ISBN 978-987-46297-5-3

Implementación de un modem SDR con Protocolo AX.25 sobre una Beaglebone Black

Julián Alfonso Ferrari
Universidad Nacional del Sur (UNS)
SIAG - Armada Argentina
julianferrari3@gmail.com

Diego Marcelo Martinez
Universidad Tecnológica Nacional (UTN-FRBB)
SIAG – Armada Argentina
dmmartinez7@gmail.com

Christian Luis Galasso
Universidad Tecnológica Nacional (UTN-FRBB)
SIAG - Armada Argentina
christian_galasso81@yahoo.com.ar

Gustavo Javier Diaz
Universidad Tecnológica Nacional (UTN-FRBB)
SIAG – Armada Argentina
ing.diazgustavo@gmail.com

Resumen – Se describe a continuación, una implementación por software del protocolo AX.25 en su forma más básica, para usarlo con un modem SDR (Software Defined Radio) como transceptor. El lenguaje de programación utilizado es Python y la herramienta de desarrollo del modem GNURadio. Se realiza una comparación con otros programas comerciales como Soundmodem o Direwolf, y se destacan las ventajas y desventajas. Se agrega además la implementación del protocolo KISS para ser usado como TNC (Terminal Node Controller). Finalmente, se detalla la implementación sobre una placa Beaglebone Black para usar el modem de forma modular.

Keywords—AX25; SDR; GNURadio; modem, python; radio comunicación; protocolo; KISS; Beaglebone Black.

I. Introducción

AX.25 es un protocolo de comunicación de datos usado ampliamente por radio aficionados. Para las bandas de VHF y UHF la configuración típica de modulación es *1200 baud Bell 202 tones* y *9600 baud G3RUH DFSK*, mientras que en la banda de HF es muy usada la modulación *300 baud Bell 103 tones*. Además muchos satélites implementan este protocolo en sus transmisiones, como el FMN-1 (FengMaNiu-1) o el NO-44 (PCSat – Navy OSCAR 44) [1].

La motivación de trabajar con este tipo de comunicación radica en compatibilizar con sistemas antiguos pero actualmente operativos en algunas estaciones de radioaficionados y unidades navales. El objetivo postrero una vez alcanzado este hito, es utilizar protocolos más robustos y modulaciones más eficientes, basándose en el mismo lenguaje de programación y sobre la misma plataforma de desarrollo.

El documento oficial del AX.25 [2] detalla la totalidad de los campos que el protocolo implementa, pero no todos resultan necesarios para establecer una comunicación. En este trabajo se implementa la configuración mínima que asegura una comunicación de datos punto a punto. Por ejemplo, los campos *Address*, *Control* y *PID* no son necesarios para el desarrollo que aquí se plantea. Solamente usando los campos *Info* y *FCS (ver figura 1)* se puede establecer un enlace de datos entre módems, e incluso también con otros programas como Soundmodem y Direwolf [3]. En este sentido, se podría establecer rápidamente y a muy bajo costo, una red de comunicación interoperable incluso con sistemas de radio amateur.

La razón de utilizar Python como lenguaje de programación para implementar el protocolo se debe a una cuestión de compatibilidad, pues la herramienta GNURadio genera archivos Python. De todas formas se podría utilizar otro lenguaje.

Con GNURadio se puede implementar cualquier estructura de comunicaciones utilizando como *front-end* la placa de sonido de una computadora, o incluso una placa de sonido externa USB. Programas como Soundmodem o Direwolf utilizan el mismo criterio de transmisión y recepción. El enlace de comunicación se establece solamente a través de audio, con una conexión directa entre salida de audio y entrada de micrófono, o también a través de un enlace de radiofrecuencia, valiéndose de la entrada de micrófono del equipo transmisor, y la salida de audio de recepción. De todas maneras, el canal de comunicación que se use no es motivo de estudio en este trabajo.

