UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS
DEPARTAMENTO DE INGENIERÍA ELECTRICA

DISEÑO, CONSTRUCCIÓN E INTEGRACIÓN DE UN MÓDULO DE INTERFAZ DIGITAL ANALOGO PARA SISTEMAS REPETIDORES DE GLOBO SONDA USADOS EN SITUACIONES DE EMERGENCIA.

MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL ELECTRICISTA

IGNACIO ANDRÉS ABARCA MESA

PROFESOR GUÍA:
MARCOS DÍAZ QUEZADA

MIEMBROS DE LA COMISIÓN:
NICOLÁS BELTRÁN MATURANA
JUAN FRANCISCO SAINZ VALENCIA

SANTIAGO DE CHILE
NOVIEMBRE 2011

RESUMEN DE LA MEMORIA PARA OPTAR AL
TITULO DE INGENIERO CIVIL ELECTRICISTA
POR: IGNACIO ABARCA MESA
FECHA: 07/11/2011
PROF. GUIA: Sr. MARCOS DÍAZ QUEZADA

"DISEÑO, CONSTRUCIÓN E INTEGRACIÓN DE UN MÓDULO DE INTERFAZ DIGITAL ANALOGO PARA SISTEMAS REPETIDORES DE GLOBO SONDA USADOS EN SITUACIONES DE EMERGENCIA"

En esta memoria se propone un diseño para un módulo de interfaz digital-análogo, usado como modem, codificador y decodificador. Su función es enviar y recibir información digital a través de un canal de audio en radiofrecuencia. Se presenta además el estudio y análisis de la factibilidad técnica de este diseño.

En periodos de catástrofe o en lugares aislados los servicios de emergencia y seguridad requieren plataformas de comunicación exclusivas, portables y de bajo costo (en lo posible desechables). Como eventual solución se plantean globos aerostáticos y/o vehículos aéreos autónomos (UAVs) de bajo costo que puedan actuar como repetidor de comunicación de audio. Sin embargo la posición GPS de estos objetos es necesaria para el monitoreo y toma de decisiones de la comunicación haciendo necesario un módulo interfaz que permita transmitir los datos digitales de la posición a través de este canal de audio.

En particular el objetivo de este trabajo es el desarrollo de un prototipo del módulo interfaz que incluye tanto hardware como software. El hardware del módulo está compuesto principalmente por un microcontrolador y un modem de audio, mientras que el software de comunicación se basa en el protocolo APRS. En este documento se especifica el diseño del software y de los algoritmos que manejan el protocolo y las funcionalidades del sistema.

El hardware de bajo costo, el cual es requisito para esta solución, dificulta el uso de protocolos de comunicación estándares dado el ruido y distorsiones de las señales que este producen. Debido a esto el software de comunicación debe ser más robusto que los protocolos usuales. El nuevo protocolo presentado en este trabajo cambia la estrategia de muestreo y decodificación de los datos. Finalmente se realiza un análisis funcional del software que controla el hardware del módulo. Los resultados de desempeño del módulo son presentados junto con estrategias de mejoras futuras.

I. AGRADECIMIENTOS

A mis padres Ricardo Abarca y Marcela Mesa quienes siempre me han apoyado de manera incondicional y son un gran ejemplo a seguir para mí. Siempre estaré agradecido por la educación y los valores que me han entregado.

A Camila Escobar quien ha sido un pilar fundamental durante este proceso. Por todo el amor y cariño que me ha entregado en todo momento.

A mi familia por su preocupación y cariño.

A mis amigos que me han acompañado durante estos años de estudio.

II. CONTENIDO

I. Agradecimientos 3
II. Contenido 4
1 Introducción 7
1.1 Motivación 7
1.2 Alcance 7
1.3 Objetivos 8
2 Contexto 9
2.1 Comunicaciones de Radio 9
2.2 El Sistema Propuesto 12
2.3 Modos Digitales 15
2.3.1 Packet Radio AX.25 15
2.3.1.1 Capa Física 15
2.3.1.1.1 AFSK 16
2.3.1.2 Enlace de Datos (HDLC) 17
2.3.1.2.1 NRZI 17
2.3.1.2.2 Bit Stuffing 18
2.3.1.2.3 Banderas 19
2.3.1.2.4 Estructura del Paquete 20
2.3.1.2.4.1 Dirección de Destino y Origen 22
2.3.1.2.4.2 Dirección de Repetición Digital 23
2.3.1.2.4.3 Campo de Control e Identificador de Protocolo 26
2.3.1.2.4.4 Información 27
2.3.1.2.4.5 Bytes de Validación (FCS) 28
2.3.2 APRS 29
2.3.2.1 Ciclo de la Red 29
2.3.2.2 Transmisión Redundante de Paquetes 30
2.3.2.3 Identificadores de Dirección Genéricos de APRS 31

2.3.2.4 Tipos de Mensajes 31
2.3.2.4.1 Mensaje de Posición 32
2.3.2.4.2 Mensaje de Texto 33
3 Desarrollo del Sistema 34
3.1 Hardware 34
3.1.1 Modem MX614 35
3.1.2 Microcontrolador PIC16F877A 38
3.1.3 Comunicación con periféricos 40
3.2 Firmware 41
3.2.1 Recepción de Bytes y manejo de Bit Stuffing 42
3.2.2 Ajuste de fase por software en la Recepción 44
3.2.3 Recepción de Paquetes y Detección de Flags 46
3.2.4 Validación y Decodificación del Paquete 49
3.2.5 Algoritmo de Repetición 51
3.2.6 Envío de Bytes 51
3.3 Operación del Sistema 53
3.3.1 Parámetros de un TNC 53
3.3.2 Control y envió de datos a través de periféricos 54
3.3.3 Función main y ciclo principal 57
4 Análisis 58
4.1 Pruebas del Sistema 58
4.1.1 Prueba de Firmware en Sistema Reducido 58
4.1.2 Prueba de Modulación y Demodulación con el MX614 60
4.2 Discusión de los resultados 65
4.3 Mejoras realizadas para la recepción 67
4.3.1 Algoritmo de discriminación por tiempo entre cambios de nivel 67
4.3.2 Resultados obtenidos a partir de las mejoras 69
5 Conclusiones 72
6 Bibliografía 74

A. Anexo: Demodulación usando MATLAB 75
B. Anexo: Conexión con Radiotransmisor 78
C. Anexo: Detalle de los Resultado de Pruebas 79
D. Anexo: Recursos Usados en el Prototipo 81
E. Anexo: Código Firmware 82

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

El uso de las radiocomunicaciones es vital para sobrellevar distintas situaciones de emergencia. Aún cuando hoy en día existen otras tecnologías de comunicación como internet a través de fibra óptica o la red GSM de telefonía móvil, las radiocomunicaciones siguen siendo una opción real en situaciones de emergencia debido a su robustez y flexibilidad. Los sistemas de comunicación de uso diario como la red GSM e internet pueden colapsar fácilmente en situaciones de mucho tráfico o debido a la salida de operación de algún equipo de la red, sin embargo los sistemas de radiocomunicación no colapsan fácilmente ante estas eventualidades. Aun así estos sistemas están expuestos a factores que afectan su buen funcionamiento, como puede ser la topografía del terreno o zonas de silencio que no permitan establecer una comunicación. Ante esto, y motivado por el desarrollo que existe en el ámbito de los Vehículos Aéreos No Tripulados (UAV, Unmanned Aerial Vehicle) se ha pensado en una solución al problema que afecta las radiocomunicaciones donde se haga uso de globos aerostáticos capaces de repetir las señales de radio.

Para poder aplicar esta solución, además de un globo que repita las señales de radio, es necesario tener información telemétrica de éste como la posición, para que así el sistema sea efectivo. De lo anterior se desprende que es necesario recibir y eventualmente enviar información digital a este globo aerostático.

1.2 ALCANCE

La empresa IDETEC UAV CHILE, que se dedica a diseñar y construir aviones no tripulados (UAV) para monitoreo e integrar distintas tecnologías relacionadas, ha propuesto preliminarmente un sistema que incorpora la solución mencionada. El sistema propuesto consta de un globo aerostático y un repetidor de señales de bajo costo, pero no con un sistema de localización. El repetidor en cuestión solo posee una entrada de audio para poder transmitir comunicación por voz. En base a esto es necesario desarrollar un módulo que permita enviar información de posición usando el equipo repetidor mencionado o uno similar en cuanto a precio.

En resumen, se debe implementar un sistema que transmita información digitar a través de un canal analógico de voz.

Como alcance de este trabajo se plantea estudiar y proponer un módulo de interfaz digital-análoga que permita enviar la posición del globo aerostático y evaluar la posibilidad de que éste reciba información digital de tal manera que pueda formar una red de globos repetidores.

Para lograr esto se propuso usar parte del protocolo de comunicación APRS *(Automatic Position Report System)* que cumple con los requisitos. En cuanto al diseño del hardware, se propone adaptar el diseño de la tarjeta PIC-E de la empresa TAPR *(Tucson Amateur Packet Radio)* como referencia.

1.3 OBJETIVOS

De acuerdo a lo mencionado se plantean dos objetivos generales para este trabajo:

- Evaluar y diseñar un sistema que permita localizar la posición de un globo repetidor de señales usando un canal de audio.
- Identificar y evaluar las posibilidades para que el sistema anterior además pueda transmitir otro tipo de información digital como mensajes de texto.

El primer objetivo hace referencia al desarrollo e implementación de un sistema de seguimiento del globo, lo cual tiene relación directa con la etapa de desarrollo actual del producto.

El segundo objetivo está relacionado con la escalabilidad que se espera tener en este desarrollo. Si bien hoy día el interés principal es conocer la posición del globo, se espera que en desarrollos posteriores este producto sea capaz de transmitir otro tipo de información digital como mensajes de texto, hecho por el cual se deben explorar estas opciones y proponer la manera de implementarlas.

2 CONTEXTO

Este capítulo corresponde a una contextualización de los principios fundamentales necesarios para entender el trabajo realizado y el contexto en el que éste se enmarca. El capitulo comienza con la descripción de los principios básicos de las radio comunicaciones, luego se explica de manera general el sistema diseñado y las motivaciones de éste, terminando el capitulo con una revisión detallada de la tecnología y el protocolo usado en el diseño.

2.1 COMUNICACIONES DE RADIO

Se les denomina comunicaciones de radio [1] a aquellas que utilizan el espectro radioeléctrico para transmitir la información. Este espectro está conformado por todas las ondas electromagnéticas cuyas frecuencias están entre los 3 KHz y los 300 GHz. Dentro de este rango de frecuencias existen bandas de frecuencias, las cuales poseen distintos comportamientos físicos y a las que también se les da distintos usos dentro de las comunicaciones.

Debido al amplio uso que poseen en las comunicaciones, se destacan tres bandas de frecuencia:

- Frecuencias Altas HF (High Frecuency) : 3 – 30 MHz
- Frecuencias Muy Altas VHF (Very High Frecuency) : 30 – 300 MHz
- Frecuencias Ultra Alas UHF (Ultra High Frecuency) : 300 – 3000 MHz

De acuerdo a la regulación de la Subsecretaría de Telecomunicaciones de Chile, la banda de HF está asignada principalmente a comunicaciones de vehículos aéreos y marítimos, además de comunicaciones de dispositivos móviles y fijos. En la banda de VHF se encuentran las frecuencias destinadas a la radiodifusión televisiva y sonora. También en esta banda se encuentran otras frecuencias destinadas a la comunicación móvil. En la banda de UHF se encuentran frecuencias destinadas a la radiolocalización, ayuda a la meteorología, comunicación satelital y dispositivos móviles y fijos.

Dentro de la categoría de dispositivos móviles y fijos se encuentran las comunicaciones entre equipos de radio para comunicación de voz, usualmente usados por instituciones de seguridad y de gobierno como carabineros, cuerpo de bomberos, fuerzas armadas, la ONEMI e instituciones civiles de seguridad. Además es usada por entidades privadas como apoyo a sus tareas de logística y por radioaficionados.

Debido a que para la comunicación; usando transmisión vía radiofrecuencia; no se requiere de una red de comunicación, como podría ser la red de telefonía celular GSM o la red TCP/IP de internet, es un medio de comunicación muy bien evaluado para su utilización en caso de emergencia donde eventualmente las redes de comunicaciones tradicionales pudiesen quedar fuera de servicio.

Los equipos utilizados en la comunicación de voz en radiofrecuencia son denominados transceptores (*transceivers*) y son la pieza fundamental dentro de los sistemas de comunicación por radio frecuencias entre las entidades mencionadas, en la figura 1 se aprecia la imagen de un transceptor fabricado por la empresa YAETSU. Además de los transceptores existe otro elemento clave en la comunicación por radiofrecuencia, éste es el repetidor, cuya función es repetir señales emitidas por algún transceptor cuando estas no son capaces de alcanzar al transceptor que debe recibir la información.

FIGURA 1: TRANSCEPTOR YAESU FT-897D USADO PARA RADIOCOMUNICACION POR VOZ

Una señal puede no alcanzar al transceptor receptor por varias razones. Tres de éstas motivaron el diseño de sistema descrito en este documento. La primera razón es que la potencia de la señal emitida simplemente no sea suficiente para alcanzar al receptor debido a la atenuación en espacio libre que sufre cualquier onda electromagnética. La segunda razón afecta principalmente a las ondas en las bandas UHF y VHF, y es provocada por las irregularidades geográficas del terreno, como montañas existentes en el camino de la onda. La tercera razón afecta a las ondas de la banda de HF y es provocada por el fenómeno conocido como zona de silencio (skip zone). En la figura 2 se muestran esquemas representativos de las situaciones mencionadas.

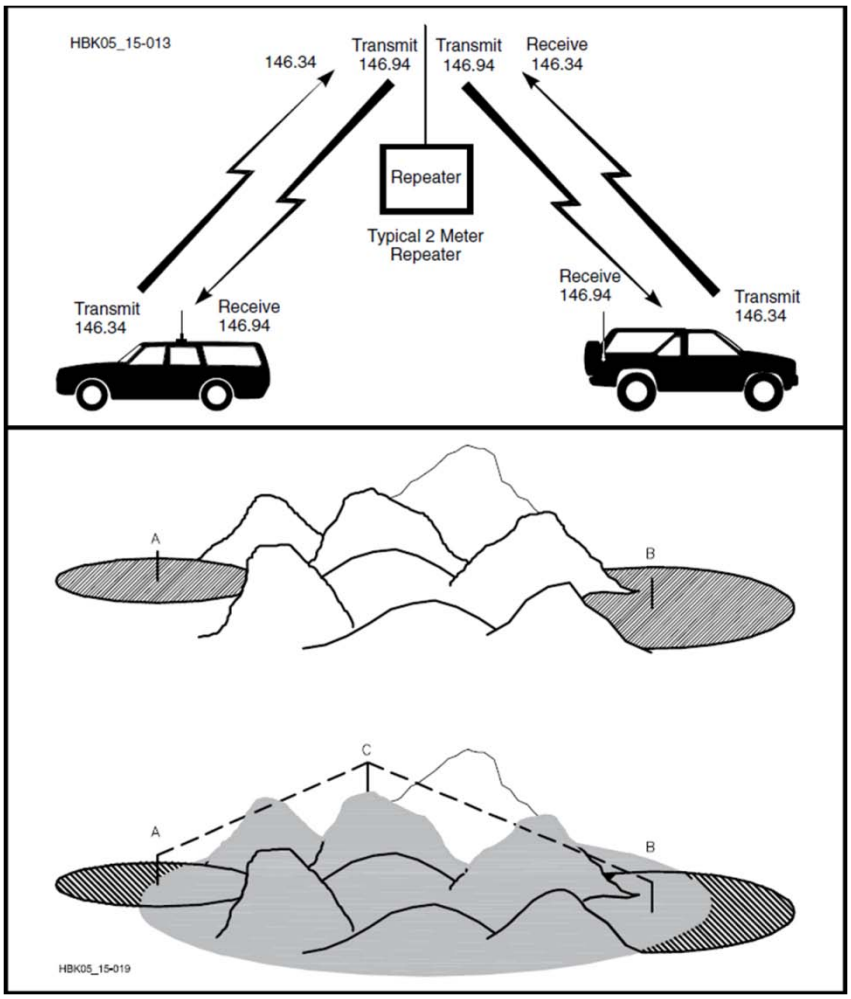

FIGURA 2: EN LA PARTE SUPERIOR DE LA IMAGEN SE MUESTRA EL FUNCIONAMIENTO DE UN REPETIDOR. EN LA PARTE INFERIOR SE ILUSTRA LA PERDIDA DE SEÑAL POR CONDICION GEOGRAFICA

El fenómeno de la zona de silencio se produce debido a que las ondas de HF, luego de ser emitidas y alcanzar la ionosfera son reflejadas hacia la superficie de la tierra. Esto permite alcanzar grandes distancias y para cortas distancias la onda es transmitida siguiendo la superficie de la tierra. Existen sin embargo zonas de media distancia donde la porción de la onda que es transmitida a nivel de la tierra no llega, pues se ha atenuado y la onda reflejada no es capaz de llegar, pues su ángulo de incidencia con la ionosfera es muy pequeño lo que provoca que la onda se difracte mas allá de la ionosfera. Esta zona es conocida como la zona de silencio, pues si un receptor se sitúa en esta zona no podrá recibir la señal debido al fenómeno descrito.

2.2 El Sistema Propuesto

IDETEC UAV CHILE es una empresa que nace en 2005 dedicada a las aeronaves no tripuladas o UAV (Unmaned Aerial Vehicle) y ha desarrollado con éxito su plataforma con avanzados sistemas de control de vuelo autónomo.

Esta empresa planteó la problemática ya mencionada acerca de la cobertura de las comunicaciones de radio y propone una solución preliminar que hace uso de un sistema de UAV y que está enfocada principalmente a situaciones críticas como emergencias nacionales o regionales. La solución está compuesta por un globo aéreo, similar a los usados para mediciones meteorológicas. Este globo lleva como carga un repetidor de señales de radiofrecuencia de bajo costo y que solo es capaz de transmitir señales de audio.

El bajo costo es un requerimiento fundamental, pues el sistema ha sido pensado para que la recuperación del globo no sea una necesidad además de que éstos puedan ser lanzados en cualquier momento de requerirse una mayor cobertura de señal. Sin embargo, no es posible hacer que el sistema sea eficiente solo con una comunicación de audio pues no es posible tener información telemétrica del globo. Especialmente relevante, es la información sobre la posición del globo, siendo fundamental para saber si éste está cubriendo la zona deseada o si es necesario enviar otro globo para que la cubra. A raíz de esto y por la necesidad de que el sistema que sea flexible ante futuras mejoras, es que se decide desarrollar un modulo de interfaz digital-análoga exclusivo para este sistema y que use el mismo canal de audio del repetidor de señales para enviar datos y así obtener la telemetría del globo.

En primera instancia se he planteado la posibilidad de monitorear solamente la posición de cada globo con un GPS instalado en éste y enviando los datos a través del canal de audio, sin embargo el hecho de enviar información digital abre la posibilidad de además transmitir mensajes de texto usando los globos como una red de comunicaciones. Por esta razón es que se ha pensado en un sistema que incorpore tres funcionalidades que se muestran en la figura 3 y se detallan a continuación:

- Repetición de Audio: Esta es la funcionalidad principal para la cual ha sido pensado el sistema, su funcionamiento depende del repetidor.

- Reporte de Posición: Esta funcionalidad envía desde cada globo la posición de éste a través del modulo digital-análogo usando el canal de audio del repetidor. El mensaje enviado es recibido por cualquier estación en tierra compatible con el sistema.

- Mensaje de texto: Esta funcionalidad usa el mismo modulo digital-análogo usado por el Reporte de Posición, pero en este caso se envía mensajes de una estación en tierra de origen a otra estación de destino usando los globos como una red de comunicaciones.

El trabajo detallado en este documento describe el diseño del modulo de interfaz digital-análoga necesario para los últimos dos modos de funcionamiento referidos, es decir el Reporte de Posición y el envío de Mensaje de Texto. Para esto se decidió hacer uso el protocolo AX.25 [2], también conocido como *Packet Radio,* el cual es ampliamente usado por radioaficionados y equipos de experimentación [3] como globos sondas para transmitir datos por los canales de audio de los equipos de radiofrecuencia. Lo anterior convierte a este protocolo en una herramienta ideal para los requerimientos del sistema.

Además, para el reporte de posición se usa un protocolo de más alto nivel que ocupa parte del protocolo AX.25 llamado APRS [4] (Automatic Positioning Report System) que también es muy usado por radioaficionados y en equipos de experimentación. APRS es un protocolo abierto y varios equipos de radio comunicación lo incorporan, por lo que el sistema propuesto poseería gran compatibilidad con equipos de otros fabricantes, lo que es visto como una ventaja para un eventual producto comercial.

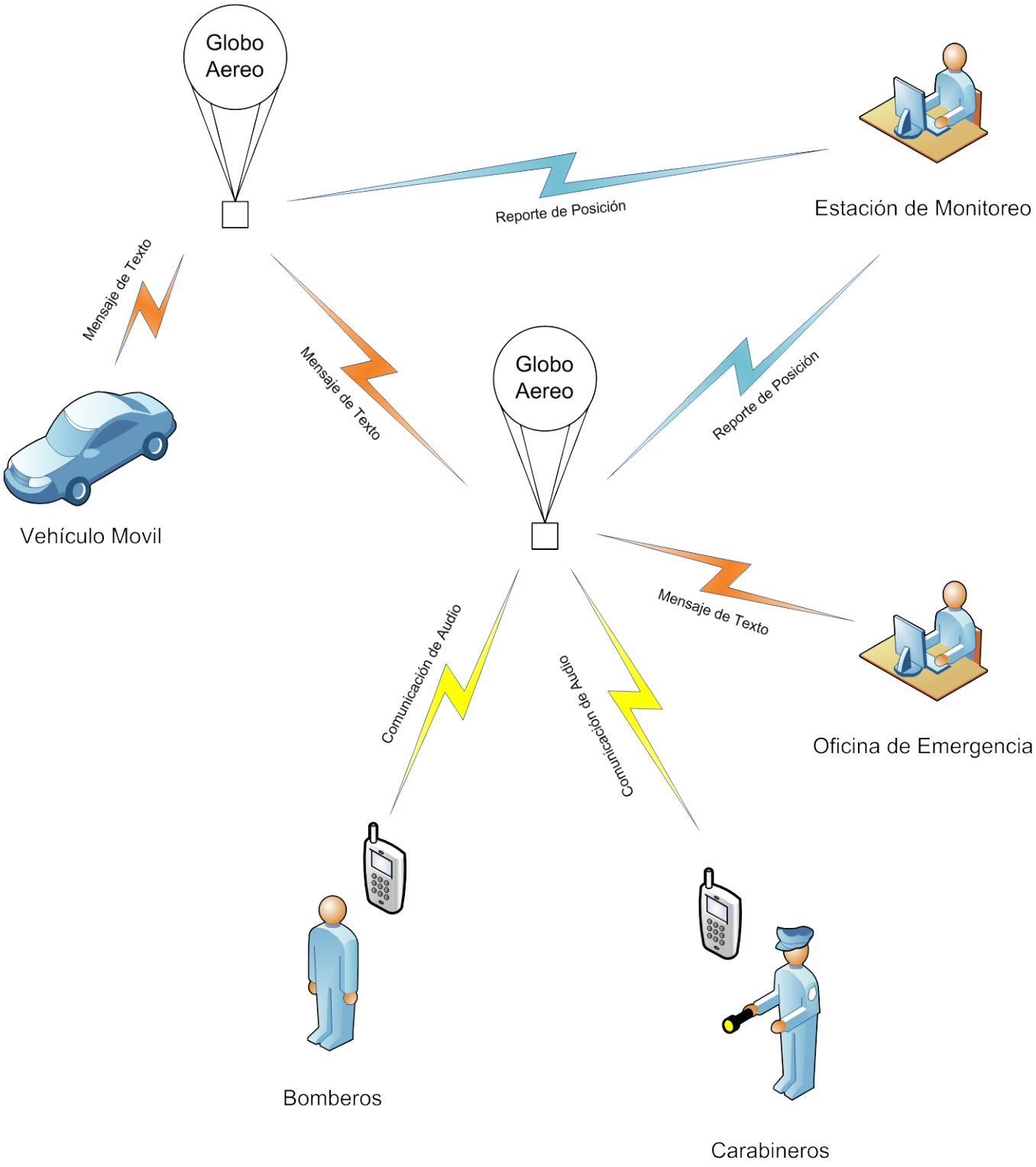

FIGURA 3: ESQUEMA DEL SISTEMA DE IDETEC CON LOS TRES MODOS DE FUNCIONAMIENTO DESCRITOS

2.3 MODOS DIGITALES

A continuación se revisará el protocolo AX.25 utilizado para la transmisión de paquetes de datos [3] sobre el canal de audio de un sistema de comunicación de radio frecuencia. Posteriormente se revisará el protocolo APRS usado para ubicar geográficamente estaciones móviles que poseen un GPS.

