UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERIA ELÉCTRICA – PROCESAMIENTO DIGITAL DE SEÑALES

CODIFICACIÓN PARA EL CONTROL DE ERRORES EN RADIOS DEFINIDOS POR SOFTWARE

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
RAYMUNDO SALDAÑA ROSAS

TUTOR PRINCIPAL
DR. FRANCISCO JAVIER GARCIA UGALDE, FACULTAD DE INGENIERÍA

MÉXICO, D. F. DICIEMBRE 2015

JURADO ASIGNADO:

Presidente:	Dr. Moctezuma Flores Miguel
Secretario:	M. I. Escobar Salguero Larry
Vocal:	Dr. García Ugalde Francisco
1er. Suplente:	Dr. Vega Hernández Gerardo
2do. Suplente:	Dr. Tapia Recillas Horacio

Lugar donde se realizó la tesis:

Laboratorio de codificación y seguridad de los sistemas de información. Posgrado de Ingeniería UNAM. México, D.F.

TUTOR DE TESIS:

Dr. Francisco Javier García Ugalde

_ _

FIRMA

Agradecimientos

A mi director de tesis, el Dr. Francisco Javier García Ugalde por su paciencia, orientación y su apoyo para la realización de esta tesis. Y a mis sinodales que contribuyeron en la revisión del trabajo.

Esta Investigación realizada gracias al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT) de la UNAM IN-112513 " Diseño e implementación de algoritmos de marca de agua digital para la autenticación y protección de derechos de autor en imágenes digitales". Agradezco a la DGAPA-UNAM la beca recibida.

Por ultimo doy gracias al consejo nacional de ciencia y tecnología (CONACyT) por la beca otorgada para realizar de mis estudios de maestría.

ÍNDICE

1. INTRODUCCIÓN

En su versión tradicional, un equipo de radio consiste en una antena, encargada de recibir y enviar información en radiofrecuencias, y un *hardware* de propósito específico encargado de procesar esa señal de información, filtrarla, modificar su frecuencia, etc. Éste *hardware* de propósito específico no se podía modificar de forma notable en su funcionalidad.

Debido a la poca flexibilidad del radio convencional, en las últimas décadas del siglo pasado nació la tecnología de la **Radio Definida por *Software***, o por sus siglas en inglés **SDR**, donde su gran contribución, consiste en que un sistema de radio comunicaciones donde los componentes típicamente implementados en *hardware* ahora están implementados en *software*, utilizando para ello una computadora personal, u otros dispositivos de computación embebida. Un SDR básico puede estar conformado por una computadora, y algún adaptador de radiofrecuencia. De esta manera, gran parte del procesamiento de las señales se realiza en procesadores de propósito general, en vez de utilizar *hardware* de propósito específico, con la gran ventaja de que esta configuración permite cambiar los protocolos y formas de onda simplemente cambiando el *software*.

En este trabajo, el objetivo principal es el estudio de la teoría de la codificación de canal para mejorar el desempeño de las comunicaciones, permitiendo que las señales transmitidas estén protegidas contra los efectos del canal, tales como el ruido, la interferencia y el desvanecimiento, mediante el estudio e implementación de los algoritmos de los códigos detectores-correctores de errores de bloque (BCH, Reed-Solomon), utilizando los radios definidos por *software*. Para así poder verificar como la probabilidad de error disminuye al aplicar los códigos correctores de errores, y también poder comprobar el segundo teorema de Shannon [31] sobre la codificación para un canal discreto sin memoria, con ruido, el cual establece que si la tasa de transmisión R es menor que la capacidad teórica del canal C, existe algún código tal que la probabilidad de error sea tan pequeña como se desee. Shannon estableció la existencia de esos códigos, pero no definió como estos eran construidos.

A partir de la década de los 50, del siglo pasado, se han hecho múltiples esfuerzos para desarrollar códigos que mejor se acerquen al límite teórico de Shannon. Hasta antes de los 90, los únicos códigos con los que se pretendía alcanzar el límite de Shannon fueron los códigos de bloque, códigos cíclicos (BCH por sus autores Bose-Chauduri-Hocquenghem, y Reed-Solomon) que son el objeto de estudio de este trabajo, y los códigos convolucionales. Sin embargo, aunque por razones de complejidad no serán tratados en esta tesis, existen en la actualidad códigos que se acercan más al límite de Shannon; en el año de 1993, con los trabajos de Claude Berrou y Alain Glavieux y Thitimajshima [32], se desarrollaron los turbo códigos, que junto con los códigos LDPC definidos por Robert G. Gallager [33], son los códigos correctores con mejor desempeño, esto debido a su carácter aleatorio y a los algoritmos de decodificación iterativa que ellos utilizan.

1.1. Objetivos del proyecto

- Estudiar los algoritmos de codificación y decodificación de los códigos de bloque BCH y de Reed-Solomon.
- Estudiar la arquitectura, funcionamiento y programación de los radios definidos por *software*.
- Implementar en *software* los algoritmos de codificación y decodificación de los códigos de bloque BCH y de Reed-Solomon.
- Probar y evaluar el desempeño de los códigos de bloque en los radios definidos por *software*.

1.2. Planteamiento del problema

En los sistemas de comunicaciones digitales resulta necesario implementar algoritmos de codificación de canal, ya que los sistemas recepción deben de corregir los errores que aparecen durante la transmisión de la información. En este trabajo, el objetivo principal es el estudio de la teoría de la codificación de canal para mejorar el desempeño de las comunicaciones, permitiendo que las señales transmitidas estén protegidas contra los efectos del canal, tales como el ruido, la interferencia y el desvanecimiento, mediante el estudio e implementación de los algoritmos de los códigos detectores-correctores de errores de bloque (BCH, Reed Solomon), utilizando los radios definidos por *software*.

1.3. Metodología

- Investigar los algoritmos de codificación y de decodificación de los códigos de bloque cíclicos de Reed-Solomon y BCH.
- Implementar y probar en *software* los algoritmos de los códigos de bloque BCH y de Reed-Solomon y verificar su correcto funcionamiento.
- Analizar el hardware, instalación y funcionamiento del dispositivo SDR USRP2 N210 de la compañía ETTUS research, con el software libre GNURADIO en el sistema operativo Linux Ubuntu.
- Analizar y probar las herramientas de procesado de señal disponibles en la plataforma GNURADIO, además de investigar cómo implementar nuestros propios bloques de procesado de señal dentro de la plataforma.

- Evaluar el desempeño de los códigos de bloque implementados, en los radios definidos por *software*.

2. ESTADO DEL ARTE

2.1. Orígenes del “Software Radio”

En la evolución de las telecomunicaciones inalámbricas han aparecido diversas tecnologías para el intercambio de información, con requisitos cada vez más exigentes de calidad y rapidez. Las incompatibilidades entre las diferentes tecnologías han supuesto un problema para reutilizar equipos, o prestar determinados servicios. La SDR soluciona estos inconvenientes de compatibilidad e interoperabilidad, definiendo un conjunto de procedimientos y técnicas orientadas a realizar el procesamiento de señales de radio por medio de un dispositivo de propósito general, el cual puede ser modificado mediante software logrando así un cambio dinámico. El dispositivo debe ser configurable para que sea posible recibir distintos anchos de banda y frecuencias, para posteriormente, con un procesado digital, poder realizar la sincronización y la detección de la señal recibida.

La primera implementación importante del concepto SDR fue en el proyecto militar estadounidense SpeakEasy, cuyo objetivo era implementar más de 10 tipos de tecnologías de comunicaciones inalámbricas en un equipo programable, operando en la banda de frecuencias de 2MHz a 200MHz. Un objetivo adicional del proyecto consiste en que el prototipo debía tener la posibilidad de actualizar su código, para que así se pudieran tener en cuenta los estándares futuros [3].

2.2. Plataforma SDR

Los equipos de radio están diseñados internamente por circuitos electrónicos especiales; su funcionamiento hace posible la transmisión y recepción de la información a través del enlace inalámbrico de radio, por consiguiente se dice que el diseño de los radios es totalmente dependiente del hardware. A lo largo del tiempo se han utilizado dispositivos electrónicos cada vez más sofisticados, los cuales han mejorado el funcionamiento del radio. Este concepto se ve renovado cuando Joe Mitola [4], en 1991 describe los principios de la arquitectura, sin detalles de la implementación en el artículo, "*Software* Radio: Survey, Critical Analysis and Future Directions", que se convirtió en la primera publicación de la IEEE en emplear el término en 1992.

Dentro del concepto SR, los componentes físicos de un sistema de radio que han sido típicamente implementados en hardware, por ejemplo: filtro, amplificador, modulador, demodulador, detector, etc., son implementados por medio de un *software* en una computadora, o en un sistema embebido [17].

Un transceptor SR ideal se ilustra en la Figura 1, los convertidores A/D y D/A en el extremo de la antena de trasmisión/recepción y en el del teléfono, permiten la transmisión y recepción de radio; las funciones de generación de señales, modulación/demodulación, sincronización, control, codificación y decodificación deben realizarse en software [4].

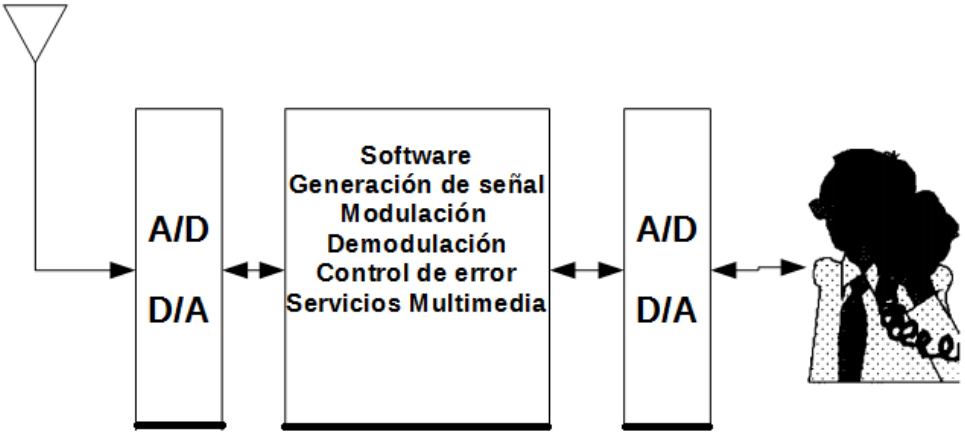

Figura 1 Software Radio ideal

En la mejora de un diseño en la radio definida por *software*, la gran mayoría de los nuevos contenidos consisten en un software y el resto son mejoras en el diseño de los componentes de hardware.

Por otra parte, un sistema SDR es un sistema de radiocomunicación que ha ido evolucionando con los años, pero que se sigue basando en el esquema básico que se muestra en la Figura 2, compuesto por tres bloques funcionales: sección de banda base, sección de frecuencia intermedia (IF), y sección de radiofrecuencia (RF).

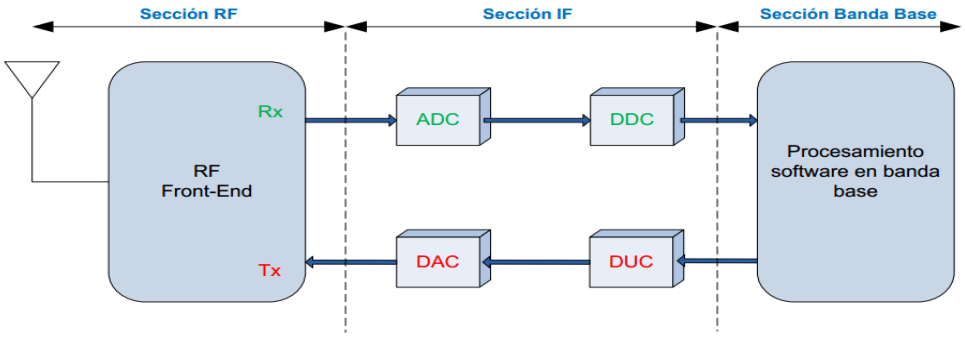

Figura 2 Esquema básico de un SDR

La sección de RF, también denominada RF Front-End, o cabecera de RF, es la encargada de transmitir/recibir las señales de radio frecuencia para adecuarlas y convertirlas en frecuencia intermedia (IF) en recepción, o amplificar y modular las señales de IF en el caso de la transmisión.

La sección de IF se encarga de pasar la señal de IF a banda base y digitalizarla en recepción, o pasar la señal de banda base a IF y hacer la conversión digital-analógica de la señal en el caso de la transmisión. 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.

Por último la 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, etc., llevándose a cabo en software [3].

2.3. USRP

El dispositivo Universal Software Radio Peripheral (USRP) es un periférico del fabricante Ettus Research, diseñado para trabajar en conjunto con un procesador externo (PC, Workstation), a través de una FPGA (Field Programmable Gate Array) y permite la realización de *Software* Radios, este periférico realiza las funciones de llevar la señal de RF a banda base a través de la sección de IF y viceversa, tal como se muestra en la Figura 3. La interfaz Ethernet permite comunicar al USRP con la computadora que realiza el procesamiento en software [3].

Estos dispositivos trabajan con el software libre de GNU Radio y recientemente son compatibles con productos de National Instruments como LabView y Simulink de Matlab. A partir de la versión 2011a Matlab y Simulink se conectan a la familia USRP para proporcionar un entorno de diseño y modelado a través del paquete Communications System Toolbox, esto permite diseñar y verificar prácticamente los sistemas SDR; otra opción para programar con un enfoque de diseño gráfico es utilizando el software NI LabVIEW, este software permite crear prototipos de sistemas inalámbricos de manera más rápida y acortar significativamente el tiempo para obtener resultados [5], [6].

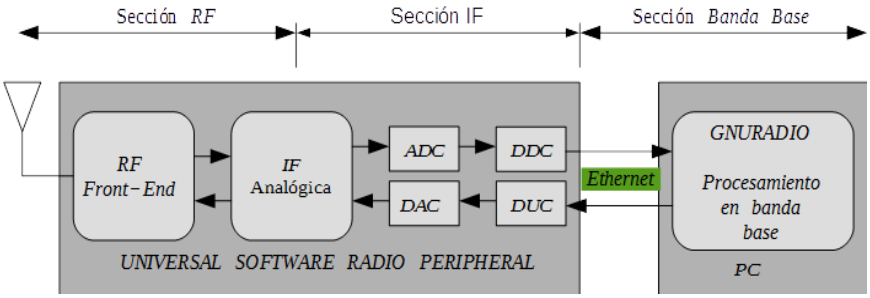

Figura 3 Diagrama básico del Universal Software Radio Peripheral

2.3.1 USRP Hardware

El modelo USRP N210 está diseñado para aplicaciones que requieran gran ancho de banda, cuenta con una FPGA Xilinx Spartan 3A-DSP 3400, un convertidor DAC de doble canal con 16 bits de resolución y una tasa de muestreo de 400 MS/s (Mega muestras por segundo), un convertidor ADC de doble canal de 14 bits de resolución y una tasa de muestreo de 100 MS/s, y módulos DDC (Digital down converter) para decimar las señales y DUC (Digital up converter, o interpolador) para la interpolación de las señales [9].

En el panel frontal del USRP N210, Figura 4, aparecen una serie de LEDs que son útiles en la depuración de problemas de hardware y software. Los LED revelan lo siguiente sobre el estado del dispositivo:

- LED A: Este led indica que el USRP está en transmisión.
- LED B: Este led indica que el cable MIMO está conectado.
- LED C: Este led indica que el USRP está en recepción.
- LED D: Este led indica que el firmware está cargado.
- LED E: Este led indica cuando la referencia está bloqueada.
- LED F: Este led indica cuando el CPLD (Complex Programmable Logic Device) está cargado.

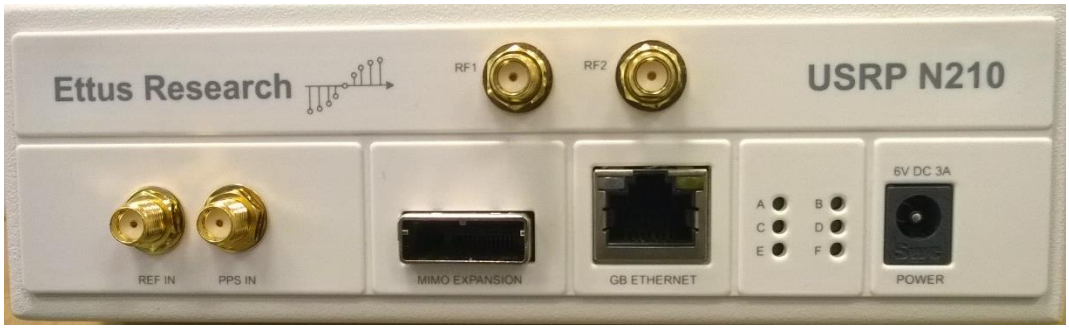

Figura 4 USRP N210

También en el panel frontal se encuentran 4 conectores SMA, dos de ellos REF 1 y REF 2 son los encargados de interconectar la señal de RF con la Daughterboard, y en donde REF 1 es un transceptor por lo que puede funcionar como receptor y transmisor, y RF2 solo opera como receptor; los otros dos conectores SMA sirven para la sincronización de dispositivos USRP con el objetivo de transmitir, o recibir, muestras alineadas en tiempo para MIMO, u otras aplicaciones que requieren múltiples dispositivos USRP funcionando sincrónicamente, los dispositivos USRP toman estas dos señales de referencia de los conectores SMA REF IN y REF PPS IN con el fin de sincronizar los relojes y el tiempo, una referencia de 10 MHz se toma en el conector SMA REF IN, y sirve para proporcionar una única referencia de frecuencia para ambos dispositivos, finalmente el cuarto conector SMA

Pulse-per-second (PPS IN) nos sirve para sincronizar el tiempo de la muestra a través de los dispositivos [13].

2.3.2 Tarjeta principal (Motherboard)

El USRP cuenta con dos niveles de tarjetas. El primer nivel es la tarjeta principal, también denominada madre, o Motherboard, que se muestra en la Figura 5, en donde se encuentra la FPGA, los convertidores ADC´s y DAC´s, los módulos DDC y DUC, la alimentación y la conexión vía Ethernet, esta placa es la encargada de comunicar la señal generada vía *software* desde el host hacia el módulo de RF, su función comienza a la salida de la computadora y acaba cuando la señal atraviesa el DAC en el caso de la transmisión. Mientras que en la recepción su función inicia cuando la señal sale de la Daughterboard, y termina cuando la señal es conducida hacia la computadora.

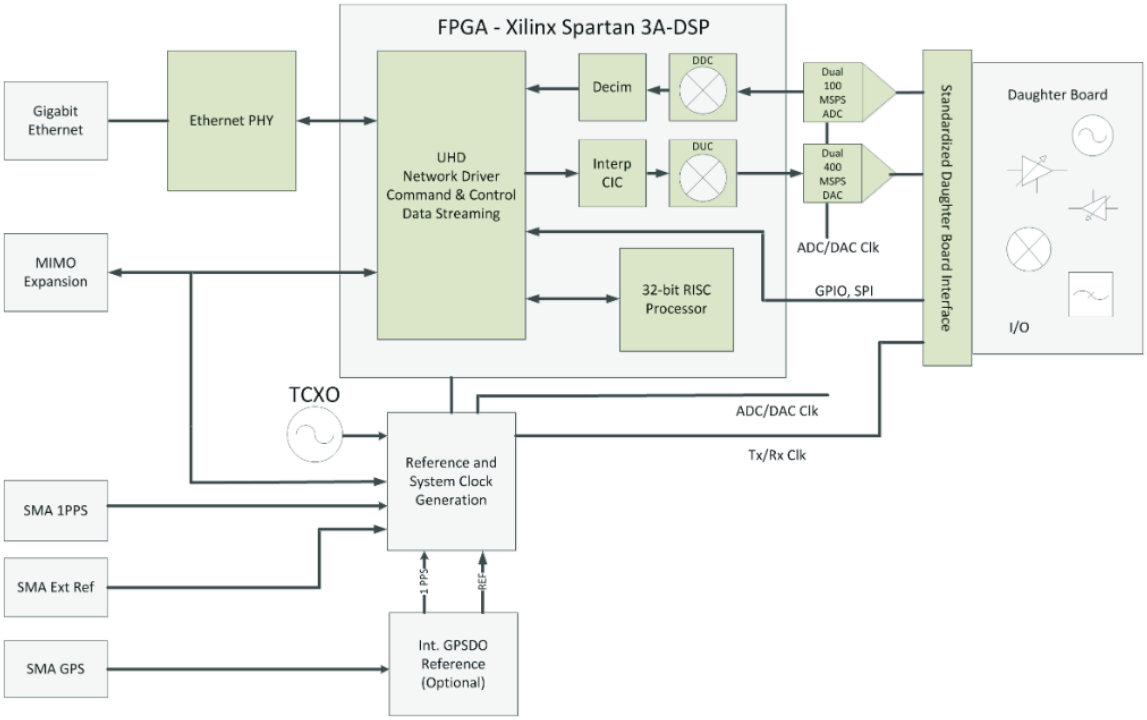

Figura 4 Tarjeta madre del USRP N210

Las señales en recepción en banda base en la tarjeta madre son muestreadas por el ADC, y las muestras digitales son sincronizadas dentro de la FPGA. La imagen existente en la FPGA provee conversión digital hacia abajo, funcionalidad que incluye ajuste fino de frecuencia y varios filtros para decimar. Después de decimar muestras en bruto y otros datos, se transmiten a una computadora host a través de la interfaz de esta, en el caso del USRP N210

la transferencia es por Ethernet. En el proceso de transmisión, las señales generadas en *software* por computadora son transferidas por Ethernet hacia la tarjeta madre, donde la imagen existente en la FPGA nos provee conversión digital hacia arriba, para la interpolación de las señales, finalmente las señales son conducidas al DAC de doble canal, el cual convertirá las señales digitales en analógicas para su transmisión y las enviará hacia la Daughterboard. En la Figura 5 se muestra la trayectoria que siguen las señales a través de la Motherboard, tanto para la transmisión como para la recepción.