II. Etapas del protocolo AX.25

El estándar define un paquete de transmisión con varios campos, entre ellos un campo de dirección donde se detalla la fuente que envía el mensaje y el destinatario, un campo de control que identifica el tipo de paquete (I. S o U), y en algunos casos un campo PID que identifica el tipo de protocolo de la capa 3 del modelo OSI, en caso de utilizarse alguno.

El paquete básico de AX.25 se compone de tres campos:

ISBN 978-987-46297-5-3

Flag	Info	FCS	Flag
01111110	N*8 Bits	16 Bits	01111110

Figura 1: Campos básicos del paquete AX.25

En las siguientes secciones se describen cada uno de los campos a implementar:

A. Flag Field

Este campo consiste de un octeto o palabra de ocho bits, y se usa para delimitar los paquetes a transmitir. Es decir se usa uno al comienzo de los datos, y otro al final. El octeto es:

0b01111110 o en hexadecimal: 0x7E

El receptor debe detectar estos *Flags* para poder recuperar la información que hay entre ellos. Más adelante se detalla una técnica para que el receptor pueda encontrarlos.

Este *Flag* es único e irrepetible, es decir que este patrón de bits no puede existir dentro de los datos, y para asegurarse de ello se realiza un *bitstuffing* [4].

B. Info Field

Contiene la información a enviar, y debe estar compuesto de un número entero de bytes u octetos. Cada uno de estos octetos se debe enviar con el bit menos significativo primero (LSB). Entonces es necesario invertir cada octeto, de forma que un mensaje como el siguiente:

0x41 0x42 0x43 0x44 0x45

resulta en:

0x82 0x42 0xC2 0x22 0xA2

C. FCS Field

El campo FCS (Frame Check Sequence) es un número de 16 bits calculado tanto por el transmisor como por el receptor. De esta manera se asegura que el paquete no se corrompió debido al medio de transmisión. El FCS se calcula de acuerdo a las recomendaciones de la norma ISO 3309 (HDLC), que define un código CRC-CCITT con el siguiente polinomio [5]:

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

El resultado son dos bytes o dos octetos, que representan el código correspondiente al campo de información *Info Field.* Solo el campo FCS debe enviarse con el bit más significativo primero (MSB).

D. Bit Stuffing

Hasta este momento, el paquete resultante se compone de un campo de información *Info Field*, con cada uno de sus octetos invertidos (LSB), y de un campo FCS con cada uno de sus octetos normal (MSB). Antes de aplicarle los campos *Flag Field* al principio y final del paquete, es necesario aplicar *bitstuffing* a los datos

Como el *Flag Field* se compone de seis "unos" consecutivos, este patrón no puede existir en los datos, porque de otra manera el receptor no sabría dónde empieza y termina un paquete. Por eso, cuando en los datos aparecen cinco "unos" consecutivos se debe insertar un "cero" como sexto elemento, o sea después del quinto "uno", independientemente del valor del sexto bit. El receptor sabe que, si encuentra cinco "unos" consecutivos, el sexto valor (el "cero" insertado) debe ser eliminado.

Por ejemplo, los datos originales:

…0100110111111100101111100110111001 01…

resultan en:

…0100110111111**0**1100101111**0**0011011100101…

De ser necesario, se deben agregar "ceros" al final para conservar un número entero de octetos. Se debe tener esto en cuenta para el receptor.

E. Paquete Final

El paquete completo para enviar debe tener al menos un *Flag Field* al principio y otro al final (*Pueden agregarse más de un Flag Field, dependiendo del algoritmo de detección del receptor*). Esto es:

Flag + Info + FCS + Flag

Es importante destacar que, para que el receptor pueda sincronizarse adecuadamente, es conveniente enviar un tono de portadora o señal de portadora, al inicio y final de cada paquete. Esta señal de portadora (*carrier)* se agrega directamente sobre el paquete AX.25 de la siguiente manera:

… 0x00 0x00 0x00 ... 0x00 +
+ Flag + Info + FCS + Flag +
+ 0x00 … 0x00 0x00 0x00 …

Ahora el paquete está listo para ser enviado a la estructura de modulación.

