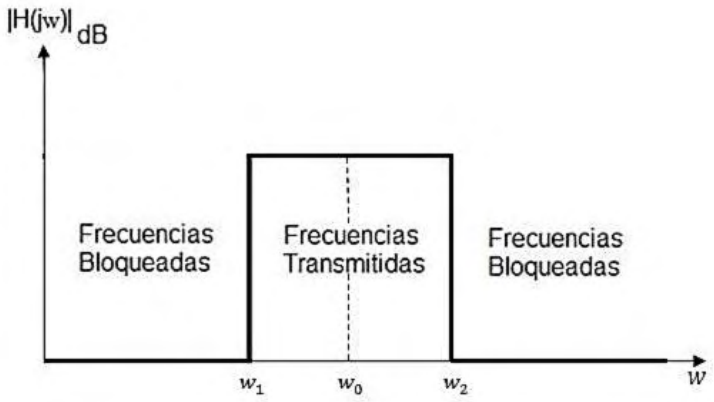
El rango de frecuencias que es transmitido es llamado ancho de banda (*Bandwidth,* BW) y es definido como la diferencia entre las frecuencias que delimitan la banda de paso. Si se utiliza w_1 y w_2 como los limites, se obtiene.

$$BW = w_2 - w_1 \tag{11}$$

La frecuencia central w_0 de la banda de paso está definida como la medida geométrica de las frecuencias en el límite.

$$w_0 = \sqrt{w_1 w_2} \tag{12}$$

Figura 15. Respuesta ideal de un filtro pasa banda.



Fuente: *Tomada de LABORATORIO ANALÓGICO, Gabriela Quiroz Córdova, Universidad de las Américas Puebla, Tesis profesional para obtener el título Licenciatura en Ingeniería en Electrónica y Comunicaciones, 2007 Pag12.*

La magnitud del filtro pasa-banda a la frecuencia cero y a frecuencias infinitas debe ser cero. Todos los pasa banda con características en magnitud, tienen la mitad de sus ceros en el origen y la otra mitad de los ceros en el infinito. La función racional tiene la forma:

$$T(s) = \frac{Hs^{n/2}}{D(s)} \tag{13}$$

Donde H es una constante y n es el grado del polinomio denominador $D(s)$ y es siempre un número entero par.

2.8.2.4 Filtro rechaza-banda:

Un filtro rechaza-banda es lo opuesto al filtro pasa-banda. Todas las señales de frecuencias son transmitidas excepto aquellas que caen dentro de cierta banda o rango especifico. Debido a que rechazan o atenúan componentes de frecuencia no deseados, son utilizados para eliminar señales de ruido, tal como la señal de 60 Hz, inducida por las líneas de corriente alterna.

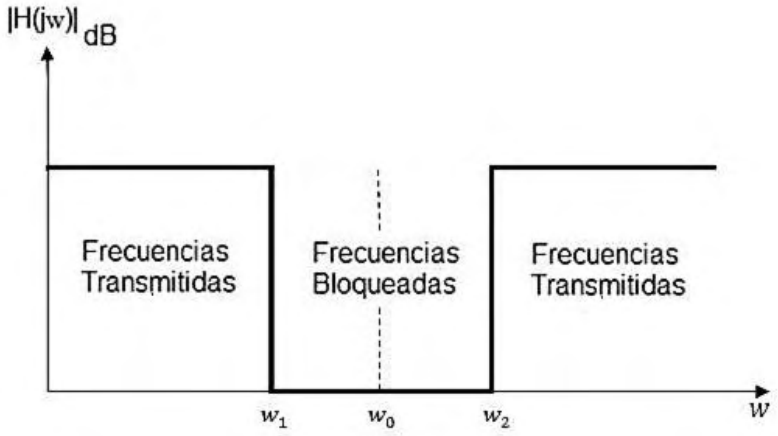
La respuesta ideal de un filtro rechaza-banda es ilustrada en la Figura 16. Idealmente, este filtro tiene una atenuación infinita en el punto w_0, y la ganancia en decibeles a la frecuencia w_0, se llama profundidad nula.

Este tipo de filtro también es conocido como filtro Notch, ya que la respuesta tiene un “hueco” en la salida. La función racional tiene la forma.

$$T(s) = \frac{H(s^n + w_0^2)}{D(s)} \tag{14}$$

Al igual que en el pasa-banda, WB y w_0, están definidas por las ecuaciones (11) y (13) respectivamente. Los ceros de la función de red con respuesta rechaza-banda, son finitos ubicados en $\pm jw_0$ de multiplicidad $n/2$, en total existen n ceros.

Figura 16. Respuesta ideal de un filtro rechaza banda.



Fuente: *Tomada de LABORATORIO ANALÓGICO, Gabriela Quiroz Córdova, Universidad de las Américas Puebla, Tesis profesional para obtener el título Licenciatura en Ingeniería en Electrónica y Comunicaciones, 2007 Pag 12.*

2.8.2.5 Filtro pasa-todo:

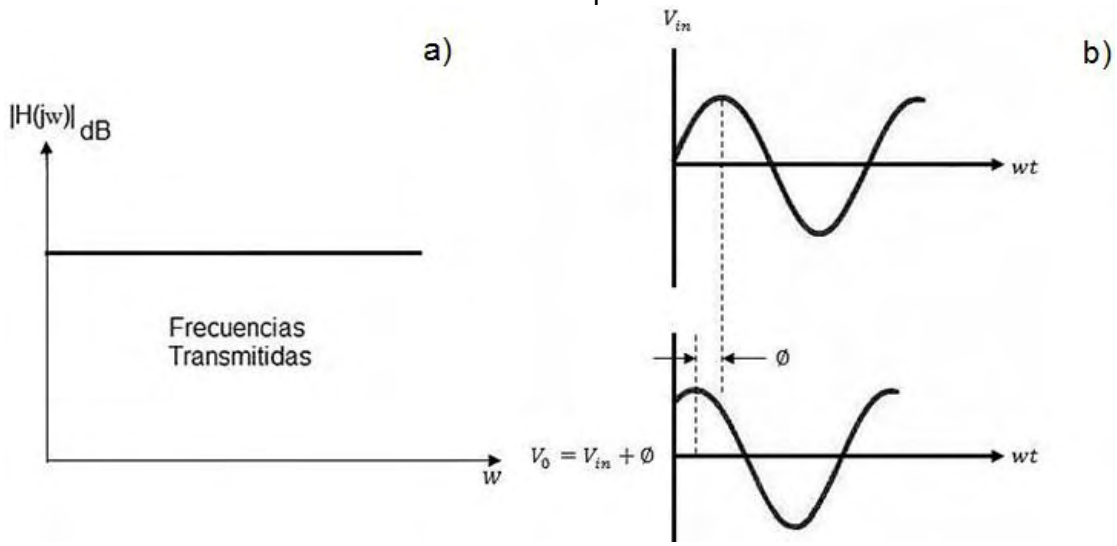
Transmiten señales de todas las frecuencias. Idealmente estos filtros cubren todo el espectro. Son diseñados para proveer una ganancia constante a todas las señales a cualquier frecuencia, sin embargo la relación de fase entre la señal de entrada y señal de salida varia como una función de la frecuencia. La Figura 17 a) ilustra la respuesta en magnitud de este filtro y la figura 17 b) ilustra las formas de onda de la señal de entrada y salida.

Estos filtros son generalmente utilizados en el diseño de ecualizadores de fase. Su función de transferencia de segundo orden es:

$$T(s) = H\frac{s^2 - \frac{w_n}{Q} + w_n^2}{s^2 + \frac{w_n}{Q} + w_n^2} \qquad (15)$$

Los ceros de la función están ubicados sobre el semiplano derecho del plano jw.

Figura 17. a) Respuesta ideal de un filtro pasa-todo. b) Fase de la señal de entrada y salida del filtro pasa-todo



Fuente: *Tomada de LABORATORIO ANALÓGICO, Gabriela Quiroz Córdova, Universidad de las Américas Puebla, Tesis profesional para obtener el título Licenciatura en Ingeniería en Electrónica y Comunicaciones, 2007 Pag 13, 14.*

2.8.3 Filtros microondas

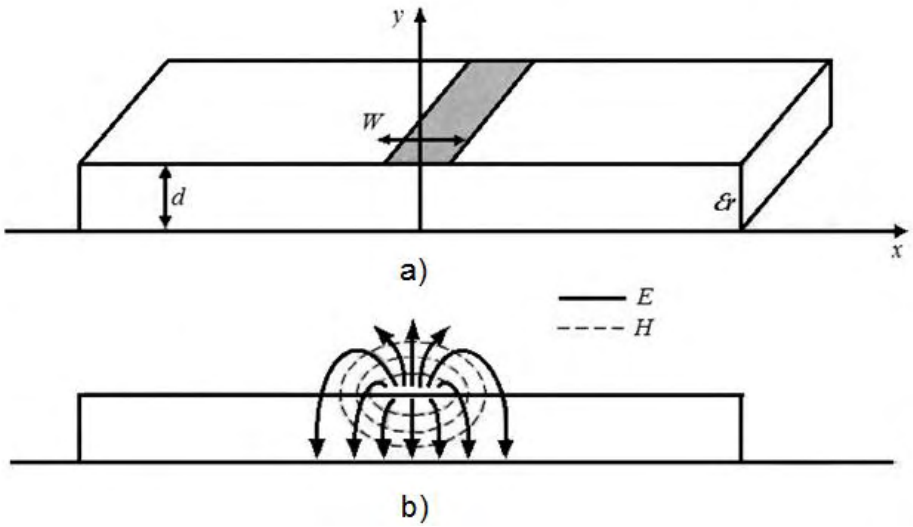
Un filtro microondas es una red de dos puertos usada para el control de la respuesta en frecuencia en cierto punto de un sistema microondas, provee transmisión de las frecuencias dentro de la banda de paso de los filtros y atenuación en la banda de rechazo. Típicamente la respuesta de frecuencia incluye características pasa bajas, pasa altas, pasa banda y rechaza banda. Aplicaciones pueden ser encontradas virtualmente en cualquier tipo de comunicaciones microondas, radar o sistemas de testeo y medida.

En cuanto al hardware que se necesita para implementar estos sistemas, el desarrollo de dispositivos activos de estado sólido, tales como transistores bipolares y transistores de efecto de campo (*Field-Effect Transistors*, FET), ha tenido un impacto importante en el campo de la ingeniería de microondas. Con la disponibilidad de los transistores de microondas, el desarrollo de componentes se ha trasladado y centrado sobre estructuras de líneas de transmisión planares, como líneas *Microstrip* y líneas de transmisión coplanar. Estas estructuras se pueden fabricar usando técnicas de circuito impreso. Por eso son compatibles con los dispositivos de estado sólido, ya que éstos son fáciles de conectar al circuito Microstrip pero difíciles de incorporar como parte de un circuito en guía de onda.

2.8.4 Tecnología Microstrip

Es uno de los más populares tipos de transmisión de línea planar, principalmente porque pueden ser fabricados por procesos fotolitográficos y es fácilmente integrable con otros dispositivos activos o pasivos. La geometría de línea Microstrip es como se muestra en la Figura 18. a). Un conductor con ancho W es impreso sobre un substrato dieléctrico delgado de ancho d y permeabilidad relativa ε_r, un diagrama de las líneas de campo es mostrado en la Figura 18 b).

Figura 18. Líneas de transmisión Microstrip a) geometría, b) líneas de campo eléctrico y magnético.



Fuente: *Basada en MICROWAVE ENGINEERING. David M. Pozar. Second Edition, University Massachusetts at Amherst. John Wiley & Sons, Inc. Pag 161.*

Si el dieléctrico no está presente ($\varepsilon_r = 1$), se podría pensar en la línea Microstrip como dos tiras de cable que consisten de dos líneas conductoras de ancho W, separadas una distancia 2d (el plano de tierra puede ser eliminado, por la teoría de imagen [12]). En este caso se tendría una transmisión electromagnética (TEM) sencilla en la línea de transmisión, con $V_p = c$ y $\beta = K_0$. La presencia del dieléctrico y el hecho de que éste no llena la región de aire por encima de la línea ($y > d$), complica el comportamiento y el análisis de la línea Microstrip. Diferente a stripline, donde todos los campos están dentro de una región dieléctrica homogénea, Microstrip tiene la mayoría de estas líneas de campo en la región del dieléctrico, concentradas entre la tira del conductor, el plano de tierra y algunas fracciones en la región de aire por encima del substrato. Por esta razón la línea Microstrip no puede soportar una TEM de onda pura. La velocidad de fase de campos de TEM en la región del dieléctrico es $c/\sqrt{\varepsilon_r}$, pero la velocidad de fase de los campos TEM en una región de aire podría ser c. Así, un acople de fase a la interface de aire-dieléctrico podría ser imposible de lograr para una onda tipo TEM.

En la actualidad, los campos exactos de una línea Microstrip constituyen ondas híbridas de transmisión magnética-eléctrica (TM-TE) y requieren técnicas de análisis más avanzadas. En la mayoría de aplicaciones, sin embargo, el sustrato dieléctrico es eléctricamente muy angosto, $(d \ll \lambda)$, asi que los campos son cuasi TEM. En otras palabras, los campos son esencialmente los mismos que aquellos del caso estático. Así, buenas aproximaciones de velocidad de fase, constante de propagación y características de impedancia pueden ser obtenidas de una solución estática y casi estática, luego la velocidad de fase y la constante de propagación puede ser expresada como:

$$v_p = \frac{c}{\sqrt{\varepsilon_e}} \tag{16}$$

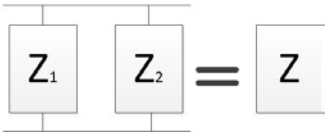
$$\beta = k_0\sqrt{\varepsilon_e} \tag{17}$$

Donde ε_e es la constante efectiva del dieléctrico de la línea Microstrip, desde que algunas de las líneas de campo estén en la región del dieléctrico y algunas en el aire, la constante efectiva dieléctrica satisface la relación, $1 < \varepsilon_e < \varepsilon_r$ y es independiente del grosor del sustrato d y el ancho del conductor W.

2.8.5 Resonadores de comportamiento dual (DBR)

La Figura 19 muestra la estructura básica de los resonadores de comportamiento dual (*Dual Behavior Resonator,* DBR), la cual es creada usando dos elementos resonantes, con impedancias Z_1 y Z_2 en configuración paralelo

Figura 19. Estructura básica de un DBR.



Este arreglo tiene una impedancia total:

$$Z = \frac{Z_1 Z_2}{Z_1 + Z_2} \tag{18}$$

Así que, los DBR pueden ser configurados con dos ceros de transmisión a frecuencias donde $Z_1 = 0$ y $Z_2 = 0$. Sin embargo este tiene una frecuencia de resonancia donde la impedancia total tiende hacia infinito en $Z_1 + Z_2 = 0$. Esta configuración está disponible para la construcción de filtros pasa-banda, considerando que los ceros de transmisión pueden ser ubicados en las bandas laterales a la frecuencia resonante para lograr un alto rechazo.

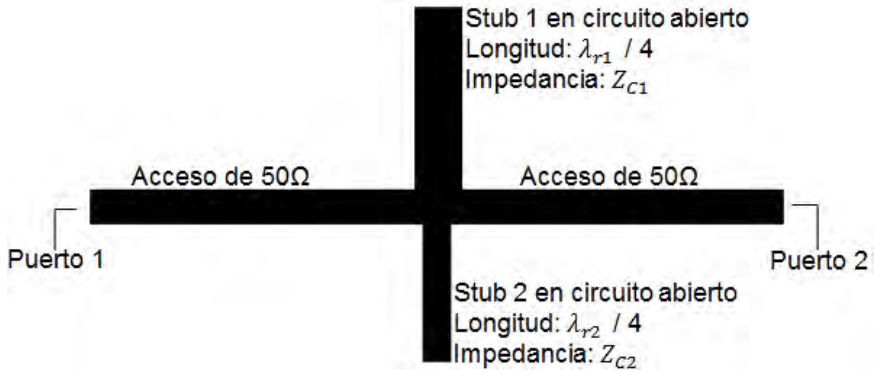
2.8.5.1 Uso de stubs en circuito abierto para diferentes longitudes en un DBR:

La configuración mostrada en la Figura 20 puede ser implementada en tecnología Microstrip usando stubs de diferentes longitudes, impedancias y terminaciones. En este trabajo se usan *stubs* Microstrip en circuito abierto y configuración paralela, cuyas estructuras resonantes individuales son rechaza banda con una longitud $\frac{\lambda_r}{4}$. La impedancia de entrada de esta clase de *stubs* es

$$Z_{stub} = \frac{-jZ_{line}}{\tan\left(\frac{\Pi}{2}\frac{\lambda_r}{\lambda}\right)} \qquad (19)$$

Donde Z_{line} es la impedancia característica de la línea, λ es la longitud de onda de la señal de entrada y λ_r es la longitud de onda de la frecuencia resonante deseada (la cual hace $Z_{stub} = 0$) acorde con (19), las impedancias $Z_1\ y\ Z_2$ de la estructura DBR pueden ser implementadas usando stubs Microstrip en circuito abierto. La Figura 20 muestra una configuración Microstrip DBR usando una línea de acceso de 50Ω, λ_{r1} y λ_{r2} son longitudes de onda a las frecuencias donde la transmisión de ceros está configurada. Z_{C1} y Z_{C2} son las impedancias características de cada *stub*, los cuales son controlados variando el ancho de la línea.

Figura 20. Configuración típica de una estructura DBR en tecnología Microstrip.

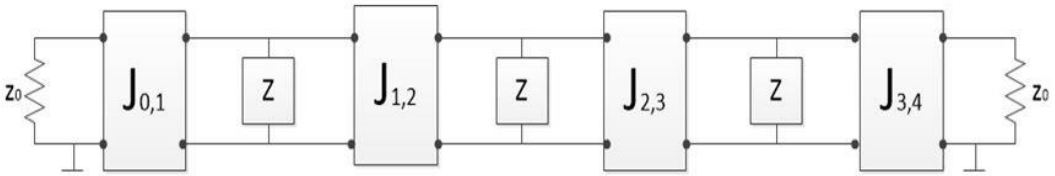


Fuente: *Basada en DESIGN AND SIMULATION OF A MULTILAYER DUAL BEHAVIOR, RESONATOR MICROWAVE FILTER. Carlos A. Viteri, Juan C. Bohórquez. Comunicaciones (LATINCOM), 2010 Conferencia Latinoamericana IEEE en Bogotá, Colombia.Pag 2.*

2.8.5.2 Síntesis de filtros DBR de tercer orden

Para diseñar un filtro DBR de tercer orden se utiliza el procedimiento definido en [13], tomando la configuración de la Figura 21 tres DBRs idénticos son acoplados usando inversores de admitancia, los cuales son diseñados basados en dos parámetros: el ancho de banda fraccional w_f y el parámetro de susceptancia b, la Figura 21 muestra el modelo del filtro.

Figura 21. Modelo del filtro DBR de tercer orden



Fuente: *Basada en DESIGN AND SIMULATION OF A MULTILAYER DUAL BEHAVIOR, RESONATOR MICROWAVE FILTER. Carlos A. Viteri, Juan C. Bohórquez. Comunicaciones (LATINCOM), 2010 Conferencia Latinoamericana IEEE en Bogotá, Colombia.Pag 2.*

Los inversores de admitancia son implementados en secciones Microstrip en la forma de un cuarto de longitud de onda, las impedancias de estas secciones son calculadas usando:

$$Z_{i,i+1} = \frac{Z_0}{J_{i,i+1}} \tag{20}$$

$$J_{0,1} = \sqrt{\frac{G_A\ b\ w_f}{g_0 g_1 w_1'}} \tag{21}$$

$$J_{j,j+1} = \frac{w_f\ \ b}{w_1' \sqrt{g_j\ g_{j+1}}} \tag{22}$$

$$J_{3,4} = \sqrt{\frac{G_B\ b\ w_f}{g_3\ g_4\ w_1'}} \tag{23}$$

Donde Z_0 es la impedancia característica deseada (50Ω para este diseño), G_A y G_B son los terminales de conductancia normalizados del circuito (con un valor de 1 cuando las terminaciones son cargas adaptadas), los parámetros g_i son los coeficientes del filtro pasa-bajas máximamente plano y w_1' es la frecuencia de corte normalizada de un prototipo pasa bajas (se toma el valor de 1). La longitud de los inversores es un cuarto de longitud de onda de la frecuencia central deseada.

Las estructuras DBR deben tener un cuarto de longitud de onda a las frecuencias de parada deseadas (una más larga y otra más corta que el de la frecuencia central), las cuales deben estar configuradas cerca del filtro pasa banda.

Por lo tanto, si la transmisión más baja se encuentra a una frecuencia f_1 y la frecuencia más alta a f_2 (con longitudes de onda λ_{r1} y λ_{r2} respectivamente), los stubs que constituyen DBR deben tener longitudes $l_1 = \lambda_{r1}/4$ y $l_2 = \lambda_{r2}/4$.

Las expresiones analíticas para las impedancias características de los stubs son derivadas de (18) y (19) y la condición $Z_1 + Z_2 = 0$. Las siguientes expresiones son adaptadas de [13].

$$Z_{C1} = -Z_{C2}\frac{\tan\left(2\pi\frac{l_1}{\lambda_0}\right)}{\tan\left(2\pi\frac{l_2}{\lambda_0}\right)} \tag{24}$$

$$Z_{C2} = Z_0\frac{\pi}{b}(R - S) \tag{25}$$

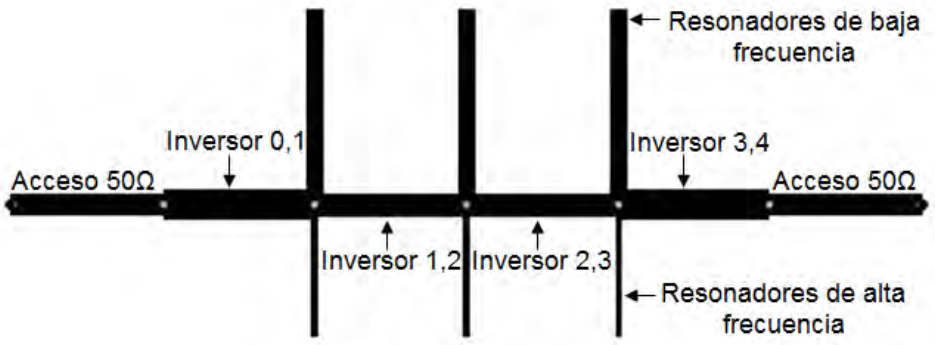
Con

$$R = \frac{l_2}{\lambda_0}\left[1 + tan^2\left(2\pi\frac{l_2}{\lambda_0}\right)\right] \tag{26}$$

$$S = \frac{l_1}{\lambda_0}\left[1 + tan^2\left(2\pi\frac{l_1}{\lambda_0}\right)\right]\left[\frac{\tan\left(2\pi\frac{l_2}{\lambda_0}\right)}{\tan\left(2\pi\frac{l_1}{\lambda_0}\right)}\right] \tag{27}$$

Con estas expresiones, es posible calcular las dimensiones de un filtro DBR Microstrip de tercer orden. La Figura 22 muestra la configuración de este.

Figura 22. Configuración de un filtro DBR de tercer orden



2.9 ANTENA

The IEEE *Standard Definitions of Terms for Antennas* (IEEE Std 145-1983) define la antena como “un medio para irradiar o recibir ondas de radio.” En otras palabras, la antena es la estructura de transición entre el espacio libre y un dispositivo guiado. La

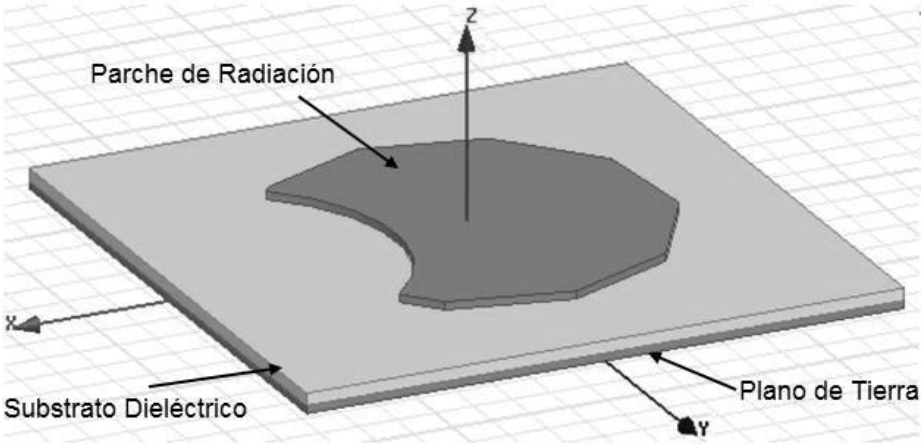
línea de guía o dispositivo de transmisión puede tomar la forma de una línea coaxial o una guía de onda y se utiliza para transportar la energía electromagnética desde la fuente de transmisión a la antena, o desde la antena al receptor. En el primer caso tenemos una antena de transmisión y en este último una antena receptora.

2.9.1 Antenas Microstrip

Una antena Microstrip en su configuración más simple consiste de un parche de radiación sobre uno de los lados del substrato dieléctrico ($\varepsilon_r \leq 10$), como se observa en la Figura 23 el cual tiene un plano de tierra del otro lado. El parche conductor, normalmente de cobre u oro, puede asumir virtualmente cualquier forma, pero formas regulares son generalmente usadas para simplificar el análisis y predecir su rendimiento. Idealmente, la constante del dieléctrico ε_r del substrato debe ser baja ($\varepsilon_r < 2.5$), para mejorar el margen de campos que cuentan para la radiación. Sin embargo, otro requisito de desempeño puede ser el uso de materiales de substrato cuyas constantes dieléctricas sean más grandes que cuatro. Varios tipos de substratos tienen un largo rango de constante dieléctrica y perdida de tangente. Algunos de estos substratos son flexibles en su naturaleza, los cuales los hacen adecuados para la conformación de antenas envolventes.

Los parches rectangulares y circulares son los más populares por la facilidad de análisis, fabricación y sus atractivas características de radiación, especialmente la baja radiación de polarización cruzada ([14], Pág 66).

Figura 23. Parche en tecnología Microstrip.



Las antenas Microstrip son de bajo perfil, adecuadas a superficies planares y no planares, simples y económicas para la fabricación usando tecnología moderna de impresión de circuitos, mecánicamente robusta cuando se montan en superficies rígidas, compatibles con diseños de circuitos integrados monolíticos de microondas

(*monolithic microwave Integrated Circuits,* MMIC) y muy versátiles en términos de la frecuencia de resonancia, polarización, patrones e impedancia. Estas antenas pueden ser montadas sobre superficies de alto rendimiento, tales como aeronaves, naves espaciales, satélites, misiles, automóviles y teléfonos móviles.

Las antenas impresas, de tipo parche se diseñan a partir de líneas de transmisión o resonadores sobre sustrato dieléctrico. Las dimensiones se eligen de forma que la estructura disipe la potencia en forma de radiación. Se pueden encontrar radiadores de las formas más diversa, aunque las geometrías más habituales son las circulares y rectangulares.

Las antenas Microstrip tienen varias ventajas comparadas con las antenas microondas convencionales, cubren rangos de frecuencias de 100 MHz a 100 GHz por tanto son usadas en muchas aplicaciones. Algunas de sus principales ventajas son:

- Livianas, ocupan poco volumen.
- Bajos costos de fabricación.
- La polarización lineal y circular son posibles con una alimentación simple.
- Pueden diseñarse antenas de frecuencia dual y polarización dual.
- Fácilmente integrables con circuitos integrados de microondas (MICs).
- Trabajan a distintas frecuencias.
- Las líneas de alimentación y la red de adaptación pueden ser fabricadas simultáneamente con la estructura de la antena.

Sin embargo también tienen algunas limitaciones:

- Son de pequeño ancho de banda asociados a problemas de tolerancia.
- Algo bajas en ganancia (-6 dB).
- Grandes pérdidas óhmicas en la estructura de arreglos de alimentación.
- Complejas estructuras de alimentación requeridas para un alto rendimiento.
- Es difícil de lograr una polarización pura.
- Extrañas radiaciones en las alimentaciones y uniones.
- Baja capacidad de manejo de potencia.
- En un ambiente de altas frecuencias, reduce la ganancia y eficiencia, así como es inaceptable los altos niveles de polarización cruzada y acoplamiento mutuo en muchos casos.
- Excitación de las ondas en la superficie.
- Una antena fabricada sobre un substrato con alta constante dieléctrica es preferida por su fácil integración con MMIC RF. Sin embargo, una alta constante dieléctrica en el substrato acarrea una eficiencia baja y un estrecho ancho de banda.

2.10 MATLAB

Es un lenguaje de alto nivel que posee ambiente interactivo para computación numérica, visualización y programación. Usando MATLAB® se puede analizar datos, desarrollar algoritmos, crear modelos y aplicaciones. El lenguaje, las herramientas y funciones matemáticas incorporadas permiten explorar múltiples enfoques y llegar a una solución más rápida que usando hojas de cálculo tradicionales o lenguajes de programación, tales como C/C++ o Java ™.

Matlab® es usado para un rango de aplicaciones, incluyendo procesamiento de señales y comunicaciones, procesamiento de imagen y video, pruebas y mediciones, finanzas computacionales y biología computacional. Más de un millón de ingenieros y científicos en la industria y academia usan MATLAB®, el lenguaje de la computación técnica.

Características principales:

- Lenguaje de alto nivel para computación numérica, visualización y desarrollo de aplicaciones.
- Ambiente interactivo para exploración iterativa, diseño y resolución de problemas.
- Funciones matemáticas para algebra lineal, estadística, análisis de Fourier, filtros, optimización, integración numérica y resolución de ecuaciones diferenciales ordinarias.
- Construcciones de gráficos para visualización de datos y herramientas para crear gráficos personalizados.
- Desarrollo de herramientas para mejorar la calidad de códigos, sostenibilidad y máximo rendimiento.
- Herramientas para la construcción de aplicaciones con interfaces gráficas personalizadas.
- Funciones para integrar algoritmos basados MATLAB® con aplicaciones externas y lenguajes como C, Java, .Net y Microsoft ® Excel ®

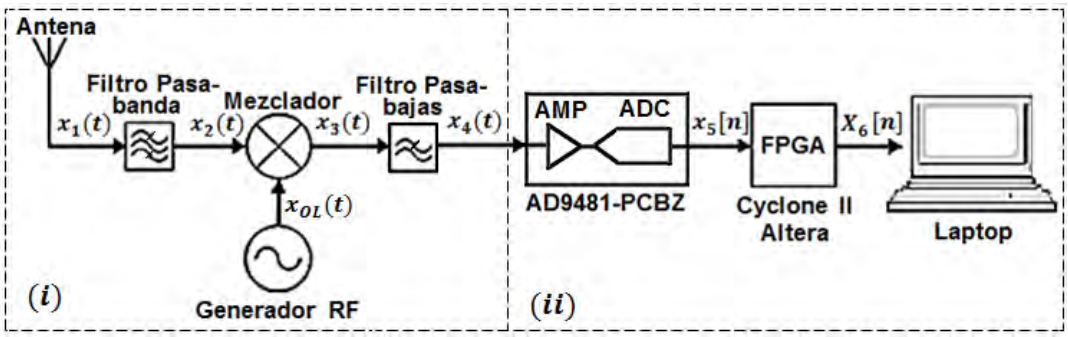
3. METODOLOGÍA

3.1 DISEÑO

En esta sección se hace el diseño de un prototipo de analizador de espectro para señales radioeléctricas de alta frecuencia, en las bandas de uso libre de hasta 2.7 GHz y con un ancho de banda de alrededor de 20 MHz. Este prototipo se caracteriza por la flexibilidad en el diseño, que permite realizar actualizaciones en software y hardware, y por ser de bajo costo en la implementación. También se requiere la visualización de las componentes y la DEP en el dominio de la frecuencia, aproximadamente en tiempo real, y a través de una computadora portátil.

En la Figura 24 se muestra el diagrama de bloques del prototipo. El analizador de espectro que aquí se propone consta de dos subsistemas conectados en cascada: (i) El receptor superheterodino y (ii) el sistema de desarrollo, o sistema embebido.

Figura 24. Diagrama de bloques del prototipo de analizador de espectro basado en un receptor superheterodino y un sistema embebido FPGA.



3.1.1 Receptor superheterodino

Es un receptor de ondas de radio que utiliza un proceso de mezcla de frecuencias para convertir la señal recibida en una de frecuencia intermedia fija, que es más fácil de filtrar y amplificar que la frecuencia de radio de la portadora original. Los receptores superheterodinos mezclan una señal generada por un oscilador local con la señal entrante. De esta resulta una señal que posee dos bandas de frecuencia: una superior y otra inferior a la frecuencia entrante. Una de ellas, normalmente la inferior, es elegida como la frecuencia intermedia (f_i).

Este tipo de receptor tiene características superiores, tanto en selectividad como en estabilidad de frecuencia, frente a otro tipo de receptores. Es relativamente fácil estabilizar un oscilador con la tecnología moderna de sintetizadores de frecuencia, y que se utiliza en el Generador RF, y los filtros de f_i pueden tener una banda de paso mucho más estrecha, para un mismo factor de calidad, que un filtro equivalente para

RF. Sin embargo, la radiación generada por el oscilador puede afectar e interferir la f_i, pero este efecto se puede corregir usando jaulas de Faraday[2] a manera de blindaje electromagnético.

Usualmente los diseños de receptores superheterodinos tienen una etapa de amplificación RF, después de la antena, que implica, para nuestros propósitos, la implementación de un amplificador de muy amplio ancho de banda y que incrementa en gran medida el costo total del proyecto. En este prototipo se opta por una conexión directa de la antena al filtro pasa-banda y el diseño de una antena de ganancia apropiada.

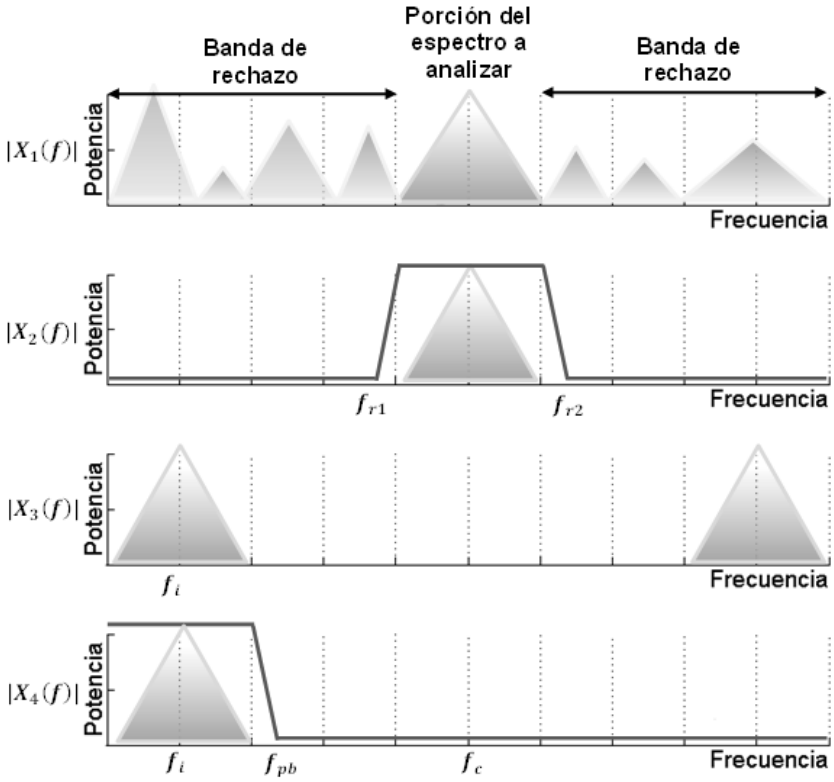
En el diagrama de diseño planteado, $x_1(t)$: $\mathbb{R} \rightarrow \mathbb{R}$, donde t es el tiempo, con $t \in [0, \infty]$, es la señal recibida y que se conduce a un filtro pasa-banda que selecciona la porción de espectro electromagnético que se desea analizar. Esta señal filtrada, definida como $x_2(t)$: $\mathbb{R} \rightarrow \mathbb{R}$, se mezcla con la señal proveniente del generador RF local, $x_{OL}(t)$: $\mathbb{R} \rightarrow \mathbb{R}$, para obtener la señal $x_3(t)$: $\mathbb{R} \rightarrow \mathbb{R}$, que tiene dos bandas de frecuencia: una superior que equivale a la suma de las frecuencias de las señales $x_2(t)$ y $x_{OL}(t)$ y otra inferior que equivale a la sustracción sus frecuencias. Finalmente, para garantizar que $x_3(t)$ sea de banda limitada, ésta es filtrada con un filtro pasa-bajas. La señal de salida de esta etapa se denota como $x_4(t)$: $\mathbb{R} \rightarrow \mathbb{R}$.

En la Figura 25 se observa una representación general del espectro de frecuencia de las señales de esta etapa. De arriba hacia abajo, el primer espectro muestra la magnitud de la densidad espectral de la señal recibida por la antena, $|X_1(f)|$, donde $X_1(f)$ es la transformada de Fourier de $x_1(t)$ y f es la frecuencia en Hz, f_c es la frecuencia de la portadora; el segundo espectro detalla la magnitud de la densidad espectral de la señal que pasa por el filtro pasa-banda, $|X_2(f)|$, f_{r1} y f_{r2} son las frecuencias de corte del filtro; el tercer espectro detalla la magnitud de la densidad espectral de la señal que ha sido desplazada en frecuencia por el mezclador, $|X_3(f)|$, f_i es la frecuencia intermedia; y finalmente, el cuarto espectro muestra la magnitud de la densidad espectral de la señal limitada en frecuencia, que resulta después de aplicarle un filtro pasa-bajas,$|X_4(f)|$, este filtro se conoce como filtro *anti-aliasing*.

A continuación se describe más detalladamente el proceso de diseño de la etapa del receptor superheterodino. Se comienza con la descripción de la tecnología utilizada y la metodología de diseño que se sigue en cada hito.

[2] es un efecto provocado en el que el campo electromagnético en el interior de una caja de material conductor en equilibrio es nulo, de paso anula el efecto de los campos externos.

Figura 25. Representación del espectro de las señales presentes en el receptor superheterodino y el filtro anti-aliasing.

