UNIVERSIDAD NACIONAL DEL CALLAO

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

DESARROLLO DE UN SISTEMA TELEMÉTRICO AVANZADO PARA SENSORES MEDIO-AMBIENTALES REMOTOS

TESIS PARA OPTAR EL TÍTULO PROFESIONAL DE

INGENIERO ELECTRÓNICO

RENZO JOSÉ CHAN RIOS

Callao, 2017

PERÚ

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

TESIS PARA OPTAR EL TÍTULO PROFESIONAL DE
INGENIERO ELECTRÓNICO

"DESARROLLO DE UN SISTEMA TELEMÉTRICO AVANZADO PARA SENSORES MEDIO-AMBIENTALES REMOTOS"

PRESENTADO POR EL BACHILLER:
RENZO JOSÉ CHAN RIOS

ASESOR:
MSc. Ing. RAÚL BENITES SARAVIA

CALIFICACIÓN:
(17) DIECISIETE

MSc. Ing. ARMANDO PEDRO CRUZ RAMÍREZ	**MSc. Ing. JACOB ASTOCONDOR VILLAR**
Presidente de Jurado	**Secretario de Jurado**

MSc. Ing. JULIO CESAR BORJAS CASTAÑEDA
Vocal de Jurado

Callao, 2017

PERÚ

DEDICATORIA

Este tesis la dedico con mucho cariño y estima a mis queridos padres María y Enrique, a mis hermanos Pamela y Mathías, a mis abuelitos Paulina, María, José y Alejandro, y a mi enamorada Ellizabeth, por brindarme su apoyo durante la ejecución de este documento tan importante para mi carrera profesional de Ingeniero Electrónico.

ÍNDICE

TABLAS DE CONTENIDO

A. Contenido de Figuras

B. Contenido de Tablas

C. Contenido de Gráficos

RESUMEN

La presente tesis describe el desarrollo de un prototipo de sistema telemétrico avanzado para el monitoreo de diferentes tipos de sensores, ubicados en zonas remotas para adquirir distintos parámetros medio-ambientales como por ejemplo temperatura, niveles de toxicidad de algunos gases y niveles de radiación nuclear, usando como hardware para la transmisión y recepción de datos, Radios Definidas por Software o SDR (por sus siglas en inglés Software Defined Radio), que son equipos de radiocomunicación, los cuales reemplazan algunos componentes de hardware (de un sistema de comunicación común) por rutinas de software, permitiendo así reducir costos y tamaños de implementación.

Las rutinas de software se diseñaron usando GNU Radio, el cual es un conjunto de herramientas de desarrollo de software libre y de código abierto, que proporciona bloques de procesamiento de señales para implementar radios definidas por software.

El uso de estas dos herramientas de código abierto, permitió desarrollar un sistema de radiocomunicación muy versátil y escalable, el cual fácilmente puede adaptarse o ampliar sus funcionalidades, realizando un rediseño del software según las necesidades del ambiente donde se encuentra implementado el sistema de telemetría.

Éste proyecto se realizó en el marco del convenio 198-FINCyT-IA-2013 y el Programa Nacional de Innovación para la Competitividad y Productividad, Innóvate-Perú, financiamiento otorgado al Instituto Peruano de Energía Nuclear (IPEN).

ABSTRACT

This thesis describes the development of a prototype advanced telemetry system for monitoring different types of sensors, in remote areas to acquire different environmental parameters such as temperature, toxicity levels of some gases and levels of nuclear radiation, using as hardware for the transmission and reception of data, Software Defined Radio, which are radio equipment, that replace some hardware components (of a communication system common) by software routines, thus decreasing implementation costs and sizes.

Software routines were designed using GNU Radio, which is a free & open-source software development toolkit that provides signal processing blocks to implement software radios.

The use of these open source tools, allowed the development of a versatile and scalable radio system which can easily adapt or extend its functionality, making a redesign of the software according to the needs of the environment where the telemetry system is implemented.

This project was carried out under the agreement 198-FINCyT-IA-201 and the National Program of Innovation for Competitiveness and Productivity, Innóvate-Peru, financing granted to the Peruvian Institute of Nuclear Energy (IPEN).

CAPÍTULO I
PLANTEAMIENTO DE LA INVESTIGACIÓN

1.1. Identificación del Problema

La medición de variables o parámetros medio ambientales para evaluar niveles de contaminación o condiciones seguras para la población es un requerimiento cada vez más imperativo en todo el mundo, y la necesidad de realizar estas mediciones de forma remota o en zonas rurales, usando sistemas telemétricos es cada vez más creciente.

Es por ello la necesidad de complementar los sensores medio-ambientales que existen actualmente, con sistemas de radiocomunicación avanzados, que permitan el envío y recepción de los parámetros, que estos dispositivos obtienen, sin involucrar costos mensuales.

1.2. Formulación del Problema

Hoy en día existen muchos prototipos de medición autónoma desarrollados en el país para diferentes aplicaciones medioambientales, como por ejemplo los sistemas de medición de calidad química del agua en ríos y lagos, o de gases contaminantes en el aire desarrollados por el IPEN (1)(4). Estos prototipos son en realidad dispositivos simples equipados con un sistema de sensores y de telemetría, para tomar medidas de diferentes variables en condiciones de campo y enviar los resultados a una estación central por medio de un sistema de comunicación basado en la tecnología GSM (de segunda generación o 2G).

El problema principal de este método es que, por cada dispositivo, se debe pagar a un operador de telefonía celular en forma mensual debido a que se usan bandas electromagnéticas que no son libres, y también que se restringen a la cobertura del operador. Otras soluciones emplean redes Wifi y el uso de internet para el envío de los datos que adquieren pero estos sistemas al igual que las redes GSM, dependen de la cobertura e involucran un costo por el acceso a la red. Por último existen equipos que usan módulos RF o módulos zigbee, para realizar la transmisión/recepción de datos pero en su mayoría son de corto alcance.

Otro problema de estos equipos es que la modulación, codificación y frecuencia en la cual trabajan no es variable, esto quiere decir, que se necesitaría comprar diferentes equipos para transmitir los datos de los sensores a diferentes frecuencias o con diferente modulación y codificación, según la aplicación y el entorno requiera, lo cual incrementa los costos.

¿Cómo se puede mejorar los actuales sistemas de telemetría aplicados en sensores medio-ambientales para lograr equipos flexibles, confiables y con un costo comparativamente menor?

¿Qué características necesita un sistema de telemetría para trabajar en zonas rurales y con sensores remotos?

¿Qué métodos, tecnología e instrumentos, se usarán en la implementación del proyecto?

¿Qué software especializado se aplicará en este diseño?

1.3. Objetivos de la Investigación

1.3.1. Objetivo General

Este proyecto tiene el objetivo de diseñar e implementar un sistema de telemetría avanzado para un radioenlace punto a punto aplicado a sistemas de monitoreo remoto de parámetros medio ambientales.

1.3.2. Objetivos Específicos

- Desarrollar la arquitectura del sistema de telemetría avanzado, para un radioenlace punto a punto.
- Diseñar una tarjeta electrónica para la adquisición de diferentes tipos de sensores.
- Diseñar e implementar el hardware y software del sistema cuando funciona como maestro
- Diseñar e implementar el hardware y software del sistema cuando funciona como esclavo.
- Desarrollar una aplicación de monitoreo de gases y radiación nuclear usando el nuevo sistema de telemetría.

1.4. Justificación

Existen en el Perú muchos proyectos, sistemas, instalaciones que usan tecnologías inalámbricas como GSM en la mayoría de los casos y también Wifi, u otras tecnologías especializadas. Estas soluciones demandan un costo relativamente alto y dependen de la cobertura que las operadoras de telefonía tengan. En el caso del

GSM hay que abrir una cuenta con los operadores telefónicos lo cual incurre en pagos mensuales. Los otros equipos son relativamente caros. Este proyecto justifica su desarrollo al permitir los mismos servicios a un costo mucho más económico debido al uso de bandas libres del espectro electromagnético, también al uso de diseños abiertos tanto de software como de hardware y su flexibilidad para trabajar en diferentes frecuencias del espectro electromagnético.

1.5. Importancia

1.5.1. Importancia social

Este proyecto es importante socialmente, porque permite conocer variables o parámetros medio ambientales para evaluar niveles de contaminación o condiciones seguras para la población, tanto en zonas urbanas como rurales, logrando así mejorar la calidad de vida de las personas.

1.5.2. Importancia tecnológica

La importancia tecnológica de esta investigación radica en que las radios definidas por software, que se utilizaron para el desarrollo del presente trabajo, son plataformas tecnológicas de comunicación relativamente nuevas, que están tomando mucha importancia en diferentes aplicaciones alrededor del mundo, pero que en nuestro país aún no se difunden, solo instituciones especializadas en las telecomunicaciones como INICTEL, el Radio Observatorio de Jicamarca y algunas universidades, cuentan ya, con este tipo de hardware, que incluso facilita el aprendizaje de temas como circuitos y sistemas de radio, telecomunicaciones, antenas, líneas de transmisión, etcétera.

CAPÍTULO II
MARCO TEÓRICO

2.1. Antecedentes del estudio

En primer lugar se consideró el trabajo realizado en el IPEN denominado "Desarrollo de un medidor autónomo de bajo costo para la determinación de calidad química del agua", publicado en el Informe Científico Tecnológico del Instituto Peruano de Energía Nuclear del año 2012 (4), con el cual determinaron plomo y otros metales pesados disueltos en agua, así como el pH, conductividad y temperatura. El prototipo incluyó un potenciostato, operado de manera autónoma por un micro-controlador, así como un sistema de comunicación mediante tecnología GSM, en la modalidad de mensajes de texto.

Otro antecedente es el trabajo de Duc Toan Nguyen, "Implementation of OFDM systems using GNU Radio and USRP", presentado en la Escuela de Ingeniería de Telecomunicaciones, Informática y Eléctrica de la Universidad de Wollongong en Australia, como parte de los requerimientos para obtener el grado de Master by Research en el año 2013. (7)

La investigación es un estudio e implementación de la Multiplexación por División de Frecuencias Ortogonales (OFDM) en un Radio Definido por Software (SDR), en este caso, el USRP (Universal Software Radio Peripheral) de Ettus Research, en esta investigación se analizó la modulación OFDM desde un punto de vista matemático, para luego implementarla en software con ayuda de las librerías de GNU Radio, también hacen un análisis profundo de los SDR, que partes lo

componen, como funcionan internamente, para finalmente ver su desempeño con este tipo de modulación. Todas las pruebas las realizaron en las instalaciones de la universidad y en diferentes condiciones. El análisis de los resultados permitió determinar los aspectos positivos y negativos de este tipo de modulación y hardware.

De esta manera también se tiene como antecedente el trabajo de Julio Andrés Santana Fuentes, "GNU Radio en la enseñanza de comunicaciones inalámbricas", presentado en la Facultad de Ingeniería de la Universidad de Concepción en Chile, como requisito para obtener el título de Ingeniero Civil en Telecomunicaciones en el año 2012 (8). Este trabajo sirvió para entender de todas las bondades de las librerías GNU Radio, ya que mediante ejercicios de laboratorio, va mostrando todo lo que se puede lograr con los SDR y GNU Radio, abarca temas como modulación, filtrado, empaquetado de datos y muchos otros tópicos que fueron de mucha ayuda para lograr nuestro objetivo.

Por último se consideró el trabajo de Lei Zhang, "Implementation of Wireless Communication based on Software Defined Radio", presentado en el Departamento de Ingeniería Audiovisual y Comunicaciones de la Universidad Politécnica de Madrid, como parte final del Programa Oficial de Postgrado en Ingeniería de Sistemas y Servicios para la Sociedad de la Información, para la obtención del grado de Master (10). En su trabajo se evaluó la plataforma de hardware USRP y el software GNU Radio, empleando técnicas de modulación básicas para la comunicación inalámbrica, verificando así el performance de estas herramientas.

2.2. Fundamentación

2.2.1. Ontológica

Se desarrolló e implementó un sistema de telemetría versátil que permite enviar datos de distintos tipos de sensores medio-ambientales (análogos y digitales) en diferentes rangos de frecuencia y modulación, usando un radioenlace punto a punto, con la finalidad de medir y registrar variables o parámetros para evaluar niveles de contaminación o condiciones seguras para la población, realizando estas mediciones de forma remota en zonas rurales.

2.2.2. Metodológica

Se determinó el procedimiento realizando una investigación de tipo analítico-experimental, definiendo las partes más generales del proyecto mediante la ingeniería básica del mismo que permitió reconocer la arquitectura del sistema a utilizar, hasta la técnica más particular del sistema de telemetría, teniendo como estrategia desarrollar las secciones de software y hardware, para finalmente realizar pruebas que nos permitieron obtener una base de datos empírica, que justifican la elección de los instrumentos para la recolección de datos usados en la investigación.

2.2.3. Epistemológica

Se desarrolló el sistema de telemetría usando radios definidas por software porque permite tener un sistema de comunicación por radiofrecuencia adaptable y escalable; el cuál se aplicó para mantener un registro de la calidad de aire y radiación en el centro nuclear RACSO.

Este estudio también se realizó con la intención de generar una base en el uso de las radios definidas por software para el envío de variables y parámetros medio-ambientales.

2.3. Definición de términos

A continuación se brinda el significado preciso, según el contexto de los conceptos principales, expresiones y/o variables vertidas en éste proyecto.

ADC

Analog to Digital Converter o Conversor Análogo a Digital es un dispositivo electrónico capaz de convertir una señal analógica de voltaje en una señal digital con un valor binario.

Banda Base

El término banda base se refiere a la banda de frecuencias producida por un transductor, tal como un micrófono, un manipulador telegráfico u otro dispositivo generador de señales que no es necesario adaptarlo al medio por el que se va a trasmitir

BER

La tasa de error binario o Bit Error Rate se define como el número de bits recibidos de forma incorrecta respecto al total de bits enviados durante un intervalo especificado de tiempo.

DAC

Digital to Analog Converter o Conversor Digital a Análogo, es un dispositivo para convertir señales digitales con datos binarios en señales de corriente o de tensión analógica.

DDC

Digital Down Converter o un convertidor digital descendente (DDC) convierte una señal real digitalizada centrada a una frecuencia intermedia (IF) a una señal

compleja en banda base centrada en la frecuencia cero. Además su conversión es en sentido descendente.

DUC

Digital Up Converter o Conversor Digital Ascendente realiza la misma función que un DDC solo que su conversión es en sentido, ascendente es decir, muestrea a una mayor frecuencia.

FFT

FFT es la abreviatura usual (del inglés Fast Fourier Transform) de un eficiente algoritmo que permite calcular la transformada de Fourier discreta (DFT) y su inversa.

GNU

GNU es un sistema operativo basado en Unix desarrollado por el Proyecto GNU. Está formado en su totalidad por software libre.

GPL

La Licencia Pública General de GNU o más conocida por su nombre en inglés GNU General Public License, es la licencia más ampliamente usadal en el mundo del software y garantiza a los usuarios finales la libertad de usar, estudiar, compartir (copiar) y modificar el software.

GUI

La interfaz gráfica de usuario, es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

IF

Se denomina Frecuencia intermedia (IF) a la Frecuencia que en los aparatos de radio que emplean el principio superheterodino se obtiene de la mezcla de la señal sintonizada en antena con una frecuencia variable generada localmente en el propio aparato mediante un oscilador local y que guarda con ella una diferencia constante. Esta diferencia entre las dos frecuencias es precisamente la frecuencia intermedia.

Open Source

Código abierto o expresión con la que se conoce al software distribuido y desarrollado libremente.

PYTHON

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

QT

Qt es una biblioteca multiplataforma ampliamente usada para desarrollar aplicaciones con interfaz gráfica de usuario, así como también para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores.

Sensores medio-ambientales

Son sensores o transductores enfocados en la medición de algún parámetro medio-ambiental, por ejemplo calidad del agua, sensores de gases tóxicos, cuentan con una electrónica capaz de convertir esos factores en datos digitales para su posterior procesamiento.

SWIG

Simplified Wrapper and Interface Generator es una herramienta de software de código abierto que se utiliza para conectar los programas de ordenador o bibliotecas escritas en C o C ++ con lenguajes de script como Lua, Perl, PHP, Python, R, Ruby, Tcl.

RF

El término radiofrecuencia, también denominado espectro de radiofrecuencia o RF, se aplica a la porción menos energética del espectro electromagnético, situada entre unos 3 Hz y unos 300 GHz.

WX

Son unas bibliotecas multiplataforma y libres, para el desarrollo de interfaces gráficas programadas en lenguaje C++. Están publicadas bajo una licencia

LGPL, similar a la GPL con la excepción de que el código binario producido por el usuario a partir de ellas, puede ser propietario, permitiendo desarrollar aplicaciones empresariales sin coste de licencias.

CAPÍTULO III
VARIABLES E HIPÓTESIS

3.1. Variable de la Investigación

3.1.1. Variables Independientes

- **Datos adquiridos**

Se refiere a los datos que se adquieren y entregan los sensores medio-ambientales, los cuales en su mayoría, cuentan con circuitos de acondicionamiento para entregar valores analógicos entre 0 y 5 Voltios, rango permisible por los módulos ADC, de los circuitos integrados a usar.

- **Señal de Ruido**

Toda señal no deseada que se mezcla con la señal útil que se quiere transmitir. Es el resultado de diversos tipos de perturbaciones que tiende a enmascarar la información cuando se presenta en la banda de frecuencias del espectro de la señal, es decir, dentro de su ancho de banda.

3.1.2. Variables Dependientes

- **Frecuencia de trabajo**

Es la frecuencia del espectro electromagnético que se usara para transmitir los datos adquiridos y modulados.

- **Tipo de modulación y Demodulación**

Variable en la cual se modifican ciertas características de una señal de forma proporcional al mensaje o señal que queremos transmitir o recepcionar.

- **Interfaz Gráfica de Usuario**

Software para el manejo del sistema telemétrico, que estará en la etapa de recepción y en la cual se visualizarán los datos adquiridos por los sensores medio-ambientales. Se usarán GUIs basadas en las librerías QT o WX.

3.2. Operacionalización de variables

TABLA N° 3.1

OPERACIONALIZACIÓN DE VARIABLES

Variable	Tipo	Definición	Indicador
Datos Adquiridos	Continua Independiente	Aquellos que se obtendrán a partir de los sensores medio-ambientales.	Trama de datos
Señal de Ruido	Independiente	Señal no deseada que se mezcla con la señal útil que se quiere transmitir.	Relación Señal a Ruido (SNR)
Frecuencia de Trabajo	Dependiente	Parte del espectro electromagnético que se usará para transmitir y recepcionar los datos adquiridos.	Antena que se usará para transmitir los datos.
Modulación y Demodulación	Dependiente	Variación de las características de la señal adquirida de manera proporcional, según la frecuencia en la que se va a trabajar.	Algoritmo de transmisión y recepción de los datos.
Interfaz Gráfica de Usuario	Dependiente	Programa que permitirá al usuario interactuar y visualizar los datos	Software para la recepción y visualización de

		recepcionados, y que permitirá seleccionar las frecuencias de trabajo y tipos de modulación usados.	los datos adquiridos por los sensores.

3.3. Hipótesis

Se plantea una hipótesis general en el entendimiento de que la investigación trata de desarrollar un sistema de telemetría usando los avances tecnológicos actuales, para así obtener un radioenlace punto a punto flexible usando frecuencias libres de nuestro espectro electromagnético y hardware existente para implementar sistemas inalámbricos complejos denominados radio definido por software (software defined radio - SDR), los cuales tienen diseños de hardware de código abierto, donde no cuesta el tener acceso a los esquemáticos, de la misma manera el uso de software libre para el desarrollo de aplicaciones .

De la hipótesis planteada se derivan los procedimientos para el desarrollo del sistema de telemetría avanzado, cuya instalación y prueba se realizaron en el Centro Nuclear RACSO, Huarangal.

CAPÍTULO IV
METODOLOGÍA

4.1. Tipo de Investigación

4.1.1. Analítico

Para realizar un análisis de los elementos que conforman este proyecto el método de investigación a utilizar es el Método Analítico, para así desintegrar, descomponer un todo (el sistema de telemetría), en sus partes para estudiar en forma intensiva cada uno de sus elementos, así como las relaciones entre sí y con el todo.

4.1.2. Experimental

Porque se realizaron diferentes pruebas preliminares para desarrollar el sistema de telemetría y pruebas finales que nos permitieron comprobar su performance.

4.1.3. Espacial

Porque la investigación se desarrolló como referencia en el Centro Nuclear RACSO, Huarangal.

4.2. Diseño de la Investigación

Para lograr el objetivo propuesto, la investigación estuvo compuesta por 3 grandes etapas, dentro de las cuales existieron actividades de tipo teórica, de simulación, y experimental.

4.2.1. Diseño e implementación de la Arquitectura del Sistema

En primer lugar se usaron los principales resultados y modelos teóricos (estado del arte) que existen en el área de las telecomunicaciones, acerca de las radios definidas por software, las herramientas computaciones y lenguajes de programación para trabajar con ese tipo de hardware, para así definir la arquitectura del sistema, para ello se realizó una investigación teórica de las radios definidas por software, modulación digital y requerimientos de los sistemas telemétricos para zonas rurales, las cuales se presentan a continuación.

Software Defined Radio (SDR)

Tradicionalmente los equipos receptores y transceptores de radiocomunicaciones son equipos constituidos por multitud de componentes electrónicos, los cuales forman circuitos sintonizadores, etapas de frecuencia intermedia, detectores, amplificadores de baja frecuencia, etc..., es decir, están constituidos por "hardware". Posteriormente, en los años 1980´s y 1990´s se introdujeron microprocesadores en estos equipos para el control de funciones internas (controles desde teclados y pulsadores) y para añadir nuevas prestaciones (relojes, pantallas informativas, programadores, etc...), y también se introdujo la posibilidad de controlar los equipos de radio desde un ordenador, añadiendo al equipo de radio puertos de comunicación o interfaces para la conexión al ordenador. En estos casos, y usando el software adecuado, es posible controlar desde el ordenador numerosas funciones del equipo de radio, igual o mejor que desde los controles del propio equipo. También en la década de los 1990´s comenzó la introducción en los modernos equipos de radio de los chips

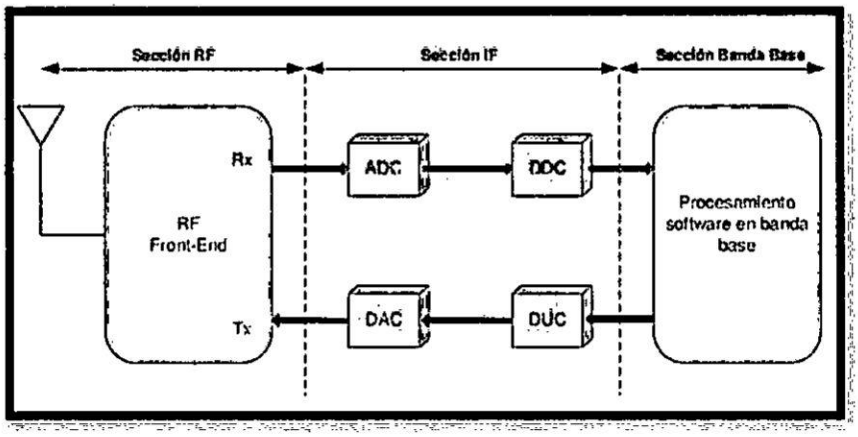
DSP o "Procesadores Digitales de Señal", los cuales permiten mediante técnicas digitales realizar filtros de paso de banda y de supresión de ruidos, entre otras posibilidades, muy eficaces, mejor que los realizados tradicionalmente con circuitos analógicos. Un chip DSP es básicamente una especie de CPU en miniatura, un chip microprocesador con un juego de instrucciones pequeño, pero capaz de ejecutarlas a una velocidad muy superior a la velocidad de una CPU corriente, gracias a una estructura simplificada y al reducido juego de instrucciones.

En cualquier caso, siempre se trata de equipos de radio realizados enteramente con componentes electrónicos, o sea, en términos informáticos se definirían como "radios hardware". Pero desde principios de la década del 2000 radioaficionados como Gerald Youngblood (AC5OG), comenzaron a investigar y desarrollar un nuevo concepto de equipos de radiocomunicaciones, los equipos de radio desarrollados por programa o "radios software", en siglas SDR (Software Defined Radio) (10), en los que la parte hardware (circuitería) es mínima, y la mayor parte de las funciones que definen un equipo de radio se definen por software (programas) en un ordenador PC o de otro tipo, dotado de tarjeta de sonido (requisito necesario).

El concepto SDR ha ido evolucionando con los años pero se siguen basando en el esquema básico que se muestra más adelante (véase la figura N° 4.1, en la página 24), compuesta por tres bloques funcionales: sección de RF, sección de IF y sección Banda Base. La parte de RF e IF se implementan en hardware mientras que la sección de Banda Base en software.

FIGURA N° 4.1

DIAGRAMA DE BLOQUES DE UN SDR



Bloques Funcionales de un SDR

- **Sección de RF:**

También denominada RF Front-End. Como Rx es la encargada de recibir las señales de radiofrecuencia para adecuarlas y convertirlas en Frecuencia Intermedia (IF, Intermediate Frequency). Como Tx es la encargada de Amplificar y modular las señales de IF.