III. Protocolo KISS

Este protocolo es usado para la comunicación entre el modem y una PC. Al igual que con el protocolo AX.25, el protocolo KISS se implementa en su forma más básica. Toda la información que se transmite y recibe se estructura entre los siguientes caracteres:

0xc0 0x00 data 0xc0

Los delimitadores *0xc0* indican el comienzo y final de un paquete, y el caracter *0x00* indica que el paquete contiene datos. De esta manera el protocolo KISS establece que lo que se quiere enviar son solo datos. Existen otros caracteres para realizar otras operaciones, pero con este ya se puede establecer una comunicación en su forma más simple. El protocolo tiene

ISBN 978-987-46297-5-3

muchas más funcionalidades que pueden observarse en el documento original [6].

IV. GNURADIO MODEM

Con GNURadio se implementó una arquitectura de transmisión y recepción de un modem FSK (*Frequency Shift Keying*) para un *baudrate* de 1200 y 300 bps. Para la entrada de datos se dispuso de un socket TCP al igual que para la salida, o sea para la recepción. Al socket de entrada se envía el paquete AX.25 previamente descripto y la arquitectura de modulación genera el audio de salida correspondiente.

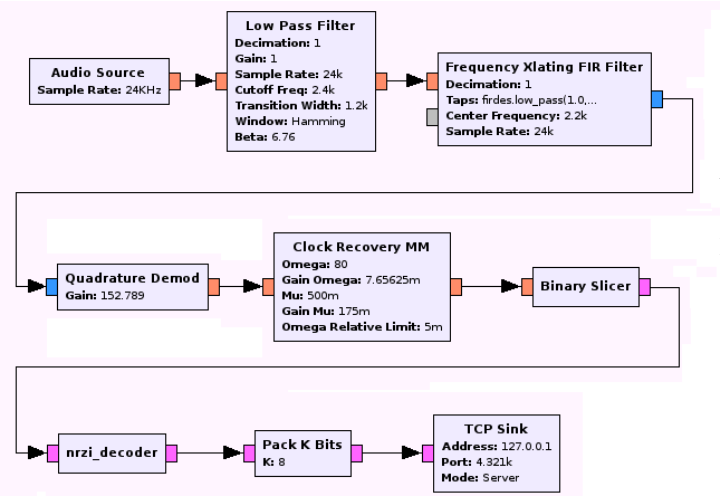

Figura 2: Receptor FSK

La etapa de recepción es más compleja. La arquitectura de demodulación recibe el audio y lo convierte en un vector binario de longitud variable *(ver figura 2)*. GNURadio envía estos datos continuamente por el puerto TCP de salida, incluso si es ruido lo que está ingresando. Estos datos son los que deben analizarse. Aquí se aplican algoritmos de detección adecuados para detectar los *Flag Field* y recuperar la información contenida en ellos. A continuación se propone un método de detección simple:

A. *Deteccion de portadora*

Dependiendo del largo de portadora que el transmisor haya definido, se establece un criterio de detección de portadora. Por ejemplo, para este modem se fijaron 30 bytes de portadora, por lo que una detección de 10 bytes de portadora es adecuada.

Se supone que la variable "dataIn" contiene el *stream* de datos binarios recibidos. Un criterio de detección de portadora puede ser el siguiente:

```
while True:
    if dataIn == '0x00':
            contador += 1
    if contador == 10:
            carrierDetect = True
```

Cuando se detecta presencia de señal de portadora se debe comenzar a grabar la secuencia de bits que está entrando hasta que se encuentren el *Flag* inicial y el *Flag* final, o hasta detectar la señal de portadora final.