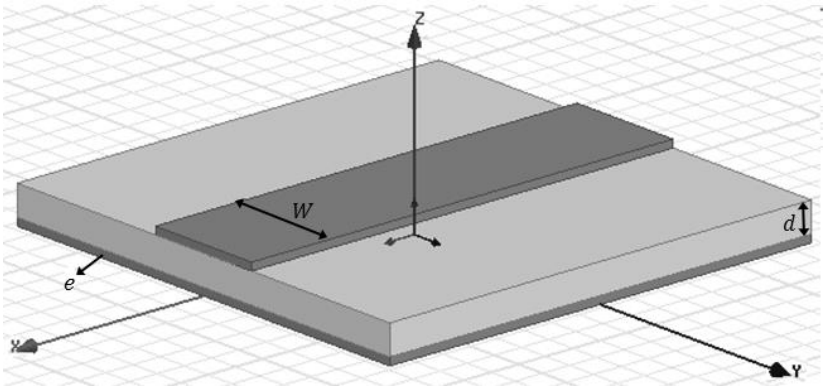


3.1.1.1 Tecnología Microstrip-line

Para la construcción de los filtros pasa-banda y la antena se escoge el método de estructura *Microstrip* [13], [15] en placas de circuito impreso (*Printed Circuit Board*, PCB) debido a la facilidad de implementación, bajo costo y de ofrecer una respuesta adecuada a los requerimientos del proyecto. En esta tecnología se usa una franja de conducción que se encuentra separada de la franja de masa por una capa de sustrato dieléctrico. Las líneas *Microstrip* se utilizan en diseños PCB digitales de alta velocidad donde las señales deben ser enrutadas con la mínima distorsión posible, evitando el ruido y las interferencias de radiación.

Se escogen las placas RT/duroid® 5880 y 6202 de ROGERS® Corporation para el diseño de la antena y los filtros. Estas placas se distinguen por las características de los materiales tales como ε_r, que es la constante dieléctrica del material aislante, y tan, que es la tangente de pérdidas o factor de disipación, y características físicas como el espesor de la placa de cobre e, el grosor del dieléctrico d y el ancho de los caminos conductores de cobre W. En la Figura 26 se muestra un esquema de una estructura en esta tecnología.

Figura 26. Configuración típica de una estructura Microstrip.



3.1.1.2 Antena de ultra ancho de banda

Se utiliza la tecnología *Microstrip* en configuración circular con acople; ya que brinda un mayor ancho de banda y un patrón de radiación omnidireccional. Debido a que el sustrato utilizado en el diseño de la antena tiene una altura $d < 0.05\,\lambda$ que es despreciable, donde λ es la longitud de onda de la frecuencia resonante, entonces se puede considerar el eje Z como constante y la frecuencia de resonancia se determina con la ecuación:

$$f_r = \frac{1.8412(C_0)}{2\pi a_{ef}\sqrt{\varepsilon_r}} \tag{28}$$

Donde C_0 es la velocidad de la luz en el vacío y a_{ef} es el radio efectivo. Este radio se define como:

$$a_{ef} = \frac{8.79108\ \text{x}10^7}{f_r\sqrt{\varepsilon_r}} \tag{29}$$

Para calcular el radio a de la antena se usa la expresión.

$$a = \frac{a_{ef}}{\left\{1 + \left(\frac{2d}{\pi\varepsilon_r a_{ef}}\right)\left[ln\left(\frac{\pi a_{ef}}{2d}\right) + 1.7726\right]\right\}^{0.5}} \tag{30}$$

Se utiliza la lámina RT/duroid® 6202 [9] que tiene las características:

$\varepsilon_r \quad = 2.94$
$d \quad = 1.574\ mm$
$e \quad = 33.02\ \mu m$
$tan = 0.0012$

Se procede al cálculo del radio. Con la ecuación (29) se calcula a_{ef} teniendo en cuenta que $f_r = 650\ MHz$ posteriormente con (30) se halla el radio $a = 31.7\ mm$.

La longitud Ls es la distancia mínima que debe existir entre el límite del sustrato y el radio del parche, ver Figura 27, y se calcula con la relación

$$Ls = 6d + 2a \tag{31}$$

Para el cálculo de la línea de acople se sigue el procedimiento descrito en [12]

$$\frac{W}{d} = \begin{cases} \dfrac{8e^A}{e^{2A} - 2} \ \ para\ \dfrac{W}{d} < 2 \\ \dfrac{2}{\pi}\left[B - 1 - \ln(2B-1) + \dfrac{\varepsilon_r - 1}{2\varepsilon_r}\left\{\ln(B-1) + 0.39 - \dfrac{0.61}{\varepsilon_r}\right\}\right] \ para\ \dfrac{W}{d} > 2 \end{cases} \tag{32}$$

Dónde:

$$A = \frac{z_0}{60}\sqrt{\frac{\varepsilon_r + 1}{2}} + \frac{\varepsilon_r - 1}{\varepsilon_r + 1}\left(0.23 + \frac{0.11}{\varepsilon_r}\right) \tag{33}$$

$$B = \frac{377\pi}{2Z_0\sqrt{\varepsilon_r}} \tag{34}$$

Los cálculos se hacen teniendo en cuenta la condición $\frac{w}{d} > 2$, y con la ecuación (5) se calcula el ancho del acople W_{50}.

La constante dieléctrica efectiva ε_e esta dada por la ecuación (35).

$$\varepsilon_e = \frac{\varepsilon_r + 1}{2} + \frac{\varepsilon_r - 1}{2}\left(\frac{1}{1 + \frac{12d}{W}}\right) \tag{35}$$

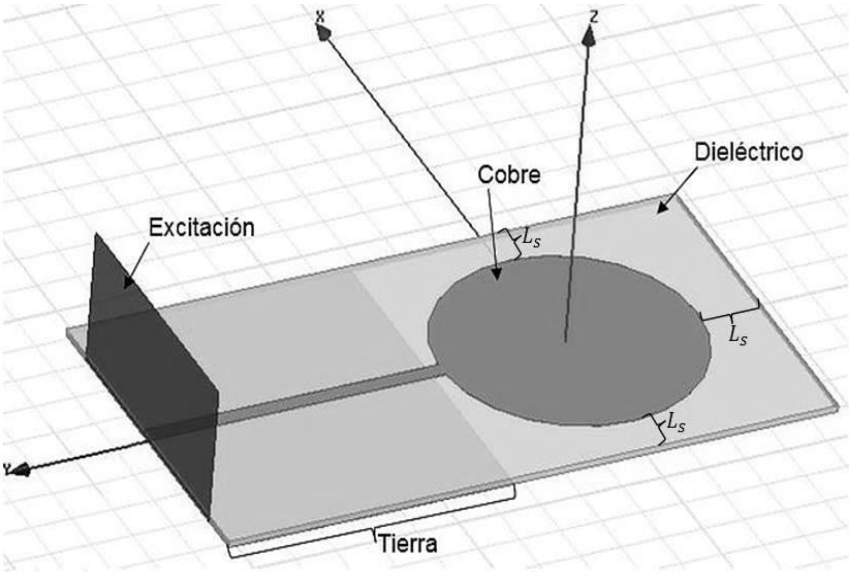
Teniendo ε_e se calcula la longitud de onda de la frecuencia central λ_0 teniendo en cuenta que la formula general es:

$$\lambda = \frac{c}{\sqrt{\varepsilon_e} f} \tag{36}$$

Donde f es la frecuencia de resonancia de la línea de acople ($f = f_r$) y c es la velocidad de la luz. Con λ_0 se obtiene la longitud del acople L_{50}, que es ¼ de λ_0.

Se optimizan algunos datos, esto para obtener el ancho de banda necesario. $W_{50} = 3.8mm$ y $L_{50} = 70mm$. Se procede a la construcción del modelo Figura 27 y posterior simulación 3D.

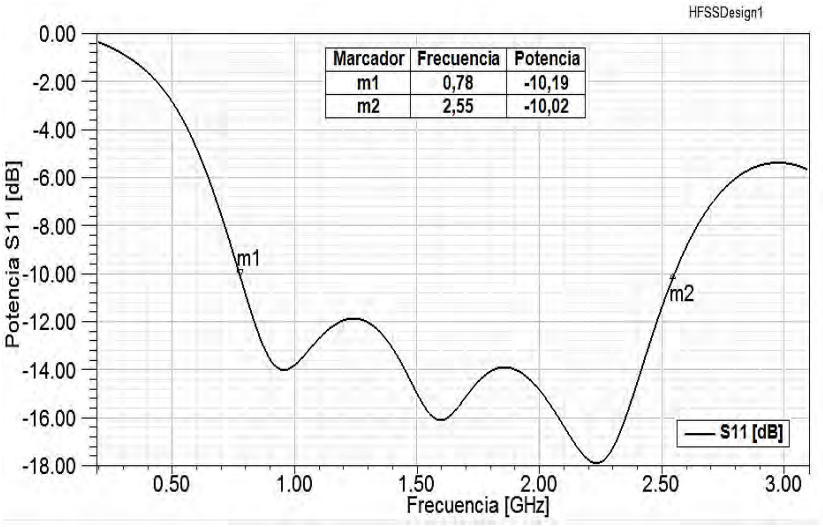
Figura 27. Modelo de simulación en 3D de la antena de ultra-ancho de banda tipo parche circular en tecnología *Microstrip*.



La respuesta en frecuencia del modelo se observa en la Figura 28, donde se varían algunos de los parámetros para obtener una mejor respuesta, para evaluar los parámetros de dispersión o parámetros S donde:

- S_{11}: también llamado coeficiente de reflexión del puerto de entrada, mide la cantidad de potencia que es reflejada en comparación con la cantidad de potencia que se esta aplicando en el puerto de entrada.
- S_{12}: mide la potencia de salida en el puerto de entrada en comparación con la enviada por el puerto de salida.
- S_{21}: mide la potencia recibida en el puerto de salida en comparación con la enviada en el puerto d entrada.
- S_{22}: mide la potencia recibida en el puerto de salida en comparación con la enviada en el puerto de entrada.

Figura 28. Respuesta en frecuencia de la antena de ultra-ancho de banda tipo parche circular en tecnología *Microstrip.*



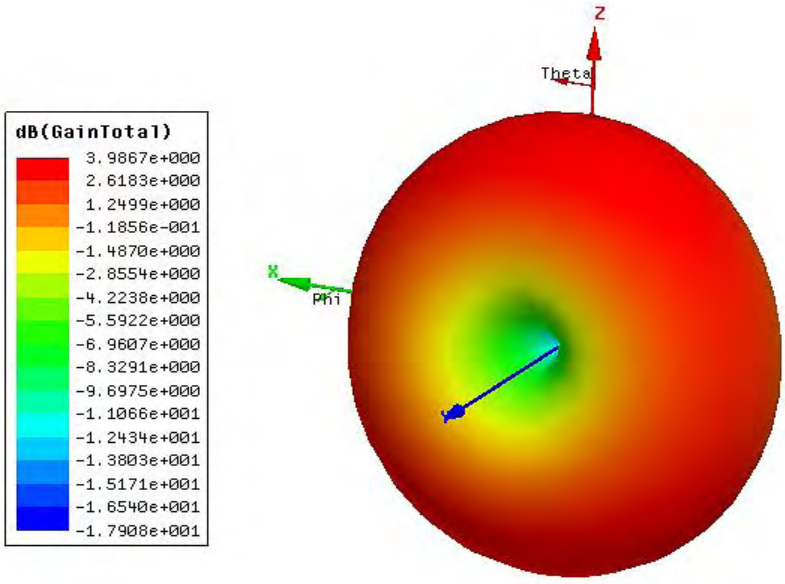
El parámetro S_{11} caracteriza la calidad de la antena y está relacionado con el parámetro de relación de onda estacionaria (Voltaje Standing Wave Ratio, VSWR), comúnmente proporcionado por fabricantes de antenas. VSWR es la relación entre el voltaje máximo y el mínimo de la señal presente a lo largo de la línea de transmisión. El valor ideal de VSWR es uno lo que significa que ningún porcentaje de la onda de entrada es reflejada, sin embargo valores de VSWR menores que dos son aceptables (porcentaje de reflexión menor que 10%). Esto se ilustra en la Tabla 1.

Tabla 1. Valores de VSWR, S_{11} y la potencia reflejada

VSWR	$\boldsymbol{S_{11}}$	Potencia Reflejada (%)
1	∞	0
1.1	26.44	0.228
1.2	20.83	0.816
1.3	17.69	1.71
1.4	15.56	2.78
1.5	13.98	4
1.6	12.74	5.5
1.7	11.73	6.8
1.8	10.88	8.2
1.9	10.16	9.6
2	**9.54**	**11**
3	6.02	24.9

Se realizan varias simulaciones dentro del rango de frecuencia de trabajo de la antena para observar su ganancia, una de ellas se ve en la Figura 29.

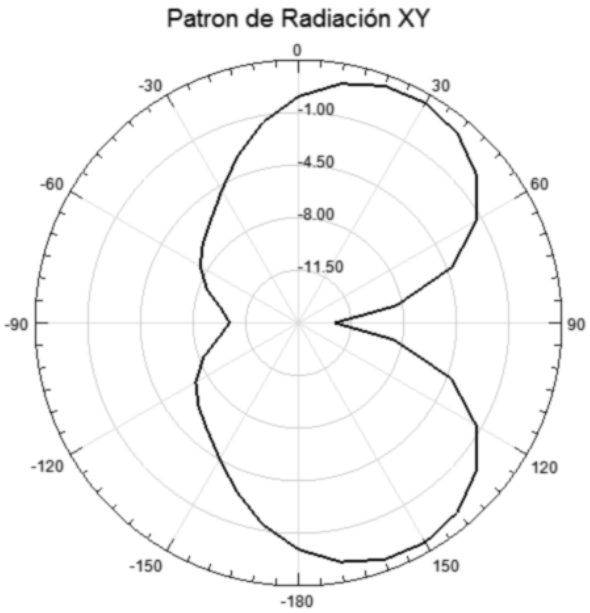
Figura 29. Ganancia de la antena de ultra-ancho de banda tipo parche circular en tecnología *Microstrip,* para una frecuencia de solución de 1.5 GHz.



El diagrama de radiación es la representación gráfica de la magnitud y dirección en que una antena radia su energía, esto proyectado en el plano vertical y horizontal. La Figura 30 y 31 presenta los cortes del patrón de radiación de la antena en los planos XY y XZ.

Figura 30. Patrón de radiación de la antena de ultra-ancho de banda tipo parche circular en tecnología *Microstrip*, corte en el plano XZ para una frecuencia de 1.5 GHz.

Figura 31. Patrón de radiación de la antena de ultra-ancho de banda tipo parche circular en tecnología *Microstrip*, corte en el plano XY para una frecuencia de 1.5 GHz.



3.1.1.3 Filtros pasa-banda

Para optimizar las características de diseño del filtro se utiliza el método de resonadores de comportamiento dual (*Dual Behavior Resonator,* DBR) [13] que permite una considerable reducción en el tamaño de los componentes y nuevas formas de acople electromagnético con circuitos de microondas.

Se requieren tres filtros pasa-banda en los rangos de 0.8 a 1 GHz, 1.8 a 2 GHz y 2.4 a 2.5 GHz. Las dos primeras son las bandas de trabajo utilizadas para comunicación celular y la última abarca las trasmisiones de la banda industrial, científica y médica (*Industrial, Scientific and Medical,* ISM), tales como las redes de área local inalámbrica. Se opta por una construcción de filtros de tercer orden, porque el tamaño del filtro es directamente proporcional a su orden y al rechazo, se busca un equilibrio entre el tamaño y la respuesta deseada. Se usa las ecuaciones propias de diseño de resonadores de comportamiento dual.

3.1.1.3.1 Cálculo filtro 0.8 a 1 GHz

Para el diseño del filtro de 0.8 a 1 GHz se utiliza la placa RTduroit 5880 [18] que tiene las siguientes características:

$\varepsilon_r \quad = 1.96$
$d \quad = 1.574\ \text{mm}$
$e \quad = 33.02\ \mu\text{m}$

$tan = 0.0019$

Los mismos pasos que en la realización del acople impedancia de la antena es seguido a continuación para el cálculo de los acoples de los filtros teniendo en cuenta que f en la ecuación (36) es la frecuencia central $f_c = 900\ MHz$ de la banda de operación del filtro.

Se sigue el procedimiento de diseño descrito en el artículo [13]. Las frecuencias de resonancia se eligen de manera conveniente siendo la inferior $f_{r1} = 0.6\ GHz$ y la superior $f_{r2} = 1.2\ GHz$. Las longitudes de onda de las frecuencias de resonancia se calculan con la ecuación (36).

La longitud del resonador L_{c1} es ¼ de λ_{r1}, al igual que Lc_2 es ¼ de λ_{r2}.

Para calcular las impedancias Z_{C1} y Z_{C2} se utiliza las ecuaciones (24), (25), (26) y (27) con un parámetro de pendiente de susceptancia $b = 5$

Para el cálculo del ancho de los resonadores se realizan los siguientes pasos:

Se calcula B_{C1} con la ecuación (34) reemplazando Z_0 por Z_{C1}, se encuentra W_{c1} usando la ecuación (32), reemplazando B_{C1} por B, Se realiza el mismo procedimiento anterior para el cálculo de W_{c2}.

Ahora se calculan los inversores con un ancho de banda fraccional $w_f = 30\%$, se tiene en cuenta los valores de la Tabla II para prototipos de filtros pasa-bajas máximamente planos de tercer orden [19].

Tabla 2. Valor de los elementos para prototipos de filtros pasa-bajas máximamente planos. ($g_0 = 1,\ w_c = 1,\ N = 1\ a\ 10$).

N	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}	g_{11}
1	2.0000	1.0000									
2	1.4142	1.4142	1.0000								
3	1.0000	2.0000	1.0000	1.0000							
4	0.7654	1.8478	1.8478	0.7654	1.0000						
5	0.6180	1.6180	2.0000	1.6180	0.6180	1.0000					
6	0.5176	1.4142	1.9318	1.9318	1.4142	0.5176	1.0000				
7	0.4450	1.2470	1.8019	2.000	1.8019	1.2470	0.4450	1.0000			
8	0.3902	1.1111	1.6629	1.9615	1.9615	1.6629	1.1111	0.3902	1.0000		
9	0.3473	1.0000	1.5321	1.8794	2.0000	1.8794	1.5321	1.0000	0.3473	1.0000	
10	0.3129	0.9080	1.4142	1.7820	1.9754	1.9754	1.7820	1.4142	0.9080	0.3129	1.0000

Fuente: *G.L. Matthaei, L. Young, y E.M. Jones, Microwave Filters, Impedance-Matching Networks, and Coupling Structures. Artech House, 1980.*

El cálculo del inversor de admitancia $J_{0,1}$ viene dado por la ecuación (21), el dato obtenido se reemplaza en la ecuación (20).para encontrar la impedancia $Z_{0,1}$

Con la ecuación (22) se calculan los demás inversores de admitancia. Teniendo los inversores de admitancia con (20) se obtienen las impedancias restantes. Para encontrar el inversor de admitancia $J_{3,4}$ se usa la ecuación (23).

Los valores $J_{1,2} = J_{2,3}$, $J_{0,1} = J_{3,4}$ por tanto los valores de impedancia $Z_{2,3} = Z_{1,2}$ y la impedancia $Z_{3,4} = Z_{0,1}$.

Teniendo las impedancias se procede al cálculo del ancho de los inversores, con la ecuación (34) se calcula $B_{0,1}$ reemplazando $Z_{0,1}$ por Z_0. Se toma la ecuación (32), se reemplaza $B_{0,1}$ por B para obtener $W_{0,1} = W_{3,4}$

Se realiza el procedimiento anterior para el cálculo de $W_{1,2}$, este valor es igual a $W_{2,3}$.

Para calcular la longitud de los inversores se utiliza la ecuación (35) para encontrar $\varepsilon_{e_{0,1}}$, este valor se reemplaza en la ecuación (36) esto para encontrar la longitud de onda $\lambda_{0,1}$. Con esta longitud de onda se calcula el largo del inversor $L_{0,1}$ y el valor $L_{0,1} = L_{3,4}$. Se utiliza el mismo procedimiento para calcular la longitud de los inversores $L_{1,2}$ y $L_{2,3}$, estos valores también son equivalentes.

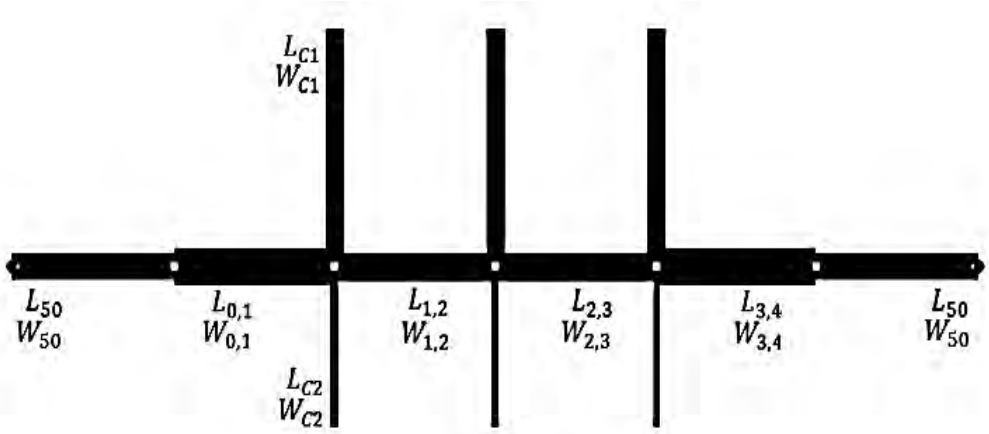
Con el fin de comparar los datos obtenidos se procede a ingresarlos a una página web especializada en el cálculo de líneas de trasmisión *Microstrip* [20] y se obtiene los resultados presentados en la Tabla 3.

Tabla 3. Parámetros de salida y dimensiones de un filtro DBR (0.8 - 1) GHz.

Parámetro	Teórico(mm)	Web(mm)
W_{50}	5.00	4.93
L_{50}	66.20	63.81
W_{c1}	6.90	6.73
L_{c1}	99.30	94.94
W_{c2}	1.60	1.54
L_{c2}	49.70	49.24
$W_{0,1} = W_{3,4}$	6.70	6.59
$L_{0,1} = L_{3,4}$	65.70	63.31
$W_{1,2} = W_{2,3}$	5.50	5.38
$L_{1,2} = L_{2,3}$	66.10	63.66

Con los valores de la Tabla 3 se realiza el diseño esquemático del filtro, Ver Figura 32. Este se modela en el software Advanced Design System ADS 2011.10 de Agilent Technologies, versión de prueba [21].

Figura 32. Modelo esquemático de un filtro DBR de tercer orden en tecnología *Microstrip*.



Se procede a la simulación teniendo en cuenta las características del material y parámetros de barrido en frecuencia, esto se muestra en la Figura 33.

También se realiza una simulación más ajustada a la realidad en 3D. En la Figura 34 se muestra este modelo de simulación.

Figura 33. Respuesta del filtro DBR de 0.8 a 1 GHz en el software ADS 2011.10 versión de prueba. f1 y f2 definen el ancho de banda y fr1 y fr2 son las frecuencias de resonancia.

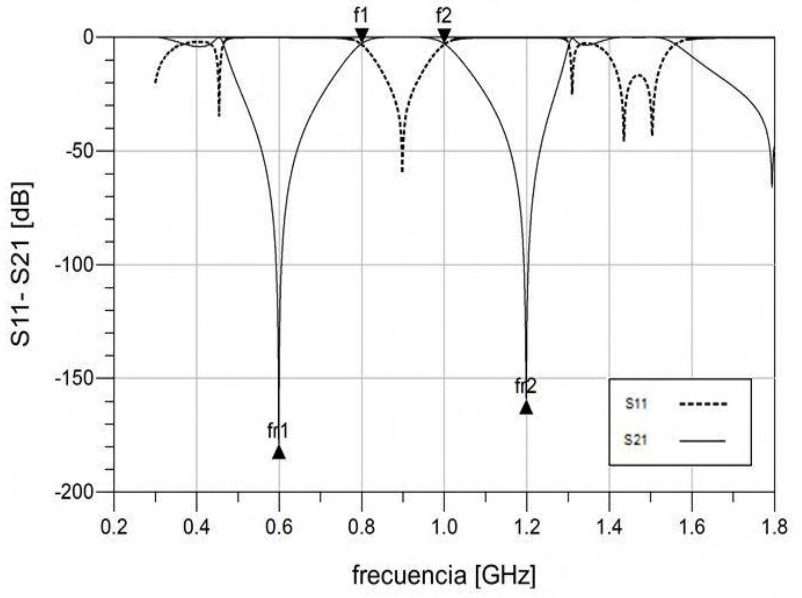
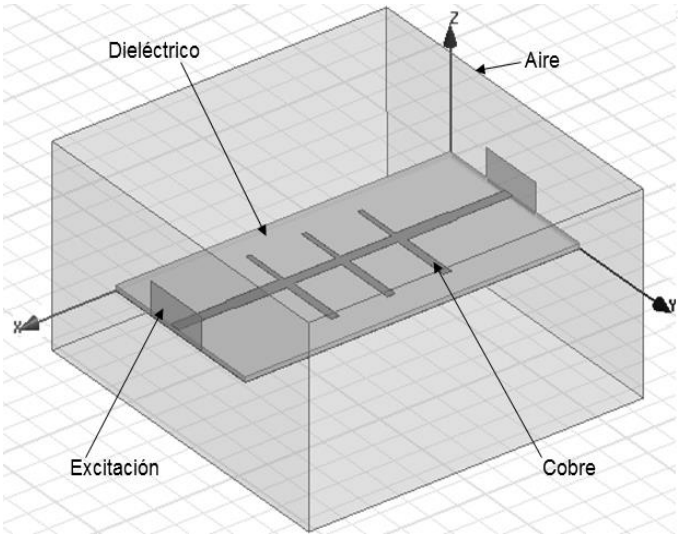
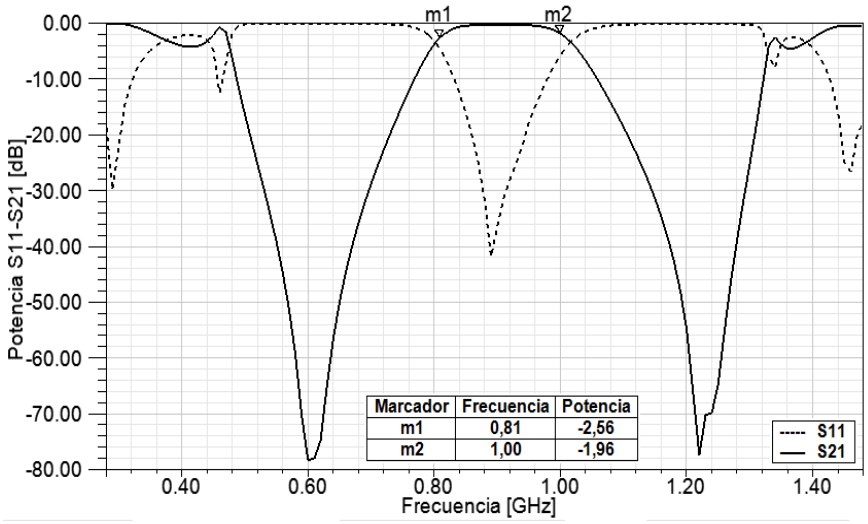


Figura 34. Modelo de simulación en 3D de un filtro DBR de tercer orden.



Se realizan varias simulaciones cambiando algunos de los parámetros para optimizar el diseño y obtener una mejor respuesta en frecuencia. Figura 35.

Figura 35. Respuesta en frecuencia del filtro DBR de tercer orden pasa banda de 0.8 a 1 GHz. Donde m1 y m2 indican el ancho de banda.



3.1.1.3.2 Cálculo del filtro 1.8 a 2 GHz

Para el diseño del filtro de 1.8 a 2 GHz se utiliza la placa ROGERS® RT/duroid® 6202, las frecuencias de resonancia utilizadas para los cálculos son $f_{r1} = 1.5\,GHz$ y $f_{r2} = 2.3\,GHz$, la frecuencia central $f_c = 1900\,MHz$, el parámetro de pendiente de susceptancia $b = 10$ y el ancho de banda fraccional $w_f = 15\%$, se tiene en cuenta los

valores de la tabla para prototipos de filtros pasa bajas máximamente planos de tercer orden, ver en Tabla 2.

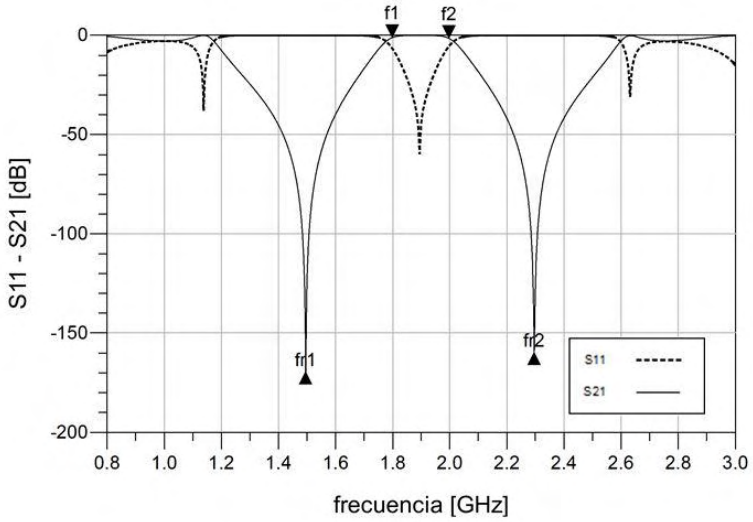
Se realizan los mismos pasos que para el cálculo del filtro de 0.8 a 1 GHz. Los resultados obtenidos se muestran en la Tabla 4.

Tabla 4. Parámetros de salida y dimensiones de un filtro DBR (1.8 - 2) GHz.

Parámetro	Teórico(mm)	Web(mm)
W_{50}	3.90	3.79
L_{50}	27.00	25.63
W_{c1}	3.00	2.98
L_{c1}	34.20	32.79
W_{c2}	1.20	1.20
L_{c2}	22.30	21.99
$W_{0,1} = W_{3,4}$	5.30	5.13
$L_{0,1} = L_{3,4}$	26.70	25.32
$W_{1,2} = W_{2,3}$	4.30	4.15
$L_{1,2} = L_{2,3}$	26.90	25.54

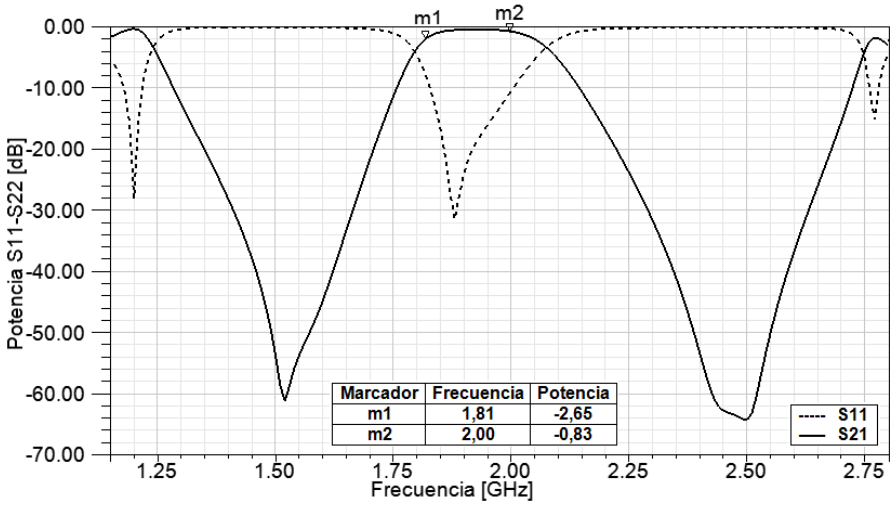
Con los valores de la Tabla 4 se procede a realizar el diagrama esquemático del filtro DBR y se procede a la simulación, Figura 36.

Figura 36. Respuesta del filtro de 1.8 a 2 GHz DBR en el software ADS 2011.10 versión de prueba. Donde f1 y f2 indican el ancho de banda y fr1 y fr2 son las frecuencias de resonancia.



La respuesta obtenida en la simulación se observa en la Figura 37.

Figura 37. Respuesta en frecuencia del filtro DBR pasa banda de 1.8 a 2 GHz. Donde m1 y m2 indican el ancho de banda.



3.1.1.3.3 Cálculo del filtro 2.4 a 2.5 GHz

Para el diseño del filtro de 2.4 a 2.5 GHz se utiliza la placa ROGERS® RT/duroid® 6202, las frecuencias de resonancia utilizadas para los cálculos son $f_{r1} = 2.1\ GHz$ y $f_{r2} = 2.8\ GHz$, la frecuencia central $f_c = 2450\ MHz$, el parámetro de pendiente de susceptancia $b = 23$ y el ancho de banda fraccional $w_f = 6\%$.

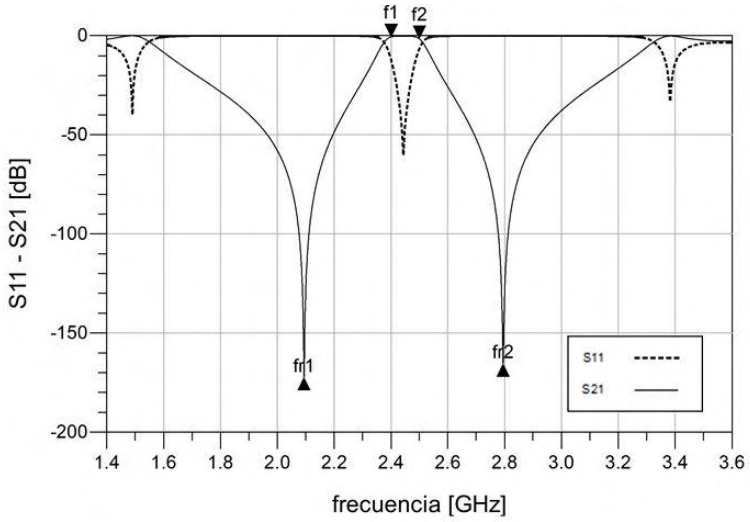
Se siguen los mismos pasos utilizados para el cálculo del filtro de 0.8 a 1 GHz, los resultados obtenidos se muestran en la Tabla 5.

Tabla 5. Parámetros de salida y dimensiones de un filtro DBR (2.4 - 2.5) GHz.

Parámetro	Teórico(mm)	Web(mm)
W_{50}	3.90	3.78
L_{50}	20.90	19.84
W_{c1}	3.00	2.91
L_{c1}	24.40	23.41
W_{c2}	1.70	1.67
L_{c2}	18.30	17.88
$W_{0,1} = W_{3,4}$	4.90	4.81
$L_{0,1} = L_{3,4}$	20.70	19.65
$W_{1,2} = W_{2,3}$	3.70	3.64
$L_{1,2} = L_{2,3}$	21.00	19.87

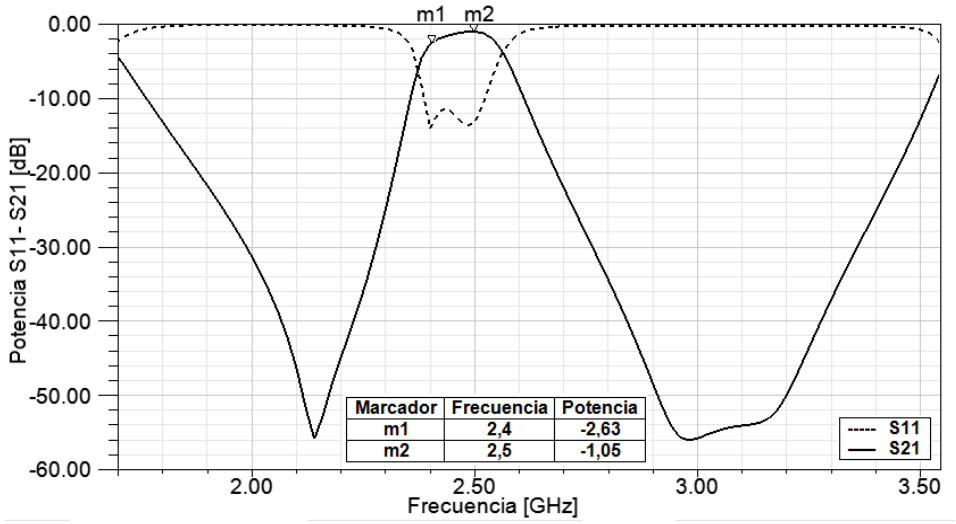
Con los valores de la Tabla 5 se procede a realizar el diagrama esquemático del filtro DBR y se procede a su simulación. Figura 38.

Figura 38. Respuesta del filtro 2.4 a 2.5 GHz DBR en el software ADS 2011.10 versión de prueba. Donde f1 y f2 indican el ancho de banda y fr1 y fr2 son las frecuencias de resonancia.



Se obtiene la Figura 39 como respuesta en frecuencia.

Figura 39. Respuesta en frecuencia del filtro DBR pasa banda 2.4 a 2.5 GHz. Donde m1 y m2 indican el ancho de banda



3.1.1.4 Filtro pasa-bajas

La salida del mezclador $x_3(t)$ se dirige a un filtro pasa-bajas de para evitar el efecto del *aliasing,* cancelando la banda superior, resultado de la suma de las frecuencias de las señales $x_2(t)$ y $x_{OL}(t)$, la señal resultante se denomina $x_4(t)$ que es la porción de espectro que es digitalizada.

Se tiene una señal compuesta por varias frecuencias donde solo la banda de f_i es necesaria para la digitalización y se debe garantizar que otras frecuencias del espectro original no sean adquiridas. Además se procura que la banda inferior de $x_3(t)$ sufra la mínima distorsión posible lo que implica la implementación de un filtro pasa-bajas de excelentes características como:

- Un alto factor de calidad.
- Bajas pérdidas de inserción.
- Acople de impedancia de 50 Ω.
- Un alto rechazo.

3.1.2 Sistema embebido

Una vez que se obtiene la señal en el rango de frecuencia deseado, esta etapa se encarga de digitalizar y analizar la señal $x_4(t)$. En este proyecto el sistema de desarrollo se compone: (i) un amplificador operacional para adecuar la señal (ii) un conversor análogo-digital (*Analog-to-Digital Converter*, ADC) de alta velocidad (iii) una FPGA, que se encarga de calcular la FFT de la señal $x_5[n]$ y finalmente, (iv) el resultado de este cálculo $X_6[n]$ se visualiza en un computador portátil, por medio de una interfaz gráfica de usuario. Ver Figura 1.

A continuación se describe más detalladamente el proceso de diseño de la etapa del sistema embebido, se comienza con la descripción matemática y de los dispositivos, concluye con la descripción de los algoritmos de programación.