La frecuencia intermedia puede ser 0, dando lugar al concepto de Zero-IF; el cual es posible gracias a los avances en los componentes hardware.

- **Sección de IF:**

Durante la recepción se encarga de pasar la señal de IF a banda base y digitalizarla. En el caso de la transmisión pasa la señal de banda base a IF y hace la conversión digital-analógica de la señal. Las encargadas de la conversión Analógica-Digital o Digital-Analógica de la señal son los módulos ADC/DAC. A su vez, se insertan los módulos DDC/DUC para

poder bajar/subir, respectivamente, la tasa de muestreo en el sentido de recepción/transmisión, consiguiendo que la tasa de muestras por la interfaz entre IF y banda base sea inferior.

- **Sección de Banda Base:**

  Es la encargada de todo el procesamiento en banda base de la señal como modulación/demodulación, análisis espectral de la señal, llevándose a cabo en software.

GNU Radio

GNU Radio es un software de desarrollo de herramientas de código libre y abierto que proporciona bloques de procesamiento digital de señales para implementar aplicaciones usando hardware RF externo de bajo costo o sin hardware, utilizando el entorno de simulación. Es ampliamente utilizado en entornos de aficionados, académicos y comercial para contribuir tanto en la investigación de comunicaciones inalámbricas y sistemas de radio del mundo real.

Las aplicaciones de GNU Radio son escritas en principio utilizando el lenguaje de programación Python, mientras que el suministro de herramientas críticas de procesamiento de señales que requieren alto rendimiento son implementados en C++ usando extensiones de procesamiento de punto flotante, cuando este está disponible. Así, el desarrollador es capaz de implementar, de manera simple, sistemas de radio de alto rendimiento funcionando a tiempo real aprovechando el ambiente de desarrollo de aplicaciones de manera inmediata.

Aunque no es una herramienta principalmente de simulación, GNU Radio complementa el desarrollo de algoritmos de procesamiento de señales a partir de datos previamente grabados o generados, evitando la necesidad de hardware de RF.

Cabe destacar que GNU Radio está licenciado bajo el GNU General Public License (GPL) versión 3. Todo el código es propiedad de la Fundación del Software Libre.

¿Cómo trabaja GNU RADIO?

Radio GNU realiza todo el procesamiento de señales. Se puede utilizar para diseñar aplicaciones que reciban y/o envíen flujos de datos digitales, que luego se transmiten usando hardware. GNU Radio tiene filtros, códigos de canal, elementos de sincronización, ecualizadores, demoduladores, vocoders, decodificadores, y muchos otros elementos, que se conocen como bloques y que típicamente se encuentran en los sistemas de radio.

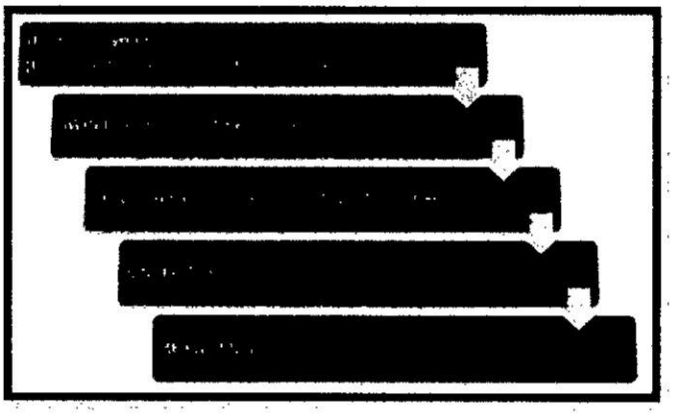
Desde GNU Radio, sólo se pueden manejar datos digitales. Por lo general, las muestras de banda de base complejas son el tipo de datos de entrada para los receptores y el tipo de datos de salida para los transmisores. El Hardware analógico se usa entonces para desplazar la señal a la frecuencia central deseada. Dejando de lado ese requisito, cualquier tipo de datos se puede pasar de un bloque a otro ya sea de bit, byte, vectores, o tipos de datos complejos.

Para construir un sistema de radio con GNU Radio se debe crear un grafo, donde los nodos son bloques de procesado de señal y los enlaces entre nodos

representan el flujo de datos entre ellos. Los bloques de procesado de señal son implementados en C++ y portados a Python a partir del programa SWIG, es decir, para implementar los diseños se crearán los módulos en Python que a su vez se apoya en los bloques implementados en C++. Las interacciones entre los diferentes niveles de GNU Radio aparecen en la Figura N° 4.2.

FIGURA N° 4.2

ARQUITECTURA GNU RADIO



GNU Radio Companion

La construcción de módulos utilizando esta herramienta gráfica se basa en añadir bloques e interconectarlos de manera esquemática. Los bloques que se usan se pueden agrupar de la siguiente manera:

- **Sources (fuentes):**

  Estos bloques especifican cualquier tipo de fuente como por ejemplo un archivo de audio wav, un generador de señal de forma arbitraria y con las características que se requieran, una fuente binaria aleatoria, un fichero de cualquier formato, un micrófono, un SDR, etc.

- **Bloques de procesado de señal:**

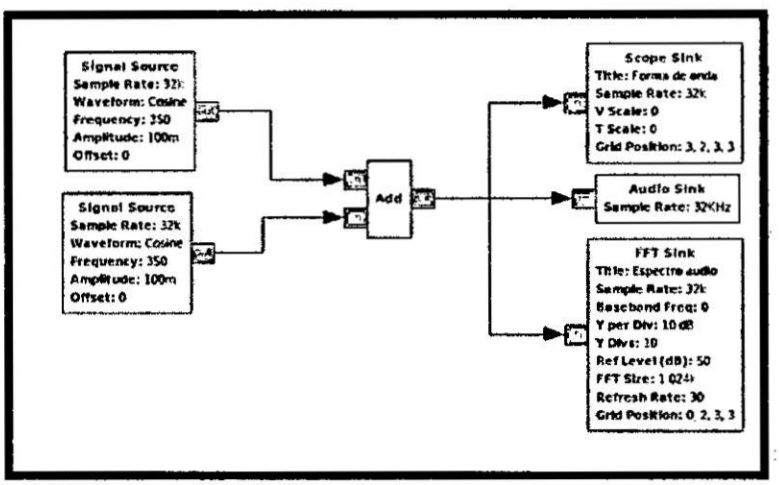
A este grupo pertenecen todos aquellos bloques que realizan un tratamiento de la señal de cualquier tipo. Se pueden citar como ejemplos los moduladores, filtros, remuestreadores, multiplicadores, amplificadores en software, etc.

- **Sinks (sumideros):**

La señal tendrá un destino final como bien puede ser un fichero de cualquier formato, la tarjeta de sonido o algún hardware SDR. Cabe indicar que a este conjunto también pertenecen los bloques de visualización de señales (Graphical sinks) como FFT sink (para visualizar la FFT de la señal en un punto), Constelation sink u Osciloscope sink (para representar la señal en tiempo en cualquier lugar del diseño) entre otros.

FIGURA N° 4.3

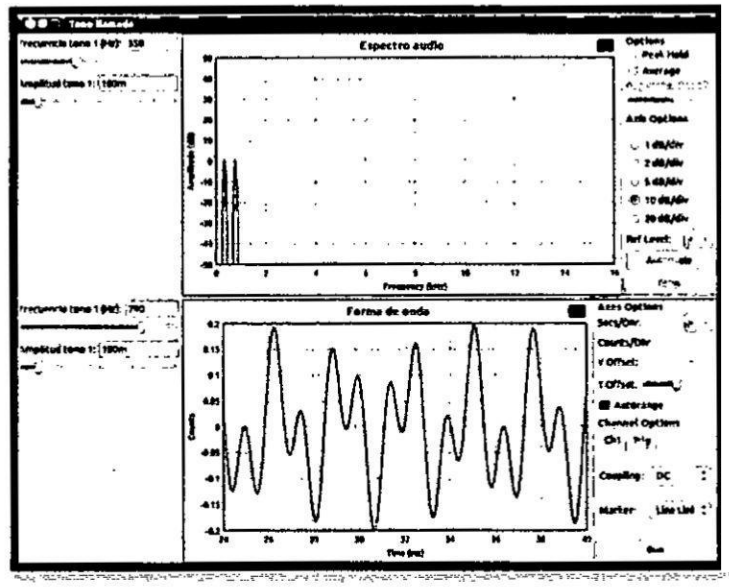
DIAGRAMA DE BLOQUES EN GNU RADIO COMPANION



Como se puede ver en la Figura N° 4.3, además de los bloques mencionados se añaden una serie de variables para controlar los parámetros desde la interfaz gráfica (en este caso Variable Slider, aunque hay otras opciones), de tal manera que en el parámetro frecuencia de la primera fuente se le asigna el identificador f1 y al ejecutar el esquemático se podrá controlar dicha frecuencia. Igual sucede con el resto de parámetros. Una vez se tenga el diseño completado, éste se guardará en un fichero *.grc que al compilarlo (simplemente con pulsar F6 dentro de la aplicación) genera un fichero *.py que se puede ejecutar para visualizar lo siguiente:

FIGURA N° 4.4

VISUALIZACIÓN DEL MÓDULO DIAL_TONE.PY



Ahora se puede visualizar el espectro y la forma de onda y su modificación al variar los parámetros creados con las Variables Slider.

Siguiendo el mismo proceso se pueden llevar a cabo infinitas aplicaciones teniendo en cuenta que existen bloques para cualquier procesado de señal e incluso bloques moduladores y demoduladores completos, simplemente añadiendo bloques y configurando todo adecuadamente, a partir de una interfaz gráfica intuitiva que ahorra el trabajo de programar directamente en Python.

Modulación Digital

- **Modulación por Desplazamiento Mínimo Gaussiano**

  La Modulación por desplazamiento mínimo gaussiano, también conocida por su acrónimo en inglés GMSK (Gaussian minimum shift keying), es un esquema de modulación digital por desplazamiento de frecuencia de fase continua, similar a la MSK.

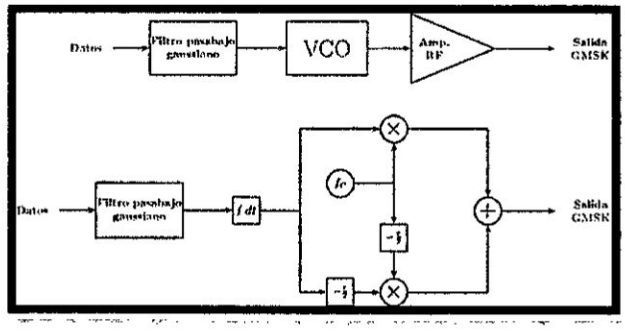
  Aunque es similar a la modulación por desplazamiento mínimo (MSK), su diferencia respecto a ésta es que el flujo de datos digitales atraviesa un filtro pasabajo gaussiano antes de ser aplicada al circuito modulador, lo que tiene la ventaja de "suavizar" las transiciones de fase de la señal durante la transmisión y así reducir el ancho de banda necesario, a su vez, reduce la interferencia fuera de banda entre portadoras de señal en canales de frecuencias adyacentes. Sin embargo, en general, la duración del pulso a la salida del filtro gaussiano es mayor que el tiempo de un bit lo que ocasiona interferencia entre símbolos, 1 por lo que es más difícil diferenciar entre diferentes valores transmitidos de los datos y requiere ecualización adaptativa en el receptor puesto que no está bien definida la transición entre un bit y el siguiente.

- **Modulación GMSK**

Existen dos modos de generar señales mediante GMSK. Uno de ellos, consiste en una modulación por FSK que usa un circuito de VCO y el otro usa un modulador de cuadratura 2 como los usados para QPSK y QAM. Después del filtrado pasabajo gaussiano, la señal atraviesa un integrador y, la salida es dividida en dos partes: una de ellas es aplicada a un modulador de producto para ser mezclada con la señal de portadora que tiene una frecuencia f_c; a la otra se le aplica un desfase de $\pi/2$ radianes (90°) para ser mezclada en otro modulador de producto con la portadora mencionada, desfasada también en $\pi/2$ radianes. Las salidas de los dos moduladores son las entradas de un sumador lineal. Con este montaje alterno, es posible mantener en 0,5 (50%) el índice de modulación sin necesidad de mayores ajustes, además de que representa un menor costo para su implementación.

FIGURA N° 4.5

DIAGRAMA DE BLOQUES DE DOS TIPOS DIFERENTES DE MODULADORES PARA GMSK



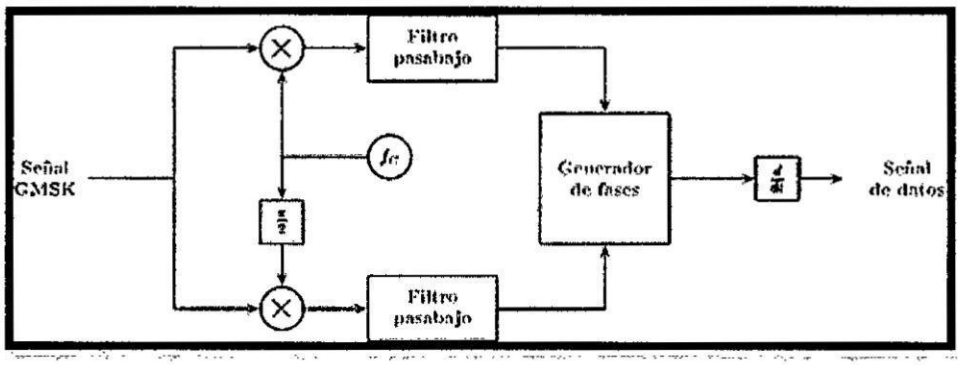
- **Demodulación GMSK**

La demodulación hace uso de un montaje similar a los que se usan en las modulaciones digitales QAM y QPSK, aunque presenta ciertas dificultades

ya que en el caso del modulador que usa el VCO, el índice de modulación oscila con la temperatura. Un demodulador típico consiste en un montaje con dos demoduladores de producto a los cuales se aplica la misma portadora ya recuperada, pero en uno de ellos se recibe la portadora desfasada en $\pi/2$ radianes (90°). Las señales a la salida son filtradas adecuadamente y son aplicadas a un generador de fase que reconstruye las posibles transiciones de fase. Finalmente, un bloque derivador reconstruye los bits en forma bipolar o NRZ (-V y +V).

FIGURA N° 4.6

DIAGRAMA DE BLOQUES DE UN DEMODULADOR TÍPICO DE SEÑALES GMSK



- **Respuesta del filtro gaussiano**

El filtro pasabajo gaussiano debe ser diseñado de modo tal que permita suaves transiciones de fase en la señal de salida del modulador y así acorte el ancho de banda necesario. La respuesta temporal del filtro ante un impulso, matemáticamente viene dada por la ecuación:

$$g(t) = \frac{1}{2T}\left[Q\left(2\pi B_t * \frac{t - t/2}{\sqrt{\ln 2}}\right) - Q\left(2\pi B_t * \frac{t + t/2}{\sqrt{\ln 2}}\right)\right] \forall 0 \leq B_b T \leq \infty \quad (4.1)$$

Siendo:

B_b: Ancho de banda del filtro gaussiano.

T: Tiempo de bit.

B_t: Ancho de banda normalizado.

Donde la función Q(t) es expresada como:

$$Q(t) = \int_t \frac{1}{\sqrt{2}} \exp\left(-\frac{x^2}{2}\right) dx \quad (4.2)$$

Probabilidad de error (P(e)) en las Modulaciones Digitales

Los términos probabilidad de error, P (e) y la tasa o frecuencia de errores de bits (BER, por bit error rate) se usan con frecuencia de forma indistinta, aunque en la práctica sí tienen significados un poco distintos. La P (e) es la expectativa teórica, o matemática, de que determinado sistema tenga una tasa de errores. La tasa de errores de bits es un registro empírico (histórico) del funcionamiento real del sistema en cuanto a errores. Por ejemplo, si un sistema tiene una P (e) de 10-5, eso quiere decir que en el pasado hubo un bit erróneo en cada 100,000 bits transmitidos. Una tasa de error de bits se mide y a continuación se compara con la probabilidad esperada de error, para evaluar el desempeño de un sistema.

La probabilidad de error es una función de la relación de potencia de portadora a ruido (o en forma más específica, el promedio de la relación de energía por bit

entre la densidad de potencia de ruido) y de la cantidad de condiciones posibles de codificación que se usan (M-aria). La relación de potencia de portadora a ruido es la de la potencia promedio de la portadora (la potencia combinada de la portadora con sus bandas laterales asociadas) entre la potencia del ruido térmico. La potencia de portadora se puede expresar en watts o en dBm, siendo

$$C_{(dBm)} = 10\log\frac{C_{(watts)}}{0.001} \tag{4.3}$$

El ruido térmico se describe con la ecuación:

$$N = KTB\ (watts) \tag{4.4}$$

En donde:

N = potencia de ruido térmico (watts)

K = constante de proporcionalidad de Boltzman ($1.38x10^{-23}$joules por kelvin)

T = temperatura (Kelvins: 0 K = -273 °C; temperatura ambiente = 290 K)

B = ancho de banda (hertz)

Expresada en dBm,

$$N_{(dBm)} = 10\log\frac{KTB}{0.001} \tag{4.5}$$

La ecuación de la relación de potencia de portadora a ruido es:

$$\frac{C}{N} = \frac{C}{KTB}\ (\text{relación adimensional}) \tag{4.6}$$

Siendo:

C = potencia de portadora (watts)

N = potencia de ruido (watts)

Expresada en dB,

$$\frac{C}{N}(dB) = 10\log\frac{C}{N} = C_{(dB)} - N_{(dB)} \tag{4.7}$$

La energía por bit no es más que la energía de un solo bit de información, que se define con la siguiente ecuación:

$$E_b = CT_b \text{ (J/bit)} \tag{4.8}$$

En donde:

Eb = energía de un solo bit (joules por bit)

Tb = tiempo de un solo bit (segundos)

C = potencia de la portadora (watts)

Expresada en dBJ:

$$E_{b(dBJ)} = 10\log E_b \tag{4.9}$$

Y como $T_b = 1/f_b$ es la rapidez de bits, siendo f_b la frecuencia de bits, en bits por segundo, se puede expresar E_b como:

$$E_b = \frac{C}{f_b} \text{ (J/bit)} \tag{4.10}$$

Expresada en dBJ:

$$E_{b(dBJ)} = 10\log\frac{C}{f_b} = 10\log C - 10\log f_b \tag{2.11}$$

La densidad de potencia del ruido es la potencia de ruido normalizada a un ancho de banda de 1Hz, es decir, la potencia del ruido presente en un ancho de banda de 1Hz. La ecuación correspondiente es:

$$N_0 = \frac{N}{B} \text{ (W/Hz)} \tag{4.12}$$

En donde:

N0= densidad de potencia de ruido (watts por hertz)

N = potencia de ruido térmico (watts)

B = ancho de banda (hertz)

Expresada en dBm,

$$N_{0(dBm)} = 10\log\frac{N}{0.001} - 10\log B = N_{(dBm)} - 10\log B \tag{4.13}$$

Al combinar con las ecuaciones anteriores se obtiene:

$$N_0 = \frac{KTB}{B} = KT \text{ (W/Hz)} \tag{4.14}$$

Y Expresada en dBm,

$$N_{0(dBm)} = 10\log\frac{KTB}{0.001} + 10\log T \tag{4.15}$$

La relación de energía por bit a densidad de potencia de ruido se usa para comparar dos o más sistemas digitales de modulación que usen distintas velocidades de transmisión (frecuencias de bits), esquemas de modulación (FSK, PSK, QAM) o técnicas M-arias de codificación. La relación de energía por bit a

densidad de potencia de ruido es tan sólo la relación de la energía de un solo bit a la potencia de ruido presente en 1 Hz de ancho de banda. Así, Eb/N0 normaliza todos los esquemas multifásicos de modulación a un ancho de banda común, permitiendo una comparación más sencilla y más exacta de su desempeño con errores. La ecuación de Eb/N0 es:

$$\frac{E_b}{N_0} = \frac{C/f_b}{N/B} = \frac{CB}{Nf_b} \tag{4.16}$$

En donde Eb/N0 es la relación de energía por bit a densidad de potencia de ruido. Esta ecuación se reordena para llegar a lo siguiente:

$$\frac{E_b}{N_0} = \frac{C}{N} x \frac{B}{f_b} \tag{4.17}$$

En donde:

Eb/N0 = relación de energía por bit a densidad de potencia de ruido

C/N = relación de potencia de portadora a ruido

B/fb = relación de ancho de banda de ruido a frecuencia de bits

Expresada en dB,

$$\frac{E_b}{N_0}(dB) = 10\log\frac{C}{N} + 10\log\frac{B}{f_b} \tag{4.18}$$

$$\frac{E_b}{N_0}(dB) = 10\log C - 10\log f_b - (10\log N - 10\log B) \tag{4.19}$$

$$\frac{E_b}{N_0}(dB) = 10\log E_b - 10\log N_0 \tag{4.20}$$

En la ecuación 2.18 se ve que la relación Eb/N0 no es más que el producto de la relación de potencia de señal a ruido por la relación de ancho de banda de ruido entre la frecuencia de bits. También se ve en esa ecuación que cuando el ancho de banda es igual a la frecuencia de bits, Eb/N0 =C/N.

En general, la relación mínima de potencia de portadora a ruido necesaria para los sistemas QAM es menor que la requerida en los sistemas PSK comparables. También, mientras mayor sea el nivel de codificación usado (valores mayores de M), la relación mínima de potencia de portadora a ruido es mayor.

Principales requerimientos para la telemetría en áreas rurales

Si bien se pueden usar soluciones de red con componentes comerciales (COTS) para diseñar un sistema de telemetría para áreas rurales, su costo y complejidad son típicamente altos. Además, la funcionalidad de estos tipos de redes no coincide exactamente con los de telemetría para zonas rurales. Un ejemplo de estas redes COTS es WirelessHART que está destinado principalmente para monitoreo de procesos y aplicaciones de control, incluyendo componentes DCS (Distributed Control Systems) and SCADA (Supervisory Control And Data Acquisition). La zona rural donde el sistema de telemetría debe trabajar, es tal que no existe WiFi de manera típica, no hay servicio de telefonía celular y no hay alimentación AC disponible, sin embargo, es posible encontrar algunos lugares con conexión a Internet. Por lo tanto, cualquier solución de red inalámbrica debe atender los requisitos específicos impuestos por las aplicaciones típicas de telemetría en las zonas rurales.

Los requerimientos fundamentales más importantes para el sistema de telemetría para zonas rurales son:

- Muchos sensores dispersos en una zona con varios kilómetros de diámetro.
- Extremadamente bajo consumo de energía en los nodos de sensores, que permitan durar a las baterías varios años.
- No requerir un alto rendimiento de datos.
- Cada nodo sensor deberá tener una cobertura física de unos 100 metros.
- Usar las bandas de los espacios de Televisión en blanco.
- No necesitar de una interface para redes (quizás a excepción de la Estación Central y Clúster Head)
- La red deberá ser capaz de coexistir con otras redes que usan las mismas bandas de frecuencia.

4.2.2. Diseño e implementación del Hardware del Sistema

Una vez definida la arquitectura del sistema, la segunda etapa fue la selección del hardware, para cada parte que comprende el sistema, teniendo en cuenta, las características que debe tener un sistema de telemetría para trabajar en zonas rurales.

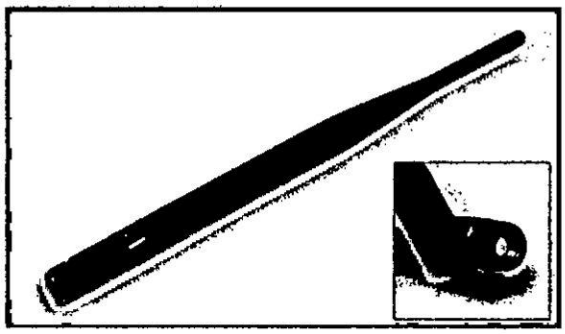
a. Antena RF

Como uno de los objetivos del sistema es diseñar un sistema con la capacidad de trabajar en diferentes frecuencias, se diseñó una antena del tipo Ground Plane de ¼ de onda, por su naturaleza omnidireccional y construcción

relativamente sencilla, para las frecuencias de 433MHz, y antenas de tipo RP SMA comerciales para las frecuencias de 915 -928MHz.