Basándose en resultados empíricos, no se recomienda hacer una detección de portadora inicial y final para recuperar el paquete, pues el largo de esta señal puede variar según los algoritmos. Por ejemplo Soundmodem utiliza una señal de portadora final más corta que la inicial, y a la vez más corta que la que aquí se propone. Con Direwolf sucede algo similar. Esta incompatibilidad en cuanto al largo de la señal de portadora puede provocar que nuestro algoritmo de detección confunda la señale inicial como final y viceversa, y recupere datos incorrectos.

La mejor opción es hacer una detección de portadora inicial, y a partir de allí buscar el primer *Flag Field*. Una vez encontrado, empezar a buscar el siguiente *Flag Field* que corresponde al final del paquete. De esta manera se evita analizar datos incorrectos, si sucede el caso que se detecte la señal de portadora final en lugar de la inicial.

B. *Deteccion de Flag Field*

Una vez detectada la señal de portadora se espera que lo primero que aparezca sea un *Flag*. Dentro del *stream* recibido se debe buscar el patrón binario del *Flag* inicial y final. A continuación se ejemplifica un criterio para hacerlo:

```
if carrierDetect == True:

    flagInicial = dataIn.find('01111110')
    if flagInicial != -1:
            packet += dataIn

    flagFinal = dataIn.find('01111110')
    if flagFinal != -1:
    carrierDetect = False
```

Una vez logrado detectar un paquete, este debe ser decodificado siguiendo el proceso inverso con el que fue conformado. Primero los *Flag Field* se eliminan, conservando la información que entre ellos se encuentra. A continuación se debe hacer un *bitstuffing* inverso. Sabiendo que si se encuentran cinco "unos" consecutivos el sexto será un bit redundante, se procede a eliminar estos bits en todos los casos que se encuentren. Tener en cuenta que, si se agregaron "ceros" para conservar el número entero de bytes, éstos también deben ser eliminados.

A continuación se debe diferenciar la información recibida entre los campos *Info Field* y *FCS*. Se sabe que *FCS* siempre tiene una longitud de dos bytes, así que se puede asegurar que los últimos dos bytes del paquete recibido será éste código. Todos los bytes anteriores corresponden a los datos. Entonces, el siguiente paso es hacer una inversión de todos los octetos del *Flag Field*, recordando que éstos enviaron en formato LSB. Hecho esto, se debe calcular nuevamente el código *FCS* de los datos, y compararlo con el recibido. Si son iguales, se puede asegurar que los datos recibidos llegaron sin errores. Caso contrario podría pedirse una retransmisión del mensaje.

ISBN 978-987-46297-5-3

V. IMPLEMENTACIÓN SOBRE BEAGLEBONE BLACK

La idea de llevar el software a una placa de desarrollo se fundamenta en tener un modem portable y modular para ser instalado en una unidad de superficie. La Beaglebone Black [7] *(ver figura 3)* posee un procesador lo suficientemente potente para ejecutar los algoritmos y además tiene varios puertos de comunicación y pines GPIO para control de los equipos de radio.

Para la etapa de audio se utilizó una placa de sonido externa USB. El sistema no requiere de altas velocidades de muestreo, por lo que una placa de propósitos generales funciona adecuadamente.

Aprovechando la gran cantidad de pines GPIO con que se dispone, se realizó un circuito impreso estilo *"shield"* que se conecta sobre la Beaglebone Black directamente sobre la tira de pines *(ver figura 4)*. Con este circuito se administran las señales de control para los equipos de comunicaciones (*Push To Talk* - PTT), y los indicadores LED para saber cuándo se activa la radio y cuando se detecta señal de portadora.

