

UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA



**“CONSTRUCCIÓN DE UN MÓDEM PARA
COMUNICACIÓN SATELITAL (TNC) DE OPERACIÓN
CON MODULACIONES MÚLTIPLES MEDIANTE
SELECCIÓN AUTOMÁTICA”**

TESIS
PARA OPTAR EL TÍTULO DE INGENIERO
ELECTRÓNICO

PRESENTADA POR
DANTE INGA NARVÁEZ

PROMOCIÓN
2009-II

LIMA, PERÚ
2013

CONSTRUCCIÓN DE UN MÓDEM PARA
COMUNICACIÓN SATELITAL (TNC) DE OPERACIÓN
CON MODULACIONES MÚLTIPLES MEDIANTE
SELECCIÓN AUTOMÁTICA

Dedicatoria

Dedico este trabajo a mi padre Génesis, a mi madre Cira, y a mis hermanos Daisy y Génesis, por ser ejemplos de responsabilidad, sacrificio, bondad e integridad, cualidades claves para obtener el éxito y la felicidad.

SUMARIO

Desde el lanzamiento del primer satélite artificial construido por radioaficionados, en 1961, se han desarrollado muchos satélites de este tipo, pequeños y de bajo costo en comparación con los satélites artificiales comerciales. Estos pequeños satélites tienen un sistema de comunicaciones, que les permite recibir y enviar información desde y hacia una estación en Tierra equipada de manera adecuada para cada satélite. Dentro de los modos de comunicación existentes entre pequeños satélites y sus respectivas estaciones terrenas, los más comunes usan modulaciones digitales AFSK, GMSK, FSK, PSK, OFDM, etc. Cada satélite artificial es diseñado para operar en algunos de los modos mencionados, por lo que una estación terrena debe contar con varios equipos o reconfigurar el modo de operación de sus equipos una y otra vez, si se quiere recibir información de varios satélites artificiales.

Para aumentar la eficiencia de una estación terrena, se puede automatizar el proceso de reconfiguración del modo de operación de los equipos de comunicación, entre los cuales está aquel encargado de modular/demodular y codificar/decodificar los datos provenientes del satélite: el TNC. Con esta finalidad, se construyó un Controlador de Nodo Terminal - TNC hardware capaz de operar en varios tipos de modulación, con un mecanismo de selección automático de tipo de modulación. El TNC propuesto está constituido por chips módem, multiplexores y dos microcontroladores, uno para codificar y decodificar los datos, y el otro para seleccionar automáticamente el módem.

Se probó el algoritmo de control con la recepción automatizada de señales de varios pequeños satélites con modos de operación distintos. También se probó el mecanismo de conmutación del TNC con la decodificación de señales grabadas que simulaban dos satélites pasando simultáneamente sobre una estación terrena.

ÍNDICE

CAPÍTULO I	1
INTRODUCCIÓN	1
1.1. Justificación	1
1.2. Motivación	1
1.3. Estructura de la tesis	1
CAPÍTULO II	3
OBJETIVOS Y ALCANCES	3
2.1. Objetivos	3
2.2. Alcances	3
CAPÍTULO III	4
MARCO TEÓRICO	4
3.1. Introducción	4
3.3. Sistemas de comunicación y arquitecturas de red	10
3.4. Comunicaciones en pequeños satélites	13
3.5. Modulaciones digitales	14
3.6. Codificaciones usadas en satélites	18
3.7. Protocolos de comunicación en TNCs	18
3.8. Análisis y estimación del efecto Doppler	20
3.9 Órbitas de satélites	28
CAPÍTULO IV	29
PLANTEAMIENTO DEL PROBLEMA	29
4.1. Descripción de la problemática	29
4.2. Alternativas de solución	31
CAPÍTULO V	33
SOLUCIÓN PROPUESTA	33
5.1. Metodología de diseño del Hardware	35
5.2. Metodología de diseño del Software	46
5.3. Software de gestión para la PC	49
5.4. Plan de pruebas del mecanismo de selección	50