FIGURA N° 4.7

ANTENA RP SMA PARA 915MHZ



Para calcular las dimensiones de la antena Ground Plane, se desarrolló una aplicación GUI (Interfaz Gráfica de Usuario) con Python y Qt, que muestra las medidas que deben tener el radial central y los radiales que (Ver Figura N° 4.1 en la página 35) conforman el plano de tierra, tan solo con ingresar la frecuencia. Estos valores se obtienen según la siguiente fórmula:

$$X = {}^{c}/_{4f} \tag{4.21}$$

$$Y = 95\%\left({}^{c}/_{4f}\right) \tag{4.22}$$

Donde:

c: Velocidad de la luz

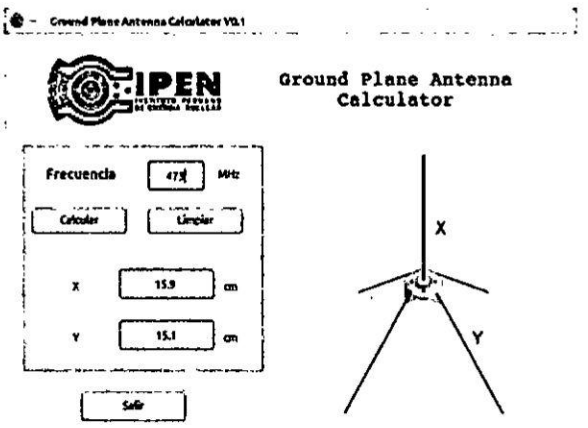
f: Frecuencia de trabajo

X: Radial Central

Y: Radiales del Plano de Tierra

FIGURA N° 4.8

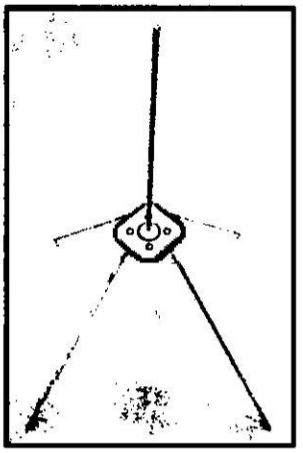
SOFTWARE PARA EL CÁLCULO DE RADIALES



Las antenas (ver Figura N° 4.2, en la página 37) se construyeron en el Taller de mecánica de la dirección de Investigación y desarrollo del IPEN, según las medidas obtenidas por nuestra aplicación.

FIGURA N° 4.9

ANTENA GROUND PLANE CONSTRUIDA PARA EL SUBSISTEMA DE RF



b. Subsistema de comunicación RF

El núcleo del sistema consta de una radio definida por software es por ello que en esta sección hablaremos de los SDRs más importantes, que se encuentran actualmente en el mercado, y de los cuales se vienen desarrollando importantes aplicaciones en todo el mundo.

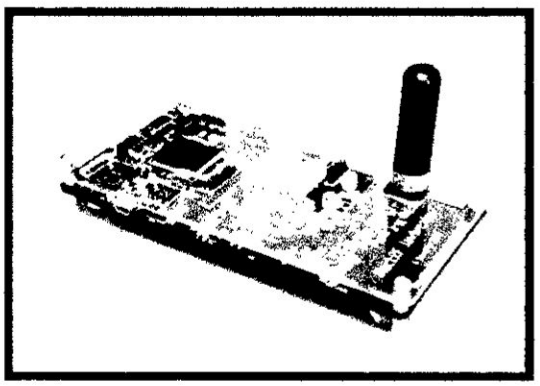
HackRF

HackRF producido por Great Scott Gadgets, el bladeRF producido por nuand y el USRP producido por Ettus.

El HackRF es desarrollado por Michael Ossmann, ha estado en desarrollo durante muchos meses y Michael ha distribuido 500 unidades beta del HackRF de forma gratuita a los desarrolladores de todo el mundo. Actualmente tienen el HackRF One, que es la versión de producción y venta. El HackRF One es uno de los SDR menos costosos y más completos actualmente y es capaz de abarcar una gran porción del espectro radioeléctrico.

FIGURA N° 4.10

MÓDULO SDR HACKRF DE GREAT SCOTT GADGETS

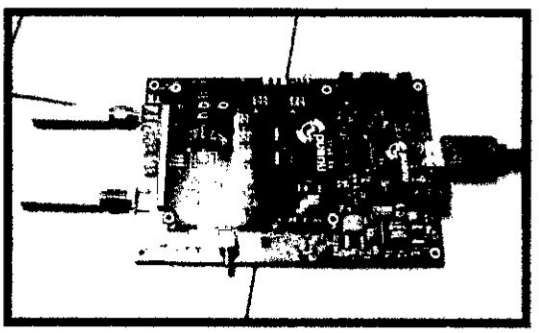


BladeRF

El bladeRF proporciona acceso a una gran porción del espectro de radio junto con una gran FPGA y enlace USB3 rápido. Los desarrolladores han invertido mucho en su arquitectura de reloj y proporcionar un VCTCXO que se ha calibrado dentro de 50 ppb. Todos los componentes fueron diseñados para estar en sintonía para que ningún Clock Timer sea necesario.

FIGURA N° 4.11

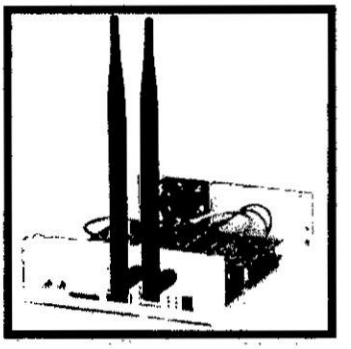
MÓDULO SDR BLADERF DE NUAND



Ettus es probablemente el más antiguo productor de Radio Definida por Software y han hecho muchos productos diferentes con sus respectivas revisiones. Su USRP B210/B200 es drásticamente diferente a la totalidad de sus productos anteriores, ya que es una solución de tarjeta única, en lugar de una combinación de "tarjeta madre/tarjeta hija". Utiliza USB3 y puede sintonizar una enorme porción del espectro. El B210/B200 también tiene una cabecera para un GPSDO de nuevo diseño (no incluido) de modo que puede ser calibrado a unos pocos ppb.

FIGURA N° 4.12

MÓDULO SDR USRP DE ETTUS RESEARCH



- Especificaciones

Se requieren placas secundarias separadas para recibir / transmitir.

El transceptor WBX se incluye en este kit.

La mitad de esto si se utilizan muestras de 16 bits.

56 MHz para el canal half duplex single, 30.72 MHz por canal full dúplex.

Hay un CPLD en el tablero, pero no FPGA.

Ettus confirmó que las HDL + código + Esquemas será lanzado para la B210/B200.

Precio estimado de venta al público, más barato aunque Kickstarter.

Todos estos dispositivos presentan ventajas entre uno y otro, la elección de alguno ellos dependerá principalmente de la aplicación y el uso que el desarrollador pretenda darle.

TABLA N° 4.1

TABLA COMPARATIVA DE SDR

	HackRF	bladeRF		USRP		
		x40	x115	B100 Starter	B200	B210
Radio Spectrum	30 MHz – 6 GHz	300 MHz – 3.8 GHz		50 MHz – 2.2 GHz [1]	50MHz – 6 GHz	
Bandwidth	20 MHz	28 MHz		16 MHz [2]	61.44 MHz [3]	
Duplex	Half	Full		Full	Full	2x2 MIMO
Sample Size (ADC/DAC)	8 bit	12 bit		12 bit / 14 bit	12 bit	
Sample Rate (ADC/DAC)	20 Msps	40 Msps		64 Msps / 128 Msps	61.44 Msps	
Interface (Speed)	USB 2 HS (480 megabit)	USB 3 (5 gigabit)		USB 2 HS (480 megabit)	USB 3 (5 gigabit)	
FPGA Logic Elements	[4]	40k	115k	25k	75k	150k
Microcontroller	LPC43XX	Cypress FX3		Cypress FX2	Cypress FX3	
Open Source	Everything	HDL + Code Schematics		HDL + Code Schematics	Host Code [5]	
Availability	January 2014	Now		Now	Now	
Cost	$300 [6]	$420	$650	$675	$675	$1100

Con respecto al Espectro Radioeléctrico

El HackRF y la USRP B210 pueden sintonizar a una enorme cantidad de espectro radioeléctrico. El HackRF puede sintonizar alrededor de 20 MHz por debajo de la B210/B200, pero ambos dispositivos pueden sintonizar tan alto como 6 GHz. El transceptor B210/B200 se basa en gran medida en el

AD9361 IC, que puede sintonizar a partir de 70 MHz a 6 GHz, con base en la información disponible por parte del fabricante. El HackRF por otra parte utiliza muchos componentes diferentes en lugar de un solo IC para la sintonización. Si nos fijamos en los esquemas, se encuentra una mezcolanza de fichas cada uno diseñado para operar en una determinada porción del espectro, Finalmente, está el BladeRF que puede sintonizar desde 300 MHz a 3,8 GHz gracias a la LMS6002D. Esto solo chip es responsable de la mayoría del trabajo de radio. Contiene todos los mezcladores, ADCs , DACs y mucho más. Es comparable a la AD9361 utilizado en el B210/B200.

Con respecto a la Comunicación

El método de comunicación es importante para la implementación de un Radio definido por Software, ya que determina la cantidad de datos que pueden ser transmitidos a un host y si se puede transmitir de forma fiable. Tanto el USRP B100 y el HackRF se basan en USB 2.0 de alta velocidad. Debido a los gastos generales del USB, esto limita la velocidad de transmisión de datos de pico a alrededor de 35 Mbytes / segundo. Sin embargo ya que la mayoría de nosotros tenemos más que nuestra SDR enchufado a los puertos USB, el bus debe ser compartido con todos los dispositivos en la red.

Tanto el bladeRF y la USRP B210/B200 usan USB 3.0, que es capaz de mover 400 MBytes segundo. Esto debería ser más que suficiente para el SDR. Sin embargo, si algún otro dispositivo está acaparando el bus, todavía podría causar perdida de muestras, sin embargo, es mucho menos probable.

Con respecto al FPGA

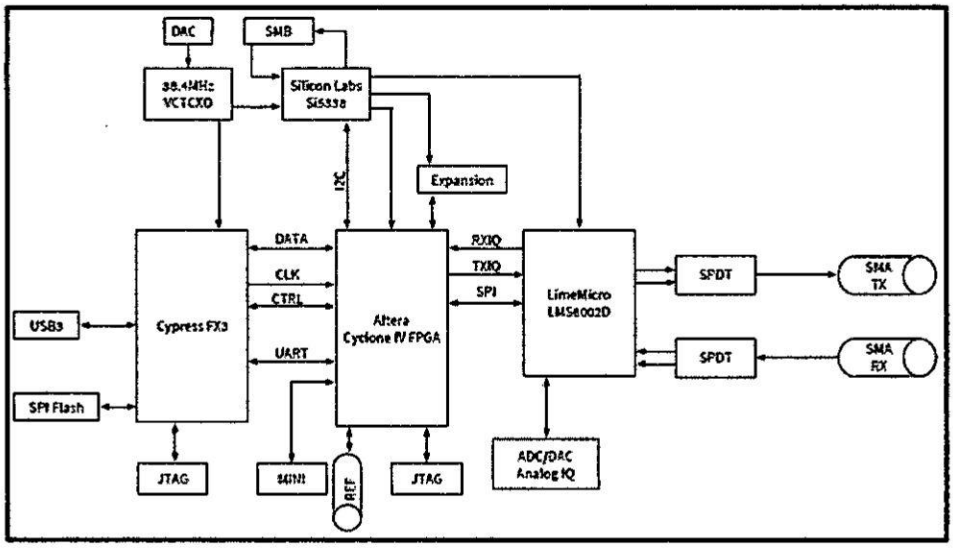
El B210 tiene un Spartan 6 LX150 FPGA con 150k elementos lógicos y la B200, tiene un FPGA LX75 con 75k elementos lógicos. El bladeRFx40 tiene un FPGA Cyclon IV con 40k elementos lógicos y el x115 tiene 115k elementos lógicos. El USRP B100 tiene un relativamente pequeño FPGA, con 25k elementos lógicos. El HackRF tiene una CPLD, pero sin FPGA, basándose principalmente en el microcontrolador a bordo.

El número de elementos lógicos se correlaciona directamente con lo poderoso que la FPGA es, así que obviamente cuantos más mejor. La ventaja de un FPGA es que el procesamiento se puede realizar en paralelo, en lugar de en serie. La desventaja es que a menudo los FPGAs tienen un reloj más lento que los microcontroladores, y los diseños son creados por el usuario y puede que no sea tan eficiente si el creador no está bien versado en el HDL.

Finalmente se seleccionó el hardware SDR bladeRF, debido a las ventajas que presenta con respecto a las otras tarjetas analizadas en esta sección.

FIGURA N° 4.13

ARQUITECTURA DEL HARDWARE DE UNA TARJETA BLADERF



c. Nodo Maestro

Una vez definido el sistema SDR a utilizar el siguiente paso fue implementar el nodo maestro (ver Figura 4.14) que se encargó de realizar las peticiones recopiló la información enviada por el nodo esclavo.

El nodo maestro tiene como subsistema de RF un hardware SDR bladeRF x40 conectado por el puerto USB 3.0 a una laptop con las siguientes características:

Procesador: Intel Core i7-3630QM de tercera generación a 2,24 GHz, hasta 3,40 GHz con tecnología Turbo Boost

- ✓ Memoria RAM: SDRAM DDR3 de 8 GB (2 DIMM)
- ✓ Sistema Operativo: Ubuntu 16.04
- ✓ Disco Duro: 1 TB 7200 RPM

Y al cuál se le instaló GNU Radio como se indican en el Anexo B.

FIGURA N° 4.14

ARQUITECTURA DEL NODO MAESTRO

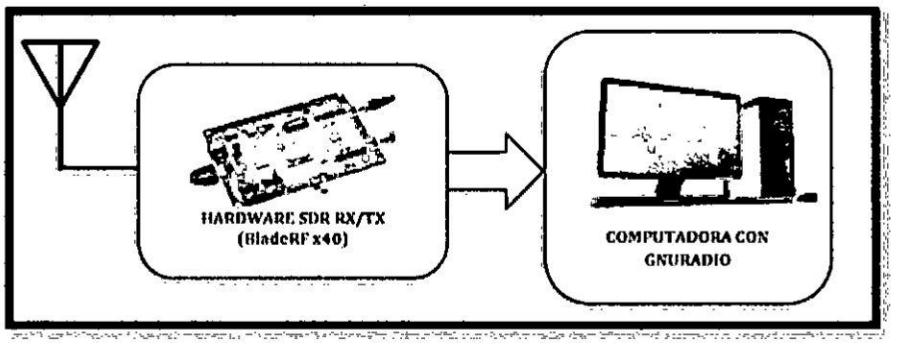
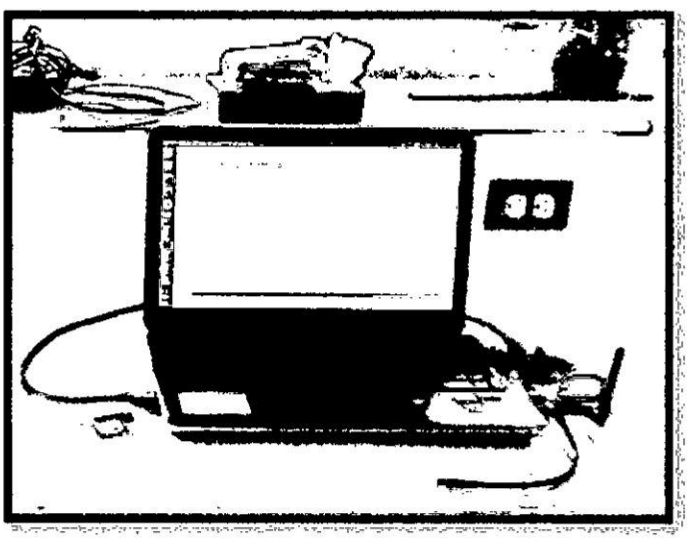


FIGURA N° 4.15

ESTACION BASE ENCARGADA DE RECIBIR DATOS



d. Nodo Esclavo

Actualmente las radios definidas por Software necesitan de un computador mediante el cual se programa la radio y se ingresan los datos que deberá

transmitir y/o reciben los datos en la sección de RX para su posterior procesamiento y almacenamiento.

Es por ello que para el Nodo esclavo que debe ser de pequeñas dimensiones se escogió un computador de placa reducida (Single Board Computer o SBC) Raspberry Pi 2 modelo B (ver Figura N° 4.16), que tiene la capacidad para instalar un SO compatible con GNU Radio y los drivers para manejar la tarjeta BladeRF, a continuación se presentan sus características:

- Un CPU ARM Cortex-A7 quad-core a 900MHz
- 1GB RAM
- 4 puertos USB
- 40 pines GPIO
- 1 puerto Full HDMI
- Puerto Ethernet
- Jack 3.5mm de audio combinado y video compuesto
- Interfaz para cámara (CSI)
- Interfaz para display (DSI)
- Ranura para tarjeta Micro SD
- Núcleo de gráficos VideoCore IV 3D

FIGURA N° 4.16

RASPBERRY PI 2 MODEL B

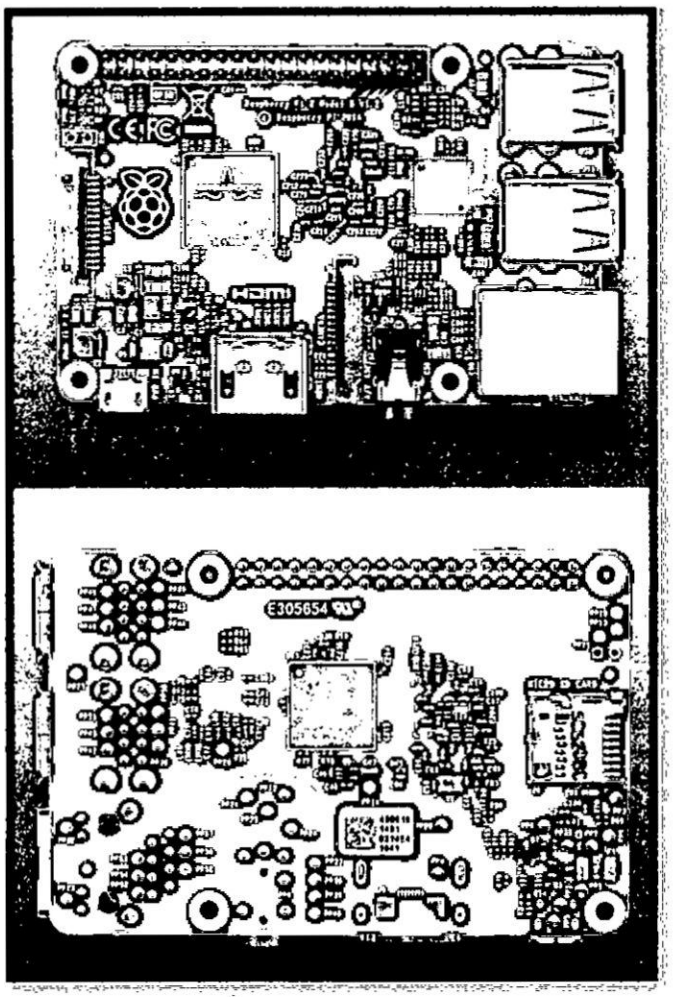
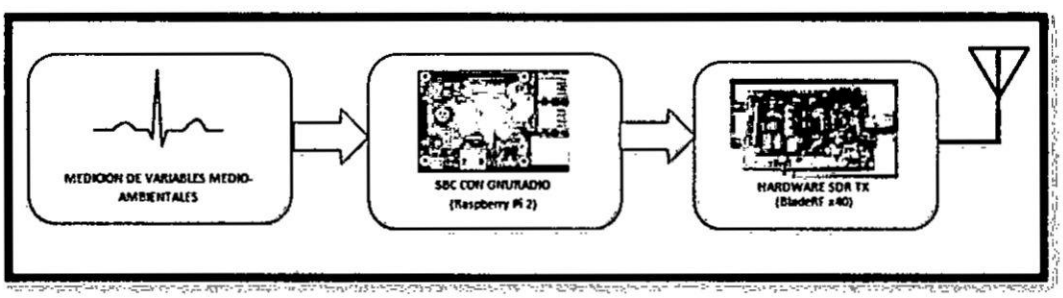


FIGURA N° 4.17

ARQUITECTURA DEL NODO ESCLAVO



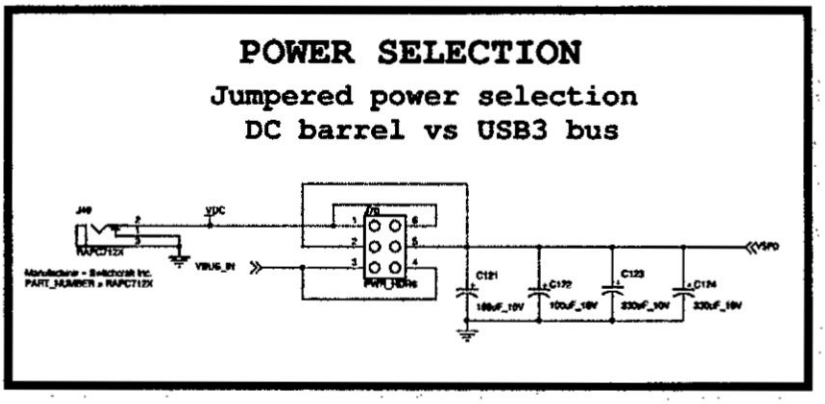
Para usar el BladeRF x40 con el Raspberry Pi 2, se debe usar una fuente externa como alimentación para el BladeRF, y no alimentar la tarjeta mediante el puerto USB, debido a que el bladeRF requiere una tensión de 5V, que debe tener una capacidad de suministro de 1,5 a 2A, corriente que el Raspberry Pi 2 no puede otorgar a través de sus puertos usb. Para que la tarjeta bladeRF se alimente con una fuente DC externa, se debe cambiar de posición unos jumpers que se detallan en la Figura N° 4.18 y Figura N° 4.19.

FIGURA N° 4.18

UBICACION DE LOS JUMPERS PARA USO DE FUENTE EXTERNA

FIGURA N° 4.19

CONFIGURACIÓN DEL BLADERF X40 PARA SU ALIMENTACIÓN CON FUENTE EXTERNA.



La fuente externa usada para alimentar el nodo sensor fue una batería de plomo ácido de 12V-7AH, con módulos Step Down LM2596 para conseguir voltajes de 5V necesarios para alimentar el bladeRF y el Raspberry Pi 2.

El módulo LM2596 permite tener un voltaje regulado a partir de una fuente de alimentación con un voltaje mayor, el Regulador DC-DC Step Down LM2596 es un circuito integrado monolítico adecuado para el diseño fácil y conveniente de una fuente de conmutación tipo buck. Es capaz de conducir una corriente de hasta 3A. Maneja una carga con excelente regulación de línea y bajo voltaje de rizado, debido a que es una fuente de alimentación conmutada, así que su eficiencia es significativamente mayor en comparación con los populares reguladores lineales de tres terminales, especialmente con tensiones de entrada superiores.

Características del módulo Step Down:

- Basada en el regulador LM2596, salida entre 1,5 y 35Vdc
- Voltaje de entrada: 4.5-40V
- Voltaje de salida: 1.5-35V (Ajustable)
- Corriente de salida: Máxima 3A
- Dimensiones: 43*20*14mm
- Frecuencia de switching: 150 KHz

FIGURA N° 4.20

MÓDULOS STEP DOWN LM2596S ENCARGADOS DE ALIMENTAR AL RASPBERRY PI 2 Y BLADE RF X40

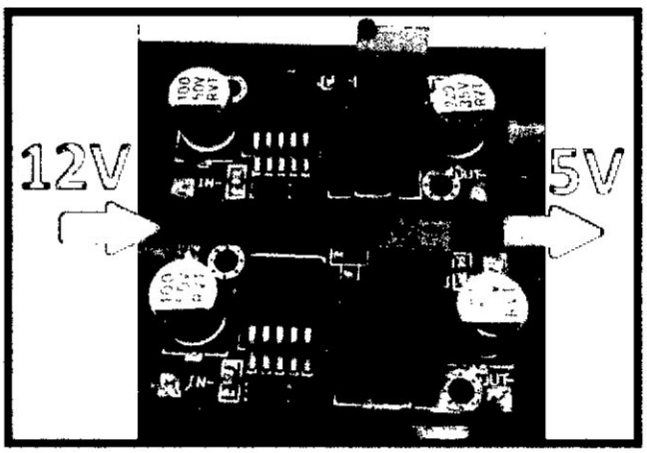
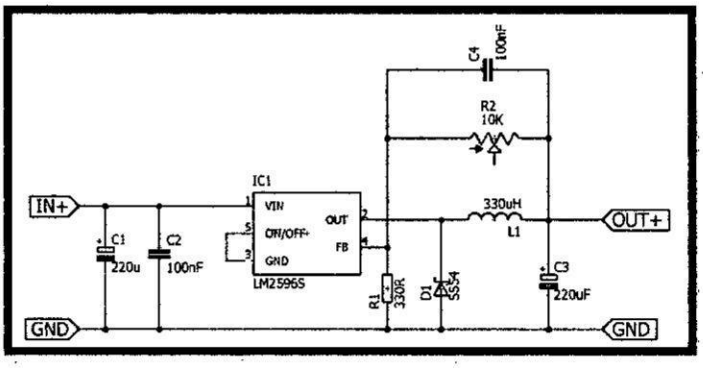


FIGURA N° 4.21

CIRCUITO ESQUEMÁTICO DEL MÓDULO STEP DOWN LM2596S



El nodo sensor también cuenta con una tarjeta microcontroladora Arduino Pro Mini, que usa un atmega328p a 8MHz y 5V (ver Figura N° 4.22), encargado de adquirir las señales analógicas de los sensores y estructurar un frame, que finalmente se enviará al nodo Maestro mediante sus pines de comunicación serial, a una velocidad de 115 200 baudios, cada vez que el Raspberry pi, hace la petición enviando el carácter "a", esto se puede ver claramente en el diagrama de flujos del algoritmo grabado en el microcontrolador, expuesto en la sección 4.2.3.a.