Figura 3: Beaglebone Black

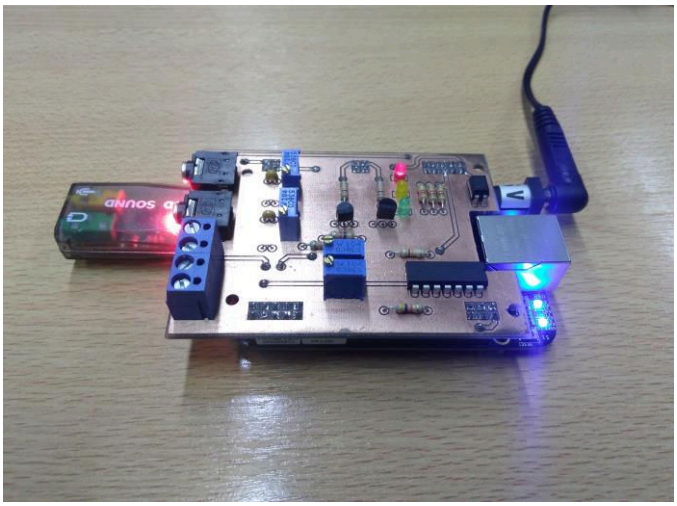

Figura 4: "Shield" y Placa de Sonido USB

La conexión al modem se realiza por Ethernet, a través de un puerto TCP por donde se envían los datos a transmitir, y los datos recibidos. Desde una computadora personal, utilizando el comando *Telnet* o desde cualquier software como *Realterm* o *Putty*, se puede enlazar al modem rápidamente, y empezar a transmitir y recibir *(ver figura 5)*. Desde los conectores del *"shield"* se cablea la salida *Push to Talk* (PTT) al pulsador del equipo de radio, y el audio de transmisión a la entrada de micrófono del mismo. La salida de audio de la radio, o sea el audio recibido por antena, se cablea a la entrada de micrófono de la placa de sonido del modem.

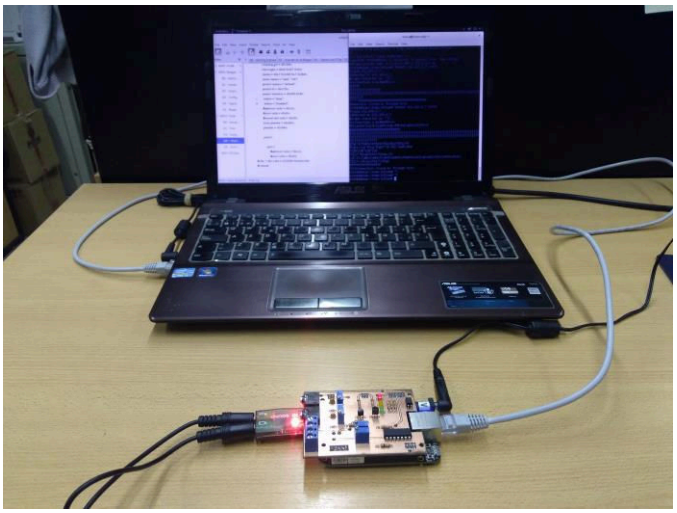

Figura 5: Notebook enlazada al modem por Ethernet

En la figura 6 se aprecia un esquema de funcionamiento general del sistema completo, desde la PC del usuario hasta el equipo de radiofrecuencia.