CAPÍTULO VI	52
SIMULACIÓN DEL ALGORITMO DE SEGUIMIENTO DE SATÉLITES	52
6.1. Introducción	52
6.2. Programas usados.....	53
6.3. Pruebas de caja blanca	58
6.4. Pruebas de caja negra.....	62
CAPÍTULO VII	64
IMPLEMENTACIÓN Y PRUEBAS DEL TNC PROPUESTO	64
7.1. Introducción.	64
7.2. Pruebas individuales y del sistema	64
7.3. Consumo de recursos.....	82
7.4. Comparación con TNCs similares.....	84
7.5. Cronograma de tiempos	84
CONCLUSIONES	86
ANEXO A	88
CODIFICACIÓN MORSE	88
ANEXO B	92
CODIFICACIÓN AX.25	92
ANEXO C	98
DIAGRAMAS ESQUEMÁTICOS	98
ANEXO D	102
PLACAS IMPRESAS	102
ANEXO E	105
LISTA DE COMPONENTES	105
ANEXO F	111
CÓDIGOS FUENTE	111
ANEXO G	171
RESULTADOS DERIVADOS	171
ANEXO H	173
LISTA DE FIGURAS, ABREVIATURAS	173
BIBLIOGRAFIA	180

CAPÍTULO I

INTRODUCCIÓN

1.1. Justificación

El presente trabajo forma parte de la línea de investigación y desarrollo satelital de la Universidad Nacional de Ingeniería (UNI, Lima, Perú), línea que es ejecutada por el Centro de Tecnologías de la Información y Comunicaciones (CTIC-UNI) y el Instituto Nacional de Investigación y Capacitación de Telecomunicaciones (INICTEL-UNI).

1.2. Motivación

El motivo del presente trabajo se ampara en la necesidad de contar con un equipamiento Controlador de Nodo Terminal (TNC) que automáticamente conmute el tipo de modulación del satélite en observación, facilitando la toma de datos de diferentes sistemas de comunicación. Todo esto en coherencia con la línea de investigación y desarrollo satelital de la Universidad Nacional de Ingeniería (UNI, Lima, Perú), fomentando el desarrollo de tecnología propia en el área de satélites de investigación, además de documentar el proceso de diseño y construcción de un TNC, dispositivo importante usado en estaciones en Tierra para lograr comunicación con satélites de investigación. Estos desarrollos permiten el ahorro de divisas, evitando la compra de equipos y resolviendo un problema para el que no existe solución a la fecha.

1.3. Estructura de la tesis

La tesis está formada por siete capítulos y ocho anexos, además de las conclusiones, observaciones y recomendaciones, y referencias usadas.

El capítulo 1 contiene la motivación del desarrollo de la tesis, así como una descripción breve del contenido de cada capítulo y de los anexos.

El capítulo 2 describe los objetivos y alcances de la tesis, es decir, lo que se espera lograr y las limitaciones del trabajo realizado.

El capítulo 3 resume los conceptos que son importantes para el desarrollo del presente trabajo, desde los antecedentes, conceptos de modulaciones y codificaciones, hasta los efectos del movimiento de los satélites en la recepción de señales emitidas por dichos satélites en forma de ondas electromagnéticas.

El capítulo 4 presenta la problemática existente en la actualidad para aumentar la eficiencia de una estación en Tierra, debido a la gran cantidad de satélites de investigación disponibles.

El capítulo 5 plantea la solución propuesta al problema descrito en el capítulo 4. Aquí se detallan la metodología de diseño del hardware del TNC propuesto, la metodología de diseño del firmware para los dispositivos programables del TNC, y la metodología de diseño del software para un computador que interactuará con el TNC.

El capítulo 6 hace referencia a la implementación y simulación del algoritmo de seguimiento automático de satélites, que es una parte importante para la funcionalidad del TNC propuesto, pues está implementado en el computador e interactúa con el TNC, además de controlar otros equipos de la estación en Tierra.