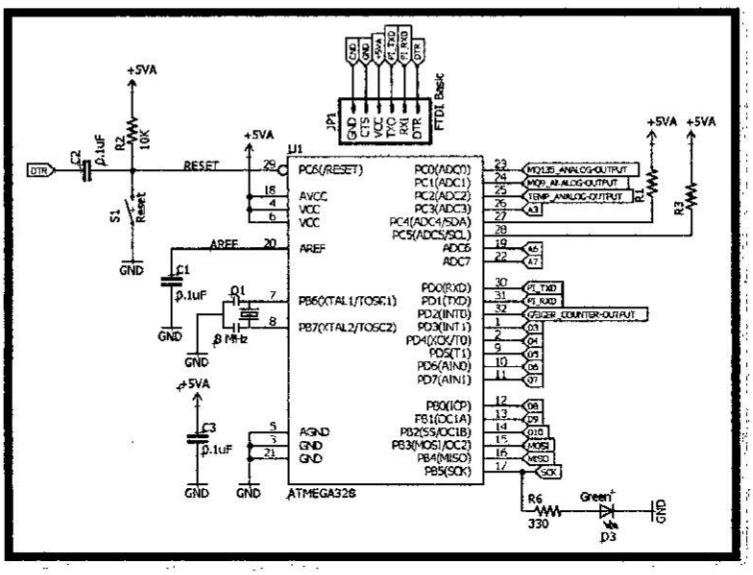
A la tarjeta Arduino Pro Mini, se le retiro el regulador de voltaje MIC5205, con la finalidad que la tarjeta solo tenga los recursos mínimos para su funcionamiento y obtener un consumo de corriente mínimo, ya que el Raspberry Pi 2, ya cuenta con niveles de 5V regulados a través de sus pines GPIO 2 y 4.

Características de la tarjeta Arduino Pro Mini:

- Microcontrolador ATmega328
- Voltaje de operación 5V
- 14 pines de entrada/salida digital (de las cuales 6 se puede usar como salidas PWM)
- 6 pines de entrada analógica
- Corriente DC por pin de entrada/salida: 40 mA
- Memoria Flash de 16 KB (de los cuales 2 KB son usados para el bootloader)
- 1 KB de SRAM 1 KB
- EEPROM de 512 bytes
- Velocidad de reloj de 8 MHz

FIGURA N° 4.22

DIAGRAMA ESQUEMÁTICO DE LA TARJETA ARDUINO PRO MINI MODIFICADA



Los pines utilizados para los sensores son:

- Analog 0 (pin 23): Para la salida análoga del sensor de gas MQ 135
- Analog 1 (pin 24): Para la salida análoga del sensor de gas MQ 9
- Analog 2 (pin 25): Para el valor analógico del sensor LM35
- External Interrupt 0 (pin 32): Para los pulsos emitidos por el Contador Geiger Müller.

Los cuales se detallarán en la sección 4.2.2.e Sensores, de ésta tesis.

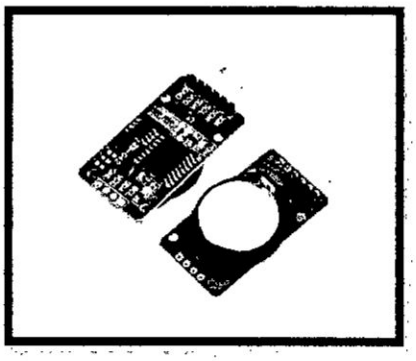
Los pines utilizados para la comunicación serial con el Raspberry Pi 2, son:

- RXD (Pin 30): Hacia el pin PI_TXD (Pin 8 GPIO)
- TXD (Pin 31): Hacia el pin PI_RXD (Pin 10 GPIO)

En la tarjeta de sensores también se agregó un módulo RTC DS3231, ya que el computador de placa reducida (Single Board Computer o SBC) Raspberry Pi 2 no cuenta con uno, y es necesario para temas de estadísticas almacenar los datos con fechas y horas.

FIGURA N° 4.23

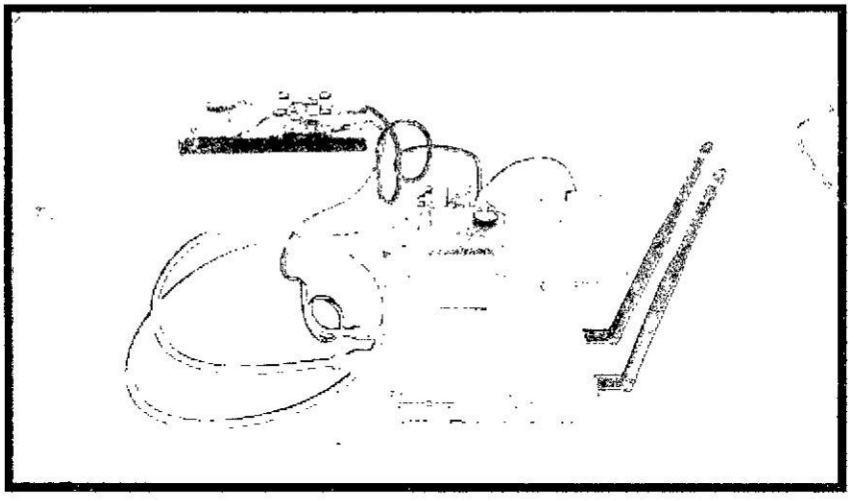
MÓDULO RTC DS3231



El sistema completo e implementado se puede apreciar en la Figura N° 4.24.

FIGURA N° 4.24

NODO SENSOR ENCARGADO DE ENVIAR LOS DATOS ADQUIRIDOS POR LOS SENSORES MEDIO-AMBIENTALES



e. Sensores

El nodo sensor estuvo compuesto por sensores medio-ambientales, que fueron integrados en una tarjeta de tipo shield (ver Figura 4.24 y Figura 4.21) diseñado para el computador de placa reducida (Single Board Computer o SBC) Raspberry Pi 2, y conectada a los pines análogos de la tarjeta Arduino Pro Mini modificada, aprovechando sus pines ADC con 10bits de resolución.

FIGURA N° 4.25

DIAGRAMA PCB DEL SHIELD PARA LOS SENSORES

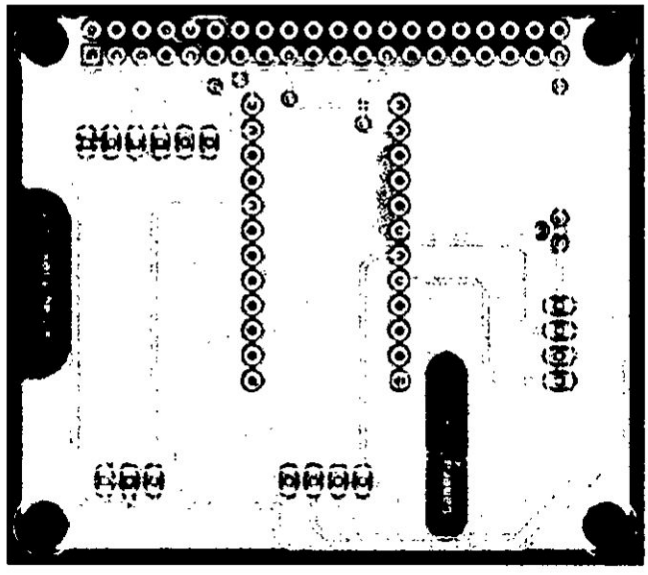


FIGURA N° 4.26

ETAPA DE SENSORES DESARROLLADA

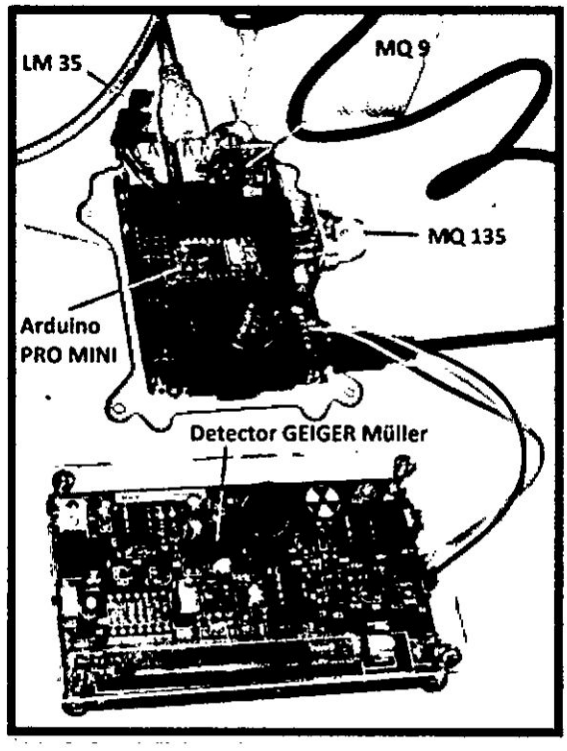
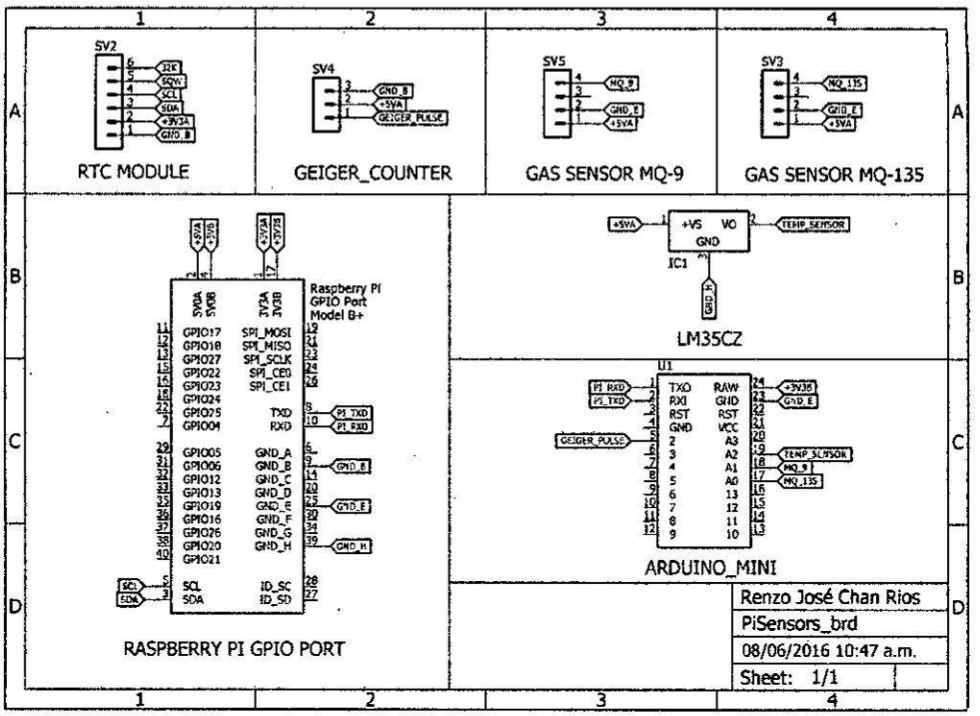


FIGURA N° 4.27

DIAGRAMA ESQUEMATICO DEL CIRCUITO PARA LOS SENSORES



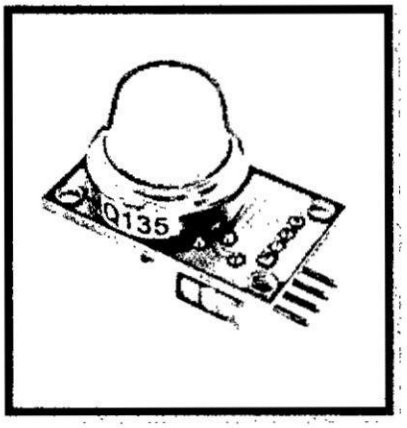
A continuación se describen los sensores utilizados para la recolección de datos:

- **Sensor de Gas MQ-135**

El sensor MQ-135 es usado en equipos de control de calidad del aire para edificios, oficinas y en detectores portátiles de contaminación del aire, ya que es sensible en similar proporción a los gases de NH3, NOx, alcohol, benceno, CO2, etc. (2)

FIGURA N° 4.28

MÓDULO SENSOR DE GAS MQ-135

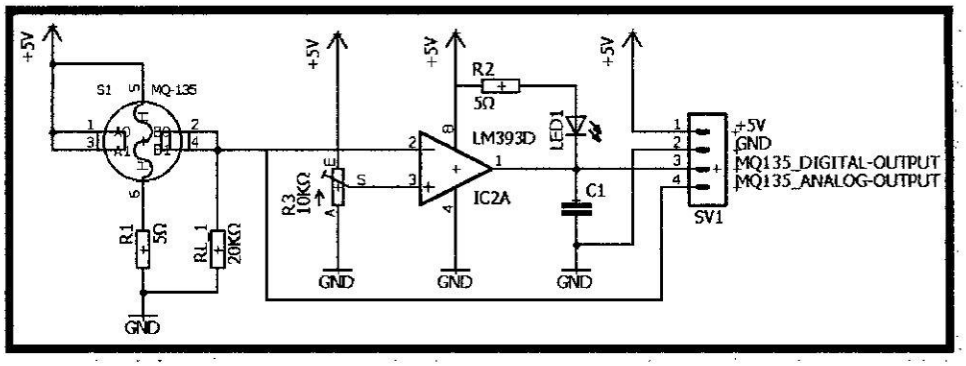


El material sensible del sensor de gas MQ135 es SnO2, que tiene una conductividad más baja en aire limpio. Cuando existe presencia de alguno de los gases mencionados anteriormente, la conductividad del sensor es más alta variando con el aumento de la concentración de gas. Su alimentación es de 5V±0.1 DC, y se recomienda trabajar con una resistencia de 20KΩ en su salida.

Como se mencionó anteriormente éste sensor estuvo conectado al pin Análogo A0 del atmega328p, mediante el cual se adquiere y se realiza un filtro promedio de 1000 muestras para disminuir el ruido.

FIGURA N° 4.29

DIAGRAMA ESQUEMATICO DEL SENSOR MQ135



Para obtener valores que sirvieron como referencia para establecer una condición normal de ambiente, el sensor se dejó alimentando por 24 Horas en un ambiente libre de gases a los cuales es sensible, representando una condición normal de ambiente, obteniendo un voltaje de 0.208 V.

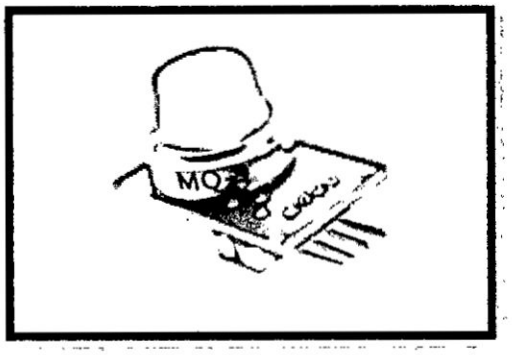
La referencia (2) muestra que el valor promedio de concentración de CO2 en la atmósfera fue de 398.55 ppm, asumiendo este valor como referencia cuando el sensor entrega 0.208 V en condiciones normales.

- **Sensor de Gas MQ-9**

El sensor MQ-9 es usado en la detección de gas de monóxido de carbono, CH4 y GLP, al igual que el sensor MQ-135, éste es usado en equipos de control de calidad del aire para edificios, oficinas y en detectores portátiles de contaminación del aire. Su alimentación es de 5V±0.1 DC y se recomienda disponer de una resistencia de 20KΩ a su salida (3).

FIGURA N° 4.30

MÓDULO SENSOR DE GAS MQ-9

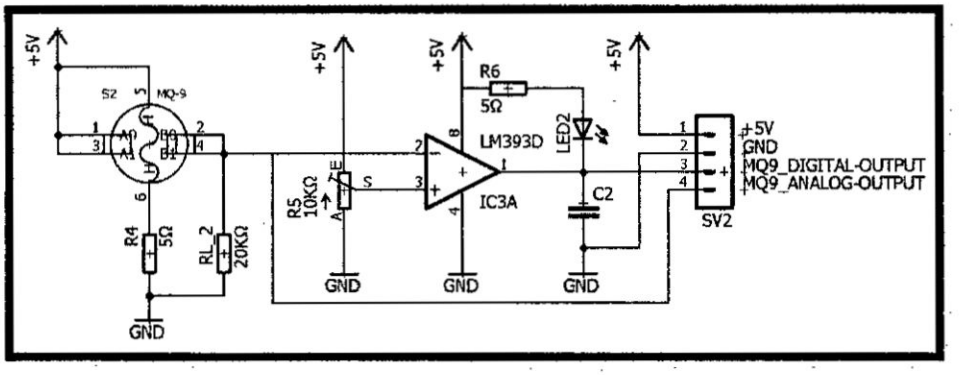


Se realizaron pruebas de respuesta del sensor, obteniendo en una condición normal del ambiente 0.75V, y con alcohol isopropílico de 3.50V en promedio.

El sensor de gas MQ 9 estuvo conectado al pin Análogo A1 del atmega328p, mediante el cual se adquiere y se realiza un filtro promedio de 1000 muestras para eliminar el ruido.

FIGURA N° 4.31

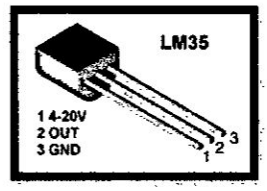
DIAGRAMA ESQUEMATICO DEL SENSOR MQ9



- **Sensor de Temperatura LM35.**

FIGURA N° 4.32

SENSOR DE TEMPERATURA LM35, ENCAPSULADO TO-92

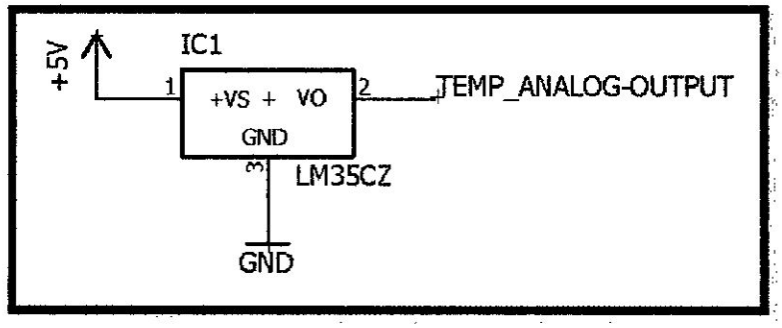


El sensor LM35 es un sensor analógico que proporciona un voltaje de salida linealmente proporcional a la temperatura en grados centígrados. Para su funcionamiento se puede alimentar con voltajes de 4 a 30V, teniendo un rango de sensado de temperatura de -55°C a +150°C, siendo su factor de escala lineal de +10mv/°C.

El sensor de temperatura LM35, como se menciona en una sección anterior, está conectado al pin analógico A2.

FIGURA N° 4.33

DIAGRAMA ESQUEMATICO DEL SENSOR LM35



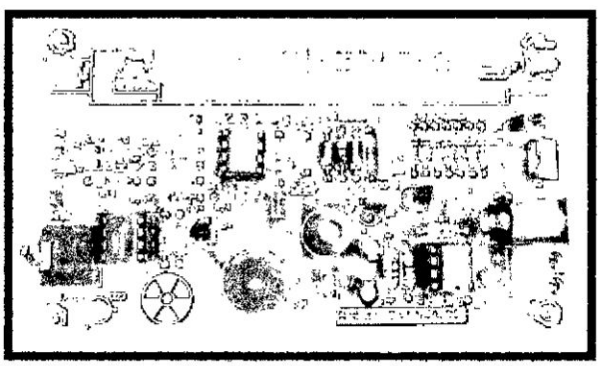
- **Contador Geiger Müller**

Un contador Geiger es un instrumento que permite medir la radiactividad de un objeto o lugar. Es un detector de partículas y de radiaciones ionizantes.

Está formado, normalmente, por un tubo con un fino hilo metálico a lo largo de su centro. El espacio entre ellos está aislado y relleno de un gas, y con el hilo a unos 300-1000 voltios con respecto al tubo.

FIGURA N° 4.34

TUBO GEIGER MÜLLER CON CIRCUITO DE ACONDICIONAMIENTO



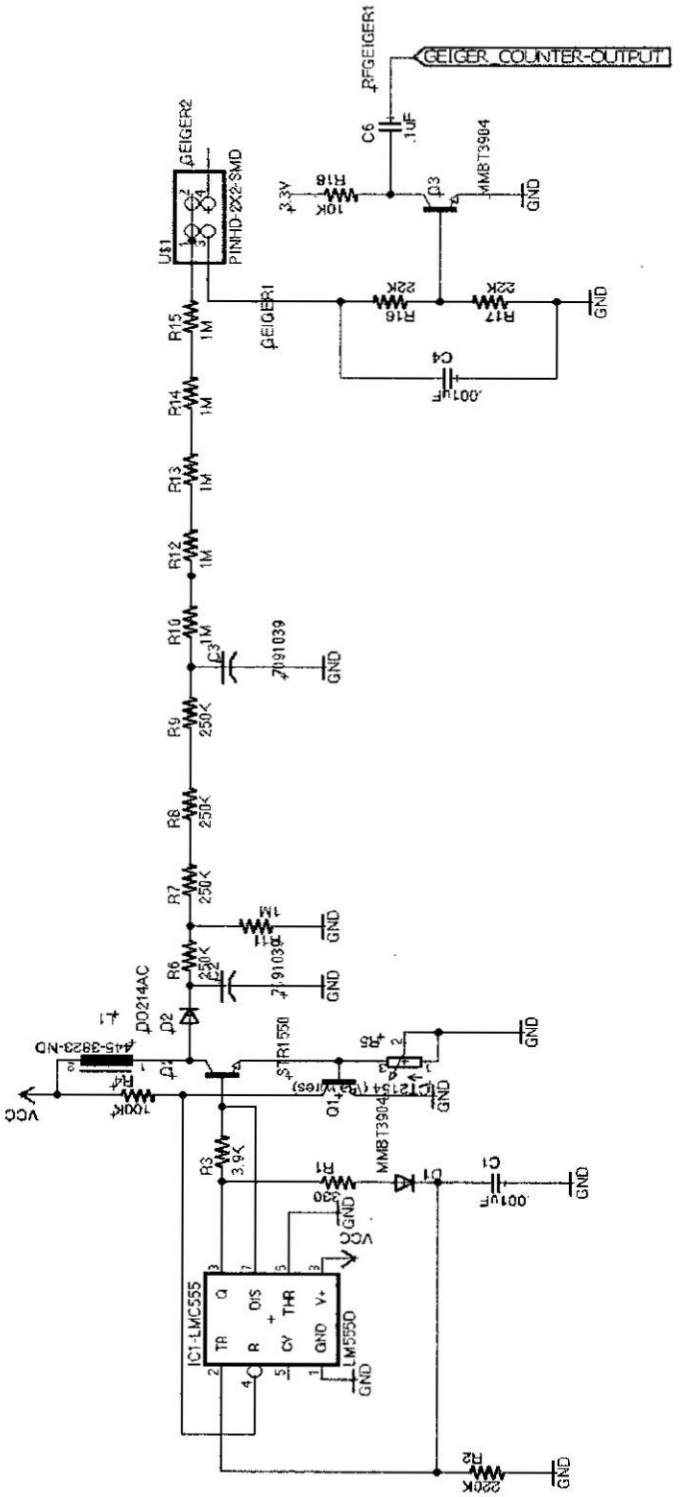
Cuando un ion o un electrón penetran en el tubo (o bien se libera un electrón de su pared por efecto de los rayos X o gamma), se desprenden electrones de los átomos del gas que rellena el tubo. Debido al voltaje positivo del hilo central, son atraídos hacia él, y al hacer esto gana energía, colisionan con los átomos del gas y liberan más electrones, hasta que el proceso se convierte en una avalancha que produce un pulso de corriente detectable. Relleno de un gas adecuado, el flujo de electricidad se para por sí mismo o incluso el circuito eléctrico puede ayudar a pararlo.

Al instrumento se le llama un "contador" debido a que cada partícula que pasa por él produce un pulso idéntico, permitiendo contar las partículas (normalmente de forma electrónica) pero sin proporcionar datos acerca del tipo de radiación o sobre su energía (excepto que tienen energía suficiente como para penetrar las paredes del contador). Los contadores de Van Allen estaban hechos de un metal fino con conexiones aisladas en sus extremos.

El Diseño que se muestra en la Figura 4.35 es el circuito de acondicionamiento del tubo Geiger Müller, que tiene la característica de no usar un transformador para la generación de alta tensión. Si no simplemente un inductor al cual le llega voltaje PWM a través del transistor, cargando y permitiendo generar el alto voltaje, este tipo de circuitos se conoce como Boost converter.