2.3.1 PACKET RADIO AX.25

El protocolo AX.25 [2] o *Packet Radio,* es un protocolo diseñado para la transmisión de datos usando la radiofrecuencia. Este protocolo fue desarrollado como una adaptación enfocada a la radiofrecuencia del protocolo X.25 usado para transmitir paquetes de datos a través de las líneas telefónicas. Actualmente el protocolo AX.25 es bastante usado para la transmisión de datos a través de equipos de radio especialmente en experimentaciones con globos aéreos y sondas. El AX.25 fue pensado principalmente para trabajar en la banda de VHF, sin embargo no es de exclusivo uso para ésta banda, pudiendo también ser utilizado en HF o UHF.

El protocolo AX.25 ha sido diseñado conforme a los estándares de la Organización Mundial de Estándares, ISO (International Standards Organization), la cual usa el modelo de capas OSI para distinguir los distintos niveles de los protocolos de comunicación. El protocolo AX.25 forma parte de las dos primeras capas del modelo OSI, esto es: la capa física y la capa de enlace de datos. A continuación se detalla la operación de este protocolo de acuerdo a las dos capas mencionadas del modelo OSI.

2.3.1.1 Capa Física

El protocolo AX.25 utiliza una modulación AFSK (Audio Frequency Shift Keying) con una tasa de transferencia en 1200 bps. Esta especificación es la misma usada por los módems Bell 202[1].

[1] El modem Bell 202 es un modem usado para transmitir datos a través de cables telefónicos.

2.3.1.1.1 AFSK

La modulación AFSK usa dos tonos de audio para modular la señal digital a diferencia de la modulación regular FSK donde la frecuencia de *carrier* es modulada en frecuencia con la señal digital lo cual requiere de una conexión directa al equipo de radio frecuencia. AFSK, al trabajar con señal de audio, puede usar cualquier transceptor que tenga una entrada de audio diseñado para voz. En resumen la señal digital es transformada en una señal de audio que posee la información y luego ésta es transformada en una señal de radio siendo modulada en FM por el transceptor, así viaja por el espectro radioeléctrico como una señal de audio FM cualquiera, sin embargo contiene información digital. En la figura 4 se muestra una imagen de esta modulación, en azul se aprecia la señal digital y en rojo la señal análoga modulada.

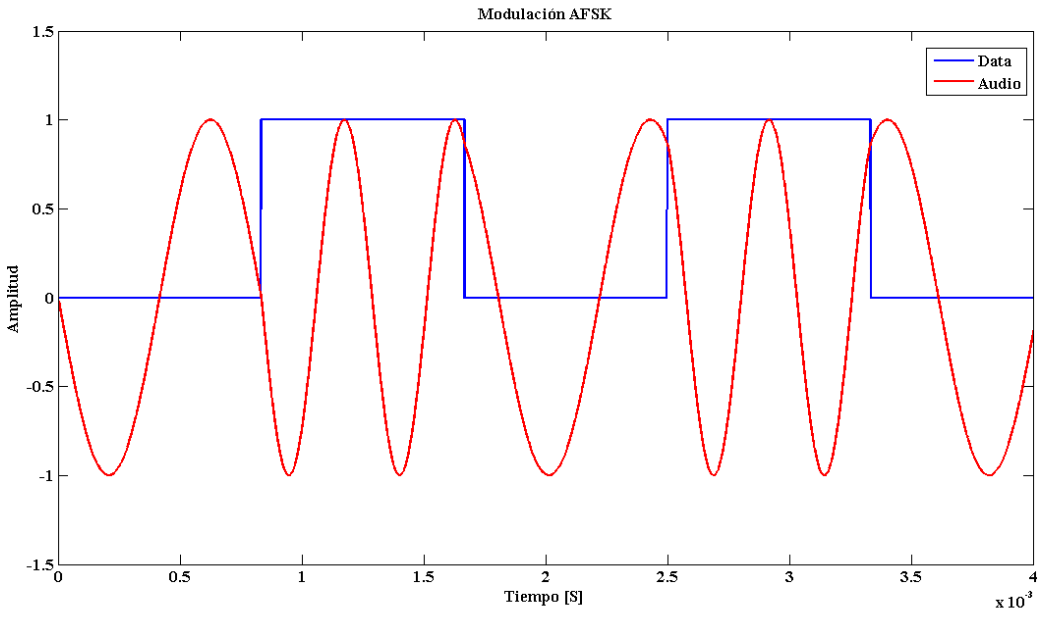

FIGURA 4: MODULACIÓN AFSK

El sistema usa un tono de 1200 Hz para indicar un nivel bajo (L) y un tono de 2200 Hz para indicar un nivel alto (H). Además el protocolo AX.25 utiliza una codificación NRZI (Non Return to Zero Inverted) la cual se detallará más adelante. Es importante también saber que el cambio de frecuencia debe ser continuo, es decir

no puede existir un desfase en la señal pues los dispositivos no están diseñados en general para señales de este tipo.

En la figura 4 se observa una imagen de una señal con una tasa de transferencia de 1200 bps modulada con AFSK en los tonos 1200 Hz y 2200 Hz al igual que la señal del sistema diseñado.

2.3.1.2 Enlace de Datos (HDLC)

La capa de red del protocolo AX.25 usa la estructura del protocolo de comunicación HDLC (High Level Data Link Control). Los bits en este sistema son enviados de menos significativo a más significativo y a una tasa de transferencia de 1200 bps como ya se ha mencionado. Además, el protocolo HDLC posee otras características que hacen que la comunicación sea confiable evitando desincronizaciones y detectando errores en la transmisión, estas características se detallan a continuación.

2.3.1.2.1 NRZI

La comunicación en el sistema es asíncrona, es decir que no existe un reloj común que sincronice la recepción y envío de bits. Al ser la onda transmitida por el aire, existe una gran susceptibilidad por parte de la señal a que su tasa de transferencia varíe durante la comunicación. Los dispositivos semiconductores de los equipos de envió y recepción también pueden introducir variaciones de tiempo. Esto pudiese provocar que no se detecten los bits válidos en los instantes correspondientes. Por estas razones el protocolo AX.25 utiliza una codificación de la información NRZI.

En la codificación NRZI la información depende del cambio de estado de la señal, donde el bit 1 se representa por un no cambio del estado de la señal y el bit 0 se representa por un cambio en la señal, de esta manera varios bit 0 seguidos se representaría por una señal oscilante entre estado alto (H) y nivel bajo (L) con la frecuencia igual a la tasa de transferencia, es decir 1200 bps para AX.25, y varios bits 1 seguidos se representaría como una señal que no cambia su estado en el tiempo siendo continuamente alto (H) o bajo (L).

En la figura 5 se puede observar un ejemplo donde se codifica el byte 01100010 usando NRZI y usando una taza de trasferencia de 1200 bps como en el sistema diseñado.

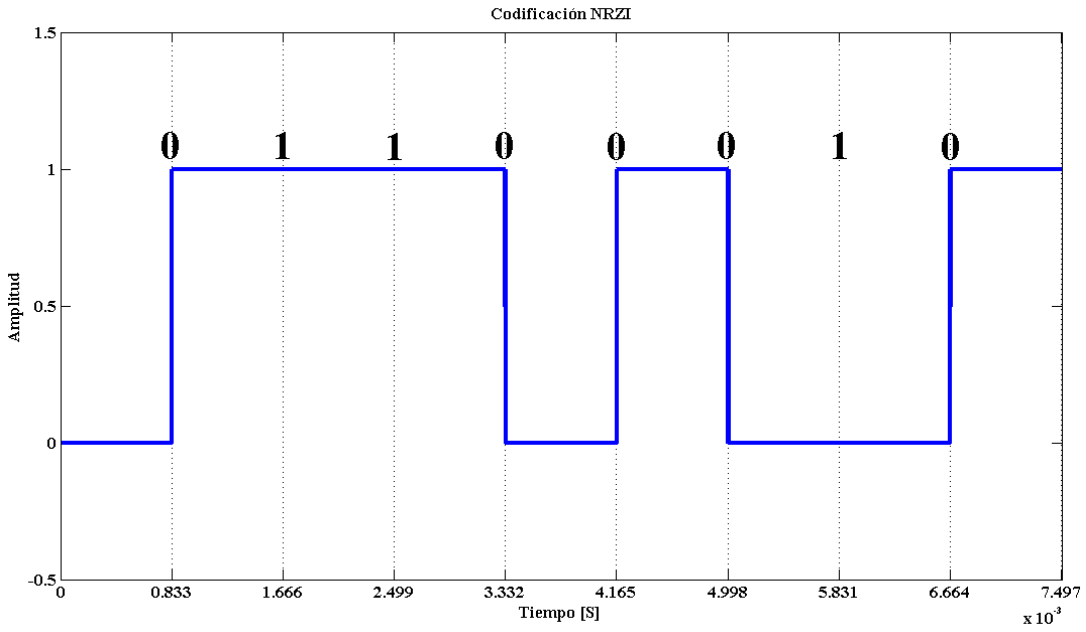

FIGURA 5: CODIFICACIÓN NRZI

2.3.1.2.2 Bit Stuffing

En el protocolo HDLC no existe una forma de separación entre bytes, a diferencia por ejemplo del protocolo RS232 donde cada byte tiene un bit de inicio y uno de fin que permite una sincronía en la recepción, en cambio HDLC envía los bits de un paquete de manera continua despreocupándose de los bytes. De acuerdo a lo anterior pudiese ocurrir que se envíe una trama de información con muchos bits 1 seguidos lo cual implica una señal sin cambios de estado, es decir una señal constante, lo cual puede hacer que el receptor entre en desincronización con la señal del transmisor.

Para evitar esta desincronización el protocolo establece que no se pueden enviar más de 5 bits 1 seguidos y en caso de que se deba enviar más de 5 bits 1 seguidos se debe introducir un bit 0 después del quinto bit 1 para luego continuar con la trama normal de información. A este procedimiento se le denomina *Bit Stuffing*. Bajo esta misma

lógica el receptor debe detectar que ha recibido cinco bit 1 seguidos y luego desechar el bit 0 que reciba a continuación pues este no es parte de la información y solo tiene por objetivo mantener la sincronía.

En la figura 6 se muestra un ejemplo donde se aplica el *Bit Stuffing*. En este caso se están enviando dos bytes que corresponden al número en notación hexadecimal 0x2FEB y se puede observar que se inserta un bit 0, que se destaca en color rojo, después de cinco bit 1 seguidos. Hay que recalcar que como se ha mencionado el envío de bits se hace de menos significativo a más significativo.

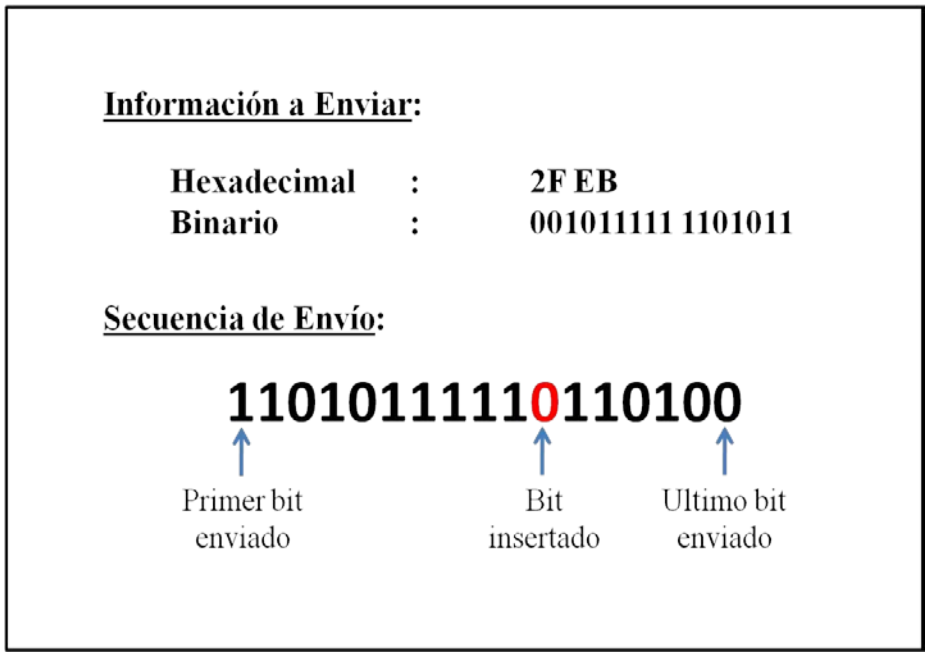

FIGURA 6: EJEMPLO DE BIT STUFFING

2.3.1.2.3 Banderas

Como se ha mencionado el protocolo HDLC no indica en qué momento termina la transmisión de un byte, enviando la información de manera continua, sin embargo sí se debe tener un indicador para saber cuándo se está recibiendo información valida. Para esto se usan bytes banderas o *Flags* al inicio y al final del paquete. El *Flag* para indicar el comienzo y fin de un paquete en HDLC es el byte 01111110 en el cual no se aplica el *Bit Stuffing* y por lo tanto la trama de datos enviadas corresponde efectivamente a la trama 01111110.

Cuando el receptor detecta cinco bits 1 seguidos debe esperar recibir un bit 0 para sincronizar, sin embargo cuando se recibe un *Flag* esto no ocurre, ya que el receptor recibe seis bit 1 seguidos, siendo esto lo que le indica al receptor que está recibiendo un *Flag* y que la información que reciba a continuación es información válida hasta que reciba otro *Flag*.

Es importante tener en cuenta que la validación del *Flag* como tal no depende únicamente de los seis bit 1 recibidos de manera continua, sino que también estos deben estar seguidos de un bit 0. Puede darse el caso que se reciban siete bit 1 seguidos o más. Estos casos pueden corresponder a un error o a un estado ocioso (*Idle)*. Si se reciben continuamente entre siete y quince bits 1 seguidos de un bit 0 implica que existe un error en la transmisión y por lo tanto el sistema aborta el proceso de recepción comenzando uno nuevo. Si se reciben más de quince bit 1 seguidos significa que el sistema esta ocioso, o sea no se está recibiendo información.

Por la forma en que se transmiten los datos no es posible recibir la información de a bytes y luego analizarla y decodificarla. Lo que se debe hacer en este caso es usar una maquina de estado que va cambiando su estado de acuerdo a los bits que se reciben y no de acuerdo a los bytes, de manera de poder saber qué información esperar.

2.3.1.2.4 Estructura del Paquete

A continuación se describirá de manera general la estructura de los paquetes del protocolo AX.25 para luego detallar en particular el tipo de paquete usado en el sistema desarrollado. En Ax.25 existen tres familias de paquetes:

- **Paquete de Información, (I) (Information Frame):** Este paquete es usado para transmitir información de manera numerada, es decir que varios paquetes forman un conjunto de información y estos deben ser numerados para luego ser rearmados y codificar la información ya que al usar una red de comunicación los paquetes no necesariamente se obtendrán en el orden que se requieren para la codificación de la información.

- **<u>Paquete de Supervisión, (S) (Supervision Frame)</u>:** Este tipo de paquete es usado para monitorear el estado de transmisión en la red, como por ejemplo avisar si la red se encuentra libre para recibir o si el buffer del receptor está lleno.

- **<u>Paquete No Numerado, (U) (Unnumbered Frame)</u>:** Este tipo de paquete es usado para hacer solicitudes a la red, como solicitudes de conexión o desconexión. Además existe un sub tipo de paquete que sirve para enviar información como la que es enviada por los tipo I, sin embargo estos paquetes no son numerados y por lo tanto toda la información está contenida en un solo paquete, este tipo se denomina "Paquete No Numerado de Información, (UI) (Unnumbered Information Frame)" y es el que también usa el protocolo APRS.

En la figura 7 se muestra el formato de un paquete UI. A las subdivisiones del paquete de le llamaran *campos*. Los cuatro primeros campos (sin contar el primer campo qué corresponde al *Flag*) son comunes para los tres tipos de paquetes mencionados, y es a través del cuarto campo; el *Control Field,* que el sistema sabe que debe tratar al paquete como uno de los tipos mencionados y es a través de este campo también que el sistema sabe en qué orden debe codificar la información en el caso de los paquetes de tipo I.

En particular el sistema desarrollado solo soporta paquetes de tipo UI por lo tanto no realiza ninguna acción sobre los otros tipos de paquete, esto mismo ocurre con los sistemas APRS pues como ya se ha mencionado APRS solo trabaja con paquetes UI. A continuación se detalla la funcionalidad de cada campo del paquete UI.

	AX.25 UI-FRAME FORMAT								
	Flag	***Destination Address***	***Source Address***	***Digipeater Addresses (0-8)***	***Control Field (UI)***	***Protocol ID***	***INFORMATION FIELD***	***FCS***	***Flag***
Bytes:	1	7	7	0–56	1	1	1–256	2	1

FIGURA 7: PAQUETE UI

2.3.1.2.4.1 Dirección de Destino y Origen

El sistema reconoce en el campo de dirección hasta diez direcciones. La primera corresponde a la dirección de destino, es decir, hacia que estación está dirigido el mensaje; y la segunda corresponde a la dirección de origen desde donde es enviado el mensaje. Las siguientes direcciones corresponden a direcciones de repetición que serán abordadas en el siguiente punto.

Tanto las direcciones de repetición como la dirección de origen son opcionales siendo la de destino la única imprescindible en el protocolo. La dirección de origen no es necesaria pues al utilizar un paquete UI no se está creando una conexión entre las estaciones, sino que solo se están enviando paquetes entre estas.

Una dirección, ya sea de destino, origen o repetición, está compuesta por 7 bytes. Los primeros 6 bytes corresponden al identificador de radio comunicación y se compone por caracteres alfanuméricos y espacios para llenar los seis bytes en caso que no se usen todos. Algunos ejemplos de estos identificadores son WJN3, JK92, NW32E, etc. El séptimo byte se denomina SSID (Secondary Station IDentifier) y corresponde a un número que se les asigna a distintos equipos de una misma estación con un identificador de radiofrecuencia común.

La manera que el sistema tiene de detectar el fin del campo de dirección incluyendo las direcciones de repetición es que todos los bytes terminan con un bit cero excepto el último bit del último byte del campo de dirección que se designa como 1. Para lograr esto con los caracteres de identificador de radio frecuencia en formato ASCII, a los bytes se les inserta un 0 a la derecha y se desplaza el valor de la izquierda que siempre es 0 en los caracteres alfanuméricos ASCII por lo cual no influye para realizar el mismo procedimiento en la decodificación, este procedimiento de desplazar el bit se conoce como *shift* en computación.

Un ejemplo de lo anterior es el siguiente; el carácter ‘N’ en ASCII corresponde al byte 01001110, pero para enviarlo se le debe aplicar un *shift*, por lo tanto el byte a enviar es el 10011100.

La estructura del séptimo byte, el ID, se representa de la forma CRR**SSID**X, donde cada letra representa uno de los ocho bits del byte y su significado es el siguiente:

- El primer bit representado por la letra **C** es el Command/Response bit y sirve para indicar que versión del protocolo se está usando. Si el bit **C** de la dirección de destino es igual al de la dirección de origen significa que se está usando la versión actual, si son distintos se está usando una versión antigua. Este bit no tiene mayor relevancia en el sistema diseñado.

- Los dos bit representados por una **R** son reservados para el usuario, es decir que pueden ser usados para aplicaciones que se estime conveniente.

- Los cuatro bits representados por las letras **SSID** corresponden al Secondary Station IDentifier propiamente tal y se puede asignar valores desde 0 a 15. Si se usan los bits **R** eventualmente se podría extender hasta 63.

- Finalmente el bit **X** indica si el campo de dirección ha llegado a su final con un 1 o si continúa con un 0.

En la figura 8 se muestra un ejemplo de cómo se codificaría un paquete de tipo I sin información. La codificación de la dirección se realiza entre los octetos A1 y A14. En este caso particular el paquete es enviado desde la estación N7LEM hacia la estación NJ7P ambas direcciones con SSID de valor cero.

2.3.1.2.4.2 Dirección de Repetición Digital

Por repetición digital se entiende al proceso de repetir los paquetes digitales a través de las estaciones que soportan el protocolo AX.25, a este proceso se le conoce también como *digipeating* y a las estaciones capaces de repetir estos paquetes se les conoce como *digipeaters*. Así como una estación de destino toma una acción al recibir un paquete que posee su identificador en el campo de la dirección de destino, un *digipeater* toma la acción de repetir un paquete cuando el identificador en el campo de dirección de repetición corresponde a ciertos identificadores genéricos de repetición.

Este campo tiene la función de controlar las repeticiones que se hacen de los paquetes a través de la red y se pueden incluir hasta ocho direcciones de repetición que tienen el mismo formato de las direcciones de origen y destino, es decir, seis

caracteres ASCII alfanuméricos que corresponden al identificador de repetición y un byte de SSID.

Octet	ASCII	Bin Data	Hex Data
Flag		01111110	7E
A1	N	10011100	98
A2	J	10010100	94
A3	7	01101110	6E
A4	P	10100000	A0
A5	space	01000000	40
A6	space	01000000	40
A7	SSID	11100000	E0
A8	N	10011100	98
A9	7	01101110	6E
A10	L	10011000	98
A11	E	10001010	8A
A12	M	10011010	9A
A13	space	01000000	40
A14	SSID	01100001	61
Control	I	00111110	3E
PID	none	11110000	F0
FCS	part1	XXXXXXXX	HH
FCS	part2	XXXXXXXX	HH
Flag		01111110	7E
	Bit position	76543210	

FIGURA 8: EJEMPLO DE CODIFICACIÓN DE DIRECCIÓN

Existen varios identificadores para repetir un paquete a través de *digipeaters*, sin embargo se profundizará en dos: el identificador **WIDE** y **WIDEn-N**.

El primer identificador para repetición que definió el protocolo es el WIDE, éste funciona añadiendo direcciones en este campo con este identificador (en vez de los identificadores de radio frecuencia mencionados anteriormente) tantas veces como el máximo de veces que el paquete debe ser repetido.

El byte SSID posee la misma estructura del byte de las direcciones de origen y destino a excepción del primer bit que correspondía al bit **C** e indicaba la versión usada del protocolo. En el caso de las direcciones de repetición, a este bit se le

denomina **H** (*Have been repeated*) y si su valor es 0 indica que el paquete no ha sido repetido; si es 1 indica que sí ha sido repetido. De esta manera cuando un paquete con identificador WIDE llega a un digipeater, éste inspecciona el bit **H** y si es 0 lo marca en 1 y repite el mensaje. El digipeater debe inspeccionar sucesivamente los identificadores WIDE hasta encontrar uno con el bit **H** en 0; si todos están marcados con 1 significa que el paquete ha sido repetido el máximo de veces definidas.

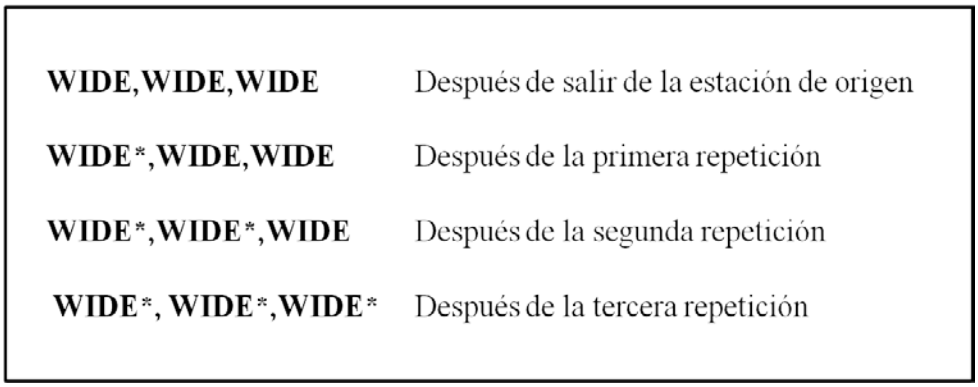

FIGURA 9: REPETICION USANDO WIDE

En la figura 9 se muestra un ejemplo de los identificadores en el campo de dirección de repetición para un máximo de tres repeticiones del paquete. El asterisco en la imagen representa que el bit **H** esta marcado como 1; la ausencia de éste indica que el bit esta marcado como 0.

En la versión más reciente del protocolo se introdujo el identificador WIDEn-N que permite ahorrar en uso de direcciones de repetición, y por lo tanto el paquete enviado es más pequeño; de todos modos el identificador WIDE sigue siendo válido pero menos usado que WIDEn-N.

En esta modalidad de repetición el identificador es de cinco bytes, los cuatro primeros forman la palabra WIDE y el quinto corresponde a un numero en formato ASCII de 1 a 7. Este número representa las veces que debe ser repetido como máximo. Además del identificador, este modo usa el SSID, en el cual se indica las veces que debe ser repetido al igual que con el quinto byte del identificador pero en este caso no es en codificación ASCII sino que en binario usando los cuatro bits reservados para el SSID. Cada vez que el paquete pasa por un digipeater, éste revisa el SSID y si es distinto de 0 le resta 1 al SSID y repite el paquete, de esta manera el SSID se decrementa en uno cada vez que es repetido. Cuando el SSID llega a cero,

el digipeater también pone un 1 en el bit **H** que también es válido para el modo de repetición WIDEn-N.