El capítulo 7 presenta la implementación y pruebas del TNC propuesto, detallando sus componentes, resultados de pruebas individuales y pruebas con datos reales. Se realiza también una comparación con otros TNC de características similares. Además, se realizan análisis de costos y de los tiempos estimados de duración de las etapas de desarrollo.

Los anexos contienen información adicional de interés, como descripción de las dos formas más comunes de codificación de datos en satélites de investigación, recopilación de diagramas esquemáticos y diseños de placas impresas, lista de materiales y códigos fuente usados para programar los dispositivos y el computador, así como publicaciones logradas en el desarrollo del presente trabajo.

CAPÍTULO II

OBJETIVOS Y ALCANCES

2.1. Objetivos

El objetivo general de la tesis es contribuir al desarrollo de dispositivos módem de estaciones en Tierra para comunicación con satélites de investigación, mediante una propuesta innovadora que facilite la comunicación con una variedad de estos sistemas de comunicación, acorde a las necesidades actuales de la línea de investigación y desarrollo satelital de la Universidad Nacional de Ingeniería y usando materiales accesibles en el mercado local.

Los objetivos específicos se derivan del anterior y son:

- ✓ Diseñar e implementar un módem para comunicación con pequeños satélites.
- ✓ Aumentar la eficiencia en tiempo de comunicación de una estación en Tierra.
- ✓ Automatizar el proceso de recepción y decodificación de señales de diversos satélites.
- ✓ Usar materiales y herramientas disponibles localmente y de bajo costo.

2.2. Alcances

El módem para comunicación satelital (TNC) propuesto está orientado al uso en estaciones terrenas de pequeños satélites o de investigación. Al tratarse de satélites de investigación se tienen diversos tipos de modulación implementados en dichos satélites, por tanto existen variedad de TNCs, algunos dedicados a un satélite y otros que intentan ser genéricos. El presente trabajo consiste en la construcción de un módem que busca integrar las tecnologías existentes de hardware y software, con el fin de conseguir un método casi totalmente automatizado de interpretación de señales enviadas por satélites de investigación. Se han implementado los dos tipos de modulación más usados en la actualidad, sin embargo, el TNC propuesto puede albergar hasta cuatro tipos de modulación cualesquiera. Asimismo, se han integrado programas de software en una aplicación gráfica para un computador que controla el TNC y otros equipos de la estación terrena, a la vez que muestra los datos obtenidos de la comunicación con los satélites que pasan sobre la estación terrena.

CAPÍTULO III

MARCO TEÓRICO

3.1. Introducción

Desde 1961 se han lanzado al espacio satélites de pequeño tamaño, desarrollados inicialmente por radioaficionados con la finalidad de experimentar en temas de radiopropagación. El desarrollo de pequeños satélites se extendió cuando la Universidad Estatal Politécnica de California junto a la Universidad de Stanford en Estados Unidos desarrollaron el proyecto de estandarización de pequeños satélites denominados Cubesat [1], creando además oportunidades para otras universidades. Actualmente, cada vez más universidades e institutos de investigación en varios países desarrollan este tipo de tecnología, con la finalidad de explorar el campo aeroespacial y encontrar aplicaciones que aprovechen el bajo costo y la rapidez relativa de su fabricación.

Se denominan pequeños satélites a aquellos satélites artificiales que son de tamaño mucho menor a los satélites comerciales, y que son desarrollados por radioaficionados y/o universidades con fines de investigación del espacio, hasta ahora. Los satélites artificiales considerados de tamaño pequeño son: microsátélites, nanosatélites, picosatélites y hasta femtosatélites. Por tanto, estos satélites comparten ciertas características: bajo costo, rápido desarrollo, fines educativos, entre otros. Este tipo de satélites en su mayoría sigue una trayectoria de órbita baja conocida como tipo LEO (Low-Earth Orbit), orbitando en una trayectoria circular alrededor de la tierra, aproximadamente entre 400 y 2000 Km de distancia de la superficie de la tierra. Sin embargo, también existen satélites de este tipo que siguen trayectorias de tipo HEO (High-Earth Orbit) con distancias a la superficie de la tierra bastante mayores.