3.1.2.1 Amplificador operacional de instrumentación

Para la digitalización de la señal se requiere que $x_4(t)$ sea procesada a través de un amplificador operacional de instrumentación puesto que la señal es débil, dicho dispositivo debe cumplir las siguientes características:

- Alta impedancia de entrada para evitar corrientes parásitas.
- Alto rechazo en modo común para garantizar pureza de la señal.
- Una respuesta adecuada en la banda de trabajo (20 MHz).
- Baja adición de ruido a la entrada.
- Baja distorsión armónica.

Existen en el mercado varias soluciones que cumplen con los requerimientos para esta etapa del diseño, algunas muy específicas para el acople de señales hacia conversores análogo-digitales de alta velocidad, muchos de estos amplificadores diferenciales especializados no requieren de complicadas interfaces de conexión. Además existe la posibilidad de adquirirlos en placas de desarrollo de costos aceptables para el proyecto.

3.1.2.2 Digitalización

La señal $x_4(t)$ amplificada necesita ser muestreada a una alta velocidad, pues dicha velocidad repercute en la visualización final de la DEP, de acuerdo a los objetivos se requiere de al menos una velocidad de 40 Mega muestras por segundo (Mmps) para cumplir el teorema de muestreo de Nyquist en la banda de DC hasta 20 MHz.

3.1.2.3 FPGA

Para comprender el cálculo y el proceso de visualización de la DEP de señales en varios intervalos del espectro electromagnético, se analizan algunas tecnologías digitales reconfigurables, por ejemplo las FPGAs y los procesadores digitales de señal (*Digital Signal Processing,* DSP), esto con el fin de elegir la mejor opción. En primer lugar se procede a una revisión bibliográfica del tema, usando como recursos literatura especializada y las búsquedas en Internet. Se elige una FPGA porque brinda una mayor velocidad de procesamiento que es de vital importancia para el cálculo de operaciones y visualización de los procesos en tiempo real.

3.1.2.3.1 Programación de la FPGA

En este apartado se presenta lo referente a la programación para la FPGA, donde se realizan las operaciones necesarias de acuerdo a los requerimientos del proyecto, se estudian posibles soluciones y se elige debido a la simplicidad y fiabilidad una interfaz serial entre el PC y la FPGA, dicha elección acarrea grandes ventajas, entre las cuales se encuentra la posibilidad de mejorar la interfaz a una transmisión USB simulando un puerto serial COM convencional, que se puede lograr fácilmente mediante la adquisición de un cable comercial genérico y de bajo costo que permite su instalación mediante controladores gratuitos en equipos con diferentes sistemas operativos. Otra ventaja notable de una transmisión serial de datos radica en que los dispositivos conectados a través de este puerto son reconocidos como genéricos que envían datos binarios al sistema de cómputo, así que no es necesario desarrollar un controlador personalizado para que la comunicación con el PC sea adecuada. Por otra parte, la interfaz de usuario que se diseña en MATLAB® [22] no tiene problemas para manejar la información adquirida ya que posee las herramientas de software necesarias para dicha tarea.
Se observa entonces la conveniencia de separar en tres etapas la programación del dispositivo, con el fin de probar y validar su correcto funcionamiento; de la siguiente manera:

- Programación de la interfaz de comunicación de la FPGA con el PC a través del puerto serial RS232.
- Programación de la interfaz de comunicación entre el modulo del ADC y la FPGA.
- Programación del algoritmo de FFT.

3.1.2.3.2 Programación de la interfaz de comunicación del FPGA con el PC a través del puerto serial RS232.

Es conveniente un intercambio de información de dos vías entre los dispositivos que se realiza de forma asíncrona, esto es útil ya que permite el envío no solo de los datos hacia el PC sino también cabe la posibilidad de enviar órdenes a la FPGA a través de comandos sencillos para cambiar parámetros en tiempo de ejecución del programa.

3.1.2.3.3 Programación de la interfaz de comunicación entre el modulo del ADC y la FPGA.

La mayoría de ADCs transmiten en paralelo por lo que se necesita un bus de datos para la transmisión de información en codificación binaria entre los dispositivos, esto asegura una alta tasa de transmisión ya que consiste en enviar datos en forma simultánea por varios canales sincronizados de alta velocidad, minimizando errores y latencias.

3.1.2.4 Programación del algoritmo de FFT

El análisis de Fourier es un proceso en el cual una señal es descompuesta en ondas sinusoidales con variaciones en frecuencia y amplitud. Esto se ejecuta para facilitar operaciones matemáticas de una señal en el dominio de la frecuencia y observar el contenido de frecuencias de la misma. Hay cuatro representaciones de Fourier distintas, cada una aplicable a una clase diferente de señales, se definen por medio de las propiedades de periodicidad y dependiendo si es en tiempo discreto o continuo como se indica en la Tabla 6.

Tabla 6. Relación entre las propiedades de tiempo de una señal y su representación de Fourier apropiada

Propiedad de tiempo	Periódica	No periódica
Continua	Serie de Fourier	Transformada de Fourier
Discreta	Serie de Fourier en tiempo discreto	Transformada de Fourier en tiempo discreto

La transformada discreta de Fourier (*Discrete Fourier Transform*, DFT). Puede ser expresada como la ecuación (37).

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W^{nk} \qquad K = 0,1,\dots,N-1 \tag{37}$$

Donde N es el número de muestras y W son constantes conocidas como factores *Twiddle* que son coeficientes trigonométricos constantes definidos como:

$$W^n = \cos\left(\frac{2\pi}{N}n\right) - j\,\mathrm{sen}\left(\frac{2\pi}{N}n\right) \tag{38}$$

El número de sumas complejas que se deben realizar es de $(N-1)N$ y la cantidad de multiplicaciones complejas asciende a N^2. Es claro que ésta cantidad de operaciones es alta y requiere de un enorme poder de cálculo.

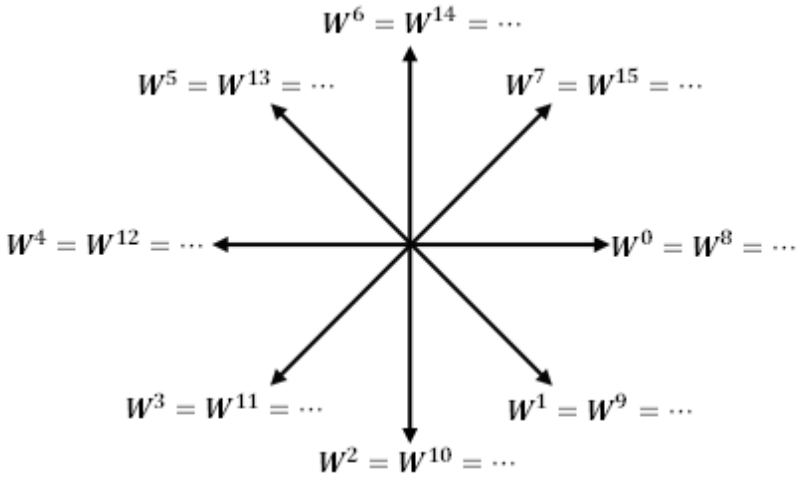
El cálculo directo de la DFT no es eficiente debido, fundamentalmente, a que no explota las propiedades de simetría y periodicidad del factor W_N, ya que $W_0 = 1$ no es necesario realizar las N^2 multiplicaciones. Además existen las siguientes propiedades de periodicidad y simetría:

$$W^{K+N} = W^K \tag{39}$$

$$W^{K+\frac{N}{2}} = -W^K \tag{40}$$

La simetría y periodicidad de los factores W quedan de manifiesto en la Figura 40. El ejemplo mostrado en dicho diagrama es para N = 8.

Figura 40. Periodicidad y simetría de los factores *Twiddle* para N=8



Es posible llegar a un método de cálculo mucho más eficiente, que entrega los mismos resultados con un número menor de operaciones. Es llamado algoritmo FFT.

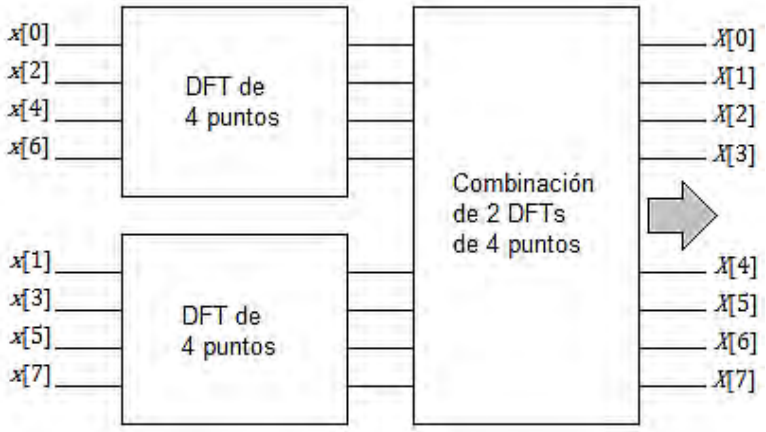
Entre los algoritmos más sobresalientes para la solución de la FFT se encuentran: Algoritmo Cooley-Tukey [23], Algoritmo del factor primo [24], Algoritmo de Bruun [25], Algoritmo Rader [26], Algoritmo de Bluestein [27].

Se elige trabajar con el algoritmo Cooley-Tukey, puesto que es el más popular y documentado. En busca de un equilibrio de uso de recursos hardware y velocidad se opta por *radix*-4, puesto que el rendimiento del algoritmo es determinado por las

tecnologías de procesamiento actuales que permiten algoritmos con bases más grandes a la convencional (*radix*-2), lo que significa una disminución considerable de operaciones que repercute en el tiempo de ejecución.

El algoritmo que se presenta en este trabajo descompone la DFT de 1024 puntos en transformadas más pequeñas. La DFT es descompuesta en dos DFTs de 512 puntos, cada una se descompone a su vez en dos DFTs de 256 puntos y así sucesivamente. Al final de la descomposición se obtienen 256 DFTs de 4 puntos. La transformada más pequeña determina la base de la FFT. Para una FFT *radix*-4, N debe ser una potencia de 4 y la transformada más pequeña es la DFT de 4 puntos.

Figura 41. FFT radix-4 de 8 puntos, basada en Proakis y Manolakis, Digital signal processing, 2003



El diezmado de la secuencia de datos se repite $v = log_4 N$ veces, ya que se tienen $N = 4^v$ datos. Por lo tanto el número total de multiplicaciones complejas se reduce a $\left(\frac{N}{4}\right) log_4 N$, mientras que el número de sumas complejas es $N log_4 N$.

3.1.2.5 Software de comunicación y visualización en PC

La programación en el PC se divide en dos secciones: en primer lugar la lectura y transmisión de datos en forma serial y la presentación de la DEP usando una interfaz gráfica de usuario.

3.1.2.5.1 Comunicación serial

La herramienta de diseño MATLAB® posee *scripts* y comandos específicos que permiten la correcta configuración de una comunicación serial de dos vías entre el PC y un dispositivo electrónico en la que el programador tiene la libertad de establecer todos los parámetros del protocolo de comunicación serial.

3.1.2.5.2 Interfaz gráfica de usuario

Se utiliza el entorno de desarrollo de interfaz gráfica de usuario de MATLAB® *(Graphical User Interface Development Enviroment*, GUIDE) para crear un programa que permita presentar los resultados al usuario de la señal correspondiente de la DEP, así como algunas herramientas comunes en analizadores de espectro comerciales para la manipulación de la interfaz.

GUIDE es un entorno de programación visual para realizar y ejecutar programas que necesiten ingreso continuo de datos, provee aplicaciones de software de fácil manejo eliminando la necesidad de utilizar un lenguaje de programación para correr una aplicación, brinda la posibilidad de automatizar tareas de cálculo mediante el uso de menús, barras de herramientas, botones, barras de desplazamiento entre otros.

Posibilita herramientas para el diseño de interfaces personalizadas de acuerdo a la aplicación requerida, con todas las ventajas que ofrece el lenguaje de alto nivel especializado para cálculos matemáticos.

3.2 IMPLEMENTACIÓN

3.2.1 Receptor superheterodino

3.2.1.1 Dispositivos construidos bajo la tecnología Microstrip-line

Teniendo las placas de ROGERS® Corporation a disposición se procede a la transferencia del diseño utilizando un *plotter* de corte de alta precisión y papel vinilo, se sigue el procedimiento tradicional de implementación de tarjetas PCB que utiliza como agente corrosivo tricloruro de hierro. Se protege las pistas de cobre usando una mezcla de colofonia y alcohol industrial.

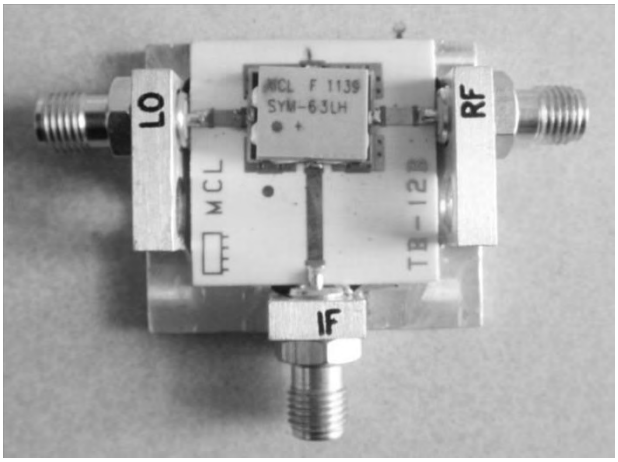
Se compran terminales dorados SMA de alta calidad para soldar a los acoples de impedancia, garantizan muy bajas pérdidas en la implementación de dispositivos construidos bajo la tecnología *Microstrip*.

Se construyen cajas de aluminio para los filtros que actúan como jaulas de Faraday y protección física de los dispositivos, a la vez que brindan una mejor manipulación y presentación visual.

3.2.1.2 Mezclador SYM-63LH+ soldado a la placa TB-12

Se utiliza el mezclador de montaje superficial de doble balanceo de referencia SYM-63LH+ nivel 10 (LO Power +10 dBm) y ancho de banda de 1 a 6000 MHz de la empresa Mini-Circuits® [28], montado sobre su respectiva *board* de evaluación TB-12 [29] como se mira en la Figura 42.

Figura 42. Mezclador SYM-63LH+ soldado a la placa TB-12



Los terminales de conexión SMA presentes en la placa TB-12 son convenientes en la implementación ya que facilita ya que facilita la conexión con las demás etapas del sistema, impidiendo el ingreso de ruido de elevada amplitud y permite la flexibilidad del proyecto.

3.2.1.3 Filtro pasa-bajas PLP-70+

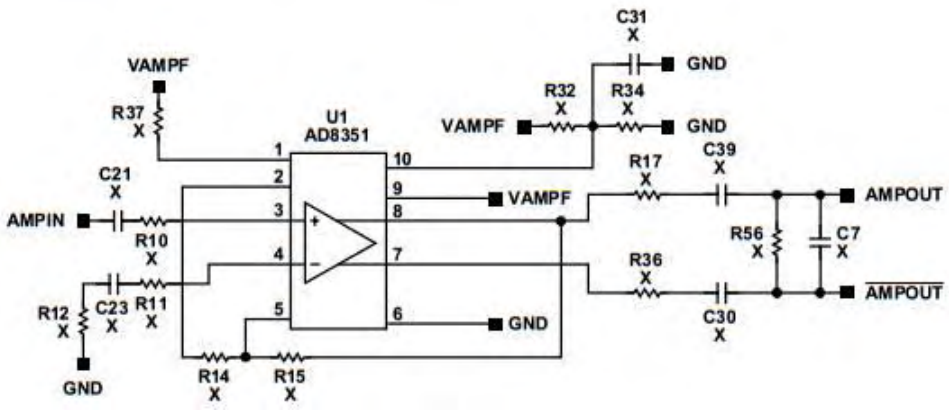
De Mini-Circuits® se adquiere el filtro pasa-bajas PLP-70+ [30] que satisface todos los criterios de diseño planteados, su costo es aceptable para el proyecto y sus dimensiones físicas adecuadas para el diseño y construcción de una placa PCB tradicional con terminales SMA dorados de alta calidad, se encuentra acoplado a 50 Ω, tal como el resto de los elementos del sistema, su banda de paso es de DC hasta 60 MHz que cubre la ventana de visualización de la DEP requerida.

3.2.2 Sistema embebido

3.2.2.1 AD9481 PCBZ

Los siguientes dispositivos planteados en el diseño son un amplificador operacional y un ADC, los cuales se encuentran integrados en la placa de evaluación AD9481 PCBZ [10] fabricada por Analog Devices Inc. El amplificador diferencial disponible en la placa es el AD8351 [31] que cuenta con un circuito de aplicación optimizado para funcionar en conjunto con el conversor AD9481 de 8 bits de precisión como se muestra en la Figura 43, el circuito del amplificador esta deshabilitado por defecto, pero con la adición de algunas resistencias y capacitores de montaje superficial y la desconexión de otros componentes se logra activar.

Figura 43. Circuito de aplicación amplificador operacional diferencial AD8351



El ADC funciona en el rango de frecuencias de reloj desde 20 MHz hasta 250 MHz, la placa de evaluación cuenta con un puerto SMA para la conexión de un sistema de reloj de alta estabilidad. Para obtener una excelente exactitud y precisión se adquiere un oscilador controlado por horno (OCXO) de referencia AOCJY1 100MHz de Abracon® Corporation [11], el cual genera ondas RF de forma sinusoidal o CMOS, frecuentemente utilizado en aplicaciones de infraestructura celular, sistemas de radar y equipamiento de medición.

El puerto de entrada analógica es de conexión SMA, la placa posee una bornera para la conexión de diferentes fuentes de voltaje necesarias para el funcionamiento del sistema, pero este puede funcionar en modo de voltaje común a 3.3 V si se cuenta con una fuente de alimentación DC de alta calidad, la cual se puede obtener del kit de desarrollo ALTERA® DE2 desarrollado por TerasIC Technologies Inc. [8] ya que el sistema necesita baja potencia para su funcionamiento.

Los pines de salida están configurados bajo el estándar 40-*pin header* con separación entre pines de 2,54 mm y altura de 11,5 mm la cual es utilizada en el kit de evaluación ALTERA® DE2 [9] y la construcción del bus de datos es sencilla.

3.2.2.2 FPGA

La tarea más importante que ejecuta la FPGA es un algoritmo eficaz que realiza una FFT de 1024 puntos a una velocidad adecuada para la presentación de información en tiempo real con precisión de datos de 8 bits en paralelo. Otra de las tareas de la FPGA es leer los datos binarios provenientes de la conversión análoga a digital que realiza el módulo de conversión de alta velocidad AD9481.

3.2.2.2.1 Programación de la interfaz de comunicación del FPGA con el PC a través del puerto serial RS232.

Se procede a comprobar la recepción de datos desde el PC haciendo uso de MATLAB® enviando la asignación de pines de la FPGA hacia la placa ALTERA® DE2. Se usa una codificación binaria de 8 bits, se prueba cada número enviado en decimal desde MATLAB® y se comprueba en la placa DE2 su contraparte binaria con señal lumínica de 8 leds consecutivos en la placa, como es posible representar 256 códigos diferentes con 8 bits, esta es la cantidad de pruebas a realizar, resultando todas exitosas.

El siguiente paso consiste en validar el envío de datos desde la FPGA para su posterior lectura en MATLAB®, en programación se procede a asignar los pines en la FPGA de tal manera que se puedan utilizar 8 de los interruptores disponibles en la placa de pruebas, así mismo, se cambia el método de transmisión de modo que en un periodo de espera modificable, se pueda forzar el envío de un código especifico a un *buffer* programado de tamaño variable. Se realizan pruebas exhaustivas con un buffer de 1024 puntos a la más alta velocidad de transmisión posible permitida por el controlador de RS232 del PC utilizado (128000 baudios).

El método inicial consiste en ingresar manualmente un código binario en los interruptores de la placa DE2, ejecutar un *script* de lectura del puerto serial en MATLAB® y presionar el pulsador programado en la placa para iniciar la captura de los datos; el resultado es una matriz 1x1024 que contiene la contraparte decimal del código binario ingresado. Una vez más se procede a realizar las 256 pruebas posibles con dos o tres repeticiones por cada una con el fin de observar que en efecto, el código programado no presenta errores en la captura de los datos.

Un paso más allá en las pruebas necesarias para validar el código consiste en ingresar un flujo cambiante en el tiempo de datos binarios a la FPGA para que MATLAB® pueda leerlos y graficar su resultado, entonces se procede a la programación de una memoria de solo lectura (*Read-Only Memory*, ROM) directamente alojada en la FPGA, la cual simula el comportamiento del conversor análogo a digital. Las siguientes son las características necesarias para dicha memoria ROM.

- La señal simulada debe ser reprogramable con facilidad y flexibilidad; donde cada parámetro de la señal pueda ser manipulado por el usuario.
- El consumo de recursos de la FPGA no debe ser muy elevado, así que no debe ser auto programable en tiempo de ejecución, la solución más viable consiste en la escritura de varios *scripts* en MATLAB®, los cuales crean un periodo de una señal con parámetros ingresados a conveniencia por el usuario, dicha información se codifica en binario y se escribe en lenguaje de descripción de Hardware de forma automática.

3.2.2.2.2 Programación de la interfaz de comunicación entre el modulo del ADC y la FPGA.

Para esta parte de la verificación del código y teniendo en cuenta que ya se realizó una simulación del comportamiento del ADC mediante la memoria ROM que se describió anteriormente, solo resta reemplazar este bloque de código por la conexión física del módulo del ADC AD9481PCBZ. Se realiza la captura de datos de la misma manera, pero esta vez surgen varios inconvenientes relativos a la calidad de las conexiones entre dispositivos, al manejo de osciladores en alta frecuencia que presentan un ruido eléctrico considerablemente elevado, la fuente de alimentación DC, que presenta un filtro no adecuado de la señal de 60 Hz de la red eléctrica, un rizado en el nivel DC elegido y finalmente a la fiabilidad, precisión y exactitud del equipamiento al que se tiene acceso (generador de señales 9205C de 2 MHz de Protek® Test and Measurement).

De inmediato se procede a la construcción de una interfaz más limpia entre los dispositivos, considerando las pérdidas que se introducen al sistema por conexión de cables de excesiva longitud y manejo de ancho de banda no garantizado en las frecuencias de trabajo del proyecto, una solución temporal consiste en seguir la recomendación de la hoja de datos del módulo ADC AD9481PCBZ que indica cables de conexión convencionales de cobre de máximo 2 pulgadas de longitud, esto mejora en cierta medida los resultados, pero después de observar un comportamiento no convencional en los datos obtenidos; se procede a mejorar las conexiones de fuente de alimentación y se reemplaza la fuente de alimentación externa, por la que puede obtenerse de la placa de pruebas ALTERA® DE2 previo estudio de los niveles de corriente requeridos. Esta es la opción más recomendable, ya que los métodos de filtrado de la señal de la red eléctrica del dispositivo son notablemente mejores, su nivel DC es justo el necesario y las exigencias de potencia son adecuadas.

Pruebas posteriores presentan una mejora en las frecuencias máximas que pueden obtenerse con el equipamiento al que se tiene acceso.

El siguiente paso es la prueba del sistema a su máxima frecuencia de muestreo (50 MHz) con equipamiento de mejor calidad y mejores prestaciones. Se tiene acceso entonces a un generador de funciones digital UTG9020A de 0 a 20 MHz de la marca Uni-Trend® Group Limited, que nos permite realizar un barrido de frecuencias adecuado a las características propuestas del prototipo. Estas últimas pruebas presentan notable mejoría, aunque permiten sin embargo concluir que existe la necesidad de adquirir cables de conexión con los más altos estándares de calidad.

Figura 44. Señal sinusoidal de 10MHz, prueba realizada con el generador de funciones UTG9020A de Uni-trend, 38 muestras visualizadas en MATLAB® con una frecuencia de muestreo de 50 MHz

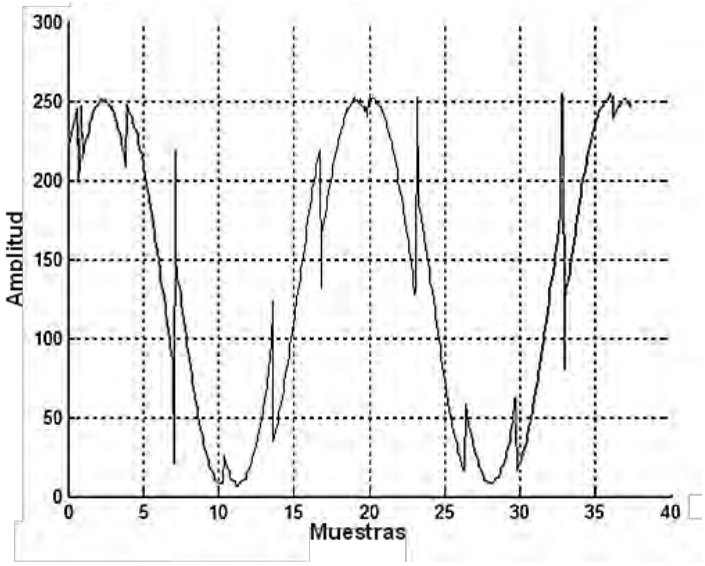
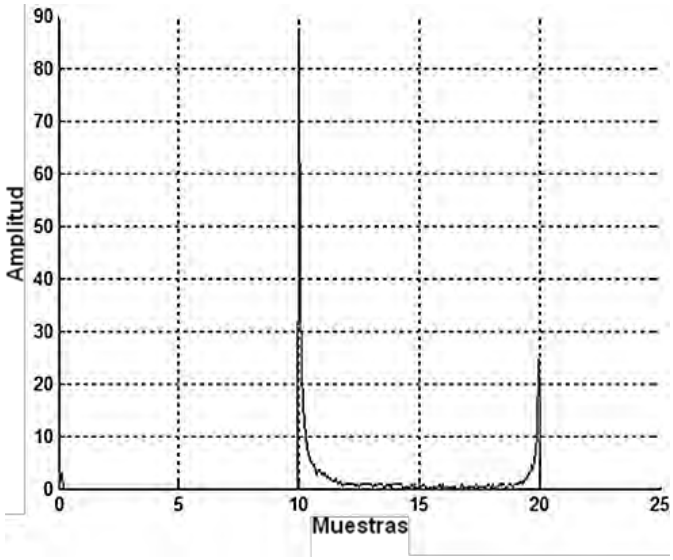


Figura 45. Espectro de la señal sinusoidal de 10MHz, prueba realizada con el generador de funciones UTG9020A de Uni-trend, 38 muestras visualizadas en MATLAB® con una frecuencia de muestreo de 50 MHz.



3.2.2.2.3 Programación del algoritmo de FFT

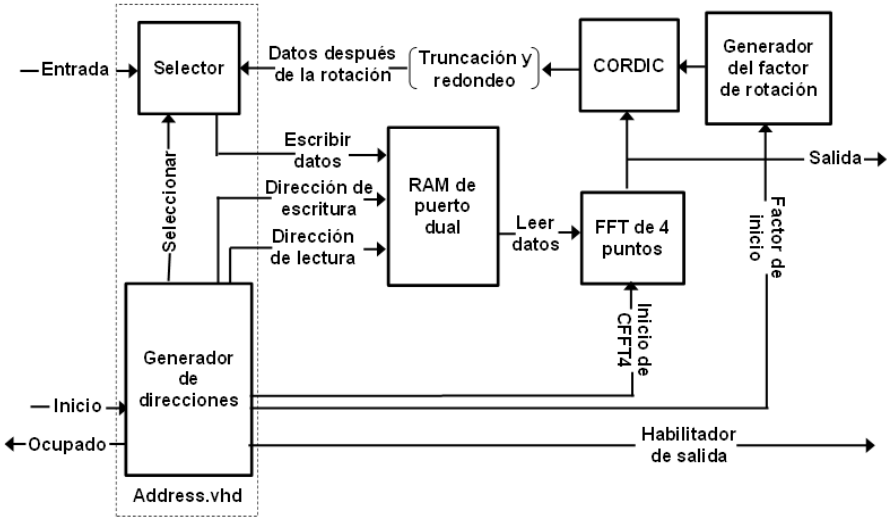
Utilizando el kit de desarrollo EP2C35 de ALTERA que basa su tecnología en la FPGA Cyclone® II ep2c35f672c6, las cuales están soportadas por el Software especializado QUARTUS® II [32] para el análisis y la síntesis de diseños realizados en HDL, permiten al desarrollador compilar sus diseños, realizar análisis temporal,

examinar diagramas de bloques y configurar el dispositivo de destino con el programador.

El núcleo de este prototipo se basa en el desarrollo de una FFT, para esto se utiliza el proyecto de código abierto CFFT de la página OpenCores.org [33] cuyo diagrama de bloques se muestra en la Figura 46. Diseñado originalmente para una plataforma de desarrollo de XILINX™ que es uno de los principales fabricantes de FPGAs y probada usando el software XILINX™ ISE4.1 en una FPGA de la serie VIRTEX® XVC50E-6, a 1024 puntos, una entrada de 12 bits con una entrada de reloj de 90 MHz, sin embargo es sintetizable en las principales plataformas de desarrollo para FPGAs como en el caso de ALTERA® Cyclone® II a la cual se tiene acceso.

El código genera transformadas de 256, 1024 y 4096 puntos, siendo 1024 el número de puntos elegido con entrada de datos configurable, en este caso 8 bits son permitidos. Este procesador de FFT compleja utiliza el algoritmo *radix-4*. El proyecto CFFT utiliza factores *Twiddle* implementados con el método de dígito por dígito (*COordinate Rotation DIgital Computer*, CORDIC) que es simple y eficiente para calcular funciones hiperbólicas y trigonométricas. Es utilizado típicamente por Microcontroladores y FPGAs, pues las únicas operaciones que requiere son sumas, restas, desplazamiento de bits y búsqueda en tablas [33].

Figura 46. Diagrama de bloques del proyecto CFFT



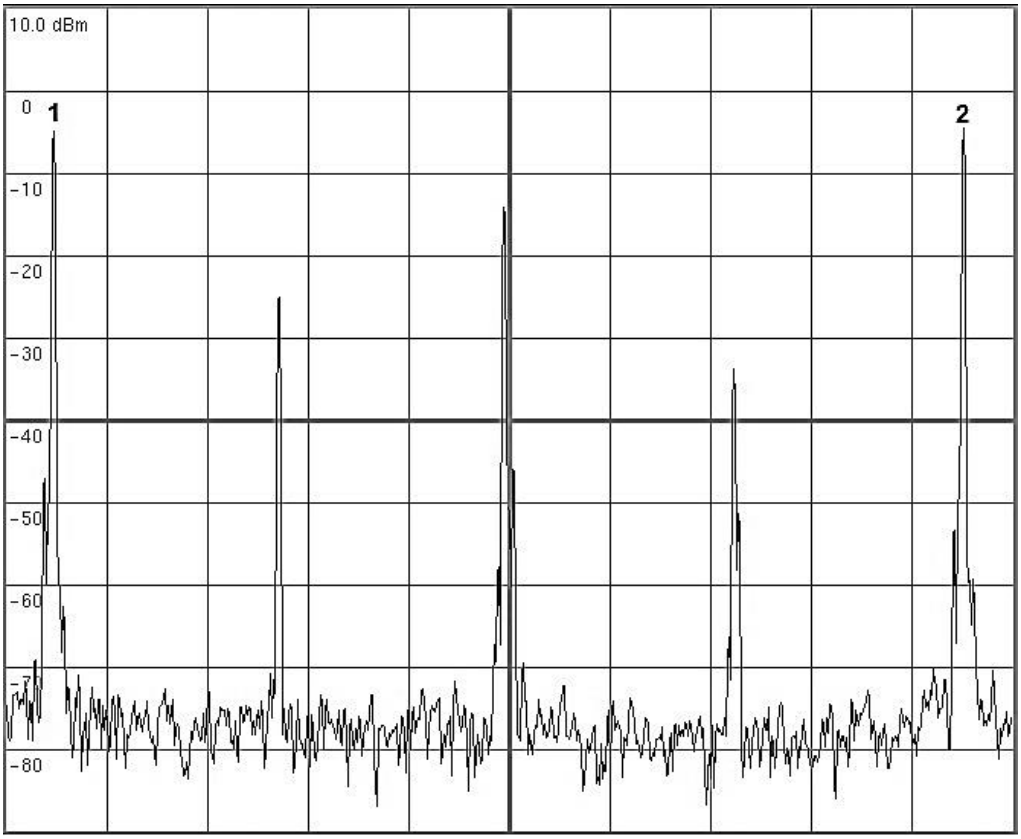
La ganancia que se obtiene después de la CFFT es fija, diferente del algoritmo FFT estándar pero se puede aplicar coeficientes constantes para conseguir una equivalencia al algoritmo de la FFT convencional.

4. RESULTADOS

Se logra los resultados deseados cumpliendo los objetivos propuestos, al culminar el proyecto se tiene un analizador de espectros modular con su núcleo basado en una FFT *radix-4* de código abierto, implementada en la FPGA Cyclone® II de ALTERA®, con un ancho de banda de 25 MHz y su banda de operación de hasta 2,7 GHz.

Se valida el funcionamiento del mezclador SMY-63LH+ conectando dos generadores de funciones de barrido Protek 9205C con salida sinusoidal a las entradas RF y LO del mezclador y se observa la respuesta en frecuencia haciendo uso del analizador de espectro Anritsu™ MS2723B la salida IF se muestra en la Figura 47.

Figura 47. Imagen del espectro en frecuencias de la salida IF del mezclador, donde en 1 se indica la diferencia y en 2 la suma de dos ondas sinusoidales.



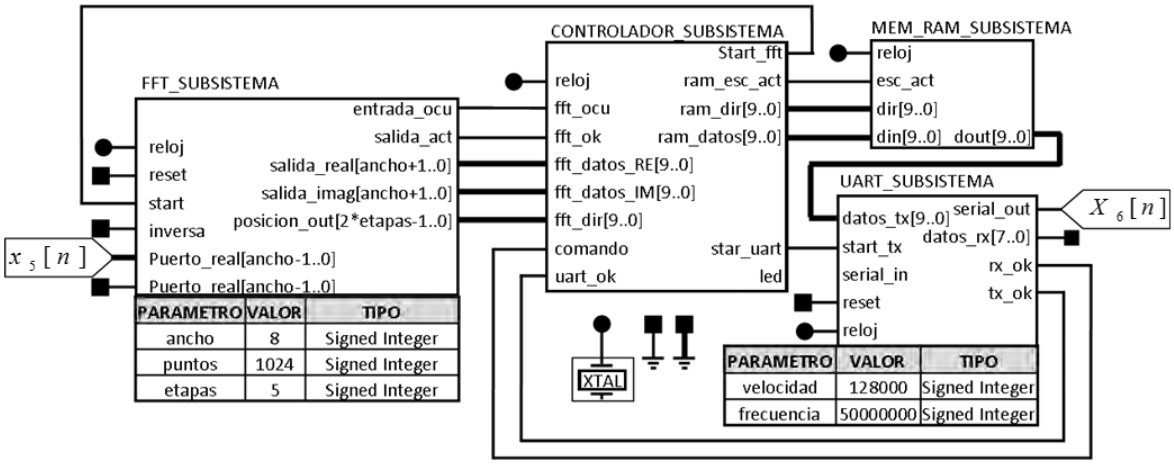
Se observa que el desplazamiento en frecuencia es el adecuado, para realizar la prueba se ingresan señales de frecuencia en $LO = 1498\ KHz$ y $RF = 996\ KHz$, en el marcador **1** se tiene $503{,}334\ KHz$ y en el marcador **2** $2493{,}334\ KHz$ que corresponden a los valores esperados.

La salida del mezclador se dirige al filtro pasa-bajas de 60 MHz de referencia PLP 70+, se prueba el filtro con generador de funciones UTG9020A y se observa que las pérdidas son despreciables en la banda paso y en la banda de rechazo se mira únicamente piso de ruido que indica una excelente calidad del filtro.

4.1 SISTEMA EMBEBIDO

Una vez la señal proveniente del mezclador de frecuencias es muestreada en tiempo por el ADC, la FPGA calcula la FFT de la señal digitalizada entrante, el diagrama de bloques de programación desarrollado se ve en la Figura 48.

Figura 48. Diagrama de bloques del programa implementado en la FPGA Altera Cyclone.



La etapa FFT_SUBSISTEMA es la etapa que se adecua a partir del proyecto CFFT de Opencores.com, en la Figura 48 se muestran los parámetros de funcionamiento utilizados, los cuales indican una precisión de punto fijo de 8 bits en paralelo, una FFTde 1024 puntos y una resolución de 5 etapas es decir$log_{radix}(N) = etapas$.

El sistema tiene seis entradas y dos salidas, el reloj del sistema es común a todos los bloques, se utiliza el del kit de desarrollo ALTERA® DE2 de 50 MHz, para mayor flexibilidad del sistema el bloque de comunicación serial es parametrizable, en cuanto a su velocidad de procesamiento e intercambio de datos, en este caso se utiliza una velocidad de 128000 baudios que es la más alta que alcanza el PC.

La etapa CONTROLADOR_SUBSISTEMA en conjunto con el módulo de memoria RAM MEM_RAM_SUBSISTEMA se encarga de ejecutar las tareas de reordenamiento de los datos provenientes del módulo de la FFT, calcula la DEP que es guardada temporalmente en el módulo de memoria hasta ser enviado al módulo

UART_SUBSISTEMA que transmite los datos bajo el protocolo RS232 al PC. Se conecta la FPGA al PC por medio de una conexión serial-USB para luego mostrar en pantalla a través de una interfaz gráfica de usuario desarrollada en la herramienta GUIDE del software MATLAB®.

La verificación del código CFFT se realiza mediante la comparación directa de este algoritmo con el algoritmo de MATLAB®, que se observa en la Figura 50 y Figura 51. Para ambas pruebas se computa la FFT de 1024 puntos y se utiliza la misma matriz de entrada, Figura 49.

