Implementación de Transmisión y Recepción Digital con GNU Radio

Lema, Adan J.A. - Paz, Matias J.

En el contexto actual de la comunicación inalámbrica y la tecnología digital, el diseño y desarrollo de sistemas para la transmisión y recepción de señales de audio han adquirido una relevancia crucial. El continuo progreso en las telecomunicaciones ha impulsado la creación de tecnologías cada vez más sofisticadas, posibilitando una transmisión eficiente y de alta fidelidad de datos de audio a través de una amplia gama de canales disponibles.

Este informe se centra en el proceso completo de concepción y operatividad de un transmisor y receptor de audio, concebidos y realizados mediante el empleo de GNU Radio. Su propósito fundamental es ofrecer una exposición detallada y exhaustiva que abarque desde el diseño inicial hasta las pruebas realizadas en un sistema de transmisión y recepción de señales de audio. Este proyecto se enmarca en la aplicación práctica de los conocimientos adquiridos durante el curso de Sistemas de Comunicaciones Digitales I.

Se aspira a que este informe no solo sea un documento informativo, sino también una referencia fundamental para próximos proyectos y desarrollos en el campo de la transmisión y recepción de señales de audio. La minuciosa exploración de cada etapa del proceso de diseño tiene como objetivo proporcionar una comprensión clara y completa, permitiendo a otros estudiantes reproducir y optimizar estos sistemas con mayor facilidad y eficacia.

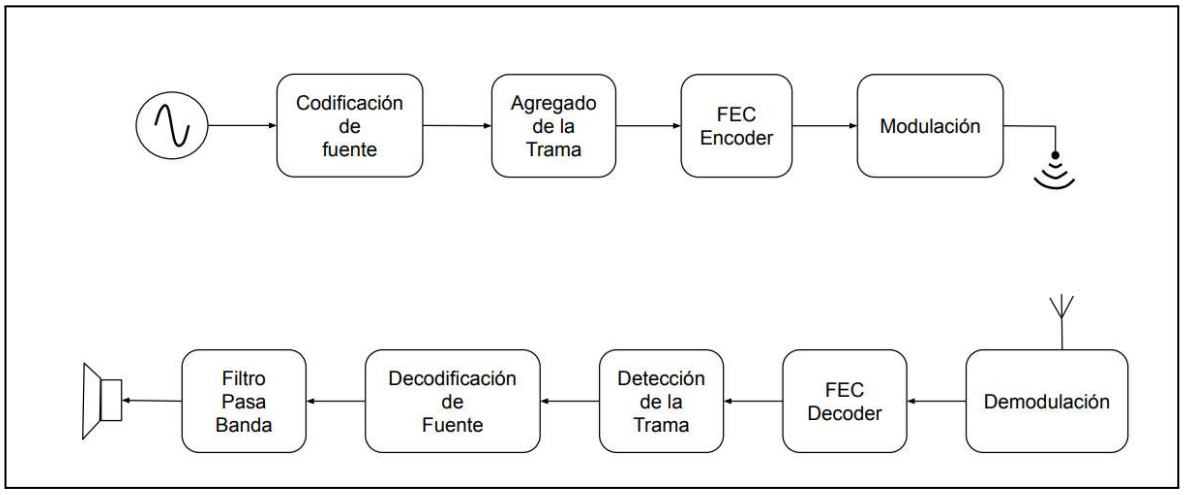

Figura 1: Diagrama de Bloques del Transmisor-Receptor.

Transmisor

Codificador de Fuente

Partimos de un tono compuesto por una señal sinusoidal de frecuencia 1 [kHz] y otra de 2 [kHz] para transmitir.