FIGURA N° 4.35

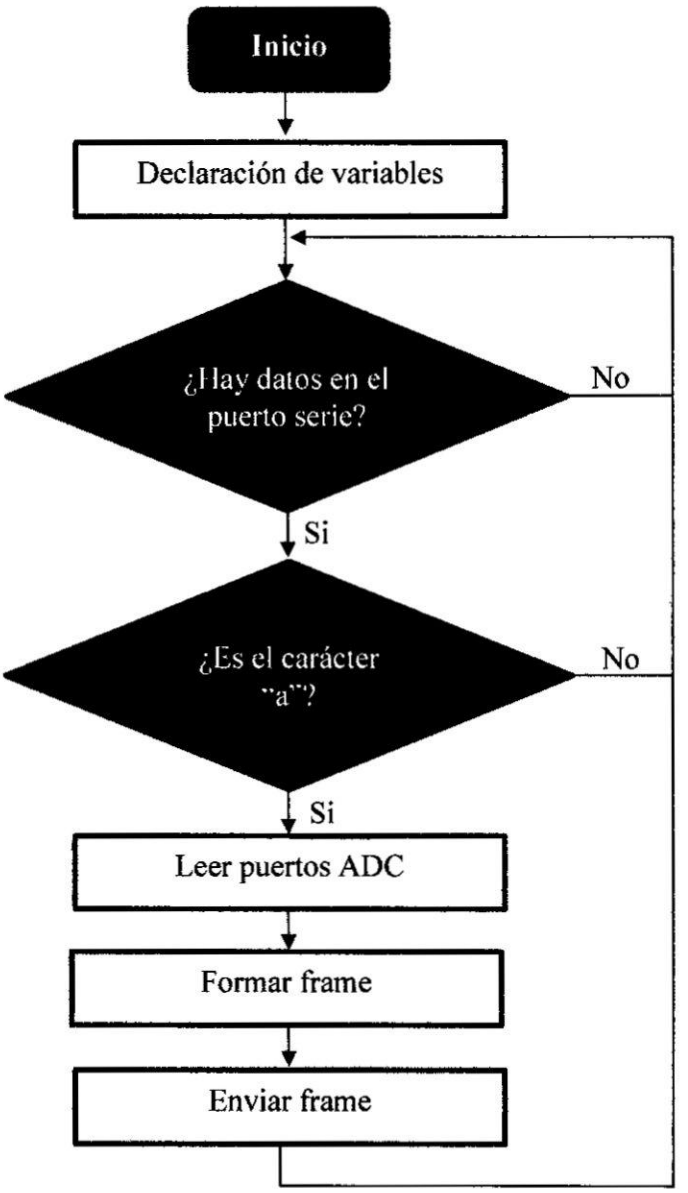
DIAGRAMA ESQUEMÁTICO DEL GEIGER COUNTER



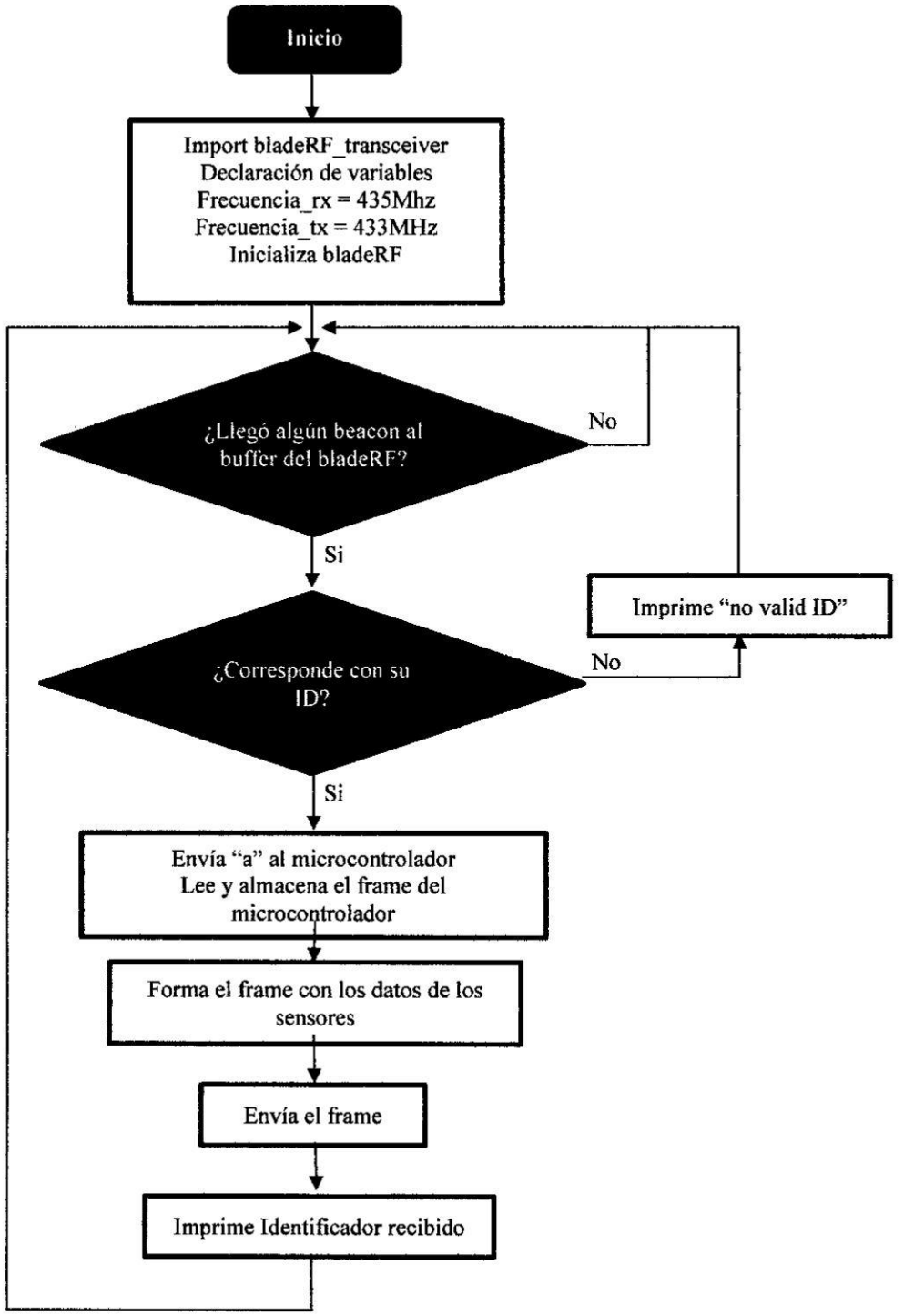
4.2.3. Diseño e implementación del Software del Sistema

a. Diagrama de Flujos

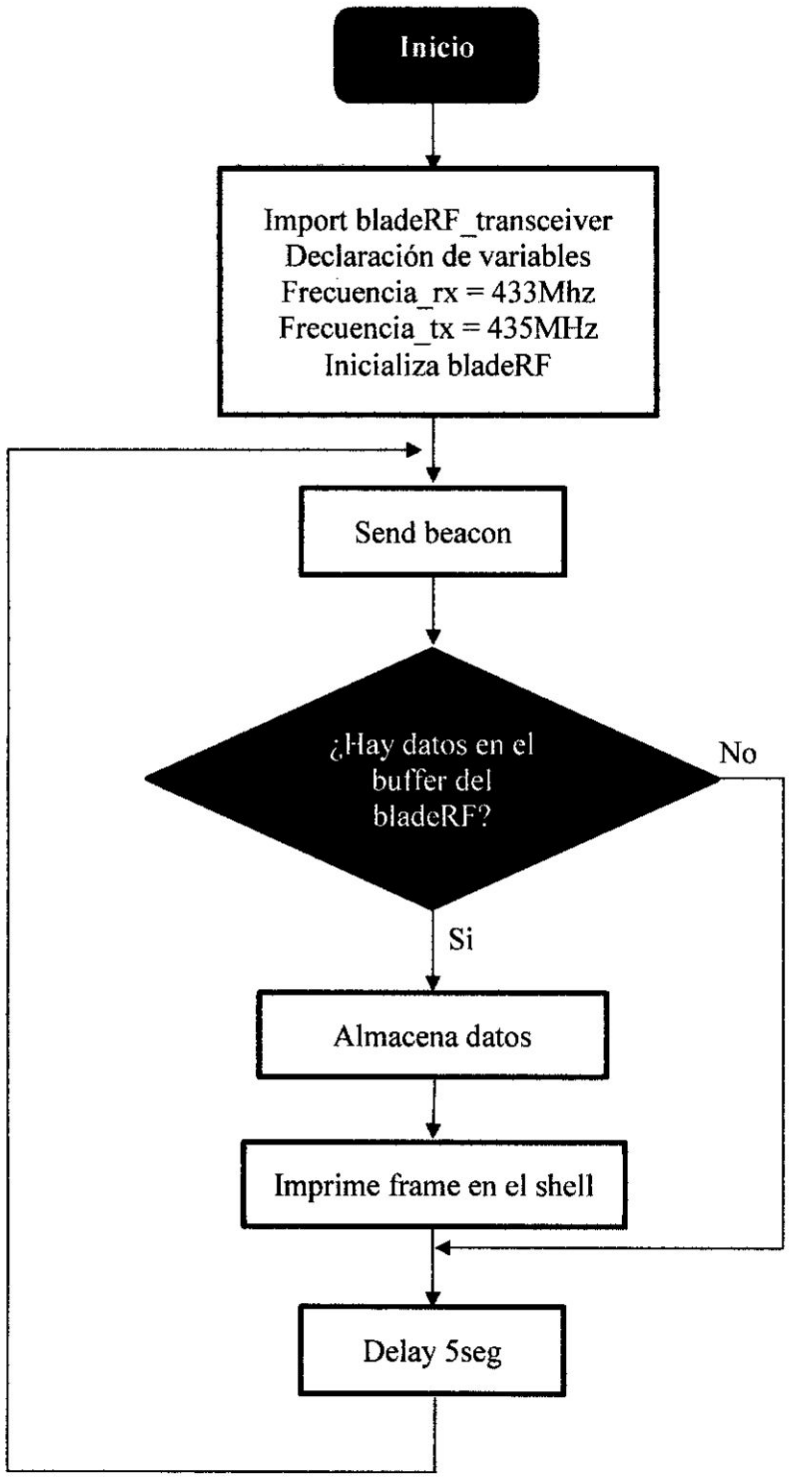
- Diagrama de flujos del algoritmo grabado en el microcontrolador:



- Diagrama de flujos del nodo Esclavo:



- Diagrama de flujos del nodo Maestro:



b. Algoritmo de Transmisión y Recepción usando BladeRF y GNU Radio

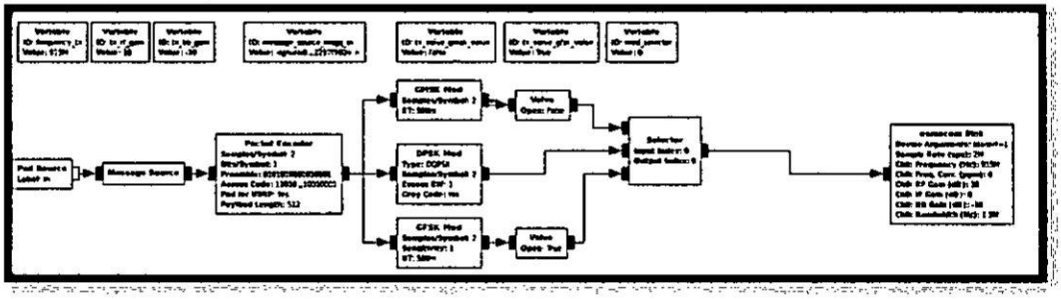
A continuación se muestra el diagrama de bloques de la configuración del bladeRF como transceptor usando GNU Radio Companion (GRC), este diagrama permite que nuestro bladeRF envíe y reciba datos de forma paralela. Al compilar el diagrama de bloques y generar un script de python en GRC, estamos generando el módulo bladeRF_transceiver.py, que tiene a la clase "bladeRF_transceiver" con diferentes métodos y variables para ser usados en nuestros scripts de python principales.

En la etapa de Transmisión se cuenta con bloque "Message Source", responsable de poder ingresar frames a GNU Radio Companion, para luego ser empaquetado y codificado usando el bloque "Packet Encoder", una vez empaquetado y codificado se procedió a realizar la modulación digital, para ésta tesis se desarrollaron 3 tipos de modulación GMSK, BPSK y GFSK, las cuales se usan en diferentes sistema de comunicación en la actualidad.

Luego se tiene un bloque selector, el cual se controla desde la interfaz gráfica y permitió al usuario seleccionar una de las modulaciones mencionadas anteriormente, finalmente se encuentra el bloque Osmocom Sink, encargado de enviar los datos de tipo complejo que se forman dentro de GNU Radio hacia el sistema de RF, BladeRF x40.

FIGURA N° 4.36

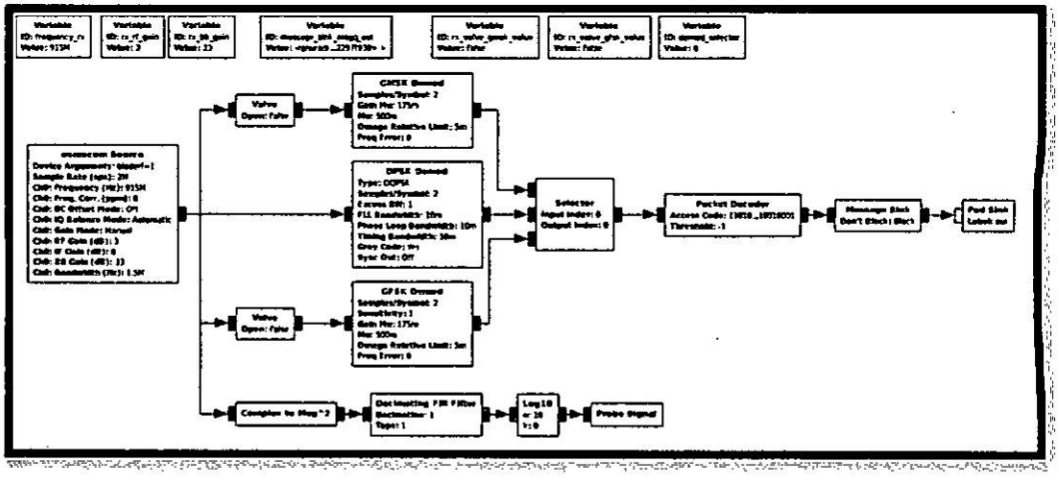
DIAGRAMA DE BLOQUES DE LA ETAPA DE TRANSMISIÓN.



En la etapa de recepción se contó inicialmente con el bloque "Osmocom Source" (ver Figura N° 38) que permitió manipular los datos recibidos por la tarjeta BladeRF x40, para luego ser demodulada según la modulación escogida por el usuario, luego se desempaquetaron los datos recibidos para finalmente ser enviados a un bloque Message Sink para su manipulación en Python.

FIGURA N° 4.37

DIAGRAMA DE BLOQUES DE LA ETAPA DE RECEPCIÓN.

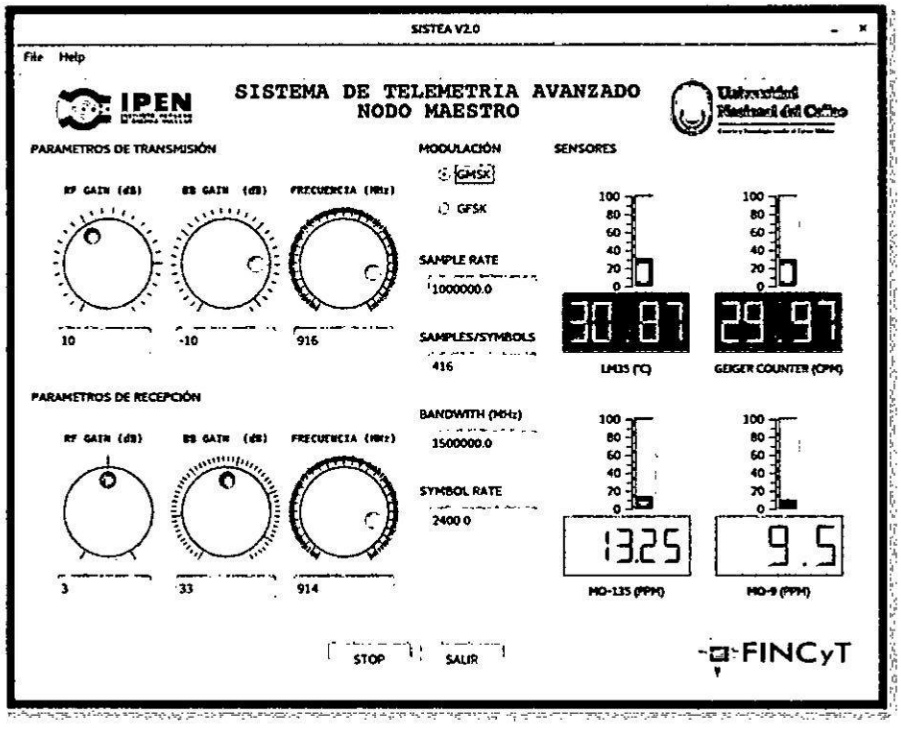


c. Interfaz Gráfica de Usuario del Nodo Maestro

Como se mencionó anteriormente las interfaces gráficas para el nodo Maestro y Nodo esclavo fueron desarrolladas con Python 2.7 y la biblioteca para interfaz gráfica PyQT.

FIGURA N° 4.38

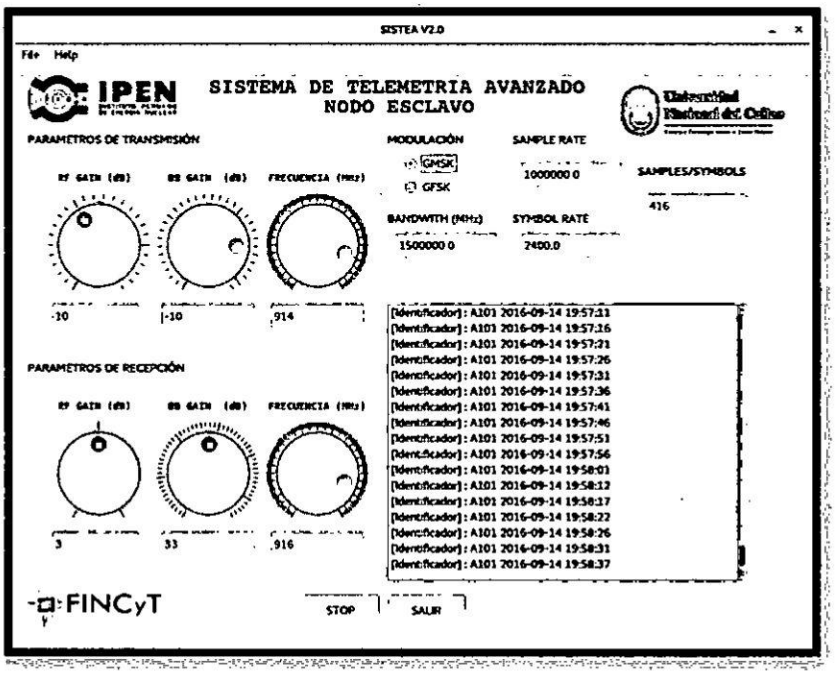
INTERFAZ GRAFICA DEL NODO MAESTRO



d. **Interfaz Gráfica de Usuario del Nodo Esclavo**

FIGURA N° 4.39

INTERFAZ GRÁFICA DEL NODO ESCLAVO



4.3. Técnicas e instrumentos de recolección de datos

El sistema de telemetría se probó en diferentes escenarios, con obstáculos y líneas de vista, a diferentes frecuencias y técnicas de modulación para así comprobar la robustez del mismo. Todos los datos adquiridos se almacenaron en un archivo de registro para su posterior análisis y procesamiento.

Para todas estas pruebas se utilizó como instrumentos:

- 1 Laptop con sistema operativo Ubuntu 14.04 LTS y el software GNU Radio 3.7.8.

- 1 Raspberry Pi2, con sistema operativo Raspbian (basado en Debian 8.0) y software GNU Radio 3.7.8.
- 2 hardware SDR BladeRF x40 con antenas para diferentes frecuencias.
- 1 Smartphone.
- Un modem inalámbrico TP-Link modelo TL-MR3040.

4.4. Procesamiento estadístico y análisis de datos

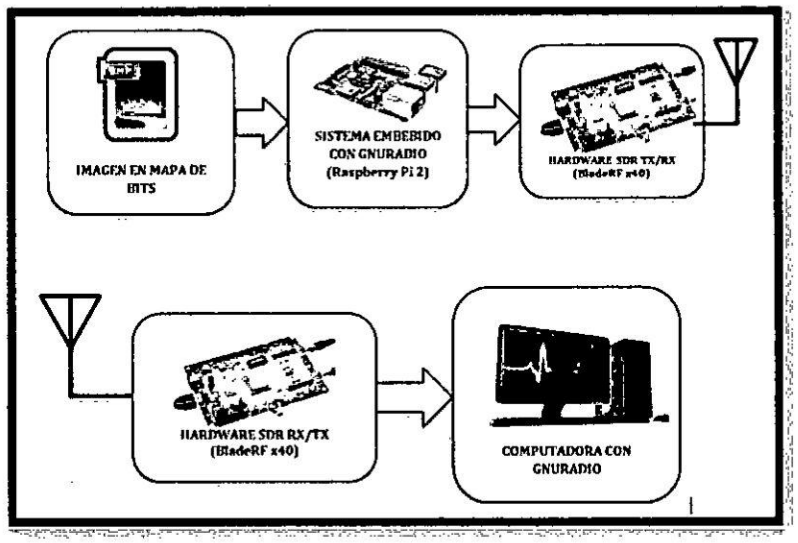
Para el análisis estadístico de los datos se implementó un algoritmo para el cálculo del BER (tasa de error de bits), basado en los bloques funcionales con los que cuenta GNU Radio Companion, el cual envía una imagen en mapa de bits hacia la estación base, para luego analizar cuantos datos erróneos hubo.

Los resultados se pueden observar en tablas que presentan de forma clara los resultados de cada prueba o etapa, incluyendo cada uno sus respectivos comentarios y conclusiones.

En esta etapa del proceso de investigación se procede a racionalizar los datos colectados a fin de explicar e interpretar las posibles relaciones que expresan las variables estudiadas.

FIGURA N° 4.40

ESQUEMA DEL LOGARITMO PARA LA DETECCION DE TASA DE ERROR DE BITS



También se desarrolló un banco de pruebas (ver Figura N° 4.25) en el cuál se sometió el sistema a diferentes tipos de modulación digital, y analizar la tasa de error de bits, obteniendo la siguiente tabla.

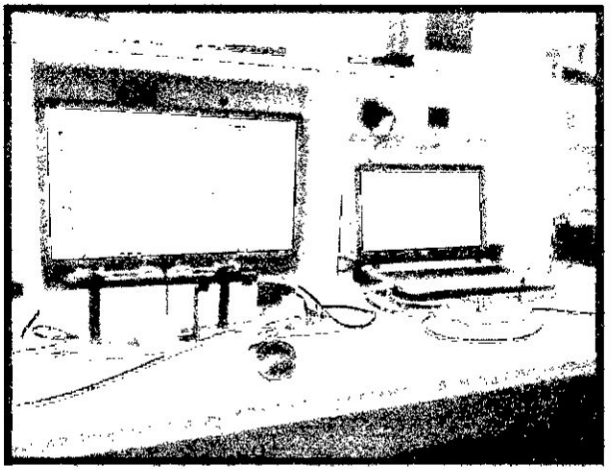
TABLA N° 4.2

TASA DE ERROR DE BITS CON DIFERENTES TIPOS DE MODULACION

Modulación	*Eb/N0*	BER
BPSK	10	$4.2x10^{-6}$
	20	$0.9x10^{-6}$
QPSK	10	$10.17x10^{-6}$
	20	$1.16x10^{-6}$
8-PSK	10	$1047x10^{-6}$
	20	$17.96x10^{-6}$
GMSK	10	$4.02x10^{-6}$
	20	$1.05x10^{-6}$

FIGURA N° 4.41

PRUEBA DEL NODO SENSOR Y ESTACION BASE A 915MHZ EN EL LABORATORIO DE DESARROLLO ELECTRÓNICO

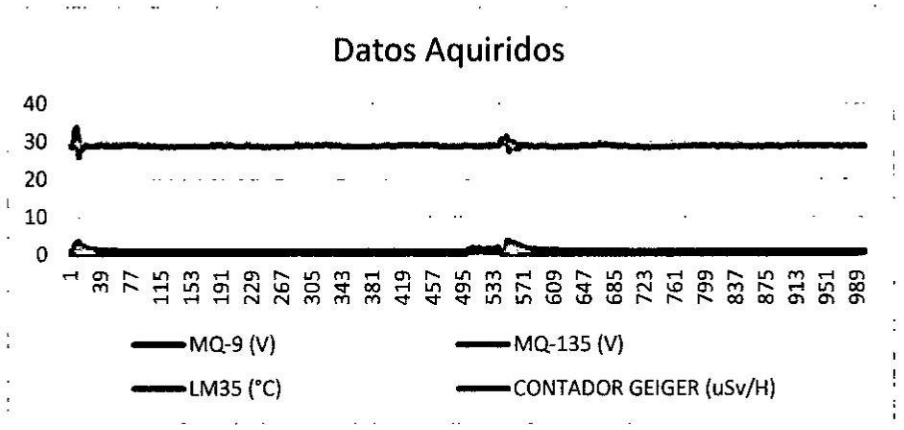


Luego se pasó a analizar los datos almacenados por la Estación Base, las cuales se enviaron cada 5 segundos y hasta acumular 1000 muestras.

Los pequeños picos percibidos en la gráfica corresponden a exposiciones de alcohol isopropílico para el caso de los sensores de gas y una muestra patrón de Co 60, para el contador Geiger.