Figura 49. Matriz de entrada, 1024 muestras de una señal cuadrada de 3 MHz con un voltio de amplitud, obtenida del generador de funciones digital UTG9020A visualizada en MATLAB®

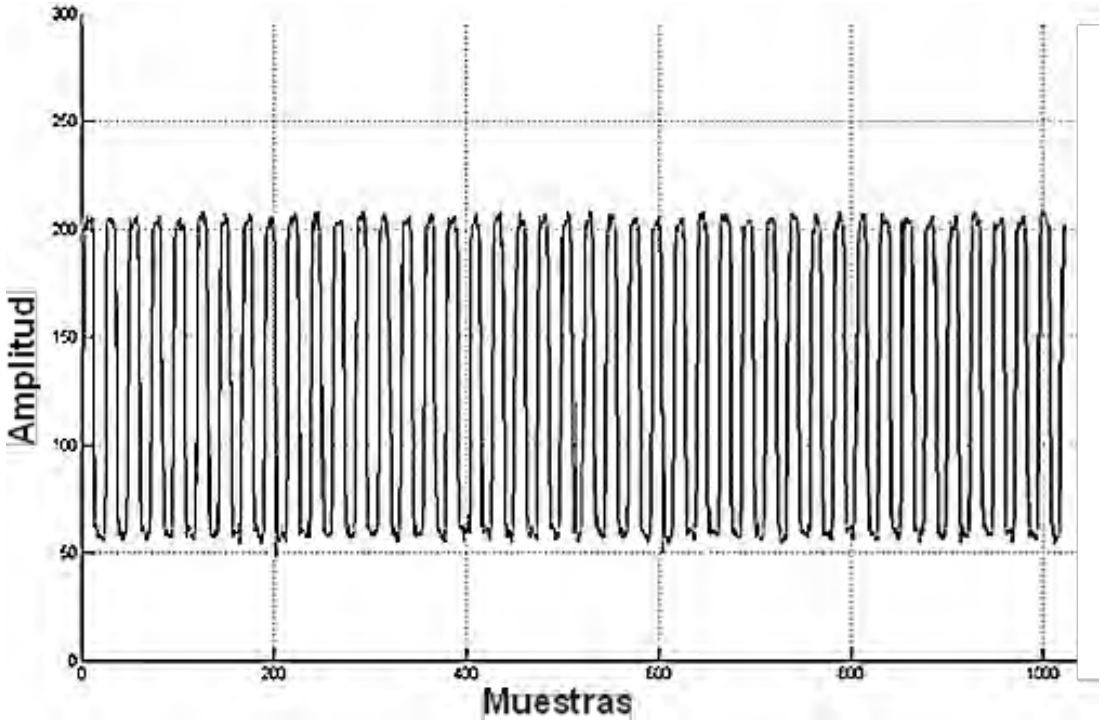


Figura 50. FFT de 1024 puntos de la señal cuadrada de 3 MHz. Figura 49. Algoritmo de MATLAB®

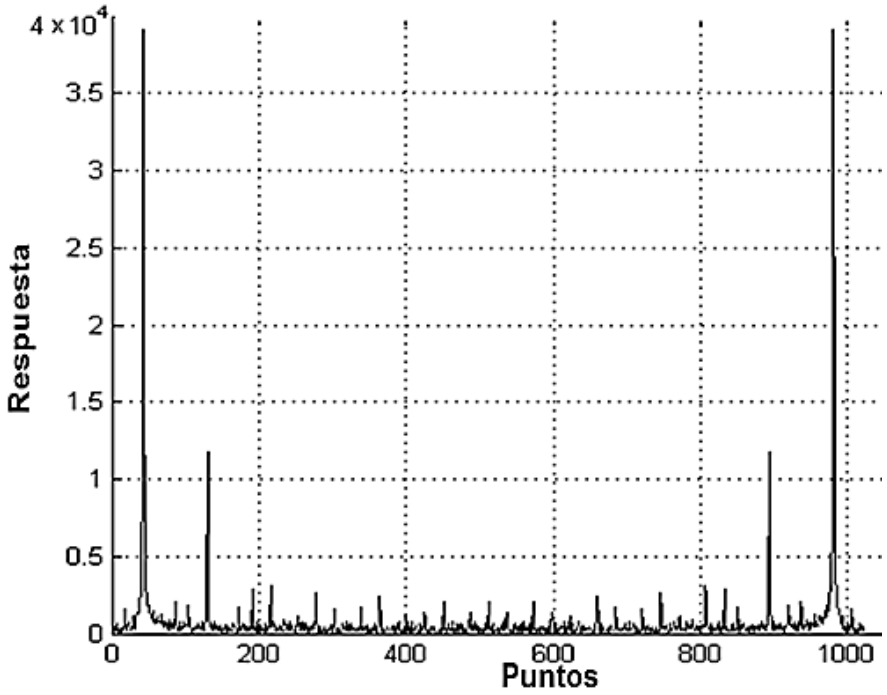
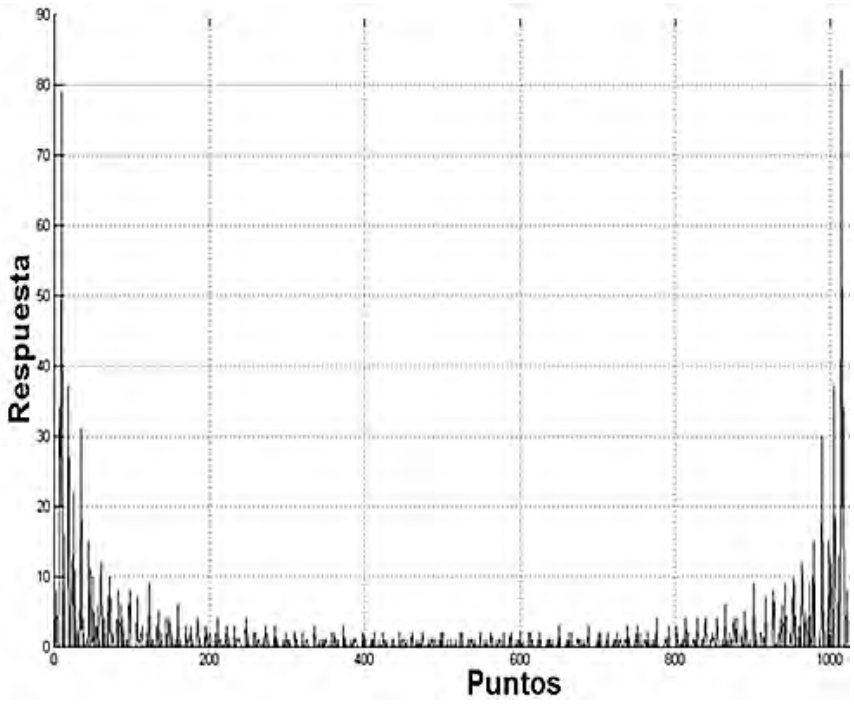


Figura 51. FFT de 1024 puntos de la señal cuadrada de 3 MHz. Figura 49. Algoritmo CFFT.



Debido a que el algoritmo de MATLAB® se calcula a una resolución de 32 bits con precisión doble en punto flotante, mientras que el algoritmo CFFT fue configurado para funcionar con una resolución de 8 bits con precisión en punto fijo siendo esta la configuración más alta soportada por la FPGA. Es notable la diferencia en los resultados, pero es claro que los algoritmos tienen un funcionamiento similar lo que nos permite continuar con la verificación de otras etapas del sistema.

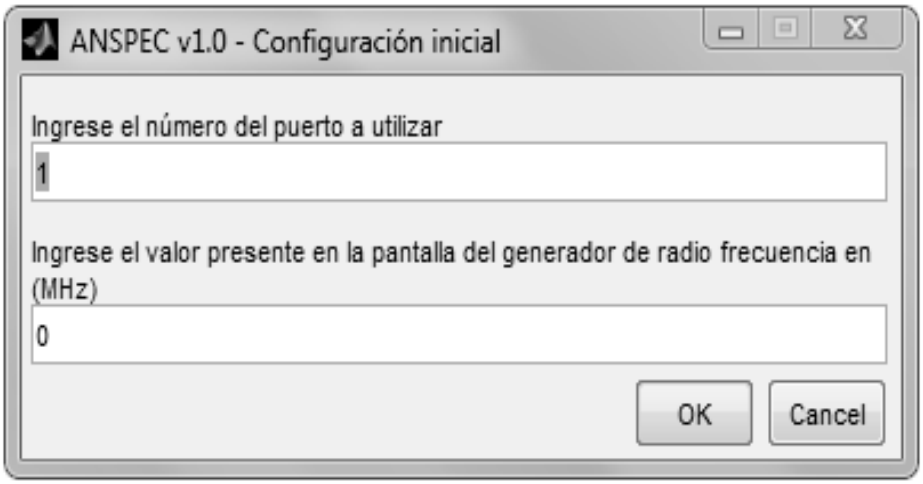
Para mejorar la calidad de los resultados, teniendo en cuenta a las propiedades de simetría de la transformada de Fourier, se opta por realizar el promedio aritmético punto a punto las mitades derecha e izquierda de la matriz y desplazar las componentes de frecuencia cero al centro del espectro.

Se comparan los resultados con los obtenidos mediante el comando `fftshift` de MATLAB® el cual desplaza las componentes de frecuencia cero al centro del espectro, e intercambia las mitades derecha e izquierda de la matriz.

4.2 INTERFAZ GUIDE

La interfaz de usuario se realiza usando la herramienta GUIDE de MATLAB®. Al ejecutar el programa Standalone[3] la primera ventana en mostrarse, solicita al usuario ingresar la frecuencia que desea analizar y el número de puerto COM donde se realiza la conexión física de la FPGA al computador, como se muestra en la Figura 52.

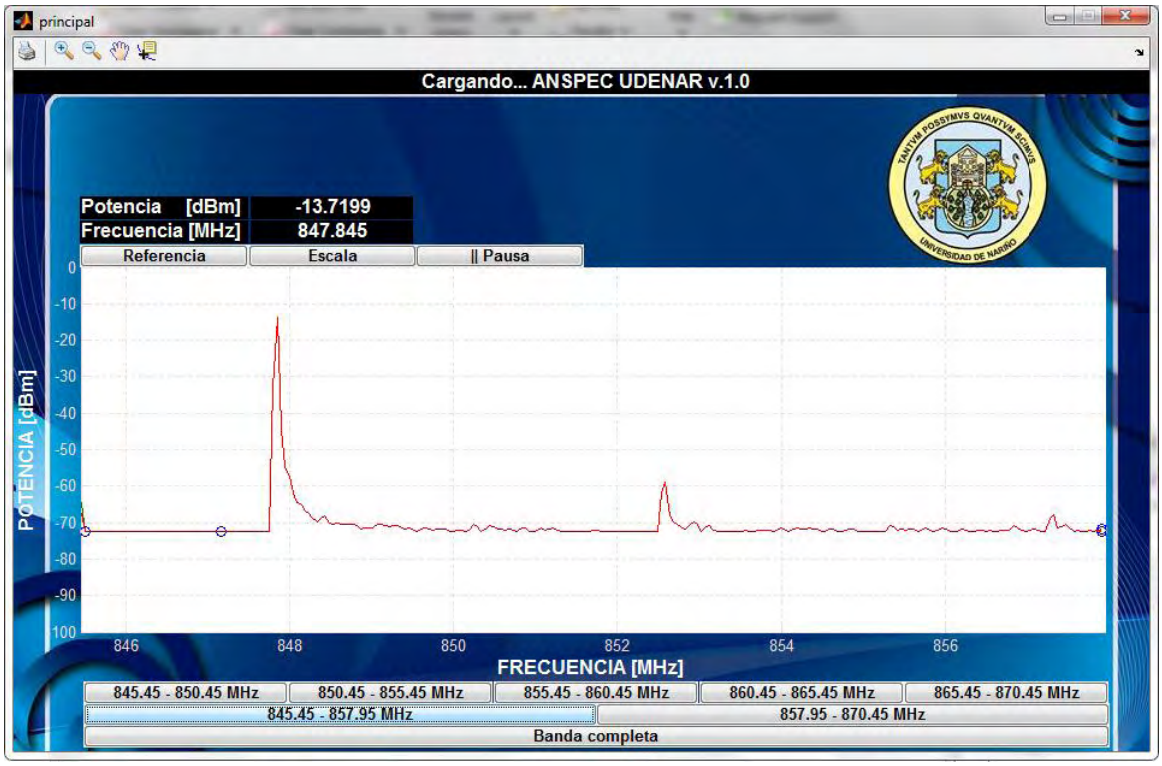
Figura 52. Ventana de configuración inicial de la interfaz de usuario, ingreso del número de puerto y frecuencia de trabajo.



Si los datos ingresados son erróneos el programa emite un mensaje de error que una vez validado devuelve al usuario a la ventana de configuración inicial, si el puerto digitado no es válido un mensaje aparece indicando error y sugiere el puerto disponible. Cuando los parámetros son correctos se muestra la ventana principal donde la gráfica mostrada en pantalla se refresca e indica automáticamente la frecuencia en Mega hercios de donde se encuentran los picos de potencia.

La ventana disponible al usuario consta de cinco botones de ventanas de espectro de 5 MHz, dos botones de 12.5 MHz y uno de banda completa, es decir de 25 MHz, posee una barra de herramientas con las funciones *Zoom*, impresión, herramienta de manipulación *Pan* y un cursor que muestra punto a punto los valores de potencia y frecuencia. Como se puede apreciar en la Figura 53.

Figura 53. Interfaz de usuario, pantalla principal, espectro de una señal en la banda de 0 a 5MHz.



4.2.1 Espectro de frecuencias de señales conocidas

Se realizan pruebas con el equipamiento disponible para observar el espectro de señales conocidas y verificación el funcionamiento del sistema. Se tiene acceso a un generador de señales con la capacidad de producir las formas de onda más conocidas con una salida aceptable. Se presentan las imágenes de algunas de estas pruebas con diferentes formas de onda y frecuencias Figura 54 a Figura 62.

Figura 54 Señal sinusoidal de 2 MHz en tiempo adquirida en el AD9481-PCBZ y visualizada usando MATLAB®. a) 1024 muestras de la señal, b) 128 muestras de la señal.

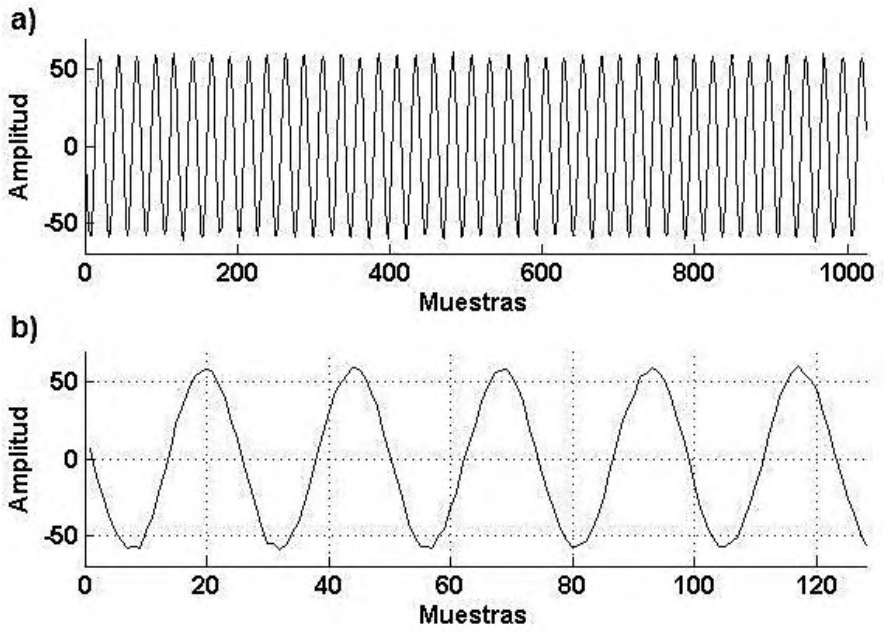


Figura 55. Espectro de frecuencias de una señal sinusoidal de 2 MHz, en la banda de 0 a 12.5 MHz.

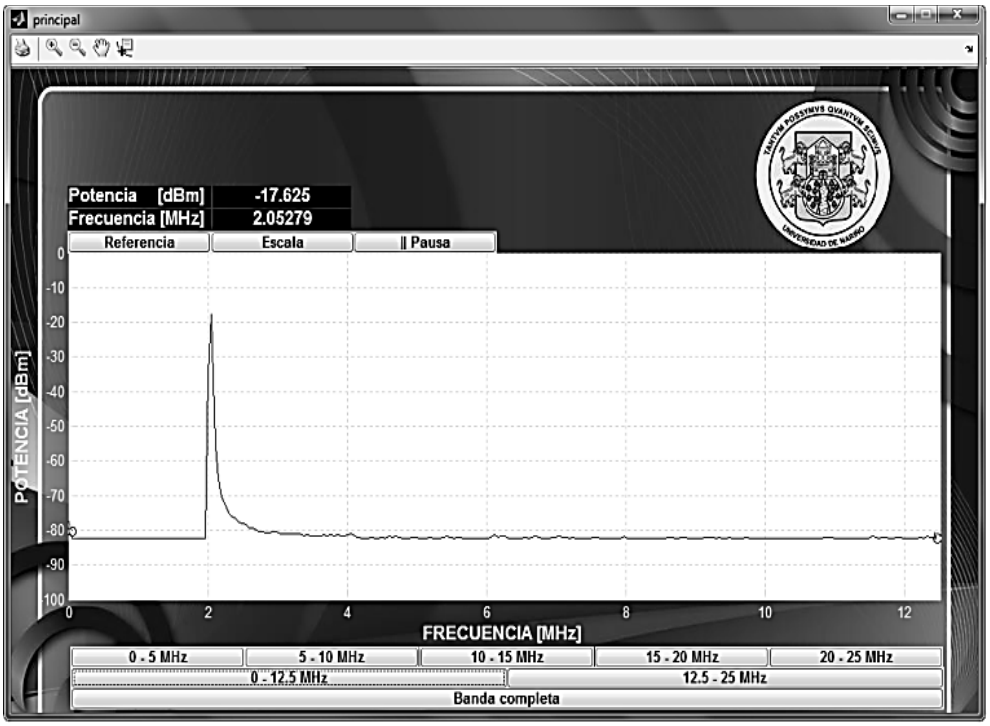


Figura 56. Imagen del espectro de frecuencias de una señal sinusoidal de 2 MHz en el analizador de espectro Anritsu™ MS2723B, en la banda de 0 a 12.5MHz.

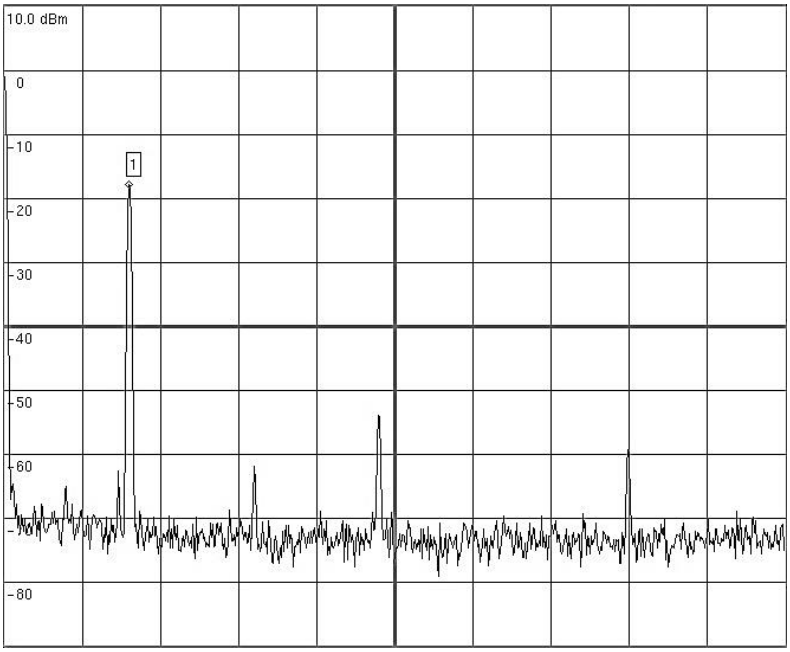


Figura 57. Señal triangular de 1MHz en tiempo adquirida en el AD9481-PCBZ y visualizada usando MATLAB®. a) 1024 muestras de la señal, b) 128 muestras de la señal.

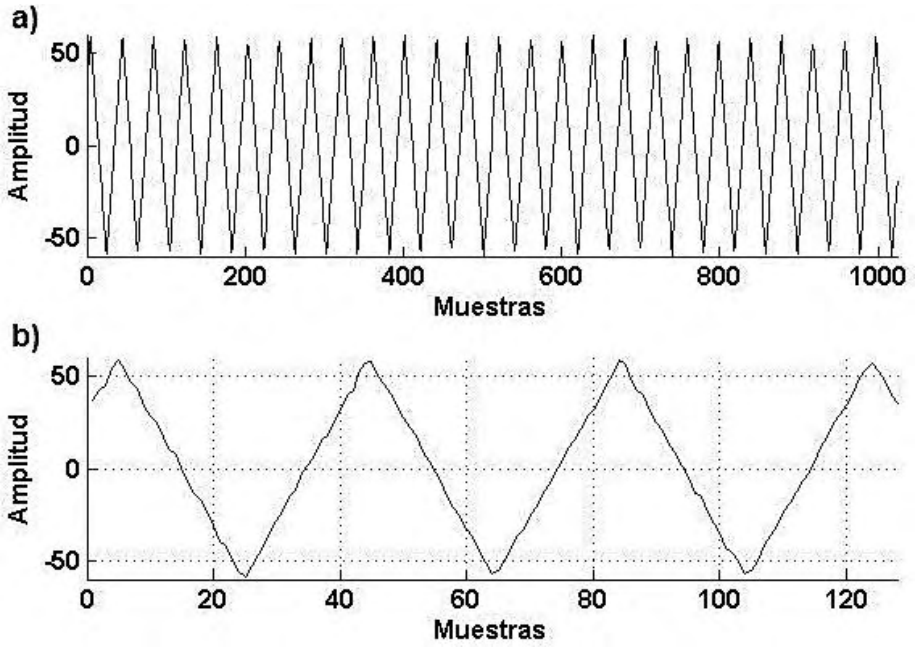


Figura 58. Espectro de frecuencias de una señal triangular de 1 MHz, en la banda de 0 a 12.5 MHz.

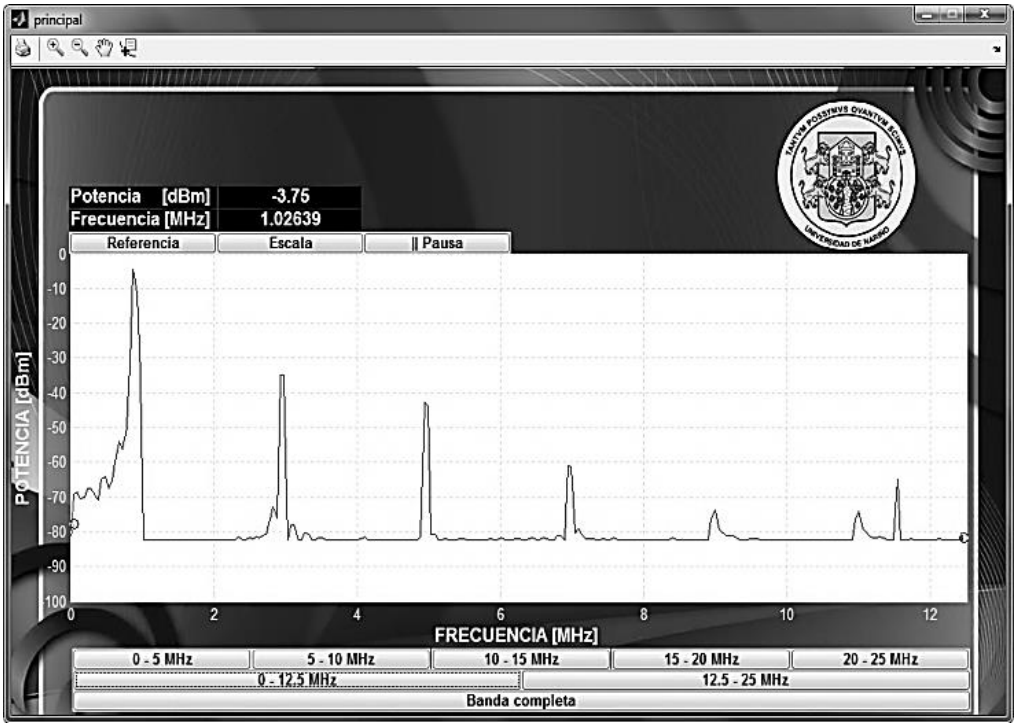


Figura 59. Imagen del espectro de frecuencias de una señal triangular de 1 MHz en el analizador de espectro Anritsu™ MS2723B, en la banda de 0 a 12.5 MHz.

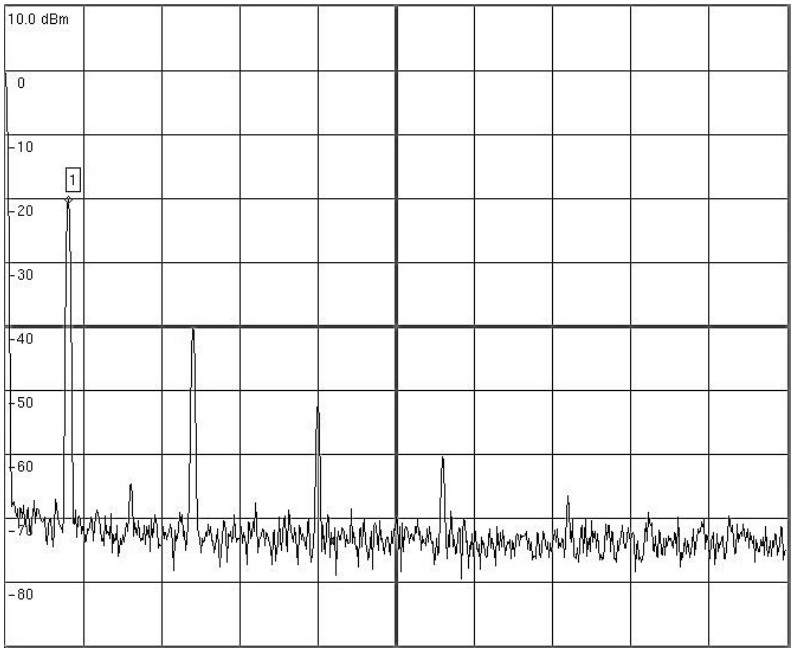


Figura 60. Señal cuadrada de 500 KHz en tiempo adquirida en el AD9481-PCBZ y visualizada usando MATLAB®. a) 1024 muestras de la señal, b) 128 muestras de la señal.

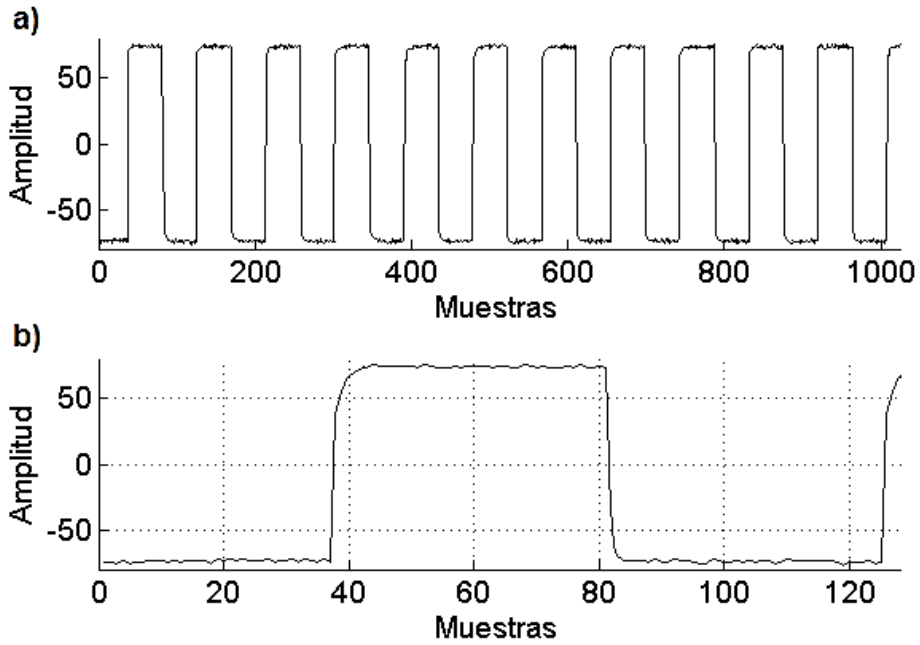


Figura 61. Espectro de frecuencias de una señal cuadrada de 500 KHz, en la banda de 0 a 12.5 MHz.

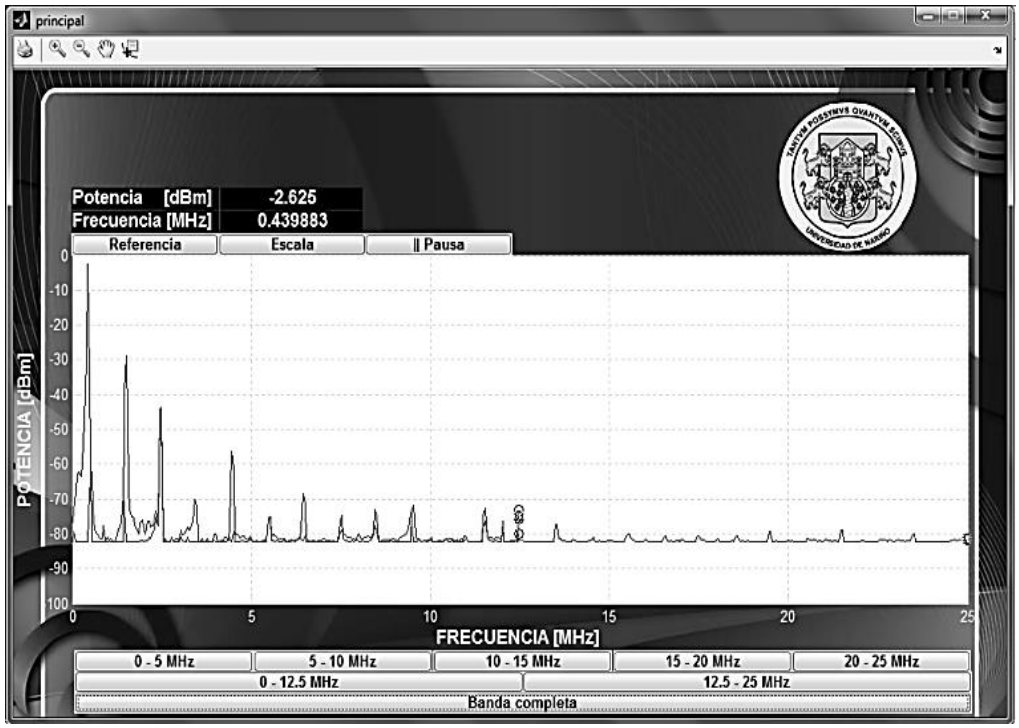
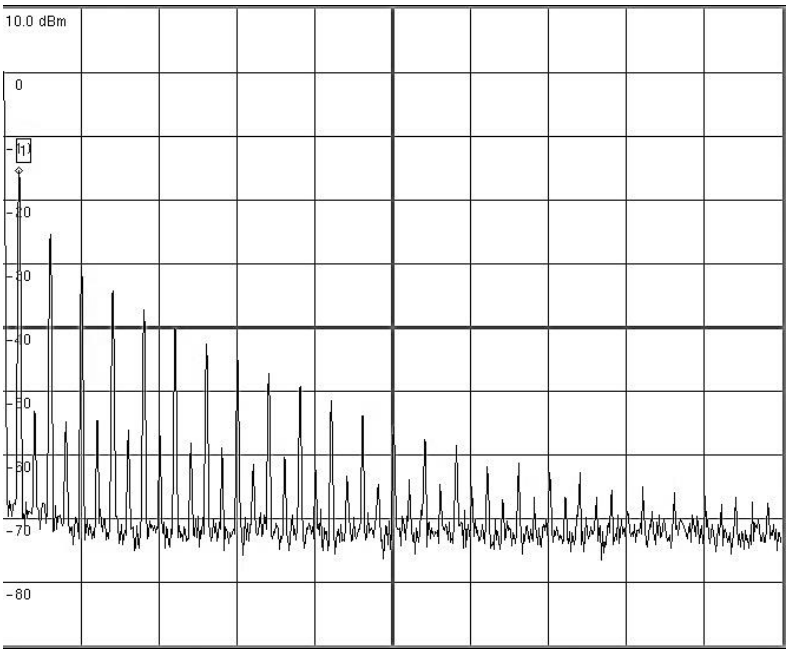


Figura 62. Espectro de frecuencias de una señal cuadrada 500 KHz en el analizador de espectro Anritsu™ MS2723B, en la banda de 0 a 25 MHz.



Como se puede observar los espectros obtenidos están son aproximados a la solución esperada de acuerdo a la teoría de Fourier. Se pueden observar los armónicos propios de cada forma de onda así como el comportamiento de cada uno de ellos considerando las restricciones propias del equipo al que se tiene acceso al realizar las pruebas.

4.2.2 Espectro de frecuencias de señales presentes en el ambiente cuando se captan transmisiones en las bandas VHF y UHF

Se procede a realizar una serie de pruebas en las bandas de interés para el análisis de espectro de alta frecuencia y se presentan los resultados obtenidos con el prototipo de analizador de espectro construido, en comparación a los obtenidos con el analizador de espectro comercial Anritsu™ MS2723B perteneciente al departamento de electrónica de la universidad de Nariño.

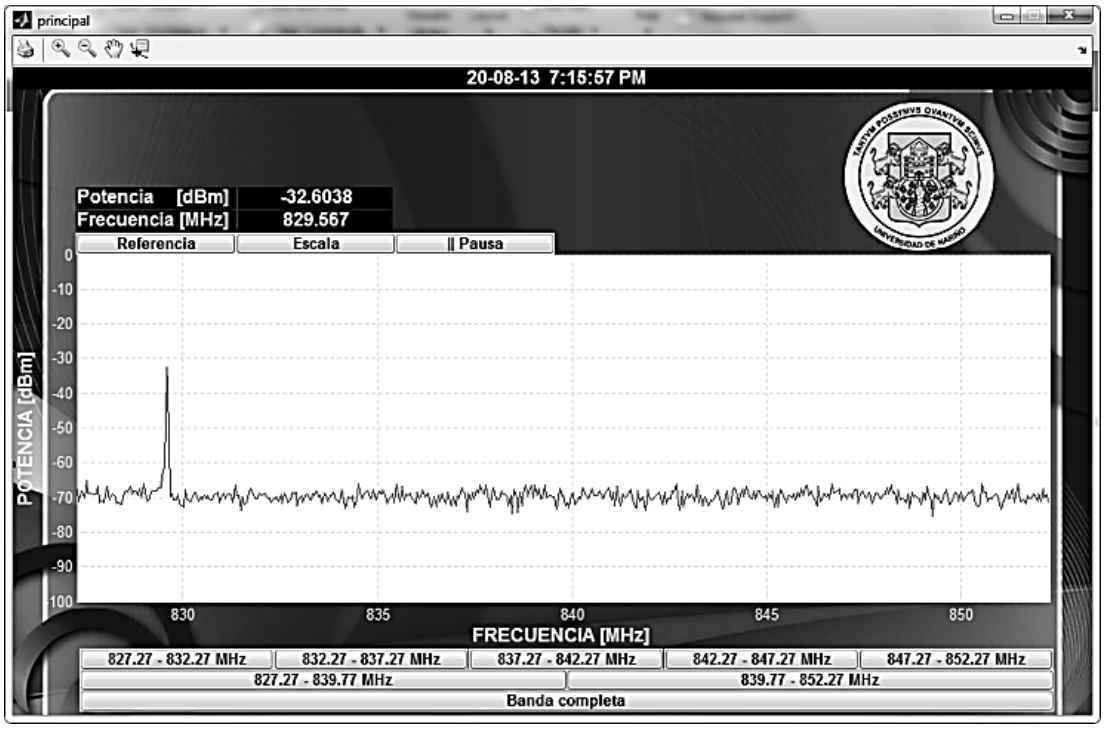
Se utiliza la antena tipo parche de ultra-ancho de banda, construida como parte del prototipo para realizar todas las pruebas. Se consideran muy bajas las pérdidas de potencia inherentes al uso de adaptadores de conexión SMA y similares para cada dispositivo.

Se establece la transmisión de datos haciendo uso del servicio de comunicación celular prestado por tres de los operadores nacionales. Que ofrecen los servicios

GSM (2.5G) y UMTS/HSDPA (3G) en las bandas de operación de 850 y 1900 MHz [48].

Finalmente se capta la transmisión continua de datos que ocurre cuando se transfiere material multimedia en video a través de la red de área local inalámbrica. Figura 63 a Figura 70.

Figura 63. Imagen del espectro de frecuencias obtenido de la comunicación celular a través de una llamada a servicio al cliente del operador Colombia Telecomunicaciones, Movistar (*Virgin Mobile*), en la banda GSM (850 MHz). UMTS/HSDPA (850 MHz). *Low channels* Prototipo construido



.

Figura 64. Imagen del espectro de frecuencias obtenido de la comunicación celular a través de una llamada a servicio al cliente del operador Colombia Telecomunicaciones, Movistar (*Virgin Mobile*). En la banda GSM (850 MHz). UMTS/HSDPA (850 MHz). *Low channels* Anritsu™ MS2723B.

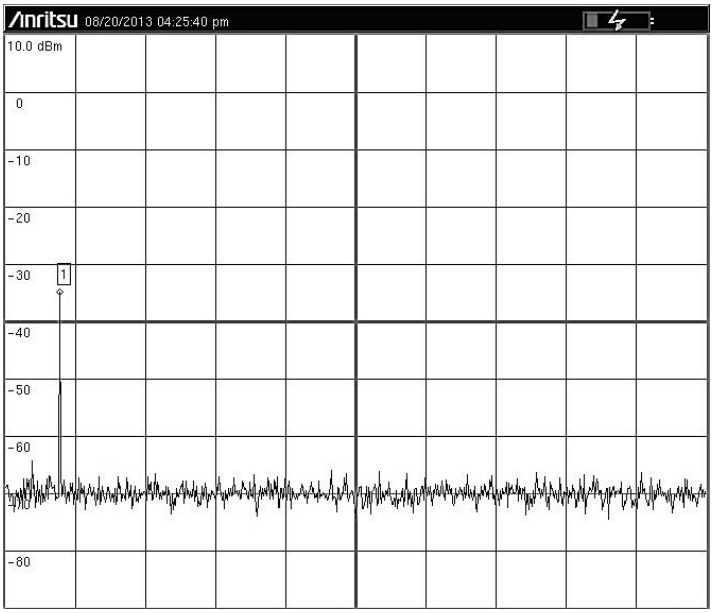


Figura 65. Imagen del espectro de frecuencias obtenido de la comunicación celular a través de una llamada a servicio al cliente del operador Claro™ (COMCEL S.A). En la banda GSM (850 MHz). UMTS/HSDPA (850 MHz). *High channels* Prototipo construido.