En la figura 10 se muestra un ejemplo de la dirección de repetición para el caso WIDEn-N. En la figura, el número después del guión indica el valor del SSID y el asterisco indica que el bit **H** tiene valor 1.

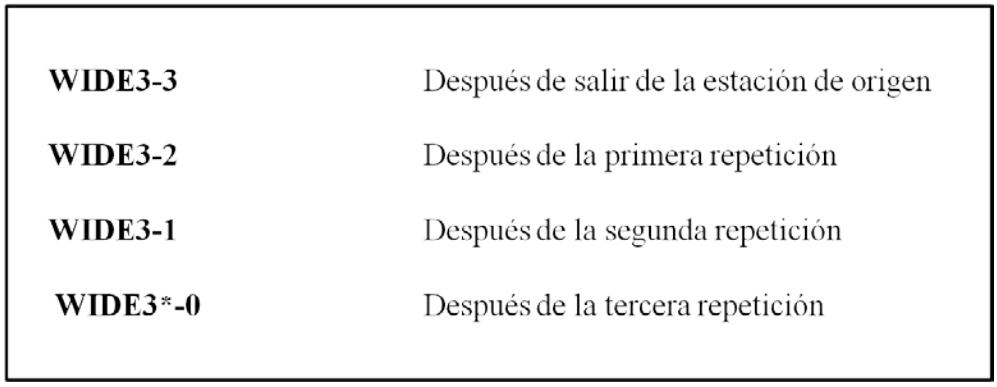

FIGURA 10: REPETICIÓN USANDO WIDEN-N

En el sistema desarrollado el modo usado para repetición corresponde a WIDEn-N y no soporta el modo WIDE. La mayoría de los equipos existentes en el mercado no soportan el modo WIDE siendo los más antiguos los que lo soportan.

2.3.1.2.4.3 Campo de Control e Identificador de Protocolo

En este punto se describe la estructura del cuarto y quinto campo del paquete UI (sin considerar el primer campo del *Flag*). El cuarto campo corresponde al Campo de Control (*Control Field* en la figura 7) y como se ha descrito, sirve para indicar qué tipo de paquete es el que se está recibiendo.

Como el sistema diseñado solo soporta paquetes de tipo UI, al igual que lo hace el protocolo APRS, solo existe un byte que el sistema aceptara y que corresponde a los paquetes UI, este byte es el 000**P**0011, donde el bit **P** indica si se espera respuesta de recibo por parte del receptor. En caso de que se espere respuesta el bit debe tener valor 1 y el receptor debe enviar al emisor un paquete de tipo DM si se está trabajando en modo de desconexión, como lo hace el sistema diseñado, o un paquete

de tipo RNR si se está trabajando en modo de conexión. Los paquetes de tipo DM y RNR son un subtipo de paquetes *Unnumbered* (U) al igual que el subtipo UI con el cual trabaja el sistema diseñado. Sin embargo este sistema no ha sido diseñado para trabajar con petición de respuesta y el protocolo APRS tampoco lo hace, pero es un una modalidad que pudiese ser usada en futuras versiones del sistema.

Como no se espera un acuse de recibo del paquete el único byte de control que acepta el sistema es con el bit **P** igual a 0, es decir el byte **00000011**.

El quinto campo (sin considerar el primer campo del *Flag*) corresponde al identificador de protocolo (Protocol ID en la figura 7) y sirve para avisar cual es el protocolo que se debe usar en la capa superior a la de enlace de red, o sea la capa de red (capa 3). Lo anterior se debe aplicar solo en caso de que realmente se deba usar un protocolo en una capa superior. El protocolo AX.25 define varios identificadores para protocolos de capa tres, por ejemplo para TCP/IP, TEXNET, ARPA y otros.

Si el sistema no usa un protocolo en la capa de nivel tres, como el sistema diseñado y como también lo hace el protocolo APRS, se debe enviar el identificador de "*sin protocolo de capa tres implementado"* que corresponde al byte **11110000**.

En resumen los bytes de control y de identificador de protocolo para este sistema y para el protocolo APRS son:

Control Field	Protocol ID
00000011	11110000

FIGURA 11: CONTROL Y PID

2.3.1.2.4.4 Información

El campo de información es el más grande, permite enviar hasta 256 bytes. Usualmente al contenido de este campo se le denomina *payload* del paquete. En este campo debe ir la información relevante, en el caso del sistema de IDETEC aquí va la posición si es un paquete de reporte de posición o el mensaje en caso de ser un mensaje de texto. La información en este campo, ya sea posición o mensaje, se codifica en formato ASCII.

Los reportes de posición se envían en el formato NMEA que es el formato en el cual el GPS entrega la información, sobre este formato se profundiza más adelante. El protocolo APRS define varios tipos de mensajes para distintas funciones, y estos poseen un identificador determinado dentro del campo de información, esta diferenciación de tipo de mensajes es la funcionalidad más importante que APRS agrega al protocolo AX.25 es por esto que en la sección de APRS se profundizará en la estructura que puede poseer el *payload* del paquete.

2.3.1.2.4.5 Bytes de Validación (FCS)

Los últimos dos bytes corresponden a la secuencia de validación del paquete o FCS (Frame Check Sequence). Estos bytes resultan del un calculo que se realiza a partir de los bits previamente enviados, sin considerar el *Flag*, es decir que en el cálculo se incorporan todas las direcciones, el byte de control, el PID y el *payload*. El cálculo del FCS se hace usando el algoritmo del polinomio CRC-CCIT y es enviado en orden desde el bit menos significativo hasta el bit más significativo.

Existen varios algoritmos CRC-CCIT y en muchos de ellos se calcula el FCS de acuerdo a los bytes del paquete que son enviados, sin embargo el protocolo AX.25 usa un algoritmo CRC-CCIT de 16 bits y donde el cálculo se realiza de acuerdo a cada bit, en vez de cada byte, que es enviado.

El cálculo de los bytes FCS es de la siguiente forma, los dos bytes forman un numero binario de 16 bits, es decir que uno es el byte más significativo y el otro el menos significativo de un mismo número. El algoritmo comienza con el FCS igual al valor en hexadecimal 0xFFFF (o en binario 11111111 11111111), por cada bit de datos enviado se realiza un desplazamiento o *shift*[2] hacia la derecha del FCS (se desplazan los dos bytes en conjunto, por lo tanto el bit menos significativo del byte más significativo pasa a ser el bit más significativo del byte menos significativo), si el bit desplazado, o sea el menos significativo del byte menos significativo, es distinto del bit que fue enviado se realiza la operación “o exclusivo” entre el FCS actual y el valor en hexadecimal 0x8408 (o en binario 10000100 00001000), si los bit son iguales no se realiza ninguna operación adicional al *shift*. Una vez enviados todos los bits, se obtiene el complemento uno del FCS (es decir que los bit de valor 1 toman valor 0 y los de valor 0 toman valor 1) y este valor corresponde al FCS del

[2] El procedimiento de desplazamiento o *shift* se describe en el punto 2.3.1.2.4.1

paquete y es enviado como los últimos dos bytes del paquete. Luego de enviar estos dos bytes se envían un *Flag* para indicar que el paquete ha llegado a su fin.

Es importante mencionar que a diferencia del campo de dirección ni el campo de control, ni el PID, ni el *payload* poseen alguna forma de identificar que el byte corresponde a uno de estos campos, por esta razón el sistema sabe que inmediatamente después de la dirección se encuentran los bytes de control y PID respectivamente y también que los últimos dos bytes corresponden a los bytes de FCS.

2.3.2 APRS

APRS [4] es la sigla para Automatic Position Report System y fue introducido en la conferencia de comunicación digital de TAPR/ARRL de 1992. APRS es una comunicación por paquetes que usa parte del protocolo AX.25, específicamente los paquetes de tipo UI. Este sistema sirve para enviar información en tiempo real a distintos usuarios y es conocido principalmente por combinar sistemas GPS con el Packet Radio (AX.25) permitiendo a usuarios de radio comunicación publicar su posición geográfica a través de un computador en un mapa. Además de la posición, permite reportar información relacionada a la posición como datos meteorológicos, nombre y tipo de la estación y alertas.

2.3.2.1 Ciclo de la Red

Es importante notar que APRS es principalmente una herramienta de comunicación táctica y en tiempo real para ayudar al flujo de información durante eventos especiales como emergencias, alertas aéreas y operaciones de oficinas de emergencia en donde el trafico por la red debiese aumentar, en el caso de el sistema diseñado este aumento puede ocurrir por mayor envío de mensajes de texto o por un aumento en la cantidad de globos en vuelo. Sin embargo estos eventos ocurren de manera muy escasa y la mayor parte del tiempo el sistema estará funcionando de manera rutinaria, aun así se plantea como una filosofía de APRS que el sistema en ningún momento debe ser usado más de lo debido para no sobrecargarlo con datos ante una eventual situación de emergencia.

A raíz de esto el protocolo APRS plantea un ciclo de la red, que se define como el tiempo en que una estación debiese obtener un panorama general de las estaciones que se encuentran dentro de su área de alcance en cuanto a la señal, de este modo las estaciones deben enviar su posición cada un ciclo de red.

Los ciclos de la red se definen de 10 minutos para un sistema donde no exista o exista solo un *digipeater* capaz de repetir el paquete, en el caso del sistema de IDETEC esto corresponde al escenario en que haya uno o dos globo en el aire. Si existen dos *digipeaters* capaz de repetir la información el ciclo de la red debe ser de 20 minutos, este es el caso de tres globos en el aire y finalmente para más de tres *digipeaters* capaces de repetir la señal, es decir cuatro globos o más, el ciclo de la red se define en 30 minutos.

2.3.2.2 Transmisión Redundante de Paquetes

Los paquetes de APRS garantizan estar libres de errores al seguir el protocolo AX.25, específicamente como paquete de tipo UI, pero no garantiza la recepción pues no se trabaja con una red en modo de conexión donde se pide acuse de recibo. Por esta razón APRS realiza una transmisión redundante de paquetes de acuerdo a alguno de los algoritmos que se describen a continuación.

- **Algoritmo de decaimiento:** Se transmite por primera vez un nuevo paquete y *n* segundos después se vuelve a transmitir. Luego se sigue transmitiendo el paquete pero doblando una vez cada nueva transmisión el valor de segundos *n* inicial entre cada envío. Este procedimiento se realiza hasta que el tiempo entre cada paquete alcanza el tiempo del ciclo de la red.

- **Tasa Fija:** Se transmite por primera vez un nuevo paquete y *n* segundos después se vuelve a enviar. Se repite este procedimiento por *x* veces.

APRS introduce el concepto de Time-Out para determinar cuándo una estación debe ser considerada por el sistema como fuera de actividad o fuera de la zona de alcance, de acuerdo al ciclo de la red se considera 2 horas un tiempo suficiente de Time-Out.

2.3.2.3 Identificadores de Dirección Genéricos de APRS

Para que una estación receptora responda ante un determinado paquete, éste debe estar dirigido a la estación correspondiente. Una de las características de los sistemas APRS es que se define una serie de direcciones de destino genéricas ante las cuales cualquier estación que funcione con APRS debiese reconocer. Cada dirección posee un significado distinto que identifica el contenido del paquete. En la figura 12 se muestran todas las direcciones genéricas de APRS.

AIR	ALL	AP	BEACON	CQ	GPS	DF
DGPS	DRILL	DX	ID	JAVA	MAIL	MICE
QST	QTH	RTCM	SKY	SPACE	SPC	SYM
TEL	TEST	TLM	WX	ZIP		

FIGURA 12: DIRECCIONES GENERICAS DE APRS

A pesar de que en teoría una estación APRS debiese responder a todas las direcciones lo usual es que estas respondan solo a algunas dependiendo de la funcionalidad. En el caso del sistema diseñado solo se soportara la el identificador **GPS**.

2.3.2.4 Tipos de Mensajes

En el sistema APRS la información más importante, es decir los datos, se encuentra en el campo de información y existen diez formas o tipos de datos principales que pueden ser enviados en este campo. Los tipos de datos posibles son: *Posición*, *Buscador de Dirección*, *Objetos e Ítems*, *Clima*, *Telemetría*, *Mensajes o Anuncios*, *Solicitudes*, *Respuestas*, *Estatus* y *Otros*. El sistema diseñado solo incorpora dos tipos: *Mensajes o Anuncios* y *Posición*.

Para identificar qué tipo de datos posee el campo de información del paquete, este campo debe poseer un identificador al inicio del campo, a continuación se describe el formato de estos tipos de datos.

2.3.2.4.1 Mensaje de Posición

APRS describe varias formas de codificar la posición en el campo de información sin embargo el sistema diseñado ha sido ideado para funcionar con el formato NMEA pues es este el formato que reporta el GPS. A pesar que éste sea el formato elegido para codificar la posición en el diseño actual, es absolutamente posible que en futuras versiones se puedan implementar otras codificaciones sin tener que intervenir en gran medida el firmware.

El identificador para el formato NMEA es un *símbolo de peso* "**$**" (codificado en ASCII) en el primer byte del campo de información. En la figura 13 se muestra la estructura del campo de información del paquete.

Raw NMEA Position Report Format	
NMEA Received Sentence	
$	..., ..., ..., ..., ..., ..., ...
Bytes: 1	25-209

Examples
$GPGGA,102705,5157.9762,N,00029.3256,W,1,04,2.0,75.7,M,47.6,M,,*62
$GPGLL,2554.459,N,08020.187,W,154027.281,A
$GPRMC,063909,A,3349.4302,N,11700.3721,W,43.022,89.3,291099,13.6,E*52
$GPVTG,318.7,T,,M,35.1,N,65.0,K*69

FIGURA 13: CODIFICACIÓN DE POSICION NMEA EN CAMPO DE INFORMACIÓN

El formato NMEA siempre comienza con el símbolo de peso y sobre este existen varios tipos de mensajes pero siempre llevan después del símbolo de peso los dos caracteres "**GP**". Dependiendo el tipo de mensaje NMEA se ordena de distina forma la latitud, longitud, tiempo y otros valores de interés. Los valores en NMEA siempre están separados por comas y el mensaje finaliza con un asterisco seguido de dos caracteres de verificación. En la parte inferior de la figura 13 se observan cuatro formatos distintos de mensajes NMEA.

2.3.2.4.2 Mensaje de Texto

El protocolo señala que el contenido del mensaje no puede superar los 67 caracteres y este tipo de datos se identifica porque comienza con el carácter *dos puntos* "**:**" seguido de la dirección de quien la envía (*Addressee* en la figura 14), que no tiene que ser necesariamente igual a la dirección de la estación que envía el paquete. A la dirección la sigue otro carácter de *dos puntos* y luego sigue el mensaje que finaliza con el carácter "**{**". Finalmente el campo de información finaliza con un número binario de uno a cinco bits que identifica el mensaje.

	Message Format					
					Message ID	
	:	Addressee	:	Message Text (max 67 chars)	{	Message No xxxxx
Bytes:	1	9	1	0-67	1	1-5

FIGURA 14: MESNAJE DE TEXTO EN EL CAMPO DE INFROMACIÓN

El protocolo además incorpora un mensaje de recepción el cual tiene el mismo formato que el de mensaje de texto pero después del segundo carácter de *dos puntos* posee los tres caracteres "**ACK**" y luego el numero que identifica el mensaje (sin el carácter "**{**") que ha sido recibido. Este mensaje es enviado a la estación de origen en caso de una recepción correcta. Si la recepción fue errada se envía un mensaje con los tres caracteres "**REJ**".

3 DESARROLLO DEL SISTEMA

En este capítulo se describe el diseño del sistema, o específicamente el diseño del Terminal Node Controler (TNC) que corresponde al núcleo del sistema. En primer lugar este capítulo se refiere al hardware seleccionado para trabajar y luego al funcionamiento del firmware diseñado.

3.1 HARDWARE

La unidad física que permite el funcionamiento de una red de comunicación AX.25 es el TNC (Terminal Node Controler) y tiene por función controlar la información en la red; es decir que toma acciones como repetir, responder o procesar los paquetes de determinada manera. El TNC posee un firmware que permite que la red funcione de acuerdo al protocolo AX.25.

Generalmente un TNC está compuesto por un microcontrolador con el firmware correspondiente y un modem que modula la información digital[3] proveniente del microcontrolador, usando la modulación AFSK, transformándola en una señal de audio (también demodula la señal de audio proveniente del transceptor transformándola en información digital). Esta señal de audio posteriormente es transmitida a un transceptor de radiofrecuencia que transforma el audio en una señal de radio usando algún tipo de modulación como FM, para luego ser transmitida a la red.

El diseño del hardware para el sistema se basó en el TNC *PIC-E* [5] de la empresa TAPR (Tucson Amateur Packet Radio) para uso en sistemas de radioaficionados. El *PIC-E* está compuesto por el microcontrolador PIC16F84 de la empresa Microchip y por un modem MX614 [6] de la Empresa *MX-COM, Inc*. Sin embargo en el diseño se ha decidido cambiar el microcontrolador por un PIC16F877A [7], también de la empresa *Microchip*, porque este posee más memoria RAM lo que permite ejecutar programas que procesen más datos, esto pensando en capacidades adicionales que se le pueden añadir al sistema de IDETEC; además de contar con interrupciones necesarias a nivel de hardware que el PIC16F84 no posee.

[3] El microcontrolador envía la información digital en el formato descrito en el punto 2.3.1.2

El hardware, tanto el PIC16F877A como el MX614, funcionan con tensión de 5 V y la comunicación entre ellos es TTL, es decir que 5 V equivale a un nivel alto (H) y 0 V a un nivel bajo (L); la comunicación entre el PIC y el modem usa el protocolo HDLC. El PIC además de comunicarse con el modem debe hacerlo con un periférico que podría ser un dispositivo GPS o un computador según sea la funcionalidad del TNC; esta comunicación con el dispositivo periférico se realiza usando el protocolo RS232. La otra entrada/salida del TNC es la señal de audio que va conectada al transceptor de radiofrecuencia.

En la figura 15 se muestra el diagrama de bloque del TNC con sus respectivos canales de comunicación ya sean internos o externos.

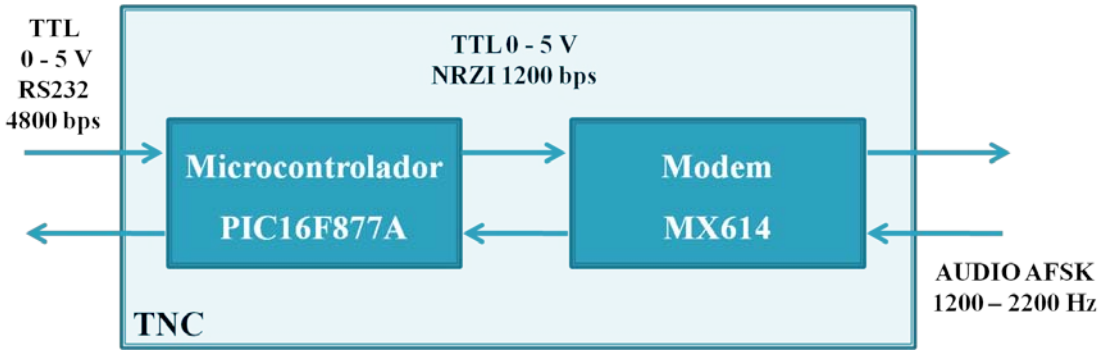

FIGURA 15: DIAGRAMA DE BLOQUES TNC

3.1.1 MODEM MX614

El modem MX614 es un circuito integrado diseñado para la recepción o transmisión asíncrona a una tasa de transferencia de 1200 bps. Este dispositivo es compatible con el conocido modem Bell 202 desarrollado por *Bell System*. Este modem funciona con alimentación de 3.3 V a 5 V (en el diseño se ha optado por 5 V pues el PIC debe funcionar en esta tensión) y necesita un oscilador de 3.58 MHz.

El MX614 posee algunas funcionalidades que no son requeridas en este sistema como la capacidad de recibir paquetes de nueve bits con un bit de partida (como el formato RS232) y almacenarlo en un buffer interno y luego enviarlo a través de una interfaz SPI (Serial Pheripheral Interface) usando un puerto como reloj. Sin embargo el protocolo HDLC no posee bits de partida y tampoco usa paquetes de nueve bits por lo que no es de utilidad esta funcionalidad. La otra funcionalidad no usada en

este diseño es la ecualización de la señal de audio. Conocer las funcionalidades no usadas ayudará a entender porque algunos pines no fueron conectados.

En la figura 16 se muestra el esquemático del circuito integrado MX614 y las conexiones básicas para su funcionamiento, no se muestra la conexión con el resto del hardware pero se explicará cual es el uso de los pines conectados a éste. En color azul se representan los pines usados para la comunicación.

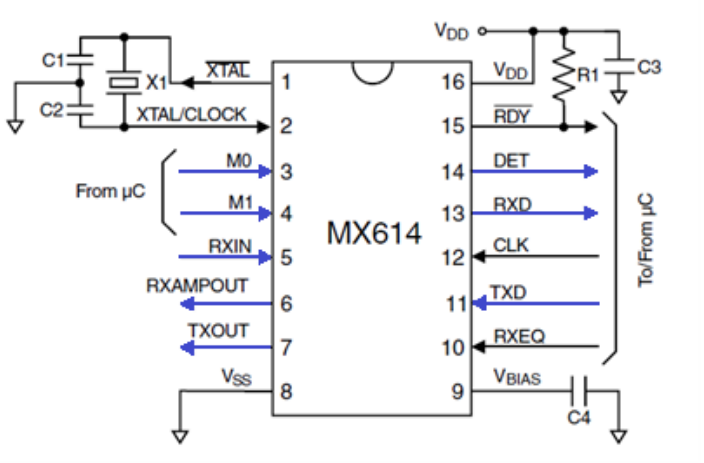

FIGURA 16: MX614

A continuación se describen el uso de los pines del circuito MX614 en el hardware. Los nombres de pines usados en la descripción corresponden a los mismos nombres usados en la figura 16 y que a su vez corresponden a los nombres usados por el fabricante en la hoja de especificaciones del circuito integrado.

- **Pines Básicos**

  - $\boldsymbol{V_{DD}}$ **y** $\boldsymbol{V_{SS}}$**:** Alimentación del circuito integrado.
  - $\boldsymbol{V_{BIAS}}$**:** Polarización interna.
  - $\boldsymbol{XTAL}$ **y** $\boldsymbol{\overline{XTAL}}$**:** Circuito de oscilación.

- **Pines No Usados**

    - $\overline{\boldsymbol{RDY}}$**:** Este pin indica cuando se ha llenado el buffer de recepción de bytes de nueve bits en caso de estar usando esta modalidad. Como se mencionó esta recepción no es compatible con el protocolo HDLC.
    - ***CLK*:** Este pin corresponde al reloj para transmitir la información, en modo SPI, del buffer de recepción de bytes de nueve bits.
    - ***RXEQ*:** Cuando este pin se conecta a un estado de tensión alto (H) se habilita la ecualización, si está en nivel bajo se deshabilita. Al no usar ecualización este pin se conecta a tierra.

- **Pines de Comunicación Usados**

    - ***M*0 y *M*1:** El sistema puede enviar y recibir datos, pero no puede hacerlo al mismo tiempo (comunicación Half-Duplex). Estos pines ponen al modem en estado de recepción o de transmisión. Si M0 está en nivel alto (H) y M1 en nivel bajo (L) el modem esta en modo de transmisión, si M0 está en nivel bajo (L) y M1 está en nivel alto (H) el modem está en modo de recepción.
    - ***RXIN*:** En este pin se recibe la señal de audio del transceptor.
    - ***TXOUT*:** Por este pin se envía la señal de audio hacia el transceptor.
    - ***RXAMPOUT*:** de este pin sale la señal que entra por RXIN pero amplificada. Este pin debe conectarse a través de una resistencia de 100 KΩ al pin RXIN para ajustar la señal a la amplitud correcta para demodulación.
    - ***DET*:** Este pin indica cuando el modem detecta potencia recibida RXIN.
    - ***RXD*:** Este pin envía la señal recibida demodulada en forma digital hacia el microcontrolador.
    - ***TXD* :** Este pin recibe la señal digital para modularla desde el microcontrolador.

3.1.2 MICROCONTROLADOR PIC16F877A

Una de las razones de la elección de este microcontrolador es que su memoria, tanto de datos (memoria SRAM) como de programa (memoria FLASH), es adecuada en cuanto a tamaño para el firmware.

La memoria Flash tiene un tamaño de 16.3 Kbyte y la memoria SRAM un tamaño de 368 Bytes. La memoria SRAM en este sistema juega un rol fundamental pues la recepción de los datos en el sistema es controlada por software, no existiendo un modulo en el hardware que almacene el paquete recibido en un buffer designado para esta tarea, y por lo tanto el almacenamiento de los bytes del paquete debe realizarse en la memoria de datos SRAM. Un paquete UI del protocolo AX.25 puede tener un tamaño máximo de 332 Bytes, lo que prácticamente equivale al tamaño de la memoria SRAM y evidentemente no solo el paquete recibido usa la memoria SRAM sino que esta memoria debe ser compartida con las variables propias del firmware. Por esta razón el tamaño de los paquetes usados en el sistema deben ser de tamaño restringido. A pesar de esta restricción el tamaño de la memoria SRAM del microcontrolador es suficiente para mandar paquetes con la posición y mensajes de texto cortos.