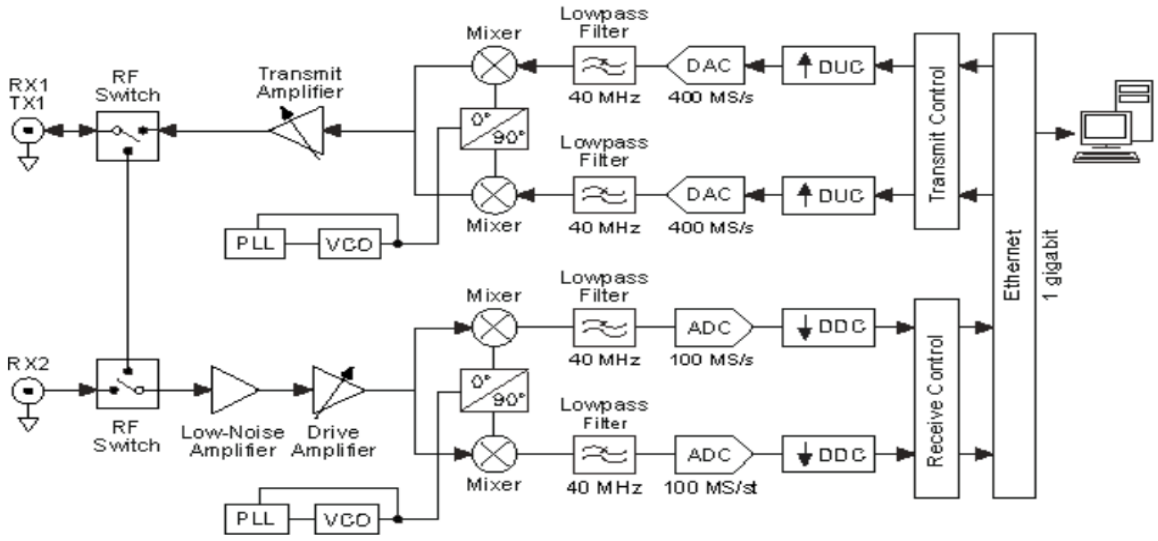

Figura 5 Arquitectura general del USRP N210

2.3.3 Tarjeta secundaria (Daughterboard)

El segundo nivel de tarjetas se compone de las tarjetas secundarias Daughterboards, o hijas, estas existen para transmisión y/o recepción. Así, el USRP puede trabajar con varias tarjetas secundarias, éstas pueden funcionar como transmisores, receptores o transceptores (transceiver, TRX), en este último caso pueden transmitir y recibir a la vez.

La función de las tarjetas secundarias comienza a la salida del DAC y termina cuando la señal es conducida hasta el conector SMA en transmisión, en el caso de la recepción su función comienza cuando la señal llega al conector SMA y termina cuando la señal es conducida al ADC.

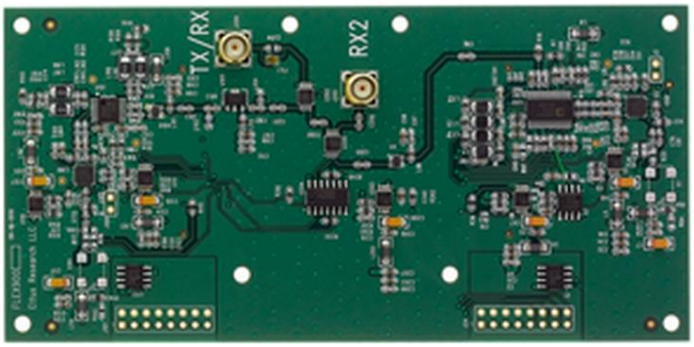

Figura 6 Tarjeta hija FRX1800

El RFX1800 es un transmisor-receptor de alto rendimiento diseñado específicamente para operar en la banda de 1900 MHz a 2100 MHz. Con una potencia típica de 100 mW, y una Figura de ruido de 8 dB. El RFX1800 utiliza osciladores locales independientes para la transmisión y recepción.

2.4. GNU Radio

GNU Radio es una herramienta de software libre y de código abierto, constituida por un conjunto de archivos y librerías que proporcionan bloques de procesado de señales, permitiendo así el diseño y la simulación de sistemas basados en software radio.

Esta herramienta de software puede ser utilizada con hardware externo adicional (como puede ser el USRP, RTL2832, OsmosSDR…) brindando la posibilidad de implementar físicamente un sistema basado en software radio, o bien, puede ser utilizada en un entorno de simulación [7].

La interfaz gráfica de GNU radio se puede concebir como un grafo, donde los nodos simbolizan los bloques de procesado de señal, y la interconexión entre ellos determina el camino que la señal seguirá comenzando en una fuente y terminando en un sumidero.

En GNU Radio existen tres tipos principales de bloques, los cuales se agrupan de la siguiente manera:

- **Fuentes:** Estos bloques especifican cualquier tipo de fuente como por ejemplo un archivo de audio wav, un archivo de texto, una imagen, un generador de señales, una fuente aleatoria, un micrófono, o el propio USRP.
- **Sumideros:** La señal tendrá un destino final como bien puede ser un archivo de audio, texto, imagen, la tarjeta de sonido, o el USRP. Se hace notar 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.

- **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, multiplicadores, amplificadores en software, etc.

Los bloques de GNU Radio se caracterizan por procesar los datos de manera continua desde su entrada hasta su salida. Idealmente los bloques desempeñan únicamente una función para así hacer a GNU Radio más flexible. Estos, son caracterizados por su número de puertos de entrada y de salida, así como del tipo de dato que manejan.

Las fuentes están caracterizadas por tener solo puerto de salida, mientras que los sumideros tienen solo puertos de entrada. Los tipos de datos que se manejan son byte, short, int, float, complex. El procesado de señal y en general todo el trabajo a bajo nivel está implementado en C++, mientras que se hace uso del lenguaje Python para escribir la aplicación. La interacción entre los diferentes niveles de GNU Radio se muestra en la Figura 7.

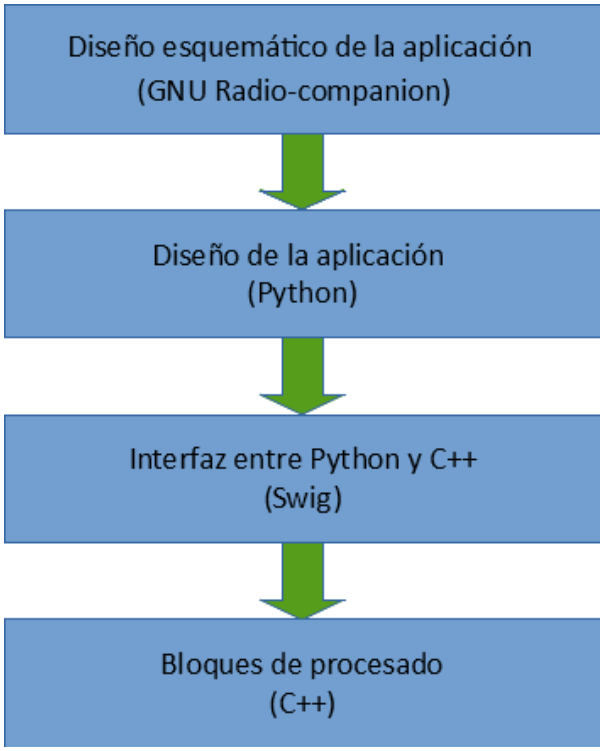

Figura 7 Arquitectura GNU Radio

La creación de una aplicación en GNU Radio puede llevarse a cabo típicamente de dos formas:

1) Programando directamente la aplicación en Python.
2) Diseñarla mediante la herramienta gráfica GNU Radio-companion.

GNU Radio-companion surge como alternativa a la programación directa en Python de la aplicación. Se trata de una interfaz que permite el diseño de sistemas mediante programación visual. Esta herramienta, muy similar a Simulink de Matlab, genera el código Python de la aplicación de forma automática, permitiendo así, contemplar y modificar directamente el código.

Para añadir un bloque en un esquemático de GNU Radio-companion bastará con dar "doble click" sobre el que se desea añadir y para interconectar los bloques, bastará simplemente con seleccionar los bloques a unir siguiendo el orden en el que vayan. En la Figura 8 se muestra la herramienta GNU Radio-companion.

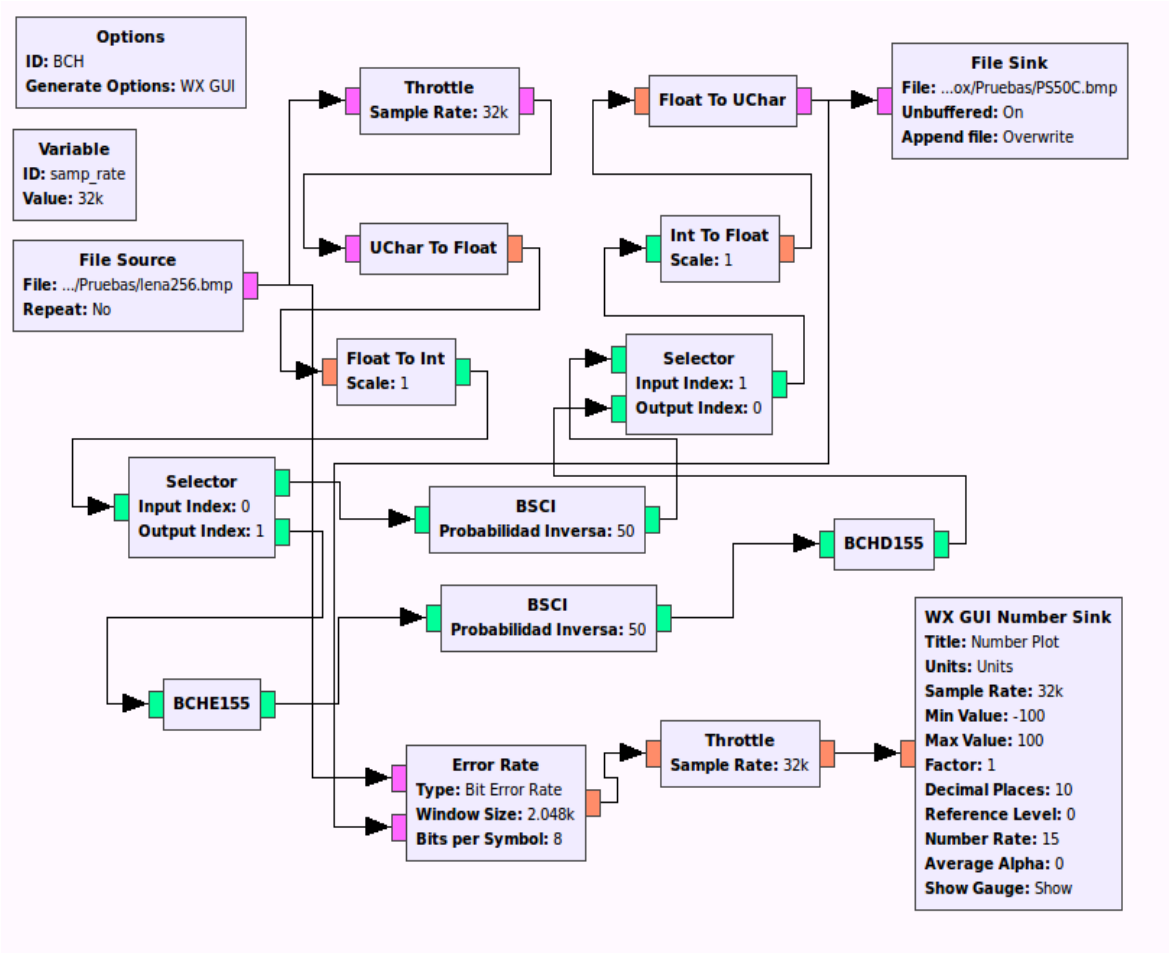

Figura 8 Grafo de GNU Radio companion

2.4.1 Módulos del GNU Radio

GNU Radio viene con varias bibliotecas y módulos [11]. Las bibliotecas y módulos que conforman GNU Radio, son agrupados dependiendo de la función que desempeñen, y los principales módulos que presenta son:

- **gr**: Biblioteca principal GNU Radio. De utilización muy frecuente.
- **gr-analog**: Este módulo contiene todo lo relacionado con las señales analógicas y modulación analógica.
- **gr-audio:** Este módulo proporciona control sobre la tarjeta de sonido, permite enviar y recibir audio a las tarjetas de sonido.
- **gr-blocks:** Contiene muchos de los bloques genéricos, estándar, o simples, utilizados para muchos aspectos de la construcción de gráficos de flujo en GNU Radio [12].
- **gr-channels:** Este módulo contiene modelos de canal para simulación.
- **gr-digital:** Este módulo contiene todo lo relacionado con la modulación digital.
- **gr-fec:** Este módulo contiene todo lo relacionado con la corrección de errores, Forward Error Correction.
- **gr-fft:** Contiene bloques de procesamiento de señal para realizar la FFT y funciones relacionadas con la FFT.
- **gr-filter**: Contiene bloques de procesado de señal para realizar operaciones de filtrado.
- **gr-qtgui**: Contiene herramientas gráficas para analizar datos (tiempo, frecuencia, espectrograma, constelación, histograma) utilizando la biblioteca QT.
- **gr-trellis:** Contiene bloques y herramientas para la construcción de enrejados (trellis) y modulación con codificación usando estos enrejados.
- **gr-vocoder:** Contiene bloques relacionados con la codificación y decodificación de voz.
- **gr-wavelet:** Este módulo proporciona bloques de procesado para las trasformadas wavelet.
- **gr-wxgui:** Este módulo es un sub-módulo, que contiene utilidades para crear rápidamente interfaces gráficas de usuario para los gráficos de flujo.
- **gr-uhd**: Es la interfaz con la biblioteca UHD para conectarse, enviar y recibir datos con la línea de productos Ettus Research.

Los módulos en GNU Radio son estructurados en carpetas, las cuales son las encargadas de agrupar las librerías y archivos, un módulo en GNU Radio presenta la siguiente estructura:

- **Apps:** Esta carpeta contiene ejemplos y aplicaciones de prueba del módulo.

- **Cmake**: Esta carpeta contiene archivos de configuración necesarios para la correcta instalación del módulo.
- **GRC:** Esta carpeta contiene los diferentes archivos ".xml" que describen los bloques para poder usarlos en la aplicación GNU Radio-companion.
- **Include:** Esta carpeta contiene los archivos fuente de las librerías ".h" de los bloques de procesado.
- **Lib**: Esta carpeta contiene los archivos fuente ".cc" de los bloques de procesado.
- **Python**: Esta carpeta contiene los diferentes scripts de Python.
- **Swig**: Esta carpeta contiene los archivos utilizados por la herramienta Swig para crear interfaces entre Python y C ++.

3. CÓDIGOS DE REED SOLOMON

3.1 Introducción

En diciembre de 1958 I. S. Reed y G. Solomon terminaron el informe, titulado "códigos basados en polinomios sobre ciertos campos finitos" en el laboratorio Lincoln del M.I.T [18]. En 1960, una ligera modificación de este informe fue publicada como un artículo [19] en la revista de la Sociedad industrial y Matemáticas Aplicadas (SIAM). Este documento de cinco páginas describe una nueva clase de códigos cíclicos de corrección de errores que actualmente se llaman códigos de Reed-Solomon (RS). En las décadas posteriores a su descubrimiento los códigos RS han disfrutado de innumerables aplicaciones como el disco compacto y televisión digital, hasta las naves espaciales y satélites en el espacio exterior.

Debido a los enormes avances en las técnicas digitales para computadoras, los nuevos sistemas de comunicaciones digitales están reemplazando rápidamente a los sistemas analógicos, ampliamente usados en el pasado. El código RS es una de las técnicas más poderosas para garantizar la integridad de los datos digitales transmitidos, o almacenados, contra errores, o borrados (erasures). La codificación RS se ha convertido en una tecnología clave en las comunicaciones digitales modernas, impulsada por la creciente demanda de un mayor rendimiento, menor costo, y la productividad sostenida para aplicaciones en la vida diaria [2].

3.1.1 Hitos en el desarrollo de los códigos de Reed-Solomon

En términos generales, las técnicas de codificación de Reed-Solomon han pasado por dos grandes etapas de desarrollo: (1) los estudios teóricos y el desarrollo de algoritmos, y (2) el diseño de arquitecturas para la rápida decodificación y su realización para aplicaciones comerciales. Antes de la década de 1970, los estudios de la codificación de Reed-Solomon se centraron en sus propiedades, como: la distancia mínima, la distribución del peso, y los algoritmos de codificación y decodificación.

Incluso entonces se consideraron a los códigos de Reed-Solomon como entre los códigos de control de error más versátiles y potentes que tenían la capacidad de corregir errores tanto aleatorios y de ráfaga [22]. También fue bien entendido por estas primeras investigaciones que uno de los principales obstáculos para la aplicación práctica de los códigos de Reed-Solomon era la complejidad de decodificación.

El algoritmo de decodificación más antiguo para los códigos de Reed-Solomon nos remonta a marzo de 1959 en un informe del laboratorio Lincoln del M.I.T [23]. En este trabajo, I. S. Reed y G. Solomon presentan un procedimiento de decodificación que utiliza una versión de campo finito basada en la expansión de una función en una serie de potencias. Esto hizo

posible resolver ciertos sistemas de ecuaciones simultáneas. Aunque mucho más eficiente que una tabla de búsqueda, su algoritmo era todavía útil sólo para los códigos de Reed-Solomon menos poderosos.

En 1960 Peterson proporciona la primera descripción explícita de un algoritmo de decodificación para los códigos binarios BCH [24]. Se encontró que su algoritmo para resolver explícitamente las ecuaciones de síndrome fue bastante útil para decodificar también códigos de Reed-Solomon para corregir un número relativamente pequeño de errores. Sin embargo se hizo computacionalmente intratable cuando el número de errores se hizo grande. El algoritmo de Peterson fue mejorado y ampliado a los códigos no binarios por Gorenstein y Zierler (1961) [25], Chien (1964) [26], y Forney (1965) [27]. Estos esfuerzos fueron productivos, pero los códigos de Reed-Solomon capaces de corregir, por ejemplo, más de 6 errores todavía no podían llevarse a cabo de una manera eficiente. La complejidad de los algoritmos de decodificación de tipo Peterson era $O(n^3)$. Los detractores de la investigación de la codificación, a principios de 1960 utilizan la falta de un algoritmo de decodificación eficiente para argumentar que los códigos de Reed-Solomon no eran más que una curiosidad matemática. Afortunadamente para el mundo digital moderno de las telecomunicaciones, las evaluaciones de estos detractores resultó ser errónea.

El avance se produjo en 1967 cuando E. Berlekamp demostró un algoritmo de decodificación extremadamente eficiente para ambos códigos BCH y Reed-Solomon [28]. El algoritmo de Berlekamp permitió por primera vez la posibilidad de una decodificación rápida y eficiente de decenas de errores de símbolo en algunos muy potentes códigos RS. En 1968, Massey mostró que el problema de la decodificación BCH es equivalente al problema de sintetizar el registro de desplazamiento de realimentación lineal (LFSR) más corto, que es capaz de generar una secuencia dada [29]. Massey demostró entonces un algoritmo de decodificación basada en registros de desplazamiento rápido para códigos BCH y de Reed-Solomon que es equivalente a la versión original de Berlekamp. Este enfoque basado en el registro de desplazamiento es ahora comúnmente conocido como el algoritmo de Berlekamp-Massey (BM). La complejidad del algoritmo de decodificación de BM es $O(n^2)$.

El algoritmo de Euclides para la solución de la ecuación clave se desarrolló por primera vez en 1975 por Sugiyama et al [30]. Demostró que un algoritmo, que tiene un orden de complejidad $O(n^2)$, también se puede utilizar para decodificar de manera eficiente códigos de Reed-Solomon, BCH y códigos de tipo Goppa.

Estas técnicas de decodificación eficientes han hecho a los códigos de Reed-Solomon muy populares para un gran número de diferentes aplicaciones. Aproximadamente en 1970, las primeras aplicaciones de los códigos de Reed-Solomon que no fueran para uso militar, consistieron en el uso de un código de corrección de errores de un byte en un sistema de telecomunicaciones en el espacio profundo. El primero uso sin restricciones de la codificación de Reed-Solomon fue en la misión Voyager II en la exploración del espacio profundo, a partir de 1977. La misión Voyager II empleó un código de Reed-Solomon

$RS(255,223)$, que corrige 16 bytes erróneos. El algoritmo usado para la decodificación de este código específico de Reed-Solomon, fue el algoritmo de Euclides, empleado para determinar el polinomio localizador de errores, y finalmente corregir los errores [2].

Finalmente, una de las propiedades a destacar de los códigos de Reed-Solomon, es que son códigos de "máxima distancia separable" (MDS), lo que significa que ningún otro código de longitud y distancia mínima tiene más palabras que un código MDS de iguales parámetros. Otra forma de verlo sería decir que no existe ningún código de parámetros, y con mayor capacidad de corrección de errores, que un código MDS con los mismos parámetros [20].

3.2 Codificación

La ecuación (1) expresa la forma más convencional de los códigos no binarios de Reed-Solomon, en términos de los parámetros n, k, t, y cualquier número entero positivo $m > 2$ [1].

$$(n, k) = (2^m - 1, 2^m - 1 - 2t) \quad (1)$$

Donde

- n: Es el número total de símbolos en el bloque codificado
- k: Es el número de símbolos de datos que se está codificado
- t: Es la capacidad de corrección de símbolos erróneos del código

Y $n - k = 2t$ es el número de símbolos de paridad. Para los códigos no binarios, como los Reed-Solomon, la distancia entre dos palabras de código se define como el número de símbolos en la que las secuencias difieren. Para estos códigos la distancia mínima está dada por.

$$d_{min} = n - k + 1 \quad (2)$$

Y la capacidad de corrección de errores t está dada por

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor = \left\lfloor \frac{n-k}{2} \right\rfloor \quad (3)$$

El polinomio generador de un código Reed-Solomon adopta la siguiente forma.

$$g(x) = g_0 + g_1 x + g_2 x^2 + \cdots\cdots + g_{2t-1} x^{2t-1} + x^{2t} \quad (4)$$

El grado del polinomio generador es igual al número de símbolos de paridad. Los códigos Reed-Solomon son un subconjunto de los códigos BCH. Por lo tanto, esta relación entre el grado del polinomio generador y el número de símbolos de paridad se mantiene igual que para los códigos BCH. Dado que los polinomios generadores son de grado $2t$, debe haber precisamente $2t$ sucesivas potencias de la raíz primitiva del campo finito α, que son raíces del polinomio.