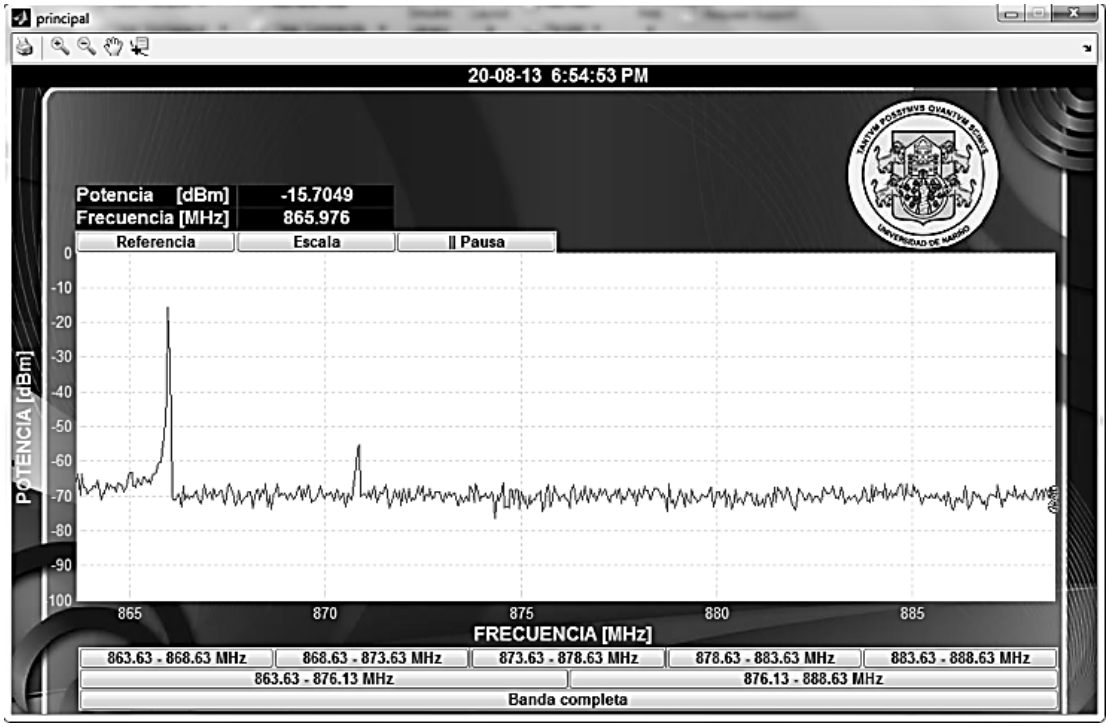


Figura 66. Imagen del espectro de frecuencias obtenido de la comunicación celular a través de una llamada a servicio al cliente del operador Claro™ (COMCEL S.A). En la banda GSM (850 MHz). UMTS/HSDPA (850 MHz). *High channels* Anritsu™ MS2723B.

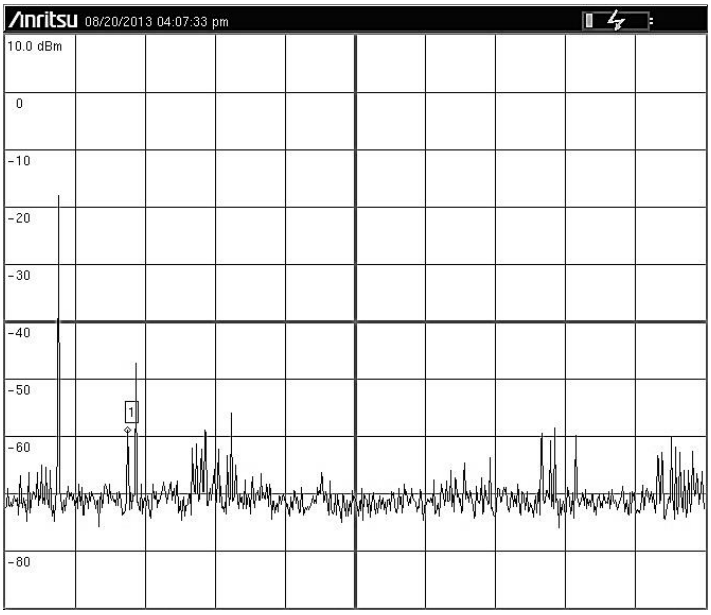


Figura 67. Imagen del espectro de frecuencias obtenido de la comunicación celular a través de una llamada a servicio al cliente del operador Tigo® (Uff, UNE y ETB). En la banda GSM (1900 MHz). UMTS/HSDPA (1900 MHz). Prototipo construido.

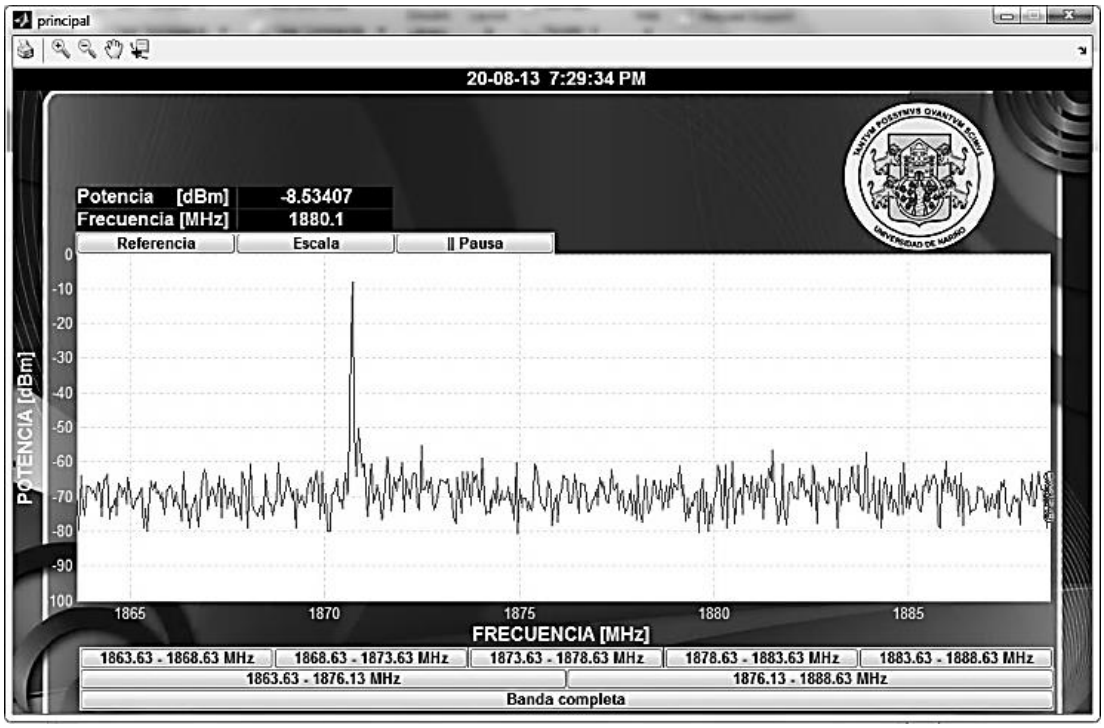


Figura 68. Imagen del espectro de frecuencias obtenido de la comunicación celular a través de una llamada a servicio al cliente del operador Tigo® (Uff, UNE y ETB). En la banda GSM (1900 MHz). UMTS/HSDPA (1900 MHz). Anritsu™ MS2723B.

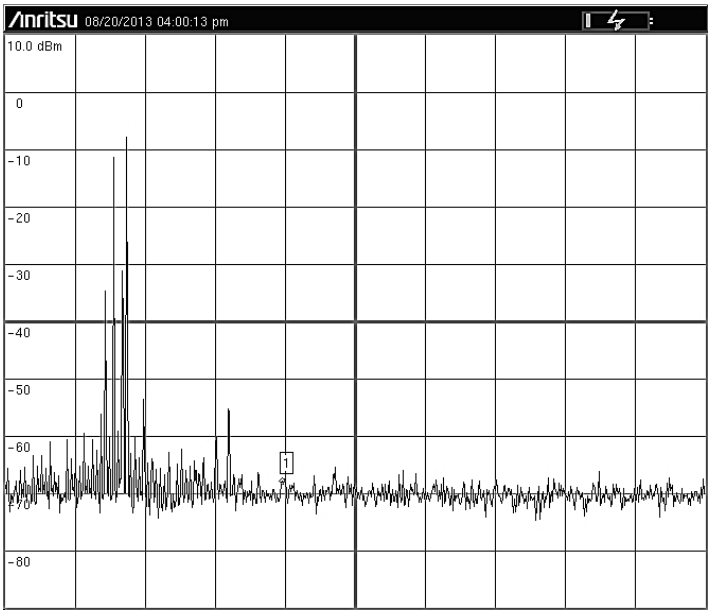


Figura 69. Imagen del espectro de frecuencias obtenido a través de la transferencia continua de material multimedia en video a través de la red de área local inalámbrica. (WiFi®). En la banda ISM (2400 MHz). Prototipo construido.

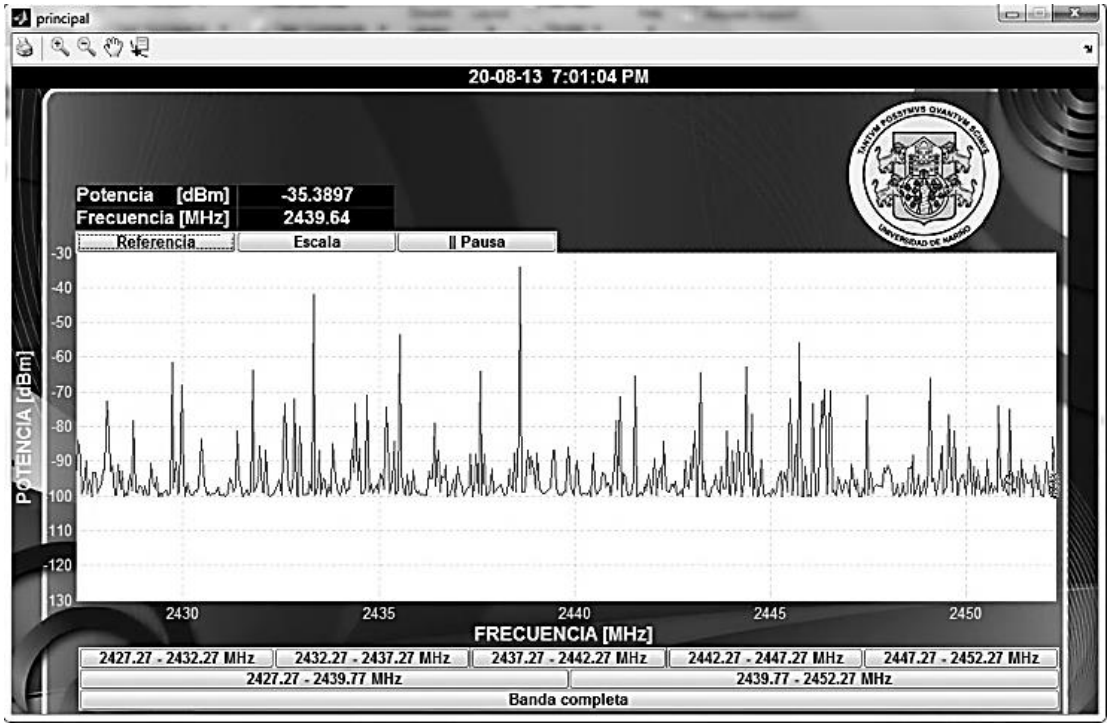
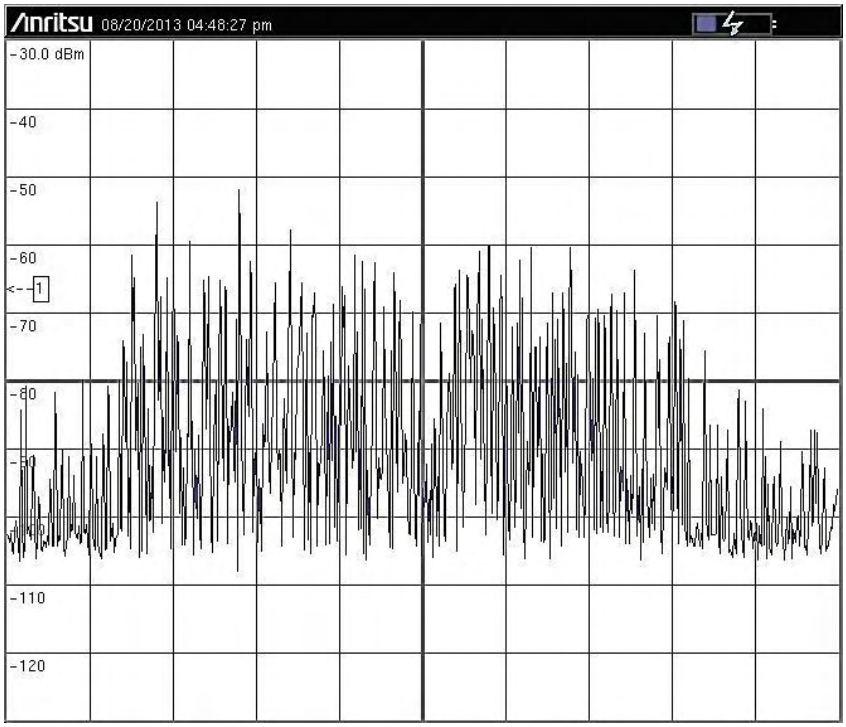


Figura 70. Imagen del espectro de frecuencias obtenido a través de la transferencia continua de material multimedia en video a través de la red de área local inalámbrica. (WiFi®). En la banda ISM (2400 MHz). Anritsu™ MS2723B.



4.3 CARACTERÍSTICAS DEL PROTOTIPO DE ANALIZADOR DE ESPECTRO IMPLEMENTADO

En este apartado de resultados se presenta un resumen de características técnicas y parámetros de fucionamiento del prototipo de analizador de espectro implenmentado.

4.3.1 Descripción

- Cobertura: 100 KHz - 2.725 GHz.
- Preamplificador de banda ancha en torno a 20 dBs
- Un modo de barrido, con actualización de 40 ms.
- Resolución de ancho de banda fijo: 5, 12.5 y 25 MHz.
- Interfaz gráfica de usuario que incluye botones de cambio de resolución de ancho de banda fijo, referencia en dBm, escala en dB/div, botón actualizar, pausa, barra de herramientas de manipulación, área de trazado con ratio de aspecto panorámico 16:9, barra de información y reloj del sistema, presentación en cuadros de texto de la información de la frecuencia fundamental y su potencia en cada barrido.
- Visualización en alto contraste con fondo blanco para facilitar la lectura de datos.
- Fuente de alimentación: simple DC, 3.0 – 3.6 V.

- Rango dinamico de entrada: 50 mW.
- Máxima corriente de entrada: 40 mA.
- Temperatura de operación: -40 - +85°C.
- Interfaz de conexión análoga: SMA de bajo ruido.
- Interfaz de conexión digital: bajo el estándar 40-pin header, 2.54 mm y altura de 11.5 mm.

CONCLUSIONES

Se obtine como producto final un sistema modular y escalable analizador de espectro de conexión SMA con cobertura desde 100 KHz hasta 2.7 GHz, barrido de 40 ms, resolución de ancho de banda fijo de 25 MHz, interfaz gráfica de usuario que incluye área de trazado con ratio de aspecto panorámico.

Los temas de investigación involucrados en el diseño y construcción del prototipo implementado son diversos puesto que abarcan las áreas de telecomunicaciones, instrumentación electrónica, sistemas digitales reconfigurables y lenguajes de programación. Que brinda la oportunidad de plantear diferentes proyectos de investigación que permiten el aprendizaje interdiciplinario en las diferentes ramas de la ingeniería electrónica.

El manejo de señales en radiofrecuencia es un campo casi inexplorado en la universidad de Nariño, en parte por la carencia de dispositivos de medición adecuados. Proyectos de investigación inherentes al tema pueden ser abordados a partir de los módulos del prototipo construido, para el estudio de fenómenos propios del campo de las comunicaciones, la actualización del sistema y la implementación de nuevos prototipos modulares y escalables.

El diseño y construcción de equipos a partir de tecnologías de desarrollo de propósito general tales como las FPGA conlleva en la mayoría de las situaciones una disminución considerable de costos de implementación, que no implica una disminución en la calidad de los equipos construidos, sino una mejor relación de costo beneficio.

Las FPGAs son una solución adecuada cuando se trata de implementar algoritmos que exigen gran velocidad de procesamiento, dependiendo de la gama de estos dispositivos se puede conseguir alta precisión y exactitud en los cálculos; no obstante el dispositivo usado en este trabajo es de bajo costo y recomendado para aplicaciones estudiantiles, sin embargo es capaz de realizar algoritmos de alta exigencia como la FFT de 1024 puntos de una señal digital y transmitirla a un PC a muy alta velocidad con una resolución baja pero aceptable.

Las herramientas de visualización en software provistas por MathWorks son una solución robusta para la programación de aplicaciones en el entorno Microsoft Windows, puesto que presenta características de utilidad para el proyecto tales como un lenguaje de programación de alto nivel con gran soporte y funciones matemáticas preestablecidas, un paquete para aplicaciones visuales en tiempo real y un compilador que permite realizar software Standalone en empaquetado ejecutable.

La tecnología Microstrip en combinación con el método DBR es una de las mejores alternativas para la implantación de filtros microondas, porque reduce sustancialmente el tamaño físico del dispositivo, existen soluciones flexibles en

software especializado para el diseño y simulación, su fabricación es sencilla y demanda costos relativamente bajos. A su vez Microstrip es igualmente buena para la construcción de antenas tipo parche en la banda de utilidad para el proyecto, presenta ventajas similares a las expuestas para la construcción de filtros.

El desarrollo de proyectos bajo el concepto modular con un buen sistema de acople posibilita el reúso, mejoramiento del sistema, reconfiguración de módulos y el uso de parte del proyecto en la construcción de dispositivos para futuras aplicaciones. Los acoples entre módulos son importantes ya que evitan perdida de información, potencia y distorsión por interferencia.

RECOMENDACIONES Y TRABAJOS FUTUROS

El sistema se puede mejorar utilizando hardware de mayores prestaciones, específicamente un ADC de mayor resolución y velocidad similar a la utilizada, así mismo una FPGA de gama media o alta independiente de la tecnología.

Para una mayor versatilidad se puede utilizar tecnologías de trasmisión de video tales como VGA o HDMI con el fin de implementar una pantalla embebida en el sistema. Esto a su vez implica el diseño y programación de un sistema operativo simple que interactúe con el usuario para el manejo directo del hardware implicado.

La característica de portabilidad del sistema requiere del diseño y construcción de un módulo de alimentación recargable con la suficiente autonomía para el funcionamiento adecuado de todos los módulos del sistema tales como bloque de procesamiento digital y periféricos como pantalla y sistema de interacción con el usuario en pruebas de campo.

Teniendo en cuenta que el prototipo es un dispositivo de análisis microondas de ultra ancho de banda es recomendable diseñar y construir un juego de antenas con buena ganancia, directividad omnidireccional y manejo de bandas de frecuencia que cubran el rango de frecuencia de trabajo.

Para una mayor exactitud en las medidas se recomienda calibrar el equipo haciendo uso de recursos tecnológicos de alta precisión y evitando interferencia en una cámara anecoica.

Se recomienda usar una etapa de amplificación de ultra-ancho de banda después de la antena porque esto permite la recepción de señales de muy baja potencia lo cual es muy útil para realizar experimentos de campo, ampliando el rango de aplicaciones del prototipo y permitiendo una visualización más detallada del espectro de frecuencias.

REFERENCIAS

[1] MUNGUÍA VALIENTE, Héctor Leonel. Implementación de un analizador de espectro para frecuencias de audio utilizando un procesador digital de señales (DSP), Facultad de Ingeniería, Escuela Mecánica Eléctrica, Universidad de San Carlos de Guatemala, julio de 2006. Documento en línea disponible en: http://biblioteca.usac.edu.gt/tesis/08/08_0171_EO.pdf

[2] TORO CADAVID, Juan Camilo. Diseño De un analizador de espectro aplicado a redes de voltaje monofásicas de 120v a frecuencia industrial, Universidad De San Buenaventura Medellín (USB), 2009. Documento en línea disponible en: http://bibliotecadigital.usbcali.edu.co/jspui/bitstream/10819/335/1/Dise%C3%B1o_Analizador_Espectro_Toro_2009.pdf

[3] GARFIAS, Alejandro S, Diseño y construcción de un analizador de espectros usando una plataforma basada en FPGA, México, D.F. Agosto de 2008. Documento en línea disponible en: http://tesis.ipn.mx/dspace/bitstream/123456789/456/1/Tesis_Analizador_Espectros.pdf

[4] MILLER Adam Robert, Development and verification of parameterized digital signal processing macros for microelectronic systems. Thesis presented for the Master of Science, universidad of Tennessee, Knoxville. Agosto 2003

[5] National Instruments Corporation. 2.7 GHz RF Vector Signal Analyzer With Digital DownConversion, 2006. Documento en línea disponible en: http://www.ni.com/pdf/products/us/cat_vectorsignalanalyzer.pdf

[6] Anritsu™, Spectrum Master MS2723B, disponible en línea en: http://downloadfile.anritsu.com/Files/en-US/Technical-Notes/Technical-Note/SpectrumAnalyzer_EE1300.pdf

[7] PÉREZ VEGA, Constantino. SAINZ DE LA MAZA, José Zamanillo. Manuales de funcionamiento de la instrumentación del Laboratorio de Radiocomunicaciones y Televisión, Universidad de Cantábria, 2002. Documento en línea disponible en: http://personales.unican.es/perezvr/pdf/Manuales_lock.pdf

[8] Altera Corporation, Cyclone® II 2C35 FPGA Version 1.4 Copyright © 2006 manual de usuario disponible en: ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf

[9] Altera Corporation, Cyclone® II Device Handbook, Volume 1.Copyright © 2008. All rights reserved. Documento en linea disponible en: http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf

[10] Analog Devices Inc. 8-Bit 250 MSPS AD9481 PCBZ, 2004, hoja de datos. Documento en línea disponible en: http://www.analog.com/static/imported-files/data_sheets/AD9481.pdf

[11] Abracon® Corporation. Oscillator, OCXO, AOCJY1 100MHz, Hoja de datos disponible en: http://www.abracon.com/Precisiontiming/AOCJY1.pdf
[12] POZAR David M, Microwave Engineering. Second Edition, University Massachusetts at Amherst. John Wiley & Sons, Inc. ISBN 0-471-1704b-8
[13] VITERI Carlos Andrés, BOHÓRQUEZ Juan Carlos, Design and Simulation of a Multilayer Dual Behavior, Resonator Microwave Filter. Comunicaciones (LATINCOM), Conferencia Latinoamericana IEEE en Bogotá, Colombia 2010
[14] Constantine A. Balanis, Antenna Theory Analysis and Desing, second edition, chapter 1.
[15] WINTER, Christian Peter. CONTRERAS MUÑOZ, Francisco. CAMPS PASCUAL, Oscar. Diseño y fabricación de un filtro pasa banda pasivo con líneas acopladas, Universidad de las Islas Baleares, 2009/2010. Documento en línea disponible en: http://ibdigital.uib.es/greenstone/collect/enginy/import/Enginy_2010/Enginy_2010v02p013.pdf
[16] ROGERS® Corporation, RT/duroid® 6202 High Frequency Laminates, Printed in U.S.A, All rights reserved 2013, hoja de datos disponible en linea en: http://www.rogerscorp.com/documents/610/acm/RT-duroid-6202-laminate-data-sheet.pdf
[17] ROGERS® Corporation, RT/duroid® 5870/5880 High Frequency Laminates, Printed in U.S.A., All rights reserved 2013. Hoja de datos disponible en linea en: http://www.rogerscorp.com/documents/606/acm/RT-duroid-5870-5880Data-Sheet.aspx
[18] MATTHAEI G, YOUNG L, JONES E, Microwave Filters, Impedance Matching Networks, and Coupling Structures. Artech House, 1980
[19] Calculadora de Línea Microstrip, disponible en línea en: http://www1.sphere.ne.jp/i-lab/ilab/ tool/ms_line_e .htm
[20] Agilent® Technologies, Advanced Design System, Versión estudiantil, disponible en línea en: http://www.home.agilent.com/agilent/software.jspx?ckey=2212036&lc=eng&cc=CO&nid=-34346.0&id=2212036&pageMode=CV
[21] MathWorks®, MATLAB, disponible en línea en: http://www.mathworks.com/products/matlab/
[22] COOLEY, James W, TUKEY, John W. An algorithm for the machine calculation of complex Fourier series, 1965.
[23] GOOD, I. J.The interaction algorithm and practical Fourier analysis, 1958.
[24] BRUUN Georg, z-Transform DFT filters and FFTs, IEEE Trans. on Acoustics, Speech and Signal Processing, 1978.

[25] RADER C. M, Discrete Fourier transforms when the number of data samples is prime, 1968.
[26] BLUESTEIN, Leo I, A linear filtering approach to the computation of the discrete Fourier transform, Northeast Electronics Research and Engineering Meeting Record 10, pág 218-219 1968.
[27] Mini-Circuits®, Frecuency Mixer SYM-63LH+, hoja de datos. Documento en línea disponible en: http://www.minicircuits.com/pdfs/SYM-63LH+.pdfBOYLESTAD Robert L, NASHELSKY, Teoría de Circuitos y Dispositivos Electrónicos, Octava Edición pag 837-846-847- Prentice Hall
[28] Mini-Circuits®, Evaluation board and circuit for pin connections. Disponible en línea en: http://www.minicircuits.com/pcb/WTB-12_P02.pdf
[29] Mini-Circuits®, Plug-in Low pass filter PLP 70+, hoja de datos. Documento en línea disponible en: http://www.minicircuits.com/pdfs/PLP-70+.pdf
[30] Analog Devices Inc. Low distortion differential RF/IF amplifier AD 8351. Documento en línea disponible en: http://www.analog.com/static/imported-files/data_sheets/AD8351.pdf
[31] ALTERA® Corporation, QUARTUS® II Design Software 2013. Documento en línea disponible en: http://www.altera.com/literature/br/br-quartus2-software.pdf
[32] OpenCores.org. Radix-4 complex FFT, Septiembre 30, 2010. Proyecto de código abierto disponible en línea en: http://opencores.org/project,cfft
[33] FERNÁNDEZ R, Alfredo José, Algoritmos CORDIC, Documento en linea disponible en: http://delep.uah.es/antigua/PLANES%20ANTIGUOS/I.ECA/2/Dise%F1o%20de%20Ctos%20y%20Sist%20Ecos/Apuntes/Cordic.pdf
[34] CHU Pong P, FPGA Prototyping by Verilog examples, XILINX™ Spartan® -3 version, Cleveland State University. John Wiley & Sons, INC. Publication. 11 de Julio de 2008. ISBN: 9780470185322
[35] FLOYD Thomas L, Fundamentos de sistemas digitales, séptima edición. Prentice hall, Pag. 478 – 496. ISBN: 9788483220856
[36] GARG R, BHARTIA P, BAHL I, ITTIPIBOON A, Microstrip Antenna Design Handbook, Artech House. 2001
[37] KUMAR Girish, RAY K. P, Broadband Microstrip Antennas, Artech House. Octubre 30, 2002. ISBN-13: 978-1580532440
[38] MEYER-BAESE Uwe, Digital signal processing with field programmable gate arrays, Third Edition, Springer 2007 ISBN 978-3-540-72613-5
[39] SAYRE Cotter W, Complete Wireless Desing, Mc Graw Hill, Junio 2008, 693 pag, capítulo 7
[40] HUNTER Matthew T, KOURTELLIS Achilleas G, ZIOMEK Christopher D and MIKHAEL Wasfy B, Fundamentals of modern spectral analysis, 2010

[41] ALCOCER Jesús Rufino, Aproximación y Síntesis De Filtros Elípticos, Universidad de las Américas Puebla, Tesis profesional para obtener el título Licenciatura en Ingeniería en Electrónica y Comunicaciones, 2005. Documento disponible en línea en:
http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/rufino_a_j/capitulo2.pdf

[42] Departamento de Teoría de la Señal y Comunicaciones. Análisis y Síntesis de Circuitos: Filtros. Universidad de Alcalá. Madrid. Documento en línea disponible en:
http://agamenon.tsc.uah.es/Asignaturas/ittse/asc/apuntes/TEMA1.pdf

[43] MathWorks® Documentation center. Permute data into bit-reversed order, Documento en linea disponible en:
http://www.mathworks.com/help/signal/ref/bitrevorder.html

[44] MathWorks®. MATLAB® Sofware. The Language of Technical Computing © 2012. Documento disponible en línea en:
http://www.mathworks.com/products/datasheets/pdf/matlab.pdf

[45] National Instruments Corporation. Perform RF streaming successfully, 2009. Documento en línea disponible en:
http://zone.ni.com/devzone/cda/tut/p/id/7209

[46] PÉREZ VEGA, Constantino. SAINZ DE LA MAZA, José Zamanillo. Manuales de funcionamiento de la instrumentación del Laboratorio de Radiocomunicaciones y Televisión, Universidad de Cantábria, 2002. Documento en línea disponible en:
http://personales.unican.es/perezvr/pdf/Manuales_lock.pdf

[47] University of Rhode Island Department of Electrical and Computer Engineering, FFT Tutorial. Documento en línea disponible en:
http://www.ele.uri.edu/~hansenj/projects/ele436/fft.pd f

[48] QUIROZ C Gabriela, Laboratorio Analógico, Universidad de las Américas Puebla, Tesis profesional para obtener el título Licenciatura en Ingeniería en Electrónica y Comunicaciones, 2007. Documento disponible en línea en:
http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/quiroz_c_g/capitulo2.pdfB
ALANIS Constantine A, Antenna Theory Analysis and Desing, Segunda Ed. Chapter 1. ISBN 0471592684