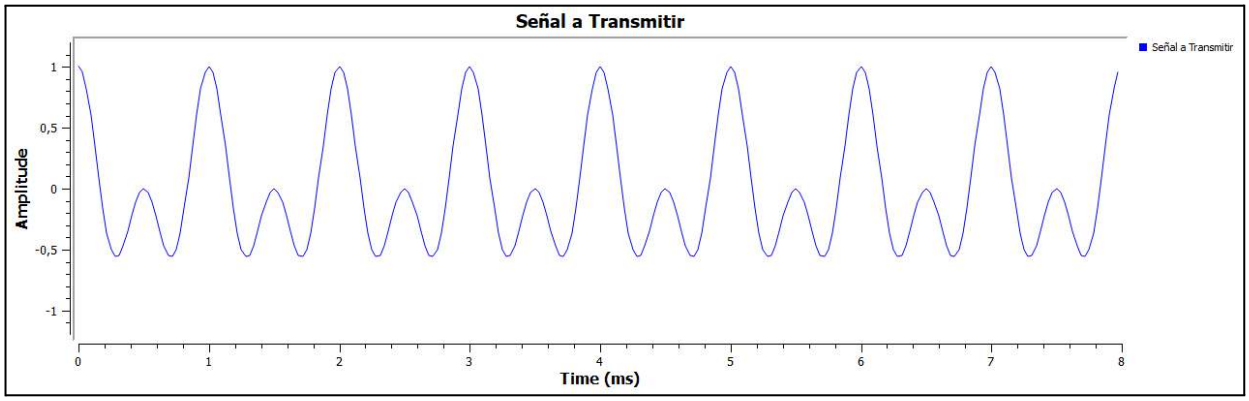

Figura 2: Señal Sinusoidal a transmitir.

La primera etapa consiste en una codificación de fuente, cuyo objetivo principal es reducir la redundancia en los datos para minimizar la cantidad de información a transmitir, manteniendo al mismo tiempo la integridad y calidad de la información original.
En esta etapa utilizamos el componente **CVSD Encoder** (Continuous Variable Slope Delta), el cual se encarga de realizar una interpolación y filtrado necesario para su funcionamiento con la codificación de voz. Este módulo convierte un flujo de datos de entrada en formato float (+-1) a formato short, lo escala (a 32000, ligeramente por debajo del valor máximo), realiza interpolación y luego lo somete a codificación de voz.

En el uso del codificador CVSD, las tasas de muestreo adecuadas oscilan entre 8k y 64k, con tasas de re-muestreo de 1 a 8. Una tasa de 8k con una tasa de re-muestreo de 8 proporciona una señal de buena calidad. Este módulo también cuenta con parámetros ajustables como el "Resample" (tasa de re-muestreo o interpolación antes de la decodificación del vocoder) y "Frac. Bandwidth" (ancho de banda del filtro pasa bajo, relativo, que debe ser <= 0.5).

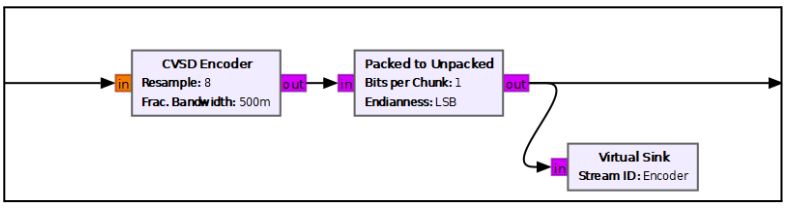

Figura 3: Etapa de Codificación de Fuente.

El CVSD Encoder entrega paquetes de 8 bits y a estos se los debe entregar la trama, para el correcto funcionamiento de la siguiente etapa se desempaqueta los bits usando el bloque **Packed to Unpacked**.

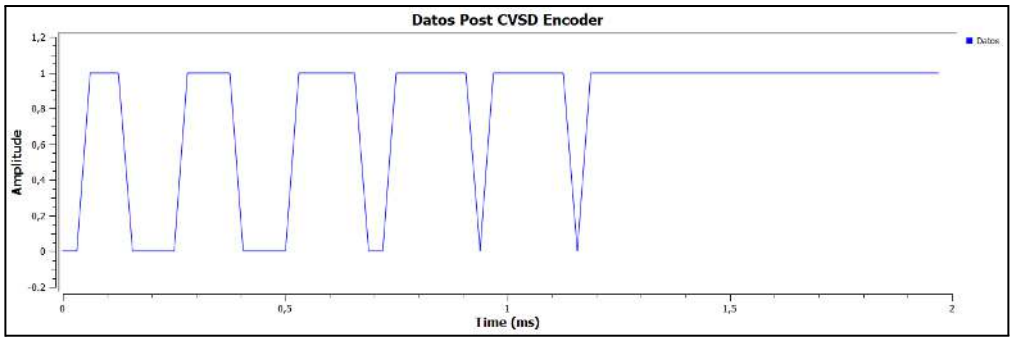

Figura 4: Datos Post CVSD Encoder.