Para un código corrector de *t* errores, designamos las raíces del polinomio generador $g(x)$ como: α , α^2, … , α^{2t}. Consideremos como ejemplo, el código Reed-Solomon (15,9) sobre el campo finito $GF(2^4)$ de triple corrección de errores *t*=3. Se describe el polinomio generador en términos de sus $2t = n - k = 15 - 9 = 6$ raíces, como sigue:

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i) \quad (5)$$

De la ecuación (5) tenemos que:

$$g(x) = \prod_{i=1}^{6} (x - \alpha^i) = (x - \alpha^1)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6)$$

Debido a que en el campo binario de orden 2, $GF(2)$, la aritmética es módulo 2, se tiene: $+1 = -1$, y el polinomio generador $g(x)$ puede ser expresado como.

$$g(x) = (x + \alpha^1)(x + \alpha^2)(x + \alpha^3)(x + \alpha^4)(x + \alpha^5)(x + \alpha^6)$$

Sea α un elemento primitivo del campo finito $GF(2^4)$, raíz del polinomio irreducible y primitivo $p(x) = x^4 + x + 1$, por lo que $1 + \alpha + \alpha^4 = 0$. De esta manera, haciendo el cálculo en el campo $GF(2^4)$, y siguiendo el formato de orden bajo, a orden alto, el polinomio generador $g(x)$ queda como.

$$g(x) = \alpha^6 + \alpha^9 x^1 + \alpha^6 x^2 + \alpha^4 x^3 + \alpha^{14} x^4 + \alpha^{10} x^5 + x^6 \quad (6)$$

Usando circuitos para codificar una secuencia de 9 símbolos en forma sistemática, con un código Reed-Solomon (15,9) descrito por $g(x)$ en la ecuación (6), se requiere la aplicación de un LFSR (linear feedback shift register, o registro de desplazamiento con

retroalimentación lineal), como se muestra en la Figura 9. Se puede verificar fácilmente que los factores de los multiplicadores en la Figura 9, tomados de izquierda a derecha, corresponden con los coeficientes del polinomio generador de la ecuación (6).

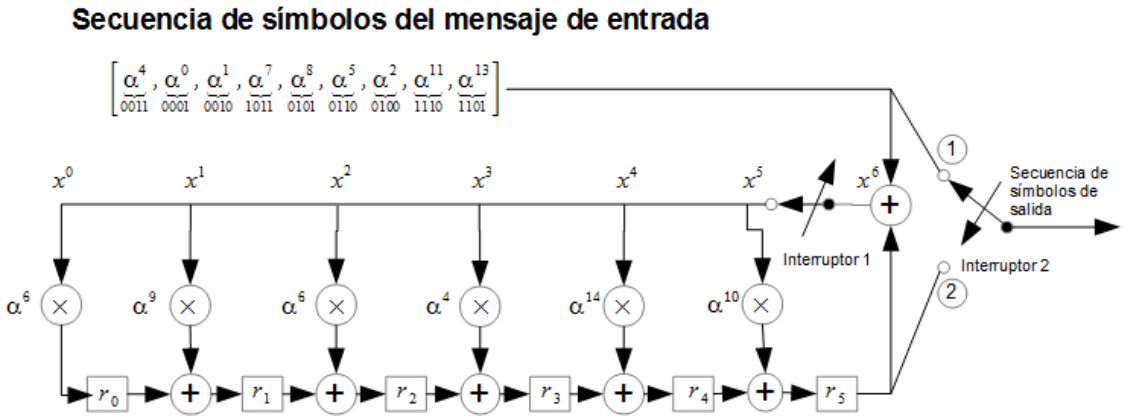

Figura 9 Codificador LFSR para un Reed-Solomon (15,9)

El cálculo con operaciones no binarias realizado sobre el campo finito $GF(2^4)$ por el codificador de la Figura 9, procede de la siguiente manera [1].

1. El interruptor 1 permanece cerrado durante los primeros k ciclos de reloj para permitir el corrimiento de los símbolos del mensaje dentro de las (n-k) etapas del registro de corrimiento.
2. El interruptor 2 permanece en la posición 1 durante los primeros k ciclos de reloj para permitir la transferencia simultánea de los símbolos del mensaje directamente a un registro de salida (que no se muestra en la Figura 9).
3. Después de la transferencia del k-ésimo símbolo del mensaje al registro de salida, el interruptor 1 es abierto y el interruptor 2 pasa a la posición 2.
4. En los restantes (n-k) ciclos de reloj se limpian los símbolos de paridad contenidos en el registro de corrimiento, moviéndolos al registro de salida.
5. El número total de ciclos de reloj es igual a n, y el contenido del registro de salida es la palabra de código en forma de polinomios $p(x) + x^{n-k}m(x)$, donde $p(x)$ representa los símbolos de paridad, y $m(x)$ los símbolos del mensaje.

Usando los símbolos del mensaje que se muestran en la Figura 9, los símbolos de paridad que se obtienen de la operación de codificación aplicada por el codificador son: $\alpha^1, \alpha^4, \alpha^5, \alpha^8,\ \alpha^8\ y\ \alpha^1$. Por consiguiente, la palabra de salida, escrita en forma de polinomial, se puede expresar como.

$$U(x) = p(x) + x^{n-k}m(x) \qquad (7)$$

Donde:

$$x^{n-k}m(x) = x^6(\alpha^4 + \alpha^0 x + \alpha^1 x^2 + \alpha^7 x^3 + \alpha^8 x^4 + \alpha^5 x^5 + \alpha^2 x^6 + \alpha^{11} x^7 + \alpha^{13} x^8)$$

$$p(x) = \alpha^1 + \alpha^4 x^1 + \alpha^5 x^2 + \alpha^8 x^3 + \alpha^8 x^4 + \alpha^1 x^5$$

$$U(x) = \alpha^1 + \alpha^4 x^1 + \alpha^5 x^2 + \alpha^8 x^3 + \alpha^8 x^4 + \alpha^1 x^5 + \alpha^4 x^6 + \alpha^0 x^7 + \alpha^1 x^8 + \alpha^7 x^9 + \alpha^8 x^{10} + \alpha^5 x^{11} + \alpha^2 x^{12} + \alpha^{11} x^{13} + \alpha^{13} x^{14}$$

Por ser un código cíclico, las raíces del polinomio generador $g(x)$ deben ser también las raíces de la palabra de código $U(x)$ generada por $g(x)$, por lo que una palabra de código $U(x)$ válida es de la forma.

$$U(x) = m(x)g(x) \tag{8}$$

Por lo tanto una palabra de código $U(x)$ arbitraria, cuando se evalúa en cualquier raíz de $g(x)$, debe ser cero. En otras palabras, esto significa comprobar que los síndromes son nulos.

$$U(\alpha) = U(\alpha^2) = U(\alpha^3) = U(\alpha^4) = U(\alpha^5) = U(\alpha^6) = 0$$

3.3 Decodificación

En la Sección 3.1 un mensaje de prueba es codificado en forma sistemática mediante un código corrector de tres errores, $RS(15,9)$, el resultado de esta codificación es la palabra de código $U(x)$ dada por la ecuación (7). Ahora, supongamos que durante la transmisión esta palabra de código se corrompe, de modo que a lo sumo 3 símbolos son recibidos con error. Este número de errores corresponde con la máxima capacidad de corrección de errores del código, el patrón de error puede ser descrito en forma de polinomio como.

$$e(x) = \sum_{n=0}^{14} e_n x^n \tag{9}$$

Donde $e_n \in GF(2^4)$, y el mensaje corrupto es $U(x) + e(x)$. Para este ejemplo tenemos que los errores ocurren en las posiciones de las potencias 5^0, 6^0 y 7^0, con valores α^4, α^2 y α^5, respectivamente, dando el polinomio de error.

$$e(x) = 0x^0 + 0x^1 + 0x^2 + 0x^3 + 0x^4 + \alpha^4x^5 + \alpha^2x^6 + \alpha^5x^7 + 0x^8 + 0x^9 + 0x^{10} + 0x^{11} + 0x^{12} + 0x^{13} + 0x^{14}$$

$$e(x) = \alpha^4x^5 + \alpha^2x^6 + \alpha^5x^7 \quad (10)$$

En el patrón de error podemos constatar que, un símbolo de paridad se ha dañado con un error de dos bits (en $GF(2^4)$ correspondiente a α^4), y dos símbolos de información adicionales de la palabra de código han sido también dañados, el primero ha sido dañado con error de un bit (en $GF(2^4)$ α^2) y el segundo símbolo de información ha sido dañado con un error de dos bits (en $GF(2^4)$ α^5). Considerando un ruido aditivo, la palabra recibida $r(x)$ es entonces representada por la suma de la palabra transmitida $U(x)$ y el patrón de error $e(x)$ como sigue:

$$r(x) = U(x) + e(x) \quad (11)$$

De la ecuación (11), se suma $e(x)$ de la ecuación (10), a $U(x)$ de la ecuación (7) usando la aritmética de $GF(2)(\alpha)/(\alpha^4 + \alpha + 1)$, para producir:

$$r(x) = \alpha^1 + \alpha^4x^1 + \alpha^5x^2 + \alpha^8x^3 + \alpha^8x^4 + \alpha^0x^5 + \alpha^{10}x^6 + \alpha^{10}x^7 + \alpha^1x^8 + \alpha^7x^9 + \alpha^8x^{10} + \alpha^5x^{11} + \alpha^2x^{12} + \alpha^{11}x^{13} + \alpha^{13}x^{14} \quad (12)$$

En este ejemplo de corrección de errores de tres símbolos no binarios, tenemos seis incógnitas, tres correspondientes a las tres posiciones de los errores y tres que corresponden a los valores de los símbolos de error. Puesto que hay seis incógnitas en este ejemplo, seis ecuaciones se requieren para su solución.

3.3.1 Calculo del síndrome

El síndrome es el resultado de una comprobación de paridad en $r(x)$, para determinar si $r(x)$ es un miembro valido del conjunto de 2^k palabras de código $U(x)$. De la teoría de códigos cíclicos, cuando $r(x)$ es una palabra de código, el vector de síndrome **S** tiene un valor nulo. Cualquier valor distinto de cero de **S** indica la presencia de errores. El síndrome **S** se compone de $(n-k)$ elementos, $\{S_i\}$ $(i = 1, \ldots\ldots, n-k)$. Por lo tanto para el código RS (15,9), hay seis elementos en el vector de síndrome.

A partir de la ecuación (8) que da lugar a cada palabra de código, se puede ver que cada palabra de código válida $U(x)$ es un múltiplo del polinomio generador $g(x)$. Por lo tanto

las raíces de $g(x)$ deben ser también las raíces de $U(x)$. Dado que $r(x) = U(x) + e(x)$, entonces $r(x)$ evaluada en cada una de las raíces de $g(x)$ debe ser cero, sí $e(x) = 0$. Entonces el cálculo de un elemento del síndrome puede realizarse como.

$$S_i = r(\alpha^i) \qquad i = 1, \dots\dots\dots, n-k \tag{13}$$

Para este ejemplo, los seis elementos del síndrome son calculados sobre $GF(2^4)$, como sigue:

$$S_1 = r(\alpha^1) = 0 \qquad S_2 = r(\alpha^2) = \alpha^4 \qquad S_3 = r(\alpha^3) = \alpha^7$$

$$S_4 = r(\alpha^4) = \alpha^6 \qquad S_5 = r(\alpha^5) = \alpha^9 \qquad S_6 = r(\alpha^6) = \alpha^1$$

3.3.2 Localización del error

Supongamos que hay t errores en la palabra de código en las posiciones $x^{j_1}, x^{j_2}, \dots\dots, x^{j_t}$. Entonces, el polinomio de error que se muestra en las ecuaciones (9) y (10), se puede escribir como.

$$e(x) = e_{j_1}x^{j_1} + e_{j_2}x^{j_2} + \cdots\cdots + e_{j_t}x^{j_t} \tag{14}$$

Donde los índices $1,2, \dots\dots, t$ se refieren al $1^{er}, 2^{do}, \dots\dots, t^{ésimo}$ error, y el índice j_t se refiere a la posición del error. Para corregir la palabra corrompida, cada valor de error e_{j_l} y su ubicación x^{j_l}, donde $l = 1,2, \dots\dots, t$ debe ser determinada. Definimos un número localizador de error como $\beta_l = \alpha^{j_l}$. A continuación, obtenemos los $n-k = 2t$ elementos de los síndromes sustituyendo la raíz de $g(x)$, α^i en el polinomio recibido $r(x)$ para $i = 1,2, \dots\dots, 2t$:

$$\begin{aligned} S_1 &= r(\alpha^1) = e_{j_1}\beta_1 + e_{j_2}\beta_2 + \cdots\cdots + e_{j_t}\beta_t \\ S_2 &= r(\alpha^2) = e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + \cdots\cdots + e_{j_t}\beta_t^2 \\ &\vdots \qquad\quad \vdots \qquad\quad \vdots \\ S_{2t} &= r(\alpha^{2t}) = e_{j_1}\beta_1^{2t} + e_{j_2}\beta_2^{2t} + \cdots + e_{j_t}\beta_t^{2t} \end{aligned} \tag{15}$$

En este sistema de ecuaciones hay $2t$ incógnitas (t valores erróneos y t ubicaciones), y $2t$ ecuaciones simultaneas.

Cuando un valor de síndromes distinto de cero ha sido calculado, significa que al menos un error ha sido recibido. A continuación, es necesario conocer la ubicación del error, o errores. Un polinomio localizador de error puede definirse como.

$$\sigma(x) = (1 + \beta_1 x)(1 + \beta_2 x) \cdots \cdots \cdots (1 + \beta_t x) \tag{16}$$

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots \cdots \cdots + \sigma_t x^t$$

Donde las raíces de $\sigma(x)$ son $\frac{1}{\beta_1}, \frac{1}{\beta_2}, \ldots\ldots\ldots, \frac{1}{\beta_t}$ donde $\beta_i \neq 0$ para $i = 1,2, \ldots, t$. El recíproco de las raíces de $\sigma(x)$ son las posiciones de los errores del patrón de error $e(x)$. Una vez que se han calculado los síndromes, se construye la matriz de los síndromes **A**. La matriz de síndromes **A** de la ecuación (17) es no singular si la palabra de código recibida contiene exactamente t errores. Si hay menos de t errores la matriz **A** es singular. Si **A** es singular, a continuación la columna más a la derecha y la fila inferior de **A** se eliminan y el determinante de la matriz resultante es calculado. Este proceso se repite hasta que la matriz resultante es no singular. Los coeficientes del polinomio localizador de errores son entonces calculados por la solución de la ecuación (18) empleando técnicas algebraicas estándar con operaciones realizadas en $GF(2^m)$, para $m = 4$ [14].

$$A = \begin{pmatrix} S_1 & S_2 & S_3 & \cdots & S_{t-1} & S_t \\ S_2 & S_3 & S_4 & & S_t & S_{t+1} \\ \vdots & & & \ddots & \vdots & \\ S_{t-1} & S_t & S_{t+1} & \cdots & S_{2t-3} & S_{2t-2} \\ S_t & S_{t+1} & S_{t+2} & & S_{2t-2} & S_{2t-1} \end{pmatrix} \tag{17}$$

$$\begin{pmatrix} S_1 & S_2 & S_3 & \cdots & S_{t-1} & S_t \\ S_2 & S_3 & S_4 & & S_t & S_{t+1} \\ \vdots & & & \ddots & \vdots & \\ S_{t-1} & S_t & S_{t+1} & \cdots & S_{2t-3} & S_{2t-2} \\ S_t & S_{t+1} & S_{t+2} & & S_{2t-2} & S_{2t-1} \end{pmatrix} \begin{pmatrix} \sigma_t \\ \sigma_{t-1} \\ \vdots \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} -S_{t+1} \\ -S_{t+2} \\ \vdots \\ -S_{2t-1} \\ -S_{2t} \end{pmatrix} \tag{18}$$

En el ejemplo, sea $r(x) = \alpha^1 + \alpha^4 x^1 + \alpha^5 x^2 + \alpha^8 x^3 + \alpha^8 x^4 + \alpha^0 x^5 + \alpha^{10} x^6 + \alpha^{10} x^7 + \alpha^1 x^8 + \alpha^7 x^9 + \alpha^8 x^{10} + \alpha^5 x^{11} + \alpha^2 x^{12} + \alpha^{11} x^{13} + \alpha^{13} x^{14}$ el polinomio recibido, y $S_1 = r(\alpha^1) = 0,\ S_2 = r(\alpha^2) = \alpha^4,\ S_3 = r(\alpha^3) = \alpha^7,\ S_4 = r(\alpha^4) = \alpha^6,\ S_5 = r(\alpha^5) = \alpha^9,$ $S_6 = r(\alpha^6) = \alpha^1$, la secuencia de síndromes obtenidos, y conociendo de antemano que hay t errores, usamos la matriz de dimensión más grande que tiene determinante distinto de cero. Para el código $RS(15,9)$ corrector de tres símbolos erróneos. La ecuación (18) se reduce al siguiente sistema de ecuaciones.

$$\begin{pmatrix} S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \\ S_3 & S_4 & S_5 \end{pmatrix}\begin{pmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} S_4 \\ S_5 \\ S_6 \end{pmatrix} \tag{19}$$

$$\begin{pmatrix} 0 & \alpha^4 & \alpha^7 \\ \alpha^4 & \alpha^7 & \alpha^6 \\ \alpha^7 & \alpha^6 & \alpha^9 \end{pmatrix}\begin{pmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha^6 \\ \alpha^9 \\ \alpha^1 \end{pmatrix} \tag{20}$$

Después de resolver la ecuación (20) empleando técnicas algebraicas estándar, con operaciones realizadas en $GF(2^4)$ para $\sigma_1, \sigma_2\ y\ \sigma_3$ tenemos que:

$$\begin{pmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha^3 \\ \alpha^6 \\ \alpha^0 \end{pmatrix} \tag{21}$$

De las ecuaciones (16) y (21) tenemos que

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 + \sigma_3 x^3 \tag{22}$$

$$\sigma(x) = 1 + \alpha^0 x + \alpha^6 x^2 + \alpha^3 x^3 \tag{23}$$

Las raíces de $\sigma(x)$ son los inversos de las ubicaciones de error. Una vez que estas raíces se encuentran, se conocerá la ubicación del error. En general, las raíces de $\sigma(x)$ pueden ser uno, o más, de los elementos del campo. Determinamos estas raíces por pruebas exhaustivas del polinomio $\sigma(x)$ con cada uno de los elementos del campo, como se muestra a continuación. Cualquier elemento x que produce $\sigma(x) = 0$ es una raíz y nos permite localizar un error:

$$\sigma(\alpha^0) = \alpha^2 \qquad \sigma(\alpha^1) = \alpha^9 \qquad \sigma(\alpha^2) = \alpha^3 \qquad \sigma(\alpha^3) = \alpha^{14} \qquad \sigma(\alpha^4) = \alpha^9$$

$$\sigma(\alpha^5) = \alpha^{13} \qquad \sigma(\alpha^6) = \alpha^{14} \qquad \sigma(\alpha^7) = \alpha^5 \qquad \sigma(\alpha^8) = 0 \qquad \sigma(\alpha^9) = 0$$

$$\sigma(\alpha^{10}) = 0 \qquad \sigma(\alpha^{11}) = \alpha^{11} \qquad \sigma(\alpha^{12}) = \alpha^8 \qquad \sigma(\alpha^{13}) = \alpha^{10} \qquad \sigma(\alpha^{14}) = \alpha^9$$

Una vez evaluado el polinomio $\sigma(x)$ en cada uno de los elementos del campo, tenemos que $\sigma(x)$ tiene tres raíces. Las posiciones de error son la inversa de las raíces del polinomio $\sigma(x)$. Por lo tanto, $\sigma(\alpha^8) = 0$ indica que una raíz aparece en $\frac{1}{\beta_l} = \alpha^8$. Así, $\beta_l = \frac{1}{\alpha^8} = \alpha^7$. Así mismo $\sigma(\alpha^9) = 0$ indica que otra raíz se encuentra en $\frac{1}{\beta_{l'}} = \alpha^9 \Rightarrow\ \beta_{l'} = \alpha^6$, por último

$\sigma(\alpha^{10}) = 0$ indica que una tercera raíz se encuentra en $\frac{1}{\beta_{l''}} = \alpha^{10} \Rightarrow \beta_{l''} = \alpha^5$, donde $l, l'\ y\ l''$ hacen referencia al $1^{er}, 2^{do}\ y\ 3^{er}$ error respectivamente. Puesto que hay 3 símbolos erróneos, el polinomio de error es de la forma.

$$e(x) = e_{j_1}x^{j_1} + e_{j_2}x^{j_2} + e_{j_3}x^{j_3} \tag{24}$$

Los tres errores fueron encontrados en $\alpha^5, \alpha^6\ y\ \alpha^7$. Entonces, para este ejemplo podemos designar a $\beta_l = \alpha^{j_l}$ como $\beta_1 = \alpha^{j_1} = \alpha^5$, $\beta_2 = \alpha^{j_2} = \alpha^6\ y\ \beta_3 = \alpha^{j_3} = \alpha^7$, por lo tanto la ecuación (24) queda como.

$$e(x) = e_{j_1}x^5 + e_{j_2}x^6 + e_{j_3}x^7 \tag{25}$$

3.3.3 Valores de error

Un error se ha indicado como e_{j_l}, donde el índice j_l se refiere a la ubicación del error y el índice l identifica al $l - ésimo$ error. Puesto que cada valor de error está acoplado a un lugar determinado, la notación puede simplificarse haciendo a e_{j_1} como e_1, e_{j_2} como e_2 y e_{j_3} como e_3. Ahora bien, la preparación para determinar los valores de error e_1, e_2 y e_3, asociados con las ubicaciones β_1, β_2 y β_3, consiste en usar cualquiera de las seis ecuaciones de los síndromes. De la ecuación (15), tenemos que.

$$\begin{aligned} S_1 &= r(\alpha^1) = e_1\beta_1 + e_2\beta_2 + e_3\beta_3 \\ S_2 &= r(\alpha^2) = e_1\beta_1^2 + e_2\beta_2^2 + e_3\beta_3^2 \\ S_3 &= r(\alpha^3) = e_1\beta_1^3 + e_2\beta_2^3 + e_3\beta_3^3 \end{aligned} \tag{26}$$