ANEXOS

Anexo 1 código fuente en lenguaje VHDL

```
--------------------------------------------------------------------------
-- SISTEMA ANALIZADOR DE ESPECTRO UDENAR v1.0                           --
--------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.NUMERIC_STD.ALL;
--------------------------------------------------------------------------
ENTITY ANSPEC_UDENAR IS
PORT (
      reloj         : IN  STD_LOGIC;
      din           : IN  STD_LOGIC;
      adc_out       : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      dout          : OUT STD_LOGIC;
      led           : OUT STD_LOGIC);
END ENTITY ANSPEC_UDENAR;
--------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF ANSPEC_UDENAR IS
SIGNAL
comando           : STD_LOGIC;
SIGNAL
fft_dir           : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL
fft_ocu           : STD_LOGIC;
SIGNAL
ram_dir           : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL
ram_datos         : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL
ram_out0          : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL
ram_out1          : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL
ram_esc_act       : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL
datos_reales      : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL
start_fft         : STD_LOGIC;
SIGNAL
start_uart        : STD_LOGIC;
SIGNAL
uart_ok           : STD_LOGIC;
SIGNAL
fft_lista         : STD_LOGIC;
CONSTANT
datos_imaginarios : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
CONSTANT
tierra            : STD_LOGIC                    := '0';

COMPONENT CONTROLADOR_SUBSISTEMA
```

```
PORT (
      reloj         : IN  STD_LOGIC;
      fft_ocu       : IN  STD_LOGIC;
      uart_ok       : IN  STD_LOGIC;
      comando       : IN  STD_LOGIC;
      fft_ok        : IN  STD_LOGIC;
      fft_dir       : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      fft_datos     : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      start_fft     : OUT STD_LOGIC;
      start_uart    : OUT STD_LOGIC;
      ram_esc_act   : OUT STD_LOGIC;
      ram_dir       : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
      ram_datos     : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
      led           : OUT STD_LOGIC);
END COMPONENT;

COMPONENT FFT_SUBSISTEMA
PORT (
      reloj        : IN  STD_LOGIC;
      reset        : IN  STD_LOGIC;
      start        : IN  STD_LOGIC;
      inversa      : IN  STD_LOGIC;
      puerto_real  : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      puerto_imag  : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      entrada_ocu  : OUT STD_LOGIC;
      salida_act   : OUT STD_LOGIC;
      salida_real  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
      salida_imag  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
      posicion_out : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END COMPONENT;

COMPONENT MEM_RAM_SUBSISTEMA
PORT (
      reloj        : IN  STD_LOGIC;
      esc_act      : IN  STD_LOGIC;
      dir          : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      din          : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      dout         : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END COMPONENT;

COMPONENT SENO_MEM_ROM_SUBSISTEMA
PORT (
      reloj        : IN  STD_LOGIC;
      dout         : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT; -- Componente opcional de prueba y validación.

COMPONENT UART_SUBSISTEMA
PORT (
      reloj        : IN  STD_LOGIC;
      reset        : IN  STD_LOGIC;
      serial_in    : IN  STD_LOGIC;
      start_tx     : IN  STD_LOGIC;
      datos_tx     : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      rx_ok        : OUT STD_LOGIC;
```

```
      tx_ok         : OUT STD_LOGIC;
      serial_out    : OUT STD_LOGIC;
      datos_rx      : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;
BEGIN

SUBSISTEMA_DE_CONTROL : CONTROLADOR_SUBSISTEMA
PORT MAP (
      reloj                    => reloj,
      comando                  => comando,
      fft_dir(9 DOWNTO 0)      => fft_dir(9 DOWNTO 0),
      fft_ocu                  => fft_ocu,
      fft_datos(9 DOWNTO 0)    => datos_reales(9 DOWNTO 0),
      fft_ok                   => fft_lista,
      uart_ok                  => uart_ok,
      led                      => led,
      ram_dir(9 DOWNTO 0)      => ram_dir(9 DOWNTO 0),
      ram_datos(9 DOWNTO 0)    => ram_datos(9 DOWNTO 0),
      ram_esc_act              => ram_esc_act(0),
      start_fft                => start_fft,
      start_uart               => start_uart);

SUBSISTEMA_DE_CALCULO_DE_FFT : FFT_SUBSISTEMA
PORT MAP (
      reloj                    => reloj,
      puerto_real(7 DOWNTO 0)  => adc_out(7 DOWNTO 0),
      inversa                  => tierra,
      puerto_imag(7 DOWNTO 0)  => datos_imaginarios(7 DOWNTO 0),
      reset                    => tierra,
      start                    => start_fft,
      entrada_ocu              => fft_ocu,
      salida_real(9 DOWNTO 0)  => datos_reales(9 DOWNTO 0),
      salida_act               => fft_lista,
      posicion_out(9 DOWNTO 0) => fft_dir(9 DOWNTO 0),
      salida_imag              => OPEN);

SUBSISTEMA_DE_MEMORIA_RAM : MEM_RAM_SUBSISTEMA
PORT MAP (
      dir(9 DOWNTO 0)          => ram_dir(9 DOWNTO 0),
      reloj                    => reloj,
      din(9 DOWNTO 0)          => ram_datos(9 DOWNTO 0),
      esc_act                  => ram_esc_act(0),
      dout(9 DOWNTO 0)         => ram_out0(9 DOWNTO 0));

SUBSISTEMA_DE_COMUNICACION : UART_SUBSISTEMA
PORT MAP (
      reloj                    => reloj,
      reset                    => tierra,
      serial_in                => din,
      start_tx                 => start_uart,
      datos_tx(9 DOWNTO 0)     => ram_out1(9 DOWNTO 0),
      datos_rx                 => OPEN,
      rx_ok                    => comando,
      serial_out               => dout,
```

```
      tx_ok                    => uart_ok);

ram_out1 <= ram_out0 WHEN ram_out0(9) = '0' ELSE (OTHERS => '0');
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- NUCLEO DE LA FFT, ALGORITMO RADIX-4                                  --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_SIGNED.ALL;
-------------------------------------------------------------------------
ENTITY CFFT4 IS
GENERIC (ancho : NATURAL);
PORT (
      reloj       : IN  STD_LOGIC;
      reset       : IN  STD_LOGIC;
      start       : IN  STD_LOGIC;
      inversa     : IN  STD_LOGIC;
      imagx       : IN  STD_LOGIC_VECTOR(ancho-1 DOWNTO 0);
      realx       : IN  STD_LOGIC_VECTOR(ancho-1 DOWNTO 0);
      salida_real : OUT STD_LOGIC_VECTOR(ancho+1 DOWNTO 0);
      salida_imag : OUT STD_LOGIC_VECTOR(ancho+1 DOWNTO 0));
END ENTITY CFFT4;
-------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF CFFT4 IS
TYPE
REG_TIPO_A IS ARRAY (  3 DOWNTO 0) OF
STD_LOGIC_VECTOR(ancho-1 DOWNTO 0);
TYPE
REG_TIPO_B IS ARRAY (  3 DOWNTO 0) OF
STD_LOGIC_VECTOR(  ancho DOWNTO 0);
SIGNAL
contador       : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";
SIGNAL
regaim         : REG_TIPO_A;
SIGNAL
regare         : REG_TIPO_A;
SIGNAL
regbim         : REG_TIPO_B;
SIGNAL
regbre         : REG_TIPO_B;
BEGIN

CONTEO : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
contador     <= "00";
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF start = '1' THEN
      contador <= "00";
      ELSE
      contador <= contador + 1;
```

```
      END IF;
END IF;
END PROCESS CONTEO;
-------------------------------------------------------------------------
-- 0 rA(0) <= A0    rB(1) <= rA(0)-rA(2)    rB(2) <= rA(1)+rA(3)   --
--   B3 <= rB(1)-rB(3)                                             --
-------------------------------------------------------------------------
-- 1 rA(1) <= A1    rB(3) <= (-j)*(rA(1)-rA(3))                   --
--   B0 <= rB(0)+rB(2)                                             --
-------------------------------------------------------------------------
-- 2 rA(2) <= A2                                                  --
--   B1 <= rB(1)+rB(3)                                             --
-------------------------------------------------------------------------
-- 3 rA(3) <= A3    rB(0) <= rA(0)+rA(2)                          --
--   B2 <= rB(0)-rB(2)                                             --
-------------------------------------------------------------------------
CALC_RADIX_4 : PROCESS(reloj)
BEGIN
IF reloj'EVENT AND reloj = '1' THEN
CASE contador IS
-------------------------------------------------------------------------
-- 0 rA(0) <= A0    rB(1) <= rA(0)-rA(2)    rB(2) <= rA(1)+rA(3)   --
--   B3 <= rB(1)-rB(3)                                             --
-------------------------------------------------------------------------
WHEN "00" =>
regaim(0)    <= imagx;
regare(0)    <= realx;
regbim(1)    <= SXT(regaim(0), ancho+1) - SXT(regaim(2), ancho+1);
regbre(1)    <= SXT(regare(0), ancho+1) - SXT(regare(2), ancho+1);
regbim(2)    <= SXT(regaim(1), ancho+1) + SXT(regaim(3), ancho+1);
regbre(2)    <= SXT(regare(1), ancho+1) + SXT(regare(3), ancho+1);
salida_real  <= SXT(regbim(1), ancho+2) - SXT(regbim(3), ancho+2);
salida_imag  <= SXT(regbre(1), ancho+2) - SXT(regbre(3), ancho+2);
-------------------------------------------------------------------------
-- 1 rA(1) <= A1    rB(3) <= (-j)*(rA(1)-rA(3))                   --
--   B0 <= rB(0)+rB(2)                                             --
-------------------------------------------------------------------------
WHEN "01" =>
regaim(1)    <= imagx;
regare(1)    <= realx;
-- Para FFT se multiplica por *(-j)
IF inversa = '0' THEN
regbim(3)    <= SXT(regare(1), ancho+1) - SXT(regare(3), ancho+1);
regbre(3)    <= SXT(regaim(3), ancho+1) - SXT(regaim(1), ancho+1);
-- Para IFFT se multiplica por *(j)
ELSE
regbim(3)    <= SXT(regare(3), ancho+1) - SXT(regare(1), ancho+1);
regbre(3)    <= SXT(regaim(1), ancho+1) - SXT(regaim(3), ancho+1);
END IF;
salida_real  <= SXT(regbim(0), ancho+2) + SXT(regbim(2), ancho+2);
salida_imag  <= SXT(regbre(0), ancho+2) + SXT(regbre(2), ancho+2);
-------------------------------------------------------------------------
-- 2 rA(2) <= A2                                                  --
--   B1 <= rB(1)+rB(3)                                             --
```

```
-------------------------------------------------------------------
WHEN "10" =>
regaim(2)    <= imagx;
regare(2)    <= realx;
salida_real  <= SXT(regbim(1), ancho+2) + SXT(regbim(3), ancho+2);
salida_imag  <= SXT(regbre(1), ancho+2) + SXT(regbre(3), ancho+2);
-------------------------------------------------------------------
-- 3 rA(3) <= A3    rB(0) <= rA(0)+rA(2)                          --
--   B2 <= rB(0)-rB(2)                                           --
-------------------------------------------------------------------
WHEN "11" =>
regaim(3)    <= imagx;
regare(3)    <= realx;
regbim(0)    <= SXT(regaim(0), ancho+1) + SXT(regaim(2), ancho+1);
regbre(0)    <= SXT(regare(0), ancho+1) + SXT(regare(2), ancho+1);
salida_real  <= SXT(regbim(0), ancho+2) - SXT(regbim(2), ancho+2);
salida_imag  <= SXT(regbre(0), ancho+2) - SXT(regbre(2), ancho+2);
-------------------------------------------------------------------
WHEN OTHERS => NULL;
END CASE;
END IF;
END PROCESS CALC_RADIX_4;
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------
-- SUBSISTEMA DE CONTROL MAESTRO                                 --
-------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
-------------------------------------------------------------------
ENTITY CONTROLADOR_SUBSISTEMA IS
PORT (
      reloj       : IN  STD_LOGIC;
      fft_ocu     : IN  STD_LOGIC;
      uart_ok     : IN  STD_LOGIC;
      comando     : IN  STD_LOGIC;
      fft_ok      : IN  STD_LOGIC;
      fft_dir     : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      fft_datos   : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      start_fft   : OUT STD_LOGIC;
      start_uart  : OUT STD_LOGIC;
      led         : OUT STD_LOGIC;
      ram_esc_act : OUT STD_LOGIC;
      ram_dir     : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
      ram_datos   : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END ENTITY CONTROLADOR_SUBSISTEMA;
-------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF CONTROLADOR_SUBSISTEMA IS
TYPE ESTADO IS(e0, e1, e2, e3, e4, e5_ok);
SIGNAL
estado_prt      : ESTADO                      := e0;
SIGNAL
```

```
estado_sig      : ESTADO                        := e0;
SIGNAL
cont_prt        : INTEGER RANGE 0 TO 1023       := 0;
SIGNAL
cont_sig        : INTEGER RANGE 0 TO 1023       := 0;
SIGNAL
dir_prt         : STD_LOGIC_VECTOR(9 DOWNTO 0) := (OTHERS => '0');
SIGNAL
dir_sig         : STD_LOGIC_VECTOR(9 DOWNTO 0) := (OTHERS => '0');
SIGNAL
dato_prt        : STD_LOGIC_VECTOR(9 DOWNTO 0) := (OTHERS => '0');
SIGNAL
dato_sig        : STD_LOGIC_VECTOR(9 DOWNTO 0) := (OTHERS => '0');
SIGNAL
esc_act_prt     : STD_LOGIC                     := '0';
SIGNAL
esc_act_sig     : STD_LOGIC                     := '0';
BEGIN

PROCESS(reloj)
BEGIN
IF RISING_EDGE(reloj) THEN
estado_prt      <= estado_sig;
cont_prt        <= cont_sig;
dir_prt         <= dir_sig;
dato_prt        <= dato_sig;
esc_act_prt     <= esc_act_sig;
END IF;
END PROCESS;

PROCESS (
      estado_prt,
      fft_ocu,
      uart_ok,
      fft_ok,
      fft_dir,
      fft_datos,
      cont_prt,
      dir_prt,
      dato_prt,
      esc_act_prt,
      comando)
BEGIN
estado_sig      <= estado_prt;
cont_sig        <= cont_prt;
dir_sig         <= dir_prt;
dato_sig        <= dato_prt;
esc_act_sig     <= esc_act_prt;
start_fft       <= '0';
start_uart      <= '0';
led             <= '0';
CASE estado_prt IS

WHEN e0 =>
```

```
IF fft_ocu /= '1' THEN
start_fft       <= '1';
estado_sig      <= e1;
END IF;

WHEN e1 =>
IF fft_ok = '1' THEN
      IF cont_prt  =  1023 THEN
      esc_act_sig <= '0';
      dir_sig     <= (OTHERS => '0');
      cont_sig    <= 0;
      estado_sig  <= e5_ok;
      ELSE
      esc_act_sig <= '1';
      dato_sig    <= fft_datos;
      dir_sig     <= fft_dir;
      cont_sig    <= cont_prt + 1;
      END IF;
END IF;

WHEN e5_ok =>
led             <= '1';
IF comando       = '1' THEN
estado_sig      <= e2;
END IF;

WHEN e2 =>
start_uart      <= '1';
estado_sig      <= e3;

WHEN e3 =>
IF dir_prt       =  1023 THEN
dir_sig         <= (OTHERS => '0');
dato_sig        <= (OTHERS => '0');
esc_act_sig     <= '1';
estado_sig      <= e4;
ELSIF uart_ok    =  '1' THEN
dir_sig         <= dir_prt + 1;
estado_sig      <= e2;
END IF;

WHEN e4 =>
IF dir_prt       =  1023 THEN
esc_act_sig     <= '0';
dir_sig         <= (OTHERS => '0');
estado_sig      <= e0;
ELSE
dir_sig         <= dir_prt + 1;
END IF;
END CASE;
END PROCESS;
ram_esc_act     <= esc_act_prt;
ram_dir         <= dir_prt;
ram_datos       <= dato_prt;
```

```
END ARCHITECTURE COMPORTAMIENTO;

--------------------------------------------------------------------------
-- SUBSISTEMA PARA EL MANEJO DE DIRECCIONES                              --
--------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
USE     IEEE.STD_LOGIC_SIGNED.ALL;
--------------------------------------------------------------------------
ENTITY DIRECIONES IS
GENERIC (
      ancho          : NATURAL;
      puntos         : NATURAL;
      etapas         : NATURAL);
PORT(
      reloj          : IN  STD_LOGIC;
      reset          : IN  STD_LOGIC;
      start          : IN  STD_LOGIC;
      puerto_real    : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      puerto_imag    : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      fft_imag       : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      fft_real       : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      esc_datos_imag : OUT STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      esc_datos_real : OUT STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      lee_dir        : OUT STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0);
      esc_dir        : OUT STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0);
      act_esc        : OUT STD_LOGIC;
      start_factor   : OUT STD_LOGIC;
      start_cfft4    : OUT STD_LOGIC;
      salida_act     : OUT STD_LOGIC;
      entrada_ocu    : OUT STD_LOGIC;
      posicion_out   : OUT STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0));
END ENTITY DIRECIONES;
--------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF DIRECIONES IS

FUNCTION CONTADOR_A_DIRECCIONES (
      contador       : STD_LOGIC_VECTOR;
      mascara1       : STD_LOGIC_VECTOR;
      mascara2       : STD_LOGIC_VECTOR) RETURN STD_LOGIC_VECTOR IS
VARIABLE resultado : STD_LOGIC_VECTOR(contador'RANGE);
BEGIN
FOR n IN mascara1'RANGE LOOP
IF mascara1(n) = '1' THEN
resultado(2*n+1 DOWNTO 2*n)   := contador(1 DOWNTO 0);
ELSIF mascara2(n) = '1' AND n /= etapas-1 THEN
resultado(2*n+1 DOWNTO 2*n)   := contador(2*n+3 DOWNTO 2*n+2);
ELSE
resultado(2*n+1 DOWNTO 2*n)   := contador(2*n+1 DOWNTO 2*n);
END IF;
END LOOP;
RETURN resultado;
```

```
END FUNCTION CONTADOR_A_DIRECCIONES;

FUNCTION CONTADOR_A_DIRECCIONES_OUT (
      contador       : STD_LOGIC_VECTOR) RETURN STD_LOGIC_VECTOR IS
VARIABLE resultado : STD_LOGIC_VECTOR(contador'RANGE);
BEGIN
FOR n IN 0 TO etapas-1 LOOP
resultado(2*n+1 DOWNTO 2*n) :=
contador(contador'HIGH-2*n DOWNTO contador'HIGH-2*n-1);
END LOOP;
RETURN resultado;
END FUNCTION CONTADOR_A_DIRECCIONES_OUT;
SIGNAL
lee_estado         : STD_LOGIC_VECTOR(         3 DOWNTO 0);
SIGNAL
esc_estado         : STD_LOGIC_VECTOR(         3 DOWNTO 0);
SIGNAL
estado             : STD_LOGIC_VECTOR(         3 DOWNTO 0);
SIGNAL
lee_mascara1       : STD_LOGIC_VECTOR(  etapas-1 DOWNTO 0);
SIGNAL
lee_mascara2       : STD_LOGIC_VECTOR(  etapas-1 DOWNTO 0);
SIGNAL
esc_mascara1       : STD_LOGIC_VECTOR(  etapas-1 DOWNTO 0);
SIGNAL
esc_mascara2       : STD_LOGIC_VECTOR(  etapas-1 DOWNTO 0);
SIGNAL
contador           : STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0);
SIGNAL
esc_contador       : STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0);
SIGNAL
lee_contador       : STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0);
SIGNAL
contador_out       : STD_LOGIC_VECTOR(  etapas*2 DOWNTO 0);
CONSTANT retardo_fft    : INTEGER := 12 + 2*etapas;
CONSTANT retardo_factor : INTEGER :=  6;
CONSTANT retardo_salida : INTEGER :=  7;
BEGIN
salida_act       <= contador_out(etapas*2);
posicion_out     <=
CONTADOR_A_DIRECCIONES_OUT(contador_out(etapas*2-1 DOWNTO 0 ));

CONTEO : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
contador         <= (OTHERS => '0');
estado           <= CONV_STD_LOGIC_VECTOR(etapas + 1, 4);
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF start = '1' THEN
      contador     <= (OTHERS => '0');
      estado       <= (OTHERS => '0');
      ELSIF UNSIGNED(estado) /= etapas + 1 THEN
      contador     <= UNSIGNED(contador) + 1;
            IF SIGNED(contador) = -1 THEN
```

```
            estado   <= UNSIGNED(estado) + 1;
            END IF;
      END IF;
END IF;
END PROCESS CONTEO;

LECTURA_DIR : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
lee_dir          <= (OTHERS => '0');
lee_contador     <= (OTHERS => '0');
lee_estado       <= (OTHERS => '0');
lee_mascara1     <= (OTHERS => '0');
lee_mascara2     <= (OTHERS => '0');
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF UNSIGNED(estado) = 0 AND SIGNED(contador) = -1 THEN
      lee_mascara1(etapas-1)          <= '1';
      lee_mascara1(etapas-2 DOWNTO 0) <= (OTHERS => '0');
      lee_mascara2(etapas-1)          <= '0';
      lee_mascara2(etapas-2 DOWNTO 0) <= (OTHERS => '1');
      ELSIF SIGNED(contador) = -1 THEN
      lee_mascara1 <= '0' & lee_mascara1(etapas-1 DOWNTO 1);
      lee_mascara2 <= '0' & lee_mascara2(etapas-1 DOWNTO 1);
      END IF;
      IF UNSIGNED(estado) /= etapas + 1 AND SIGNED(contador) = -1 THEN
      lee_contador <= (OTHERS => '0');
      lee_estado   <= estado;
      ELSE
      lee_contador <= UNSIGNED(lee_contador) + 1;
      END IF;
      lee_dir      <=
      CONTADOR_A_DIRECCIONES(lee_contador, lee_mascara1, lee_mascara2);
END IF;
END PROCESS LECTURA_DIR;

ESCRITURA_DIR : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
esc_dir          <= (OTHERS => '0');
esc_contador     <= (OTHERS => '0');
esc_estado       <= (OTHERS => '0');
esc_mascara1     <= (OTHERS => '0');
esc_mascara2     <= (OTHERS => '0');
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF UNSIGNED(estado) = 0 THEN
      esc_dir      <= contador;
      ELSE
            IF UNSIGNED(lee_estado) = 0 AND
            UNSIGNED(lee_contador) = retardo_fft - 1 THEN
            esc_mascara1(etapas-1)          <= '1';
            esc_mascara1(etapas-2 DOWNTO 0) <= (OTHERS => '0');
            esc_mascara2(etapas-1)          <= '0';
            esc_mascara2(etapas-2 DOWNTO 0) <= (OTHERS => '1');
            ELSIF UNSIGNED(lee_contador)  = retardo_fft - 1 THEN
```

```
            esc_mascara1 <= '0' & esc_mascara1(etapas-1 DOWNTO 1);
            esc_mascara2 <= '0' & esc_mascara2(etapas-1 DOWNTO 1);
            END IF;

            IF UNSIGNED(lee_estado) < etapas AND
            UNSIGNED(lee_contador)  = retardo_fft - 1 THEN
            esc_contador                <= (OTHERS => '0');
            esc_estado                  <= lee_estado;
            ELSE
            esc_contador                <= UNSIGNED(esc_contador) + 1;
            END IF;
            esc_dir <= CONTADOR_A_DIRECCIONES (
                  esc_contador,
                  esc_mascara1,
                  esc_mascara2);
      END IF;
END IF;
END PROCESS ESCRITURA_DIR;

ACTIVADOR_ESCRITURA : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
act_esc <= '0';
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF UNSIGNED(estado)     =  0  THEN
      act_esc <= '1';
      ELSIF UNSIGNED(estado)  = 1 AND
            UNSIGNED(contador)  = 0 THEN
      act_esc <= '0';
      ELSIF UNSIGNED(lee_estado) = 0 AND
            UNSIGNED(lee_contador) = retardo_fft THEN
      act_esc <= '1';
      ELSIF UNSIGNED(lee_estado) = etapas - 1 AND
      UNSIGNED(lee_contador)     = retardo_fft THEN
      act_esc <= '0';
      END IF;
END IF;
END PROCESS ACTIVADOR_ESCRITURA;

OTRO_INICIO : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
start_factor    <= '0';
start_cfft4     <= '0';
contador_out    <= (OTHERS => '0');
entrada_ocu     <= '0';
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF start     = '1' THEN
      entrada_ocu <= '1';
      ELSIF UNSIGNED(estado)  = etapas AND
      UNSIGNED(contador)      = retardo_fft  THEN
      entrada_ocu <= '0';
      END IF;
```

```
      IF UNSIGNED(estado)     = 1 AND UNSIGNED(contador) = 0 THEN
      start_cfft4 <= '1';
      ELSE
      start_cfft4 <= '0';
      END IF;

      IF UNSIGNED(lee_estado) = 0 AND
      UNSIGNED(lee_contador)  = retardo_factor THEN
      start_factor <= '1';
      ELSE
      start_factor <= '0';
      END IF;

      IF UNSIGNED(estado) = etapas AND
            UNSIGNED(lee_contador) = retardo_salida THEN
      contador_out  <= CONV_STD_LOGIC_VECTOR(puntos, 2*etapas+1);
      ELSIF contador_out(etapas*2) = '1' THEN
      contador_out  <= UNSIGNED(contador_out) + 1;
      END IF;
END IF;
END PROCESS OTRO_INICIO;

DATOS : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
esc_datos_imag     <= (OTHERS => '0');
esc_datos_real     <= (OTHERS => '0');
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF UNSIGNED(estado) = 0 THEN
      esc_datos_imag <= puerto_real;
      esc_datos_real <= puerto_imag;
      ELSE
      esc_datos_imag <= fft_imag;
      esc_datos_real <= fft_real;
      END IF;
END IF;
END PROCESS DATOS;
END ARCHITECTURE COMPORTAMIENTO;

------------------------------------------------------------------------
-- DIV4LIMIT                                                          --
------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
------------------------------------------------------------------------
ENTITY DIV4LIMIT IS
GENERIC(ancho : NATURAL);
PORT (
      reloj     : IN  STD_LOGIC;
      datos     : IN  STD_LOGIC_VECTOR(ancho+3 DOWNTO 0);
      realx     : OUT STD_LOGIC_VECTOR(ancho-1 DOWNTO 0));
END ENTITY DIV4LIMIT;
------------------------------------------------------------------------
```

```
ARCHITECTURE COMPORTAMIENTO OF DIV4LIMIT IS
BEGIN
PROCESS(reloj)
VARIABLE temp_d : STD_LOGIC_VECTOR(ancho+1 DOWNTO 0);
BEGIN
IF reloj'EVENT AND reloj = '1' THEN
temp_d := datos(ancho+3 DOWNTO 2) + datos(1);
      IF temp_d(ancho+1) = '1' AND temp_d(   ancho DOWNTO ancho-1) /=
      "11" THEN
      temp_d( ancho+1 DOWNTO ancho-1) := "111";
      temp_d( ancho-2 DOWNTO       1) := (OTHERS => '0');
      temp_d(0)                       := '1';
      ELSIF temp_d(ancho+1) = '0' AND temp_d(ancho DOWNTO ancho-1) /=
      "00" THEN
      temp_d( ancho+1 DOWNTO ancho-1) := "000";
      temp_d( ancho-2 DOWNTO       0) := (OTHERS => '1');
      END IF;
realx <= temp_d(ancho-1 DOWNTO 0 );
END IF;
END PROCESS;
END ARCHITECTURE COMPORTAMIENTO;

--------------------------------------------------------------------------
-- SUBSISTEMA PARA EL CÁLCULO DE LA TRANSFORMADA RÁPIDA DE FOURIER --
--------------------------------------------------------------------------
-- Basado en cfft_core disponible bajo liccencia de código abierto --
-- en opencores.org                                                --
-- Se conservan los nombres originales del diseño publicado        --
--------------------------------------------------------------------------
-- reloj       : Reloj principal.                                  --
--             -> Probado hasta 250 MHz.                           --
-- reset       : Reset global.                                     --
--             -> '1' para reestablecer todo.                      --
-- start       : Inicio de la FFT.                                 --
--             -> El primer pulso alto del reloj (1) antes de la   --
--                entrada de datos.                                --
-- inversa     : Selector de función.                              --
--             -> '0' para FFT y '1' para IFFT, se muestrea cuando --
--                start está en '1'.                               --
-- puerto_real : Entrada de datos imaginarios.                     --
-- puerto_imag : Entrada de datos imaginarios.                     --
--             -> Comienzan inmediatamente, la potencia de los datos--
--                de entrada no debe ser demasiado grande.         --
-- entrada_ocu : Señal de entrada ocupada.                         --
--             -> Si cambia a '0', la siguiente FFT se habilita.   --
-- salida_act  : Habilitador de salida de datos.                   --
--             -> Cuando es '1', los datos válidos están en la     --
--                salida.                                          --
-- salida_real    : Salidas de datos imaginarios.                  --
-- salida_imag    : Salidas de datos reales.                       --
--             -> Salida de la FFT cuando salida_act es '1'.       --
--------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
```

```
USE     IEEE.STD_LOGIC_ARITH.ALL;
-----------------------------------------------------------------------
ENTITY FFT_SUBSISTEMA IS
GENERIC (
      ancho          : NATURAL :=    8;  -- 8 bits de precisión
      puntos         : NATURAL := 1024;  -- FFT de 1024 puntos
      etapas         : NATURAL :=    5); -- etapas = log4(puntos)
PORT (
      reloj          : IN  STD_LOGIC;
      reset          : IN  STD_LOGIC;
      start          : IN  STD_LOGIC;
      inversa        : IN  STD_LOGIC;
      puerto_real    : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      puerto_imag    : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      entrada_ocu    : OUT STD_LOGIC;
      salida_act     : OUT STD_LOGIC;
      salida_real       : OUT STD_LOGIC_VECTOR(   ancho+1 DOWNTO 0);
      salida_imag       : OUT STD_LOGIC_VECTOR(   ancho+1 DOWNTO 0);
      posicion_out   : OUT STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0));
END ENTITY FFT_SUBSISTEMA;
-----------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF FFT_SUBSISTEMA IS

COMPONENT DIRECIONES
GENERIC (
      ancho          : NATURAL;
      puntos         : NATURAL;
      etapas         : NATURAL);
PORT (
      reloj          : IN  STD_LOGIC;
      reset          : IN  STD_LOGIC;
      start          : IN  STD_LOGIC;
      puerto_real    : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      puerto_imag    : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      fft_imag       : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      fft_real       : IN  STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      esc_datos_imag : OUT STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      esc_datos_real : OUT STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0);
      lee_dir        : OUT STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0);
      esc_dir        : OUT STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0);
      act_esc        : OUT STD_LOGIC;
      start_factor   : OUT STD_LOGIC;
      start_cfft4    : OUT STD_LOGIC;
      salida_act     : OUT STD_LOGIC;
      entrada_ocu    : OUT STD_LOGIC;
      posicion_out   : OUT STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0));
END COMPONENT;

COMPONENT MEM_DRAM
GENERIC (
      profundidad    : INTEGER;
      ancho_datos    : INTEGER;
      ancho_dir      : INTEGER);
PORT (
```

```
      dir_a          : IN  STD_LOGIC_VECTOR(  ancho_dir-1 DOWNTO 0);
      reloj_a        : IN  STD_LOGIC;
      dir_b          : IN  STD_LOGIC_VECTOR(  ancho_dir-1 DOWNTO 0);
      reloj_b        : IN  STD_LOGIC;
      d_in_a            : IN  STD_LOGIC_VECTOR(ancho_datos-1 DOWNTO 0);
      act_esc_a      : IN  STD_LOGIC;
      d_out_b        : OUT STD_LOGIC_VECTOR(ancho_datos-1 DOWNTO 0));
END COMPONENT;

COMPONENT CFFT4
GENERIC(ancho      : NATURAL);
PORT (
      reloj          : IN  STD_LOGIC;
      reset          : IN  STD_LOGIC;
      start          : IN  STD_LOGIC;
      inversa        : IN  STD_LOGIC;
      imagx          : IN  STD_LOGIC_VECTOR(ancho-1 DOWNTO 0);
      realx          : IN  STD_LOGIC_VECTOR(ancho-1 DOWNTO 0);
      salida_real       : OUT STD_LOGIC_VECTOR(ancho+1 DOWNTO 0);
      salida_imag       : OUT STD_LOGIC_VECTOR(ancho+1 DOWNTO 0));
END COMPONENT;

COMPONENT DIV4LIMIT
GENERIC(ancho      : NATURAL);
PORT (
      reloj          : IN  STD_LOGIC;
      datos          : IN  STD_LOGIC_VECTOR(ancho+3 DOWNTO 0);
      realx          : OUT STD_LOGIC_VECTOR(ancho-1 DOWNTO 0));
END COMPONENT;

COMPONENT MULFACTOR
GENERIC (
      ancho          : NATURAL;
      etapas         : NATURAL);
PORT (
      reloj          : IN  STD_LOGIC;
      reset          : IN  STD_LOGIC;
      angulo          : IN  SIGNED(2*etapas-1 DOWNTO 0);
      imagx          : IN  SIGNED(   ancho+1 DOWNTO 0);
      realx          : IN  SIGNED(   ancho+1 DOWNTO 0);
      salida_real       : OUT SIGNED(   ancho+3 DOWNTO 0);
      salida_imag       : OUT SIGNED(   ancho+3 DOWNTO 0));
END COMPONENT;

COMPONENT ROFACTOR
GENERIC (
      puntos         : NATURAL;
      etapas         : NATURAL);
PORT (
      reloj          : IN  STD_LOGIC;
      reset          : IN  STD_LOGIC;
      start          : IN  STD_LOGIC;
      inversa        : IN  STD_LOGIC;
      angulo         : OUT STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0));
```

```
END COMPONENT;
SIGNAL
act_salida          : STD_LOGIC := '0';
SIGNAL
orden               : STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0) :=
                      (OTHERS=>'0');
SIGNAL
datos_realx         : STD_LOGIC_VECTOR(   ancho+1 DOWNTO 0) :=
                      (OTHERS=>'0');
SIGNAL
act_esc_a           : STD_LOGIC := '0';
SIGNAL
start_cfft4         : STD_LOGIC := '0';
SIGNAL
start_factor        : STD_LOGIC := '0';
SIGNAL
esc_datos_imag      : STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0) :=
                      (OTHERS => '0');
SIGNAL
esc_datos_real      : STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0) :=
                      (OTHERS => '0');
SIGNAL
fft_imag            : STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0) :=
                      (OTHERS => '0');
SIGNAL
fft_real            : STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0) :=
                      (OTHERS => '0');
SIGNAL
imagram_out         : STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0) :=
                      (OTHERS => '0');
SIGNAL
realram_out         : STD_LOGIC_VECTOR(   ancho-1 DOWNTO 0) :=
                      (OTHERS => '0');
SIGNAL
esc_dir             : STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0) :=
                      (OTHERS=>'0');
SIGNAL
lee_dir             : STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0) :=
                      (OTHERS=>'0');
SIGNAL
imagcfft4_out       : STD_LOGIC_VECTOR(   ancho+1 DOWNTO 0) :=
                      (OTHERS => '0');
SIGNAL
realcfft4_out       : STD_LOGIC_VECTOR(   ancho+1 DOWNTO 0) :=
                      (OTHERS => '0');
SIGNAL
angulo              : STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0) :=
                      (OTHERS=>'0');
SIGNAL
imagmul_out         : SIGNED(ancho+3 DOWNTO 0) := (OTHERS => '0');
SIGNAL
realmul_out         : SIGNED(ancho+3 DOWNTO 0) := (OTHERS => '0');
SIGNAL
reg_inv             : STD_LOGIC := '0';
```

```
BEGIN

DIRECCIONADOR : DIRECIONES
GENERIC MAP (
      ancho          => ancho,
      puntos         => puntos,
      etapas         => etapas)
PORT MAP (
      reloj          => reloj,
      reset          => reset,
      start          => start,
      puerto_real    => puerto_real,
      puerto_imag    => puerto_imag,
      fft_imag       => fft_imag,
      fft_real       => fft_real,
      esc_datos_imag => esc_datos_imag,
      esc_datos_real => esc_datos_real,
      lee_dir        => lee_dir,
      esc_dir        => esc_dir,
      act_esc        => act_esc_a,
      start_factor   => start_factor,
      start_cfft4    => start_cfft4,
      salida_act     => act_salida,
      entrada_ocu    => entrada_ocu,
      posicion_out   => orden);

MEMORIA_PARA_MANEJO_DE_DATOS_IMAGINARIOS : MEM_DRAM
GENERIC MAP (
      profundidad    => puntos,
      ancho_datos    => ancho,
      ancho_dir      => 2*etapas)
PORT MAP (
      dir_a          => esc_dir,
      reloj_a        => reloj,
      dir_b          => lee_dir,
      reloj_b        => reloj,
      d_in_a         => esc_datos_imag,
      act_esc_a      => act_esc_a,
      d_out_b        => imagram_out);

MEMORIA_PARA_MANEJO_DE_DATOS_REALES : MEM_DRAM
GENERIC MAP (
      profundidad    => puntos,
      ancho_datos    => ancho,
      ancho_dir      => 2*etapas)
PORT MAP (
      dir_a          => esc_dir,
      reloj_a        => reloj,
      dir_b          => lee_dir,
      reloj_b        => reloj,
      d_in_a         => esc_datos_real,
      act_esc_a      => act_esc_a,
      d_out_b        => realram_out);
```

```
NUCLEO_DE_LA_FFT_RADIX4 : CFFT4
GENERIC MAP(ancho  => ancho)
PORT MAP (
      reloj          => reloj,
      reset          => reset,
      start          => start_cfft4,
      inversa        => reg_inv,
      imagx          => imagram_out,
      realx          => realram_out,
      salida_real       => imagcfft4_out,
      salida_imag       => realcfft4_out);
salida_real <= imagcfft4_out;
salida_imag <= realcfft4_out;

LIMITE_DATOS_IMAGINARIOS : DIV4LIMIT
GENERIC MAP(ancho  => ancho)
PORT MAP (
      reloj          => reloj,
      datos          => STD_LOGIC_VECTOR(imagmul_out),
      realx          => fft_imag);

LIMITE_DATOS_REALES : DIV4LIMIT
GENERIC MAP (
      ancho          => ancho)
PORT MAP (
      reloj          => reloj,
      datos          => STD_LOGIC_VECTOR(realmul_out),
      realx          => fft_real);

CONFIGURACION_DE_FACTORES : MULFACTOR
GENERIC MAP (
      ancho          => ancho,
      etapas         => etapas)
PORT MAP (
      reloj          => reloj,
      reset          => reset,
      angulo         => SIGNED(angulo),
      imagx          => SIGNED(imagcfft4_out),
      realx          => SIGNED(realcfft4_out),
      salida_real       => imagmul_out,
      salida_imag       => realmul_out);

CONFIGURACION_DE_ROTACIONES : ROFACTOR
GENERIC MAP (
      puntos         => puntos,
      etapas         => etapas)
PORT MAP (
      reloj          => reloj,
      reset          => reset,
      start          => start_factor,
      inversa        => reg_inv,
      angulo         => angulo);

PROCESS(reloj, reset)
```

```
BEGIN
IF reset = '1' THEN
reg_inv     <= '0';
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF start = '1' THEN
      reg_inv <= inversa;
      END IF;
END IF;
END PROCESS;
salida_act   <= act_salida;
posicion_out <= orden;
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- GENERADOR DE LA SEÑAL PULSO_RX                                       --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
-------------------------------------------------------------------------
ENTITY GEN_PULSO_RX IS
GENERIC (max : INTEGER := 163);
PORT (
      reloj    : IN  STD_LOGIC;
      pulso_rx : OUT STD_LOGIC);
END ENTITY GEN_PULSO_RX;
-------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF GEN_PULSO_RX IS
SIGNAL cont_prt     : INTEGER RANGE 0 TO max - 1 := 0;
SIGNAL cont_sig     : INTEGER RANGE 0 TO max - 1 := 0;
BEGIN
PROCESS(reloj)
BEGIN
IF RISING_EDGE(reloj) THEN
cont_prt <= cont_sig;
END IF;
END PROCESS;
cont_sig <=  0  WHEN cont_prt = max - 1 ELSE cont_prt + 1;
pulso_rx <= '1' WHEN cont_prt = max - 1 ELSE '0';
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- GENERADOR DE LA SEÑAL PULSO_TX                                       --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
-------------------------------------------------------------------------
ENTITY GEN_PULSO_TX IS
GENERIC (max    : INTEGER := 2604);
-- La frecuencia del pulso es 16 veces más lenta que la del receptor
-- UART.
```

```
PORT (
      reloj       : IN  STD_LOGIC;
      pulso_tx    : OUT STD_LOGIC);
END ENTITY GEN_PULSO_TX;
-------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF GEN_PULSO_TX IS
SIGNAL cont_prt : INTEGER RANGE 0 TO max := 0;
SIGNAL cont_sig : INTEGER RANGE 0 TO max := 0;
BEGIN
PROCESS (reloj)
BEGIN
IF RISING_EDGE(reloj) THEN
cont_prt <= cont_sig;
END IF;
END PROCESS;
cont_sig <=  0  WHEN cont_prt = max ELSE cont_prt + 1;
pulso_tx <= '1' WHEN cont_prt = max ELSE '0';
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- BLOQUE DE MEMORIA DRAM                                               --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
-------------------------------------------------------------------------
ENTITY MEM_DRAM IS
GENERIC (
      profundidad : INTEGER;
      ancho_datos : INTEGER;
      ancho_dir   : INTEGER);
PORT (
      dir_a       : IN  STD_LOGIC_VECTOR(  ancho_dir-1 DOWNTO 0);
      reloj_a     : IN  STD_LOGIC;
      dir_b       : IN  STD_LOGIC_VECTOR(  ancho_dir-1 DOWNTO 0);
      reloj_b     : IN  STD_LOGIC;
      d_in_a      : IN  STD_LOGIC_VECTOR(ancho_datos-1 DOWNTO 0);
      act_esc_a   : IN  STD_LOGIC;
      d_out_b     : OUT STD_LOGIC_VECTOR(ancho_datos-1 DOWNTO 0));
END ENTITY MEM_DRAM;
-------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF MEM_DRAM IS
TYPE TIPO_RAM IS ARRAY(profundidad-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(      ancho_datos-1 DOWNTO 0);
SIGNAL memoria  : TIPO_RAM := (OTHERS => (OTHERS => '0'));
SIGNAL reg_dirb : STD_LOGIC_VECTOR(ancho_dir-1 DOWNTO 0);
BEGIN

ESCRITURA : PROCESS(reloj_a)
BEGIN
IF RISING_EDGE(reloj_a) THEN
      IF act_esc_a = '1' THEN
      memoria(CONV_INTEGER(dir_a)) <= d_in_a;
```

```
      END IF;
END IF;
END PROCESS ESCRITURA;

LECTURA : PROCESS(reloj_b)
BEGIN
IF RISING_EDGE(reloj_b) THEN
reg_dirb <= dir_b;
END IF;
END PROCESS LECTURA;

d_out_b   <= memoria(CONV_INTEGER(reg_dirb));
END ARCHITECTURE COMPORTAMIENTO;

--------------------------------------------------------------------------
-- SUBSISTEMA DE MEMORIA RAM                                            --
--------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------------------------------------------------
ENTITY MEM_RAM_SUBSISTEMA IS
PORT (
      reloj   : IN  STD_LOGIC;
      esc_act : IN  STD_LOGIC;
      dir     : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      din     : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      dout    : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END ENTITY MEM_RAM_SUBSISTEMA;
--------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF MEM_RAM_SUBSISTEMA IS
TYPE MEM_RAM IS ARRAY(0 TO 1023) OF STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL ram : MEM_RAM := (OTHERS => (OTHERS => '0'));
BEGIN
PROCESS(reloj)
BEGIN
IF RISING_EDGE(reloj) THEN
      IF esc_act = '1' THEN
      ram(CONV_INTEGER(dir)) <= din;
      END IF;
END IF;
END PROCESS;
dout <= ram(CONV_INTEGER(dir));
END ARCHITECTURE COMPORTAMIENTO;

--------------------------------------------------------------------------
-- MULFACTOR                                                            --
--------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_SIGNED.ALL;
--------------------------------------------------------------------------
```

```
ENTITY MULFACTOR IS
GENERIC (
      ancho       : NATURAL;
      etapas      : NATURAL);
PORT (
      reloj       : IN  STD_LOGIC;
      reset       : IN  STD_LOGIC;
      angulo      : IN  SIGNED(2*etapas-1 DOWNTO 0);
      imagx       : IN  SIGNED(   ancho+1 DOWNTO 0);
      realx       : IN  SIGNED(   ancho+1 DOWNTO 0);
      salida_real : OUT SIGNED(   ancho+3 DOWNTO 0);
      salida_imag : OUT SIGNED(   ancho+3 DOWNTO 0));
END ENTITY MULFACTOR;
--------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF MULFACTOR IS
SIGNAL fase       : SIGNED(2*etapas-3 DOWNTO 0);
SIGNAL xi         : SIGNED(   ancho+1 DOWNTO 0);
SIGNAL yi         : SIGNED(   ancho+1 DOWNTO 0);
COMPONENT SC_CORPROC
GENERIC (
      ancho       : NATURAL;
      etapas      : NATURAL);
PORT (
      reloj       : IN  STD_LOGIC;
      habilitador : IN  STD_LOGIC;
      xin         : IN  SIGNED(   ancho+1 DOWNTO 0);
      yin         : IN  SIGNED(   ancho+1 DOWNTO 0);
      ain         : IN  SIGNED(2*etapas-3 DOWNTO 0);
      seno        : OUT SIGNED(   ancho+3 DOWNTO 0);
      coseno      : OUT SIGNED(   ancho+3 DOWNTO 0));
END COMPONENT;
BEGIN

UNIDAD_1 : SC_CORPROC
GENERIC MAP (
      ancho       => ancho,
      etapas      => etapas)
PORT MAP (
      reloj       => reloj,
      habilitador => '1',
      xin         => xi,
      yin         => yi,
      ain         => fase,
      seno        => salida_imag,
      coseno      => salida_real);

PROCESS(reloj, reset)
VARIABLE temporal : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
IF reset = '1' THEN
fase <= (OTHERS => '0');
xi   <= (OTHERS => '0');
yi   <= (OTHERS => '0');
ELSIF reloj'EVENT AND reloj = '1' THEN
```

```
fase <= angulo(2*etapas-3 DOWNTO 0);
temporal := STD_LOGIC_VECTOR(angulo(2*etapas-1 DOWNTO 2*etapas-2));
CASE temporal IS
WHEN "00"       => xi <=   imagx; yi <=   realx;
WHEN "01"       => xi <= 0-realx; yi <=   imagx;
WHEN "10"       => xi <= 0-imagx; yi <= 0-realx;
WHEN "11"       => xi <=   realx; yi <= 0-imagx;
WHEN OTHERS     => NULL;
END CASE;
END IF;
END PROCESS;
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- GENERACIÓN DE ARQUITECTURA PIPELINE PARA EL ALGORITMO DE VOLDER --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
-------------------------------------------------------------------------
ENTITY P2R_CORDIC IS
GENERIC (
      pipeline    : INTEGER := 15;
      ancho       : INTEGER := 16);
PORT (
      reloj       : IN  STD_LOGIC;
      habilitador : IN  STD_LOGIC;
      xi          : IN  SIGNED(ancho-1 DOWNTO 0);
      yi          : IN  SIGNED(ancho-1 DOWNTO 0) := (OTHERS => '0');
      Zi          : IN  SIGNED(     19 DOWNTO 0);
      xo          : OUT SIGNED(ancho-1 DOWNTO 0);
      yo          : OUT SIGNED(ancho-1 DOWNTO 0));
END ENTITY P2R_CORDIC;
-------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF P2R_CORDIC IS
TYPE
vector_xy IS ARRAY(pipeline DOWNTO 0) OF SIGNED(ancho-1 DOWNTO 0);
TYPE
vector_z  IS ARRAY(pipeline DOWNTO 0) OF SIGNED(     19 DOWNTO 0);

COMPONENT P2R_CORDICPIPE
GENERIC (
      ancho       : NATURAL := 16;
      pipeid      : NATURAL :=  1);
PORT (
      reloj       : IN  STD_LOGIC;
      habilitador : IN  STD_LOGIC;
      xi          : IN  SIGNED(ancho-1 DOWNTO 0);
      yi          : IN  SIGNED(ancho-1 DOWNTO 0);
      Zi          : IN  SIGNED(     19 DOWNTO 0);
      xo          : OUT SIGNED(ancho-1 DOWNTO 0);
      yo          : OUT SIGNED(ancho-1 DOWNTO 0);
      Zo          : OUT SIGNED(     19 DOWNTO 0));
END COMPONENT P2R_CORDICPIPE;
```

```
SIGNAL x        : vector_xy;
SIGNAL y        : vector_xy;
SIGNAL z        : vector_z;
BEGIN
-- Llenando los primeros nodos.
-- Llenando x.
x(0)              <= xi;
-- Llenando y.
y(0)              <= yi;
-- Llenando z.
z(0)(19 DOWNTO 0) <= Zi;
-- Generando la arquitectura pipeline.
GEN_PIPELINE:
FOR n IN 1 TO pipeline GENERATE

PIPELINE : P2R_CORDICPIPE
GENERIC MAP (
      ancho  => ancho,
      pipeid => n -1)
PORT MAP (
      reloj,
      habilitador,
      x(n-1),
      y(n-1),
      z(n-1),
      x(n),
      y(n),
      z(n));
END GENERATE GEN_PIPELINE;
xo <= x(pipeline);
yo <= y(pipeline);
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- P2R_CORDICPIPE                                                      --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
-------------------------------------------------------------------------
ENTITY P2R_CORDICPIPE IS
GENERIC (
      ancho       : NATURAL := 16;
      pipeid      : NATURAL :=  1);
PORT(
      reloj       : IN  STD_LOGIC;
      habilitador : IN  STD_LOGIC;
      xi          : IN  SIGNED(ancho-1 DOWNTO 0);
      yi          : IN  SIGNED(ancho-1 DOWNTO 0);
      Zi          : IN  SIGNED(     19 DOWNTO 0);
      xo          : OUT SIGNED(ancho-1 DOWNTO 0);
      yo          : OUT SIGNED(ancho-1 DOWNTO 0);
      Zo          : OUT SIGNED(     19 DOWNTO 0));
```

```
END ENTITY P2R_CORDICPIPE;
-------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF P2R_CORDICPIPE IS

FUNCTION ARCOTANGENTE(n  : NATURAL) RETURN INTEGER IS
VARIABLE resultado       : INTEGER;
BEGIN
CASE n IS
WHEN      0 => resultado := 16#020000#;
WHEN      1 => resultado := 16#012E40#;
WHEN      2 => resultado :=  16#09FB4#;
WHEN      3 => resultado :=  16#05111#;
WHEN      4 => resultado :=  16#028B1#;
WHEN      5 => resultado :=  16#0145D#;
WHEN      6 => resultado :=   16#0A2F#;
WHEN      7 => resultado :=   16#0518#;
WHEN      8 => resultado :=   16#028C#;
WHEN      9 => resultado :=   16#0146#;
WHEN     10 => resultado :=    16#0A3#;
WHEN     11 => resultado :=    16#051#;
WHEN     12 => resultado :=    16#029#;
WHEN     13 => resultado :=    16#014#;
WHEN     14 => resultado :=     16#0A#;
WHEN     15 => resultado :=     16#05#;
WHEN     16 => resultado :=     16#03#;
WHEN     17 => resultado :=     16#01#;
WHEN OTHERS => resultado :=      16#0#;
END CASE;
RETURN resultado;
END ARCOTANGENTE;

-- La función DELTA es en realidad un desplazamiento aritmético a la
-- derecha.
FUNCTION DELTA(arg : SIGNED; lim : NATURAL) RETURN SIGNED IS
VARIABLE temporal  : SIGNED(arg'RANGE);
CONSTANT bajo      : INTEGER := arg'HIGH - lim + 1;
BEGIN
FOR n IN arg'HIGH DOWNTO bajo LOOP
temporal(n) := arg(arg'HIGH);
END LOOP;
FOR n IN arg'HIGH - lim DOWNTO 0 LOOP
temporal(n) := arg(n + lim);
END LOOP;
RETURN temporal;
END FUNCTION DELTA;

FUNCTION SUMADOR_RESTADOR (
      datoa   : IN SIGNED;
      datob   : IN SIGNED;
      sum_res : IN STD_LOGIC) RETURN SIGNED IS
BEGIN
IF (sum_res = '1') THEN
RETURN datoa + datob;
ELSE
```

```
RETURN datoa - datob;
END IF;
END;
--
-- Cuerpo De La Arquitectura
--
SIGNAL dx          : SIGNED(ancho-1 DOWNTO 0);
SIGNAL xresult     : SIGNED(ancho-1 DOWNTO 0);
SIGNAL dy          : SIGNED(ancho-1 DOWNTO 0);
SIGNAL yresult     : SIGNED(ancho-1 DOWNTO 0);
SIGNAL atan        : SIGNED(     19 DOWNTO 0);
SIGNAL zresult     : SIGNED(     19 DOWNTO 0);
SIGNAL zneg        : STD_LOGIC;
SIGNAL zpos        : STD_LOGIC;
BEGIN
dx      <= DELTA(xi, pipeid);
dy      <= DELTA(yi, pipeid);
atan    <= CONV_SIGNED(ARCOTANGENTE(pipeid), 20);
-- Generando estructuras de sumador.
zneg    <= Zi(19);
zpos    <= NOT Zi(19);
-- xadd
xresult <= SUMADOR_RESTADOR(xi, dy, zneg);
-- yadd
yresult <= SUMADOR_RESTADOR(yi, dx, zpos);
-- zadd
zresult <= SUMADOR_RESTADOR(Zi, atan, zneg);

GENERADOR_DE_REGISTROS : PROCESS(reloj)
BEGIN
IF(reloj'EVENT AND reloj = '1') THEN
      IF (habilitador = '1') THEN
      xo  <= xresult;
      yo  <= yresult;
      Zo  <= zresult;
      END IF;
END IF;
END PROCESS GENERADOR_DE_REGISTROS;
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- RECEPTOR                                                            --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
-------------------------------------------------------------------------
ENTITY RECEPTOR IS
PORT (
      reloj    : IN  STD_LOGIC;
      reset    : IN  STD_LOGIC;
      datos_in : IN  STD_LOGIC;
      pulso_rx : IN  STD_LOGIC;
```

```
      dout     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      carga    : OUT STD_LOGIC);
END ENTITY RECEPTOR;
-------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF RECEPTOR IS
TYPE ESTADO IS(e0, e1, e2, e3);
SIGNAL
estado_prt   : ESTADO                       := e0;
SIGNAL
estado_sig   : ESTADO                       := e0;
SIGNAL
cont_prt     : STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');
SIGNAL
cont_sig     : STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');
SIGNAL
bit_prt      : STD_LOGIC_VECTOR(2 DOWNTO 0) := (OTHERS => '0');
SIGNAL
bit_sig      : STD_LOGIC_VECTOR(2 DOWNTO 0) := (OTHERS => '0');
SIGNAL
reg_prt      : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL
reg_sig      : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL
dout_prt     : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL
dout_sig     : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL
carga_prt    : STD_LOGIC                    := '0';
SIGNAL
carga_sig    : STD_LOGIC                    := '0';
BEGIN

PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
estado_prt         <= e0;
cont_prt           <= "0000";
bit_prt            <= "000";
carga_prt          <= '0';
dout_prt           <= (OTHERS => '0');
ELSIF RISING_EDGE(reloj) THEN
estado_prt         <= estado_sig;
cont_prt           <= cont_sig;
bit_prt            <= bit_sig;
reg_prt            <= reg_sig;
carga_prt          <= carga_sig;
dout_prt           <= dout_sig;
END IF;
END PROCESS;

PROCESS (
      pulso_rx,
      estado_prt,
      reg_prt,
```

```
      cont_prt,
      bit_prt,
      datos_in,
      carga_prt,
      dout_prt)
BEGIN
cont_sig          <= cont_prt;
bit_sig           <= bit_prt;
reg_sig           <= reg_prt;
estado_sig        <= estado_prt;
carga_sig         <= carga_prt;
dout_sig          <= dout_prt;
CASE estado_prt IS

WHEN e0 =>
carga_sig         <= '0';
IF pulso_rx = '1' THEN
      IF datos_in = '0' THEN
      estado_sig     <= e1;
      END IF;
END IF;

WHEN e1 =>
IF pulso_rx = '1' THEN
      IF cont_prt = 7 THEN
      cont_sig       <= "0000";
      estado_sig     <= e2;
      ELSE
      cont_sig       <= cont_prt + 1;
      END IF;
END IF;

WHEN e2 =>
IF pulso_rx = '1' THEN
      IF cont_prt = 15 THEN
      reg_sig        <= datos_in & reg_prt(7 DOWNTO 1);
      cont_sig       <= "0000";
            IF bit_prt = 7 THEN
            bit_sig    <= "000";
            estado_sig <= e3;
            ELSE
            bit_sig    <= bit_prt + 1;
            END IF;
      ELSE
      cont_sig       <= cont_prt + 1;
      END IF;
END IF;

WHEN e3 =>
IF pulso_rx = '1' THEN
      IF cont_prt = "1111" THEN
      carga_sig      <= '1';
      dout_sig       <= reg_prt;
      estado_sig     <= e0;
```

```
      ELSE
      cont_sig       <= cont_prt + 1;
      END IF;
END IF;

END CASE;
END PROCESS;
carga              <= carga_prt;
dout               <= dout_prt;
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- FACTORES DE ROTACIÓN                                                 --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
USE     IEEE.STD_LOGIC_SIGNED.ALL;
-------------------------------------------------------------------------
ENTITY ROFACTOR IS
GENERIC (
      puntos  : NATURAL;
      etapas  : NATURAL);
PORT (
      reloj   : IN  STD_LOGIC;
      reset   : IN  STD_LOGIC;
      start   : IN  STD_LOGIC;
      inversa : IN  STD_LOGIC;
      angulo  : OUT STD_LOGIC_VECTOR(2*etapas-1 DOWNTO 0));
END ENTITY ROFACTOR;
-------------------------------------------------------------------------
ARCHITECTURE FACTORES_DE_ROTACION OF ROFACTOR IS
SIGNAL
contador  : STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0) := (OTHERS => '0');
SIGNAL
inc       : STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0) := (OTHERS => '0');
SIGNAL
iinc      : STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0) := (OTHERS => '0');
SIGNAL
fase      : STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0) := (OTHERS => '0');
SIGNAL
mascara   : STD_LOGIC_VECTOR(etapas*2-1 DOWNTO 0) := (OTHERS => '0');
BEGIN
angulo <= fase;

CONTEO : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
contador        <= (OTHERS => '0');
inc             <= (OTHERS => '0');
mascara         <= (OTHERS => '0');
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF start = '1' THEN
```

```
      contador    <= (OTHERS => '0');
      mascara     <= (OTHERS => '0');
            IF inversa = '1' THEN
            inc     <= CONV_STD_LOGIC_VECTOR( 1, etapas*2);
            ELSE
            inc     <= CONV_STD_LOGIC_VECTOR(-1, etapas*2);
            END IF;
      ELSE
      contador    <= UNSIGNED(contador) + 1;
            IF SIGNED(contador) = -1 THEN
            inc     <= inc(etapas*2-3 DOWNTO 0) & "00";
            mascara <= "11" & mascara(etapas*2-1 DOWNTO 2);
            END IF;
      END IF;
END IF;
END PROCESS CONTEO;

SALIDA : PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
fase     <= (OTHERS => '0');
iinc     <= (OTHERS => '0');
ELSIF reloj'EVENT AND reloj = '1' THEN
      IF start = '1' THEN
      iinc <= (OTHERS => '0');
      fase <= (OTHERS => '0');
      ELSE
            IF UNSIGNED(contador(1 DOWNTO 0)) = 3 THEN
            fase <= (OTHERS => '0');
            ELSE
            fase <= UNSIGNED(fase) + UNSIGNED(iinc);
            END IF;
            IF SIGNED(contador OR mascara) = -1 THEN
            iinc <= (OTHERS => '0');
            ELSIF UNSIGNED(contador(1 DOWNTO 0)) = 3 THEN
            iinc <= UNSIGNED(iinc) + UNSIGNED(inc);
            END IF;
      END IF;
END IF;
END PROCESS SALIDA;
END ARCHITECTURE FACTORES_DE_ROTACION;

--------------------------------------------------------------------
-- SUBSISTEMA DE RECEPCIÓN                                        --
--------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.NUMERIC_STD.ALL;
--------------------------------------------------------------------
ENTITY RX_MINI_SISTEMA IS
GENERIC (
      velocidad   : INTEGER :=   128000;
      frecuencia  : INTEGER := 50000000);
PORT (
```

```
      reset       : IN  STD_LOGIC;
      reloj       : IN  STD_LOGIC;
      serial_in   : IN  STD_LOGIC;
      rx_ok       : OUT STD_LOGIC;
      datos_out   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ENTITY RX_MINI_SISTEMA;
-------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF RX_MINI_SISTEMA IS
SIGNAL pulso_rx : STD_LOGIC;

COMPONENT GEN_PULSO_RX
GENERIC (max    : INTEGER := frecuencia/(velocidad*16)); -- (24.414)
PORT (
      reloj       : IN  STD_LOGIC;
      pulso_rx    : OUT STD_LOGIC);
END COMPONENT;

COMPONENT RECEPTOR
PORT (
      reloj       : IN  STD_LOGIC;
      reset       : IN  STD_LOGIC;
      datos_in    : IN  STD_LOGIC;
      pulso_rx    : IN  STD_LOGIC;
      carga       : OUT STD_LOGIC;
      dout        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

BEGIN

UNIDAD_RX1 : GEN_PULSO_RX
PORT MAP (
      reloj            => reloj,
      pulso_rx         => pulso_rx);

UNIDAD_RX2 : RECEPTOR
PORT MAP (
      reloj            => reloj,
      datos_in         => serial_in,
      reset            => reset,
      pulso_rx         => pulso_rx,
      dout(7 DOWNTO 0) => datos_out,
      carga            => rx_ok);
END ARCHITECTURE COMPORTAMIENTO;

-------------------------------------------------------------------------
-- MULTIPLICADOR MEDIANTE EL ALGORITMO DE VOLDER                       --
-------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_SIGNED.ALL;
-------------------------------------------------------------------------
ENTITY SC_CORPROC IS
GENERIC (
```

```
      ancho       : NATURAL;
      etapas      : NATURAL);
PORT (
      reloj       : IN  STD_LOGIC;
      habilitador : IN  STD_LOGIC;
      xin         : IN  SIGNED (   ancho+1 DOWNTO 0);
      yin         : IN  SIGNED (   ancho+1 DOWNTO 0);
      ain         : IN  SIGNED (2*etapas-3 DOWNTO 0);
      seno        : OUT SIGNED (   ancho+3 DOWNTO 0);
      coseno      : OUT SIGNED (   ancho+3 DOWNTO 0));
END ENTITY SC_CORPROC;
-------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF SC_CORPROC IS
CONSTANT longitud_pipe : NATURAL := 2*etapas + 2;

COMPONENT P2R_CORDIC IS
GENERIC (
      pipeline    : INTEGER := 15;
      ancho       : INTEGER := 16);
PORT (
      reloj       : IN  STD_LOGIC;
      habilitador : IN  STD_LOGIC;
      xi          : IN  SIGNED (ancho-1 DOWNTO 0);
      yi          : IN  SIGNED (ancho-1 DOWNTO 0) := (OTHERS => '0');
      Zi          : IN  SIGNED (     19 DOWNTO 0);
      xo          : OUT SIGNED (ancho-1 DOWNTO 0);
      yo          : OUT SIGNED (ancho-1 DOWNTO 0));
END COMPONENT P2R_CORDIC;

SIGNAL fase     : SIGNED (         19 DOWNTO 0);
SIGNAL xi       : SIGNED (    ancho+7 DOWNTO 0);
SIGNAL yi       : SIGNED (    ancho+7 DOWNTO 0);
SIGNAL xo       : SIGNED (    ancho+7 DOWNTO 0);
SIGNAL yo       : SIGNED (    ancho+7 DOWNTO 0);
SIGNAL ceros    : SIGNED (19-etapas*2 DOWNTO 0);
BEGIN
      xi          <= xin(ancho+1) & xin & "00000";
      yi          <= yin(ancho+1) & yin & "00000";
      ceros       <= (OTHERS => '0');
      fase        <= "00" & ain & ceros;
      coseno      <= xo(ancho+7) & xo(ancho+7 DOWNTO 5);
      seno        <= yo(ancho+7) & yo(ancho+7 DOWNTO 5);

UNIDAD_1 : P2R_CORDIC
GENERIC MAP (
      pipeline    => longitud_pipe,
      ancho       => ancho + 8)
PORT MAP (
      reloj       => reloj,
      habilitador => habilitador,
      xi          => xi,
      yi          => yi,
      Zi          => fase,
      xo          => xo,
```

```
      yo          => yo);
END ARCHITECTURE COMPORTAMIENTO;

--------------------------------------------------------------------------
-- MEMORIA ROM QUE CONTIENE UNA SEÑAL SINUSOIDAL                        --
--------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------------------------------------------------
ENTITY SENO_MEM_ROM_SUBSISTEMA IS
PORT (
      reloj  : IN  STD_LOGIC;
      dout : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ENTITY SENO_MEM_ROM_SUBSISTEMA;
--------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF SENO_MEM_ROM_SUBSISTEMA IS
SIGNAL dir_prt   : INTEGER := 0;
SIGNAL dir_sig   : INTEGER := 0;
TYPE ROM IS ARRAY(0 TO 63) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
CONSTANT sine : ROM := (
      CONV_STD_LOGIC_VECTOR  (64,8),
      CONV_STD_LOGIC_VECTOR  (70,8),
      CONV_STD_LOGIC_VECTOR  (76,8),
      CONV_STD_LOGIC_VECTOR  (82,8),
      CONV_STD_LOGIC_VECTOR  (88,8),
      CONV_STD_LOGIC_VECTOR  (94,8),
      CONV_STD_LOGIC_VECTOR (100,8),
      CONV_STD_LOGIC_VECTOR (105,8),
      CONV_STD_LOGIC_VECTOR (109,8),
      CONV_STD_LOGIC_VECTOR (114,8),
      CONV_STD_LOGIC_VECTOR (117,8),
      CONV_STD_LOGIC_VECTOR (120,8),
      CONV_STD_LOGIC_VECTOR (123,8),
      CONV_STD_LOGIC_VECTOR (125,8),
      CONV_STD_LOGIC_VECTOR (127,8),
      CONV_STD_LOGIC_VECTOR (127,8),
      CONV_STD_LOGIC_VECTOR (127,8),
      CONV_STD_LOGIC_VECTOR (127,8),
      CONV_STD_LOGIC_VECTOR (126,8),
      CONV_STD_LOGIC_VECTOR (124,8),
      CONV_STD_LOGIC_VECTOR (122,8),
      CONV_STD_LOGIC_VECTOR (119,8),
      CONV_STD_LOGIC_VECTOR (115,8),
      CONV_STD_LOGIC_VECTOR (111,8),
      CONV_STD_LOGIC_VECTOR (107,8),
      CONV_STD_LOGIC_VECTOR (102,8),
      CONV_STD_LOGIC_VECTOR  (97,8),
      CONV_STD_LOGIC_VECTOR  (91,8),
      CONV_STD_LOGIC_VECTOR  (85,8),
      CONV_STD_LOGIC_VECTOR  (79,8),
      CONV_STD_LOGIC_VECTOR  (73,8),
      CONV_STD_LOGIC_VECTOR  (67,8),
```

```
      CONV_STD_LOGIC_VECTOR  (60,8),
      CONV_STD_LOGIC_VECTOR  (54,8),
      CONV_STD_LOGIC_VECTOR  (48,8),
      CONV_STD_LOGIC_VECTOR  (42,8),
      CONV_STD_LOGIC_VECTOR  (36,8),
      CONV_STD_LOGIC_VECTOR  (30,8),
      CONV_STD_LOGIC_VECTOR  (25,8),
      CONV_STD_LOGIC_VECTOR  (20,8),
      CONV_STD_LOGIC_VECTOR  (16,8),
      CONV_STD_LOGIC_VECTOR  (12,8),
      CONV_STD_LOGIC_VECTOR   (8,8),
      CONV_STD_LOGIC_VECTOR   (5,8),
      CONV_STD_LOGIC_VECTOR   (3,8),
      CONV_STD_LOGIC_VECTOR   (1,8),
      CONV_STD_LOGIC_VECTOR   (0,8),
      CONV_STD_LOGIC_VECTOR   (0,8),
      CONV_STD_LOGIC_VECTOR   (0,8),
      CONV_STD_LOGIC_VECTOR   (0,8),
      CONV_STD_LOGIC_VECTOR   (2,8),
      CONV_STD_LOGIC_VECTOR   (4,8),
      CONV_STD_LOGIC_VECTOR   (7,8),
      CONV_STD_LOGIC_VECTOR  (10,8),
      CONV_STD_LOGIC_VECTOR  (13,8),
      CONV_STD_LOGIC_VECTOR  (18,8),
      CONV_STD_LOGIC_VECTOR  (22,8),
      CONV_STD_LOGIC_VECTOR  (27,8),
      CONV_STD_LOGIC_VECTOR  (33,8),
      CONV_STD_LOGIC_VECTOR  (39,8),
      CONV_STD_LOGIC_VECTOR  (45,8),
      CONV_STD_LOGIC_VECTOR  (51,8),
      CONV_STD_LOGIC_VECTOR  (57,8),
      CONV_STD_LOGIC_VECTOR  (63,8));
BEGIN
PROCESS (reloj)
BEGIN
IF RISING_EDGE (reloj) THEN
dir_prt <= dir_sig;
END IF;
END PROCESS;
dir_sig <= 0 WHEN dir_prt = 63 ELSE dir_prt + 1;
dout   <= Sine(dir_prt);
END ARCHITECTURE COMPORTAMIENTO;

--------------------------------------------------------------------
-- TRANSMISOR                                                     --
--------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.STD_LOGIC_ARITH.ALL;
USE     IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------------------------------------------
ENTITY TRANSMISOR IS
PORT (
      reloj       : IN  STD_LOGIC;
```

```
      reset       : IN  STD_LOGIC;
      pulso_tx    : IN  STD_LOGIC;
      start       : IN  STD_LOGIC;
      din         : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      ok          : OUT STD_LOGIC;
      dout        : OUT STD_LOGIC);
END ENTITY TRANSMISOR;
--------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF TRANSMISOR IS
TYPE ESTADO IS(e0, e1);
SIGNAL estado_prt  : ESTADO                       := e0;
SIGNAL estado_sig  : ESTADO                       := e0;
SIGNAL reg_prt     : STD_LOGIC_VECTOR(8 DOWNTO 0) := (OTHERS => '0');
SIGNAL reg_sig     : STD_LOGIC_VECTOR(8 DOWNTO 0) := (OTHERS => '0');
SIGNAL cont_prt    : STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');
SIGNAL cont_sig    : STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');
SIGNAL tx_prt      : STD_LOGIC                    := '0';
SIGNAL tx_sig      : STD_LOGIC                    := '0';
BEGIN

PROCESS(reloj, reset)
BEGIN
IF reset = '1' THEN
cont_prt       <= "0000";
estado_prt     <= e0;
tx_prt         <= '1';
ELSIF RISING_EDGE(reloj) THEN
tx_prt         <= tx_sig;
reg_prt        <= reg_sig;
cont_prt       <= cont_sig;
estado_prt     <= estado_sig;
END IF;
END PROCESS;

PROCESS(estado_prt, pulso_tx, cont_prt, start, reg_prt, tx_prt)
BEGIN
ok             <= '0';
tx_sig         <= tx_prt;
reg_sig        <= reg_prt;
cont_sig       <= cont_prt;
estado_sig     <= estado_prt;
CASE estado_prt IS
WHEN e0 =>
IF start = '1' THEN
reg_sig        <= din & '0';
estado_sig     <= e1;
END IF;
WHEN e1 =>
IF pulso_tx = '1' THEN
      IF cont_prt = 9 THEN
      ok         <= '1';
      tx_sig     <= '1';
      cont_sig   <= "0000";
      estado_sig <= e0;
```

```
      ELSE
      tx_sig     <= reg_prt(0);
      reg_sig    <= '0' & reg_prt(8 DOWNTO 1);
      cont_sig   <= cont_prt + "0001";
      END IF;
END IF;
END CASE;
END PROCESS;
dout           <= tx_prt;
END ARCHITECTURE COMPORTAMIENTO;
--------------------------------------------------------------------------
-- SUBSISTEMA DE TRANSMISIÓN                                             --
--------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.NUMERIC_STD.ALL;
--------------------------------------------------------------------------
ENTITY TX_MINI_SISTEMA IS
GENERIC (
      velocidad   : INTEGER :=   128000;
      frecuencia  : INTEGER := 50000000);
PORT (
      reloj       : IN  STD_LOGIC;
      reset       : IN  STD_LOGIC;
      start       : IN  STD_LOGIC;
      datos_tx    : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      ok          : OUT STD_LOGIC;
      serial_out  : OUT STD_LOGIC);
END ENTITY TX_MINI_SISTEMA;
--------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF TX_MINI_SISTEMA IS
SIGNAL pulso_tx : STD_LOGIC;

COMPONENT TRANSMISOR
PORT (
      reloj       : IN  STD_LOGIC;
      reset       : IN  STD_LOGIC;
      pulso_tx    : IN  STD_LOGIC;
      start       : IN  STD_LOGIC;
      din         : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      ok          : OUT STD_LOGIC;
      dout        : OUT STD_LOGIC);
END COMPONENT;

COMPONENT GEN_PULSO_TX
GENERIC(max    : INTEGER := frecuencia/velocidad); -- (390.625)
PORT (
      reloj      : IN  STD_LOGIC;
      pulso_tx   : OUT STD_LOGIC);
END COMPONENT;

BEGIN

UNIDAD_TX1 : TRANSMISOR
```

```
PORT MAP (
      reloj           => reloj,
      din(7 DOWNTO 0) => datos_tx(7 DOWNTO 0),
      reset           => reset,
      start           => start,
      pulso_tx        => pulso_tx,
      ok              => ok,
      dout            => serial_out);

UNIDAD_TX2 : GEN_PULSO_TX
PORT MAP (
      reloj           => reloj,
      pulso_tx        => pulso_tx);
END ARCHITECTURE COMPORTAMIENTO;

------------------------------------------------------------------------
-- SUBSISTEMA DE COMUNICACIÓN SERIAL ASÍNCRONA                         --
------------------------------------------------------------------------
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.NUMERIC_STD.ALL;
------------------------------------------------------------------------
ENTITY UART_SUBSISTEMA IS
GENERIC (
      velocidad   : INTEGER :=   128000;  -- En Baudios.
      frecuencia  : INTEGER := 50000000); -- En [Hz].
PORT (
      reloj       : IN  STD_LOGIC;
      reset       : IN  STD_LOGIC;
      serial_in   : IN  STD_LOGIC;
      start_tx    : IN  STD_LOGIC;
      datos_tx    : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
      rx_ok       : OUT STD_LOGIC;
      tx_ok       : OUT STD_LOGIC;
      datos_rx    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      serial_out  : OUT STD_LOGIC);
END ENTITY UART_SUBSISTEMA;
------------------------------------------------------------------------
ARCHITECTURE COMPORTAMIENTO OF UART_SUBSISTEMA IS
SIGNAL temporal : STD_LOGIC_VECTOR(7 DOWNTO 0);

COMPONENT RX_MINI_SISTEMA
GENERIC (
      velocidad   : INTEGER := velocidad;
      frecuencia  : INTEGER := frecuencia);
PORT (
      reset       : IN  STD_LOGIC;
      reloj       : IN  STD_LOGIC;
      serial_in   : IN  STD_LOGIC;
      rx_ok       : OUT STD_LOGIC;
      datos_out   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT;

COMPONENT TX_MINI_SISTEMA
```

```
GENERIC (
      velocidad   : INTEGER := velocidad;
      frecuencia  : INTEGER := frecuencia);
PORT (
      reloj       : IN  STD_LOGIC;
      reset       : IN  STD_LOGIC;
      start       : IN  STD_LOGIC;
      datos_tx    : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
      ok          : OUT STD_LOGIC;
      serial_out  : OUT STD_LOGIC);
END COMPONENT;

BEGIN

RECEPTOR : RX_MINI_SISTEMA
PORT MAP (
      reloj                => reloj,
      reset                => reset,
      serial_in            => serial_in,
      rx_ok                => rx_ok,
      datos_out(7 DOWNTO 0) => datos_rx(7 DOWNTO 0));

TRANSMISOR : TX_MINI_SISTEMA
PORT MAP (
      reloj                => reloj,
      reset                => '0',
      start                => start_tx,
      datos_tx(7 DOWNTO 0)  => temporal,
      ok                   => tx_ok,
      serial_out           => serial_out);
      temporal             <= datos_tx(9 DOWNTO 2);
END ARCHITECTURE COMPORTAMIENTO;
```