Colocación de la Trama

Debido a que estamos trabajando con señales digitales no enviamos la información de manera continua sino que la enviamos *por paquetes*, es decir que agrupamos un cierto número de bits y los enviamos, una vez enviados agrupamos los próximos bits para volver a enviarlos. Esta manera de transmitir se utiliza por que permite multiplexar la señal en el tiempo además de multiplexar en frecuencia y de esta manera aprovechar mejor el canal de transmisión.

A estos paquetes de datos debemos agregarle información adicional, como bits de inicio (para saber donde comienza el paquete) y bits que de información sobre la modulación de la señal (para conocer en qué tipo de modulación se transmitieron una vez que los recibamos), también podemos agregar bits para codigo de correccion de errores y en caso de enviar la señal por internet agregamos datos para el protocolo TCP/IP y posiblemente datos de encriptación de información.

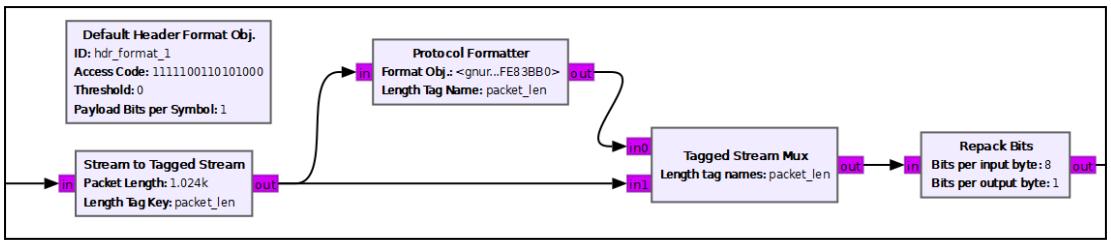

Figura 5: Etapa de Colocación de la Trama.

Toda la información se agrega en el encabezado, para agregar el encabezado comenzamos generando etiquetas, esto es lo que hace el bloque ***Stream to Tagged Stream*** que agrega una bandera cada cierta cantidad de bits, la cantidad es la que colocamos en la variable *Packet Length*, en este caso es una bandera cada 1024 bits. Podemos observar que el bloque le coloca un nombre a esta variable, en este caso se llama *packet_len*.

Luego debemos colocar el bloque ***Protocol Formatter*** el cual genera un encabezado cada una cierta cantidad de bits, es necesario que se agregue en cada paquete y por eso se coloco el valor *packet_len*. En la variable *Format Obj* se debe colocar la información que se desea agregar al encabezado, para eso se utiliza el bloque ***Default Header Format Obj***, este agrega un encabezado elegido por nosotros (el cual escribimos en la variable *Access Code*), este código de acceso en el receptor se utilizará para hacer una Convolución entre este código y el encabezado del paquete al llegar al receptor (debido al ruido algunos bits pueden cambiar, haciendo esto podemos saber si el paquete sufrió alteraciones o no debido al ruido, a mayor nivel de ruido mayor será la tasa de errores). La variable *threshold* cuenta la cantidad de errores admitidos y rechaza el paquete en caso de tener más errores que la variable colocada, la colocar 0 no rechazamos ningún paquete y la

variable *Payload Bits per Symbol* es la cantidad de bits por símbolos, en este caso es 1 debido a que anteriormente utilizamos el bloque *Packet to Unpacked* y que da a la salida 1 bit por símbolo.

Debido a que el bloque ***Protocol Formatter*** solo otorga el encabezado cada cierta cantidad de bits, debemos lograr colocar este encabezado en cada uno de los paquetes que debemos transmitir. Para esto utilizamos el bloque ***Tagged Stream Mux*** el cual coloca los símbolos en la entrada 1, luego los de la entrada 2 y así sucesivamente de esta manera obtenemos los paquetes de bits con su correspondiente encabezado.