Podemos escribir estas ecuaciones como:

$$\begin{pmatrix} \beta_1^1 & \beta_2^1 & \beta_3^1 \\ \beta_1^2 & \beta_2^2 & \beta_3^2 \\ \beta_1^3 & \beta_2^3 & \beta_3^3 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix} \tag{27}$$

$$\begin{pmatrix} \alpha^5 & \alpha^6 & \alpha^7 \\ \alpha^{10} & \alpha^{12} & \alpha^{14} \\ \alpha^0 & \alpha^3 & \alpha^6 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} 0 \\ \alpha^4 \\ \alpha^7 \end{pmatrix} \quad (28)$$

Después de resolver la ecuación (27) con técnicas algebraicas estándar, con operaciones realizadas en $GF(2^4)$ para $e_1, e_2\ y\ e_3$ tenemos que:

$$\begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} = \begin{pmatrix} \alpha^4 \\ \alpha^2 \\ \alpha^5 \end{pmatrix} \quad (29)$$

3.3.4 Corrección del error

De las ecuaciones (29) y (25), el polinomio de error estimado queda como.

$$\hat{e}(x) = e_1 x^{j_1} + e_2 x^{j_2} + e_3 x^{j_3}$$

$$\hat{e}(x) = \alpha^4 x^5 + \alpha^2 x^6 + \alpha^5 x^7$$

Finalmente, para corregir el polinomio recibido $r(x)$ usando aritmética módulo 2, tenemos que:

$$\widehat{U}(x) = r(x) + \hat{e}(x) = U(x) + e(x) + \hat{e}(x)$$

$$r(x) = \alpha^1 + \alpha^4 x^1 + \alpha^5 x^2 + \alpha^8 x^3 + \alpha^8 x^4 + \alpha^0 x^5 + \alpha^{10} x^6 + \alpha^{10} x^7 + \alpha^1 x^8 + \alpha^7 x^9 + \alpha^8 x^{10} + \alpha^5 x^{11} + \alpha^2 x^{12} + \alpha^{11} x^{13} + \alpha^{13} x^{14}$$

$$\hat{e}(x) = \alpha^4 x^5 + \alpha^2 x^6 + \alpha^5 x^7$$

$$\Rightarrow \ (\alpha^0 + \alpha^4)x^5 + (\alpha^{10} + \alpha^2)x^6 + (\alpha^{10} + \alpha^5)x^7 = \alpha^1 x^5 + \alpha^4 x^6 + \alpha^0 x^7$$

$$\widehat{U}\,(x) = \alpha^1 + \alpha^4 x^1 + \alpha^5 x^2 + \alpha^8 x^3 + \alpha^8 x^4 + \alpha^1 x^5 + \alpha^4 x^6 + \alpha^0 x^7 + \alpha^1 x^8 + \alpha^7 x^9 + \alpha^8 x^{10} + \alpha^5 x^{11} + \alpha^2 x^{12} + \alpha^{11} x^{13} + \alpha^{13} x^{14}$$

Donde los símbolos del mensaje son los 9 símbolos más a la derecha de $\widehat{U}\,(x)$, por lo tanto el mensaje decodificado es:

$\alpha^4 = 0011$ $\alpha^0 = 0001$ $\alpha^1 = 0010$ $\alpha^7 = 1011$ $\alpha^8 = 0101$

$\alpha^5 = 0110$ $\alpha^2 = 0100$ $\alpha^{11} = 1110$ $\alpha^{13} = 1101$

4. CÓDIGOS BCH

4.1 Introducción

Una clase de códigos cíclicos de amplio uso y múltiple corrección de errores fue definida por Bose y Ray-Chaudhuri en 1960, y de forma independiente por Hocquenghem en 1959; estos códigos son conocidos como los códigos BCH. Los códigos BCH proveen una amplia variedad de longitudes de bloque y correspondientes tasas de código.

Las aplicaciones originales de los códigos BCH se restringieron a códigos binarios de longitud $2^m - 1$ para algún entero m. Estos fueron extendidos más tarde por Gorestein y Zierler (1961) a códigos no binarios con símbolos del campo de Galois $GF(q)$ [8]. Los códigos cíclicos más útiles no binarios relacionados con los códigos BCH son los códigos de Reed-Solomon (1960). El primer algoritmo de decodificación para códigos binarios BCH fue ideado por Peterson en 1960. Desde entonces el algoritmo de Peterson ha sido refinado por Berlekamp, Massey, Chien, Forney y muchos otros [2].

Los códigos BCH son códigos cíclicos que se construyen mediante la especificación de sus ceros, es decir, las raíces de su polinomio generador. Un código BCH con $d_{min} \geq 2t + 1$ es un código cíclico cuyo polinomio generador $g(x)$ tiene $2t$ raíces consecutivas [15].

Así para códigos BCH sobre cualquier campo finito se tiene las siguientes relaciones:

$$n - k \quad = \quad grado\big(g(x)\big) \quad \leq \quad m * t \quad y \quad n = q^m - 1$$

Dado un campo finito $GF(q)$ y un bloque de longitud $n \geq 3$, que es un divisor de $q^m - 1$ para algún entero positivo m, un código $BCH(n, k)$ de corrección de t errores sobre $GF(q)$ es generado cíclicamente por el polinomio [2],

$$g(x) = MCM\{m_b(x), m_{b+1}(x), \ldots., m_{b+2t-1}(x)\} \tag{30}$$

Donde $m_{b+i}(x)$ para $i = 0,1, \ldots \ldots, 2t - 1$ son los polinomios mínimos de las $2t$ potencias sucesivas $\alpha^b, \alpha^{b+1}, \ldots \ldots, \alpha^{b+2t-1}$ de un elemento de campo α, cuyo orden es n en alguna extensión del campo $GF(q^m)$. Además $MCM\{\}$ denota al polinomio mínimo común múltiplo. Normalmente, el elemento de campo α es un elemento primitivo en el campo extendido $GF(q^m)$, en cuyo caso el código es llamado un código BCH primitivo. El orden de un elemento tal es $n = q^m - 1$, que es también la longitud del código. Además, los códigos con $b = 1$ se dice que son códigos BCH en sentido estricto.

Sea $m_i(x)$ el polinomio mínimo de α^i. También sea $c(x) = c_0 + c_1 x + \cdots \cdots + c_{n-1} x^{n-1}$ un polinomio de código con coeficientes en $GF(2)$. Si $c(x)$ tiene $\alpha, \alpha^2, \ldots., \alpha^{2t}$ como sus raíces,

entonces $c(x)$ es divisible por los polinomios mínimos $m_1(x), m_2(x), \ldots., m_{2t}(x)$ de $\alpha, \alpha^2, \ldots, \alpha^{2t}$, respectivamente. El polinomio generador $g(x)$ del código BCH binario corrector de t errores en un bloque de longitud $n = 2^m - 1$, y tasa $\frac{k}{n}$, es el polinomio de menor grado sobre $GF(2)$ con estas raíces. Por lo tanto, el polinomio generador del código debe ser el mínimo común múltiplo de estos polinomios mínimos. Es decir [2].

$$g(x) = MCM\{m_1(x), m_2(x), \ldots., m_{2t}(x)\} \quad (31)$$

Donde $m_i(x)$ es el polinomio mínimo de α^i para $i = 1,2, \ldots.,2t$, y consta de las $2t$ potencias sucesivas del elemento α del campo, a partir de α^i. El orden de α es n en el campo extendido $GF(2^m)$.

Considerando como ejemplo el código $BCH(15,5)$ en sentido estricto con $t = 3$ y $b = 1$. Sea α un elemento primitivo del campo finito $GF(2^4)$, tal que $1 + \alpha + \alpha^4 = 0$. Si $m_i(x)$ denota el polinomio mínimo de α^i para $i = 1,2,3,4,5,6$. Entonces la lista de los polinomios mínimos para este código está dada en la Tabla 1.

Elementos del campo	Conjugados	Polinomios mínimos
α^0	--------	$m_0(x) = 1 + x$
α^1	$\alpha^2, \alpha^4, \alpha^8$	$m_1(x) = 1 + x + x^4$
α^2	$\alpha^4, \alpha^8, \alpha^{16} = \alpha$	$m_2(x) = 1 + x + x^4$
α^3	$\alpha^8, \alpha^{12}, \alpha^{24} = \alpha^9$	$m_3(x) = 1 + x + x^2 + x^3 + x^4$
α^4	$\alpha^8, \alpha^{16} = \alpha, \quad \alpha^{32} = \alpha^2$	$m_4(x) = 1 + x + x^4$
α^5	α^{10}	$m_5(x) = 1 + x + x^2$
α^6	$\alpha^{12}, \alpha^{24} = \alpha^9, \quad \alpha^{48} = \alpha^3$	$m_6(x) = 1 + x + x^2 + x^3 + x^4$
α^7	$\alpha^{14}, \alpha^{28} = \alpha^{13}, \alpha^{56} = \alpha^{11}$	$m_7(x) = 1 + x^3 + x^4$
α^8	$\alpha^{16} = \alpha, \alpha^{32} = \alpha^2, \alpha^{64} = \alpha^4$	$m_8(x) = 1 + x + x^4$
α^9	$\alpha^{18} = \alpha^3, \alpha^{36} = \alpha^6, \alpha^{72} = \alpha^{12}$	$m_9(x) = 1 + x + x^2 + x^3 + x^4$
α^{10}	$\alpha^{20} = \alpha^5$	$m_{10}(x) = 1 + x + x^2$
α^{11}	$\alpha^{22} = \alpha^7, \alpha^{44} = \alpha^{14}, \alpha^{88} = \alpha^{13}$	$m_{11}(x) = 1 + x^3 + x^4$
α^{12}	$\alpha^{24} = \alpha^9, \alpha^{48} = \alpha^3, \alpha^{96} = \alpha^6$	$m_{12}(x) = 1 + x + x^2 + x^3 + x^4$
α^{13}	$\alpha^{26} = \alpha^{11}, \alpha^{52} = \alpha^7, \alpha^{104} = \alpha^{14}$	$m_{13}(x) = 1 + x^3 + x^4$
α^{14}	$\alpha^{28} = \alpha^{13}, \alpha^{56} = \alpha^{11}, \alpha^{112} = \alpha^7$	$m_{14}(x) = 1 + x^3 + x^4$

Tabla 1 Polinomios mínimos de los elementos en $GF(2^m)$ para $m = 4$

De la Tabla 1 se puede observar que:

$$m_1(x) = m_2(x) = m_4(x) = m_8(x)$$

$$m_3(x) = m_6(x) = m_9(x) = m_{12}(x)$$

$$m_5(x) = m_{10}(x)$$

$$m_7(x) = m_{11}(x) = m_{13}(x) = m_{14}(x)$$

Por lo tanto de la Tabla 1 se tiene que $m_0(x), m_1(x), m_3(x), m_5(x)\ y\ m_7(x)$ son los únicos cinco polinomios irreducibles distintos de cero de $GF(2^4)$. Así, el polinomio generador para el código binario $BCH(15{,}5)$ en sentido estricto de triple corrección de error $t = 3$, $b = 1$ y de longitud $n = 15$ de acuerdo con la ecuación (30) está dado por

$$g(x) = MCM\{m_1(x), m_2(x), m_3(x), m_4(x), m_5(x), m_6(x)\}$$

$$g(x) = MCM\{m_1(x), m_3(x), m_5(x)\}$$

$$m_1(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8) = 1 + x + x^4$$

$$m_3(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^9) = 1 + x + x^2 + x^3 + x^4$$

$$m_5(x) = (x - \alpha^5)(x - \alpha^{10}) = 1 + x + x^2$$

$$g(x) = (1 + x + x^4)(1 + x + x^2 + x^3 + x^4)(1 + x + x^2)$$

$$g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10} \qquad (32)$$

De la Tabla 1 se puede ver de manera más general que cada potencia par de α, tiene el mismo polinomio mínimo que algunas potencias impar de α precedentes. Como consecuencia, el polinomio generador $g(x)$ del código binario BCH corrector de t errores de longitud $2^m - 1$, dado por la ecuación (31), puede ser reducido a

$$g(x) = MCM\{m_1(x), m_3(x), \ldots\ldots, m_{2t-1}(x)\} \qquad (33)$$

Donde el grado de cada $m_i(x)$ es inferior, o igual a m. Por lo tanto, la ecuación (33) implica que $c(x)$ es una palabra de código, sí y solo sí, $\alpha,\ \alpha^3, \ldots\ldots, \alpha^{2t-1}$ son las raíces de $c(x)$. Dado que el grado de cada polinomio mínimo es m, o inferior, el mayor valor posible para el grado de $g(x)$ es mt. Por lo tanto, el código tiene a lo más mt dígitos de chequeo de paridad, es decir, $n - k \leq mt$.

4.2 Codificación

En general, para cualquier par de enteros positivos m, t tales que $m \geq 3$ y $t < \frac{n}{2}$, existe un código BCH binario con longitud $n = 2^m - 1$, donde el número de bits de chequeo de paridad satisface $n - k \leq mt$, y la distancia mínima del código d_{min} satisface $d_{min} \geq 2t + 1$.

El proceso de codificación de los códigos BCH consta de los siguientes pasos:

1. Elegir un polinomio binario primitivo de grado m, y construir $GF(2^m)$.
2. Encontrar los polinomios mínimos $m_i(x)$ de α^i para $i = 1,3, \ldots, 2t - 1$.
3. Obtener $g(x) = MCM\{m_1(x), m_3(x), \ldots\ldots, m_{2t-1}(x)\}$
4. Realizar la codificación en forma sistemática, que involucra el cálculo de los bits de paridad como resultado de la división del polinomio mensaje $x^{n-k}i(x)$ por el polinomio generador, es decir $r(x) = x^{n-k}i(x)\ mod\ g(x)$

Usando circuitos para codificar una secuencia de bits de información en forma sistemática con un código binario BCH se requiere, al igual que para el código de Reed-Solomon, la aplicación de un LFSR. En la Figura 10 se muestra la forma general de un codificador para los códigos cíclicos, mientras que en la Figura 11 se muestra el caso particular del codificador $BCH(15,5)$ definido por $g(x)$ en la ecuación (31).

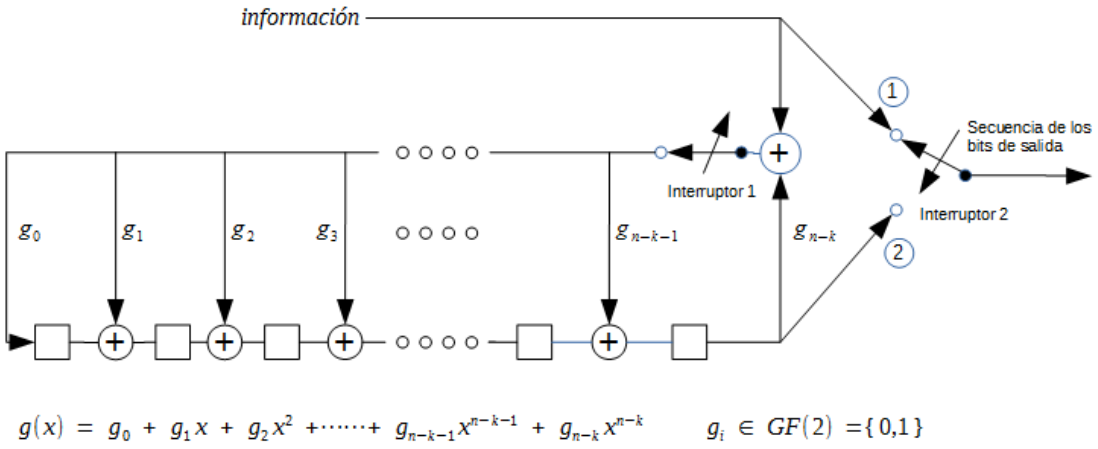

Figura 10 Circuito codificador, por división por $g(x)$

A continuación se desarrolla la codificación para el caso del código $BCH(15,5)$.

1. Polinomio primitivo de grado $m = 4$ es: $x^4 + x + 1$
2. Polinomios mínimos de $\alpha^1, \alpha^3, \alpha^5$, respectivamente:
   $m_1(x) = 1 + x + x^4$
   $m_3(x) = 1 + x + x^2 + x^3 + x^4$
   $m_5(x) = 1 + x + x^2$

3. De la ecuación (31), tenemos que:
   $g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}$
4. Para realizar la codificación sistemática del código $BCH(15,5)$, se emplea el circuito codificador LFSR de la Figura 11.

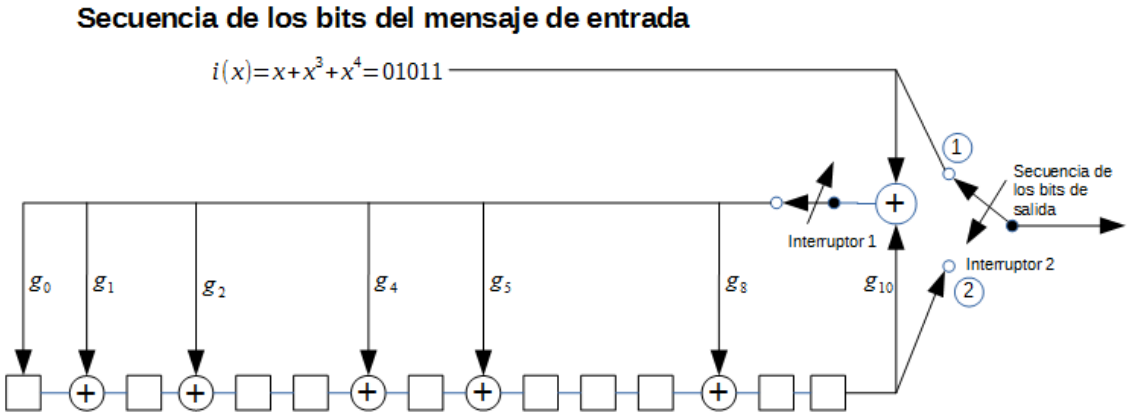

Figura 11 Codificador LFSR para un código $BCH(15,5)$

La operación binaria aplicada por el codificador de la Figura 11, donde $g_i = 1$, $i = 0,1,2,4,5,8,10$, y $g_i = 0$ en caso contrario, procede de la siguiente manera.

1. El interruptor 1 se mantiene cerrado los primeros k corrimientos, para permitir la transmisión de los bits del mensaje dentro de las $n - k$ etapas de codificación del registro de corrimiento.
2. El interruptor 2 está en la posición 1 para permitir la transmisión de los bits del mensaje directamente a un registro de salida durante los primeros k corrimientos.
3. Después de la transmisión del k-ésimo bit del mensaje, el interruptor 1 se abre y el interruptor 2 pasa a la posición 2, para permitir la transferencia de $p(x)$ al registro de salida.
4. Durante el restante de los $n - k$ corrimientos, el registro de codificación es limpiado por el movimiento de los bits de paridad $p(x)$ al registro de salida.
5. El número total de corrimientos es igual a n, y el contenido del registro de salida es el polinomio de la palabra de código $c(x) = i(x)x^{n-k} + p(x)$

Usando el mensaje que se muestra en la Figura 11, la paridad que se obtiene de la operación binaria aplicada por el codificador es:

$$p(x) = x^0 + x^1 + x^5 + x^8 + x^9 \qquad (34)$$

La palabra de código correspondiente está dada por:

$$c(x) = x^0 + x^1 + x^5 + x^8 + x^9 + x^{11} + x^{13} + x^{14} \quad (35)$$

4.3 Decodificación

Para la decodificación del código BCH se emplea el mismo algoritmo que para el código de Reed-Solomon, por lo tanto escogiendo un patrón de ruido en particular, tenemos que.

$$e(x) = x^0 + x^7 + x^{10} \quad (36)$$

Considerando el ruido aditivo, la palabra recibida es $r(x) = c(x) + e(x)$, por lo tanto tenemos que,

$$r(x) = x^1 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14} \quad (37)$$

4.3.1 Calculo del síndrome

El primer paso en la obtención de un procedimiento de corrección de errores es calcular qué información de las comprobaciones de paridad, que son el síndrome, se tiene sobre los errores. Supongamos que una palabra de código $c(x)$ se transmite, y se producen errores que resultan en un polinomio recibido $r(x) = c(x) + e(x)$. Entonces, considerando los resultados de sustituir las raíces del polinomio generador: $\alpha^b, \alpha^{b+1}, \dots, \alpha^{b+2t-1}$ en el polinomio $r(x)$. Dado que $c(x)$ es una palabra de código y por lo tanto tiene estos elementos también como sus raíces, el resultado es $e(\alpha^b)$, $e(\alpha^{b+1}), \dots, e(\alpha^{b+2t-1})$ [21].

El patrón de error $e(x)$ puede ser descrito por una lista de valores y ubicaciones de sus componentes distintos de cero. La ubicación se da en términos de un número de ubicación de error, que es simplemente α^j para el $(n-j)$ $ésimo$ símbolo. Por lo tanto, cada componente distinto de cero de $e(x)$ es descrito por un par de elementos del campo, Y_i (la magnitud del error), y X_i (el número de ubicación del error); Y_i es un elemento de $GF(q)$ y X_i es un elemento de $GF(q^m)$. Si se producen t errores, hay t componentes distintos de cero de $e(x)$, y por lo tanto se requieren t pares (X_i, Y_i) para describir los errores [21].

Entonces, en términos de los pares (X_i, Y_i)

$$e(\alpha^j) = \sum_{i=1}^{t} Y_i X_i^j = S_j \quad (38)$$

y los valores del síndrome $S_j = e(\alpha^j)$ están dados por los cálculos de verificación de paridad para $b \leq j \leq b + 2t - 1$.