Anexo 2 código fuente en lenguaje M

```
%--------------------------------------------------------------------%
%- SISTEMA DE CONTROL DE LA INTERFAZ GRAFICA DE USUARIO             -%
%--------------------------------------------------------------------%

function varargout = principal(varargin)
% Código de MATLAB para principal.fig
%      PRINCIPAL, por sí mismo, crea un nuevo PRINCIPAL o
%      replantea el vigente singleton*.
%
%      H = PRINCIPAL devuelve el identificador de un nuevo
%      PRINCIPAL o el manejador del existente singleton*.
%
%      PRINCIPAL('CALLBACK',hObject,eventData,handles,...) llama a
%      la función local llamada CALLBACK en PRINCIPAL.M con los
%      argumentos de entrada dados.
%
%      PRINCIPAL ('Propiedad', 'Valor', ...) crea un nuevo
%      PRINCIPAL o replantea el vigente singleton*. A partir de la
%      izquierda, los pares de valores de propiedad se aplican a
%      la interfaz gráfica de usuario antes de que
%      principal_OpeningFcn sea llamada. Un nombre de propiedad
%      no reconocido o un valor no válido detiene la aplicación
%      de propiedad. Todas las entradas pasan por
%      principal_OpeningFcn a través de varargin.
%
%      * Ver Opciones de interfaz gráfica de usuario en el menú
%      Herramientas de GUIDE. Elija "GUI permite sólo una
%      instancia para correr (singleton)".
%
%      Consulte también: GUIDE, GUIDATA, GUIHANDLES


% Comienza código de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @principal_OpeningFcn, ...
                   'gui_OutputFcn',  @principal_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% Termina código de inicialización - NO EDITAR


% --- Se ejecuta justo antes que principal se hace visible.
```

```
function principal_OpeningFcn(hObject, eventdata, handles, varargin)

% Declaración de variables globales.
global      mx; % Valor presente en le generador de radio frecuencias
global      b1; %    5
global      b2; %   10
global      b3; % 12.5
global      b4; %   15
global      b5; %   20
global      b6; %   25
global     ini;
global     fin;
global     ADF;
global  referencia; % ...
global  buffer; % 1024
global ventana;
global rep_ind;

% Declaración de variables.
b1      =     5;
b2      =    10;
b3      =  12.5;
b4      =    15;
b5      =    20;
b6      =    25;
uni     = 'MHz';
buffer  =  1024;
ADF     =    mx;
rep_ind =     1;
referencia  =     0; % en (dB/div)

% Ocultando botones de la barra de herramientas.
barra = findall(gcf);
hb1   = findall(barra,'ToolTipString','Show Plot Tools and Dock Figure');
set(hb1,'Visible','Off');
hb2   = findall(barra,'ToolTipString','Hide Plot Tools');
set(hb2,'Visible','Off');
hb3   = findall(barra,'ToolTipString','Insert Legend');
set(hb3,'Visible','Off');
hb4   = findall(barra,'ToolTipString','Link Plot');
set(hb4,'Visible','Off');
hb5   = findall(barra,'ToolTipString','Rotate 3D');
set(hb5,'Visible','Off');
hb6   = findall(barra,'ToolTipString','Edit Plot');
set(hb6,'Visible','Off');
hb7   = findall(barra,'ToolTipString','Open File');
set(hb7,'Visible','Off');
hb8   = findall(barra,'ToolTipString','New Figure');
set(hb8,'Visible','Off');
hb9   = findall(barra,'ToolTipString','Save Figure');
set(hb9,'Visible','Off');
hb10  = findall(barra,'ToolTipString','Insert Colorbar');
set(hb10,'Visible','Off');
hb9   = findall(barra,'ToolTipString','Brush/Select Data');
```

```
set(hb9,'Visible','Off');

% Asignación de Nombres a los botones.
nbot1ini = num2str(mx);
nbot1fin = num2str(mx+b1);
nbot1    = [nbot1ini,' - ',nbot1fin,' ',uni];
nbot2ini = num2str(mx+b1);
nbot2fin = num2str(mx+b2);
nbot2    = [nbot2ini,' - ',nbot2fin,' ',uni];
nbot3ini = num2str(mx+b2);
nbot3fin = num2str(mx+b4);
nbot3    = [nbot3ini,' - ',nbot3fin,' ',uni];
nbot4ini = num2str(mx+b4);
nbot4fin = num2str(mx+b5);
nbot4    = [nbot4ini,' - ',nbot4fin,' ',uni];
nbot5ini = num2str(mx+b5);
nbot5fin = num2str(mx+b6);
nbot5    = [nbot5ini,' - ',nbot5fin,' ',uni];
nbot6ini = num2str(mx);
nbot6fin = num2str(mx+b3);
nbot6    = [nbot6ini,' - ',nbot6fin,' ',uni];
nbot7ini = num2str(mx+b3);
nbot7fin = num2str(mx+b6);
nbot7    = [nbot7ini,' - ',nbot7fin,' ',uni];
set(handles.cincoM1,'string',nbot1);
set(handles.cincoM2,'string',nbot2);
set(handles.cincoM3,'string',nbot3);
set(handles.cincoM4,'string',nbot4);
set(handles.cincoM5,'string',nbot5);
set(handles.doceM1,'string',nbot6);
set(handles.doceM2,'string',nbot7);

% Fijando el fondo de pantalla principal
fondo_principal = imread('fondo.jpg');
axes(handles.axfondo)
image(fondo_principal);
axis off;

% Configurando el área de trazado de la grafica.
set(handles.pantalla,'vis','on')
axes(handles.pantalla)
grid on;
hold on
ventana = b6;
ini     = 1;
fin     = (buffer/2);
anspec
handles.output = hObject;
guidata(hObject, handles);


% --- Las salidas de esta función se devuelven a la línea de
%     comandos.
function varargout = principal_OutputFcn(hObject, eventdata, handles)
```

```
varargout{1} = handles.output;


% --- Se ejecuta durante la creación del objeto, después de ajustar
%     todas las propiedades.
function principal_CreateFcn(hObject, eventdata, handles)


% --- Se ejecuta cuando se pulse el botón cincoM1
function cincoM1_Callback(hObject, eventdata, handles)
global      mx;
global      b1;
global      b6;
global     ini;
global     fin;
global     ADF;
global  referencia; % ...
global  buffer;
global ventana;
global rep_ind;
ADF     = mx;
ventana = b1;
rep_ind =  1;
ini     =  1;
fin     = floor(b1*((buffer/2)/b6)) + 1;
anspec


% --- Se ejecuta cuando se pulse el botón cincoM2
function cincoM2_Callback(hObject, eventdata, handles)
global      mx;
global      b1;
global      b6;
global     ini;
global     fin;
global  referencia; % ...
global     ADF;
global  buffer;
global ventana;
global rep_ind;
ADF     = mx + b1;
ventana = b1;
rep_ind =  1;
ini     =   (floor(b1*((buffer/2)/b6))) + 1;
fin     = 2*(floor(b1*((buffer/2)/b6))) + 1;
anspec


% --- Se ejecuta cuando se pulse el botón cincoM3
function cincoM3_Callback(hObject, eventdata, handles)
global      mx;
global      b1;
global      b2;
global      b6;
```

```
global     ini;
global     fin;
global     ADF;
global  referencia; % ...
global  buffer;
global ventana;
global rep_ind;
ADF     = mx + b2;
ventana = b1;
rep_ind =  1;
ini     = 2*floor(b1*((buffer/2)/b6)) + 2;
fin     = 3*floor(b1*((buffer/2)/b6)) + 2;
anspec


% --- Se ejecuta cuando se pulse el botón cincoM4
function cincoM4_Callback(hObject, eventdata, handles)
global      mx;
global      b1;
global      b4;
global      b6;
global     ini;
global     fin;
global     ADF;
global  referencia; % ...
global  buffer;
global ventana;
global rep_ind;
ADF     = mx + b4;
ventana = b1;
rep_ind =  1;
ini     = 3*floor(b1*((buffer/2)/b6)) + 3;
fin     = 4*floor(b1*((buffer/2)/b6)) + 3;
anspec


% --- Se ejecuta cuando se pulse el botón cincoM5
function cincoM5_Callback(hObject, eventdata, handles)
global      mx;
global      b1;
global      b5;
global      b6;
global     ini;
global     fin;
global     ADF;
global  referencia; % ...
global  buffer;
global ventana;
global rep_ind;
ADF     = mx + b5;
ventana = b1;
rep_ind =  1;
ini     = 4*floor(b1*((buffer/2)/b6)) + 2;
fin     = 5*floor(b1*((buffer/2)/b6)) + 2;
```

```
anspec


% --- Se ejecuta cuando se pulse el botón doceM1
function doceM1_Callback(hObject, eventdata, handles)
global      mx;
global      b3;
global      b6;
global     ini;
global     fin;
global     ADF;
global  referencia; % ...
global  buffer;
global ventana;
global rep_ind;
ADF     = mx;
ventana = b3;
rep_ind =  1;
ini     =  1;
fin     = floor(b3*((buffer/2)/b6));
anspec


% --- Se ejecuta cuando se pulse el botón doceM2
function doceM2_Callback(hObject, eventdata, handles)
global      mx;
global      b3;
global      b6;
global     ini;
global     fin;
global     ADF;
global  referencia; % ...
global  buffer;
global ventana;
global rep_ind;
ADF     = mx + b3;
ventana = b3;
rep_ind =  1;
ini     =   floor(b3*((buffer/2)/b6)) + 1;
fin     = 2*floor(b3*((buffer/2)/b6));
anspec

% --- Se ejecuta cuando se pulse el botón "Banda completa"
function vcincoM_Callback(hObject, eventdata, handles)
global      mx;
global      b6;
global     ini;
global     fin;
global     ADF;
global  referencia; % ...
global  buffer;
global ventana;
global rep_ind;
ADF     = mx;
```

```
ventana = b6;
rep_ind =  1;
ini     =  1;
fin     = buffer/2 - 1;
anspec;


% --- Se ejecuta cuando se pulse el botón "Referencia"
function pot_rang_Callback(hObject, eventdata, handles)
global     ini;
global     fin;
global     ADF;
global  referencia;
global  buffer;
global ventana;
global rep_ind;
rep_ind    = 1;
propuesta  = {'Ingrese un valor para la referencia en [dBm]:'};
nombre     = 'ANSPEC v1.0 - Referencia';
lineas     = 1;
pordefecto = {'0'};
respuesta  = inputdlg(propuesta,nombre,lineas,pordefecto);
if isempty(respuesta)
pel1 = warndlg('Referencia NO establecida','ANSPEC v1.0','modal');
uiwait(pel1)
end
temp = str2double(respuesta{1});
if isnan(temp)
beep;
pel2 = warndlg(...
    'Los parámetros deben ser numéricos','ANSPEC v1.0','modal');
uiwait(pel2)
else
referencia = temp;
anspec;
end


% --- Se ejecuta cuando se pulse el botón "Pausa".
function pausa_Callback(hObject, eventdata, handles)
global     ini;
global     fin;
global     ADF;
global  referencia;
global  buffer;
global ventana;
global rep_ind;
rep_ind =    0; % Estado de Pausa
anspec;


% --- Se ejecuta cuando se pulse el botón "Actualizar".
function actualizar_Callback(hObject, eventdata, handles)
principal;
```

```
anspec;

%-------------------------------------------------------------------------%
%- SISTEMA DE ADECUACIÓN Y TRAZADO                                       -%
%-------------------------------------------------------------------------%

% Declaración de variables globales.
global  puerto; % Puerto serial COM?
global fftsx;
global frec;

% Declaración de variables.
velocidad =   128000; % en Baudios
ADC_frec  =       50; % en (MHz)
ancho     =        8; % en bits
negescala =      100;
formato   = 'dd-mm-yyyy HH:MM pm'; % formato del reloj del sistema

% Inicialización del bucle de ejecución.
while rep_ind == 1;
% Sí  rep_ind ==  1 y se borra o comenta la línea del contador de la
% variable rep_ind, el sistema entra en un bucle de lectura infinito.

% Declaracion del Objeto Serial.
anspec_serial = serial(puerto);

% Configuracion apropiada del puerto serial
set(anspec_serial,'BaudRate',velocidad);
set(anspec_serial,'DataBits',ancho);
set(anspec_serial,'Parity','none');
set(anspec_serial,'StopBits',1);
set(anspec_serial,'FlowControl','none');
set(anspec_serial,'Timeout',20);             % Tiempo de espera 20 s.
set(anspec_serial,'InputBufferSize',buffer); % Recibe 1024 datos

% Recepción de datos
anspec_serial.ReadAsyncMode = 'continuous';
ANSPEC_fopen(anspec_serial);          % Llama a ANSPEC_fopen.m
% --> Script personalizado para apertura del puerto, basado en java.
fwrite(anspec_serial,54);             % Señal de realimentación
b = fread(anspec_serial);             % Lectura de la señal
fclose(anspec_serial);                % Cierre del puerto serial
ahora = datestr(now,formato); % actualización del reloj del sistema

% Acoplamiento de lógica a complemento a dos.
%for i = 1:1:length(b)
%if (b(i)>=128)
%b(i) = 127 - b(i);
%else
%b(i) = b(i);
%end;
%end;

% Estabilización del buffer de lectura si existe error por perdida
```

```
% de datos durante la primera ejecución.
if length(b) ~= buffer % (1023)
limpiador;
configini; % Borra el caché y reinicia el prigrama
else
axes(handles.pantalla); % Activacion de la pantalla

% Tratamiento aritmético de la matriz obtenida del ADC.
x1    = b(1:(buffer/2));
x2    = rot90(rot90(b(((buffer/2)+1):(buffer))));
fftsx = (x1 + x2)./2;

fftsx = .62532003.*fftsx;         % Calibración
fftsx = fftsx - (.725*negescala); % Etapa de atenuacion digital

% Sistema de detección de máximos locales.
[potencia,posicion]   = max(fftsx);
[potenciam,posicionm] = max(memo);

% Trazado de la gráfica
frec = (0 + ADF):ADC_frec/(buffer-1):(ADC_frec/2 + ADF);
% 0:50/1023:25 --> Pasos de 0.0489
xlabel('FRECUENCIA [MHz]','FontWeight','bold','FontSize',12,...
      'FontName','Arial','Color',[1 1 1]);
ylabel('POTENCIA [dBm]','FontWeight','bold','FontSize',12,...
      'FontName','Arial','Color',[1 1 1]);
axis([(0+ADF) (ventana+ADF) ((escala-negescala)) (escala)]);
% Tolerancia de 1 punto
grid on;
comet(frec(ini:fin),fftsx(ini:fin));

% Sistema para mostrar puntos críticos en la interfáz.
format bank;
frecuenciapico = frec(posicion);
%frecuenciapico = frecuenciapico * 1000; % En (KHz).
set(handles.potpic,'String',potencia);
set(handles.frecpic,'String',frecuenciapico);

% Sistema para mostrar el reloj del sistema.
frecuenciapico = frec(posicion);
set(handles.reloj_sis,'String',ahora);
end;
% Comentar la línea siguiente para hacer infinito el bucle
% rep_ind = rep_ind + 1;
end;
grid on;
plot(frec(ini:fin),fftsx(ini:fin),'r');

%- SISTEMA DE CONFIGURACIÓN DE PARAMETROS DE INICIALIZACIÓN        -%
%-------------------------------------------------------------------%

function configini
% CONFIGINI permite al usuario ingresar manualmente los parámetros
%     necesarios para el funcionamiento correcto de la interfaz
```

```
%     gráfica de usuario realizada para el proyecto ANSPEC v1.0.
%
%     Esta función no necesita parámetros de entrada.

% DECLARACIÓN DE VARIABLES GLOBALES
global puerto; % Puerto serial COM?
global     mx; % Valor presente en el generador de radio frecuencias

% CREACIÓN DE LA VENTANA DE CONFIGURACIÓN INICIAL
propuesta   = {'Ingrese el número del puerto a utilizar';...
'Ingrese el valor presente en la pantalla del generador de radio...
 frecuencia en (MHz)'};
nombre      = 'ANSPEC v1.0 - Configuración inicial';
lineas      = 1;
pordefecto  = {'1';'0'};
respuesta   = inputdlg(propuesta,nombre,lineas,pordefecto);

% VALIDACIONES NECESARIAS
if isempty(respuesta)
pel1 = warndlg('Configuración inicial incompleta','ANSPEC v1.0',...
    'modal');
uiwait(pel1);
limpiador;
end
if exist('mx','var')
numpuer = str2double(respuesta{1});
mx      = str2double(respuesta{2});
if isnan(mx) || isnan(numpuer)
beep;
err1 = errordlg(...
    'Los parámetros deben ser numéricos','ANSPEC v1.0','modal');
uiwait(err1);
limpiador;
configini;

% RESTRICCIÓN PROPIA DEL SISTEMA, DADA POR LAS RESTRICCIONES DE
% HARDWARE DEL GENERADOR DE RADIOFRECUENCIAS Y POR LOS OBJETIVOS DEL
% PROYECTO
elseif mx ~= 0 && (mx < 400 || mx > 2700)
pel2 = warndlg(...
'El valor presente en el generador de radio frecuencias está por fuera del
rango permitido, asegúrese de cambiar este valor tanto en la ventana de
configuración inicial, así como en el generador de radio frecuencias. Los
valores permitidos están entre 400 y 2700 (MHz)',...
'ANSPEC v1.0','modal');
uiwait(pel2);
limpiador;
configini;
end
if isempty(numpuer) == 0
puerto = ['COM',respuesta{1}];

% LLAMADA AL SISTEMA DE CORRECCÓN DE ERRORES DEL GENERADOR RF
validarf; % Llamada al Script de validación de frecuencias
```

```
end
end
end

%- SISTEMA PERSONALIZADO DE APERTURA DE UN OBJETO SERIAL BASADO EN -%
%- LENGUAJE JAVA                                                   -%
%-------------------------------------------------------------------%

function ANSPEC_fopen(obj)
%ANSPEC_FOPEN Conecta un objeto de puerto serie al dispositivo.
%   (Funcion personalizada para apertura segura de puerto serial
%   basada en la funcion java bajo licencia OpenGL)
%
%   ANSPEC_FOPEN(OBJ) conecta el objeto puerto serie, OBJ, al
%   dispositivo. Obj puede ser una matriz de objetos de puerto serie.
%
%   Sólo un objeto puerto serie con la misma configuración puede
%   estar conectado a un instrumento a la vez. Por ejemplo, sólo un
%   objeto se puede conectar al puerto COM2. Si OBJ se ha conectado
%   correctamente al dispositivo, la propiedad Status de OBJ está
%   configurada para abrirse, de lo contrario la propiedad Status
%   permanece configurada como cerrada.
%
%   Cuando se abre OBJ, los datos permanecen en el buffer de entrada
%   y el buffer de salida se vacía entonces las propiedades
%   bytesAvailable, BytesToOutput, ValuesReceived y ValuesSent se
%     ponen a 0.
%
%   Algunos valores de las propiedades sólo pueden ser verificados
%   después de que se ha realizado la conexión al dispositivo. Los
%   ejemplos incluyen BaudRate, FlowControl y la paridad. Si alguna
%   de estas propiedades se establece en un valor no admitido por el
%   dispositivo, se devolverá un error y el objeto no se puedrá
%   conectar al dispositivo.
%
%   Algunas propiedades son de sólo lectura mientras el objeto puerto
%   serie está abierto (conectado) y se deben configurar antes de
%   usar FANSPEC_OPEN. Los ejemplos incluyen InputBufferSize y
%   OutputBufferSize.
%
%   Se devolverá un error si FOPEN se llama en un objeto de puerto
%   serie que tiene el valor 'Open' de la propiedad Status.
%
%   El orden de bytes del dispositivo se puede especificar con la
%   propiedad ByteOrder de OBJ.
%
%   Si OBJ es una matriz de objetos de puerto serie y uno de los
%   objetos no se puede conectar al dispositivo, los objetos
%   restantes de la matriz se pueden conectar al dispositivo y se
%   mostrará una advertencia.
%
%   Ejemplo:
%       s = serial('COM1');
%       ANSPEC_fopen(s);
```

```
%       fprintf(s, '*IDN?');
%       idn = fscanf(s);
%       fclose(s);
%
%   Vea también SERIAL/FCLOSE.

% Inicializando variables.
errorOccurred = false;
jobject       = igetfield(obj,'jobject');
warnState     = warning('backtrace','off');

% Llame fopen en cada objeto java. Se seguirá ejecutando incluso si
% uno de los objetos no se pudo abrir.
for i = 1 : length(jobject),
try
fopen(jobject(i));
catch aException
errorOccurred = true;
msg           = aException.message;
end
end

% Informa de un error si se da.
if errorOccurred
if length(jobject) == 1
warning(warnState);
close all;
beep;
err1 = errordlg(msg,'ANSPEC v1.0');
uiwait(err1);
configini;
else
pel1warndlg('ANSPEC v1.0:serial:ANSPEC_fopen:invalid');
uiwait(pel1);
configini;
end
end

warning(warnState);

%--------------------------------------------------------------------%
%- SISTEMA DE LIMPIEZA DE DATOS                                     -%
%--------------------------------------------------------------------%

% Limpia los datos y cierra las ventanas inactivas
clear all,close all,clc

%--------------------------------------------------------------------%
%- SISTEMA DE CORRECCÓN DE ERRORES DEL GENERADOR RF                 -%
%--------------------------------------------------------------------%

% Script de validación de frecuencias del generador

if     (mx >=  396 && mx <=  404);
```

```
mx = 400;
elseif (mx >=  496 && mx <=  504);
mx = 500;
elseif (mx >=  596 && mx <=  604);
mx = 600;
elseif (mx >=  696 && mx <=  704);
mx = 700;
elseif (mx >=  796 && mx <=  804);
mx = 800;
elseif (mx >=  896 && mx <=  904);
mx = 900;
elseif (mx >=  996 && mx <= 1004);
mx = 1000;
elseif (mx >= 1096 && mx <= 1104);
mx = 1100;
elseif (mx >= 1196 && mx <= 1204);
mx = 1200;
elseif (mx >= 1296 && mx <= 1304);
mx = 1300;
elseif (mx >= 1396 && mx <= 1404);
mx = 1400;
elseif (mx >= 1496 && mx <= 1504);
mx = 1500;
elseif (mx >= 1596 && mx <= 1604);
mx = 1600;
elseif (mx >= 1696 && mx <= 1704);
mx = 1700;
elseif (mx >= 1796 && mx <= 1804);
mx = 1800;
elseif (mx >= 1896 && mx <= 1904);
mx = 1900;
elseif (mx >= 1996 && mx <= 2004);
mx = 2000;
elseif (mx >= 2096 && mx <= 2104);
mx = 2100;
elseif (mx >= 2196 && mx <= 2204);
mx = 2200;
elseif (mx >= 2296 && mx <= 2304);
mx = 2300;
elseif (mx >= 2396 && mx <= 2404);
mx = 2400;
elseif (mx >= 2496 && mx <= 2504);
mx = 2500;
elseif (mx >= 2596 && mx <= 2604);
mx = 2600;
elseif (mx >= 2696 && mx <= 2704);
mx = 2700;
%--------------------------------------------------------------------
elseif (mx >=  423 && mx <=  432);
mx = 427.27;
elseif (mx >=  523 && mx <=  532);
mx = 527.27;
elseif (mx >=  623 && mx <=  632);
mx = 627.27;
```

```
elseif (mx >=  723 && mx <=  732);
mx = 727.27;
elseif (mx >=  823 && mx <=  832);
mx = 827.27;
elseif (mx >=  923 && mx <=  932);
mx = 927.27;
elseif (mx >= 1023 && mx <= 1032);
mx = 1027.27;
elseif (mx >= 1123 && mx <= 1132);
mx = 1127.27;
elseif (mx >= 1223 && mx <= 1232);
mx = 1227.27;
elseif (mx >= 1323 && mx <= 1332);
mx = 1327.27;
elseif (mx >= 1423 && mx <= 1432);
mx = 1427.27;
elseif (mx >= 1523 && mx <= 1532);
mx = 1527.27;
elseif (mx >= 1623 && mx <= 1632);
mx = 1627.27;
elseif (mx >= 1723 && mx <= 1732);
mx = 1727.27;
elseif (mx >= 1823 && mx <= 1832);
mx = 1827.27;
elseif (mx >= 1923 && mx <= 1932);
mx = 1927.27;
elseif (mx >= 2023 && mx <= 2032);
mx = 2027.27;
elseif (mx >= 2123 && mx <= 2132);
mx = 2127.27;
elseif (mx >= 2223 && mx <= 2232);
mx = 2227.27;
elseif (mx >= 2323 && mx <= 2332);
mx = 2327.27;
elseif (mx >= 2423 && mx <= 2432);
mx = 2427.27;
elseif (mx >= 2523 && mx <= 2532);
mx = 2527.27;
elseif (mx >= 2623 && mx <= 2632);
mx = 2627.27;
%--------------------------------------------------------------------
elseif (mx >=  433 && mx <=  441);
mx = 436.36;
elseif (mx >=  533 && mx <=  541);
mx = 536.36;
elseif (mx >=  633 && mx <=  641);
mx = 636.36;
elseif (mx >=  733 && mx <=  741);
mx = 736.36;
elseif (mx >=  833 && mx <=  841);
mx = 836.36;
elseif (mx >=  933 && mx <=  941);
mx = 936.36;
elseif (mx >= 1033 && mx <= 1041);
```

```
mx = 1036.36;
elseif (mx >= 1133 && mx <= 1141);
mx = 1136.36;
elseif (mx >= 1233 && mx <= 1241);
mx = 1236.36;
elseif (mx >= 1333 && mx <= 1341);
mx = 1336.36;
elseif (mx >= 1433 && mx <= 1441);
mx = 1436.36;
elseif (mx >= 1533 && mx <= 1541);
mx = 1536.36;
elseif (mx >= 1633 && mx <= 1641);
mx = 1636.36;
elseif (mx >= 1733 && mx <= 1741);
mx = 1736.36;
elseif (mx >= 1833 && mx <= 1841);
mx = 1836.36;
elseif (mx >= 1933 && mx <= 1941);
mx = 1936.36;
elseif (mx >= 2033 && mx <= 2041);
mx = 2036.36;
elseif (mx >= 2133 && mx <= 2141);
mx = 2136.36;
elseif (mx >= 2233 && mx <= 2241);
mx = 2236.36;
elseif (mx >= 2333 && mx <= 2341);
mx = 2336.36;
elseif (mx >= 2433 && mx <= 2441);
mx = 2436.36;
elseif (mx >= 2533 && mx <= 2541);
mx = 2536.36;
elseif (mx >= 2633 && mx <= 2641);
mx = 2636.36;
%--------------------------------------------------------------------
elseif (mx >=  442 && mx <=  459);
mx = 445.45;
elseif (mx >=  542 && mx <=  559);
mx = 545.45;
elseif (mx >=  642 && mx <=  659);
mx = 645.45;
elseif (mx >=  742 && mx <=  759);
mx = 745.45;
elseif (mx >=  842 && mx <=  859);
mx = 845.45;
elseif (mx >=  942 && mx <=  959);
mx = 945.45;
elseif (mx >= 1042 && mx <= 1059);
mx = 1045.45;
elseif (mx >= 1142 && mx <= 1159);
mx = 1145.45;
elseif (mx >= 1242 && mx <= 1259);
mx = 1245.45;
elseif (mx >= 1342 && mx <= 1359);
mx = 1345.45;
```

```
elseif (mx >= 1442 && mx <= 1459);
mx = 1445.45;
elseif (mx >= 1542 && mx <= 1559);
mx = 1545.45;
elseif (mx >= 1642 && mx <= 1659);
mx = 1645.45;
elseif (mx >= 1742 && mx <= 1759);
mx = 1745.45;
elseif (mx >= 1842 && mx <= 1859);
mx = 1845.45;
elseif (mx >= 1942 && mx <= 1959);
mx = 1945.45;
elseif (mx >= 2042 && mx <= 2059);
mx = 2045.45;
elseif (mx >= 2142 && mx <= 2159);
mx = 2145.45;
elseif (mx >= 2242 && mx <= 2259);
mx = 2245.45;
elseif (mx >= 2342 && mx <= 2359);
mx = 2345.45;
elseif (mx >= 2442 && mx <= 2459);
mx = 2445.45;
elseif (mx >= 2542 && mx <= 2559);
mx = 2545.45;
elseif (mx >= 2642 && mx <= 2659);
mx = 2645.45;
%-----------------------------------------------------------------
elseif (mx >=  460 && mx <=  468);
mx = 463.63;
elseif (mx >=  560 && mx <=  568);
mx = 563.63;
elseif (mx >=  660 && mx <=  668);
mx = 663.63;
elseif (mx >=  760 && mx <=  768);
mx = 763.63;
elseif (mx >=  860 && mx <=  868);
mx = 863.63;
elseif (mx >=  960 && mx <=  968);
mx = 963.63;
elseif (mx >= 1060 && mx <= 1068);
mx = 1063.63;
elseif (mx >= 1160 && mx <= 1168);
mx = 1163.63;
elseif (mx >= 1260 && mx <= 1268);
mx = 1263.63;
elseif (mx >= 1360 && mx <= 1368);
mx = 1363.63;
elseif (mx >= 1460 && mx <= 1468);
mx = 1463.63;
elseif (mx >= 1560 && mx <= 1568);
mx = 1563.63;
elseif (mx >= 1660 && mx <= 1668);
mx = 1663.63;
elseif (mx >= 1760 && mx <= 1768);
```