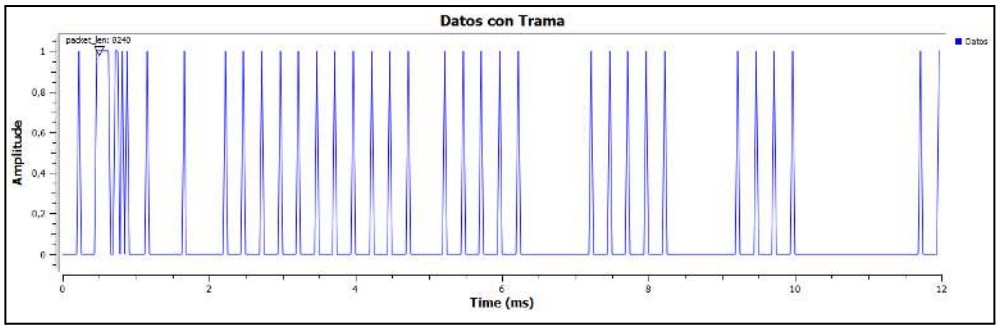

Figura 6: Datos incorporando la trama.

Codificador del Canal

Una vez agregado la trama a los datos lo siguiente es realizar la codificación del canal. Esta etapa se enfoca en asegurar la fiabilidad de la transmisión de datos a través de un medio propenso a introducir errores, como por ejemplo el ruido en una señal inalámbrica o interferencias en una transmisión por cable. La codificación del canal implementa técnicas para detectar y corregir errores que puedan surgir durante la transmisión. Algunos ejemplos de técnicas utilizadas en esta etapa son los códigos de corrección de errores como los códigos convolucionales, códigos de bloque como los códigos Reed-Solomon o códigos de repetición de bits.

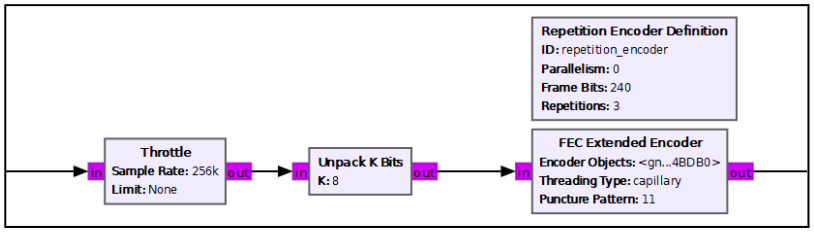

Figura 7: Etapa de Codificación del Canal.

En este proyecto, se empleó la técnica de repetición de bits mediante el uso del componente **FEC Extend Encoder**, el cual requiere un parámetro del tipo "Encoder Object". Este parámetro está asociado al tipo específico de FEC que se implementará, en este caso, el **Repetition Encoder Definition**. Este último requiere configuraciones adicionales como la determinación de paralelismo, el tamaño de los bits del marco (frame bits) y la cantidad de repeticiones por cada bit. En esta configuración particular, se desactivó el paralelismo, se estableció un tamaño de 240 bits para el marco y se aplicó una repetición de 3 veces para cada bit individual. Los demás parámetros del FEC se dejan por defecto.

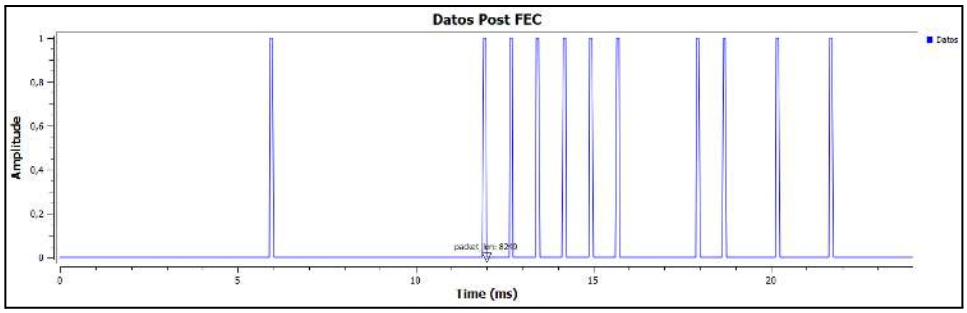

Figura 8: Datos Post FEC Extend Encoder.