Los campos de Galois que se pueden realizar tomando clases residuales de polinomios modulo un polinomio irreductible sobre $GF(p)$ se dice que son campos de característica p. Por lo tanto $GF(p^m)$ es un campo de característica p para cualquier elección de m. Entonces

$$(a+b)^p = a^p + \binom{p}{1} a^{p-1}b + \binom{p}{2} a^{p-2}b^2 + \cdots + b^p \tag{39}$$

Y todos los coeficientes binomiales $\binom{\mathrm{p}}{\mathrm{i}}$ para $0 < \mathrm{i} < \mathrm{p}$ tienen a p como un factor, y por lo tanto son cero ; de modo que en un campo con característica p, se cumple que [21]

$$(a+b)^p = a^p + b^p \tag{40}$$

Por la ecuaciones (38) y (40) tenemos que

$$\left(S_j\right)^q = \left(\sum_{i=1}^{t} Y_i X_i^j\right)^q = \sum_{i=1}^{t} Y_i^q X_i^{jq} = \sum_{i=1}^{t} Y_i X_i^{jq} = S_{(jq)} \tag{41}$$

En el caso binario, tenemos que $q = 2$ y como el código BCH es un código BCH en sentido estricto, entonces $b = 1$. Por lo tanto la ecuación (41) queda como,

$$\left(S_j\right)^2 = S_{(2j)} \implies S_j^2 = S_{2j} \tag{42}$$

Debido a la presencia de errores y dado que los códigos BCH son cíclicos, al evaluar la palabra recibida $r(x)$ de la ecuación (37), en las raíces del polinomio generador obtendremos los síndromes, estos nos indican la presencia de errores en $r(x)$; por lo tanto, si al evaluar $r(x)$ en las raíces de $g(x)$ los síndromes son diferentes de cero, se concluye que $r(x)$ no es palabra de código válida.

$$S_1 = r(\alpha^1) = \alpha^{13} \qquad S_3 = r(\alpha^3) = \alpha^6 \qquad S_5 = r(\alpha^5) = \alpha^0$$

Para calcular el resto de los síndromes se utiliza la igualdad $S_{2j} = S_j^2$, de la ecuación (42)

$$S_2 = (S_1)^2 = \alpha^{11} \qquad S_4 = (S_2)^2 = \alpha^7 \qquad S_6 = (S_3)^2 = \alpha^{12}$$

4.3.2 Localización del error

De la ecuación (18), construimos el sistema de ecuaciones y resolvemos el sistema de ecuaciones para $t = 3$ errores, que es la máxima capacidad de corrección de errores de este código $BCH(15,5)$, por lo tanto la matriz de síndromes A de mayor dimensión es no singular para este código $BCH(15,5)$ de triple corrección de errores.

$$\begin{pmatrix} S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \\ S_3 & S_4 & S_5 \end{pmatrix} \begin{pmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} S_4 \\ S_5 \\ S_6 \end{pmatrix} \tag{43}$$

$$\begin{pmatrix} \alpha^{13} & \alpha^{11} & \alpha^6 \\ \alpha^{11} & \alpha^6 & \alpha^7 \\ \alpha^6 & \alpha^7 & \alpha^0 \end{pmatrix} \begin{pmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha^7 \\ \alpha^0 \\ \alpha^{12} \end{pmatrix} \tag{44}$$

Después de resolver la ecuación (44) para $\sigma_1, \sigma_2\ y\ \sigma_3$ tenemos que

$$\begin{pmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha^2 \\ \alpha^3 \\ \alpha^{13} \end{pmatrix} \tag{45}$$

De las ecuaciones (16) y (45) tenemos que

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 + \sigma_3 x^3 \tag{46}$$

$$\sigma(x) = 1 + \alpha^{13} x + \alpha^3 x^2 + \alpha^2 x^3 \tag{47}$$

La última etapa del algoritmo consiste en determinar las raíces de $\sigma(x)$, las raíces de $\sigma(x)$ pueden ser uno, o más, de los elementos del campo. Determinamos estas raíces por pruebas exhaustivas del polinomio $\sigma(x)$ con cada uno de los elementos del campo, esto es, evaluar a $\sigma(x)$ para $x = \alpha^i$ para $i \in [0, 1, 2, \dots \dots, n-1]$, por lo que cualquier elemento x que produce $\sigma(x) = 0$ es una raíz y nos permite localizar un error:

$$\sigma(\alpha^0) = 0 \qquad \sigma(\alpha^1) = \alpha^3 \qquad \sigma(\alpha^2) = \alpha^{11} \qquad \sigma(\alpha^3) = \alpha^{10} \qquad \sigma(\alpha^4) = \alpha^1$$

$$\sigma(\alpha^5) = 0 \qquad \sigma(\alpha^6) = \alpha^8 \qquad \sigma(\alpha^7) = \alpha^5 \qquad \sigma(\alpha^8) = 0 \qquad \sigma(\alpha^9) = \alpha^{12}$$

$$\sigma(\alpha^{10}) = \alpha^8 \qquad \sigma(\alpha^{11}) = \alpha^9 \qquad \sigma(\alpha^{12}) = \alpha^6 \qquad \sigma(\alpha^{13}) = \alpha^3 \qquad \sigma(\alpha^{14}) = \alpha^8$$

Una vez evaluado el polinomio $\sigma(x)$ en cada uno de los elementos del campo, tenemos que $\sigma(x)$ tiene tres raíces. Por lo tanto los errores están en los inversos multiplicativos,

$$\frac{1}{\beta_l} = \alpha^0 \implies \beta_l = \frac{1}{\alpha^0} = \alpha^0$$

$$\frac{1}{\beta_{l'}} = \alpha^5 \implies \beta_{l'} = \frac{1}{\alpha^5} = \alpha^{10}$$

$$\frac{1}{\beta_{l''}} = \alpha^8 \implies \beta_{l''} = \frac{1}{\alpha^8} = \alpha^7$$

4.3.3 Corrección del error

Puesto que hay 3 símbolos erróneos, el polinomio de error estimado es de la forma.

$$\hat{e}(x) = x^{j_1} + x^{j_2} + x^{j_3} \tag{42}$$

Los tres errores fueron encontrados en $\alpha^0, \alpha^7 \; y \; \alpha^{10}$. Entonces, podemos designar a $\beta_l = \alpha^{j_l}$ como $\beta_1 = \alpha^{j_1} = \alpha^0$, $\beta_2 = \alpha^{j_2} = \alpha^7 \; y \; \beta_3 = \alpha^{j_3} = \alpha^{10}$, por lo tanto la ecuación (42) queda como.

$$\hat{e}(x) = x^0 + x^7 + x^{10} \tag{43}$$

Finalmente la corrección se obtiene de sumar el polinomio de error estimado de la ecuación (43), con la palabra recibida de la ecuación (36).

$$\hat{c}(x) = r(x) + \hat{e}(x)$$

$$\hat{c}(x) = (x^1 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14}) + (x^0 + x^7 + x^{10})$$

$$\hat{c}(x) = x^0 + x^1 + x^5 + x^8 + x^9 + x^{11} + x^{13} + x^{14}$$

Donde los símbolos del mensaje son los 5 símbolos más a la derecha de $\hat{c}\ (x)$, por lo tanto el mensaje decodificado es:

$$\hat{c}(x) = x^0 + x^1 + x^5 + x^8 + x^9 + x^{11} + x^{13} + x^{14} = 110001001101011$$

$$i(x) = 01011$$

5. PRUEBAS Y RESULTADOS

Dado que el objetivo principal de esta tesis consiste en poder utilizar los radios definidos por *software* como equipo de comunicaciones digitales, para probar el algoritmo del código de bloque no binario de Reed-Solomon $RS(15,9)$ y el algoritmo del código de bloque binario $BCH(15,5)$, en el sistema de radio digital, primero se realizó la implementación de sus respectivos algoritmos de codificación y decodificación en *software* mediante el lenguaje de programación Python. En estos programas se puede ingresar la información y el patrón de error de manera interactiva con el usuario, y se muestra de manera detallada todo el proceso de codificación y decodificación para ambos códigos. A continuación se ilustra el diagrama de flujo del programa que implementa el código de Reed Solomon.

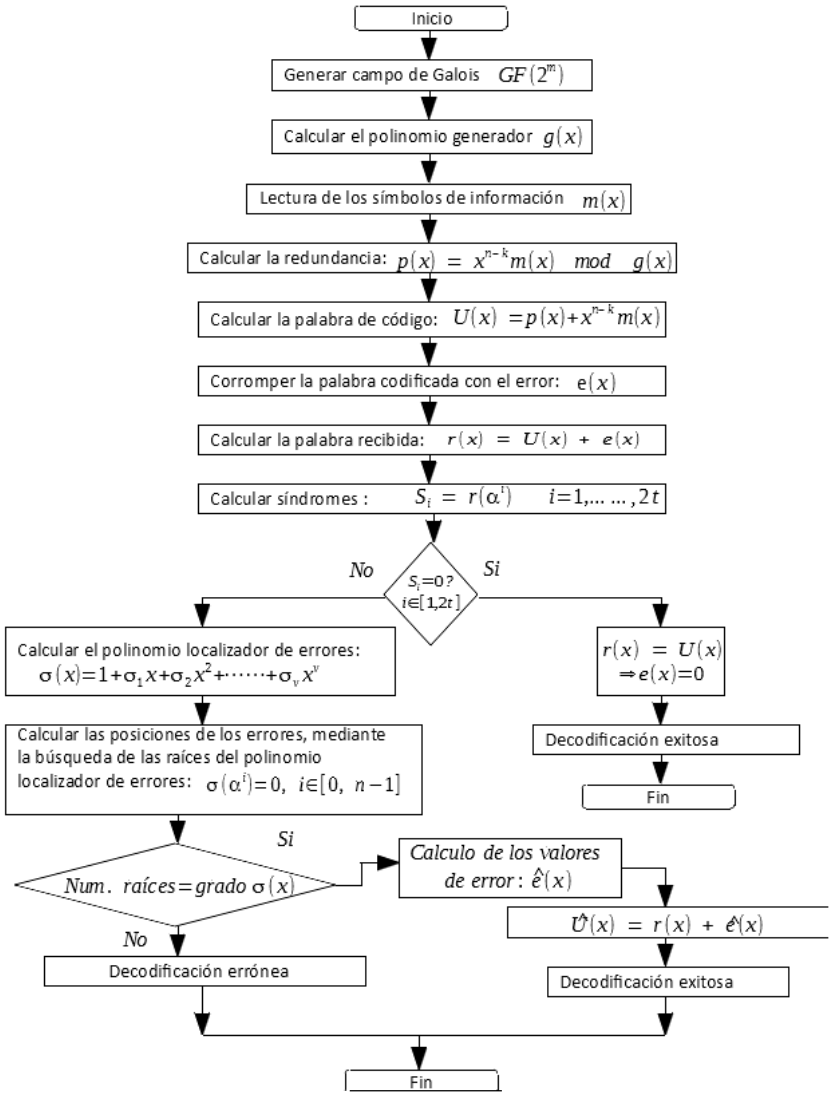

Figura 12 Diagrama de flujo del programa de Reed Solomon

Una vez implementado y probado el código de Reed-Solomon en *software*, se procedió a adaptarlo para trabajar dentro del ambiente gráfico GNU Radio. Como se observa en la Figura 13 la fuente de información para el código $RS(15,9)$, es el bloque "File Source". Con el propósito de poder comparar los resultados de manera visual y con una medida de calidad promedio, este bloque toma como datos a codificar una imagen digital. En nuestra experimentación escogimos una imagen monocromática de 256 líneas, por 256 columnas y 8 bits por pixel, denominada Lena256.bmp, la cual se convierte en un vector de caracteres de 8 bits. Por razones prácticas, como el código de $RS(15,9)$ trabaja sobre GF(2^4) con símbolos de 4 bits, este se adaptó a nivel de *software* y se acortó (*shortened*) para trabajar con vectores de cuatro elementos enteros de 8 bits. Estos vectores finalmente son codificados por el bloque $RS(15,9)$, que en la Figura 13 aparece como el bloque $RSE159$. Después de codificar la información esta es corrompida por el bloque $qasc3$ que simula el error de ráfaga. Este bloque recibe la información codificada, y dependiendo de la razón señal a ruido, modifica bloques de bits de esta, en función de los parámetros de Probabilidad Inversa y Error, que aparece en el bloque $qasc3$.

Una vez que la información codificada ha sido corrompida por el bloque $qasc3$, pasa a ser decodificada por el bloque $RSD159$ para ser corregida. En este sistema de simulación computacional, la decodificación puede dar dos casos:

1) Decodificación correcta: este caso se da cuando el número de errores es menor, o igual a t, que es la capacidad de corrección de errores del código.
2) Decodificación errónea: esta condición se da en dos casos, el primer caso, cuando los errores hacen que una palabra de código se convierta en otra palabra de código, y el segundo caso, cuando el número de errores es mayor que la capacidad de corrección de errores del código. En el primer caso cuando el decodificador evalúa los síndromes de la palabra recibida, estos serán nulos, y el decodificador asumirá que no hay errores, por lo que el error no será detectado. El segundo caso se da cuando el número de errores es mayor a la capacidad de corrección del código, entonces, cuando el decodificador evalúa los síndromes de la palabra recibida, estos serán distintos de cero, pero generarán una solución errónea, y la palabra decodificada no solo será incorrecta, sino que además tendrá aún más errores que los producidos por el canal, debido a la distancia mínima del código.

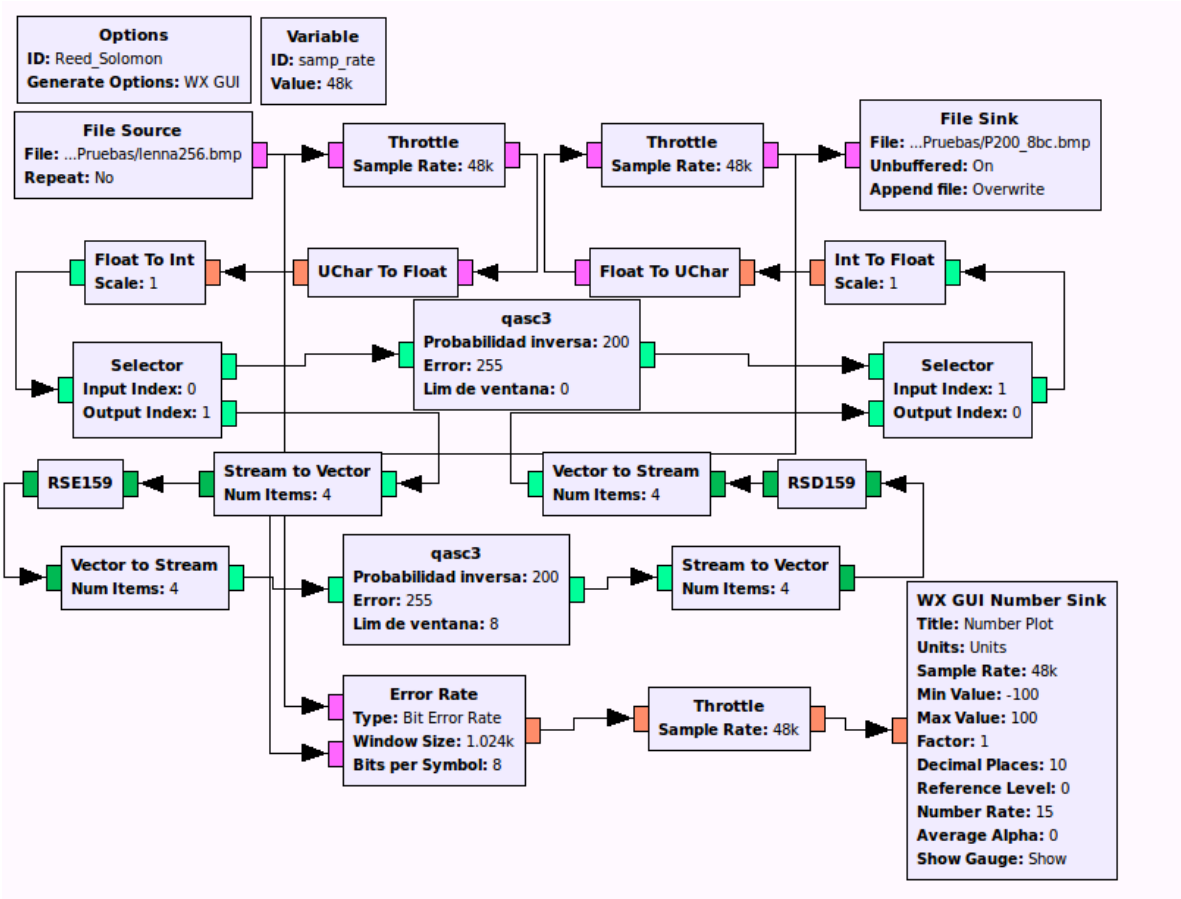

Figura 13 Grafo de prueba del código de Reed-Solomon

Figura 14 Imagen original

Para comparar la calidad de las imágenes decodificadas, se emplea el criterio de medida promedio de calidad más utilizado en procesamiento digital de imágenes: la razón señal pico a ruido *Peak Signal to Noise Ratio* (PSNR), definida por la ecuación (45). Siendo MSE el error medio cuadrático (ecuación (44)) entre la imagen original $f(x,y)$ y la imagen decodificada $F(x,y)$; en donde mientras más grande sea esta razón, mayor será la calidad de la imagen. En procesamiento digital de imágenes se considera que una PSNR superior o igual a 30 dB proporciona una buena calidad subjetiva.

$$MSE = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} [f(i,j) - F(i,j)]^2 \tag{44}$$

$$PSNR = 10 \log_{10} \left(\frac{255^2}{MSE} \right) \tag{45}$$

Para calcular las probabilidades de error de la imagen de Lenna, con codificación y sin codificación, procesada tanto con el código no binario de Reed-Solomon así como con el código binario BCH, se hace uso del bloque de GNU Radio Error Rate que se muestra en la Figura 13, este bloque calcula la tasa de bit en error (BER) sobre un número de muestras dadas por el tamaño de una ventana. Finalmente para comparar la calidad de las imágenes decodificadas, estas son procesadas en Matlab, donde se determina la PSNR de acuerdo con las ecuaciones (44) y (45).

A continuación se muestran los resultados obtenidos de la imagen de Lenna con ruido aditivo, con y sin control de errores. Los primeros resultados corresponden a la codificación con el código de bloque no binario de Reed-Solomon RS, y más adelante se presentan los resultados de la codificación con el código de bloque binario BCH.

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.089

BER con codificación RS = 0.012

PSNR calculada con Matlab

PSNR sin codificación = 16.52 dB

PSNR con codificación RS = 27.33 dB

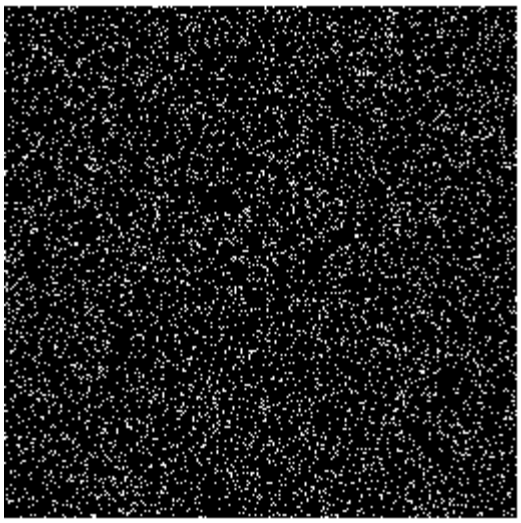

Figura 15 Diferencia entre la imagen original y la imagen con ruido

Figura 16 Imagen con ruido aditivo, sin control de errores, PSNR = 16.52 dB

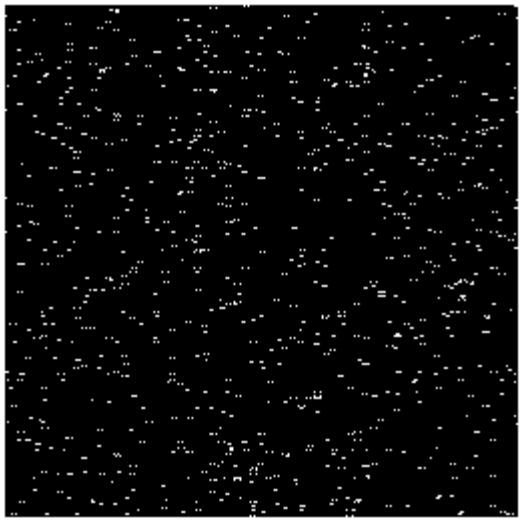

Figura 17 Diferencia entre la imagen original y la imagen decodificada

Figura 18 Imagen decodificada, con el código de Reed-Solomon, PSNR = 27.33 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.045

BER con codificación RS = 0.0033

PSNR calculada con Matlab

PSNR sin codificación = 19.42 dB

PSNR con codificación RS = 32.52 dB

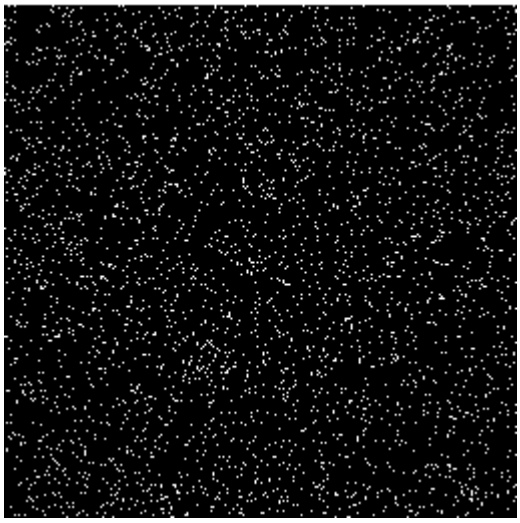

Figura 19 Diferencia entre la imagen original y la imagen con ruido

Figura 20 Imagen con ruido aditivo, sin control de errores, PSNR = 19.42 dB

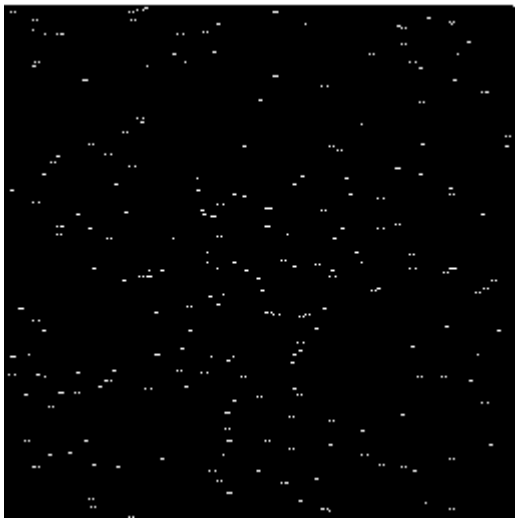

Figura 21 Diferencia entre la imagen original y la imagen decodificada

Figura 22 Imagen decodificada, con el código de Reed-Solomon, PSNR = 32.52 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.035