AMSAT (AMateur SATellites) [2] es una organización que promueve el desarrollo de pequeños satélites, a los cuales inicialmente denominó OSCAR (Orbiting Satellite Carrying Amateur Radio). Se inició como un proyecto, donde un grupo de radioaficionados planeó poner en órbita una serie de satélites con el fin de investigar las propiedades de la atmósfera en la transmisión de ondas en las bandas de frecuencia de HF, VHF, UHF y banda S. La idea era equipar satélites de pequeño tamaño con radios que puedan transmitir en dichas bandas de frecuencia, y monitorearlos desde tierra. Desde 1999, se han sumado al desarrollo de satélites

de este tipo universidades e institutos tecnológicos de diversos países, con el fin de investigar las propiedades de la atmósfera, probar de manera experimental dispositivos en condiciones extremas y tomar datos del espacio.

CubeSat es un estándar desarrollado por las Universidades de California y Stanford en California USA, que debe su nombre a la forma del satélite, que es un cubo de 10cm de lado y de 1Kg de peso como máximo. Por su tamaño, éstos satélites son clasificados como pico-satélites, y se han convertido en un estándar de referencia para el desarrollo de satélites similares en otras universidades del mundo. Actualmente siguen este estándar alrededor de 80 universidades en el mundo.

Desde sus inicios, muchos satélites desarrollados por varios países han sido lanzados al espacio, entre ellos el CANX1 (Universidad de Toronto - Canadá), Quakesat (Universidad de Standford - USA), SwissCube (Instituto EPFL - Suiza), ITUpsat-1 (Turquía), BeeSat (Alemania), GOLIAT (Universidad de Bucarest - Rumanía), FASTRAC (Universidad de Texas - USA), etc. Muchos otros se encuentran en etapa de desarrollo y pronto serán lanzados al espacio, entre ellos el nanosatélite "Chasqui I" desarrollado por la Universidad Nacional de Ingeniería en Perú.

Todo satélite artificial cuenta con un sistema de comunicación, que consiste en una antena, un transceptor, un controlador de nodo terminal (TNC) y una interfaz con el procesador principal. Asimismo, como contraparte, las estaciones de comunicación en Tierra cuentan con antena, rotor, radio transceptora, TNC y computador con software [3]. La Figura 3.1 muestra un satélite y una estación.



Figura 3.1: Satélite artificial y estación terrena

La Figura 3.2 muestra un esquema con los elementos básicos de una estación terrena para recepción de señales de pequeños satélites, que incluye PC, decodificador (TNC), radio y antena con rotores y controlador. Las flechas indican cables de conexión entre los elementos y el flujo de señal entre estos elementos.



Figura 3.2: Elementos básicos de una estación terrena

Las comunicaciones satelitales emplean a los satélites como transmisores y receptores, que pueden enviar (recibir) información desde (hacia) una estación terrena. De este modo, se usan las frecuencias de radio (HF, VHF o UHF) para comunicaciones a largas distancias, con diversas aplicaciones como prueba de dispositivos electrónicos en condiciones extremas, análisis de los materiales del espacio, envío de información de emergencia, toma de fotos y hasta prevención de desastres naturales.

Dentro de la estación terrena o en el módulo de comunicación del satélite en órbita hay por lo menos un TNC, cuya función es modular (demodular) las tramas que son enviadas (recibidas), así como empaquetar los datos en dichas tramas y desempaquetar las tramas para obtener datos digitales que puedan ser procesados por el software de un computador para su visualización.