Modulación de los Datos

La última etapa del transmisor consiste en la Modulación de la Señal, se optó por utilizar la **Modulación QPSK** (Quadrature Phase Shift Keying), en la cual los datos se dividen en pares de bits. QPSK es una técnica de modulación digital utilizada en sistemas de comunicación para transmitir datos a través de señales portadoras. En esta técnica, se manipula la fase de la onda portadora para representar múltiples bits por símbolo, lo que permite una mayor eficiencia espectral en comparación con la modulación de amplitud simple.

La QPSK es ampliamente utilizada en sistemas de comunicación modernos, como en telecomunicaciones por satélite, redes inalámbricas, sistemas de transmisión digital, entre otros, debido a su capacidad para transportar más datos en un ancho de banda limitado y su robustez frente a ciertas interferencias y atenuaciones del canal de transmisión.

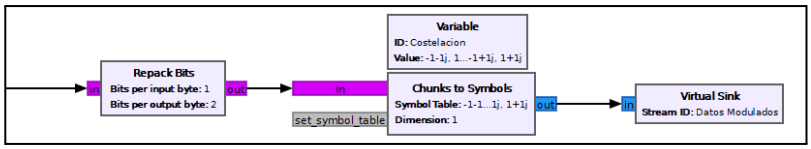

Figura 9: Etapa de Modulación.

Para llevar a cabo esta implementación, empleamos el bloque **Chunks to Symbols**. Este bloque requiere como parámetro una tabla de símbolos, la cual en nuestro contexto representa los diferentes valores utilizados para la técnica de modulación QPSK. Esta tabla de símbolos mapea los bits entrantes en los símbolos específicos de la modulación QPSK, cada uno representando una fase particular de la onda portadora. Esta conversión de bloques de datos (chunks) a símbolos es fundamental para la correcta representación y transmisión de la información en el esquema de modulación QPSK, permitiendo la eficiente codificación y decodificación de los datos para su transmisión y recepción.

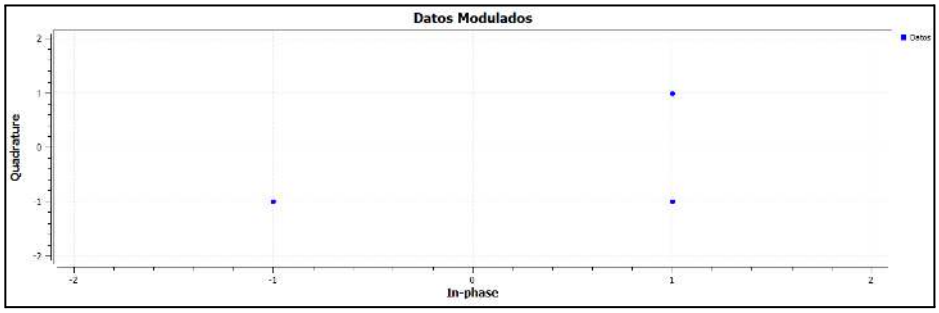

Figura 10: Datos Modulados en QPSK.

Además, se incorporó una fuente de ruido en la etapa final del transmisor para simular el ruido introducido por el canal de comunicación.

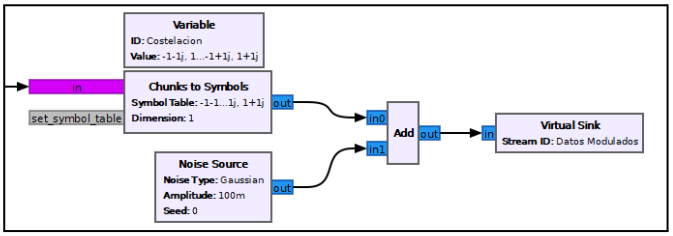

Figura 11: Etapa Final del Transmisor.