BER con codificación RS = 0.0027

PSNR calculada con Matlab

PSNR sin codificación = 20.54 dB

PSNR con codificación RS = 33.54 dB

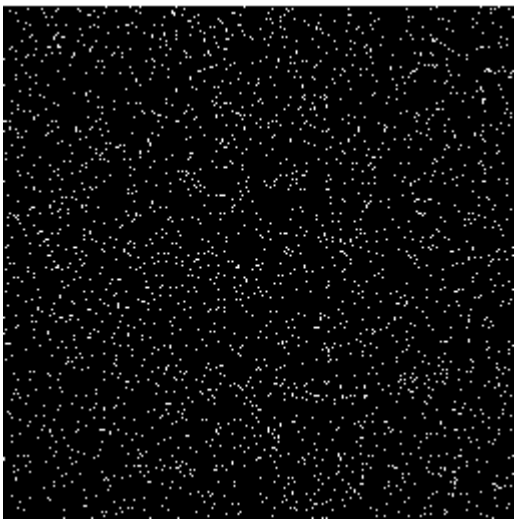

Figura 23 Diferencia entre la imagen original y la imagen con ruido

Figura 24 Imagen con ruido aditivo, sin control de errores, PSNR = 20.54 dB

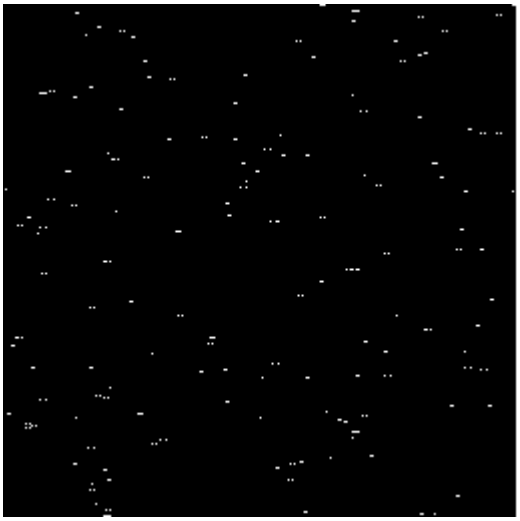

Figura 25 Diferencia entre la imagen original y la imagen decodificada

Figura 26 Imagen decodificada, con el código de Reed-Solomon, PSNR = 33.54 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.030

BER con codificación RS = 0.0016

PSNR calculada con Matlab

PSNR sin codificación = 21.08 dB

PSNR con codificación RS = 36.14 dB

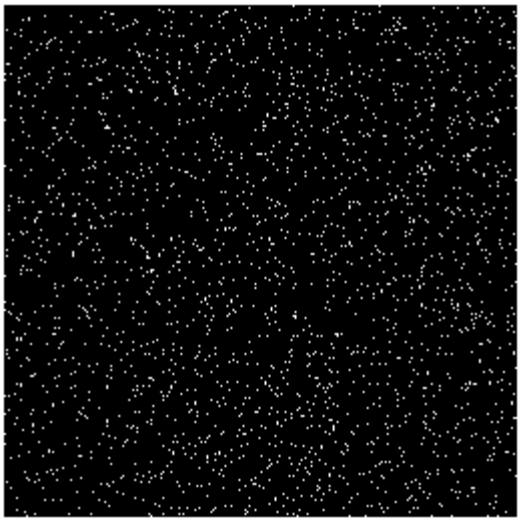

Figura 27 Diferencia entre la imagen original y la imagen con ruido

Figura 28 Imagen con ruido aditivo, sin control de errores, PSNR = 21.08 dB

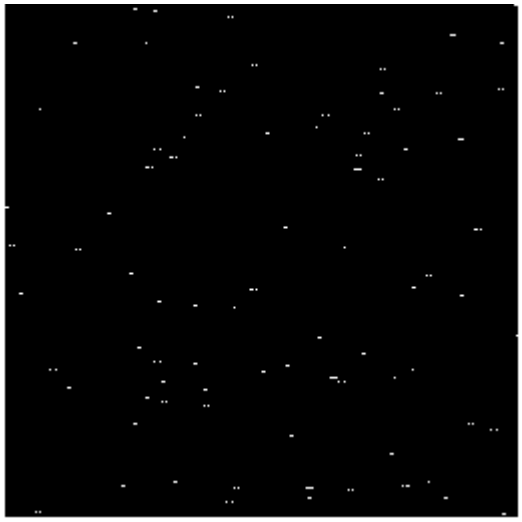

Figura 29 Diferencia entre la imagen original y la imagen decodificada

Figura 30 Imagen decodificada, con el código de Reed-Solomon, PSNR = 36.14 dB

Datos obtenidos del bloque Error Rate de GNU Radio
BER sin codificación = 0.018
BER con codificación RS = 0.00087

PSNR calculada con Matlab
PSNR sin codificación = 23.66 dB
PSNR con codificación RS = 39.10 dB

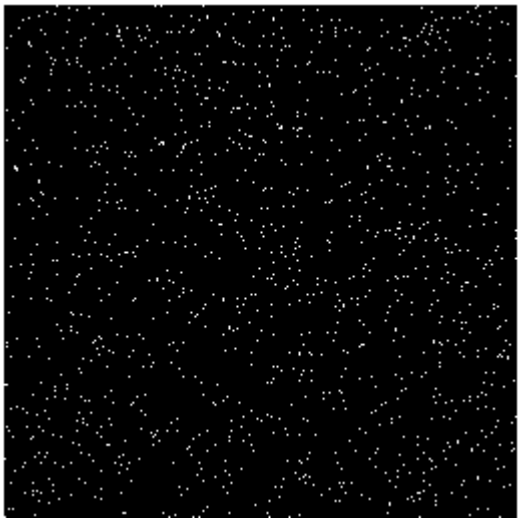

Figura 31 Diferencia entre la imagen original y la imagen con ruido

Figura 32 Imagen con ruido aditivo, sin control de errores, PSNR = 23.66 dB

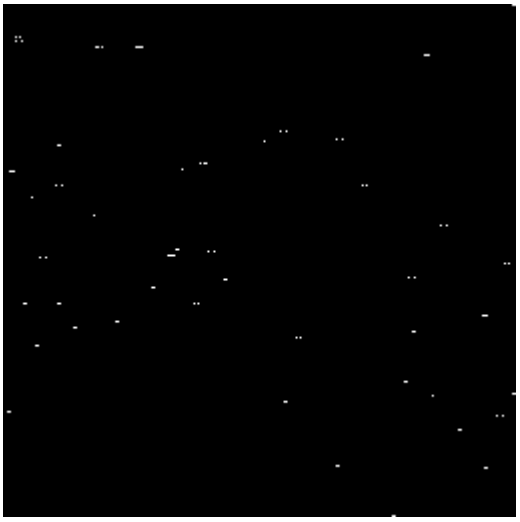

Figura 33 Diferencia entre la imagen original y la imagen decodificada

Figura 34 Imagen decodificada, con el código de Reed-Solomon, PSNR = 39.10 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.0092

BER con codificación RS = 0.00005

PSNR calculada con Matlab

PSNR sin codificación = 26.34 dB

PSNR con codificación RS = 46.96 dB

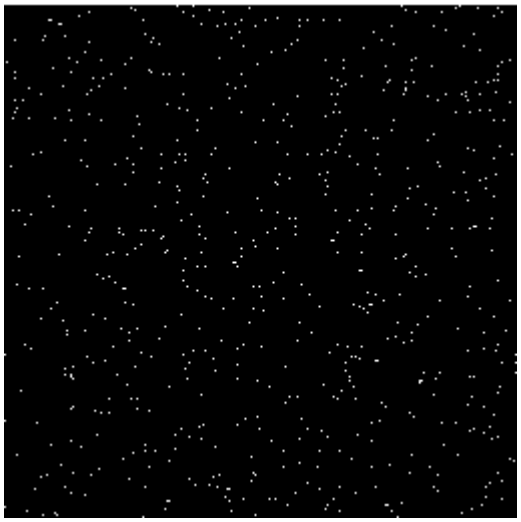

Figura 35 Diferencia entre la imagen original y la imagen con ruido

Figura 36 Imagen con ruido aditivo, sin control de errores, PSNR = 26.34 dB

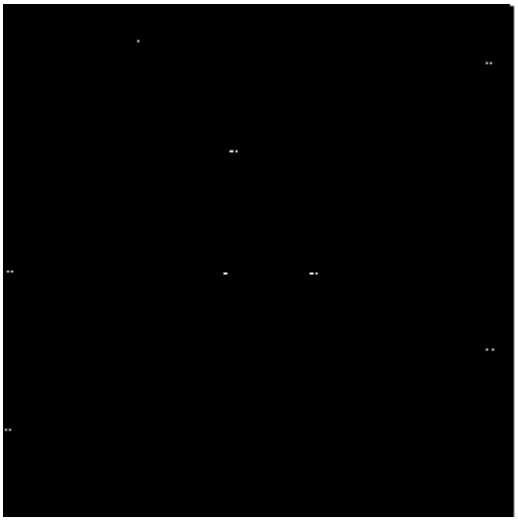

Figura 37 Diferencia entre la imagen original y la imagen decodificada

Figura 38 Imagen decodificada, con el código de Reed-Solomon, PSNR = 46.96 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.0041

BER con codificación RS = 0.0000000

PSNR calculada con Matlab

PSNR sin codificación = 29.00 dB

PSNR con codificación RS = 50.44 dB

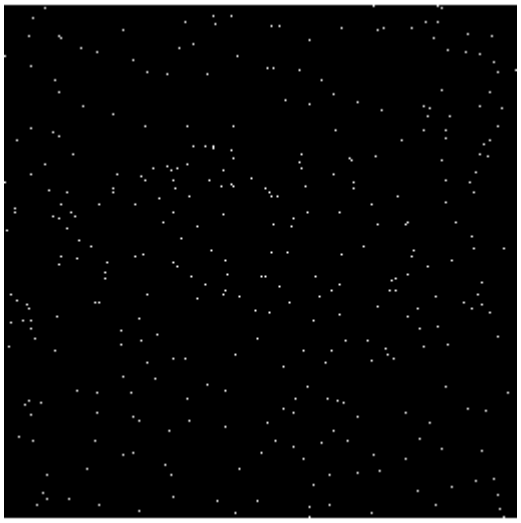

Figura 39 Diferencia entre la imagen original y la imagen con ruido

Figura 40 Imagen con ruido aditivo, sin control de errores, PSNR = 29.00 dB

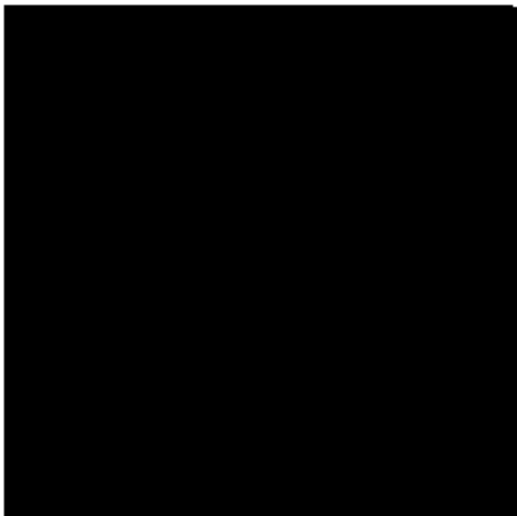

Figura 41 Diferencia entre la imagen original y la imagen decodificada

Figura 42 Imagen decodificada, con el código de Reed-Solomon, PSNR = 50.44 dB

Procediendo de igual manera que para el código de Reed-Solomon una vez implementado y probado el código BCH en *software*, se procedió a adaptarlo para trabajar dentro del ambiente gráfico GNU Radio. Como se observa en la Figura 43 la fuente de información para el código $BCH(15,5)$, es el bloque "File Source". Este bloque toma como datos a codificar una imagen digital monocromática de 256 líneas, por 256 columnas y 8 bits por pixel, denominada Lena256.bmp y la convierte en un vector de caracteres de 8 bits. Como el código $BCH(15,5)$ trabaja con 5 bits de información, este se adaptó para codificar los 8 bits del carácter de entrada con una doble codificación $BCH(15,5)$, donde 4 bits de la información son codificados en una primera codificación $BCH(15,5)$, y los cuatro bits restantes de la información son codificados en una segunda codificación BCH, quedando códigos acortados (*shortened*). Después de codificar la información esta es corrompida por el bloque $BSCI$ que simula el canal binario simétrico BSC [1].

Una vez que la información codificada ha sido corrompida por el bloque BCSI, pasa a ser decodificada por el bloque $BCHD155$ para ser corregida. Igualmente en este sistema de simulación por computadora, en la decodificación del código BCH se presentan dos casos, que son: decodificación correcta y decodificación errónea. Cuyo significado corresponde a lo ya explicado para el código de Reed-Solomon.

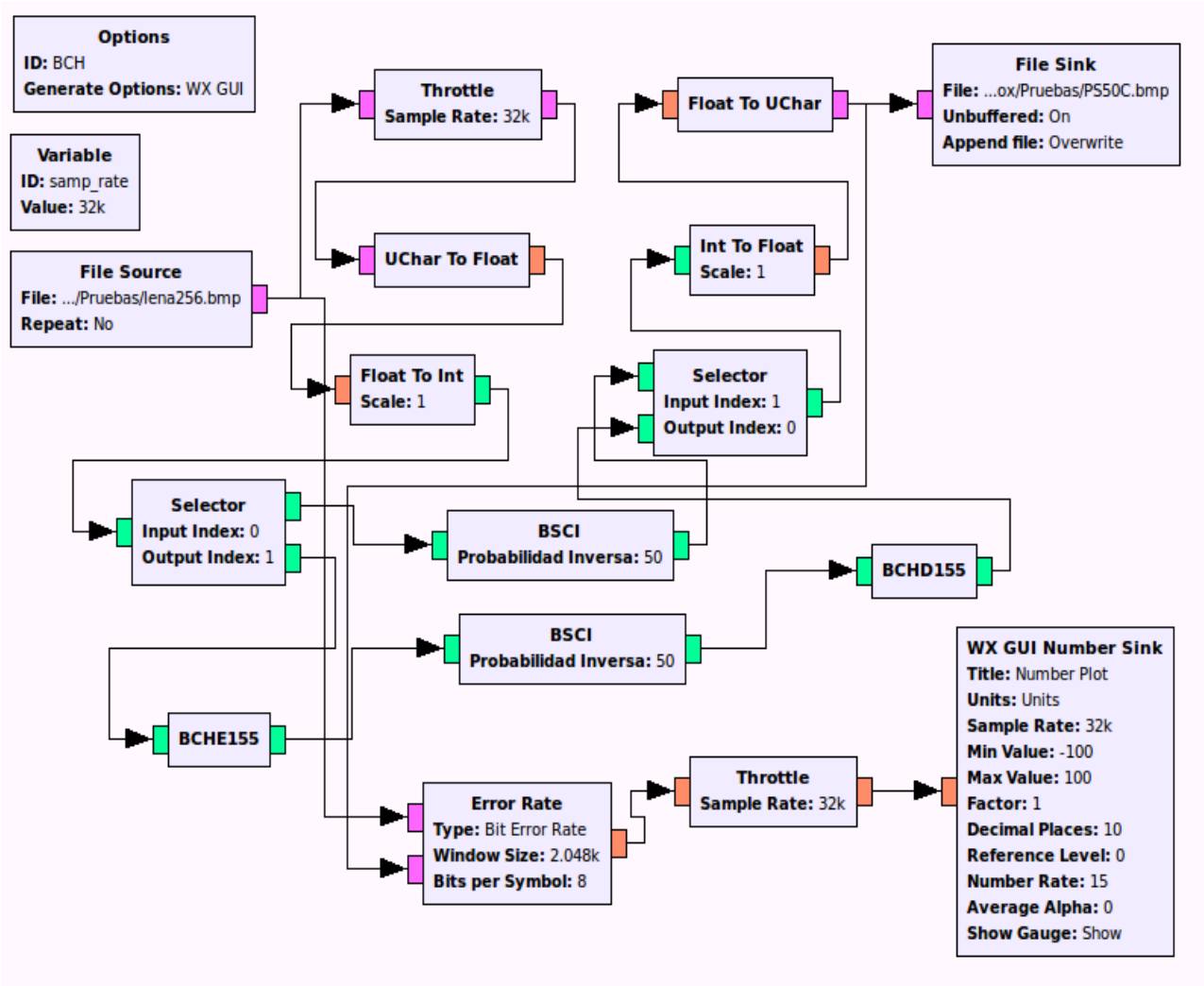

Figura 43 Grafo de prueba del código de BCH

En la Figura 44 se muestra el diagrama de flujo para el programa del código $BCH(15,5)$. Cabe mencionar que, tanto en el código BCH como en el código de Reed-Solomon en condiciones normales de operación no es posible determinar la decodificación errónea, ya que los bloques de codificación y decodificación son completamente independientes. En simulación por computadora, esta condición puede ser detectada por el bloque que calcula la tasa de bits en error *Bit Error Rate* (BER), que se muestra en el grafo de GNU Radio de la Figura 43 con el título de Error Rate. Este bloque compara en tiempo real las muestras de la imagen original con las muestras decodificadas que se obtienen a la salida de los decodificadores tanto de Reed-Solomon, como BCH.

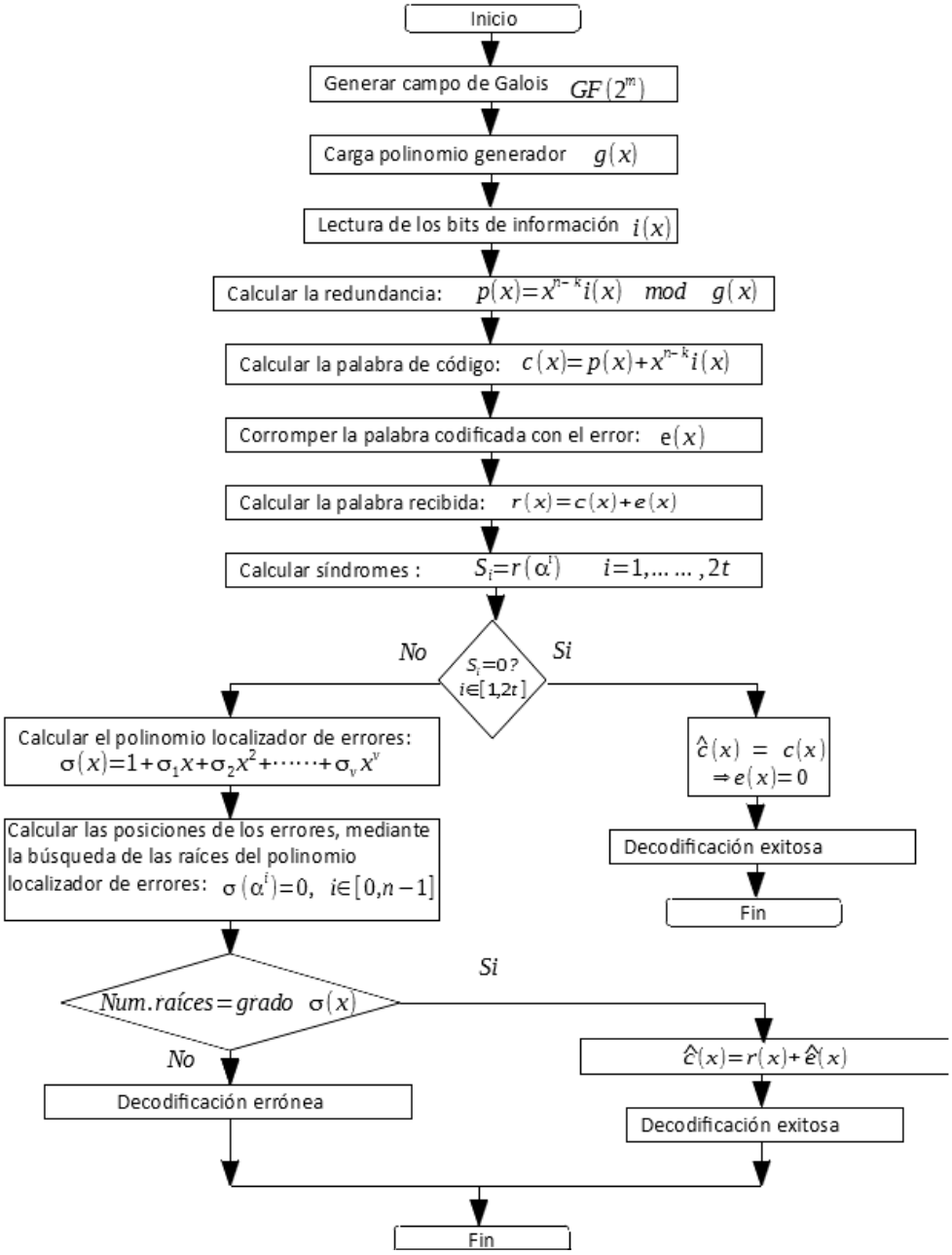

Figura 44 Diagrama de flujo del programa BCH

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.092

BER con codificación BCH = 0.019

PSNR calculada con Matlab

PSNR sin codificación = 17.16 dB

PSNR con codificación BCH = 23.30 dB

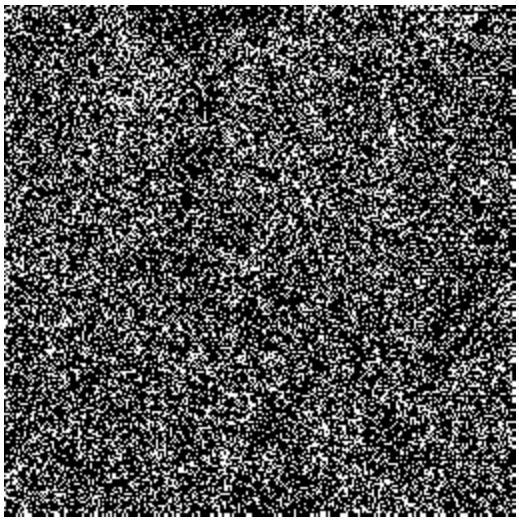

Figura 45 Diferencia entre la imagen original y la imagen con ruido

Figura 46 Imagen con ruido aditivo, sin control de errores, PSNR = 17.16 dB

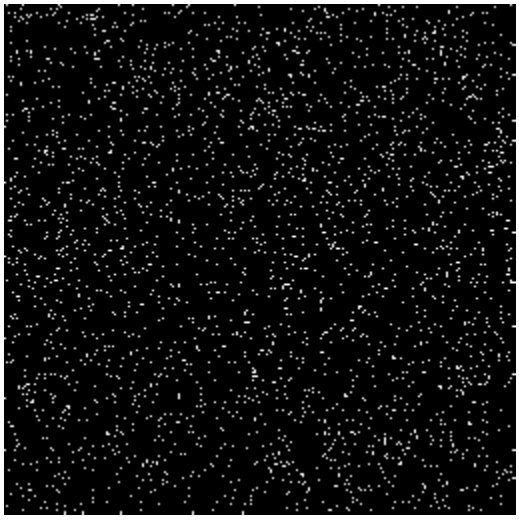

Figura 47 Diferencia entre la imagen original y la imagen decodificada

Figura 48 Imagen decodificada, con el código BCH, PSNR = 23.30 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.046