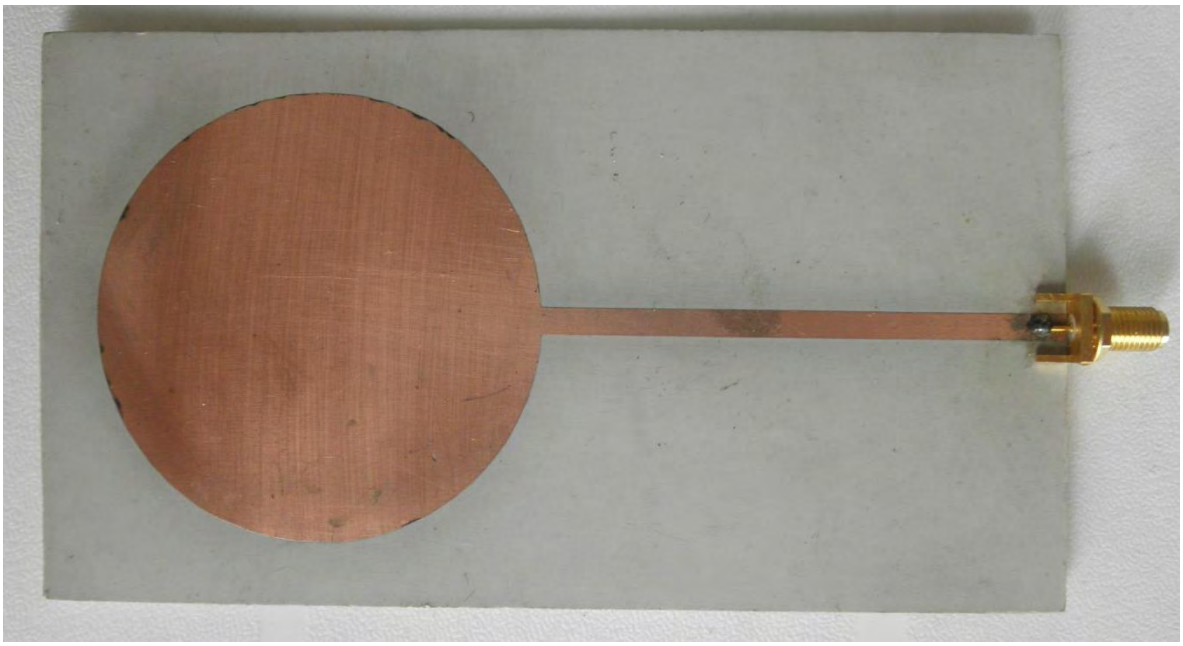
```
mx = 1763.63;
elseif (mx >= 1860 && mx <= 1868);
mx = 1863.63;
elseif (mx >= 1960 && mx <= 1968);
mx = 1963.63;
elseif (mx >= 2060 && mx <= 2068);
mx = 2063.63;
elseif (mx >= 2160 && mx <= 2168);
mx = 2163.63;
elseif (mx >= 2260 && mx <= 2268);
mx = 2263.63;
elseif (mx >= 2360 && mx <= 2368);
mx = 2363.63;
elseif (mx >= 2460 && mx <= 2468);
mx = 2463.63;
elseif (mx >= 2560 && mx <= 2568);
mx = 2563.63;
elseif (mx >= 2660 && mx <= 2668);
mx = 2663.63;
%-----------------------------------------------------------------
elseif (mx >=  469 && mx <=  477);
mx = 472.72;
elseif (mx >=  569 && mx <=  577);
mx = 572.72;
elseif (mx >=  669 && mx <=  677);
mx = 672.72;
elseif (mx >=  769 && mx <=  777);
mx = 772.72;
elseif (mx >=  869 && mx <=  877);
mx = 872.72;
elseif (mx >=  969 && mx <=  977);
mx = 972.72;
elseif (mx >= 1069 && mx <= 1077);
mx = 1072.72;
elseif (mx >= 1169 && mx <= 1177);
mx = 1172.72;
elseif (mx >= 1269 && mx <= 1277);
mx = 1272.72;
elseif (mx >= 1369 && mx <= 1377);
mx = 1372.72;
elseif (mx >= 1469 && mx <= 1477);
mx = 1472.72;
elseif (mx >= 1569 && mx <= 1577);
mx = 1572.72;
elseif (mx >= 1669 && mx <= 1677);
mx = 1672.72;
elseif (mx >= 1769 && mx <= 1777);
mx = 1772.72;
elseif (mx >= 1869 && mx <= 1877);
mx = 1872.72;
elseif (mx >= 1969 && mx <= 1977);
mx = 1972.72;
elseif (mx >= 2069 && mx <= 2077);
mx = 2072.72;
```

```
elseif (mx >= 2169 && mx <= 2177);
mx = 2172.72;
elseif (mx >= 2269 && mx <= 2277);
mx = 2272.72;
elseif (mx >= 2369 && mx <= 2377);
mx = 2372.72;
elseif (mx >= 2469 && mx <= 2477);
mx = 2472.72;
elseif (mx >= 2569 && mx <= 2577);
mx = 2572.72;
elseif (mx >= 2669 && mx <= 2677);
mx = 2672.72;
%-----------------------------------------------------------------
elseif (mx >=  478 && mx <=  486);
mx = 481.81;
elseif (mx >=  578 && mx <=  586);
mx = 581.81;
elseif (mx >=  678 && mx <=  686);
mx = 681.81;
elseif (mx >=  778 && mx <=  786);
mx = 781.81;
elseif (mx >=  878 && mx <=  886);
mx = 881.81;
elseif (mx >=  978 && mx <=  986);
mx = 981.81;
elseif (mx >= 1078 && mx <= 1086);
mx = 1081.81;
elseif (mx >= 1178 && mx <= 1186);
mx = 1181.81;
elseif (mx >= 1278 && mx <= 1286);
mx = 1281.81;
elseif (mx >= 1378 && mx <= 1386);
mx = 1381.81;
elseif (mx >= 1478 && mx <= 1486);
mx = 1481.81;
elseif (mx >= 1578 && mx <= 1586);
mx = 1581.81;
elseif (mx >= 1678 && mx <= 1686);
mx = 1681.81;
elseif (mx >= 1778 && mx <= 1786);
mx = 1781.81;
elseif (mx >= 1878 && mx <= 1886);
mx = 1881.81;
elseif (mx >= 1978 && mx <= 1986);
mx = 1981.81;
elseif (mx >= 2078 && mx <= 2086);
mx = 2081.81;
elseif (mx >= 2178 && mx <= 2186);
mx = 2181.81;
elseif (mx >= 2278 && mx <= 2286);
mx = 2281.81;
elseif (mx >= 2378 && mx <= 2386);
mx = 2381.81;
elseif (mx >= 2478 && mx <= 2486);
```

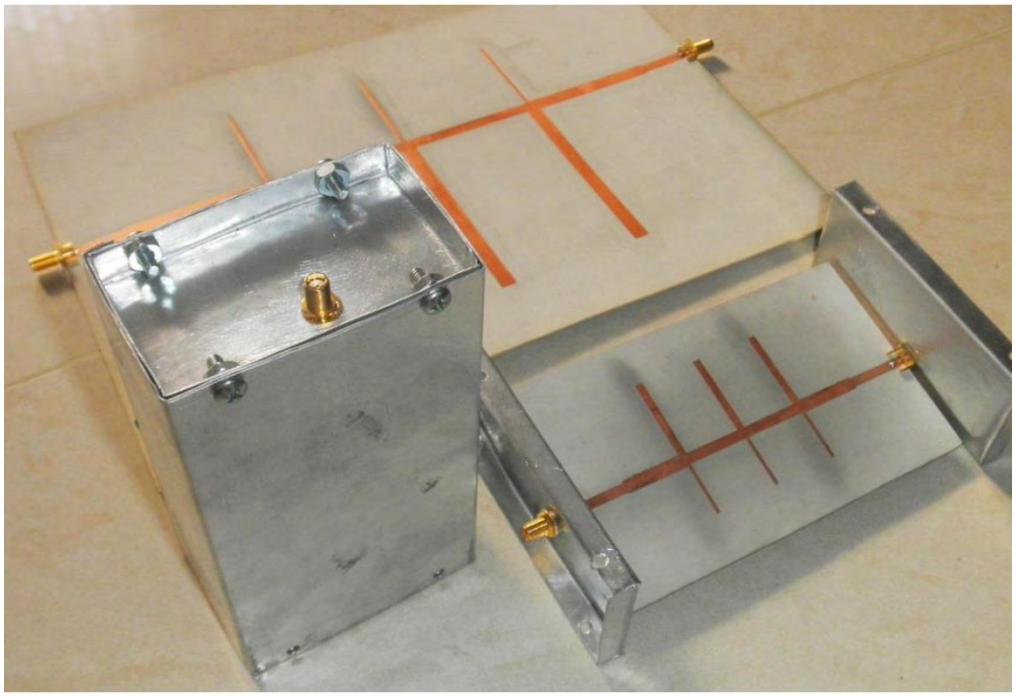
```
mx = 2481.81;
elseif (mx >= 2578 && mx <= 2586);
mx = 2581.81;
elseif (mx >= 2678 && mx <= 2686);
mx = 2681.81;
%-----------------------------------------------------------------
elseif (mx >=  487 && mx <=  495);
mx = 490.91;
elseif (mx >=  587 && mx <=  595);
mx = 590.91;
elseif (mx >=  687 && mx <=  695);
mx = 690.91;
elseif (mx >=  787 && mx <=  795);
mx = 790.91;
elseif (mx >=  887 && mx <=  895);
mx = 890.91;
elseif (mx >=  987 && mx <=  995);
mx = 990.91;
elseif (mx >= 1087 && mx <= 1095);
mx = 1090.91;
elseif (mx >= 1187 && mx <= 1195);
mx = 1190.91;
elseif (mx >= 1287 && mx <= 1295);
mx = 1290.91;
elseif (mx >= 1387 && mx <= 1395);
mx = 1390.91;
elseif (mx >= 1487 && mx <= 1495);
mx = 1490.91;
elseif (mx >= 1587 && mx <= 1595);
mx = 1590.91;
elseif (mx >= 1687 && mx <= 1695);
mx = 1690.91;
elseif (mx >= 1787 && mx <= 1795);
mx = 1790.91;
elseif (mx >= 1887 && mx <= 1895);
mx = 1890.91;
elseif (mx >= 1987 && mx <= 1995);
mx = 1990.91;
elseif (mx >= 2087 && mx <= 2095);
mx = 2090.91;
elseif (mx >= 2187 && mx <= 2195);
mx = 2190.91;
elseif (mx >= 2287 && mx <= 2295);
mx = 2290.91;
elseif (mx >= 2387 && mx <= 2395);
mx = 2390.91;
elseif (mx >= 2487 && mx <= 2495);
mx = 2490.91;
elseif (mx >= 2587 && mx <= 2595);
mx = 2590.91;
elseif (mx >= 2687 && mx <= 2695);
mx = 2690.91;
end
principal;
```

Anexo 3 fotografías

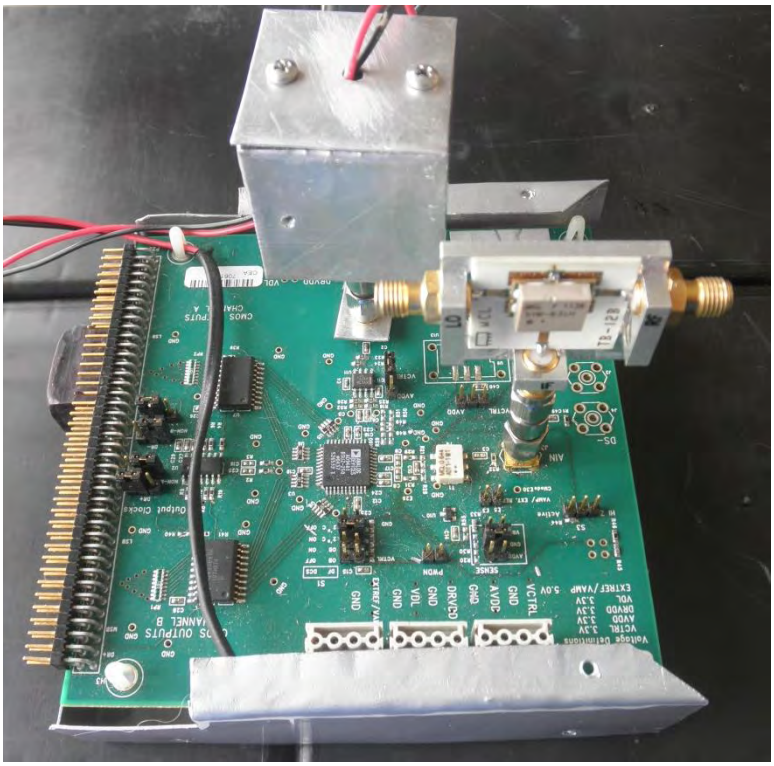
Antena tipo parche circular



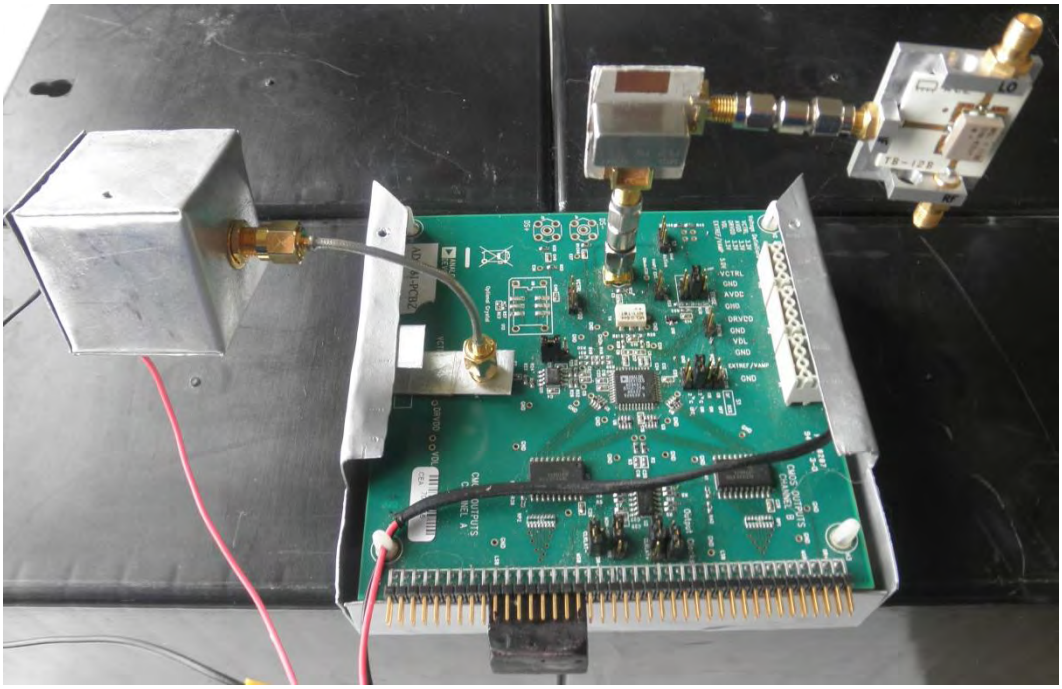
Filtros DBR



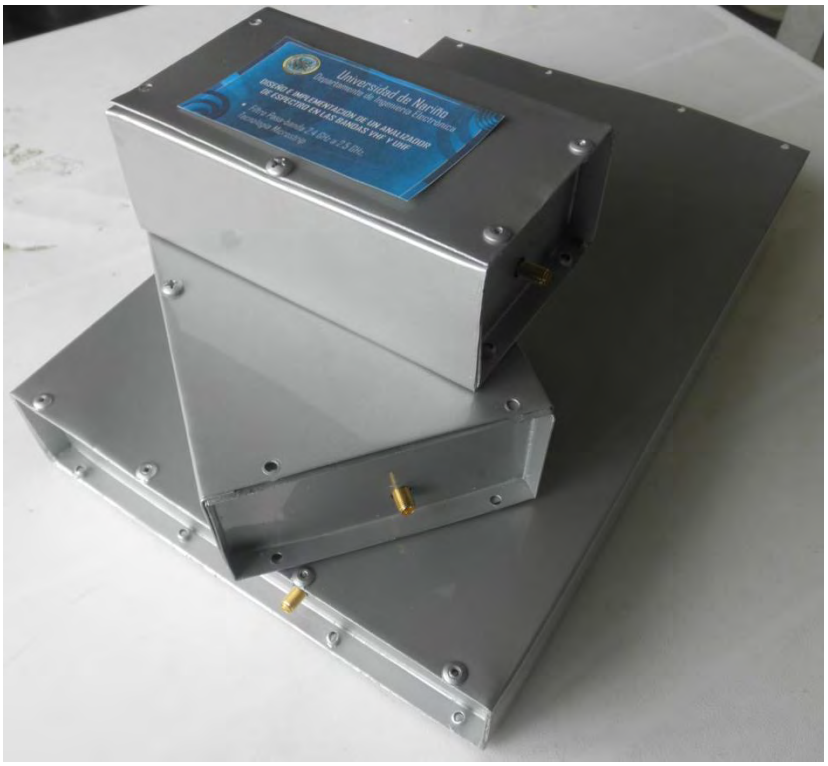
Montaje AD9481-PCBZ + Mezclador y módulo de reloj



Montaje AD9481-PCBZ + Filtro pasa-bajas + Mezclador y módulo de reloj



Cajas filtros DBR



Caja ADC + Filtro pasa-bajas + Mezclador + oscilador

Conexión ADC –FPGA