Con esta última etapa podemos concluir el transmisor. Lo siguiente será describir las etapas que componen al receptor. Las etapas del receptor son análogas a las del transmisor, con la diferencia de que se busca eliminar modulaciones y codificaciones para así recuperar la información original.

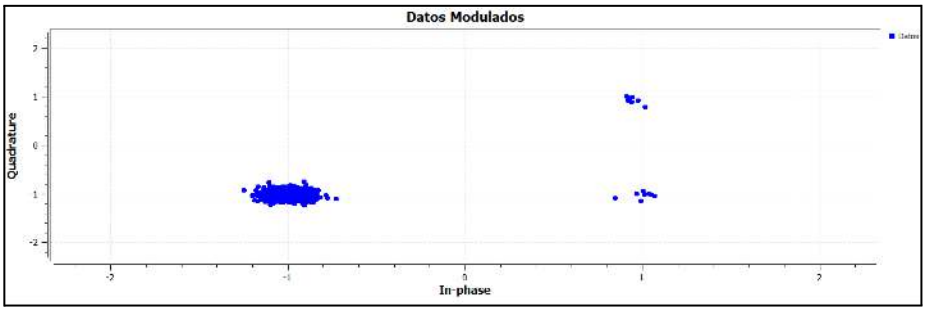

Figura 12: Datos Modulados en QPSK con Ruido Gausseano.

Receptor

Demodulación de los Datos

Se comienza recibiendo los datos modulados transmitidos, por lo tanto, debemos desmodularlos. Utilizamos el bloque **Constellation Decoder**, que recibe como parámetro la constelación utilizada para modular. Este bloque nos arroja paquetes con dos bits de información cada uno. Esto lleva a que usemos el bloque **Repack Bits** para desempaquetar los bits.

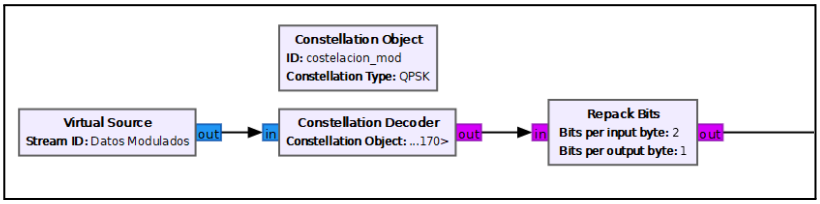

Figura 13: Etapa de Demodulación.

Decodificador del Canal

Para la decodificación del canal, utilizaremos el bloque **FEC Extend Decoder**, con el parámetro **Repetition Decoder Definition**. Este bloque es análogo al codificador utilizado, pero tiene la particularidad de que necesita un remapeo de los bits de entrada para su funcionamiento. Para realizar esto, se utiliza el bloque **Map**, donde se establece que para un bit de valor 0 se le asigna -1, y para un bit de valor 1 se mantenga en 1.

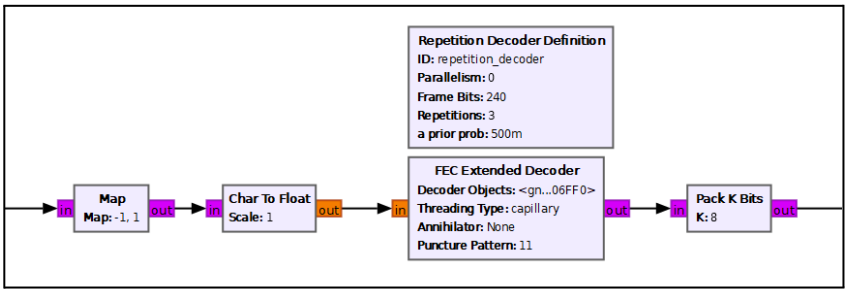

Figura 14: Etapa de Decodificación del Canal.

Para finalizar con esta etapa se coloca un bloque **Pack K Bits** con **K=8**, esto se debe a que la siguiente etapa necesita paquetes de 8 bits de información.

Detección de la Trama

Los datos agregados en la trama son solamente necesarios para una correcta transmisión. Es necesario separar la trama de la información transmitida ya que es dicha información lo que buscamos recuperar.