Otra razón para usar este microcontrolador es que posee un modulo USART asíncrono incorporado que permite comunicación RS232. La importancia de este modulo se basa en la posibilidad de generar una interrupción cuando un byte es recibido a través de éste. El modulo USART es usado para la comunicación con los periféricos, o sea con un dispositivo GPS o con un computador. El modulo, al generar una interrupción permite al TNC comunicarse con el dispositivo periférico cuando este dispositivo lo requiera y no solo cuando el TNC lo requiera. Esto permite crear una consola con una interfaz grafica para que un usuario pueda controlar el TNC o enviar mensajes a través de éste.

En la figura 17 se muestra el diagrama del PIC16F877A indicando los pines usados en el sistema y a continuación se describe la funcionalidad de cada pin.

- **Pines Básicos**
    - $\boldsymbol{V_{DD}}$ **y** $\boldsymbol{V_{SS}}$**:** Alimentación del circuito integrado.
    - $\boldsymbol{OSC1}$ **y** $\boldsymbol{OSC2}$**:** Circuito de oscilación.

- **<u>Pines de Comunicación con Periféricos</u>**

  - ***RX* y TX:** Pines usados para comunicación con periféricos usando RS232.

- **<u>Pines de Comunicación con el MX614</u>**

  - ***M0* y *M1*:** Estos dos pines controlan el modo de transmisión o recepción del MX614. Cada pin se conecta al pin del mismo nombre en el MX614.
  - ***PTT*:** Este pin corresponde al *Push To Talk* y sirve para tomar el control del canal de transmisión de la radio. En el Anexo B se muestra el funcionamiento del *Push To Talk*.
  - ***DET*:** Este pin está conectado al pin DET del MX614 que indica cuando se está recibiendo potencia en la entrada de la señal.
  - ***RXD*:** Este pin está conectado al pin RXD del MX614 y por éste se recibe la información digital proveniente del MX614.
  - ***TXD*:** Este pin está conectado al pin TXD del MX614 y por éste se transmite la información hacia el MX614.

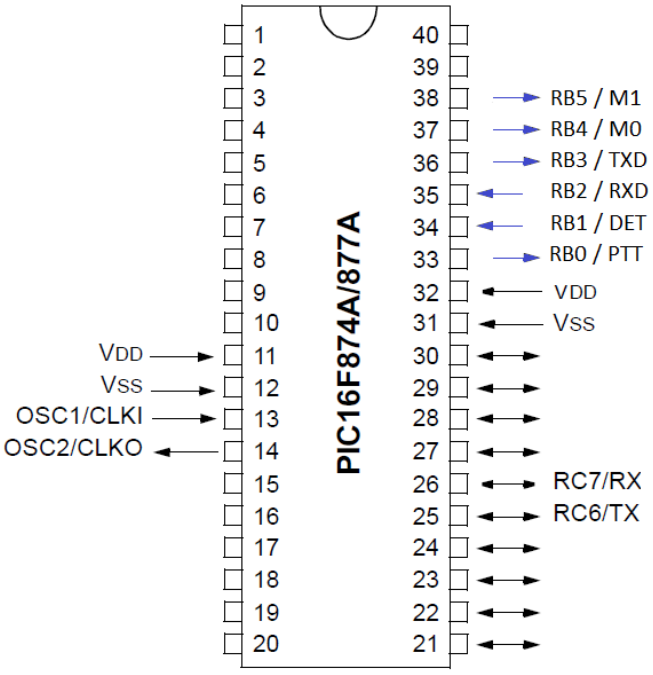

FIGURA 17: PIC16F877A

3.1.3 COMUNICACIÓN CON PERIFÉRICOS

Como se ha mencionado el TNC se comunica con dispositivos periféricos como un GPS o un computador usando el protocolo RS232 a una tasa de transferencia de 4800 bps a través del microcontrolador. Hay que considerar que el microcontrolador se comunica usando una comunicación digital TTL, 0V para nivel bajo (L) (en RS232 el nivel L equivale a un 0) y 5V para nivel alto (H) (en RS232 el nivel alto equivale a un 1), esta comunicación sirve para comunicar directamente el TNC con un GPS pues estos dispositivos también usan comunicación TTL, sin embargo para comunicar un dispositivo con un computador usando RS232 no sirve la comunicación TTL, pues el computador usa para el valor 0 típicamente un voltaje de valor -15V y para el valor 1 un voltaje de valor +15V y por lo tanto es necesario convertir los valores de los voltajes. Para esto se incorpora el circuito integrado ST232 de la empresa *ST Microelectronics* que realiza esta conversión. De acuerdo a esto, si se desea conectar el TNC a un computador se deberá usar el ST232; si en cambio se desea conectar el TNC a un GPS se debe realizar un *bypass* del ST232 y conectarlo directamente al microcontrolador.

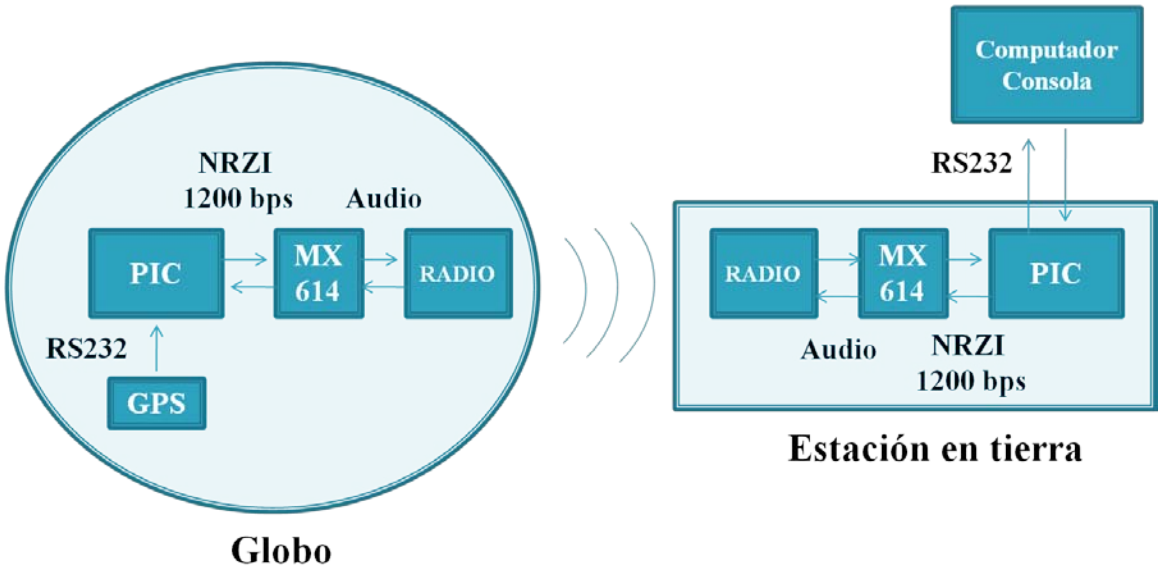

FIGURA 18: DIAGRAMA DE BLOQUES DEL SISTEMA EN OPERACIÓN

La otra conexión con un dispositivo periférico es la conexión con el transceptor, la cual consiste en cuatro líneas: Tierra, RXIN, TXOUT y PTT. Las líneas RXIN y TXOUT son las que salen del MX614 y PTT es la línea Push To Talk y sale de un pin del microcontrolador y es usada para activar el modo de transmisión en el transceptor.

Un mismo TNC puede ser usado ya sea en una estación en tierra como en un globo cambiando su configuración con una consola a través de la comunicación RS232, el detalle de esto se explicará en la sección dedicada al firmware.

En la figura 18 se aprecia el diagrama de bloques del hardware de un sistema de un globo y una estación en tierra.

3.2 FIRMWARE

En esta sección se describe el funcionamiento del firmware del dispositivo diseñado, cuya principal función es recibir los paquetes de AX.25 usando el protocolo HDLC, pues como se ha mencionado el hardware no posee un modulo que realice esta tarea.

El firmware se programó en lenguaje C usando el compilador "*PCM CCS C Compiler v.4"* [7] de la compañía *CCS Inc*. sobre la plataforma de desarrollo de Microchip "*MPLAB IDE v.8.456"*.

Para recibir un byte (8 bits) se ha creado la función recivebyte() encargada de recibir un byte usando el protocolo HDLC manejando las propiedades de éste como: el *bit stuffing*, estado de error y estado ocioso. El sistema no recibe bytes por separado, pues en HDLC no existe un indicador de comienzo y fin de byte, a raíz de esto se debe recibir un paquete completo (compuesto por varios bytes) detectando la llegada de *Flags*. La función recivebyte() detecta los *Flags* y es usada por un proceso de mayor nivel en el firmware encargado de la recepción de un paquete completo. Tanto el proceso de recepción de un byte (función recivebyte()) como el de recepción de un paquete completo se describen en esta sección en los puntos 3.2.1 y 3.2.3 respectivamente.

3.2.1 RECEPCIÓN DE BYTES Y MANEJO DE BIT STUFFING

En este punto se describe el algoritmo usado por la función recivebyte() encargada de la recepción de un byte a la vez del paquete UI del protocolo AX.25. En la figura 18 se muestra el diagrama de flujo de este algoritmo y toda descripción realizada en este punto está referida a la figura.

La función retorna como valor un byte que corresponde al byte recibido, este byte se almacena en una variable interna de la función denominada Var y como se aprecia en el algoritmo de la figura 19 se inicializa con valor 0 (0x0 en notación hexadecimal).

Además de la variable Var, se inicializa también con valor 0 la variable interna K, esta última es usada para contar los bits válidamente recibidos y por lo tanto cuando K tiene valor 8 implica que se ha recibido con éxito un byte.

El segundo objeto en la figura 19 corresponde a un cuadro de decisión donde se revisa si la variable K es menor a 8, si no lo es significa que se ha recibido exitosamente un byte y por ende se retorna la variable Var que lo almacena, si es menor a 8 comienza la detección del siguiente bit. Para obtener el valor del bit recibido se debe analizar si la señal presenta cambio (bit 0) o si no presenta cambio (bit 1) por lo cual es necesario almacenar el estado anterior de la señal, para lo cual existen dos variables globales: Act (almacena el valor actual de la señal) y Prev (almacena el valor anterior de la señal). Estas variables deben ser globales y no internas de la función ya que existe continuidad en la transmisión de bytes y el valor de la señal al final de un byte influye en el valor del primer bit del byte siguiente. Si en el cuadro de decisión K resulta ser menor a 8 se le asigna el valor actual de la señal a la variable Act y luego se espera un periodo[4] [5] para después proseguir con el algoritmo. Al final de un ciclo de detección de un bit se asigna el valor de la variable Act a la variable Prev para que en el siguiente ciclo del algoritmo sea posible realizar la comparación.

[4] Se debe esperar un periodo de 833 uS que corresponde a la tasa de transferencia de 1200 bps y luego se prosigue con el algoritmo pues el periodo de una instrucción del microcontrolador es mucho menor que el periodo de de la tasa de transferencia.

[5] En el proceso de espera de un periodo se aplica el algoritmo de ajuste de fase que se describe en 3.2.2

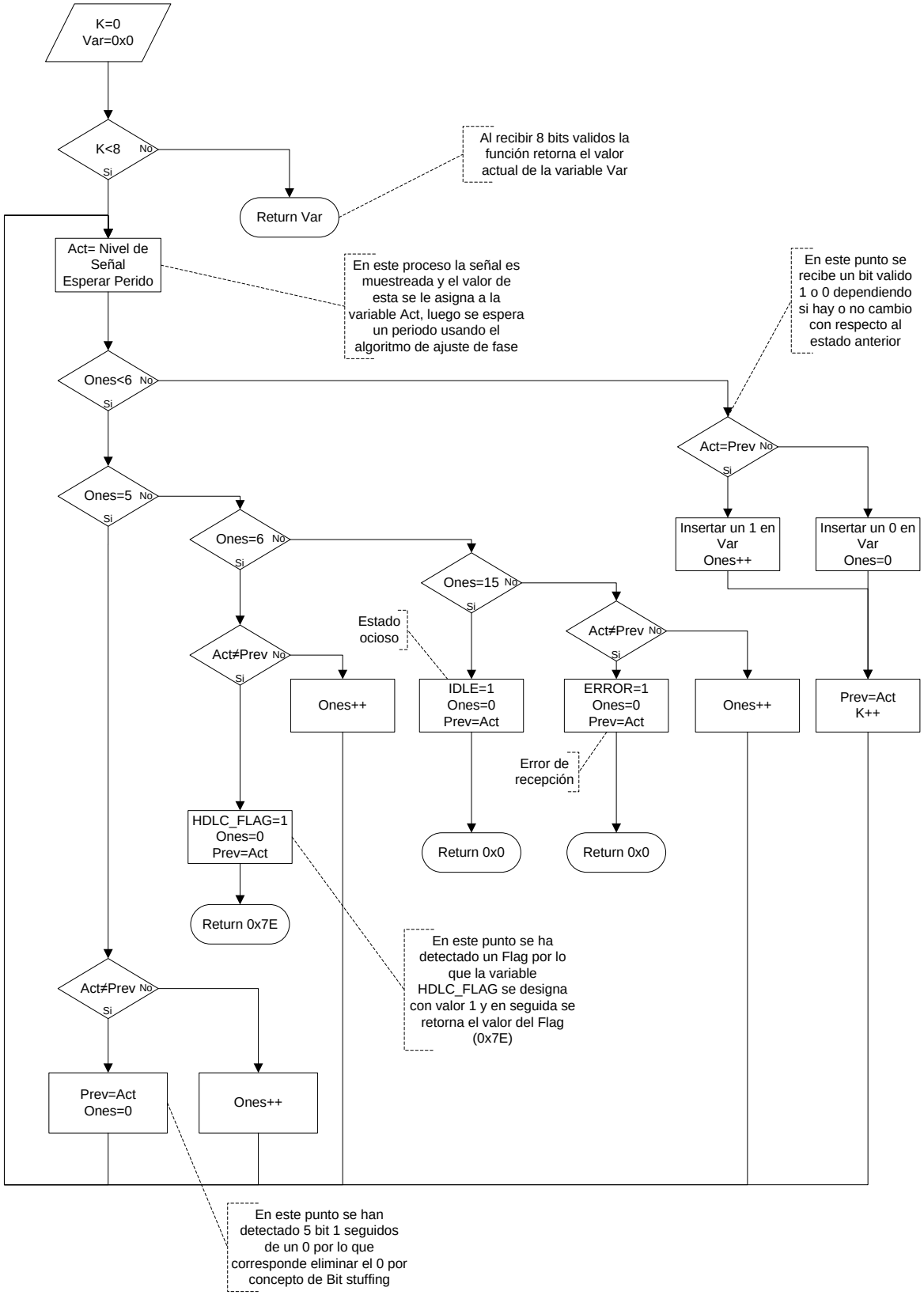

FIGURA 19: ALGORITMO DE RECEPCION DE BYTES

Para manejar el *bit stuffing* es necesario tener una variable, también global, que cuente la cantidad de bits 1 recibidos, esta variable se denomina Ones y cuando tiene valor 5 (es decir que a continuación se espera la llegada de un bit 0 que no debe ser considerado como parte de la información) no se introduce el próximo bit en la variable Var.

No solo para el manejo del *bit stuffing* es útil la variable Ones, durante la recepción de bytes es posible encontrar tres situaciones más aparte de la recepción exitosa de un byte. La primera situación corresponde a la recepción de un byte *Flag*, en donde se envían intencionalmente seis bits 1 seguidos de un bit 0 para indicar el comienzo o término de un paquete, las otras dos corresponden a la detección de un estado ocioso (IDLE) o de un error (ERROR)[6]. Para que la función indique cuando se está en presencia de una de estas tres situaciones el firmware posee tres variables globales denominadas HDLC_FLAG, IDLE y ERROR, asociadas a las tres situaciones mencionados en orden respectivo, las cuales deben ser fijadas en valor 0 antes de cada invocación a la función. Si la función detecta una de estas situaciones fija el valor de la variable respectiva en 1.

3.2.2 AJUSTE DE FASE POR SOFTWARE EN LA RECEPCIÓN

En el punto anterior se ha mencionado que luego de actualizar la variable Act con el valor actual de la señal se debe esperar un periodo para luego continuar con el algoritmo. La razón de esperar este periodo es que la velocidad de procesamiento del microcontrolador es mucho mayor que la tasa de transferencia de bits y por lo tanto el tiempo de procesamiento de la información en el algoritmo es despreciable en comparación al periodo de la tasa de transferencia (833 uS), de esta manera esperar un periodo en este punto del algoritmo permite que la próxima actualización de la variable Act corresponda efectivamente al valor del bit siguiente.

El proceso de esperar un periodo sirve para ajustar la recepción a la tasa de transferencia, pero aun así no es posible asegurar tan solo con este procedimiento que el muestreo de la señal, es decir cuándo se actualiza la variable Act, se realice en el instante correcto y así obtener correctamente la información recibida. La razón de lo anterior es que la comunicación del sistema se realiza de manera asíncrona y por lo tanto puede existir un desfase entre el instante de muestreo y el bit

[6] Tanto la detección de un estado IDLE como de un ERROR se describen en 2.3.1.2.3

correspondiente a la secuencia. Si además se considera que por diversas causas, como comportamiento no ideal de los dispositivos semiconductores y canales de transmisión o el pequeño desfase que introduce el procesamiento de la información en el microcontrolador, la señal puede variar su fase durante la transmisión con respecto al muestreo y es probable obtener una recepción errada de los paquetes. Para evitar esta recepción errada de paquetes se ha complejizado el procedimiento de espera introduciendo un algoritmo que ayuda a sincronizar las fases entre la señal y el muestreo de esta.

En el sistema, para verificar que haya transcurrido un periodo de 833 uS, se usa un Timer que genera una interrupción cuando el periodo se ha cumplido, en este momento se deja de esperar y se continúa con el algoritmo de recepción del byte. Sin embargo se introduce en esta espera otra situación en la que se puede dejar de esperar. Mientras el Timer está funcionando, es decir que aun no ha generado la interrupción, el firmware monitorea constantemente el estado de la señal y lo compara con el estado anterior, si es que detecta un cambio significa que hay un nuevo bit disponible y por lo tanto el algoritmo deja de esperar aun cuando el Timer no haya generado la interrupción, después de este cambio de bit el Timer se apaga y continua el algoritmo de recepción del byte.

En la figura 20 se muestra la señal de entrada y como ésta es muestreada de acuerdo al algoritmo de espera. Se observa que en los dos primeros bits, donde no hay cambio de la señal, pues son bit 1, el muestreo se realiza cerca del final de estos bits, es decir que existe un desfase entre la señal y el muestreo, y la actualización de la variable Act se realiza en estos casos porque el Timer genera una interrupción. Los últimos dos bits son bit 0 y por lo tanto presentan cambio en la señal, en estos casos no se alcanza a producir una interrupción pues el firmware detecta el cambio en la señal antes y actualiza en ese momento la variable Act. Como se observa en los últimos dos casos el muestreo se produce al comienzo del bit, a diferencia de los dos primeros, y por lo tanto a partir de cada cambio se produce una sincronización de la fase que hay entre la señal y el muestreo. De esta manera se evitan los errores en la recepción asociados a la asincronía de la transmisión.

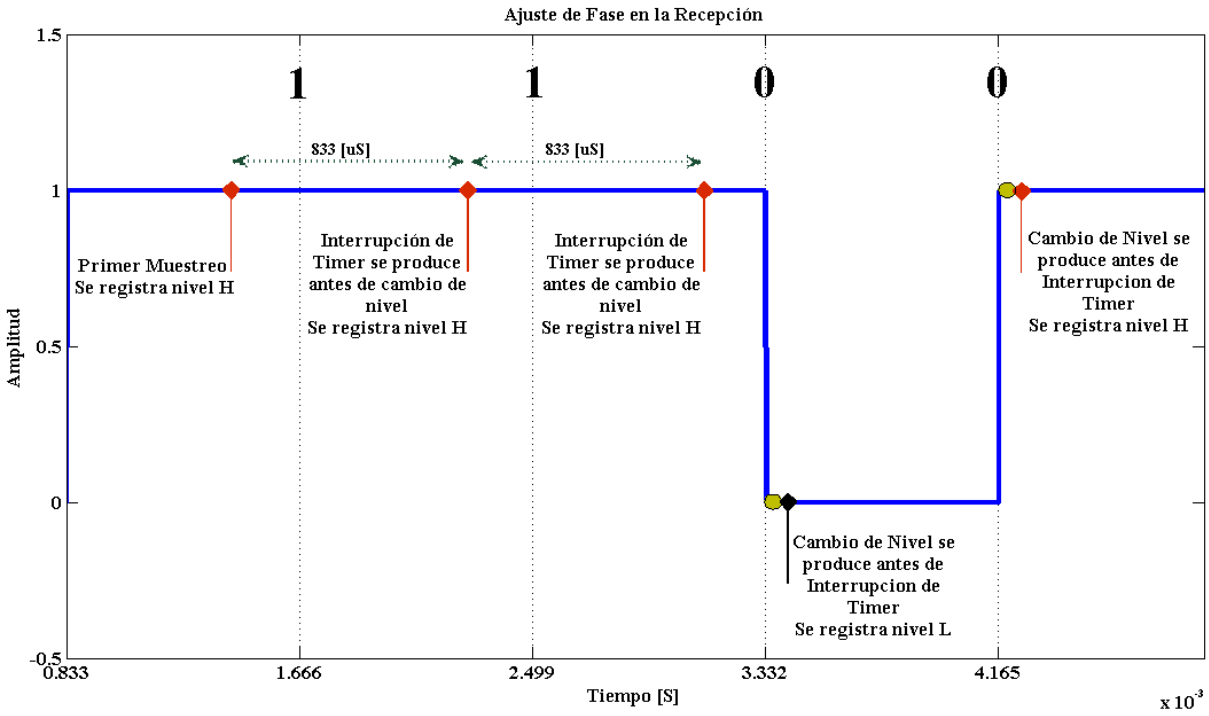

FIGURA 20: AJUSTE DE FASE POR SOFTWARE EN LA RECEPCION DE BYTES

3.2.3 RECEPCIÓN DE PAQUETES Y DETECCIÓN DE FLAGS

Hasta ahora se ha descrito la recepción de bytes, pero para poder decodificar la información en el protocolo AX.25 es necesario recibir un paquete completo. Un paquete comienza y termina con un *Flag*, pero cabe señalar que en la transmisión no se envía solo un *Flag* al comienzo del paquete sino que se envía un tren de *Flags* (varios *Flags* seguidos) para asegurar que el sistema detecte el comienzo de un paquete y lo mismo ocurre al finalizar un paquete.

A continuación se describe el algoritmo usado para recibir un paquete completo, el cual hace uso de la función recivebyte() descrita en puntos anteriores y de las variables HDLC_FLAG, IDLE y ERROR que indican el estado de la recepción de bytes. Este algoritmo es parte del ciclo principal que ejecuta el firmware y que como

tal está incluido dentro de la función main()[7] que es la encargada de ejecutar el resto de las funciones.

Para recibir un paquete se crea un arreglo de caracteres en la memoria RAM llamado Buffer donde se almacenan todos los bytes del paquete en el orden respectivo.

En la figura 21 se muestra el diagrama de flujo de este algoritmo y la descripción realizada en este punto se hace en función de este diagrama. Antes de entrar en el primer ciclo del algoritmo se verifica si la variable Det (que posee el valor del puerto DET en el modem)[8] es 1, lo cual implica que el modem está recibiendo potencia, de ser así se fijan en 0 todas las variables indicadoras del estado de recepción de un byte (HDLC_FLAG, IDLE y ERROR) usando la función denominada como resetflags() y a continuación comienza el primer ciclo del algoritmo que recibe el primer *Flag* del tren de *Flags* que llegará al TNC. Cuando esto ocurre la variable HDLC_FLAG toma valor 1 y se quiebra el primer ciclo.

El segundo ciclo corresponde a la espera del primer byte de información, antes de entrar en este ciclo se verifica que no se haya detectado un error o un estado ocioso (ERROR=0 y IDLE=0) en la recepción del último byte y luego, previo al proceso de recepción (receivebyte()), se usa nuevamente la función resetflags().

La condición para mantenerse en el ciclo es que se reciban solamente *Flags*, y en el momento en que se reciba un byte que no es *Flag*, es decir el primer byte con información, se pasa al siguiente ciclo cuya función es recibir todos los bytes con información.

En el tercer y último ciclo se reciben todos los bytes entrantes hasta que se detecte la recepción de un *Flag* o de un estado de error o estado ocioso. En cada recepción de bytes se incremente la variable j para así almacenar cada byte recibido en el arreglo Buffer por orden de recepción.

Una vez finalizado el algoritmo se prosigue con el ciclo principal de la función principal para luego validar y decodificar el paquete recibido. Cuando el algoritmo de recepción de paquetes finaliza se tendrá en el arreglo Buffer el paquete recibido y es importante destacar que el valor de la variable j será usado a continuación para la decodificación del paquete pues indica el tamaño del paquete recibido.