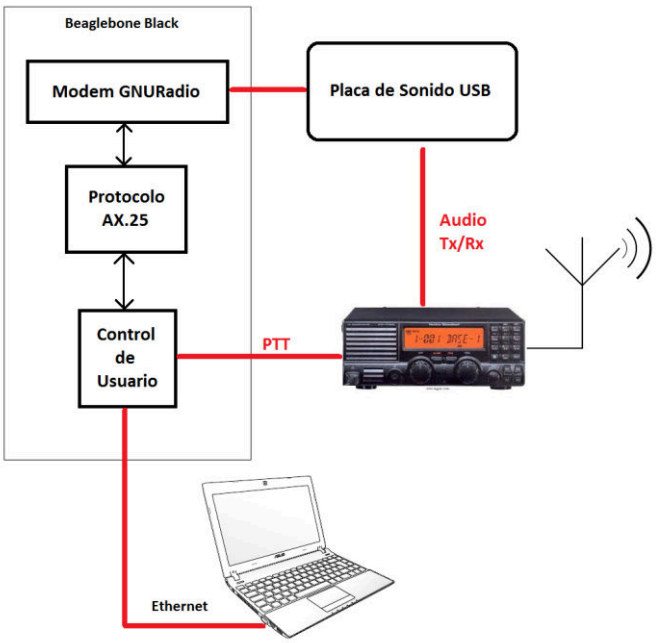

Figura 6: Esquema del sistema completo

ISBN 978-987-46297-5-3

VI. COMPARACIÓN CON OTROS SOFTWARES

Se estableció una red de comunicación entre el modem GNURadio, Soundmodem y Direwolf, y se obtuvo un 100% de compatibilidad entre los tres sistemas tanto en transmisión como en recepción. Un detalle para destacar, es que tanto Soundmodem como Direwolf no detectan paquetes menores a 13 bytes, pues un paquete menor a ese número no es considerado como un paquete AX.25, y lo descartan. Con el modem GNURadio se puede detectar desde un byte hasta N bytes, ya que los algoritmos se programaron con ese criterio.

El hecho de que la programación sea propietaria otorga la ventaja de tener mucho mayor control de los datos. Además, se tiene la posibilidad de usar y modificar señales de control como *Push To Talk* (PTT) y *Carrier Sense* (CS), y tener control absoluto de ellas.

A diferencia de Soundmodem y Direwolf, el modem GNURadio carece de interfaz gráfica, y es que no es necesaria porque la idea original siempre fue tener un sistema embebido que se ejecute con los sistemas operativos navales, y no para un usuario de forma directa.

Una ventaja del modem GNURadio es que es absolutamente reconfigurable. Por ejemplo, pueden cambiarse las frecuencias de portadora para sintonizarse con alguna transmisión en particular, o alterar la velocidad de bit para alcanzar mayores rangos de *baudrate*.

VII. RESULTADOS

Respecto a las pruebas de laboratorio, se sometió el modem a varios ensayos de transmisión y recepción. La metodología consistió en transmitir una gran cantidad de mensajes de largo aleatorio, y recibirlos con los programas Soundmodem y Direwolf. De la misma manera, el modem en la Beaglebone Black recibió mensajes de estos programas. La intención de estas pruebas fue la de verificar la compatibilidad de sistemas implementados con metodologías diferentes, y asegurar así la interoperabilidad de la red.

Otra prueba consistió en transmitir y recibir mensajes entre dos módems Beaglebone Black. Este ensayo se realizó bajo una simulación de canal AWGN con diferentes niveles de ruido, transmitiendo una cantidad considerable de mensajes de forma de tener una tasa de error de bit representativa. Se utilizó la modulación FSK para una velocidad de bit de 300 bps y 1200 bps. Los resultados se detallan en la figura 7.

Se aprecia que para niveles bajos de ruido el desempeño del modem es aceptable, pero disminuye a medida que el canal se deteriora. Esto podría interpretarse a grandes rasgos como el desempeño frente a un canal VHF o UHF, donde los niveles de ruido son relativamente bajos. Distinto es un canal HF, donde intervienen mayor cantidad de parámetros de ruido, interferencias, y dispersiones. Para este caso resulta conveniente utilizar otros protocolos que contemplen estas características, para compensarlas con algoritmos de corrección de errores, codificación, etc.

Es esperable que la modulación FSK 300 tenga mejor desempeño, en virtud de que utiliza una tasa de velocidad de bit menor.