Para lograr esto utilizamos el bloque ***Header/Payload Demux*** que es un bloque que separa la trama de la información, en la salida *out_hdr* salen los datos de la trama y por la salida *out_payload* sale la información transmitida.

Para garantizar la transmisión los datos sacados en la trama deben ingresarse al bloque ***Protocol Parser*** el cual hace una correlación entre la señal de trama y una señal en particular (no es cualquier señal sino la que se colocó en el agregado de la trama, es decir que se coloca la

información del bloque ***Default Header Format Obj***), la salida del *Protocol Parser* se *header_data*, en caso de no coincidir el encabezado se envía una señal que descarta el paquete (la cantidad de errores admitidos es una variable que se colocó en el bloque *Default Header Format Obj*) y en caso de coincidir se envía la información por la salida *out_payload*.

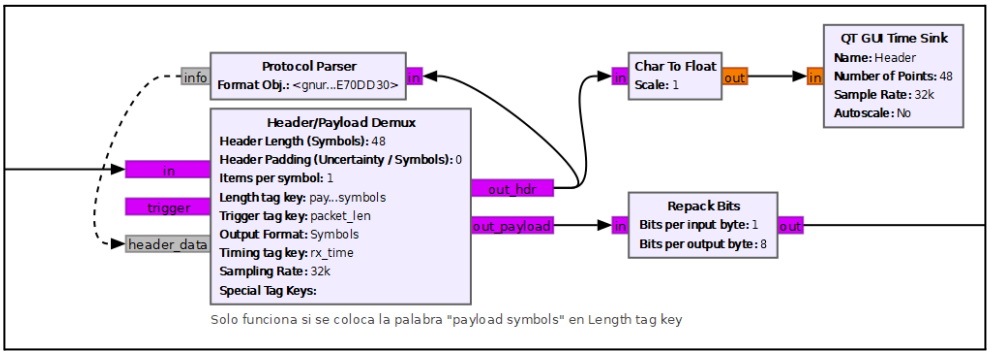

Figura 15: Etapa de Detección de la Trama.

Un dato importante que se colocó como comentario en el circuito es que el bloque ***Head/Payload Demux*** solo funciona si en la variable *Length tag key* se coloca la palabra "*payload symbols*"

Decodificador de la Fuente

Una vez que se elimina la trama de los datos se recurre a aplicar el último decodificador a los bits, el decodificador de fuente del mensaje. Utilizamos el bloque **CVSD Decoder**, dual al utilizado para codificar. Es necesario que los datos pasen previamente por un bloque **Unpacked to Packed** debido a que el decoder necesita trabajar con paquetes de datos.

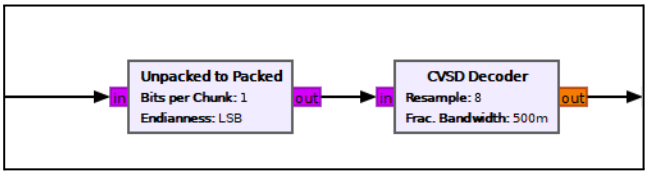

Figura 16: Etapa de Decodificación de la Fuente del mensaje.

Recuperación de la Señal

Trabajar con sistemas de comunicación digital introduce varios tipos de ruido durante el proceso, como el error de cuantización. Para minimizar esto, se emplea un filtro pasa banda que considera la frecuencia de nuestro mensaje a transmitir. En este caso, utilizamos el bloque **Band Pass Filter**.

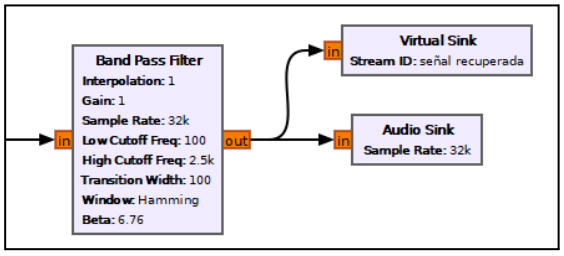

Figura 17: Etapa Final del Receptor.