GRÁFICO N° 4.1

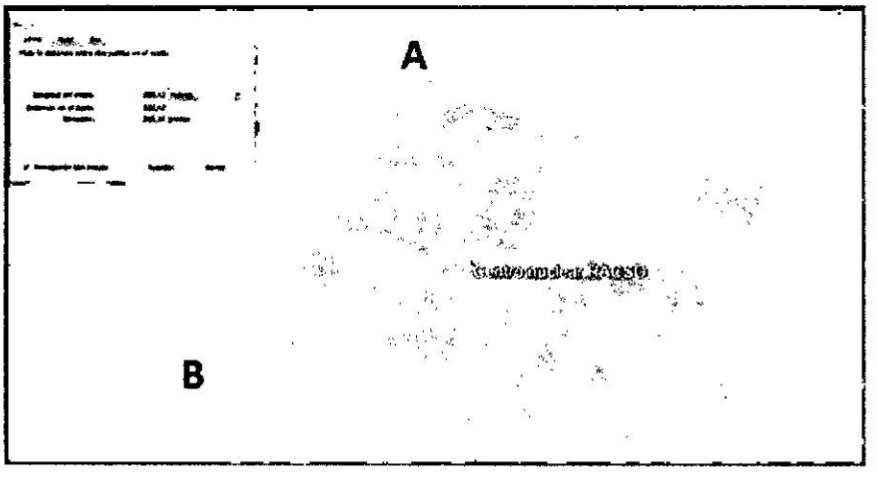
GRÁFICA DE MIL MUESTRAS ENVIADAS POR EL NODO SENSOR



Se hicieron pruebas a diferencias frecuencias (433MHz, 476MHz, 915MHz), dichas pruebas se realizaron dentro de las instalaciones del Centro Nuclear, para el procedimiento se seleccionaron dos puntos con línea de vista dentro de las 16 hectáreas, con una separación de 533 metros aproximadamente entre el punto A y B.

FIGURA N° 4.42

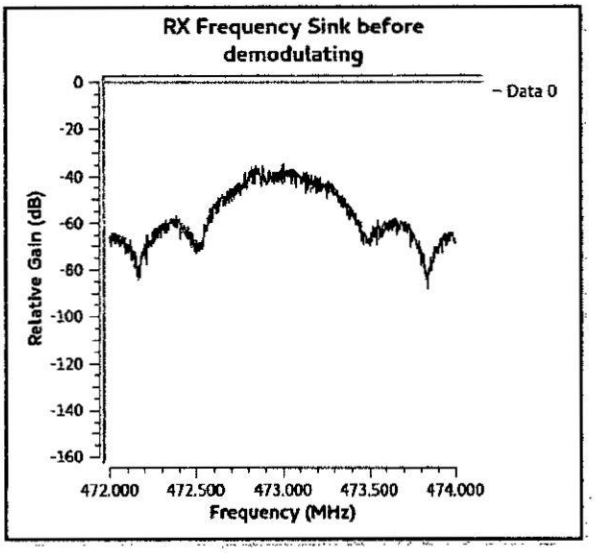
DISTANCIA CALCULADA CON EL PROGRAMA GOOGLE EARTH



Se posicionó el Nodo Sensor en el punto A y la Estación Base en el punto B, se procedió a la adquisición y visualización de los datos adquiridos. En la Figura 4.27 se observa el espectro de frecuencia de una modulación GMSK visualizada en la Estación Base para una frecuencia de 473MHz, se observa también una Ganancia pico Relativa de -40dB.

FIGURA N° 4.43

ESPECTRO DE FRECUENCIA VISUALIZADA EN LA ESTACIÓN BASE



Obteniendo distancias aproximadas de 80 a 90 m usando la frecuencia de 433MHz, sin tener en cuenta la zona de Fresnel, y una distancia de 533m a una altura de 4m para la zona de Fresnel. Para calcular la distancia de forma experimental se usó una aplicación para el SO Android que hace uso del GPS del dispositivo y puede calcular la distancia entre 2 puntos.

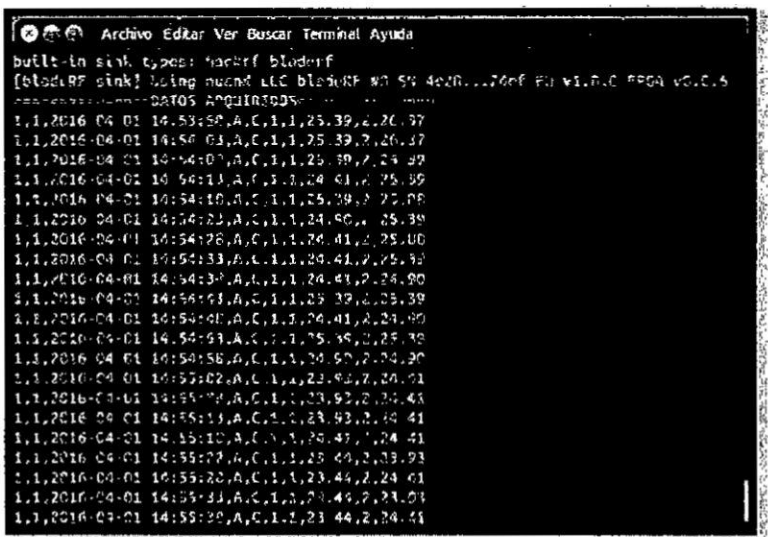
FIGURA N° 4.44

DISTANCIA MAXIMA ALCANZADA A 915MHZ

Para el caso de la Estación Base, podemos observar que en la consola se llegó a imprimir cada 5 segundos el frame que enviaba el Nodo Sensor con los valores de temperatura y con la estructura del frame de prueba definida.

FIGURA N° 4.45

CONSOLA LINUX CON LOS FRAMES RECIBIDOS EN LA ESTACIÓN BASE

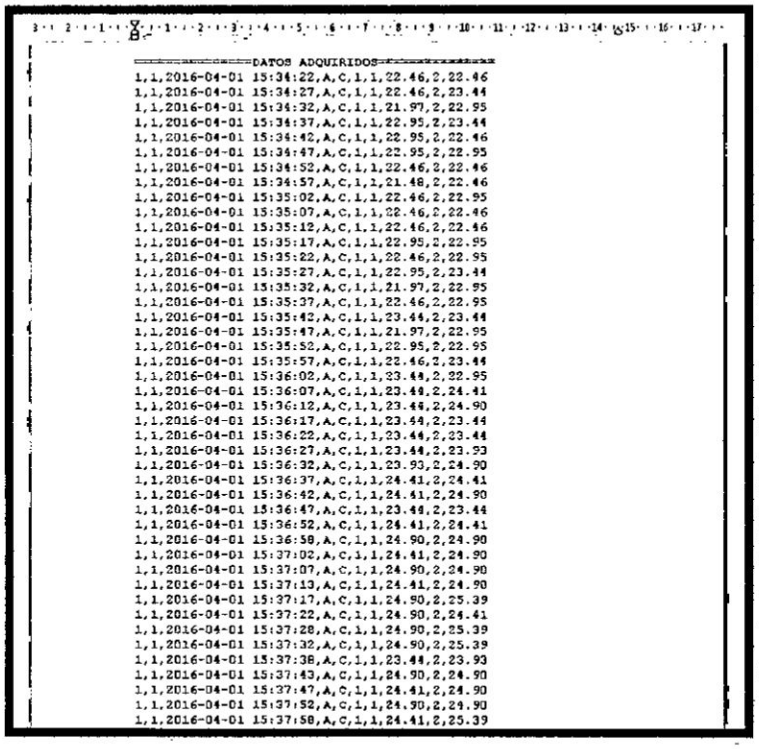


Para un análisis correcto de la probabilidad de error en el envío de paquetes, la Estación Base mientras iba recibiendo los mensajes los almacenaba en un archivo de texto, para su posterior procesamiento y análisis.

Usando un algoritmo simple de envío de paquetes de datos sin acuse de recibo o ACK, la probabilidad de error de paquetes fue de un 15%, lo cual se reduce a 1% usando un algoritmo de ACK, pero que a su vez retrasó la transmisión debido a que empezó a enviar una mayor cantidad de paquetes con la finalidad de obtener un acuse de recibo, lo cual añadió retardos en la transmisión.

FIGURA N° 4.46

ARCHIVO DE TEXTO GENERADO POR LA ESTACION BASE



```
==========DATOS ADQUIRIDOS==========
1,1,2016-04-01 15:34:22,A,C,1,1,22.46,2,22.46
1,1,2016-04-01 15:34:27,A,C,1,1,22.46,2,23.44
1,1,2016-04-01 15:34:32,A,C,1,1,21.97,2,22.95
1,1,2016-04-01 15:34:37,A,C,1,1,22.95,2,23.44
1,1,2016-04-01 15:34:42,A,C,1,1,22.95,2,22.46
1,1,2016-04-01 15:34:47,A,C,1,1,22.95,2,22.95
1,1,2016-04-01 15:34:52,A,C,1,1,22.46,2,22.46
1,1,2016-04-01 15:34:57,A,C,1,1,21.48,2,22.46
1,1,2016-04-01 15:35:02,A,C,1,1,22.46,2,22.95
1,1,2016-04-01 15:35:07,A,C,1,1,22.46,2,22.46
1,1,2016-04-01 15:35:12,A,C,1,1,22.46,2,22.46
1,1,2016-04-01 15:35:17,A,C,1,1,22.95,2,22.95
1,1,2016-04-01 15:35:22,A,C,1,1,22.46,2,22.95
1,1,2016-04-01 15:35:27,A,C,1,1,22.95,2,23.44
1,1,2016-04-01 15:35:32,A,C,1,1,21.97,2,22.95
1,1,2016-04-01 15:35:37,A,C,1,1,22.46,2,22.95
1,1,2016-04-01 15:35:42,A,C,1,1,23.44,2,23.44
1,1,2016-04-01 15:35:47,A,C,1,1,21.97,2,22.95
1,1,2016-04-01 15:35:52,A,C,1,1,22.95,2,22.95
1,1,2016-04-01 15:35:57,A,C,1,1,22.46,2,23.44
1,1,2016-04-01 15:36:02,A,C,1,1,23.44,2,22.95
1,1,2016-04-01 15:36:07,A,C,1,1,23.44,2,24.41
1,1,2016-04-01 15:36:12,A,C,1,1,23.44,2,24.90
1,1,2016-04-01 15:36:17,A,C,1,1,23.44,2,23.44
1,1,2016-04-01 15:36:22,A,C,1,1,23.44,2,23.44
1,1,2016-04-01 15:36:27,A,C,1,1,23.44,2,23.93
1,1,2016-04-01 15:36:32,A,C,1,1,23.93,2,24.90
1,1,2016-04-01 15:36:37,A,C,1,1,24.41,2,24.41
1,1,2016-04-01 15:36:42,A,C,1,1,24.41,2,24.90
1,1,2016-04-01 15:36:47,A,C,1,1,23.44,2,23.44
1,1,2016-04-01 15:36:52,A,C,1,1,24.41,2,24.41
1,1,2016-04-01 15:36:58,A,C,1,1,24.90,2,24.90
1,1,2016-04-01 15:37:02,A,C,1,1,24.41,2,24.90
1,1,2016-04-01 15:37:07,A,C,1,1,24.90,2,24.90
1,1,2016-04-01 15:37:13,A,C,1,1,24.41,2,24.90
1,1,2016-04-01 15:37:17,A,C,1,1,24.90,2,25.39
1,1,2016-04-01 15:37:22,A,C,1,1,24.90,2,24.41
1,1,2016-04-01 15:37:28,A,C,1,1,24.90,2,25.39
1,1,2016-04-01 15:37:32,A,C,1,1,24.90,2,25.39
1,1,2016-04-01 15:37:38,A,C,1,1,23.44,2,23.93
1,1,2016-04-01 15:37:43,A,C,1,1,24.90,2,24.90
1,1,2016-04-01 15:37:47,A,C,1,1,24.41,2,24.90
1,1,2016-04-01 15:37:52,A,C,1,1,24.90,2,24.90
1,1,2016-04-01 15:37:58,A,C,1,1,24.41,2,25.39
```

CAPÍTULO V
RESULTADOS

A continuación se redactan los resultados obtenidos por el sistema de telemetría avanzado:

1. El Nodo Sensor usando un computador de placa reducida (Single Board Computer o SBC) Raspberry Pi 2, funcionó correctamente para el propósito del proyecto.
2. El microcontrolador atmega328p usado para los sensores de gases, temperatura y el contador Geiger de radiación, logró adquirir correctamente los valores de los sensores y generar el frame adecuado para el envío a la Estación Base.
3. El Nodo Sensor logró enviar los datos adquiridos por el microcontrolador, hacia la Estación Base, usando frecuencias libres para radioaficionados.
4. La Estación Base y el Nodo Sensor fueron capaces de enviar y recibir datos hasta una distancia 533.5 metros usando modulación GMSK y FSK.
5. La interfaz gráfica de usuario creada para el Nodo Sensor y La Estación base, permitió visualizar de una manera gráfica los datos que iban llegando y a su configurar adecuadamente los parámetros de transmisión RF.
6. La transmisión y el envío de datos pudo ser modificada rápidamente para garantizar una comunicación fluida entre el sistema y el equipo remoto, mediante las interfaces gráficas creadas.
7. Los resultados de la investigación indican el performance de nuestro sistema, demostrando así el potencial de las radios definidas por software para este tipo de aplicaciones.

CAPÍTULO VI
DISCUSIÓN DE RESULTADOS

6.1. Contrastación de hipótesis con los resultados

Una vez desarrollado el programa, se definieron los parámetros de estudio, que nos han permitido evaluar estadísticamente el comportamiento del sistema.

En primer lugar, se han realizado múltiples pruebas para ajustar los parámetros estudiados del sistema con referencia sobre el que se ha experimentado. Tras caracterizar la arquitectura del sistema, se han realizado múltiples simulaciones obteniendo resultados que han permitido comparar las prestaciones de los mecanismos de envío y recepción de datos en diferentes situaciones. Posteriormente se ha realizado un análisis detallado de los resultados obtenidas lo cual nos ha permitido proponer las nuevas técnicas adaptivas que permiten mantener siempre al sistema de comunicaciones en el punto óptimo de trabajo, con los valores más adecuados para los parámetros de transmisión.

A continuación se resumen brevemente los efectos que causan las variaciones en el entorno y el equipo usado en el rendimiento del sistema.

Al ser el núcleo del sistema una radio definida por software hace que la curva de aprendizaje para implementar diferentes tipos de modulación y técnicas de codificación, sea muy empinada debido a que se necesitan conocimientos intermedios de Radiofrecuencia, modulación análoga y digital, FPGA, sistemas embebidos y sistemas operativos Linux.

Según los resultados, para que el sistema obtenga la mayor distancia posible y con una tasa de error de bits baja, es necesario transmitir considerando la zona de Fresnel, para la frecuencia a la cual se desea trabajar y así mismo disminuir la atenuación por cable y conectores.

Esto se realizó con el fin de lograr la menor cantidad de nodos sensores para abarcar una gran parte del medio-ambiente que se desea analizar.

Si bien el sistema logró trabajar con diferentes tipos de modulación, este obtuvo un mejor performance con la modulación GMSK, debido a la baja potencia de transmisión de los BladeRF.

CAPÍTULO VII
CONCLUSIONES

De acuerdo a los resultados obtenidos en las pruebas realizadas se concluye que:

1. El sistema demostró ser flexible y adaptable, pudiendo trabajar con diferentes tipos de modulación y en distintas frecuencias.
2. Las pruebas preliminares realizadas con el BladeRF demuestran que es posible la realización de este tipo de sistemas de telecomunicación complejos, para poder enviar datos de diferentes tipos de sensores.
3. El sistema tiene una performance aceptable, para los distintos tipos de modulación digital que se emplearon.
4. La tasa de error de bits usando las modulaciones M-PSK fueron muy altas, resultando ineficiente para el sistema.
5. El BladeRF tiene un mejor desempeño con la modulación GMSK, debido a la baja potencia de transmisión.
6. La creación de módulos y bloques permite que el desarrollo de aplicaciones con GNU Radio sea mucho más versátil ya que basándose en el código de un bloque ya hecho se puede realizar uno a medida.
7. Gracias a los diferentes tipos de bloques para codificación que existen en GNU Radio, la encriptación de datos para hacer dotar al sistema con mayor seguridad es factible.

CAPÍTULO VIII
RECOMENDACIONES

1. Se recomienda definir bien las condiciones en las que va a trabajar el sistema de telemetría para así configurar correctamente los parámetros para cada tipo de modulación.
2. Se recomienda como otra alternativa ver la forma de realizar un procesamiento digital de las señales con el FPGA integrado en el BladeRF y no usar un computador de placa reducida (Single Board Computer o SBC).
3. Se recomienda tener que realizar cálculos de la zona de Fresnel, y tener en cuenta las alturas obtenidas, para así lograr las distancias máximas posibles.
4. Se recomienda no transmitir o recibir en aquellas bandas que no tiene licencia de operación.
5. Asegurarse de usar los filtros adecuados según la banda de frecuencias que se opera.
6. Asegúrese de usar cargas de 50 Ohm en los puertos TX y RX del bladeRF.
7. No transmitir al puerto TX sin una carga adecuada.
8. Hacer uso de un amplificador de potencia de transmisión, para obtener mayores distancias si se requiere usar frecuencias más altas como 2.4GHz.

CAPÍTULO IX
REFERENCIAS BIBLIOGRÁFICAS

1. **BALTUANO, OSCAR.** Desarrollo de un medidor autónomo de bajo costo para la determinación de calidad química del agua. Disponible en: http://dspace.ipen.gob.pe/bitstream/ipen/131/3/p%2059-64%20-%20ICT-2012.pdf. artículo web. Consultada el 19 de Junio del 2014.

2. **CHINA-TOTAL.** MQ135 Semiconductor Sensor for Air Quality Control. Disponible en: http://www.china-total.com/Product/meter/gas-sensor/MQ135.pdf. artículo web.Consultada el 13 de Junio del 2014.

3. **HENAN HANWEI ELECTRONICS CO., LTD.** MQ-9 Semiconductor Sensor for CO/Combustible Gas. Disponible en: http://www.haoyuelectronics.com/Attachment/MQ-9/MQ9.pdf. artículo web. Consultada el 13 de Junio del 2015.

4. **INSTITUTO PERUANO DE ENERGÍA NUCLEAR.** PROYECTOS. Disponible en: http://www.ipen.gob.pe/index.php/investigacion-y-desarrollo/proyectos. artículo web. Consultada el 20 de Junio del 2014.

5. **KNOLL, GLENN F.** *Radiation Detection and Measurement.* s.l. : John Wiley and sons, 2000. Vol. third edition. ISBN 0-471-07338-5.

6. **MÁXIMO, Martín.** SDR - Software Defined Radio. Disponible en: http://www.ea1ddo.es/sdr/sdr.html. Consultada el 21 de Marzo del 2014.

7. **NGUYEN, DUC TOAN.** Implementation of OFDM systems using GNU Radio and USRP. Disponible en: https://www.researchgate.net/file.PostFileLoader.html?id=583ec1ab5b495205136d3606&assetKey=AS%3A434095411994638%401480507819614. artículo web.Consultada el 20 de Junio del 2014.

8. **SANTANA FUENTES, JULIO ANDRES.** *GNU- Radio en la enseñanza de comunicaciones inalámbricas.* Chile. Editorial de la Universidad de Concepción. Primera Edición. 2012.

9. **THE GNU RADIO FOUNDATION, INC.** About GNU Radio. Disponible en: https://www.gnuradio.org/about/. artículo web.Consultada el 21 de Marzo del 2014.

10. **ZHANG, LEI.** Implementation of wireless communication based on Software Defined Radio. Tesis de Maestría. Madrid. Universidad Politécnica de Madrid. 2013.

CAPÍTULO X

ANEXOS

ANEXO B

INSTALACIÓN DE GNURADIO EN UBUNTU

Para poder instalar y trabajar correctamente con GNU Radio, nuestro sistema necesita tener instalados ciertos pre-requisitos:

- Development Tools (need for compilation)
    - g++
    - git
    - make
    - cmake
    - sdcc (from "universe")
    - guile
    - ccache (not required, but recommended if you compile frequently)

- Libraries (need for runtime and for compilation)
    - python-dev
    - SWIG
    - FFTW 3.X (libfftw3-dev)
    - cppunit (libcppunit-dev)
    - Boost 1.35 (or later, but not 1.46, 1.47 or 1.52)
    - GSL GNU Scientific Library (libgsl0-dev)
    - libusb and libusb-dev
    - ALSA (alsa-base, libasound2 and libasound2-dev)

- GNU Radio Companion
    - for the GNU Radio Companion (GRC) you need to install python-numpy, python-cheetah and python-lxml
- WX GUI
    - for the WX GUI components you need to install python-wxgtk2.8 and python-numpy
- QT GUI
    - for the QT GUI components you need to install PyQT4, PyQwt5 for Qt4, QT-OpenGL, Fontconfig, Xrender and Xinput (python-qt4, python-qwt5-qt4, libqt4-opengl-dev, libqwt5-qt4-dev, libfontconfig1-dev, libxrender-dev, libxi-dev).
- Video-SDL
    - for Video-SDL you need to install the Simple DirectMedia Layer development libraries (libsdl1.2-dev)
- Polyphase Filter Bank examples
    - for the examples in gnuradio-examples/python/pfb to work you need to install python-scipy, python-matplotlib, and python-tk
- Other useful packages
    - doxygen (for creating documentation from source code)
    - octave (from "universe")

Por ello existen varias formas de instalar GNU Radio en Ubuntu, dependiendo si deseamos que se instale todo automáticamente o deseamos hacerlo de manera manual.

La forma más sencilla es utilizando el script build-gnuradio que se encuentra en la misma página de GNURadio en el siguiente link:

http://gnuradio.org/redmine/projects/gnuradio/wiki/InstallingGRFromSource

El cual es un script de instalación para los sistemas recientes de Fedora y Ubuntu proporcionadas por Marcus Leech. Solo se debe descargar el script, darle clic derecho y darle check a "Permitir ejecutar el archivo como un programa".

Inmediatamente al ejecutarlo, se abrirá la consola, y nos pedirá una serie de confirmaciones, una vez aceptemos empezará a comprobar si tenemos instalado en nuestro sistema los pre-requisitos, y si es que no los tuviésemos nos preguntará si deseamos descargarlos. Este script también instala otras herramientas entre ellas rtl-sdr, gr-osmosdr, que nos permite trabajar con hardware SDR como el BladeRF, HackRF, y Dongles SDR.

La otra forma de instalar GNU Radio Companion, instalando los pre-requisitos de manera manual usando la consola, copiar la fuente de GNU Radio desde los repositorios y compilarlo nosotros mismos, es un poco tedioso, pero a muchos les resulta más interesante hacerlo de esta manera.

Para Instalar los pre-requisitos se puede visitar la página de GNU Radio y hacer uso de los comandos de línea que brindan en el siguiente link:

http://gnuradio.org/redmine/projects/gnuradio/wiki/UbuntuInstall#Install-the-Pre-Requisites

Los comandos de línea para usar en consola están ordenados según la versión de Ubuntu con la cual va a trabajar. Una vez instalado los pre-requisitos se deberá utilizar los siguientes comandos:

- Obtener la versión más reciente de GNU Radio:

```
git clone http://gnuradio.org/git/gnuradio.git
```

- Configurar y compilar:

```
cd gnuradio
mkdir build
cd build
cmake ../
make
```

- Realizar un test:

```
make test
```

- Y finalmente instalarlo para su uso:

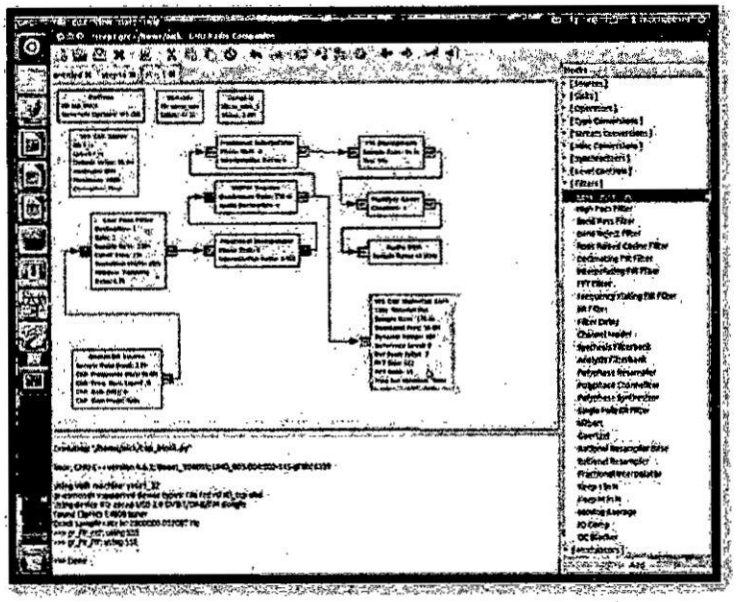
```
sudo make install
```

Tener en cuenta que mediante este método solo se instala GNU Radio, más no las otras librerías contempladas en el script build-gnuradio, para poder trabajar con hardware SDR, deberá obtener las versiones de dichas librerías y instalarlas de manera muy similar al GNU Radio.