BER con codificación BCH = 0.0019

PSNR calculada con Matlab

PSNR sin codificación = 20.09 dB

PSNR con codificación BCH = 34.00 dB

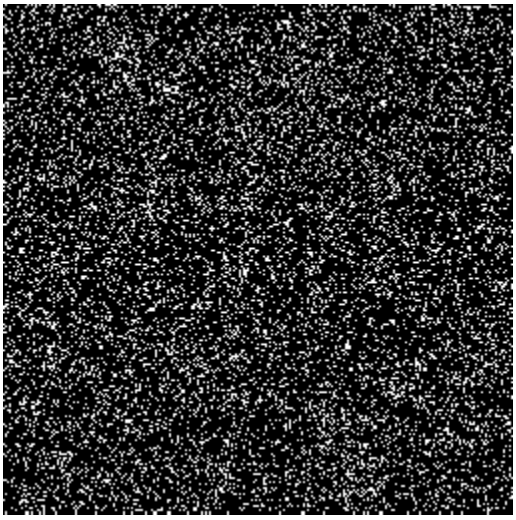

Figura 49 Diferencia entre la imagen original y la imagen con ruido

Figura 50 Imagen con ruido aditivo, sin control de errores, PSNR = 20.09 dB

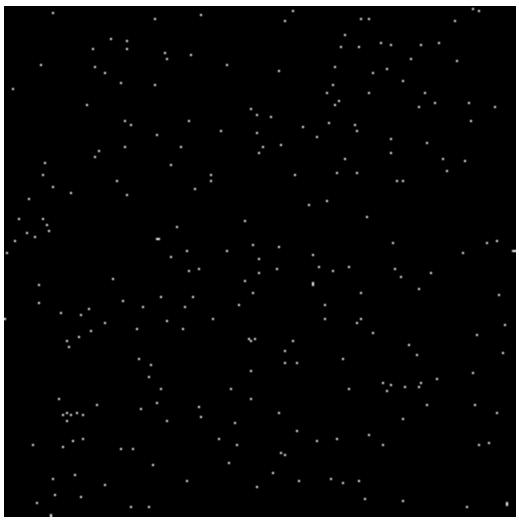

Figura 51 Diferencia entre la imagen original y la imagen decodificada

Figura 52 Imagen decodificada, con el código BCH, PSNR = 34.00 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.036

BER con codificación BCH = 0.00079

PSNR calculada con Matlab

PSNR sin codificación = 21.16 dB

PSNR con codificación BCH = 36.48 dB

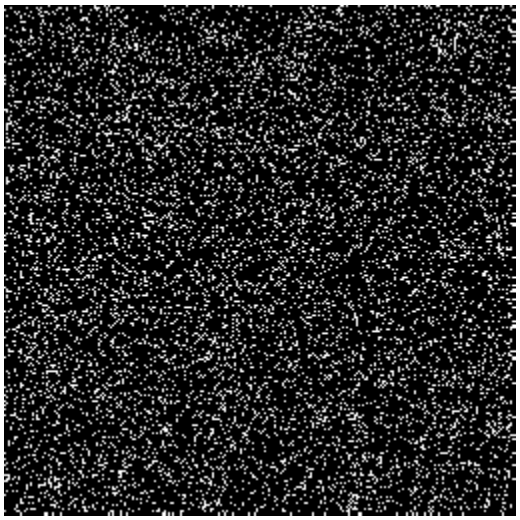

Figura 53 Diferencia entre la imagen original y la imagen con ruido

Figura 54 Imagen con ruido aditivo, sin control de errores, PSNR = 21.16 dB

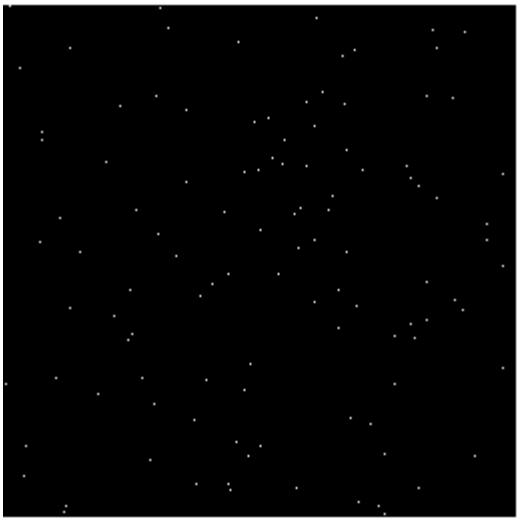

Figura 55 Diferencia entre la imagen original y la imagen decodificada

Figura 56 Imagen decodificada, con el código BCH, PSNR = 36.48 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.031

BER con codificación BCH = 0.00039

PSNR calculada con Matlab

PSNR sin codificación = 21.77 dB

PSNR con codificación BCH = 40.34 dB

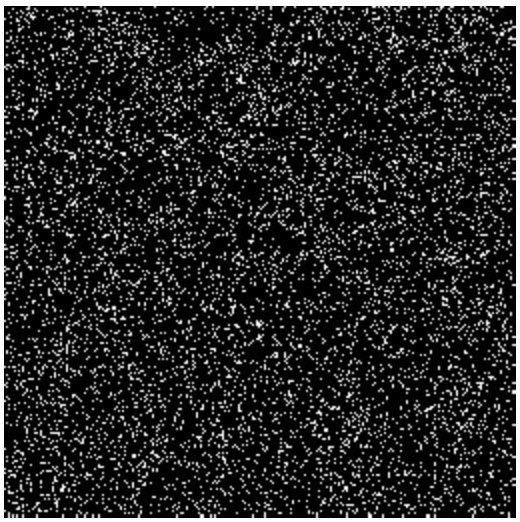

Figura 57 Diferencia entre la imagen original y la imagen con ruido

Figura 58 Imagen con ruido aditivo, sin control de errores, PSNR = 21.77 dB

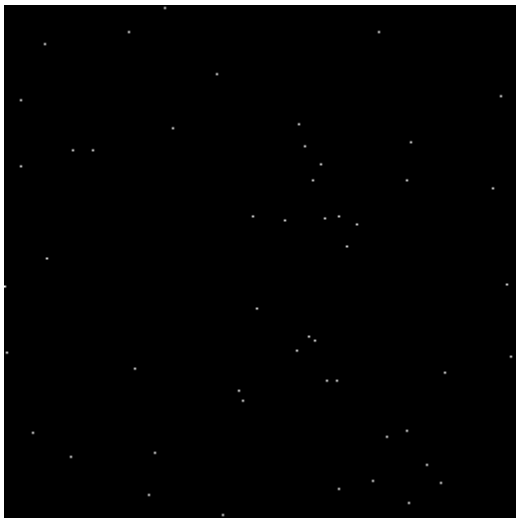

Figura 59 Diferencia entre la imagen original y la imagen decodificada

Figura 60 Imagen decodificada, con el código BCH, PSNR = 40.34 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.018

BER con codificación BCH = 0.000071

PSNR calculada con Matlab

PSNR sin codificación = 23.67 dB

PSNR con codificación BCH = 50.29 dB

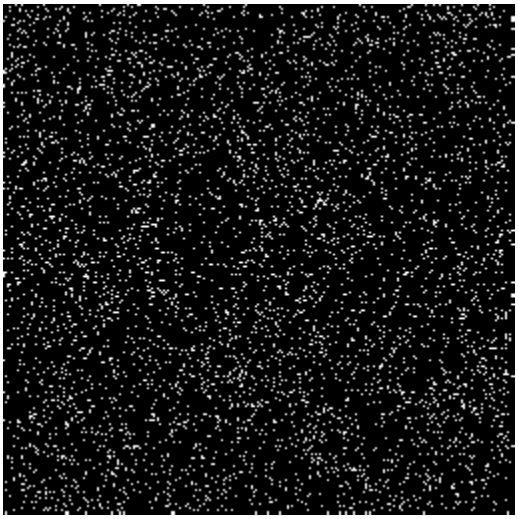

Figura 61 Diferencia entre la imagen original y la imagen con ruido

Figura 62 Imagen con ruido aditivo, sin control de errores, PSNR = 23.67 dB

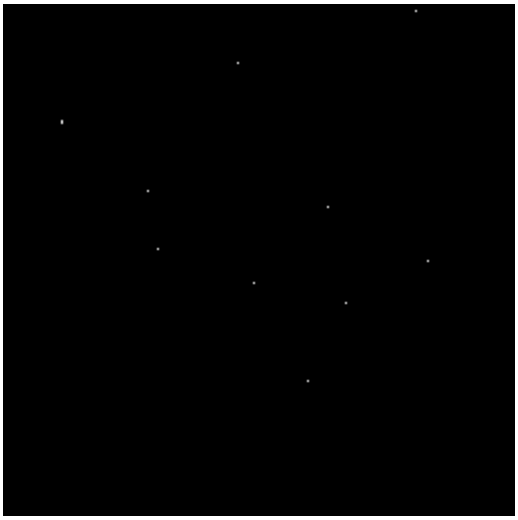

Figura 63 Diferencia entre la imagen original y la imagen decodificada

Figura 64 Imagen decodificada, con el código BCH, PSNR = 50.29 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.0092

BER con codificación BCH = 0.000000197

PSNR calculada con Matlab

PSNR sin codificación = 26.72 dB

PSNR con codificación BCH = 90.28 dB

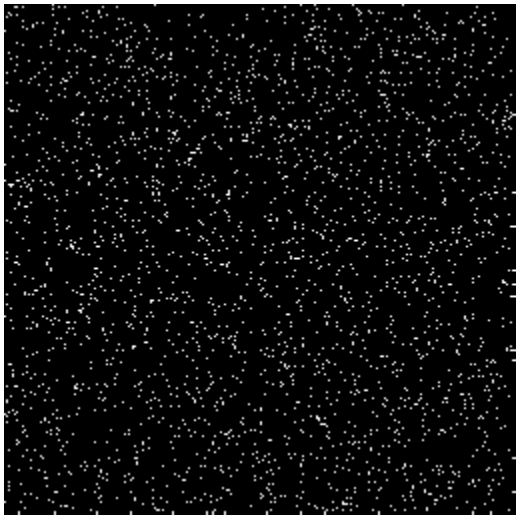

Figura 65 Diferencia entre la imagen original y la imagen con ruido

Figura 66 Imagen con ruido aditivo, sin control de errores, PSNR = 26.72 dB

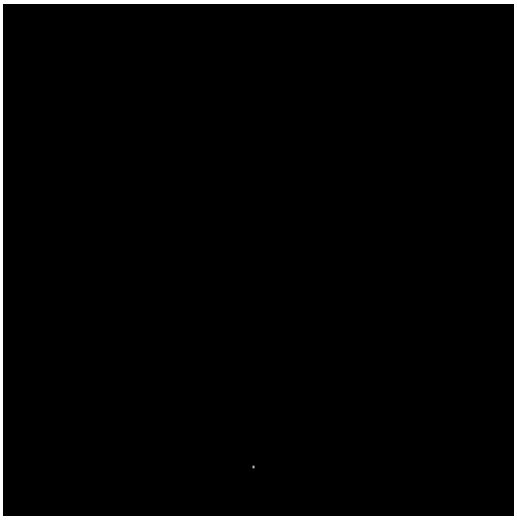

Figura 67 Diferencia entre la imagen original y la imagen decodificada

Figura 68 Imagen decodificada, con el código BCH, PSNR = 90.28 dB

Datos obtenidos del bloque Error Rate de GNU Radio

BER sin codificación = 0.0047

BER con codificación BCH = 0.000000000

PSNR calculada con Matlab

PSNR sin codificación = 29.40 dB

PSNR con codificación BCH = inf

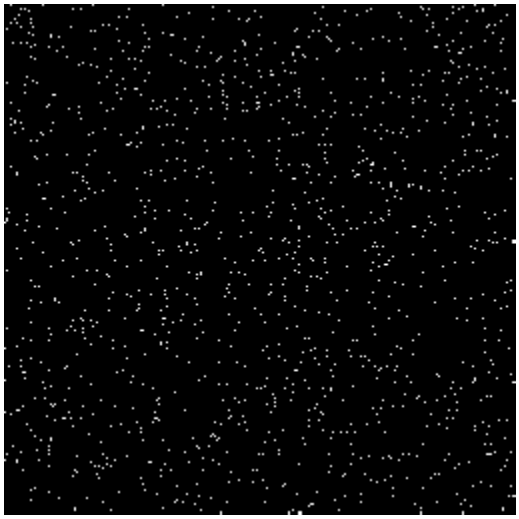

Figura 69 Diferencia entre la imagen original y la imagen con ruido

Figura 70 Imagen con ruido aditivo, sin control de errores, PSNR = 29.40 dB

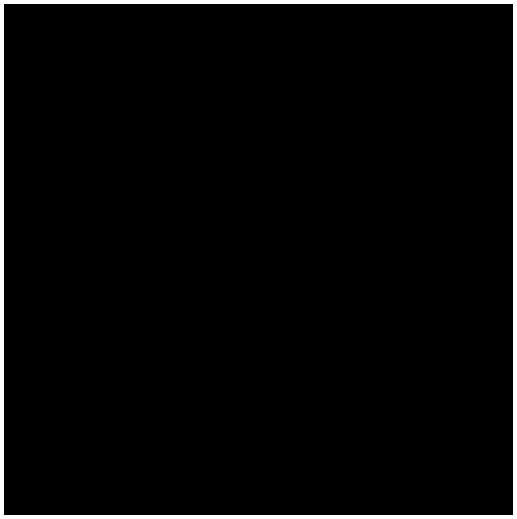

Figura 71 Diferencia entre la imagen original y la imagen decodificada

Figura 72 Imagen decodificada, con el código BCH, PSNR = inf

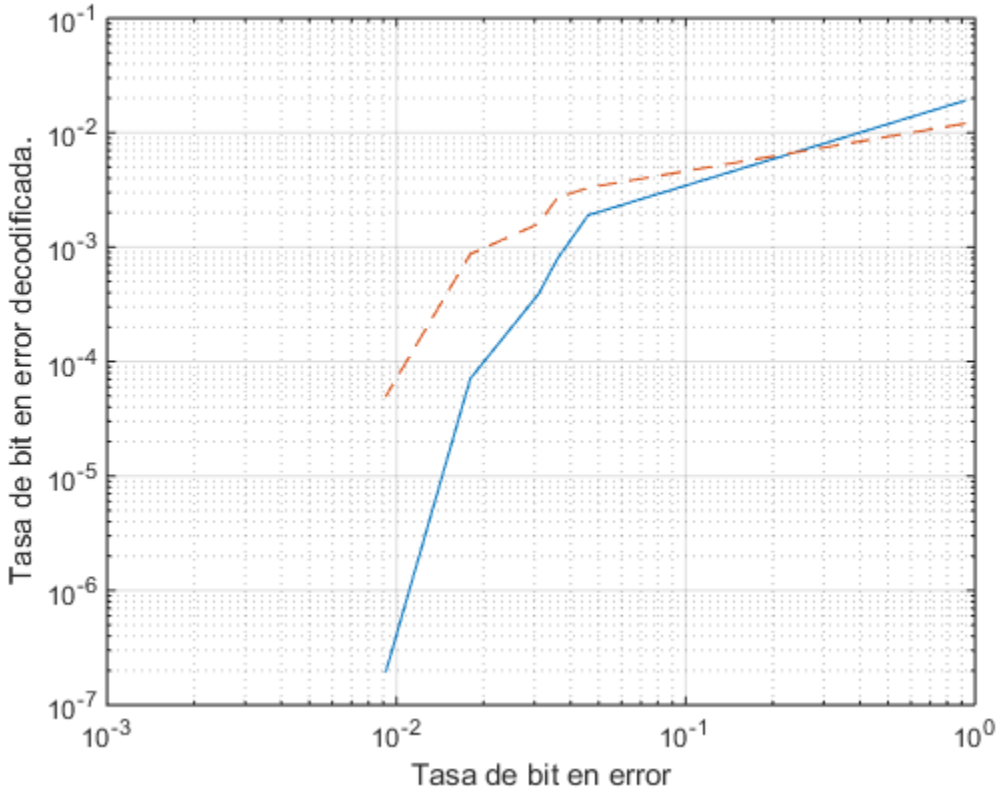

Figura 73 Grafica de desempeño del código de Reed-Solomon (línea punteada), y del código BCH (línea continua).

Reed-Solomon				BCH			
1	BER sin codificación			3	PSNR sin codificación en dB		
2	BER con codificación			4	PSNR con codificación en dB		
1	2	3	4	1	2	3	4
0.089	0.012	16.52	27.33	0.092	0.019	17.2	23.3
0.045	0.033	19.42	32.52	0.046	0.0019	20.09	34.00
0.035	0.0027	20.54	33.54	0.036	0.00079	21.16	36.48
0.030	0.0016	21.08	36.14	0.031	0.0039	21.77	40.34
0.018	0.00087	23.66	39.10	0.018	0.000071	23.67	50.29
0.0092	0.00005	26.34	49.96	0.0092	0.000000197	26.72	90.28
0.0041	0.0	29.00	50.44	0.0047	0.0	29.40	inf

Tabla 2 Resultados obtenidos de la PSNR y BER para los códigos de Reed-Solomon y BCH

Debido a la forma en que internamente GNU Radio manipula los vectores, la imagen procesada con el código de Reed-Solomon no puede ser reconstruida por completo, esto es porque el codificador de Reed-Solomon implementado en GNU Radio, trabaja con vectores de enteros de cuatro elementos, y en los casos en que GNU Radio no puede separar exactamente el vector a codificar en vectores de cuatro elementos, GNU Radio trunca los vectores. Esto hace que la imagen decodificada no sea igual a la imagen original por el truncamiento que realiza GNU Radio, efecto que se puede apreciar en la Figura 74, donde se observa como una parte del patrón de error que ha sido amplificado aparece en blanco, este patrón de error debería ser completamente negro (ya que si los pixeles de la imágen original y decodificada son iguales, en el patrón de error se pone un pixel negro, y si son diferentes, se pone un pixel blanco), y de acuerdo a los resultados mostrados en la Tabla 2, para una $BER = 0.0041$ en el código de Reed-Solomon la BER de la imagen decodificada es igual a cero, sin embargo la PSNR de la imagen decodificada no tiene un valor infinito. Esto nos indica que la imagen decodificada no es igual a la imagen original por el efecto del truncamiento.

Figura 74 Ampliación del Error no corregible en la decodificación con el código de Reed-Solomon

Finalmente en la actualidad los códigos BCH y de Reed-Solomon son muy importantes en las comunicaciones digitales, incluyendo las redes inalámbricas, muestra de ello es que son ampliamente utilizados para la codificación de canal en los estándares internacionales para la televisión digital terrestre. En la Tabla 3 se muestran las diferentes combinaciones de los códigos de bloque concatenados con códigos convolucionales, para la codificación de canal de los diferentes estándares internacionales para la televisión digital terrestre.

System	ATSC	DTMB	DVB-T	ISDB-T
Outer code	RS (207, 187, $t = 10$)	BCH(762, 752)	RS (204, 188, $t = 8$)	RS (204, 188, $t = 8$)
Interleaving of outer code	52 RS encoded blocks	None	12 Encoded RS blocks	12 Encoded RS blocks
Inner code	2/3 TCM	LDPC code rate: 0.4, 0.6, 0.8, code length 7493	Multirate punctured convolutional code rate: 1/2, 2/3, 3/4, 5/6, 7/8, constraint length 7	Multirate punctured convolutional code rate: 1/2, 2/3, 3/4, 5/6, 7/8, constraint length 7
Interleaving of inner code	12:1 Trellis code interleaving	Convolutional time interleaving and frequency interleaving	Convolutional interleaving and frequency interleaving	Convolutional interleaving, frequency interleaving, and optional time interleaving

Tabla 3 Estándares internacionales para la codificación de canal en la televisión digital terrestre [16]

6. CONCLUSIONES Y TRABAJO FUTURO

6.1 Conclusiones

En este trabajo de tesis el principal objetivo consistió en implementar en la plataforma SDR los códigos de bloque de Reed-Solomon y BCH. Estos códigos en la actualidad son muy importantes, ya que se emplean ampliamente en la codificación de canal para la televisión digital terrestre en conjunto con los códigos convolucionales. De ahí el interés para realizar el trabajo de tesis en esta área. Permitiendo mostrar también el uso práctico de la plataforma de radio definido por software con un sistema de control de errores.

En el documento se presentan las etapas más importantes de la codificación y decodificación de los códigos BCH y de Reed-Solomon, tales como la construcción del polinomio generador, que en el caso de la codificación, nos permite generar el circuito codificador LFSR, por medio del cual podemos generar la palabra de código a ser transmitida; en el caso de la decodificación también se muestra de manera detallada los pasos necesarios para la decodificación de la palabra recibida. Para la decodificación de los códigos tanto BCH, como de Reed-Solomon se hace uso del algoritmo de solución directa PGZ, este algoritmo no es el más eficiente, pero es sencillo en su implementación. Ahora bien, tomando en cuenta que esta solución directa tiene un costo computacional elevado, solo se empleó para la decodificación de pequeños códigos BCH y de Reed-Solomon, en donde el impacto computacional no es muy grande. Permitiéndonos así estudiar y dominar el uso de la plataforma de radios definidos por *software* con códigos correctores, que solo está parcialmente documentada en la literatura internacional.