En este bloque, se estableció la frecuencia de corte inferior en 100 Hz y la superior en 2500 Hz. Se optó por utilizar una ventana Hamming con una ganancia de 0 dB, mientras que los demás parámetros se mantuvieron en sus valores predeterminados.

Para poder escuchar el tono recuperado a la salida del filtro se utilizó el bloque **Audio Sink**. En las siguientes imágenes se puede observar la señal recuperada y su espectro de frecuencias.

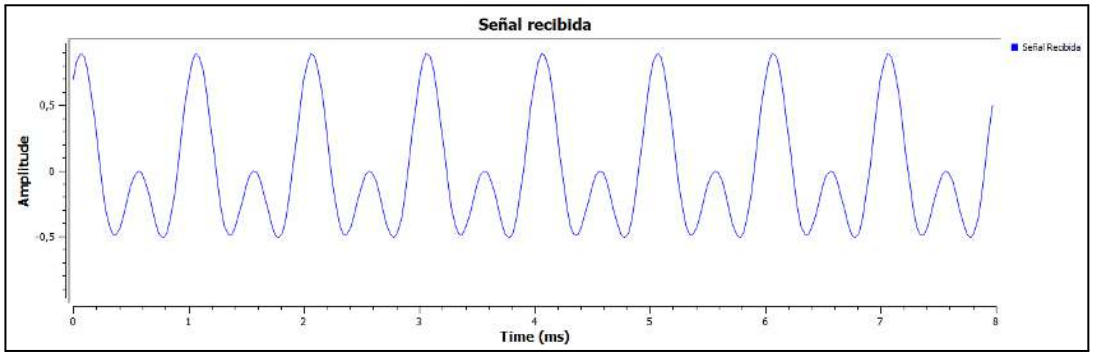

Figura 18: Señal adquirida por el Receptor.

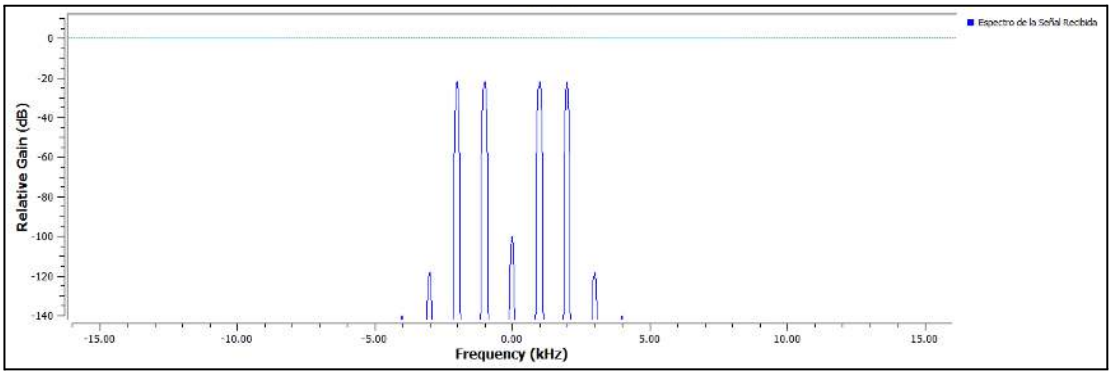

Figura 19: Espectro de la señal recibida.

Conclusión

Durante el transcurso de este proyecto, nos encontramos con varios desafíos, siendo el más notable la escasez de documentación proporcionada por el software GNU Radio. En numerosas ocasiones, nos vimos obligados a recurrir al código fuente en C++ para comprender el funcionamiento de cada bloque.

A pesar de estos obstáculos, logramos aprovechar al máximo nuestros conocimientos adquiridos durante el cursado, aplicando los principios fundamentales de transmisión y recepción digitales. Esta experiencia no solo nos permitió ampliar nuestros conocimientos, sino también fortalecer nuestra capacidad para abordar problemas técnicos complejos.

Este proyecto representó un desafío considerable que nos brindó la oportunidad de exhibir nuestcapacidad para adaptarnos y resolver problemas de manera efectiva. Además, el logro de implementar transmisiones y recepciones digitales exitosas refuerza nuestro dominio de los conceptos teóricos y su aplicación práctica