Un TNC puede fabricarse en hardware (dispositivos electrónicos interconectados), en software (programa en una PC) o en firmware (sistema embebido), dependiendo de las necesidades del proyecto.

3.2. Antecedentes

Han sido desarrollados y existen en la actualidad TNCs muy completos, que soportan muchos tipos de modos de transmisión y recepción (AFSK, FSK, PSK, GMSK por ejemplo) así como diversas bandas de frecuencia de operación (HF, VHF, UHF). Estos son del tipo hardware, software o una mezcla de ambos.

La Tabla 3.1 muestra una comparación de características de 2 ejemplos de TNC comerciales implementados en hardware y software. Se observa que los TNC en hardware suelen ser más caros que los de software, pero más confiables.

	TNC Hardware	TNC Software
Tipo	Equipo de comunicación	Licencia de software
Modelo	KAM-XL	MixW
Costo	470 USD (incluye envío)	50 USD (incluye envío)
Fabricante	Kantronics	MixW
Ventajas	Mayor rapidez, mayor confiabilidad	Menor costo, mayor flexibilidad

Tabla 3.1: Comparación entre TNCs hardware y software

3.2.1. TNCs en hardware

Un TNC puede fabricarse utilizando solamente dispositivos de hardware dedicado, es decir, procesadores, chips modem, memorias, etc. Este tipo de TNC tiene como característica principal la rapidez de procesamiento y confiabilidad en funcionamiento; generalmente esto conlleva a un costo elevado. Entre los fabricantes de este tipo de equipos tenemos a Kantronics, Symek y PacCom. Éstos TNCs soportan varios tipos de modulación, tasas de transferencia de datos, bandas de frecuencia y protocolos de encapsulamiento, y existen en diversidad de costos y materiales de construcción, desde microcontroladores y DSPs hasta FPGAs en las últimas versiones.

El TNC-2 [4] es uno de los primeros y más difundidos TNCs hardware, desarrollado por la asociación TAPR (Estados Unidos) en 1985, con la finalidad de poner al alcance de los radioaficionados un dispositivo de bajo costo que implementa el protocolo AX.25. El TNC-2 está basado en un microprocesador Zilog

Z80 y periféricos, entre los cuales están el modulador FSK XR2206, demodulador FSK XR2211, memoria EPROM 27256 y memorias SRAM 6264.

KAM-XL [5] es un módem/TNC fabricado por Kantronics y distribuido por empresas de equipos para radioaficionados como Radio Associated, que además de proporcionar los equipos brindan el soporte necesario. KAM-XL es un TNC comercial de tipo multimodo, pues permite varios modos de operación, como Packet, PSK31, RTTY, ASCII, CW, GPS NMEA-0183, KISS, entre otros. Por tanto, se puede usar en una estación terrena de comunicación para satélites de tipo aficionado, siendo necesaria una reconfiguración del equipo cada vez que se requiera comunicación con un satélite distinto. En la Figura 3.3 se muestra una foto del TNC KAM-XL.



Figura 3.3: TNC hardware KAM-XL

KAM-XL se comunica con una PC por medio de un conector DE-9 de comunicación serial RS-232; asimismo, se comunica con una radio por medio de otro conector DE-9. Las radios tienen conectores de salida tipo DIN de 6 u 8 pines para datos; por tanto, se debe fabricar el cable de conexión entre el TNC y la radio.

PK-232/DSP [6] es un TNC con módem basado en tecnología DSP, desarrollado por TimeWave, que implementa los modos AMTOR, RTTY, AX.25 HF (modulación FSK 300bps), AX.25 VHF (modulación FSK 1200bps), PACTOR, Morse. Éste TNC hardware tiene una funcionalidad especial de selección automática de filtro al ingresar el modo deseado por el terminal de comunicación del TNC en modo de configuración.