---

[7] En lenguaje C la función que se ejecuta siempre al iniciar un programa es la función main() y debe crearse en todo programa ejecutable como es el caso del firmware del TNC.

[8] En el punto 2.1.1 se describe la funcionalidad del Pin DET en el hardware.

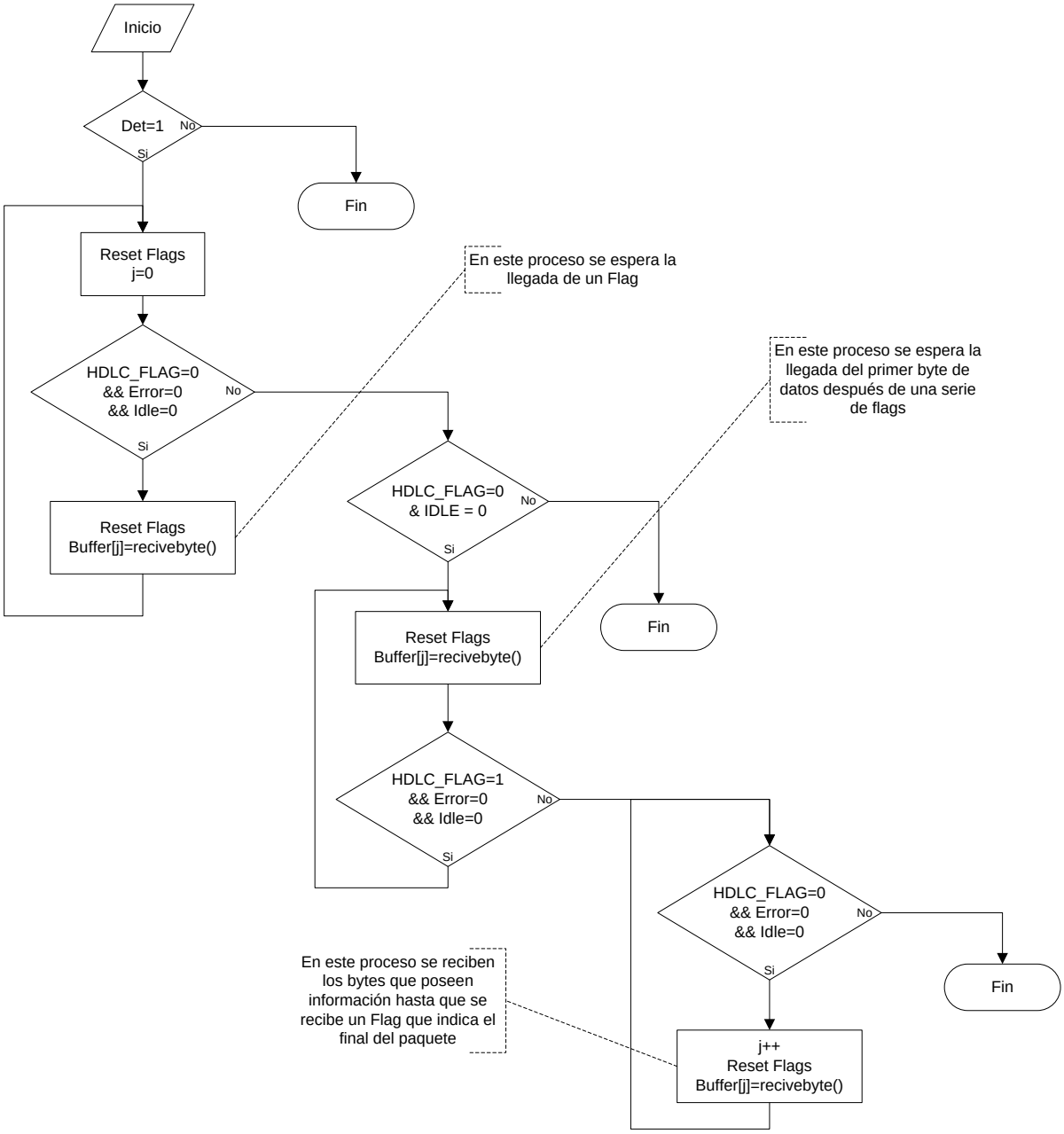

FIGURA 21: ALGORITMO DE RECEPCIÓN DE PAQUETES[9]

[9] El símbolo **&&** en la figura indica un "y lógico" en el algoritmo.

3.2.4 VALIDACIÓN Y DECODIFICACIÓN DEL PAQUETE

Al igual que el algoritmo de recepción de paquetes, el algoritmo de validación y decodificación de paquetes está incluido en el ciclo principal del programa en la función `main()` y se ejecuta inmediatamente después del término del algoritmo de recepción de paquetes. En la figura 22 se aprecia el diagrama de flujo del algoritmo.

Al comenzar el algoritmo se verifica que en la recepción del último byte en el proceso anterior no se haya detectado un estado ocioso o un error, si no se ha detectado ni uno de estos dos casos se prosigue con la validación. La validación corresponde a verificar si los bytes de FCS recibidos son correctos o no, para esto se ha creado la función `CrcPacket()` que calcula a partir del paquete recibido los bytes del FCS y los guarda en las variables CRCHI y CRCLO[10] (cada variable es de 8 bits y ambas forman el numero CRC de 16 bit). Luego se usa la función `Rcv_crc()` que extrae los bytes FCS recibidos en el paquete y los guarda en las variables CRCHI_RCV y CRCLO_RCV. El paso siguiente en el algoritmo es un cuadro de decisión donde se verifica que los bytes FCS recibidos y calculados coincidan, lo cual quiere decir que se ha recibido correctamente un paquete.

Si se ha validado el paquete el algoritmo lo decodifica, en este proceso se debe revisar la dirección de destino del paquete para saber a quién está dirigido. Como se ha mencionado en el capítulo 3 el sistema soporta la dirección genérica de APRS para la posición, es decir los caracteres GPS, además de esta dirección el firmware debe aceptar los paquetes que están dirigidos al TNC en cuestión, para otros casos el sistema debe decidir si corresponde o no repetir[11] el paquete.

Para la decodificación se han creado tres funciones, primero `getAddress()` que revisa el paquete para saber en hasta que índice del arreglo Buffer llega el campo de dirección[12], la segunda función es `getDest()` que extrae la dirección de destino de este campo y finalmente la función `DecoDest()` que decodifica la dirección de destino indicando si esta corresponde a la dirección del TNC, si es la dirección genérica GPS o si no es ninguna de estas dos opciones y se debe entrar al algoritmo de repetición.

---

[10] El cálculo de estos bytes se realiza de acuerdo al procedimiento descrito en el punto 2.3.1.2.4.5
[11] La repetición se realiza de acuerdo al procedimiento especificado en el punto 2.3.1.2.4.2.
[12] Esta detección se realiza de acuerdo a lo descrito en el punto 2.3.1.2.4.1.

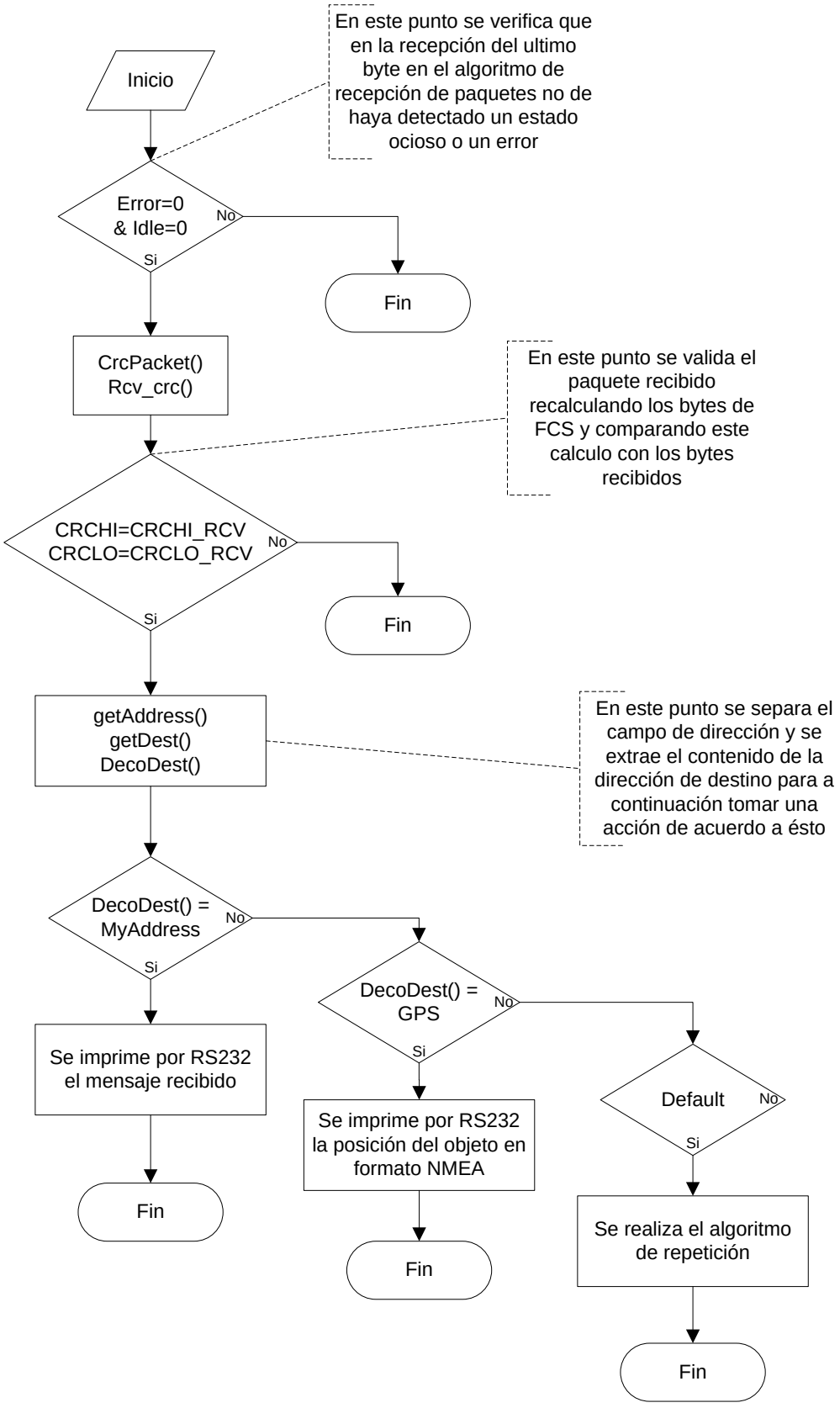

FIGURA 22: ALGORITMO DE VALIDACIÓN Y DECODIFICACIÓN

3.2.5 ALGORITMO DE REPETICIÓN

En caso de que al decodificar la dirección de destino, ésta no corresponda a la dirección del TNC ni a la dirección genérica GPS, el firmware entra en el algoritmo de repetición como se muestra en la figura 21. Este algoritmo revisa las direcciones con su respectivo SSID, desde la dirección más alejada hasta la más cercana a la dirección de destino (que es la primera dirección del campo de dirección)[13]. Cuando el identificador de alguna dirección corresponde a WIDEn y su SSID es mayor a cero el algoritmo le resta una unidad a este SSID y envía el paquete nuevamente a la red a través del modem. Si no encuentra ninguna dirección con esta condición significa que el paquete no debe ser repetido y por lo tanto no se realiza ninguna acción y el paquete se pierde en este TNC. La actualización del SSID se realiza en el mismo arreglo Buffer pues para el envío del paquete recibido y modificado se usa este mismo arreglo.

3.2.6 ENVÍO DE BYTES

Hasta el momento solo se ha comentado sobre la recepción de bytes pero no sobre el envío de los bytes. En este punto se describe el algoritmo usado para el envío de bytes que utiliza la función denominada sendbyte(). En la figura 23 se muestra el diagrama de flujo de este algoritmo y la descripción realizada en este punto se hace en base a este diagrama.

En primera instancia se inicializa la variable Tx_Byte con el byte que debe ser transmitido y la variable K con valor igual a 0 y que lleva la cuenta de cuantos bits han sido transmitidos. Además se crean dos variables globales necesarias, la primera denominada Stuff que lleva la cuenta de los bit 1 transmitidos para manejar el *bit stuffing* y es análoga a la variable Ones usada en el proceso de recepción, la segunda variable se denomina flag y es usada para que la función identifique si está transmitiendo un *Flag* u otro tipo de byte pues en el caso de los *Flags* no se aplica el *bit stuffing*.

[13]En el punto 2.3.1.2.4 se muestra la estructura del paquete y el orden en el campo de direcciones.

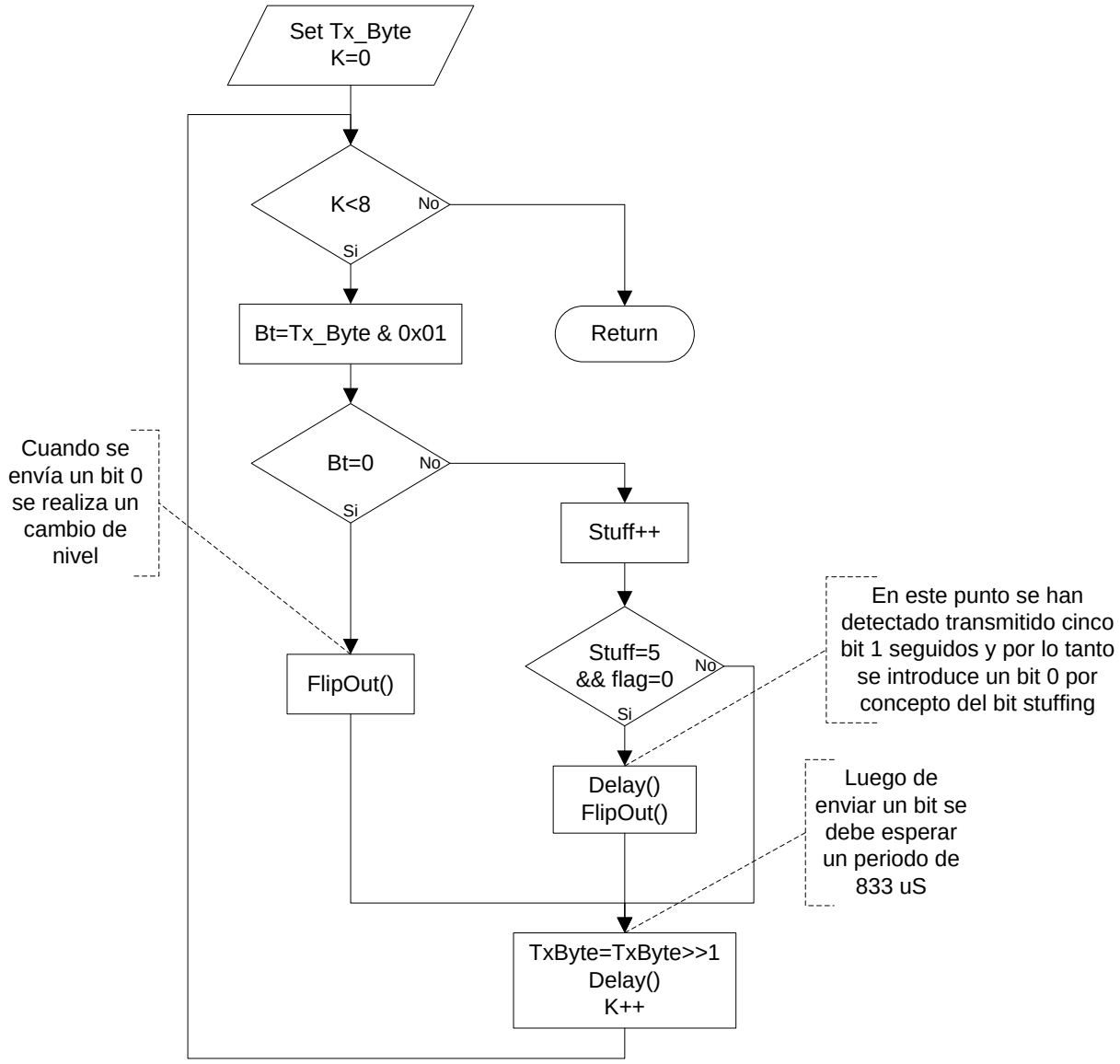

FIGURA 23: ALGORITMO DE ENVÍO DE BYTES[14]

En el primer cuadro de decisión se revisa si se han enviado los ocho bits, de no ser así se continúa con la transmisión y en el siguiente proceso se realiza una operación de *masking*, es decir que se extrae el ultimo bit de la variable Tx_Byte realizando una operación "y" bit a bit con el byte 0x01 (en hexadecimal).

[14] El símbolo **&** en la figura indica un "y bit a bit" en el algoritmo.

En el siguiente cuadro de decisión se revisa si este bit extraído corresponde a un bit 1 o a un bit 0, si es un bit 0 se aplica la función flipout() la cual cambia el valor de la señal en la salida hacia el modem y luego la variable Stuff se iguala a 0. Si el bit corresponde a un bit 1 se incrementa en uno la variable Stuff y luego se revisa la cantidad de bits 1 enviados para, de acuerdo a esto, insertar o no un bit 0 por concepto de *bit stuffing*. Después de cada envió de un bit se esperan 833uS usando la función Delay(), incluyendo a los bit 0 insertados por el *bit stuffing*; y después de cada bit de datos, es decir que se excluyen los bit 0 del *bit stuffing*, se incrementa en uno el valor de la variable k.

3.3 OPERACIÓN DEL SISTEMA

Como se ha mencionado un TNC se puede comunicar con periféricos, ya sea un computador o con un dispositivo GPS. Esta comunicación permite dos funcionalidades, primero controlar ciertos parámetros del TNC que se explicarán a continuación y segundo, permite enviar mensajes de texto y de posición a través del TNC.

Cuando el TNC se conecta a un computador se hace a través de una consola que simplemente envía lo escrito en ella por el usuario al dispositivo e imprime en pantalla lo enviado por el dispositivo. De esta manera el computador no realiza ningún procesamiento de información sino que solo muestra y envía la información.

3.3.1 PARÁMETROS DE UN TNC

Un TNC del sistema de IDETEC puede funcionar como estación en tierra o como globo sonda, la diferencia entre estas dos modalidades es que cuando el sistema funciona como estación en tierra se debe considerar una interfaz adecuada para que un usuario humano pueda interactuar con el sistema, en cambio cuando funciona como globo no tiene sentido escribir por el puerto serial lo recibido pues no habrá un usuario que lo interprete. Para controlar la modalidad de estación en tierra o globo del sistema el firmware posee la variable GLOBO. Cuando esta variable tiene valor 0 el sistema funciona como estación en tierra y cuando el valor es distinto de 0 funciona como globo.

El otro parámetro importante de un TNC es su identificador o dirección, para esto se ha creado el arreglo MyName que almacena el identificador del TNC incluyendo su SSID en el último índice. Este arreglo es necesario para realizar las comparaciones correspondientes para la decodificación del paquete y así detectar si la dirección de destino corresponde a la dirección del TNC.

3.3.2 CONTROL Y ENVIÓ DE DATOS A TRAVÉS DE PERIFÉRICOS

El PIC16F877A posee un modulo serial controlado por hardware y existe un registro que indica si el microcontrolador ha recibido información a través del modulo. Haciendo uso de este registro el TNC dejará de esperar la llegada de información del modem solo cuando se haya detectado información en el modulo serial, de ser así el firmware entrará en el proceso de recibir y decodificar la información que entra por el puerto serial.

La información recibida por el modulo serial es almacenada en el arreglo Buffer, el mismo en el que se almacena la información recibida del modem, para luego ser decodificada y formateada en el formato correspondiente y ser enviada de acuerdo al protocolo AX.25 si es que corresponde.

Hay tres tipos de comandos que se pueden recibir a través del modulo serial que soporta el TNC, éstos son los comandos de control, de datos y de posición que se describen a continuación.

- **Comando de Control:** Este comando sirve para cambiar los parámetros del TNC y debe comenzar con la palabra CONTROL: seguida de la instrucción, que puede ser de dos tipos. La primera es MODE(X) para cambiar el modo de funcionamiento del globo donde X es el valor que toma la variable GLOBO, por lo tanto si se usa X con valor 0 el TNC funcionará como estación en tierra y con otro valor funcionará como globo. El segundo tipo de instrucción es NAME(NWUSR,X) y sirve para cambiar el identificador de la estación donde NWUSR corresponde al nuevo nombre de la estación y X corresponde al SSID.

- **Comando de Datos:** Este comando sirve para enviar mensajes a través de los globos con una dirección de destino y una de origen. El comando comienza con la palabra DATA: y es seguido de la siguiente información en el orden respectivo y separada por comas: Identificador de Destino, SSID de destino, Identificador de origen, SSID de origen, repeticiones máximas y luego el mensaje.

- **Comando de Posición:** Este comando está pensado para conectar un GPS como periférico que envíe una sentencia en formato NMEA, es decir que reconoce los comando que comienzan con $GP y formatea el mensaje de acuerdo a la dirección genérica de APRS GPS

En la figura 24 se muestran ejemplos para cada tipo de comando que puede ser recibido a través del modulo serial del TNC.

Comando de Control
CONTROL: MODE(1)
CONTROL: NAME(IDETEC,4)
Comando de Datos
DATA: DEST,3,ORIGEN,2,3,:DEST: ESTE ES UN MENSAJE DE PRUEBA
DATA: IDETEC,0,IDETEC,3,2,:IDETEC: ESTACION 3 EN TIERRA 040511
Comando de Posición
$GPGGA,006000.000,5009.3540,N,00540.9440,W,1,07,1.25,00121,M,047,M,,*4E
$GPRMC,010003.000,A,5009.3504,N,00540.9278,W,25139.56,104.759,0.00,E,*72

FIGURA 24: EJEMPLOS DE COMANDOS SERIALES

La función usada para monitorear el registro que indica si es que existe información en el puerto serial se llama kbhit() y retorna un 1 si es que hay información y un 0 si es que no la hay. Cada vez que se inicia un nuevo ciclo principal de la función main() se revisa el valor que entrega esta función y si es 1 el algoritmo recibe y decodifica la información a través de modulo serial.

La función usada para recibir la información completa a través de puerto serial se llama getSerial(), cuando esta información se recibe se verifica a que comando corresponde, si es un comando de datos se formatea como tal usando la función formatData() y luego se envía con la función sendpacket() que envía todo un paquete haciendo uso de la función antes descrita sendbyte(). Si el comando es de posición se formatea usando la función formatGPS() en el formato de posición y luego se envía. Si la función es de control se cambian los parámetros correspondientes del TNC. La figura 25 muestra el diagrama de flujo de este algoritmo.

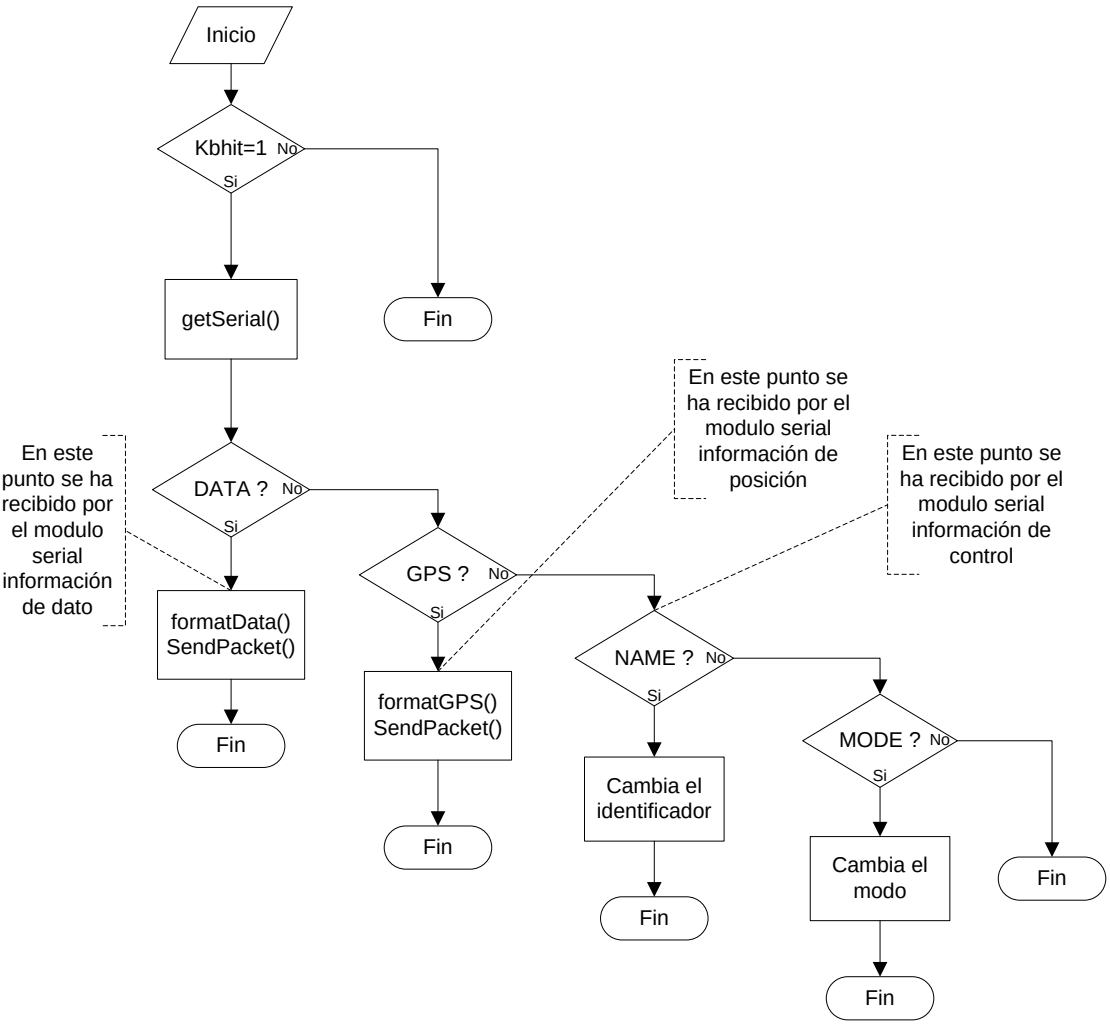

FIGURA 25: ALGORITMO DE RECEPCIÓN Y DECODIFICACIÓ DE INFORMACION DE PERIFERICOS

3.3.3 FUNCIÓN MAIN Y CICLO PRINCIPAL

El firmware posee una función principal llamada main() que es ejecutada antes que cualquiera, siendo esta función la que invoca a las demás funciones. Dentro de la función main() existe un ciclo principal que se ejecuta permanentemente y permite que el sistema funcione. Este ciclo está compuesto por los tres algoritmos principales descritos en los puntos 3.3.2, 3.2.3 y 3.2.4. El flujo de la ejecución de estos algoritmos se muestra en la figura 25.