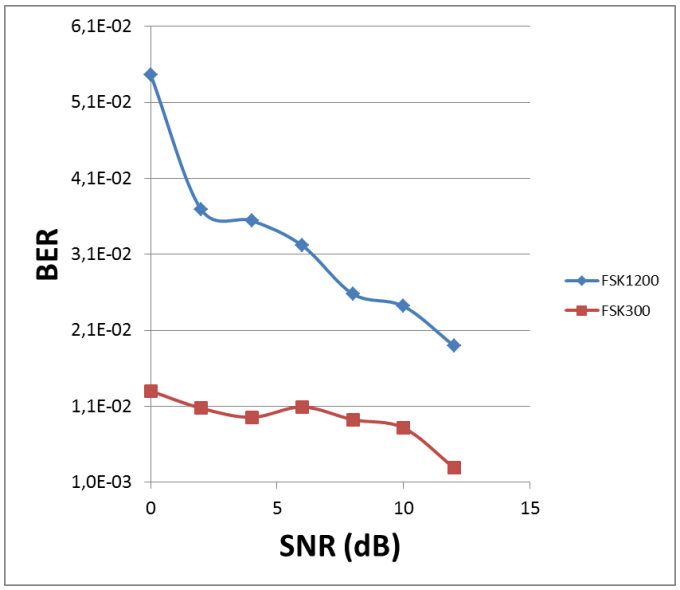

Figura 7: Simulación sobre canal AWGN

Se analizó además cuántos paquetes recibió el modem de manera correcta y cuántos erróneos. Puede suceder que un paquete recibido contenga bits corruptos que provocan que no se verifique el FCS. En este caso, se descarta el contenido de dicho paquete. En la figura 8 se estima el porcentaje de mensajes recibidos con error, es decir aquellos que no verificaron un FSC correcto. Se comparan las modulaciones FSK para una velocidad de bit de 1200bps y 300bps.

FSK1200	SNR(dB)	Mensajes con error (%)
	0	54,64684015
	2	36,89655172
	4	35,43046358
	6	32,14285714
	8	25,77639752
	10	24,16918429
	12	18,95043732
FSK300	**SNR(dB)**	**Mensajes con error (%)**
	0	13,01369863
	2	10,74380165
	4	9,554140127
	6	10,90909091
	8	9,243697479
	10	8,163265306
	12	2,898550725

Figura 8: Cantidad de mensajes recibidos con error

Se ensayó, así mismo, el modem en campo, estableciendo exitosamente enlaces de comunicación de datos entre unidades navales, tanto en puerto como en navegación. Se utilizaron

ISBN 978-987-46297-5-3

equipos de VHF ICOM IC-A110 de 9W de potencia de salida. Se preparó un cable con la disposición de pines adecuada, a saber *PTT*, *PTT GND*, *Audio Transmisión*, *Audio Recepción* y *Audio GND*, para conectar el *"shield"* de la Beaglebone con el equipo de comunicaciones. Se tomó en cuenta utilizar cable mallado, y resulta importante destacar una buena conexión de las masas de la Beaglebone, *"shield"* y placa de sonido, para evitar interferencias por la radiación del equipo de VHF cuando el modem se encuentra cerca.

Se establecieron radioenlaces relativamente cercanos, de aproximadamente 25 kilómetros de distancia, manteniendo una comunicación efectiva.

En la figura 9 se aprecia la salida de la señal modulada del modem correspondiente al modo de operación FSK 1200. La amplitud de salida de la placa de audio del modem es, en este ensayo, de 450 mV pico a pico. Es importante conocer este valor para no saturar la entrada del equipo de radio. Si bien no habría problemas de saturación para modulaciones de frecuencia (porque la información se refleja en cambios de frecuencia y no de amplitud), podrían existir inconvenientes para otros tipos de modulación. Tampoco se debe descartar la posibilidad de que un voltaje excesivo en la señal podría dañar la entrada de micrófono del equipo de radio. De todas maneras, tanto el volumen de salida como el de entrada del modem se pueden controlar por software a través de la arquitectura de sonido ALSA de LINUX. Utilizando el programa *alsamixer* se accede a estos controles y se ajustan adecuadamente.