En este trabajo se probó exitosamente la capacidad de corrección de errores de los códigos de bloque de Reed-Solomon y BCH, en donde para simular un sistema de comunicaciones completo se implementó un bloque simulador del canal binario simétrico para probar el código binario BCH, y un bloque que simula el error de ráfaga para probar el código no binario de Reed Solomon, teniendo de esta manera un sistema completo transmisor-receptor, para así probar el funcionamiento de los códigos dentro del ambiente gráfico GNU Radio y someterlos a diferentes probabilidades de error.

Como se observa en los resultados obtenidos en este documento, el código BCH muestra ligeramente un mejor desempeño en la corrección de errores que el código de Reed-Solomon, aunque con un costo computacional mayor, ya que en las pruebas realizadas de codificación y decodificación del código BCH, este tiene mayor latencia, así como también requiere de un mayor ancho de banda debido a que requiere de una mayor cantidad de redundancia, en comparación con el código de Reed-Solomon; independientemente del costo computacional de uno y otro código, ambos códigos demostraron un buen desempeño en la corrección de errores, siendo la principal virtud del código BCH la corrección de errores aleatorios de un bit. Como se puede ver en las imágenes procesadas

en este documento el código BCH corrige de manera muy satisfactoria los errores generados por el canal binario simétrico; al igual que el código de Reed-Solomon combate de manera muy efectiva los errores de ráfaga (errores de paquete), por lo que el código de Reed-Solomon resulta especialmente útil cuando la información es modulada con una modulación M-aria. En este caso la información se transmite en símbolos en vez de bits, y estos símbolos pueden ser fácilmente adaptados a los símbolos de los códigos de Reed-Solomon.

Una de aportaciones del presente trabajo de tesis es, la implementación de bloques de procesado de señal de los códigos correctores de errores de Reed-Solomon y BCH, no disponibles dentro de la plataforma de GNU Radio. Esto representó un gran reto debido a la muy escasa y poco clara información disponible, y que coadyuvará en el desarrollo de futuras implementaciones en la plataforma de GNU Radio.

6.2 Trabajo futuro.

En el presente trabajo, se realizó la decodificación de los códigos de Reed Solomon y BCH empleando el algoritmo de solución directa PZG, este algoritmo es poco eficiente, y solo tiene sentido emplearlo para pequeños códigos BCH y de Reed Solomon. Así que para poder realizar códigos de Reed Solomon y BCH con mayor capacidad de corrección de errores es necesario emplear algoritmos de decodificación más eficientes, tal es el caso del algoritmo de Berlekamp para la obtención del polinomio localizador de errores, y adicionalmente para el código de Reed Solomon es necesario emplear el algoritmo de Massey para obtener los valores de error. Estos algoritmos son computacionalmente más eficientes que el algoritmo de solución directa PGZ, y permiten realizar códigos de Reed Solomon y BCH con mayor capacidad de corrección de errores con un menor costo computacional.

APÉNDICES

A. Acrónimos

ADC	Analogic Digital Converter
A/D	Analógico-Digital
BCH	Bose-Chauduri-Hocquenghem
D/A	Digital-Analógico
DAC	Digital Analogic Converter
DDC	Digital Down Converter
DUC	Down Up Converter
FFT	Fast Fourier Transform
FPGA	Field Programable Gate Array
GF	Galois Field
IF	Intermediate Frecuency
LDPC	Low Density Parity Check
RF	Radiofrequency
SDR	Sofware Defined Radio
SR	Software Radio
UHD	USRP Hardware Driver
USRP	Universal Software Radio Peripheral

B. Instalación del software GNU Radio en Ubuntu

1. Descargar el script $build - gnuradio$ en el siguiente enlace. http://www.sbrac.org/files/build-gnuradio
   El script $build - gnuradio$ se encarga de instalar los pre-requisitos y dependencias necesarias tanto para el UHD como para GNU Radio. El software USRP Hardware Driver ™ (UHD ™) es el controlador de hardware para todos los dispositivos USRP. Funciona en todas las plataformas (Linux, Windows y Mac), y se puede construir con los compiladores GCC, Clang y MSVC. El objetivo del software UHD es proporcionar un controlador de host y API para los productos Ettus de Research [10].
2. Crear en la carpeta de Home, una nueva carpeta con el nombre de gnuradio.
3. Guardar el script $build - gnuradio$ en la carpeta gnuradio.
4. Abrir un terminal y dirigirnos a la carpeta de gnuradio que se creó.
5. Dar permisos de ejecución al script con la instrucción.
   $\$\ chmod\ a + x\ build - gnuradio$
6. Ejecutar el script como
   $\$\ ./build - gnuradio$
7. Una vez que el script termina de ejecutarse, debemos incluir la ruta de la librería de Python mediante el bash como sigue.

$\$ sudo\ gedit\ .bashrc$

8. Una vez abierto el archivo bash en el editor de texto gedit, en el inicio del archivo agregamos lo siguiente.

   $export\ PYTHONPATH = /usr/local/lib/python2.7/dist - packages$

9. Guardamos los cambios, y cerramos el archivo.
10. Actualizamos la configuración

    $\$ sudo\ ldconfig$

11. Terminado el proceso de instalación, cerramos y reiniciamos la computadora; por último para verificar si la instalación de GNU Radio es correcta ejecutamos el siguiente comando en la terminal.

    $\$ gnuradio - companion$

C. Instalación del USRP N210

Para instalar el USRP N210 en Ubuntu es necesario en primer lugar instalar GNU Radio, este proceso ya ha sido descrito en el apéndice A; ya que se ha instalado correctamente GNU Radio se puede proceder a instalar el USRP N 210, para esto es necesario, uno configurar el puerto Ethernet y dos quemar las imágenes en el FPGA del USRP N210.

1. Configuración del puerto Ethernet.

En Fedora y Ubuntu ambos utilizan Network Manager para gestionar las conexiones de red. Desafortunadamente, Network Manager a menudo trata de tomar el control de una conexión y desconecta la interfaz en la que está conectada el radio. Para evitar que esto suceda se deberá abrir la configuración de Network Manager y decirle que ignore la interfaz de red que está utilizando.
Esto no es lo mismo que simplemente establecer una dirección IP estática. Se debe informar a Network Manager para que ignore la interfaz. Para esto hay que editar el archivo de interfaces, al editar este archivo haremos que Network Manager deje de manejar el puerto Ethernet (eth0) y desconecte el radio, para hacer esto hacemos lo siguiente:

Abrimos una terminal y escribimos lo siguiente

$\$ sudo\ gedit\ /etc/network/interfaces$

A continuación nos pedirá la contraseña de administrador, después de dar la contraseña aparecerá un archivo con lo siguiente.

$\# interfaces(5)\ file\ used\ by\ ifup(8)\ and\ ifdown(8)$
$auto\ lo$
$iface\ lo\ inet\ loopback$

Esto que aparece en el archivo se deja tal como está, después de la última línea se agrega la configuración estática para el puerto $eth0$ como sigue:

$\# interfaces(5)\ file\ used\ by\ ifup(8)\ and\ ifdown(8)$
$auto\ lo$
$iface\ lo\ inet\ loopback$

$auto\ eth0$
$iface\ eth0\ inet\ static$
$address\ 192.168.10.1$
$netmask\ 255.255.255.0$

La dirección 192.168.10.1 es una dirección IP estática necesaria para poder habilitar la comunicación del radio con la PC. Después de hacer los cambios anteriores en el archivo de interfaces lo guardamos y ejecutamos:

$\$\ sudo\ /etc/init.d/networking\ restart$

Para reinicializar los servicios de red, reiniciamos la computadora y verificamos que Network Manager ya no gestiona el puerto Ethernet con el siguiente comando:

$\$\ nmcli\ dev\ status$

Debe aparecer lo siguiente:

$DISPOSITIVO\ TIPO \qquad ESTADO$
$wlan0 \quad 802-11-wireless \quad conectado$
$eth0 \quad 802-3-ethernet \quad sin\ gestión$

Como se puede ver Network Manager ya no está gestionando al puerto Ethernet ($eth0$). Después de verificar que Network Manager ya no gestiona al puerto Ethernet hay que verificar que la dirección IP estática del puerto Ethernet sea 192.168.10.1 (ya que esta dirección en necesaria para poder establecer la comunicación entre el radio y el ordenador) con el siguiente comando:

$\$\ ifconfig$

Al ejecutar el anterior comando se obtiene:

eth0 Link encap:Ethernet direcciónHW 70:5a:b6:e3:05:18
Direc. inet:192.168.10.1 Difus.:192.168.10.255 Másc:255.255.255.0
ACTIVO DIFUSIÓN MULTICAST MTU:1500 Métrica:1
Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
colisiones:0 long.colaTX:1000
Bytes RX:0 (0.0 B) TX bytes:0 (0.0 B)
Interrupción:16

Aquí se puede observar que la dirección estática de $eth0$ es: inet:192.168.10.1

2. Quemar las imágenes

Después de que se ha instalado el GNU Radio con el script, y que el puerto Ethernet ($eth0$) haya sido configurado se procede a quemar las imágenes que se encuentran en la siguiente ruta:

$/usr/local/share/uhd/images/$

Para quemar las imágenes hay que dirigirnos a la siguiente ruta con el comando Change Directory o "cd":

$ $cd\ /usr/local/share/uhd/images/$

Una vez que estamos en el directorio donde están las imágenes y con el puerto (eth0) configurado correctamente con la dirección estática 192.168.10.1 procedemos a quemar las imágenes en el $fpga$, que en nuestro caso son para el USRP N210 r4 con el siguiente comando:

$ $usrp\_n2xx\_simple\_net\_burner\ --fw\ usrp\_n210\_fw.bi --fpga\ usrp\_n210\_r4\_fpga.bin$

Por ultimo para verificar que el ordenador esta comunicado con el USRP N210 utilizamos el siguiente comando.

$ $uhd\_find\_devices$

El resultado debe ser algo como:

```
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
   type: usrp2
   addr: 192.168.10.2
   name:
   serial: E5R19S2UP
```

Si en todo lo anterior no se presentan errores el dispositivo está listo para usarse.

D. Creación de bloques de procesado de señal en GNU Radio

La creación de bloques personalizados de procesado de señal en GNU Radio se realiza mediante la herramienta $gr\_modtool$, esta herramienta permite la creación de bloques de procesado de señal tanto en el lenguaje de programación C++ como en Python, los bloques de procesado de señal pueden ser de 4 tipos.

- Síncrono $(1:1)$ En este bloque el número de elementos de entrada es igual al número de elementos de salida.
- Decimador $(N:1)$ En este bloque el número de elementos de entrada es un múltiplo fijo del número de elementos de salida.
- Interpolación $(1:M)$ En este bloque el número de elementos de salida es un múltiplo fijo del número de elementos de entrada.
- General/Básico $(N:M)$ Este bloque no provee una relación entre el número de elementos de entrada y el número de elementos de salida.

La herramienta $gr\_modtool$ dispone de los siguientes comandos.

- disable Deshabilita un bloque.
- info Devuelve información sobre un módulo dado.
- remove Elimina un bloque (Borra archivos y elimina entradas Makefile).
- makexml Crea un archivo XML para los enlaces de bloque en GNU Radio Companion (GRC).
- add Añade un bloque al módulo fuera del árbol.
- newmod Crea un nuevo módulo fuera del árbol.
- rename Realiza la misma función que add.

Vemos que hay varios comandos disponibles, pero para añadir un nuevo bloque solo son necesarios las opciones de newmod y add. Para añadir un nuevo módulo ejecutamos la siguiente instrucción.

$ $gr\_modtool\ newmod\ "Nombre\ del\ modulo"$

Por ejemplo si quisiéramos crear un módulo que contenga varios moduladores podemos nómbralo como sigue.

$ $gr\_modtool\ newmod\ Moduladores$

Una vez que se creado un nuevo módulo es necesario agregar bloques de procesado de señal al nuevo módulo, esto se hace con el comando add; para crear un nuevo bloque de procesado de señal con add, si tienen que especificar dos parámetros, el tipo de bloque de procesado de señal y el lenguaje de programación con el que se escribirá el bloque, como ya se mencionó arriba hay cuatro tipos de bloque, sync (síncrono), decimador, interpolador y general; y dos lenguajes con los que podemos escribir el bloque, que son Python y C++. A continuación se muestran los pasos para cómo crear un bloque síncrono con el lenguaje de programación Python y C++.

Ya que hemos creado el modulo filtros, para agregar bloques a este módulo es necesario situarnos en la carpeta de este módulo, esto lo hacemos con el comando cd en Ubuntu como sigue.

$ $cd\ gr - Moduladores$

Nota: cuando newmod crea un módulo le antepone las letras gr al nombre de la carpeta del módulo

Para agregar un bloque síncrono escrito en C++ al módulo ejemplo de Moduladores, escribimos el siguiente comando.

$ $gr\_modtool\ add - t\ sync - l\ cpp$

Y para agregar un bloque síncrono escrito en Python escribimos el siguiente comando.

$ $gr\_modtool\ add - t\ sync - l\ python$

Después de ejecutar los comandos anteriores $gr\_modtool$ nos solicitara el nombre del bloque, y una lista de parámetros válidos. Por otra parte, GNU Radio ofrece una opción de escribir casos de prueba. Esto proporciona la garantía de calidad para el código escrito.

Una vez que se han añadido los bloques de procesado de señal escritos en C++ o Python, y las descripciones XML de los bloques en la carpeta grc, necesarias para que los bloques estén disponibles en Radio GNU companion, que es la interfaz de usuario gráfica para GNU Radio, se procede a hacer la instalación de estos mediante los siguientes pasos.

1. Se crea la carpeta build dentro de la carpeta del módulo que se creó, que para nuestro ejemplo es la carpeta gr-Moduladores.
2. Nos posicionamos dentro de la carpeta build y ejecutamos los siguientes comandos para concluir con la instalación del módulo.

cmake ../
make
sudo makes install
sudo ldconfig

CÓDIGOS

A continuación se muestran los códigos para la simulación del canal binario simétrico, simulación del error de ráfaga y un script de Matlab que calcula la relación señal ruido pico (PSNR) de las imágenes procesadas en GNU Radio.

Código para simular el canal binario simétrico en GNU Radio

```
import numpy
from gnuradio import gr
from scipy import random

class BSCI(gr.sync_block):
    """
    docstring for block BSCI
    """
    def __init__(self, Pe):
        self.Pe = Pe
        gr.sync_block.__init__(self,
            name="BSCI",
            in_sig=[numpy.int32],
            out_sig=[numpy.int32])


    def work(self, input_items, output_items):
        in0 = input_items[0]
        out = output_items[0]
        # <+signal processing here+>
        for i in range(len(in0)):
            if i <= 54:
                out[:] = in0
            else:
                v=bin(in0[i])[2:]
                e=0
                for j in range(len(v)):
                    a = random.randint(2147483647) % self.Pe
                    if a==0:
                        PC=in0[i]
                        s=1<<j
                        #print '\ns = ',s
                        e = s^PC
                        in0[i]=e
        out[:] = in0
        return len(output_items[0])
```

Código empleado para simular el error de ráfaga en el canal empleado para probar el código de Reed-Solomon.

```
import numpy
from gnuradio import gr
from scipy import random

class qasc3(gr.sync_block):
    """
    docstring for block qasc3
    """
    def __init__(self, Pe,Patron,limite):
        self.Pe = Pe
        self.Patron=Patron
        self.limite=limite
        gr.sync_block.__init__(self,
            name="qasc3",
            in_sig=[numpy.int32],
            out_sig=[numpy.int32])

    def work(self, input_items, output_items):
        in0 = input_items[0]
        out = output_items[0]
        # <+signal processing here+>
        for i in range(len(in0)):
            if i <= 54:
                out[:] = in0
            else:
                c=self.limite+1
                e=0
                a = random.randint(2147483647) % self.Pe
                if a==0:
                    PC=in0[i]
                    sh=random.randint(c)
                    s=self.Patron<<sh
                    e = s^PC
                    in0[i]=e
        out[:] = in0
        return len(output_items[0])
```

Código para calcular la relación señal a ruido pico (PSNR) con Matlab

```
close all
clc

original=imread('lenna256.bmp');
final=imread('P20_8bc.bmp');

Tam=size(final);
Filas=Tam(1);
Columnas=Tam(2);

% Generacion de la matriz patron de error
e=zeros(Tam);
for i=1:Filas
    for j=1:Columnas
        if final(i,j)-original(i,j) ~= 0
            e(i,j)=256;
        elseif original(i,j)-final(i,j) ~= 0
            e(i,j)=256;
        else
            e(i,j)=0;
        end
    end
end

if (original == final)
    disp('Imagenes son identicas: PSNR tiene un valor infinito')
    PSNR = Inf;
    disp('PSNR: ')
    disp(PSNR)
    return;
else
    X = double(original);
    Y = double(final);
    m = sum((X(:)-Y(:)).^2) / numel(X);
    PSNR = 10*log10(255*255/m);
    disp('PSNR: ')
    disp(PSNR)
end

%Imagen original
figure(1)
imshow(original)
title('Imagen original')
%Imagen decodificada
figure(2)
imshow(final)
title('Imagen Final')
%Patron de los errores
figure(3)
imshow(e)
title('Error')
```

Finalmente los códigos realizados para la implementación del código BCH y el código de Reed-Solomon y su implementación final en GNU Radio, se anexan en [34]

En esta página se encuentran disponibles los códigos $BCH(15,5)$ y el código de Reed-Solomon $RS(15,9)$, en estos códigos se puede verificar el funcionamiento detallado de estos; adicionalmente se encuentran disponibles los archivos gnuradio BCH y gnuradio RS, en estos archivos se encuentra disponible toda la implementación necesaria para el correcto funcionamiento de los códigos $BCH(15,5)$ y $RS(15,9)$ dentro la herramienta grafica GNU Radio.

REFERENCIAS

[1] Sklar, B., *Digital Communications: Fundamentals and Applications*. Prentice Hall, second edition, 2001.
[2] Reed, I.S., and Chen, X., *Error-control coding for data networks*, Kluwer Academic Publishers, 1999.
[3] Pinar, I., Murillo, J. J., *Laboratorio de comunicaciones digitales Radio Definida por Software*, Universidad de Sevilla, 1er edición, 2011.
[4] Mitola, J., "Software Radios: Survey, Critical Evaluation and Future Directions", IEEE National Telesystems Conference, 1992.
[5] http://www.ettus.com/blog/2011/08/mathworks-now-offers-uhd-support-for-matlab-and-simulink
[6] http://www.ettus.com/site/about
[7] http://gnuradio.org/redmine/projects/gnuradio/wiki
[8] Lidl, R., Nirderreiter, H., *FINITE FIELDS*, Cambridge University Press, volume 20, part 1.
[9] http://www.ettus.com/content/files/07495_Ettus_N200-210_DS_Flyer_HR_1.pdf
[10] http://code.ettus.com/redmine/ettus/projects/uhd/wiki
[11] http://gnuradio.org/redmine/projects/gnuradio/wiki/TutorialsWritePythonApplications#GNU-Radio-Modules
[12] https://gnuradio.org/doc/doxygen/page_blocks.html
[13] http://files.ettus.com/manual/page_sync.html
[14] Wicker, S.B., *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1995.
[15] Morelos-Zaragoza, R. H., *The Art of Error Correcting Coding*, Wiley, Second edition.
[16] Song, J., Yang, Z., and Wang, J., *Digital Terrestrial Television Broadcasting: Technology and System*, John Wiley & Sons, 01/07/2015
[17] https://en.wikipedia.org/wiki/Software-defined_radio#cite_ref-1
[18] Reed, I.S., and Solomon, G., "Polynomial codes over certain finite fields", M.I.T. Lincoln Laboratory Group Report 47.23, 31 December 1958.
[19] Reed, I.S., and Solomon, G., "Polynomial codes over certain finite fields", SIAM Journal of Applied Mathematics, Vol. 8, pp. 300-304, 1960.
[20] http://www.digitalcodesign.com/es/reedsolomon
[21] Peterson, W. W., and Weldon, E.J., *Error-Correcting Codes*, MIT Press, 1972.
[22] Berlekamp, E.R., *Algebraic Coding Theory*, McGraw Hill: New York, 1968
[23] Reed, I.S., and Solomon, G., "A decoding procedure for polynomials codes", M.I.T. Lincoln Laboratory Group Report 47.24, 6 March 1959.
[24] Peterson, W. W., "Encoding and error-correction procedures for Bose-Chaudhuri Codes", IRE Trans. on Inform. Theory, Vol. IT-6, pp.459-470, Sept. 1960.
[25] Gorenstein, D., and Zierler, N., " A class of error correcting codes in p^m symbols", Journal of the Society of Industrial and Applied Mathematics, Vol. 9, pp.207-214, June 1961.
[26] Chien, R. T., "Cyclic decoding procedure for Bose-Chaudhuri-Hocquenghem codes", IEEE Trans. on Inform. Theory, Vol. IT-10, pp.357-363, Oct. 1964.

[27] Forney, G. D., “On decoding BCH codes”, IEEE Trans. on Inform. Theory, Vol. IT-11, pp.549-557, Oct. 1965.
[28] Berlekamp, E. R., “Nonbinary BCH decoding”, International Symposium on Information Theory, San Remo, Italy, 1967.
[29] Massey, J. L., “Shift Register synthesis and BCH decoding”, IEEE Trans. on Inform. Theory, Vol. IT-15, No. 1, pp.122-127, Jan. 1969.
[30] Sugiyama, Y., Kasahara, M., Hirasawa, S., and Namekawa, T., “A method for solving key equation for decoding Goppa codes”, Inf. and Contr., 1975, 27, pp.87-99.
[31] Shannon, C. E., “A mathematical theory of communication”, Reprinted with corrections from The Bell System Technical Journal, Vol. 27, pp. 379–423,623-656, July, October 1948.
[32] Berrow, C., Glavieux, A., and Thitimajshima, P., “Near Shannon limit error-correcting coding and decoding: Turbo Codes”, Proceedings of the International Conference on Communications. Geneva, Switzerland, 1993.
[33] Gallager, R. G., "Low Density Parity Check Codes", Transactions of the IRE Professional Group on Information Theory, Vol. IT-8, January 1962, pp.21-28.
[34] Saldaña, R., “Programas implementados para los códigos BCH y de Reed-Solomon”, 16 Noviembre 2015. En linea. Disponible: http://raymundosaldana.wix.com/codigos