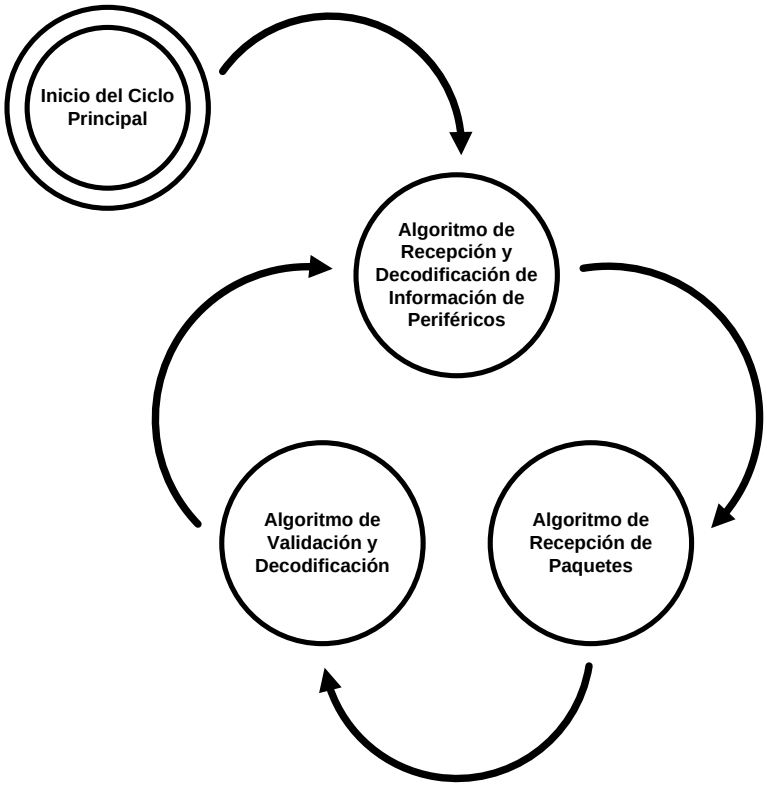

FIGURA 26: DIAGRAMA DE FLUJO DEL CICLO PRINCIPAL

4 ANÁLISIS

En este capítulo se describen las pruebas realizadas al diseño detallado en el capítulo anterior y el análisis de estos resultados.

4.1 PRUEBAS DEL SISTEMA

4.1.1 PRUEBA DE FIRMWARE EN SISTEMA REDUCIDO

Para verificar el buen funcionamiento del firmware se realizaron en primera instancia pruebas en un sistema reducido, sin incorporar la modulación AFSK. Para esto se eliminó el circuito integrado MX614 y los dispositivos de radio, conectando directamente los microcontroladores entre sus puertos de entrada y salida, de esta manera la comunicación se realiza directamente usando el protocolo HDLC a 1200bps y con codificación NRZI sin transformar la información en frecuencia. En la figura 27 se muestra el esquema de este sistema reducido.

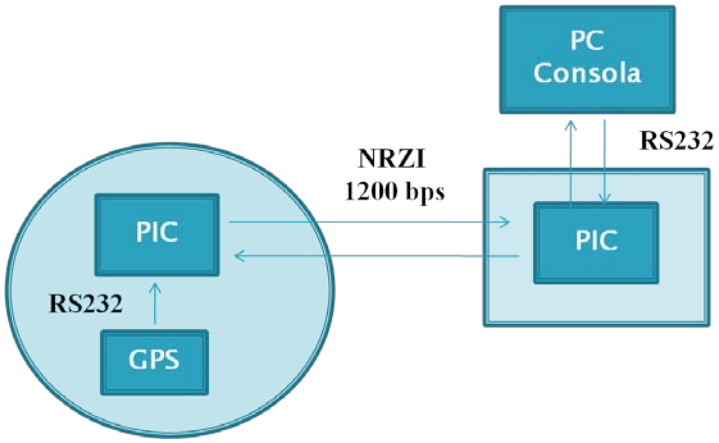

FIGURA 27: DIAGRAMA DE SISTEMA REDUCIDO USADO PARA PRUEBAS DE FIRMWARE

El objetivo de esta prueba fue verificar la correcta sincronización en la recepción de los paquetes y también la correcta decodificación de éstos.

Para el primer objetivo, la verificación de la sincronización, se enviaron 30 paquetes de 90 bytes de largo de un microcontrolador a otro esperando que estos fueran recibidos exitosamente[15] en el microcontrolador receptor. Al utilizar el algoritmo de ajuste de fase descrito en el punto 3.2.2 la recepción exitosa de paquetes fue del 100% de los casos que se probó el sistema, mientras que al eliminar este algoritmo del firmware, es decir, usando una sincronización sin ajuste de fase, la recepción exitosa de paquetes nunca supero el 50% lo cual muestra una mejora significativa a causa de este algoritmo en la recepción.

Para verificar la correcta decodificación se inicializó un microcontrolador como *globo sonda* de acuerdo a lo descrito en el punto 3.3.2, conectándolo a un puerto serial que enviaba información simulando ser un GPS en movimiento. El otro microcontrolador fue inicializado como *estación en tierra*. El dispositivo que funcionaba como *globo sonda* enviaba su posición cada 10 segundos al dispositivo que funcionaba como estación en tierra, este último fue conectado a una consola que enviaba dos mensajes distintos cada 5 segundos de manera intercalada al dispositivo *globo sonda*, uno de estos mensajes estaba destinado al mismo dispositivo *estación en tierra* que lo emitía, por lo que se esperaría que el *globo sonda* lo recibiese y lo repitiera generando un mensaje de eco. El segundo mensaje tiene como destino el *globo sonda* y por lo tanto debía ser desplegado en la consola que simulaba el GPS. Un 89% de los mensajes eco fueron recibidos correctamente. De los mensajes destinados al dispositivo *globo sonda* un 33% fueron recibidos correctamente y de los mensajes de posición un 20% fueron recibidos correctamente.

La razón de que un que un porcentaje importante de mensajes se perdieron se debe a que el sistema no puede enviar y recibir a la vez y por esta razón muchos mensajes llegan al dispositivo receptor cuando este se encuentra enviando un mensaje. A pesar de esto se debe considerar que la carga usada en cuanto a frecuencia de envío de mensajes es sumamente alta en comparación a lo que se debiese usar en un sistema APRS, donde se propone un tiempo no menor a 10 minutos para enviar actualizaciones de posición.

El resultado de estas pruebas indica que el firmware funciona correctamente en estas condiciones y de acuerdo a los requerimientos planteados.

[15] La recepción exitosa se define de acuerdo a lo descritos en los puntos 2.3.1.2.4.5 y 3.2.4

4.1.2 PRUEBA DE MODULACIÓN Y DEMODULACIÓN CON EL MX614

Al realizar las pruebas del sistema completo, que se indica en la figura 18, no se obtuvieron resultados exitosos como sí ocurrió con las pruebas descritas en el punto 5.1.1. No fue posible recibir con éxito los paquetes debido a desajustes en los tiempos de los pulsos de la señal demodulada por el circuito integrado MX614. A continuación se describen las pruebas realizadas y los resultados obtenidos de éstas.

Al no obtener los resultados esperados se realizaron pruebas en las que los paquetes enviados se redujeron a solo dos bytes de datos más los *flags* que inicializan y finalizan los paquetes. Esta prueba se realizó para poder estudiar bit a bit la transmisión y recepción y así poder acotar el problema. A continuación se describe el análisis realizado para un paquete de prueba que contiene los bytes en codificación ASCII “D” y “A”.

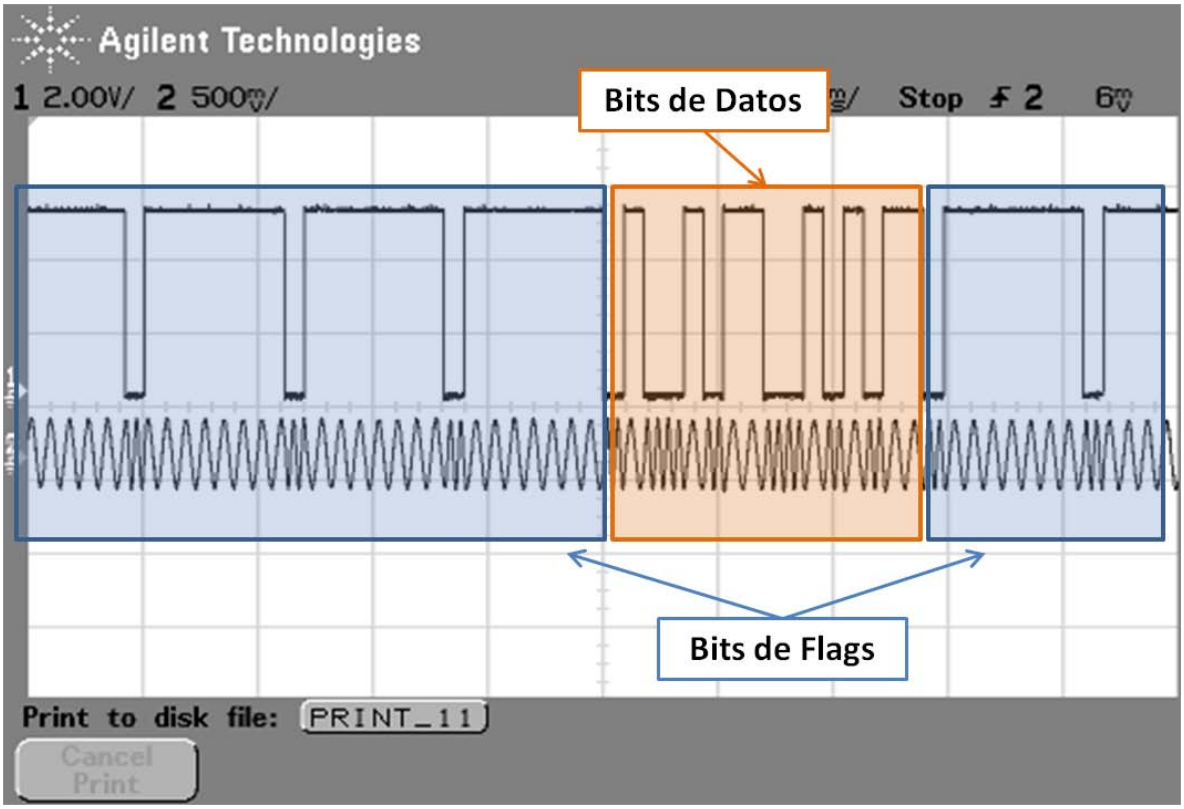

FIGURA 28: MODULACIÓN DE PAQUETE CON LOS BYTES ASCII "D” Y “A"

En la figura 28 se muestra la señal modulada correspondiente al paquete de valores ASCII “D” y “A” obtenida con un osciloscopio. Se observa como este paquete comienza con un tren de flags, continua con la información y finaliza con otro tren de flags. La señal digital que se encuentra en la parte superior de la imagen corresponde a aquella que sale desde el microcontrolador hacia el circuito MX614 y la análoga de la parte inferior corresponde a la señal modulada que sale del MX614 hacia el transmisor de radio.

En el extremo receptor, en vez de recibir el paquete esperado siempre se recibieron paquetes incorrectos. Sin embargo se detecto que en estos paquetes errados persistentemente ocurría la misma falla. Siempre en algún punto del paquete se insertaban bits 1, nunca ocurrió que se omitiera algún bit o que se agregara un bit 0 donde no correspondiera.

En la figura 29 se muestra como debe ser la decodificación correcta del paquete enviado y en la figura 30 se muestra el caso de recepción incorrecta más común encontrado durante las pruebas donde en vez de recibir los dos bytes “D” y “A” se reciben tres bytes donde el primer byte era correcto, correspondiendo al byte “D”; sin embargo el segundo byte recibido es incorrecto. Esto se debe a que dos bit 1 se insertaron en la recepción del segundo byte, entregando el byte ASCII “%” en vez de “A”. Además se recibe un tercer byte que es producto de dos bits desplazados del byte “A” a raíz de lo recién mencionado y el comienzo del primer *flag* que indica el fin del paquete. En la figura 29 se muestran resaltados en color los bit 1 que se introdujeron en el paquete y en cursiva los bit correspondientes al primer flag que indica el fin del paquete.

	Primer Byte	Segundo Byte
Caracter ASCII	**D**	**A**
Hexadecimal	0x44	0x41
Binario	0100 0100	0100 0001

FIGURA 29: DECODIFICACIÓN CORRECTA DE LOS BYTES ASCII "D" Y "A"

	Primer Byte	Segundo Byte	Tercer Byte
Caracter ASCII	**D**	**%**	
Hexadecimal	0x44	0x25	
Binario	0100 0100	0010 0101	*1111 1001*

FIGURA 30: DECODIFICACION INCORRECTA DE LOS BYTES ASCII "D" Y "A"

El motivo por el cual la recepción errada de paquetes radica en el momento en que el receptor realiza el muestreo de la señal y el hecho de que los errores se produzcan solo por la inserción errada de bits 1 y no por la omisión de algún bit cualquiera o la inserción de un bit 0 se debe a dos factores. En primer lugar el algoritmo de ajuste de fase descrito en 3.2.2 impide que no se detecte un cambio de nivel en la señal y por lo tanto los bit 0 siempre van a ser detectados, con esto solo quedan dos opciones en las que se pueden producir errores en la recepción de bits: que se omitan bits 1 debido a que no se realizó un muestreo en el momento correcto detectándose primero un cambio de nivel, o que se inserte un bit 1 pues se realizó un muestreo antes de lo debido. Experimentalmente se ha detectado que el primer caso no ocurre, no obstante no implica que este caso no pueda ocurrir, y es el segundo caso el motivo de los errores en la recepción de los paquetes

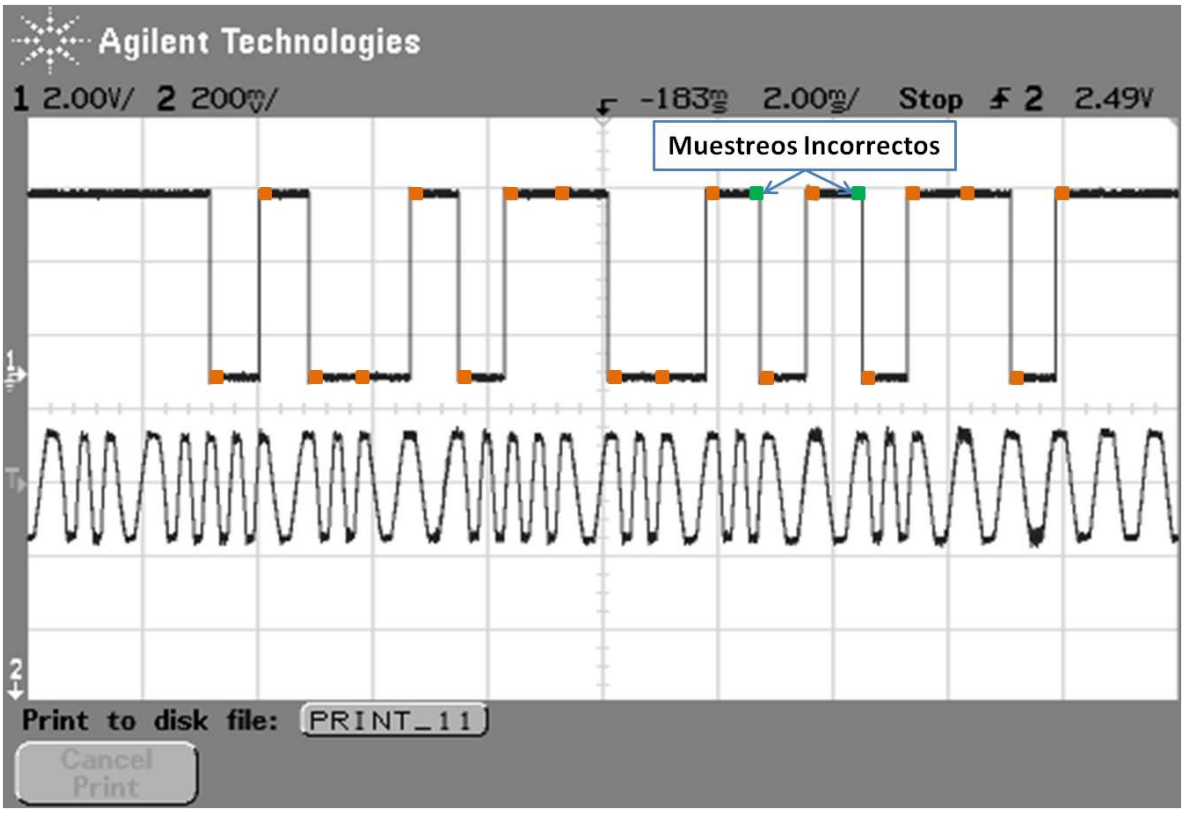

FIGURA 31: IMAGEN EN OSCILOSCOPIO DE MUESTREO INCORRECTO

En la figura 31 se muestra una imagen tomada con osciloscopio de la señal recibida y demodulada por el MX614. En la figura 31 se muestra dónde el receptor realiza los muestreos y se indica también donde se realizan muestreos incorrectos que resultan en la inserción de bits 1. Esta figura correspondería al muestreo errado que provoca la decodificación errada del paquete de la figura 30.

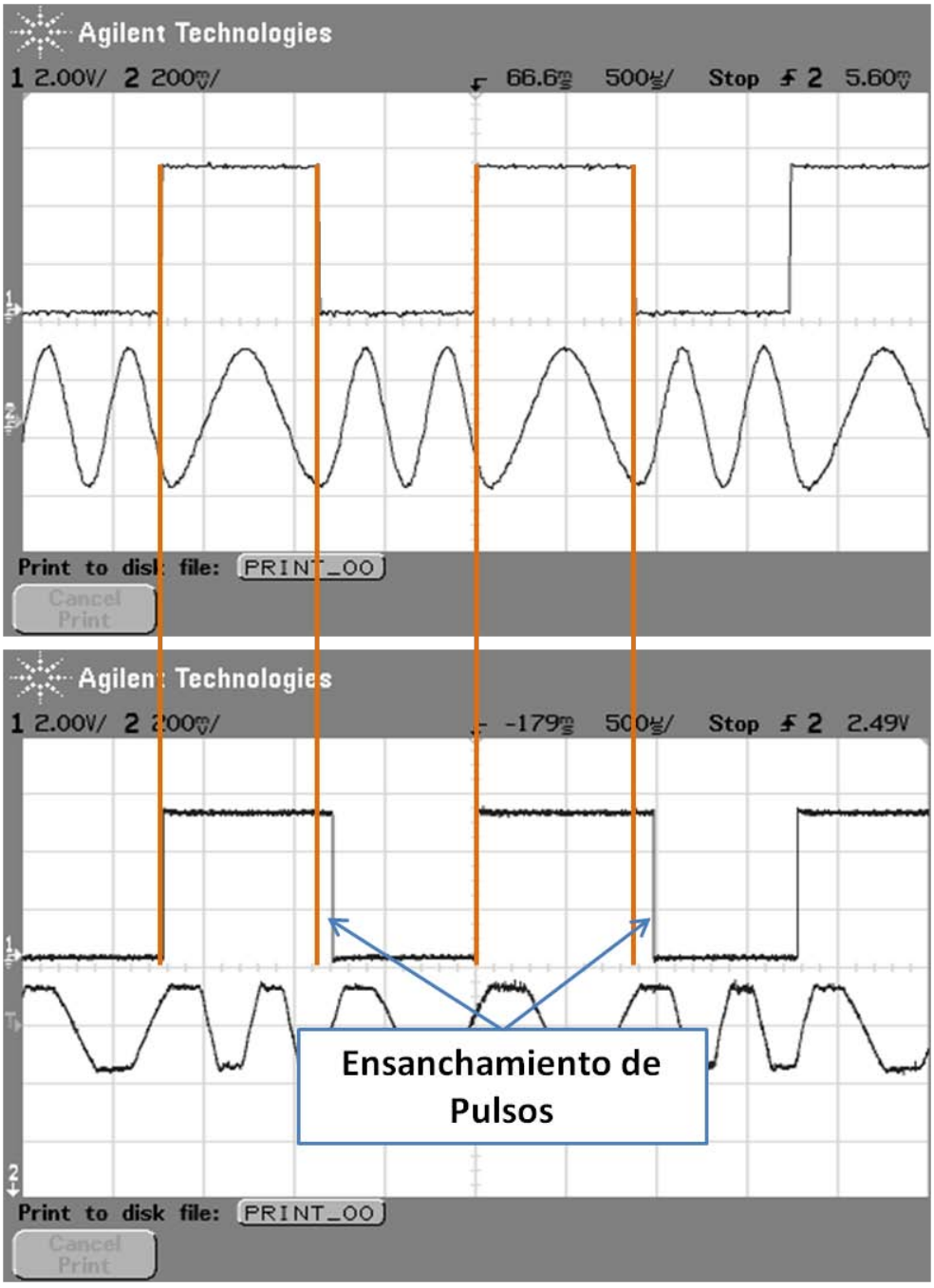

FIGURA 32: ENSANCHAMIENTO DE PULSOS EN LA DEMODULACIÓN

Al observar con más detalle los pulsos de las señales se observó que el periodo de la señal demodulada no es igual al periodo de la señal que fue modulada en el emisor. Cuando el MX614 demodula la señal recibida se detecta un ensanchamiento de algunos pulsos de aproximadamente 100 [µS] lo cual es suficiente para que se produzca un muestreo antes de lo debido y se detecte un bit 1 donde no debería detectarse. En la figura 31 se puede observar este ensanchamiento. En la parte superior de la figura se encuentra la señal modulada en el emisor y en la parte inferior se observa la señal demodulada en el receptor. En la imagen se remarca el ensanchamiento mencionado.

Para poder demostrar que el receptor efectivamente muestrea en estos puntos no deseados se programó en el firmware una función en la que el microcontrolador cambia el estado de un pin cada vez que realiza un muestreo y se encontró que efectivamente se realizan muestreos indebidos de acuerdo a lo mencionado. Además de esto se programó otra función para encontrar en qué puntos el muestreo se realizaba debido a que el *timer* generó una interrupción o bien porque se detecto un cambio de nivel en la señal de entrada. Con respecto a esto último se encontró que, como es evidente, todos los bit 1 son detectados a partir de un muestreo generado por una interrupción del *timer*, sin embargo los bit 0 no son todos detectados por un muestreo generado a partir de un cambio de nivel, sino que hay algunos que son detectados debido a un cambio de nivel y otros que son generados porque el *timer* genera una interrupción justo en el momento que la señal cambia de nivel y detecta un bit 0. Como consecuencia de que los bit 0 sean muestreados tanto por interrupción del *timer* como por cambio de nivel, no es posible identificar cada cambio de nivel. Cabe mencionar que todas estas inexactitudes en la sincronización se deben a los ensanchamientos de los pulsos en la demodulación.

Si bien en este punto se ha ejemplificado el caso de la decodificación errada de la figura 30, no necesariamente la introducción de los bits 1 se produce en estos puntos pues la variación en los periodos de los pulsos no depende de la señal enviada si no que es un factor que depende del hardware en la demodulación. Esto es apreciable ya que para una misma señal que fue grabada y enviada sucesivas veces se obtienen resultados distintos para distintos casos de demodulaciones, lo cual no implica que haya algunas decodificaciones que tiendan a repetirse mas como el caso de la figura 30.