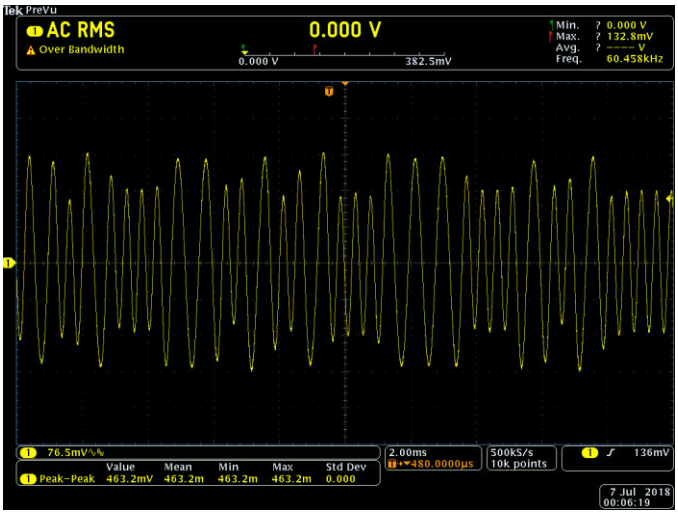

Figura 9: Señal de salida modulada FSK

Independientemente del modem, antes de comenzar a realizar pruebas se verifica un buen enlace de comunicación con los equipos de radio. Se comprueban convenientemente las conexiones a tierra y la adaptación de la antena en la frecuencia de trabajo. Es recomendable asegurar comunicación clara a la "voz", antes de establecer un enlace de datos.

VIII. Conclusiones

Se verificó que tanto la implementación del protocolo AX.25 como del modem FSK, ambos en lenguaje *Python*, son compatibles con programas como *Soundmodem* o *Direwolf*, como también lo son con equipamiento antiguo dispuesto en estaciones de radioaficionados y en unidades navales.

Se alcanzó una versión estable para ejecutarse sobre una placa de desarrollo Beaglebone Black, y se verificó la adecuada capacidad de procesamiento del procesador para ejecutar todas las instancias del sistema.

Es importante resaltar que se debe efectuar un robusto conexionado de masas entre el modem y los equipos de radio para evitar cualquier problema de interferencias por radiación. Puede utilizarse un gabinete metálico para aislar el modem aún más. Asegurando esto, tanto la Beaglebone Black como la placa de audio no sufren ninguna alteración

De las pruebas con canal AWGN simulado, se aprecia que el funcionamiento del modem es más eficiente para valores de SNR superiores a 10dB, o sea para niveles relativamente elevados. Se tiene en consideración el uso de otros protocolos de comunicación que tengan menor exigencia respecto a SNR.

.

Referencias

[1] AMSAT EA – Satélites de Radioaficionado. https://www.amsat-ea.org/

[2] AX.25 Link Access Protocol for Amateur Packet Radio. Version 2.2. Revision: July 1998. Authors: William A. Beech, NJ7P, Douglas E. Nielsen, N7LEM, Jack Taylor, N7OO.

[3] Soundmodem – The Software Packet-Radio TNC. http://uz7.ho.ua/packetradio.htm - UZ7HO.

Direwolf – Soundcard AX.25 packet modem/TNC and APRS encoder/decoder. https://github.com/wb2osz/direwolf - WB2OSZ.

[4] Section 3.6. AX.25 Link Access Protocol for Amateur Packet Radio. Version 2.2. Revision: July 1998. Authors: William A. Beech, NJ7P, Douglas E. Nielsen, N7LEM, Jack Taylor, N7OO.

[5] The Cyclic Redundancy Check (CRC) for AX.25.

http://practicingelectronics.com/articles/article-100003/article.php

[6] The KISS TNC: A simple Host-to-TNC communications protocol. Mike Chepponis, K3MC, Phil Karn, KA9Q. http://www.ax25.net/kiss.asp

[7] https://beagleboard.org/black