El tiempo de instalación dependerá más que todo de la velocidad de descarga que se disponga y del CPU con el cuál esté trabajando, ya que este último influirá en la velocidad con que se compilan los archivos antes de la instalación. Una vez instalado todo ya podremos trabajar con las librerías de GNU Radio usando python o GRC (GNU Radio Companion), que es un entorno de programación gráfico usando bloques (ver figura 1.2), de una manera muy similar a Simulink de MatLab.

FIGURA B.1

PROGRAMACIÓN MEDIANTE BLOQUES USANDO GRC



ANEXO C

BLADERF

C.1. Especificaciones Técnicas del BladeRF

- **BladeRF x40**

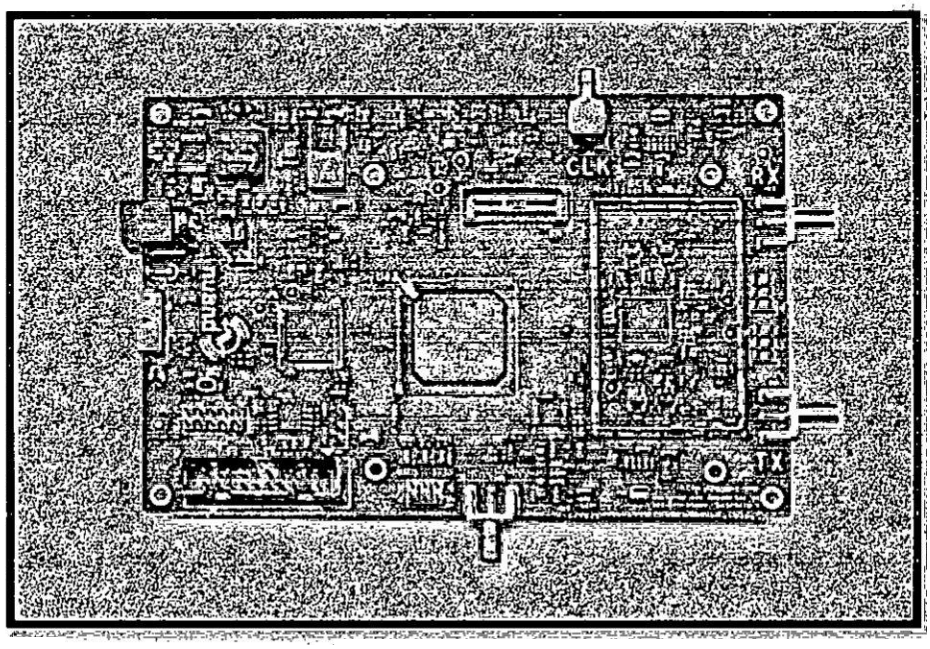
  - ✓ Totalmente alimentado por bus USB 3.0 Super Speed.
  - ✓ Portable y manipulable gracias a sus pequeñas dimensiones de 5" por 3.5".
  - ✓ Conectores RF SMA extensibles enchapados en oro.
  - ✓ Rango de frecuencia RF 300MHz - 3.8GHz.
  - ✓ Muestreo en cuadratura independiente RX/TX a 12-bit 40MSPS.
  - ✓ Capacidad de desarrollar canals full-duplex de 28MHz.
  - ✓ DAC de 16-bit calibrado de fábrica 38.4MHz +/-1ppm VCTCXO.
  - ✓ ARM9 de 200MHz con 512KB SRAM embebido (Puerto JTAG disponible).
  - ✓ Altera Cyclone 4 40KLE E FPGA (Puerto JTAG disponible).
  - ✓ Tarjeta de expansion modular GPIO, Ethernet, y sincronizador de señal PPS, para expander el rango de frecuencia, y los límites de potencia.
  - ✓ Conector de alimentación DC.
  - ✓ Arquitectura de alta eficiencia y baja potencia de ruido.
  - ✓ Soporta software en Linux, Windows, Mac y gnuradio.
  - ✓ Hardware capaz de trabajar como un analizador de espectros, analizador de señal vectorial, y generador de señal vectorial.

- **BladeRF x115**

  - ✓ Totalmente alimentado por bus USB 3.0 Super Speed.
  - ✓ Portable y manipulable gracias a sus pequeñas dimensiones de 5" por 3.5".
  - ✓ Conectores RF SMA extensibles enchapados en oro.
  - ✓ Rango de frecuencia RF 300MHz - 3.8GHz.
  - ✓ Muestreo en cuadratura independiente RX/TX a 12-bit 40MSPS.
  - ✓ Capacidad de desarrollar canals full-duplex de 28MHz.
  - ✓ DAC de 16-bit calibrado de fábrica 38.4MHz +/-1ppm VCTCXO.
  - ✓ ARM9 de 200MHz con 512KB SRAM embebido (Puerto JTAG disponible).
  - ✓ Altera Cyclone 4 115KLE E FPGA (Puerto JTAG disponible).
  - ✓ Tarjeta de expansion modular GPIO, Ethernet, y sincronizador de señal PPS, para expander el rango de frecuencia, y los límites de potencia.
  - ✓ Conector de alimentación DC.
  - ✓ Arquitectura de alta eficiencia y baja potencia de ruido.
  - ✓ Soporta software en Linux, Windows, Mac y gnuradio.
  - ✓ Hardware capaz de trabajar como un analizador de espectros, analizador de señal vectorial, y generador de señal vectorial.

FIGURA C.1

BLADERF X40 DE LA EMPRESA NUAND



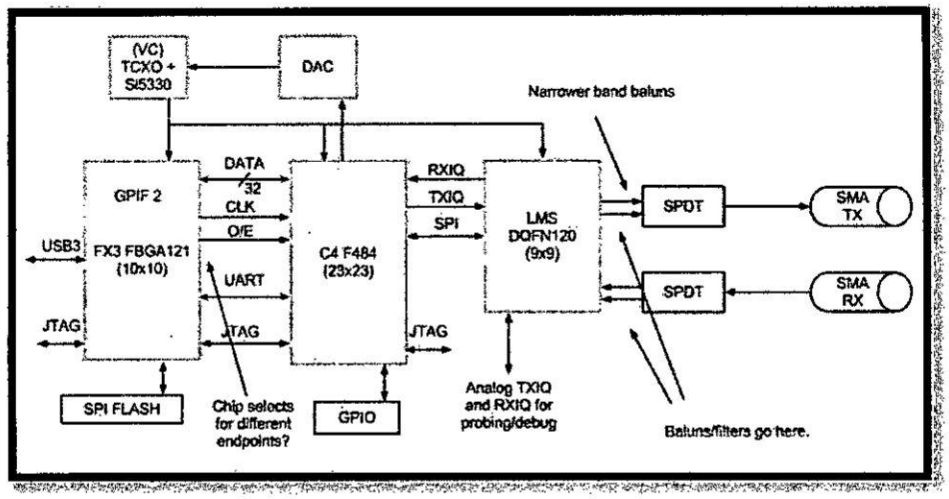
C.2. Diagrama de bloques

Este hardware se compone de:

- ✓ Si5338: Generador de reloj para la generación de frecuencias de muestreo arbitrarias.
- ✓ Lime Microsystems LMS6002D.
- ✓ FPGA Altera Cyclone IV E (40kLE or 115kLE), para el procesamiento de la señal y control.
- ✓ Microcontrolador Cypress FX3 USB 3.0 Superspeed.

FIGURA C.2

DIAGRAMA DE BLOQUES DEL BLADERF



C.3. Usando bladeRF en Linux

A continuación describiremos el proceso de construcción e instalación de las librerías del BladeRF y herramientas para un sistema operativo Linux, y un rápido funcionamiento de nuestro dispositivo usando para ello, los firmwares pre-compilados y los archivos imagen del FPGA.

Se recomienda usar la distribución de Linux Ubuntu 14.04 LTS (Trusty) y los PPA (Paquetes de Archivos Personales) de BladeRF, para una máxima estabilidad.

C.4. Instalación a través de los PPA de bladeRF

Para instalar las herramientas de bladeRF y sus drivers de funcionamiento solo se debe activar el lanzador PPA, mediante los siguientes comandos:

```
$ sudo add-apt-repository ppa:bladerf/bladerf
$ sudo apt-get update
$ sudo apt-get install bladerf
```

Como se planea trabajar usando gnuradio, gr-osmosdr, se necesita también los ficheros cabecera, para ello es necesario ejecutar el siguiente comando en el Shell de Linux:

```
$ sudo apt-get install libbladerf-dev
```

También se puede instalar un firmware compatible y los FPGA Image, debiendo actualizar manualmente el firmware usando `bladeRF-cli -flash-firmware /usr/share/Nuand/bladeRF/bladeRF_fw.img`, pero la imagen FPGA se cargará automáticamente cada vez que el bladeRF es conectado.

```
$ sudo apt-get install bladerf-firmware-fx3
$ sudo apt-get install bladerf-fpga-hostedx40   # for the 40 kLE
$ sudo apt-get install bladerf-fpga-hostedx115  # for the 115 kLE
```

Para actualizar los paquetes instalados solo bastará con ejecutar `apt-get install bladerf` nuevamente

C.5. Instalación de gr-osmosdr

Esta librería provee a GNU Radio soporte para un número de dispositivos, incluyendo el bladeRF. Los comandos a digitar son los siguientes:

```
$ cd ~/sandbox/gnuradio-builds/gr-osmosdr

$ mkdir build

$ cd build

$ cmake -DCMAKE_INSTALL_PREFIX=/opt/gnuradio-3.7.3 ../

$ make && sudo make install && sudo ldconfig
```

C.6. Verificación básica de operación dispositivo

Los comandos que mostraremos a continuación son para realizar algunas verificaciones básicas del funcionamiento del bladeRF, usando bladeRF-cli, herramienta que viene dentro de la instalación del paquete del bladeRF.

ANEXO C

CODIGO FUENTE DEL PROGRAMA DE ADQUISICIÓN DE DATOS DE LOS SENSORES

```
/*************************************************************/
// C++
// Title: PROGRAMA DE ADQUISICIÓN DE DATOS DE LOS SENSORES
// Author: Renzo Chan
// Created: OCT 25 2016
/*************************************************************/
// Conversion factor - CPM to uSV/h
#define CONV_FACTOR 0.00812

// Variables
int option;
int FID = 0;
int P = 4;
int SID1 = 1;
int SID2 = 2;
int SID3 = 3;
int SID4 = 4;
int geiger_input = 2;
float tempSD1 = 0;
float tempSD2 = 0;
float tempSD3 = 0;
float SD1 = 0.0;
float SD2 = 0.0;
float SD3 = 0.0;
float SD4 = 0.0;
float radiationValue = 0.0;
long count = 0;
long countPerMinute = 0;
long timePrevious = 0;
long timePreviousMeassure = 0;
long time = 0;
long countPrevious = 0;
String frame;
char SSD1[10],SSD2[10],SSD3[10],SSD4[10];

/* CONFIGURACIONES INICIALES DEL MICROCONTROLADOR */
void setup()
{
 pinMode(geiger_input, INPUT);        // Pin de entrada para el Contador Geieger
 Serial.begin(115200);                // Inicializando Comunicación Serial a 115200, 8 bits de datos,
ningún bit de paridad y 1 bit de parada
 attachInterrupt(0,countPulse,FALLING);// Habilitando interrupción por flanco de bajada
}
/* BUCLE PRINCIPAL */
void loop()
{
 if (millis()-timePreviousMeassure > 10000){
  countPerMinute = 6*count;
```

```
  radiationValue = countPerMinute * CONV_FACTOR;
  timePreviousMeassure = millis();
  count = 0;
 }
 if (Serial.available()>0)  // Verifica si ha llegado un dato al buffer del puerto serie
 {
  option=Serial.read();
  if (option=='a')        // Si es el caracter "a" empieza la adquisición de datos
  {
   for (int i=0; i<1000;i++) // Realiza una media de 1000 muestras para los sensores de gases y
temperatura
   {
    tempSD1 = (5*analogRead(A0))/1023.0;
    delayMicroseconds(20);
    SD1 = SD1 + tempSD1;
    tempSD2 = (5*analogRead(A1))/1023.0;
    delayMicroseconds(20);
    SD2 = SD2 + tempSD2;
    tempSD3 = (5*analogRead(A2)*100.0)/1023.0;
    delayMicroseconds(20);
    SD3 = SD3 + tempSD3;
   }
   SD1 = SD1/1000;
   SD2 = SD2/1000;
   SD3 = SD3/1000;
   SD4 = radiationValue;    //.Obtiene el valor de los puslsos nucleares
   dtostrf(SD1,5,2,SSD1);   // Convierte los datos de tipo Float a String
   dtostrf(SD2,5,2,SSD2);
   dtostrf(SD3,5,2,SSD3);
   dtostrf(SD4,5,2,SSD4);
   String c = ",";
   // Construye el frame que luego será enviado por hardware sdr
   frame = FID+c+P+c+SID1+c+SSD1+c+SID2+c+SSD2+c+SID3+c+SSD3+c+SID4+c+SSD4;
   Serial.println(frame);   // Envía el Frame Construido
  }
 }
}

/* FUNCION PARA CONTAR PULSOS NUCLEARES MEDIANTE EL PIN DE
INTERRUPCION */
void countPulse(){
 detachInterrupt(0);
 count++;
 while(digitalRead(2)==0){
 }
 attachInterrupt(0,countPulse,FALLING);
}
```

ANEXO D

CODIGO FUENTE GENERADO POR GNURADIO

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: SISTEMA DIGITAL DE COMUNICACION INALAMBRICA TX
# Author: Renzo Chan
# Generated: Fri Aug  7 21:40:37 2015
##################################################
from PyQt4 import Qt
from PyQt4.QtCore import QObject, pyqtSlot
from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from grc_gnuradio import blks2 as grc_blks2
from optparse import OptionParser
import PyQt4.Qwt5 as Qwt
import numpy
import osmosdr
import sys

from distutils.version import StrictVersion
class sis_dig_tx(gr.top_block, Qt.QWidget):

  def __init__(self):
    gr.top_block.__init__(self, "SISTEMA DIGITAL DE COMUNICACION INALAMBRICA
TX")
    Qt.QWidget.__init__(self)
    self.setWindowTitle("SISTEMA DIGITAL DE COMUNICACION INALAMBRICA TX")
    try:
      self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
    except:
      pass
    self.top_scroll_layout = Qt.QVBoxLayout()
    self.setLayout(self.top_scroll_layout)
    self.top_scroll = Qt.QScrollArea()
    self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
    self.top_scroll_layout.addWidget(self.top_scroll)
    self.top_scroll.setWidgetResizable(True)
    self.top_widget = Qt.QWidget()
    self.top_scroll.setWidget(self.top_widget)
    self.top_layout = Qt.QVBoxLayout(self.top_widget)
    self.top_grid_layout = Qt.QGridLayout()
    self.top_layout.addLayout(self.top_grid_layout)

    self.settings = Qt.QSettings("GNU Radio", "sis_dig_tx")
    self.restoreGeometry(self.settings.value("geometry").toByteArray())

    ##################################################
    # Variables
```

```
        ##################################################
        self.ss = ss = 2
        self.sel = sel = 1
        self.samp_rate = samp_rate = 1e6
        self.packet_len = packet_len = 96
        self.length_tag_key = length_tag_key = "packet_len"
        self.freq = freq = 850e6
        self.RF_Gain = RF_Gain = 9
        self.IF_Gain = IF_Gain = 20
        self.BB_Gain = BB_Gain = 0
        ##################################################
        # Blocks
        ##################################################
        self.tab4 = Qt.QTabWidget()
        self.tab4_widget_0 = Qt.QWidget()
        self.tab4_layout_0 = Qt.QBoxLayout(Qt.QBoxLayout.TopToBottom, self.tab4_widget_0)
        self.tab4_grid_layout_0 = Qt.QGridLayout()
        self.tab4_layout_0.addLayout(self.tab4_grid_layout_0)
        self.tab4.addTab(self.tab4_widget_0, "CONTROLES")
        self.top_grid_layout.addWidget(self.tab4, 0,1)
        self._sel_options = (0, 1, 2, )
        self._sel_labels = ("OFDM", "GMSK", "DBPSK", )
        self._sel_group_box = Qt.QGroupBox("Modulacion")
        self._sel_box = Qt.QHBoxLayout()
        class variable_chooser_button_group(Qt.QButtonGroup):
            def __init__(self, parent=None):
                Qt.QButtonGroup.__init__(self, parent)
            @pyqtSlot(int)
            def updateButtonChecked(self, button_id):
                self.button(button_id).setChecked(True)
        self._sel_button_group = variable_chooser_button_group()
        self._sel_group_box.setLayout(self._sel_box)
        for i, label in enumerate(self._sel_labels):
            radio_button = Qt.QRadioButton(label)
            self._sel_box.addWidget(radio_button)
            self._sel_button_group.addButton(radio_button, i)
        self._sel_callback = lambda i: Qt.QMetaObject.invokeMethod(self._sel_button_group,
"updateButtonChecked", Qt.Q_ARG("int", self._sel_options.index(i)))
        self._sel_callback(self.sel)
        self._sel_button_group.buttonClicked[int].connect(
            lambda i: self.set_sel(self._sel_options[i]))
        self.top_layout.addWidget(self._sel_group_box)
        self._samp_rate_layout = Qt.QVBoxLayout()
        self._samp_rate_tool_bar = Qt.QToolBar(self)
        self._samp_rate_layout.addWidget(self._samp_rate_tool_bar)
        self._samp_rate_tool_bar.addWidget(Qt.QLabel("samp_rate"+": "))
        class qwt_counter_pyslot(Qwt.QwtCounter):
            def __init__(self, parent=None):
                Qwt.QwtCounter.__init__(self, parent)
            @pyqtSlot('double')
            def setValue(self, value):
                super(Qwt.QwtCounter, self).setValue(value)
        self._samp_rate_counter = qwt_counter_pyslot()
        self._samp_rate_counter.setRange(1e3, 10e6, 1)
        self._samp_rate_counter.setNumButtons(2)
```

```
        self._samp_rate_counter.setValue(self.samp_rate)
        self._samp_rate_tool_bar.addWidget(self._samp_rate_counter)
        self._samp_rate_counter.valueChanged.connect(self.set_samp_rate)
        self._samp_rate_slider = Qwt.QwtSlider(None, Qt.Qt.Horizontal,
Qwt.QwtSlider.BottomScale, Qwt.QwtSlider.BgSlot)
        self._samp_rate_slider.setRange(1e3, 10e6, 1)
        self._samp_rate_slider.setValue(self.samp_rate)
        self._samp_rate_slider.setMinimumWidth(200)
        self._samp_rate_slider.valueChanged.connect(self.set_samp_rate)
        self._samp_rate_layout.addWidget(self._samp_rate_slider)
        self.tab4_layout_0.addLayout(self._samp_rate_layout)
        self._freq_layout = Qt.QVBoxLayout()
        self._freq_tool_bar = Qt.QToolBar(self)
        self._freq_layout.addWidget(self._freq_tool_bar)
        self._freq_tool_bar.addWidget(Qt.QLabel("Frecuencia del Tx"+": "))
        class qwt_counter_pyslot(Qwt.QwtCounter):
            def __init__(self, parent=None):
                Qwt.QwtCounter.__init__(self, parent)
            @pyqtSlot('double')
            def setValue(self, value):
                super(Qwt.QwtCounter, self).setValue(value)
        self._freq_counter = qwt_counter_pyslot()
        self._freq_counter.setRange(400e6, 1000e6, 1)
        self._freq_counter.setNumButtons(2)
        self._freq_counter.setValue(self.freq)
        self._freq_tool_bar.addWidget(self._freq_counter)
        self._freq_counter.valueChanged.connect(self.set_freq)
        self._freq_slider = Qwt.QwtSlider(None, Qt.Qt.Horizontal, Qwt.QwtSlider.BottomScale,
Qwt.QwtSlider.BgSlot)
        self._freq_slider.setRange(400e6, 1000e6, 1)
        self._freq_slider.setValue(self.freq)
        self._freq_slider.setMinimumWidth(200)
        self._freq_slider.valueChanged.connect(self.set_freq)
        self._freq_layout.addWidget(self._freq_slider)
        self.tab4_layout_0.addLayout(self._freq_layout)
        self._RF_Gain_layout = Qt.QVBoxLayout()
        self._RF_Gain_label = Qt.QLabel("Ganancia RF del Tx")
        self._RF_Gain_slider = Qwt.QwtSlider(None, Qt.Qt.Horizontal, Qwt.QwtSlider.BottomScale,
Qwt.QwtSlider.BgSlot)
        self._RF_Gain_slider.setRange(0, 25, 1)
        self._RF_Gain_slider.setValue(self.RF_Gain)
        self._RF_Gain_slider.setMinimumWidth(200)
        self._RF_Gain_slider.valueChanged.connect(self.set_RF_Gain)
        self._RF_Gain_label.setAlignment(Qt.Qt.AlignBottom | Qt.Qt.AlignHCenter)
        self._RF_Gain_layout.addWidget(self._RF_Gain_label)
        self._RF_Gain_layout.addWidget(self._RF_Gain_slider)
        self.tab4_layout_0.addLayout(self._RF_Gain_layout)
        self._IF_Gain_layout = Qt.QVBoxLayout()
        self._IF_Gain_label = Qt.QLabel("Ganancia IF del Tx")
        self._IF_Gain_slider = Qwt.QwtSlider(None, Qt.Qt.Horizontal, Qwt.QwtSlider.BottomScale,
Qwt.QwtSlider.BgSlot)
        self._IF_Gain_slider.setRange(0, 40, 1)
        self._IF_Gain_slider.setValue(self.IF_Gain)
        self._IF_Gain_slider.setMinimumWidth(200)
        self._IF_Gain_slider.valueChanged.connect(self.set_IF_Gain)
```