4.2 DISCUSIÓN DE LOS RESULTADOS

En el punto 4.1.1 se describen los resultados de las pruebas al sistema reducido donde se obtuvo los resultados esperados. En esta prueba la transmisión y recepción fue exitosa en todos los casos donde se esperaba que lo fuera. Sin embargo en las pruebas del sistema completo los resultados, descritos en 4.1.2, fueron malos no pudiendo decodificar adecuadamente los paquetes. La gran diferencia entre estos dos resultados radica en la modulación y demodulación del sistema completo que introduce variaciones en el periodo de la señal recibida. En este punto se analizan y comparan los resultados de estas pruebas.

Como se ha descrito en el punto 4.1.2 el problema de recepción radica en el hardware, específicamente en la demodulación realizada por el MX614. De acuerdo a lo mencionado este problema puede ser enfrentado de dos maneras. La primera forma es perfeccionando el hardware de manera de mejorar la recepción para que el periodo de los pulsos de la señal demodulada no varíe como se ha observado y que de esta manera la recepción sea exitosa, sin embargo esta solución implicaría probablemente cambiar el circuito integrado MX614 y aun así, como el hardware es sensible a condiciones externas como la temperatura, no es posible asegurar que esta solución funcione de buena manera en un ambiente no controlado. La segunda forma es modificar el software en el proceso de recepción, específicamente la función `receivebyte()`, de tal manera que se pueda realizar una discriminación más precisa del momento en que se debe realizar o no un muestreo.

En la figura 31 se indica como los muestreos de la señal son realizados al comienzo de cada pulso correspondiente a un bit. Este hecho no es casual y se debe al algoritmo de ajuste de fase. Cada vez que existe un cambio de estado en la señal el sistema lo detecta y en ese instante, que corresponde al inicio del pulso, se realiza el muestreo como se señala en la figura 31. Además, en esta figura, se señalan dos muestreos incorrectos que se encontraron con frecuencia, éstos se producen al final del pulso, y como se ha mencionado, ocurren debido a que el *timer* genera una interrupción antes de que se detecte el cambio de nivel. Hay que señalar que este muestreo se produce en un instante muy cercano al comienzo del próximo pulso o bit. De este análisis se puede proponer introducir un desfase de tiempo que sea menor que el periodo de la señal de datos (por ejemplo un tercio de periodo), entre ésta y el momento de muestreo. Introducir este desfase resolvería el problema pues el muestreo se realizaría después del primer tercio del pulso y no al comienzo con lo cual habría tiempo suficiente para que en los casos donde se pudiese dar un

muestreo antes de lo debido, se detecte primero el cambio de nivel y así obtener un decodificación correcta.

La solución recién mencionada fue implementada pero no se obtuvo mejora en la recepción y esto se atribuye a que el retardo necesario para introducir el desfase, el cual debe realizarse cuando se produzca un ajuste de fase, no siempre es introducido, lo cual pudo ser observado a través de un osciloscopio e incorporando en el firmware la función descrita en 4.2.1 que indica en qué momento se realiza un muestreo. La razón de que existan casos donde no se introduce exitosamente un desfase se atribuye a dos causas. En primer lugar y como ya se mencionó en 4.2.1 hay bits 0 (que corresponden a un cambio de nivel) que no son detectados por el cambio de fase sino que por una interrupción generada por el *timer*, estos son los casos donde no es posible saber si existe o no cambio de nivel al momento del muestreo, y al no ser detectado el cambio de nivel el desfase no se introduce. El segundo factor, y que en parte también es causa del primero, es que la variación en el periodo de la señal demodulada es mayor a lo esperado y por lo tanto introducir un desfase de un tercio de periodo en algunos casos no es suficiente, pero introducir un desfase mayor puede, en otros casos, afectar los muestreos que si son correctos.

Con todo esto surge como necesidad obtener el tiempo que existe entre los cambios de nivel de la señal y de esta manera determinar los rangos de variación de los periodos en la señal demodulada. Al cuantificar el tiempo que existe entre cambios de nivel sería posible generar un algoritmo capaz de discriminar de manera más exacta cuantos periodos y por lo tanto cuantos bits hay entre estos cambios y así decodificar la señal sin necesidad de un muestreo periódico.

En el Anexo A se describe una demodulación y decodificación aplicada a una señal recibida y guardada través de la tarjeta de audio de un computador usando el software MATLAB, en este Anexo se muestra que esta decodificación tuvo mejores resultados que los obtenidos con el hardware en la recepción. Además de la decodificación, en el Anexo A, se realizó una cuantificación del tiempo entre cambios de nivel de la señal como se muestra en los gráficos y se observa que existe una diferencia que puede ser usada como parámetro de discriminación para los intervalos entre cambios de nivel que poseen un bit o más de un bit, lo que indica que la posibilidad de decodificar la señal usando este tiempo entre cambios en vez de un muestreo periódico es posible.

4.3 MEJORAS REALIZADAS PARA LA RECEPCIÓN

A continuación se describen las mejoras realizadas para solucionar los problemas de recepción descritos en 4.1.2. En primer lugar se detalla el algoritmo de esta solución y luego se describen las pruebas realizadas para evaluar su funcionamiento.

4.3.1 ALGORITMO DE DISCRIMINACIÓN POR TIEMPO ENTRE CAMBIOS DE NIVEL

A raíz de lo descrito en el punto 4.2 se modificó el firmware para cuantificar a través del microcontrolador el tamaño de los pulsos de un periodo de duración (lo que se asocia a un bit 0 pues hay cambio de nivel) y de dos periodos seguidos (lo que equivale a un bit 1 pues no hay cambio de nivel). Se cuantificaron las variaciones de los periodos descritas en 4.2 y se registró como valor medio para un pulso de un periodo de duración 4266 ciclos del *timer* con una desviación estándar de 418 ciclos, por lo que se espera un error de 9,8% en el periodo de estos pulsos. Para los pulsos de dos periodos se registro como valor medio 8761 ciclos del *timer* con una desviación estándar de 210 ciclos, en este caso se espera un error de 2,3%. Los resultados mencionados se representan en la figura 33.

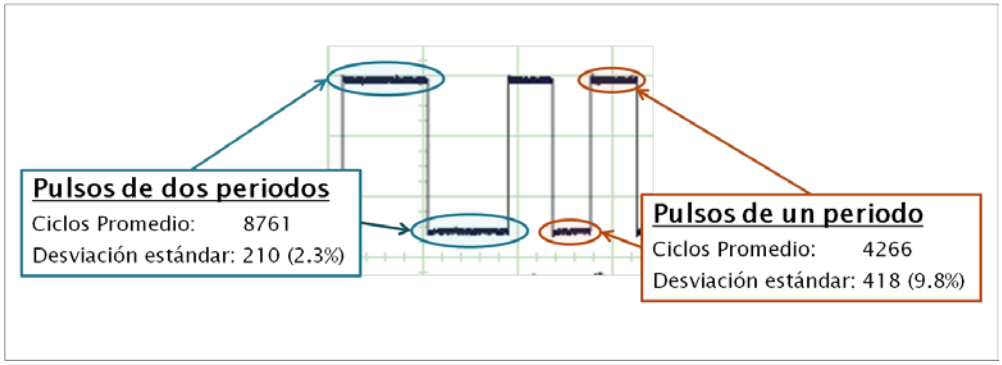

FIGURA 33: REPRESENTACIÓN DE LA CUANTIFICACION DE LOS PERIODOS DE LOS PULSOS

A partir de esto se creó un algoritmo que detecta cuantos ciclos de *timer* hay entre cada cambio de nivel y luego los divide por los ciclos correspondientes a un periodo, es decir 4266 ciclos. Con esto y teniendo en cuenta el error por la variación de los pulsos, se obtiene la cantidad de bit 1 entre cada cambio de nivel y por cada detección de cambio de nivel se obtienen los bit 0.

En la figura 34 se muestra un diagrama de tiempo que representa el funcionamiento del algoritmo mencionado. Se observa que el *timer* comienza a correr con un cambio de nivel, aproximadamente en el tiempo 0.833 [mS]. Luego de iniciado el *timer* el sistema monitorea la señal para detectar el próximo cambio de nivel, que se produce en el tiempo 3.332 [mS].

Inmediatamente después de detectar un cambio de nivel el sistema detiene el *timer*, recupera su valor (que corresponde a los ciclos entre los dos cambios de nivel mencionados) y luego reinicia el *timer* desde el valor cero para comenzar a medir la duración del siguiente pulso.

En paralelo a la medición de la duración del pulso, realizada por el *timer*, se ejecuta el procesamiento de la información como se observa en la figura 34 entre los tiempos 3.332 [mS] y 4.165 [mS]. El procesamiento de la información corresponde a un proceso que usa como variable de entrada el valor recuperado desde el *timer* en la medición anterior para determinar la cantidad de bits 1 que se deben añadir al paquete. Al termino de cada proceso se incorpora un bit 0 que corresponde al cambio de nivel.

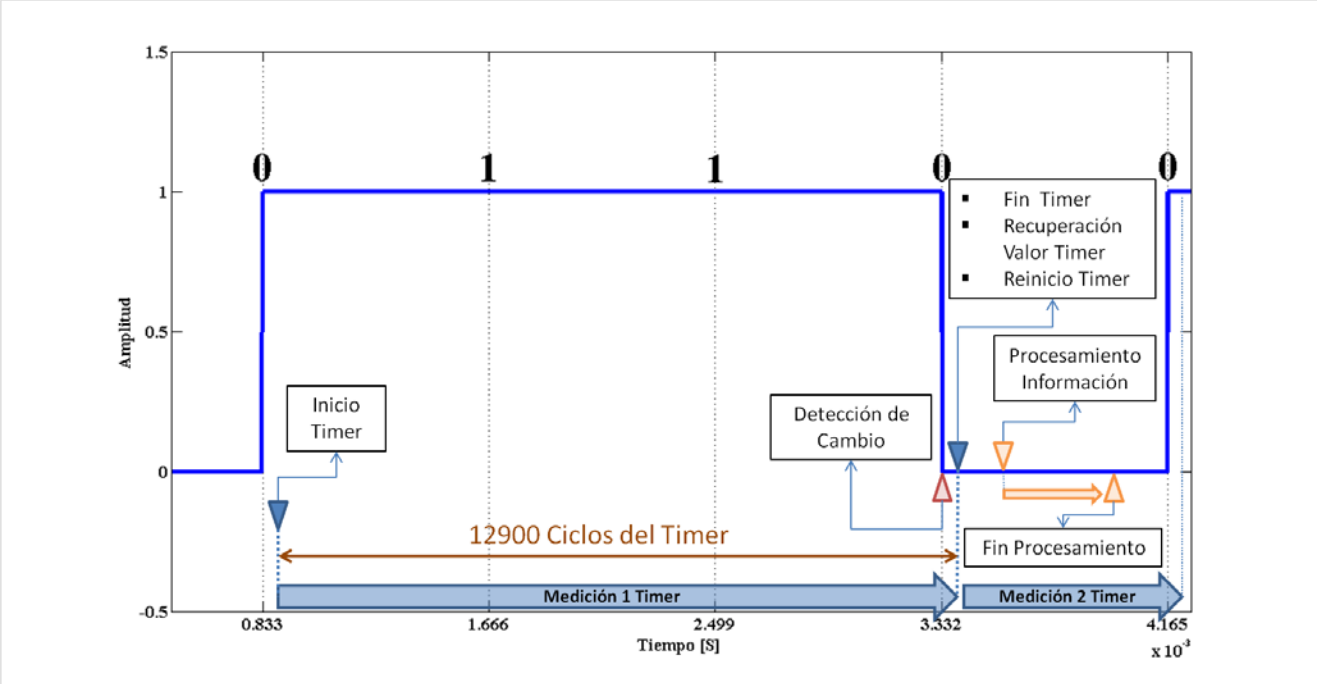

FIGURA 34: DIAGRAMA TEMPORAL DE LA DECODIFICACION USANDO EL ALGORITMO DE DISCRIMINACIÓN POR TIEMP ENTRE CAMBIO DE NIVEL

Como ya se mencionó, para determinar la cantidad de bit 1 existentes en cada pulso el valor recuperado desde el *timer* se divide por 4266 y se aproxima al entero más cercano obteniendo así la cantidad de periodos en cada pulso. En el ejemplo de la figura 34 se observa que en el primer pulso hay tres periodos lo que corresponde a dos bit 1.

Además de determinar la cantidad de bit 1 existentes en cada pulso se debe determinar si lo recibido corresponde a una situación de FLAG, una situación de ERROR o una situación de bits de información. Para esto es necesario incorporar al proceso una maquina de estado que permita discriminar cual es la situación de recepción en cada caso, esta máquina de estado se muestra en la figura 35 y su entrada corresponde a la cantidad de periodos detectados (PD) en cada pulso, por lo tanto la maquina se ejecuta luego de determinar este valor en cada procesamiento de la información[16].

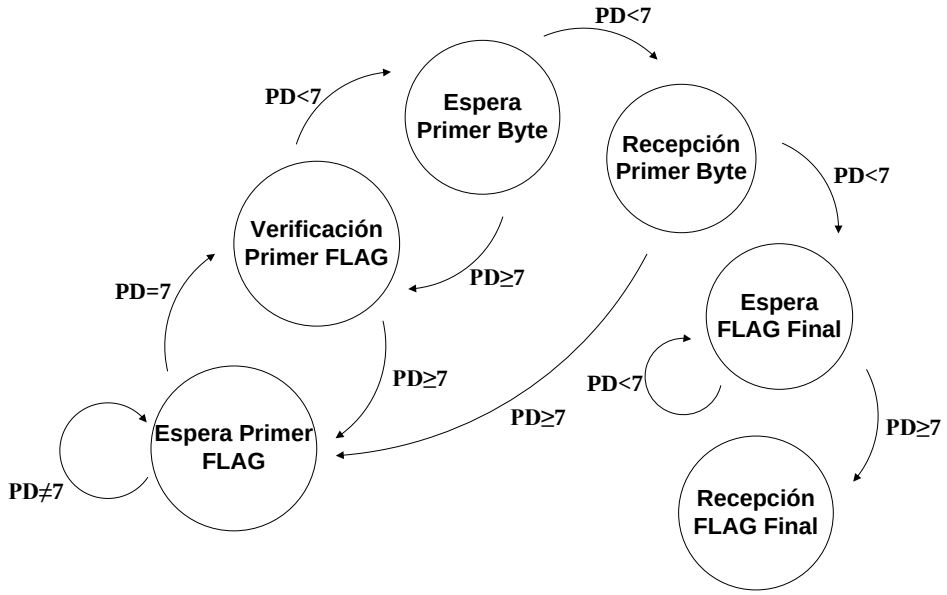

FIGURA 35: MAQUINA DE ESTADO PARA DISCRIMINAR SITUACIÓN DE RECEPCION

4.3.2 RESULTADOS OBTENIDOS A PARTIR DE LAS MEJORAS

Usando el algoritmo descrito en 4.3.2 la recepción mejora considerablemente en relación a lo descrito en 4.1.2. Para realizar las pruebas se usó el montaje del sistema

[16] En el Anexo E se muestra la sección de código del firmware que incorpora la máquina de estado.

completo (al igual como se hizo en 4.1.2) cuyo diagrama de bloques se muestra en la figura 18 y se puede apreciar una fotografía de este montaje en la figura 36.

Las pruebas realizadas consistieron en el envío de 20 trenes de 20 paquetes cada uno para tres tamaños distintos de paquetes. En cada tren los paquetes eran enviados en intervalos de 3 segundos, es decir que se utilizo una transmisión redundante de paquetes de tasa fija como se describe en 2.3.2.2.

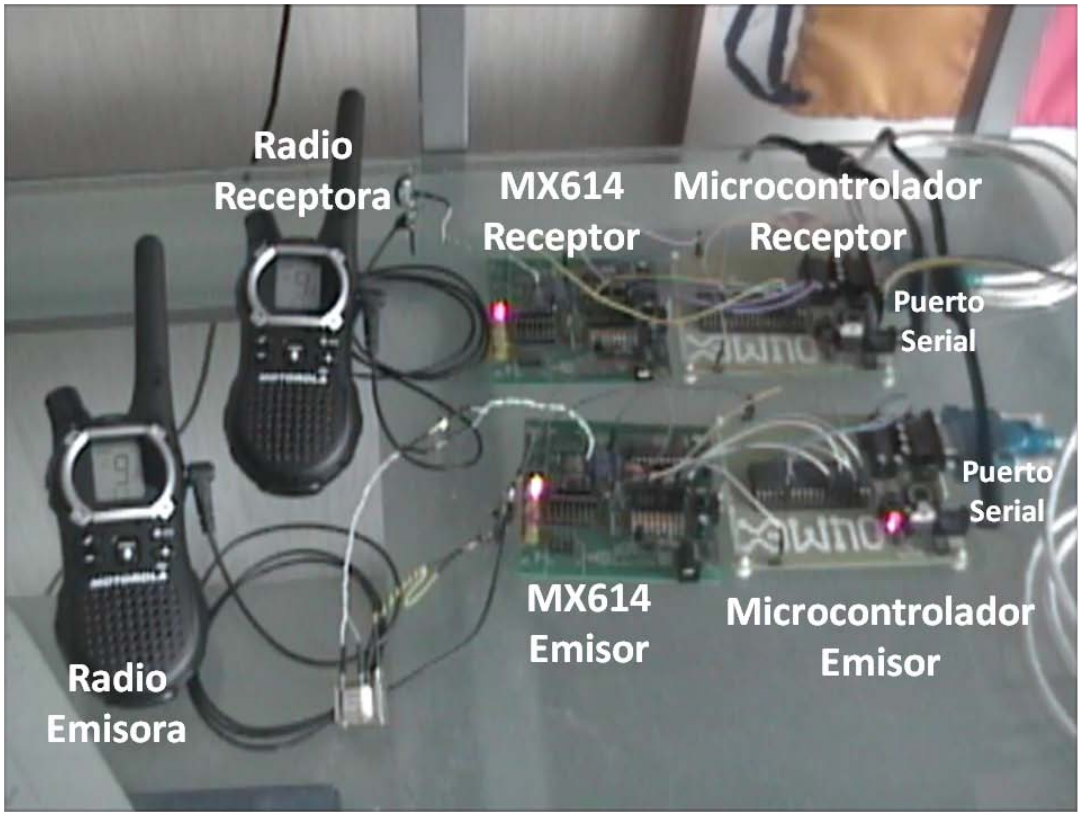

FIGURA 36: FOTOGRAFIA DEL MONTAJE DEL SISTEMA COMPLETO

En la tabla de la figura 37 se presentan los resultados de las pruebas para los tres tamaños de paquetes usados: 10, 37 y 91 Bytes[17]. Se observa que mientras mayor es el tamaño del paquete menor es la tasa promedio de paquetes correctos. Lo anterior se atribuye a que el aumento de los bytes enviados incrementa la probabilidad de fallar en la recepción de algún byte.

[17] En el Anexo C se presentan en detalle estos resultdos.

Para enviar una posición GPS usando una codificación MIC-E[18] es suficiente usar un paquete de 37 Bytes, caso en el cual se obtuvo una tasa promedio de paquetes correctos de 20.5% lo cual es razonable para el sistema. Sin embargo para enviar mensajes de texto o una posición GPS no codificada en MIC-E es necesario usar 91 Bytes, en este último caso se obtuvo una tasa muy baja de recepción, solo un 0.8%.

Tamaño de paquete en Bytes	Promedio de paquetes correctos	Paquetes correctos en la peor serie	Paquetes correctos en la mejor serie	Tasa promedio de paquetes correctos[19]
10	19.7	17	20	98.3 %
37	4.1	2	6	20.5 %
91	0.15	0	1	0.8 %

FIGURA 37: RESULTADOS DE LAS PRUEBAS DE RECEPCION

Cabe mencionar que varios de los paquetes que el sistema detecta como incorrectos solo poseen entre uno y tres bytes incorrectos, ante lo cual la verificación realizada a partir de los bytes de FCS resulta incorrecta. En base a esto se plantea que una forma de mejorar la tasa de recepción es incorporando algún algoritmo que, en base a lo anterior, realice una recuperación de los posibles bytes mal recibidos.

Otro factor destacable en cuanto a una posible mejora en la tasa promedio de paquetes correctos es el relacionado con el hardware. Como se observa en la figura 36 el montaje se realiza con placas modulares y muchos cables los cuales son una fuente importante de ruido. Además, la placa de desarrollo PIC-E usada para montar el MX614 no incorpora un filtro de audio que el fabricante del circuito integrado recomienda usar [2], este filtro eventualmente podría mejorar la recepción.

Se plantea que para seguir realizando pruebas con miras a mejorar la tasa de recepción de paquetes es necesario construir una placa impresa que incorpore todos los elementos usados de manera modular en este montaje y así disminuir lo más posible los errores provocados por el hardware para luego incorporar mejoras a nivel de software.

---

[18] La codificación MIC-E sirve para usar menos Bytes en el envío de posiciones GPS. La descripción de esta codificación se encuentra en el Capitulo 10 de [5]

[19] La *Tasa promedio de paquetes correctos* corresponde al cociente entre el *Promedio de paquetes correctos* y el total de paquetes enviados por serie, es decir 20.

$$Tasa\ promedio\ de\ paquets\ correctos = \frac{Promedio\ de\ paquetes\ correctos}{20}$$

5 CONCLUSIONES

Se plantearon dos objetivos para el desarrollo de este trabajo. El primero consistió en evaluar y diseñar un sistema que permita localizar la posición del globo-sonda descrito haciendo uso de un canal de audio. Para esto se decidió usar el protocolo APRS el cual cumple con los requerimientos, es decir, permite transmitir la posición a través de un canal de audio. Por otro lado, este protocolo es abierto y muy usado por radioaficionados lo que le entregaría al producto final mayor adaptabilidad para trabajar con equipos de otros fabricantes además de ser escalable en cuanto al futuro desarrollo de nuevas aplicaciones sobre el mismo sistema. También en relación a este objetivo, se diseñó el firmware que permitiría al sistema operar usando parte del protocolo APRS. En cuanto a este diseño, es conveniente para el análisis, dividirlo en dos partes: la primera parte corresponde al firmware de alto nivel, encargado de manejar el protocolo APRS y el enlace de datos; la segunda parte corresponde al firmware de bajo nivel encargado de detectar correctamente los bits recibidos y que está fuertemente ligada a la función `receivebyte()` y al algoritmo de ajuste de fase.

Se mostró en 4.1.1 que el diseño del firmware de alto nivel fue satisfactorio, siendo posible manejar correctamente paquetes con información digital enviada usando el protocolo HDLC, que a su vez es usado por el protocolo APRS en su capa de enlace de datos. Esto valida la lógica de los algoritmos descritos en 3.2.

En cuanto al diseño del firmware de bajo nivel se concluye que la estrategia de muestrear de acuerdo a la tasa de transferencia e incorporando el algoritmo de ajuste de fase entrega resultados satisfactorios solo en las condiciones ideales del punto 4.1.1 pues al probarse en condiciones reales como se muestra en 4.1.2 el hardware introduce variaciones de tiempo lo cual afecta la recepción y decodificación de los paquetes. A través de la obtención de la señal por medio de la tarjeta de audio de un computador e implementando el algoritmo de muestreo en el software MATLAB, como muestra el Anexo A, se obtuvo mejores resultados en la recepción. Lo anterior sugiere la posibilidad de implementar un receptor que use una tarjeta de audio como parte del sistema.

En el punto 4.3 se propuso como solución al problema de recepción cambiar la estrategia de muestreo de la señal por una de discriminación por tiempo entre pulsos, es decir, que los muestreos se realicen solamente cuando exista un cambio de nivel en la señal y luego, a partir del tiempo existente entre cada muestreo, se decodifique la información estimando la cantidad de bits que hay en este intervalo de tiempo. Para probar esta solución se implemento un firmware que incorpora la

discriminación por tiempos y se logró una decodificación exitosa de paquetes, como se muestra en el punto 4.3.2, validando así el algoritmo propuesto. Sin embargo en la recepción existe una tasa de paquetes no recibidos la cual aumenta a medida que se aumenta el tamaño del paquete enviado como se observa en la figura 37.

Como segundo objetivo de este trabajo se propuso identificar y evaluar las posibilidades para que el sistema, en futuras versiones, pueda transmitir otro tipo de información digital como mensajes de texto. El protocolo APRS, entre sus características, posee la capacidad de enviar otro tipo de información digital como mensajes de texto; el firmware diseñado incorpora esta funcionalidad. Además del envío de mensaje de texto, se ha incorporado en el firmware la posibilidad de que los mensajes sean repetidos a través de las estaciones terminales, que en este caso serían los globo-sonda, para formar una red de comunicación. Estas funciones; la repetición y envió de mensajes de texto, fueron probadas de acuerdo a lo mencionado en 4.1.1, y por lo tanto se espera una buena respuesta de esto ya que es parte del firmware de alto nivel.

Un aspecto importante y revisado en este trabajo es el que hace referencia a la integración del sistema, esto se refiere a las conexiones del hardware. En el punto 3.1 se describe cómo deben realizarse las conexiones de los circuitos integrados para que el firmware pueda realizar un buen manejo de éstos, y en el Anexo B se describe cómo debe ser el circuito que conecte el módulo con los equipos de radio. La conexión del Anexo B es útil para casi cualquier radiotransmisor portátil que posea sistema de manos libres, lo cual agrega valor a un eventual producto final.

De acuerdo al trabajo realizado se plantea que para continuar con el desarrollo del producto es necesaria la fabricación de un prototipo que incorpore los elementos y consideraciones de hardware descritas en este documento, tanto en el punto 3.1 como en el Anexo B, pues las pruebas han sido realizadas sobre placas de desarrollo modulares que incorporan ruido a las señales y no son óptimas en cuanto a comodidad para trabajar. Se estima que los resultados obtenidos en este trabajo son suficientes para elaborar un prototipo sobre el cual se pueda continuar el desarrollo y trabajar, de acuerdo a las soluciones propuestas, sobre los puntos débiles del estudio, como es la tasa de paquetes no recibidos, y que pueda ser montado en un globo aerostático.

6 BIBLIOGRAFÍA

[1] ARRL, *The ARRL Handbook*., 2010.

[2] MX-COM, INC., *MX614 Data Bulletin*., 2000.

[3] William A Beech, Douglas E Nielsen, and Jack Taylor, *AX.25 Link Acces Protocol for Amateur Packet Radio*, 22nd ed., 1992.

[4] ARRL, *VHF Digital Handbook*., 2008.

[5] The APRS Working Group, *APRS Protocol Referenc*, 101st ed., Ian Wade, Ed., 2000.

[6] TAPR, *PIC-E Assembly and Operations Manual*., 1999.

[7] Microchip, *PIC16F7XA Data Sheet*., 2003.

[8] CCS INC, "CCS C Compiler v.4 Help File,".

A. ANEXO: DEMODULACIÓN USANDO MATLAB

A raíz del resultado obtenido de las pruebas descritas en 4.1.2 se programó en MATLAB un demodulador y un decodificador que hace uso de la tarjeta de audio del computador para captar la señal enviada. El objetivo de este procedimiento es estudiar de manera más profunda los errores existentes en la recepción de paquetes.

Para programar una demodulación se hizo uso de los cruces por cero de la señal de audio, conociendo los cruces por cero es posible determinar el periodo de la señal en un instante determinado y con esto obtener la frecuencia para cada instante de la señal. Conociendo esta frecuencia instantánea se asigna un valor alto o bajo para la señal demodulada que debiese corresponder a la señal que entro al circuito integrado MX614 para generar la señal de audio recibida en el computador.

Para decodificar la señal demodulada y así obtener la información que posee el paquete se usó el mismo algoritmo que usa el firmware del sistema descrito en el punto 3.2.1.

En una primera instancia se realizó sobre este demodulador y decodificador de MATLAB la misma prueba que se realizó sobre el circuito integrado MX614 donde se envía un tren de 20 paquetes de 2 bytes que contienen los caracteres ASCII *D y A*.

A diferencia de las pruebas realizadas sobre el MX614 esta vez fue posible decodificar exitosamente 8 de los 20 paquetes enviados. Estudiando los caso fallidos de decodificación se observó que siempre, al igual a lo inferido en las pruebas del punto 3.2.1, en algún byte del paquete se introduce un bit 1 que no corresponde lo cual provoca un corrimiento del resto del paquete y por lo tanto una recepción fallida del paquete.

En las figuras 38 y 39 se muestran las señales en el proceso de recepción usando MATLAB, en la primera figura se observa una recepción correcta mientras que en la segunda se aprecia un recepción incorrecta provocada por la introducción errada de un bit 1 como se señala en la misma figura. En estas figuras la línea delgada de color rojo muestra la señal de audio recibida, la línea gruesa color azul muestra la demodulación realizada en MATLAB, los puntos azules sobre la señal demodulada son los puntos de muestreo de esta señal para la decodificación y en la línea punteada color verde se muestra el tiempo en milisegundos que hay entre dos cambios de nivel consecutivos.

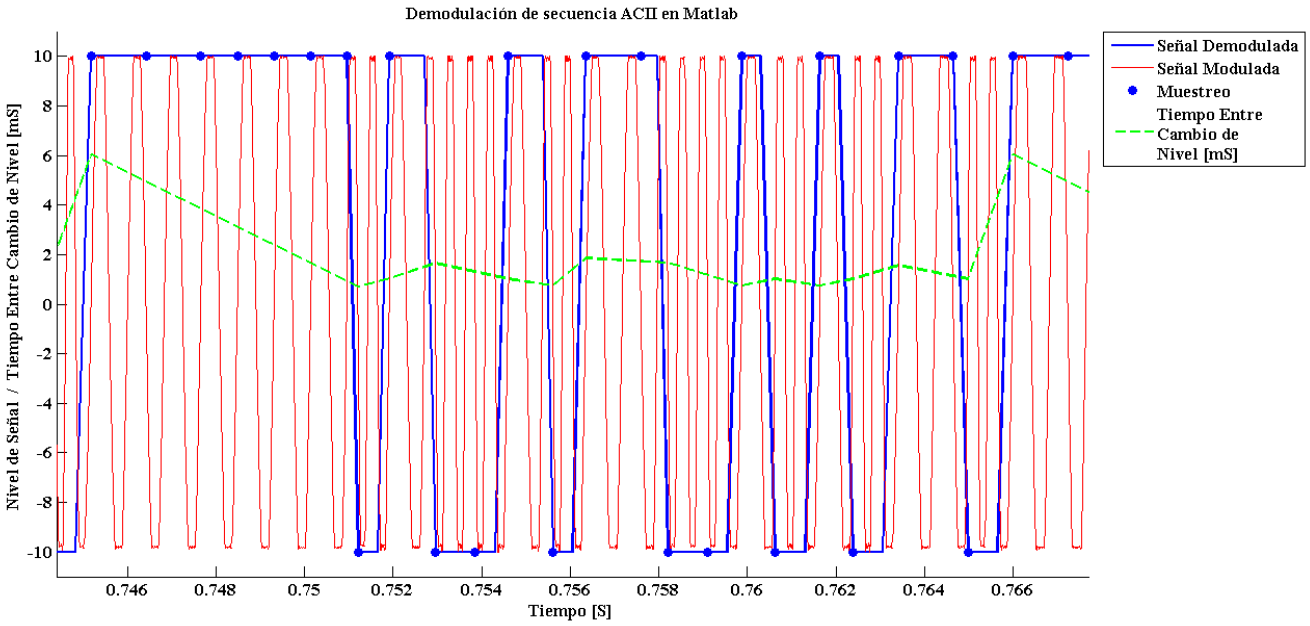

FIGURA 38: RECEPCION CORRECTA USANDO MATLAB

A pesar de que en la figura 38 se aprecia un recepción exitosa se puede ver a partir de los tiempos entre los cambios de nivel consecutivos que existe una asimetría entre la duración de los periodos para un nivel alto y para un nivel bajo, siendo el periodo del nivel alto más corto que el periodo del nivel bajo. Este efecto introduciría algún grado de asincronía en la recepción especialmente en los bit 1 donde no hay cambio de nivel, los bit 0 no se ven afectados por esto debido al algoritmo de ajuste de fase y que como se observa provoca que al inicio de cada cambio de nivel exista un muestreo. Esta asimetría en los periodos de los distintos niveles se produce por el algoritmo usando en la demodulación que no es capaz de definir de manera exacta en qué punto se produce un cambio de frecuencia.

En la figura 39 se observa una recepción errada donde se introduce un bit 1 que no corresponde. Esto se debe a que el algoritmo de decodificación realiza un muestreo de un no cambio de nivel donde no corresponde, de manera anticipada. Este efecto se produce reiteradamente en la recepción incorrecta de paquetes y es la razón de la introducción de los bit 1 que provoca la recepción incorrecta.

Además de esta prueba se enviaron 4 trenes de 20 de paquetes de 12 bytes (usando nuevamente solo los caracteres ASCII *D* y *A*) recibiendo de manera correcta 2 paquetes de los 80 enviados. Claramente la recepción correcta disminuye considerablemente a medida que el tamaño del paquete aumenta, pero aun así se

observó que los errores siguen siendo producidos por la introducción de uno o más bits 1 debido a un muestreo anticipado como se expone en la figura 39.

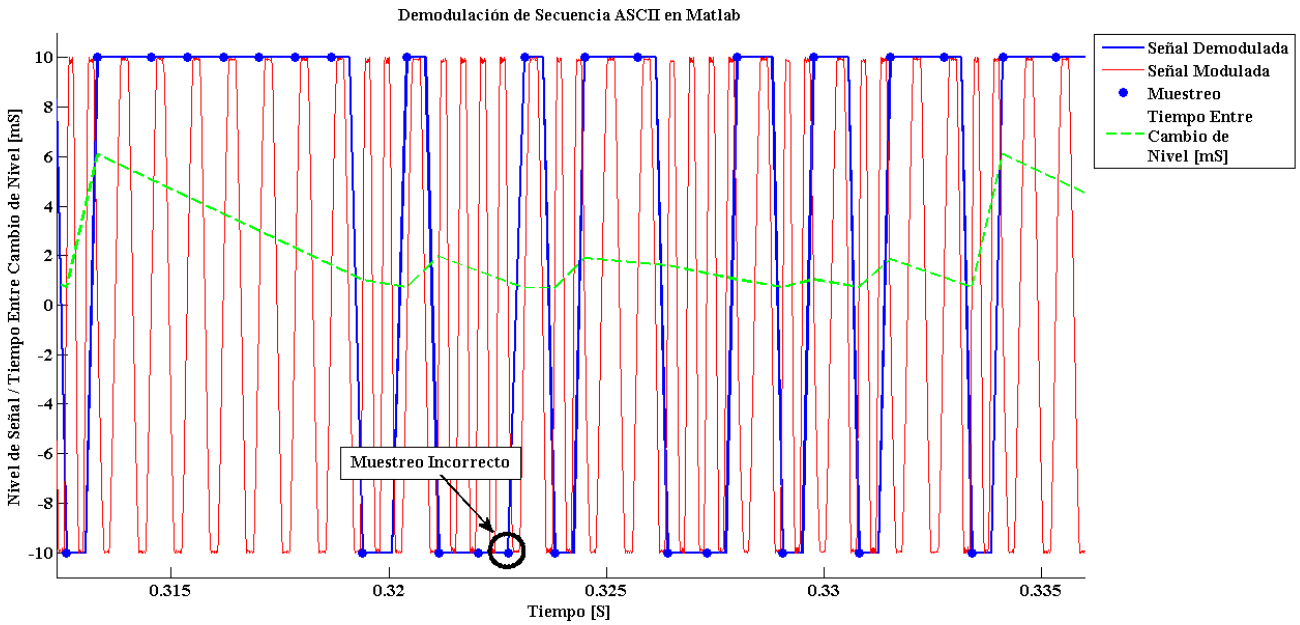

FIGURA 39: RECEPCIÓN INCORRECTA USANDO MATLAB

Este muestreo anticipado indica que el proceso de espera para realizar un muestreo mientras no exista un cambio de nivel no es adecuado cuando se realiza una demodulación.

B. ANEXO: CONEXIÓN CON RADIOTRANSMISOR

Para poder integrar el sistema es necesario tener en cuenta la conexión del TNC con el radiotransmisor. Esta conexión se realiza por el conector de audio de la radio que por lo general posee tres terminales:

- RADIO GND: Corresponde a la tierra de la radio.
- RADIO OUT: Corresponde a la salida del audio de la radio.
- RADIO IN: Corresponde a la entrada de audio de la radio.

Para enviar una señal de audio es necesario tomar control del canal de la radio. Para esto normalmente se utiliza el botón PTT (*Push To Talk*), sin embargo cuando se conecta a la radio por medio de un conector externo el PPT se activa cuando se cierra el circuito entre RADIO IN y RADIO GND. Para lograr esto se debe conectar el TNC a la radio usando el circuito que se muestra en la figura 40 donde se usa un transformador de audio y un transistor para cerrar este circuito solo cuando el pin PTT del microcontrolador esta en nivel alto lo cual sucede en el momento en que se está enviando información por el sistema.

Cada terminal mostrado en el circuito de la figura 40 va conectado con el pin del mismo nombre en el MX614 (RXIN y TXOUT) o de PIC16F877A (PTT) según corresponda.

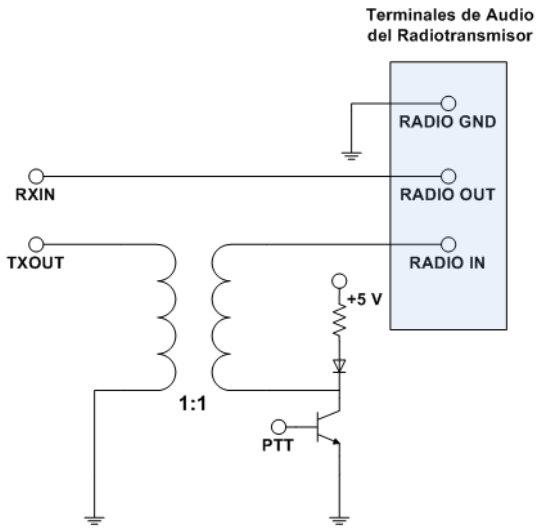

FIGURA 40: CONEXIÓN DEL PUSH TO TALK

C. ANEXO: DETALLE DE LOS RESULTADO DE PRUEBAS

En este anexo se presenta en la figura la tabla con el detalle de los resultados de las pruebas descritas en 4.3.2.

Paquetes de 91 Bytes			**Paquetes de 37 Bytes**			**Paquetes de 10 Bytes**		
Serie	Paquetes enviados	Paquetes correctos	Serie	Paquetes enviados	Paquetes correctos	Serie	Paquetes enviados	Paquetes correctos
1	20	0	1	20	4	1	20	20
2	20	0	2	20	4	2	20	18
3	20	0	3	20	3	3	20	20
4	20	0	4	20	4	4	20	20
5	20	0	5	20	5	5	20	20
6	20	0	6	20	4	6	20	20
7	20	1	7	20	5	7	20	19
8	20	0	8	20	3	8	20	20
9	20	0	9	20	4	9	20	20
10	20	0	10	20	2	10	20	17
11	20	0	11	20	5	11	20	20
12	20	1	12	20	4	12	20	20
13	20	0	13	20	6	13	20	20
14	20	0	14	20	4	14	20	20
15	20	0	15	20	3	15	20	20
16	20	1	16	20	4	16	20	20
17	20	0	17	20	5	17	20	20
18	20	0	18	20	5	18	20	19
19	20	0	19	20	4	19	20	20
20	20	0	20	20	4	20	20	20
Promedio paquetes correctos		**0,15**	Promedio paquetes correctos		**4,10**	Promedio paquetes correctos		**19,65**
Tasa de paquetes correctos		**0,75%**	Tasa de paquetes correctos		**20,50%**	Tasa de paquetes correctos		**98,25%**

FIGURA 41: DETALLE DE RESULTADOS DE LAS PRUEBAS

En la figura 42 se muestra un grafico obtenido a partir de los resultados mostrados en la tabla de la figura 41. En este grafio se presenta como la tasa de paquetes correctos decae a medida que el tamaño de los paquetes enviados aumenta.

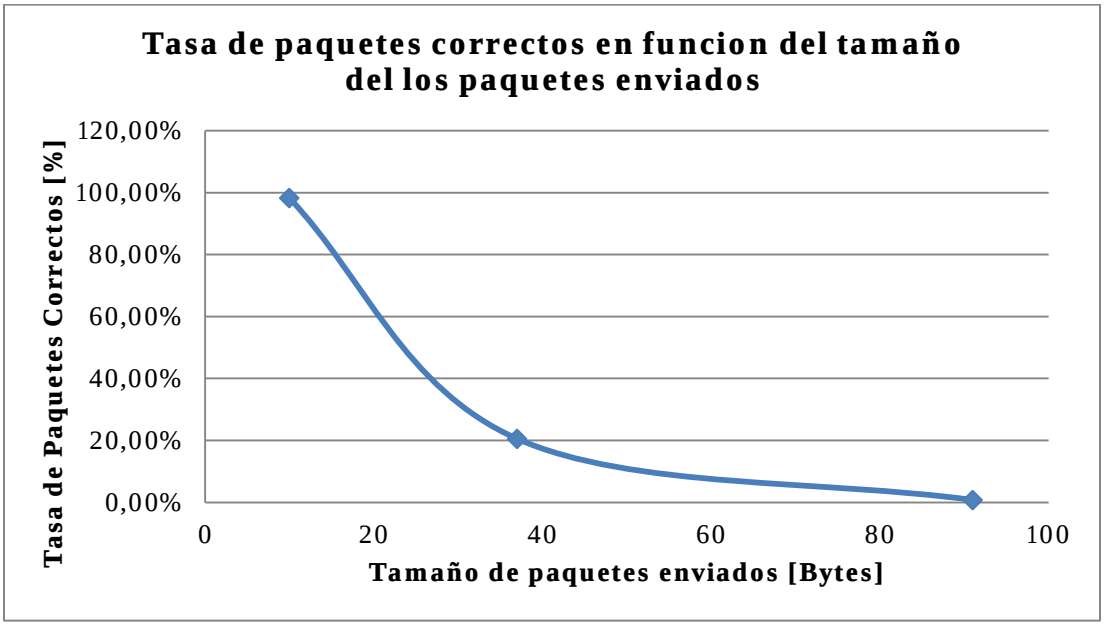

FIGURA 42: GRAFICO DE TASA DE PAQUETES CORRECTOS EN FUNCION DEL TAMAÑO DE PAQUETE

D. ANEXO: RECURSOS USADOS EN EL PROTOTIPO

En este anexo se presentan los recursos usados en el desarrollo del prototipo. En la tabla de la figura 43 se presentan los recursos monetarios usados en armar el montaje experimental. En la tabla de la figura 44 se muestran los recursos en horas hombre usadas en el desarrollo del prototipo detallado por tarea.

Además de los recursos mostrados en estas tablas se uso: 2 Adaptadores USB-Serial, 2 Equipos de Radio Handie, 2 Adaptadores 9VDC y 1 Computador.

Producto	Cantidad	Precio Unitario	Precio Total
Placa Impresa PIC-E	2	$ 7.500	$ 15.000
Componentes Placa PIC-E	1	$ 5.000	$ 5.000
MX614	2	$ 4.500	$ 9.000
Transformador de audio	2	$ 7.000	$ 14.000
Placa de desarrollo PIC16F877A	2	$ 13.114	$ 26.228
Conector manos Libres	2	$ 3.000	$ 6.000
Total Montaje de Pruebas			**$ 75.228**

Valor de unitario del prototipo	**$ 37.614**

FIGURA 43: DETALLE DE LOS RECURSOS USADOS EN EL MONTAJE EXPERIMENTAL EN PESOS CHILENOS

Tarea	Horas Hombre
Diseño Firmware	245
Diseño Pruebas	70
Implementación Plataforma Física de Pruebas	35
Realización de Pruebas	105
Depuración de Errores	91
Total	546

FIGURA 44: DETALLE DE LOS RECUSOS EN HORAS HOMBRES USADAS EN L DESARROLLO

E. ANEXO: CÓDIGO FIRMWARE

En este anexo se presenta una parte del código del firmware con sus respectivos comentarios. Esta sección de código corresponde al ciclo principal del firmware el cual incluye la máquina de estados mencionada en 4.3.1.

```
while (true){   //INICIO LOOP PRINCIPAL
        unsigned int16 i_noval;

        while(bit_test(port_b,1)==1){  //INICIO POR DETECCION DE POTENCIA

                int i_rcvTmg;
                        act=(bit) bit_test(port_b,2);  //SE VERIFICA ESTADO DE SEÑAL
                        setup_timer_1(T1_INTERNAL); //SE INICIALIZA EL TIMER
                        set_timer1(0);
                        i_rcvTmg=0;

                while(i_byte<95 && reception_end==0){

                        while( (act_frec=(bit) bit_test(port_b,2))== act){};  //SE DETECTA
                                                                              //CAMBIO DE NIVEL
                        setup_timer_1(T1_DISABLED);  //SE DETIENE EL TIMER
                        tmPulse=(int16) get_timer1();   //SE  RECUPERA EL VALOR DEL TIMER

                        act=(bit) bit_test(port_b,2);  //SE VERIFICA ESTADO DE SEÑAL
                        setup_timer_1(T1_INTERNAL);  //SE REINICIA EL TIMER
                        set_timer1(0);

                        numb_arr[0]=numb_arr[1];
                        numb_arr[1]=numb_arr[2];
                        numb_arr[2]=numb_arr[3];
                        numb_arr[3]=numb_arr[4];

                        //SE CALCULA LA CANTIDA DE PERIODOS
                        numb_arr[4]= (int) ceil( ( ((double) tmPulse)/4266 )-0.5);

                        switch(state){  //INICIO MAQUINA DE ESTADO

                                case 1:  //ESTADO: ESPERA PRIMER FLAG
                                        if (numb_arr[4]==7)
                                                state++;
                                break;
```

```
case 2:  //ESTADO: VERIFICACION PRIMER FLAG
         if(numb_arr[4]<7)
                  state=3;
         if(numb_arr[4]>=7)
                  state=1;
break;

case 3:  //ESTADO: ESPERA PRIMER FLAG
         if(numb_arr[4]>=7)
         state--;
         else
         state++;
break;

case 4:  //ESTADO: RECEPCION PRIMER FLAG
         i_rcvTmg=4;

         for(i=0;i<numb_arr[2]-1;i++){  //INSERCION DE 1's
                  var=((var  >> 1)| 0b10000000);
                  n_byte++;
                  if(n_byte>=8){
                  numb_arr2[i_byte]=var;
                  n_byte=0; var=0;
                  i_byte++;
                  }
         }
         insertZero= numb_arr[4]==6 ? 0 : 1; //BIT STUFFING


         if(insertZero==1){  //INSERCION DE 0
                  var=var>>1;
                  n_byte++;
                  if(n_byte>=8){
                  numb_arr2[i_byte]=var;
                  n_byte=0; var=0;
                  i_byte++;
                  }
         }

         for(i=0;i<numb_arr[3]-1;i++){  //INSERCION DE 1's
                  var=((var  >> 1)| 0b10000000);
                  n_byte++;
                  if(n_byte>=8){
                  numb_arr2[i_byte]=var;
                  n_byte=0; var=0;
                  i_byte++;
                  }
         }
```

```
            insertZero= numb_arr[4]==6 ? 0 : 1;  //BIT STUFFING

            if(insertZero==1){  //INSERCION DE 0
                  var=var>>1;
                  n_byte++;
                  if(n_byte>=8){
                  numb_arr2[i_byte]=var;
                  n_byte=0; var=0;
                  i_byte++;
                  }
            }

            for(i=0;i<numb_arr[4]-1;i++){  //INSERCION DE 1's
                  var=((var  >> 1)| 0b10000000);
                  n_byte++;
                  if(n_byte>=8){
                  numb_arr2[i_byte]=var;
                  n_byte=0;
                  var=0;
                  i_byte++;
                  }
            }
            insertZero= numb_arr[4]==6 ? 0 : 1;  //BIT STUFFING

            state++;
break;

case 5:  //ESTADO:  ESPERA FLAG FINAL
if(numb_arr[4]>=7){
N_timesPulse=i_rcvTmg+2;
state++;
Nbits=i_rcvTmg;
reception_end=1;
}
else{

if(insertZero==1){  //INSERCION DE 0
            var=var>>1;
            n_byte++;
            if(n_byte>=8){
            numb_arr2[i_byte]=var;
            n_byte=0;
            var=0;
            i_byte++;
            }
}
```

```
                        for(i=0;i<numb_arr[4]-1;i++){  //INSERCION DE 1's
                                var=((var  >> 1)| 0b10000000);
                                n_byte++;
                                if(n_byte>=8){
                                numb_arr2[i_byte]=var;
                                n_byte=0;
                                var=0;
                                i_byte++;
                                }
                        }
                        insertZero= numb_arr[4]==6 ? 0 : 1;  //BIT STUFFING
                        }

                break;

                default:  //ESTADO:  RECEPCION FLAG FINAL
                                reception_end=1;
                break;
                }

        i_rcvTmg++;

        } // FIN MAQUINA DE ESTADO

        setup_timer_1(T1_DISABLED);
        crcpacket(0,i_byte-2);

        if(crchi==numb_arr2[i_byte-2] && crclo==numb_arr2[i_byte-1])
                printf("********PAQUETE CORRECTO********\n\r");
        else
                printf("**********************************\n\r");

        for(i_noval=0;i_noval<i_byte-2;i_noval++)
                putc(numb_arr2[i_noval]);
        printf("| \n\n\r**********************************\n\n\n\n");

        state=1;  //REINICIO MAQUINA DE ESTADO
        i_byte=0;
        n_byte=0;
        var=0;
        reception_end=0;
        insertZero=1;
    }  // FIN CICLO POR DETECCION DE POTENCIA
}  // FIN  DEL LOOP PRINCIPAL
```