```
        num_inputs=3,
```

ANEXO E

CODIGO FUENTE DEL NODO ESCLAVO

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

# -*- coding: utf-8 -*-

##################################################

# Python

# Title: SISTEMA DE TELEMETRIA AVANZADO NODO ESCLAVO

# Author: Renzo Chan

# Created: Fri Aug  7 21:40:37 2016

##################################################

## IMPORTACION DE LIBRERIAS

import signal, sys, sistea, bladeRF_transceiver, datetime, serial

from PyQt4 import QtCore, QtGui, uic

from gnuradio import gr

from time import sleep

from PyQt4.Qwt5.Qwt import QwtThermo

## CLASE PRINCIPAL QUE CONTIENE LA INTERAZ GRAFICA DE USUARIO Y LA CLASE PARA LA MANIPULACIÓN DEL BLADERF x40

class SisTeA_App(QtGui.QMainWindow, sistea.Ui_MainWindow, QtGui.QDialog,
bladeRF_transceiver.bladeRF_transceiver):

    def __init__(self):

        # Explaining super is out of the scope of this article

        # So please google it if you're not familar with it

        # Simple reason why we use it here is that it allows us to

        # access variables, methods etc in the design.py file

        super(self.__class__, self).__init__()

        self.setupUi(self)  # This is defined in design.py file automatically

                            # It sets up layout and widgets that are defined

        ## DEFINICION DE VARIABLES INICIALES

        self.flag = 0

        self.mod = 0

        self.tx_freq.setMaximum(1000)

        self.rx_freq.setMaximum(1000)
```

```
        self.tx_freq.setProperty("value",914)
        self.rx_freq.setProperty("value",916)
        self.tx_rf_gain.setProperty("value",20)

        ## CARGANDO CONFIGURACIÓN DE LOS DIALES A SUS TEXTBOX CORRESPONDIENTES
        self.tx_freq_value.setText(str(self.tx_freq.value()))
        self.tx_rf_gain_value.setText(str(self.tx_rf_gain.value()))
        self.tx_bb_gain_value.setText(str(self.tx_bb_gain.value()))
        self.rx_freq_value.setText(str(self.rx_freq.value()))
        self.rx_rf_gain_value.setText(str(self.rx_rf_gain.value()))
        self.rx_bb_gain_value.setText(str(self.rx_bb_gain.value()))

        ## INICIALIZANDO EL NUCLEO DE TIEMPO EN PYTHON PARA PROGRAMACIÓN USANDO
THREADING
        self.timer = QtCore.QTimer(self)
        self.timer.timeout.connect(self.recv_frame) ## FUNCION recv_frame A LA QUE EL PROGRAMA
                                    ## ENTRARÁ CADA N MILISEGUNDOS

        ## ESTABLECIENDO CONECCIÓN ENTRE LOS EVENTOS DE LOS BOTONES Y SUS FUNCIONES
CORRESPONDIENTES
        self.gmsk_radioButton.released.connect(self.modulation_selection)
        self.gfsk_radioButton.released.connect(self.modulation_selection)
        self.btn_send.released.connect(self.start_tx)
        self.symbol_rate_line.returnPressed.connect(self.set_symbol_rate)
        self.tx_freq.valueChanged.connect(self.set_tx_freq)
        self.tx_rf_gain.valueChanged.connect(self.set_tx_rf_gain)
        self.tx_bb_gain.valueChanged.connect(self.set_tx_bb_gain)
        self.rx_freq.valueChanged.connect(self.set_rx_freq)
        self.rx_rf_gain.valueChanged.connect(self.set_rx_rf_gain)
        self.rx_bb_gain.valueChanged.connect(self.set_rx_bb_gain)
        self.tx_freq_value.returnPressed.connect(self.set_tx_freq_value)
        self.tx_bb_gain_value.returnPressed.connect(self.set_tx_bb_gain_value)
        self.tx_rf_gain_value.returnPressed.connect(self.set_tx_rf_gain_value)
        self.rx_freq_value.returnPressed.connect(self.set_rx_freq_value)
        self.rx_bb_gain_value.returnPressed.connect(self.set_rx_bb_gain_value)
```

```
self.rx_rf_gain_value.returnPressed.connect(self.set_rx_rf_gain_value)

## INICIANDO EL SISTEMA DE RADIOFRECUENCIA Y EL ARDUINO MICRO
## definiendo parámetros de reconocimiento iniciales
self.rf = bladeRF_transceiver.bladeRF_transceiver()
self.ID = 'A101'
    self.FID = '1'
    self.SN = '1'
    self.D_CID = 'A'
    self.S_CID = 'C'
    self.M = '1'
    self.arduinoPort = serial.Serial('/dev/ttyUSB0',115200,timeout=1)
    sleep(2)
self.rf.set_frequency_tx(long(self.tx_freq.value())*1e6)
self.rf.set_frequency_rx(long(self.rx_freq.value())*1e6)

## definiendo los parametros de ganancia, ancho de banda, sample_rate
    self.rf.set_tx_rf_gain(self.tx_rf_gain.value())        # [0,25]
self.rf.set_tx_bb_gain(self.tx_bb_gain.value())        # [-35,-4]
self.rf.set_rx_rf_gain(self.rx_rf_gain.value())        # {0, 3, 6}
self.rf.set_rx_bb_gain(self.rx_bb_gain.value())        # [5,60]

self.bandwith_line.setText(str(self.rf.get_bandwith()))
self.samp_rate_line.setText(str(self.rf.get_samp_rate()))
self.sps_line.setText(str(self.rf.get_samp_per_sym()))
self.symbol_rate_line.setText(str(self.rf.get_symbole_rate()))

## Inicializando los selectores del tipo de modulación a usar
self.rf.set_mod_selector(0)
self.rf.set_demod_selector(0)
self.rf.set_tx_valve_gmsk_value(True)
self.rf.set_tx_valve_gfsk_value(True)
self.rf.set_rx_valve_gmsk_value(True)
self.rf.set_rx_valve_gfsk_value(True)
```

```
        self.modulation_selection()
    ## DEFINICION DE FUNCIONES PARA LOS EVENTOS DE LOS BOTONES
    def set_symbol_rate(self):
        self.rf.set_symbole_rate(float(self.symbol_rate_line.text()))
        self.sps_line.setText(str(self.rf.get_samp_per_sym()))

    def set_tx_freq_value(self):
        self.rf.set_frequency_tx(long(float(self.tx_freq_value.text())*1e6))
        self.tx_freq.setProperty("value", float(self.tx_freq_value.text()))
        print "Tx Frequency = "+ str(self.rf.get_frequency_tx()) + "Hz"
    def set_tx_rf_gain_value(self):
        self.rf.set_tx_rf_gain(long(float(self.tx_rf_gain_value.text())))
        self.tx_rf_gain.setProperty("value", float(self.tx_rf_gain_value.text()))
        print "Tx rf gain = "+ str(self.rf.get_tx_rf_gain()) + "dB"
    def set_tx_bb_gain_value(self):
        self.rf.set_tx_bb_gain(long(float(self.tx_bb_gain_value.text())))
        self.tx_bb_gain.setProperty("value", float(self.tx_bb_gain_value.text()))
        print "Tx bb gain = "+ str(self.rf.get_tx_bb_gain()) + "dB"

    def set_rx_freq_value(self):
        self.rf.set_frequency_rx(long(float(self.rx_freq_value.text())*1e6))
        self.rx_freq.setProperty("value", float(self.rx_freq_value.text()))
        print "Rx Frequency = "+ str(self.rf.get_frequency_rx()) + "Hz"
    def set_rx_rf_gain_value(self):
        self.rf.set_rx_rf_gain(long(float(self.rx_rf_gain_value.text())))
        self.rx_rf_gain.setProperty("value", float(self.rx_rf_gain_value.text()))
        print "Rx rf gain = "+ str(self.rf.get_rx_rf_gain()) + "dB"
    def set_rx_bb_gain_value(self):
        self.rf.set_rx_bb_gain(long(float(self.rx_bb_gain_value.text())))
        self.rx_bb_gain.setProperty("value", float(self.rx_bb_gain_value.text()))
        print "Rx bb gain = "+ str(self.rf.get_rx_bb_gain()) + "dB"

    def set_tx_freq(self):
            self.rf.set_frequency_tx(long(self.tx_freq.value()*1e6))        # [0,25]
```

```
        self.tx_freq_value.setText(str(self.tx_freq.value()))
    def set_tx_rf_gain(self):
        self.rf.set_tx_rf_gain(self.tx_rf_gain.value())              # [0,25]
        self.tx_rf_gain_value.setText(str(self.tx_rf_gain.value()))
    def set_tx_bb_gain(self):
        self.rf.set_tx_bb_gain(self.tx_bb_gain.value())              # [-35,-4]
        self.tx_bb_gain_value.setText(str(self.tx_bb_gain.value()))
    def set_rx_freq(self):
        self.rf.set_frequency_rx(long(self.rx_freq.value()*1e6))     # [0,25]
        self.rx_freq_value.setText(str(self.rx_freq.value()))
    def set_rx_rf_gain(self):
        self.rf.set_rx_rf_gain(self.rx_rf_gain.value())              # {0, 3, 6}
        self.rx_rf_gain_value.setText(str(self.rx_rf_gain.value()))
    def set_rx_bb_gain(self):
        self.rf.set_rx_bb_gain(self.rx_bb_gain.value())              # [5,60]
        self.rx_bb_gain_value.setText(str(self.rx_bb_gain.value()))

## FUNCION PARA LA SELECCION DE MODULACION
    def modulation_selection(self):
        if self.gfsk_radioButton.isChecked() == True:
            print "Modulation Selected = GFSK"
      self.rf.set_frequency_rx_final(self.rf.frequency_rx-self.rf.frequency_shift)
            self.rf.set_mod_selector(1)
      self.rf.set_demod_selector(1)
      self.rf.set_rx_valve_gmsk_value(True)
      self.rf.set_rx_valve_gfsk_value(False)
      #print self.rf.get_frequency_rx_final()
        elif self.gmsk_radioButton.isChecked() == True:
            print "Modulation Selected = GMSK"
            self.rf.set_frequency_rx_final(self.rf.frequency_rx-self.rf.frequency_shift)
            self.rf.set_mod_selector(0)
      self.rf.set_demod_selector(0)
      self.rf.set_rx_valve_gmsk_value(False)
      self.rf.set_rx_valve_gfsk_value(True)
```

```
## FUNCION PARA LA INICIALIZACION O PARADA DE LA TRANSMISION
  def start_tx(self):
        if self.flag == 0:
          self.rf.start()
          self.timer.start(100)
      self.btn_send.setText("STOP")
      self.flag = 1

        elif self.flag == 1:
          self.btn_send.setText("START")
          self.timer.stop()
          self.rf.stop()
          self.rf.wait()
          self.flag = 0

## FUNCION PARA EL ENVIO DE MENSAJES
  def send_str(self, payload):
    self.rf.msg_source_msgq_in.insert_tail(gr.message_from_string(payload))

## FUNCION PARA LA RECEPCION DE MENSAJES
  def recv_str(self):
    self.pkt = self.rf.msg_sink_msgq_out.delete_head_nowait().to_string()
    return self.pkt

  def recv_frame(self):
    if self.rf.msg_sink_msgq_out.empty_p()==False:
      self.data = self.rf.msg_sink_msgq_out.delete_head_nowait().to_string()
      if self.data == self.ID:
        self.arduinoPort.write('a')        ## SOLICITANDO LOS DATOS DE LOS SENSORES MEDIO
AMBIENTALES
        self.SD = self.arduinoPort.readline()  ## RECIBIENDO LEYENDO LOS VALORES
        print self.SD
            self.TS = str(datetime.datetime.now()) ## OBTENIENDO EL TIMESTAMP
            self.TS = self.TS[:19
```

```
                    ## ESTRUCTURANDO EL MENSAJE ANTES DE SU TRANSMISION
                    frame =
gr.message_from_string(self.FID+','+self.SN+','+self.TS+','+self.D_CID+','+self.S_CID+','+self.M+','+self.SD[4:19])
                    self.rf.set_tx_valve_gmsk_value(False)
            self.rf.set_tx_valve_gfsk_value(False)
            sleep(1)
            ## ENVIO DEL FRAME HACIA EL NODO MAESTRO
                    for x in range(0, 5):
                            self.rf.msg_source_msgq_in.insert_tail(frame)
                sleep(1)
                    print '[Identificador] : %s '% self.data + self.TS
                    self.plainTextEdit.appendPlainText('[Identificador] : %s '% self.data + self.TS)
                    self.rf.set_tx_valve_gmsk_value(True)
            self.rf.set_tx_valve_gfsk_value(True)
                    self.rf.msg_sink_msgq_out.flush()


## INICIALIZACION DE LA INTERFAZ GRAFICA
def main():
  app = QtGui.QApplication(sys.argv)  # A new instance of QApplication
  form = SisTeA_App()             # We set the form to be our AntennaCalculatorApp (design)
  form.show()                     # Show the form
  app.exec_()                     # and execute the app


if __name__ == '__main__':              # if we're running file directly and not importing it
  main()                          # run the main function
```

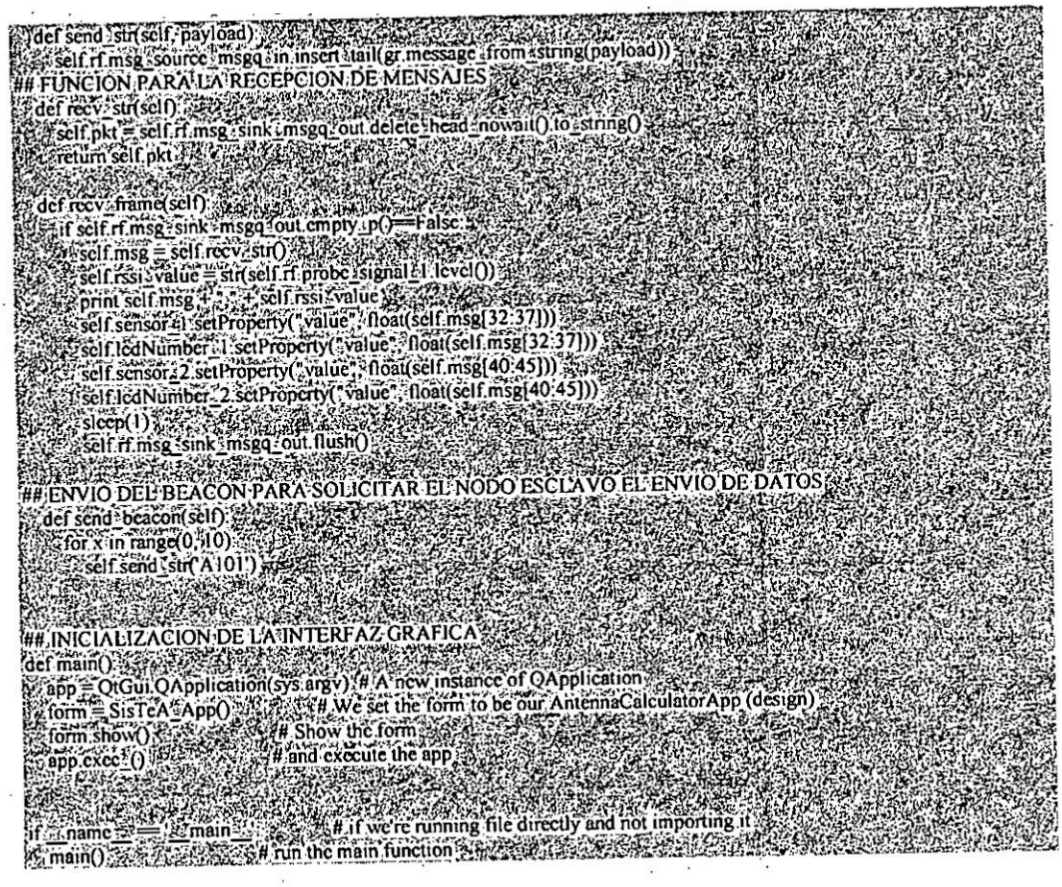
ANEXO F

CODIGO FUENTE DEL NODO MAESTRO

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
################################################
# Python
# Title: SISTEMA DE TELEMETRIA AVANZADO NODO MAESTRO
# Author: Renzo Chan
# Created: Fri Aug 7 21:40:37 2016
################################################

## IMPORTACION DE LIBRERIAS
import signal, sys, sistea, bladeRF_transceiver, datetime, serial
from PyQt4 import QtCore, QtGui, uic
from gnuradio import gr
from time import sleep
from PyQt4.Qwt5.Qwt import QwtThermo

## CLASE PRINCIPAL QUE CONTIENE LA INTERAZ GRAFICA DE USUARIO Y LA CLASE PARA LA
MANIPULACIÓN DEL BLADERF x40
class SisTeA_App(QtGui.QMainWindow, sistea.Ui_MainWindow, QtGui.QDialog,
bladeRF_transceiver.bladeRF_transceiver):

    def __init__(self):
        # Explaining super is out of the scope of this article
        # So please google it if you're not familar with it
        # Simple reason why we use it here is that it allows us to
        # access variables, methods etc in the design.py file
        super(self.__class__, self).__init__()
        self.setupUi(self)  # This is defined in design.py file automatically
                            # It sets up layout and widgets that are defined
        ## DEFINICION DE VARIABLES INICIALES
        self.flag = 0
        self.mod = 0

        self.tx_freq.setMaximum(1000)
        self.rx_freq.setMaximum(1000)

        self.tx_freq.setProperty("value", 916)
        self.rx_freq.setProperty("value", 914)

        ## CARGANDO CONFIGURACIÓN DE LOS DIALES A SUS TEXTBOX CORRESPONDIENTES
        self.tx_freq_value.setText(str(self.tx_freq.value()))
        self.tx_rf_gain_value.setText(str(self.tx_rf_gain.value()))
        self.tx_bb_gain_value.setText(str(self.tx_bb_gain.value()))
        self.rx_freq_value.setText(str(self.rx_freq.value()))
        self.rx_rf_gain_value.setText(str(self.rx_rf_gain.value()))
        self.rx_bb_gain_value.setText(str(self.rx_bb_gain.value()))
        ## INICIALIZANDO EL NUCLEO DE TIEMPO EN PYTHON PARA PROGRAMACIÓN USANDO
THREADING
        self.timer = QtCore.QTimer(self)
        self.timer.timeout.connect(self.recv_frame)
        self.timer.start(10)                ## FUNCION recv_frame A LA QUE EL PROGRAMA
                                    ## ENTRARÁ CADA 10 MILISEGUNDOS

        self.timer1 = QtCore.QTimer(self)
        self.timer1.timeout.connect(self.send_beacon)
        ## ESTABLECIENDO CONECCIÓN ENTRE LOS EVENTOS DE LOS BOTONES Y SUS FUNCIONES
CORRESPONDIENTES
        self.gmsk_radioButton.released.connect(self.modulation_selection)
        self.gfsk_radioButton.released.connect(self.modulation_selection)

        self.bandwith_line.returnPressed.connect(self.set_bandwith)
        self.samp_rate_line.returnPressed.connect(self.set_samp_rate)
```

```
        self.symbol_rate_line.returnPressed.connect(self.set_symbol_rate)

        self.tx_freq_value.returnPressed.connect(self.set_tx_freq_value)
        self.tx_bb_gain_value.returnPressed.connect(self.set_tx_bb_gain_value)
        self.tx_rf_gain_value.returnPressed.connect(self.set_tx_rf_gain_value)

        self.rx_freq_value.returnPressed.connect(self.set_rx_freq_value)
        self.rx_bb_gain_value.returnPressed.connect(self.set_rx_bb_gain_value)
        self.rx_rf_gain_value.returnPressed.connect(self.set_rx_rf_gain_value)

        self.btn_send.released.connect(self.start_tx)
        self.tx_freq.valueChanged.connect(self.set_tx_freq)
        self.tx_rf_gain.valueChanged.connect(self.set_tx_rf_gain)
        self.tx_bb_gain.valueChanged.connect(self.set_tx_bb_gain)
        self.rx_freq.valueChanged.connect(self.set_rx_freq)
        self.rx_rf_gain.valueChanged.connect(self.set_rx_rf_gain)
        self.rx_bb_gain.valueChanged.connect(self.set_rx_bb_gain)

        self.rf = bladeRF_transceiver.bladeRF_transceiver()
        self.rf.set_frequency_tx(long(self.tx_freq.value())*1e6)
        self.rf.set_frequency_rx(long(self.rx_freq.value())*1e6)
        ## definiendo los parametros de ganancia, ancho de banda, sample_rate
        self.rf.set_tx_rf_gain(self.tx_rf_gain.value())
        self.rf.set_tx_bb_gain(self.tx_bb_gain.value())      # [-35,-4]
        self.rf.set_rx_rf_gain(self.rx_rf_gain.value())      # {0, 3, 6}
        self.rf.set_rx_bb_gain(self.rx_bb_gain.value())      # [5,60]
        self.bandwith_line.setText(str(self.rf.get_bandwith()))
        self.samp_rate_line.setText(str(self.rf.get_samp_rate()))
        self.sps_line.setText(str(self.rf.get_samp_per_sym()))
        self.symbol_rate_line.setText(str(self.rf.get_symbole_rate()))
        ## Inicializando los selectores del tipo de modulación a usar
        self.rf.set_mod_selector(0)
        self.rf.set_demod_selector(0)
        self.rf.set_tx_valve_gmsk_value(False)
        self.rf.set_tx_valve_gfsk_value(True)
        self.rf.set_rx_valve_gmsk_value(False)
        self.rf.set_rx_valve_gfsk_value(True)

## DEFINICION DE FUNCIONES PARA LOS EVENTOS DE LOS BOTONES
    def set_bandwith(self):
        print self.rf.set_bandwith(float(self.bandwith_line.text()))
    def set_samp_rate(self):
        print self.rf.set_samp_rate(float(self.samp_rate_line.text()))

    def set_symbol_rate(self):
        self.rf.set_symbole_rate(float(self.symbol_rate_line.text()))
        self.sps_line.setText(str(self.rf.get_samp_per_sym()))

    def set_tx_freq_value(self):
        self.rf.set_frequency_tx(long(float(self.tx_freq_value.text())*1e6))
        self.tx_freq.setProperty("value", float(self.tx_freq_value.text()))
        print "Tx Frequency = "+ str(self.rf.get_frequency_tx()) + "Hz"
    def set_tx_rf_gain_value(self):
        self.rf.set_tx_rf_gain(long(float(self.tx_rf_gain_value.text())))
        self.tx_rf_gain.setProperty("value", float(self.tx_rf_gain_value.text()))
        print "Tx rf gain = "+ str(self.rf.get_tx_rf_gain()) + "dB"
    def set_tx_bb_gain_value(self):
        self.rf.set_tx_bb_gain(long(float(self.tx_bb_gain_value.text())))
        self.tx_bb_gain.setProperty("value", float(self.tx_bb_gain_value.text()))
        print "Tx bb gain = "+ str(self.rf.get_tx_bb_gain()) + "dB"

    def set_rx_freq_value(self):
        self.rf.set_frequency_rx(long(float(self.rx_freq_value.text())*1e6))
        self.rx_freq.setProperty("value", float(self.rx_freq_value.text()))
        print "Rx Frequency = "+ str(self.rf.get_frequency_rx()) + "Hz"
    def set_rx_rf_gain_value(self):
        self.rf.set_rx_rf_gain(long(float(self.rx_rf_gain_value.text())))
        self.rx_rf_gain.setProperty("value", float(self.rx_rf_gain_value.text()))
        print "Rx rf gain = "+ str(self.rf.get_rx_rf_gain()) + "dB"
```

```
    def set_rx_bb_gain_value(self):
        self.rf.set_rx_bb_gain(long(float(self.rx_bb_gain_value.text())))
        self.rx_bb_gain.setProperty("value", float(self.rx_bb_gain_value.text()))
        print "Rx bb gain = "+ str(self.rf.get_rx_bb_gain()) + "dB"

    def set_tx_freq_value(self):
        self.rf.set_frequency_tx(long(float(self.tx_freq_value.text())*1e6))
        self.tx_freq.setProperty("value", float(self.tx_freq_value.text()))
        print "Tx Frequency = "+ str(self.rf.get_frequency_tx()) + "Hz"

    def set_tx_freq(self):
        self.rf.set_frequency_tx(long(self.tx_freq.value()*1e6))
        self.tx_freq_value.setText(str(self.tx_freq.value()))
        print "Tx Frequency = "+ str(self.rf.get_frequency_tx()) + "Hz"
    def set_tx_rf_gain(self):
        self.rf.set_tx_rf_gain(self.tx_rf_gain.value())        # [0,25]
        self.tx_rf_gain_value.setText(str(self.tx_rf_gain.value()))
    def set_tx_bb_gain(self):
        self.rf.set_tx_bb_gain(self.tx_bb_gain.value())        # [-35,-4]
        self.tx_bb_gain_value.setText(str(self.tx_bb_gain.value()))

    def set_rx_freq(self):
        self.rf.set_frequency_rx(long(self.rx_freq.value()*1e6))        # [0,25]
        self.rx_freq_value.setText(str(self.rx_freq.value()))
    def set_rx_rf_gain(self):
        self.rf.set_rx_rf_gain(self.rx_rf_gain.value())        # {0, 3, 6}
        self.rx_rf_gain_value.setText(str(self.rx_rf_gain.value()))
    def set_rx_bb_gain(self):
        self.rf.set_rx_bb_gain(self.rx_bb_gain.value())        # [5,60]
        self.rx_bb_gain_value.setText(str(self.rx_bb_gain.value()))
## FUNCION PARA LA SELECCION DE MODULACION
    def modulation_selection(self):
        if self.gfsk_radioButton.isChecked() == True:
            print "Modulation Selected = GFSK"
            #self.rf.set_frequency_rx_final(long(self.rx_freq.value())*1e6)
            self.rf.set_frequency_rx_final(self.rf.frequency_rx-self.rf.frequency_shift)
            self.rf.set_mod_selector(1)
            self.rf.set_demod_selector(1)
            self.rf.set_tx_valve_gmsk_value(True)
            self.rf.set_tx_valve_gfsk_value(False)
            self.rf.set_rx_valve_gmsk_value(True)
            self.rf.set_rx_valve_gfsk_value(False)
        elif self.gmsk_radioButton.isChecked() == True:
            print "Modulation Selected = GMSK"
            self.rf.set_frequency_rx_final(self.rf.get_frequency_rx()-self.rf.get_frequency_shift())
            #self.rf.set_samp_per_sym(int(self.rf.get_samp_rate()/self.rf.get_symbole_rate()))
            self.rf.set_mod_selector(0)
            self.rf.set_demod_selector(0)
            self.rf.set_tx_valve_gmsk_value(False)
            self.rf.set_tx_valve_gfsk_value(True)
            self.rf.set_rx_valve_gmsk_value(False)
            self.rf.set_rx_valve_gfsk_value(True)
## FUNCION PARA LA INICIALIZACION O PARADA DE LA TRANSMISION
    def start_tx(self):
        if self.flag == 0:
            self.rf.start()
            self.send_beacon()
            self.btn_send.setText("STOP")
            self.timer1.start(5000)    ## ENVIO DE DATOS CADA 5 SEG
            self.flag = 1

        elif self.flag == 1:
            self.btn_send.setText("START")
            self.rf.stop()
            self.rf.wait()
            self.timer1.stop()
            self.flag = 0

## FUNCION PARA EL ENVIO DE MENSAJES
```



```
def send_str(self, payload):
    self.rf_msg_source_msgq_in.insert_tail(gr.message_from_string(payload))
## FUNCION PARA LA RECEPCION DE MENSAJES
def recv_str(self):
    self.pkt = self.rf_msg_sink_msgq_out.delete_head_nowait().to_string()
    return self.pkt

def recv_frame(self):
    if self.rf_msg_sink_msgq_out.empty_p()==False:
        self.msg = self.recv_str()
        self.rssi_value = str(self.rf_probe_signal_1_level())
        print self.msg + " " + self.rssi_value
        self.sensor_1.setProperty("value", float(self.msg[32:37]))
        self.lcdNumber_1.setProperty("value", float(self.msg[32:37]))
        self.sensor_2.setProperty("value", float(self.msg[40:45]))
        self.lcdNumber_2.setProperty("value", float(self.msg[40:45]))
        sleep(1)
        self.rf_msg_sink_msgq_out.flush()
## ENVIO DEL BEACON PARA SOLICITAR EL NODO ESCLAVO EL ENVIO DE DATOS
def send_beacon(self):
    for x in range(0,10):
        self.send_str('A101')

## INICIALIZACION DE LA INTERFAZ GRAFICA
def main():
    app = QtGui.QApplication(sys.argv) # A new instance of QApplication
    form = SisTeA_App()                # We set the form to be our AntennaCalculatorApp (design)
    form.show()                        # Show the form
    app.exec_()                        # and execute the app

if __name__ == '__main__':             # if we're running file directly and not importing it
    main()                             # run the main function
```