YAM (Yet Another 9k6 Modem) es un TNC basado en FPGA que apareció en 1997 como un modem FSK de 9600bps implementado en un FPGA de Xilinx. YAM en su versión actual incorpora funcionalidades de TNC y requiere de un software en la PC para manejar los datos y descargar la configuración del FPGA para uno de los 3 modos que incluye (AFSK 1200bps, FSK-Manchester 2400bps y FSK 9600bps), por el puerto serial RS-232.

3.2.2. TNCs en software

Un TNC puede fabricarse también usando como hardware solo la tarjeta de sonido de una PC, residiendo el trabajo en el software, es decir, sentencias de código almacenadas en la memoria del computador a manera de programas. En este caso, los conectores para entrada y salida de audio de la radio son ingresados a la PC por medio de la entrada de micrófono, y son obtenidos mediante la salida de audio de la PC. Se requiere circuitería adicional mínima para realizar las conexiones entre la radio y la tarjeta de sonido de la PC, como por ejemplo adaptadores de nivel de voltaje y cables con conectores especiales.

Los TNCs del tipo software suelen ser más económicos y sencillos de usar, sin embargo los TNCs del tipo hardware son más rápidos en procesamiento, por tanto son más adecuados para aplicaciones en tiempo real.

Entre los programas que cumplen funciones de TNC tenemos MixW, MMSSTV, PSK31 Deluxe, entre otros. Existen también programas que tienen funcionalidades de gestión de la información tales como Mercury y Genso, que además poseen las interfaces para la visualización de la información.

MixW [7] es un software desarrollado para controlar y procesar la información proveniente de radios de tipo aficionado, y fue desarrollado por los ucranianos Nick Fedoseev (código de radioaficionado UT2UZ) y Denis Nechitailov (código de radioaficionado UU9JDR). Este software soporta muchos modos de comunicación (SSB, AM, FM, CW, PSK31, Packet, entre otros) y muchos tipos de radios (ICOM, YAESU, KENWOOD, entre otros). La Figura 3.4 muestra una captura de pantalla del software MixW.

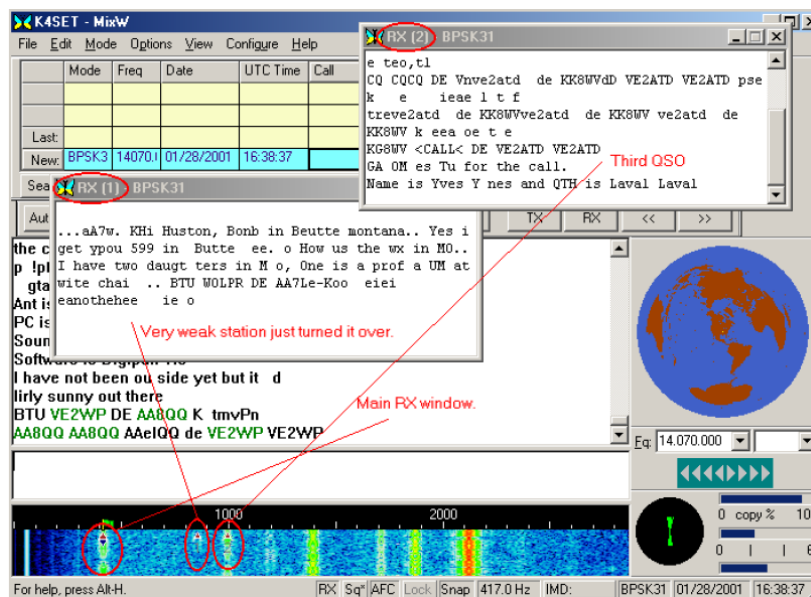


Figura 3.4: TNC software MixW

Para conectar el software MixW a una radio, se toman los terminales de audio de la radio, es decir del micrófono (incluyendo el PTT – Push To Talk) y del parlante (señal monoral - un solo canal), los cuales se conectan a la tarjeta de sonido de la PC en la que se instala el programa. Además, se puede controlar la radio o rotores de antena por medio de interfaces hardware externas como el Rig Expert Plus.

3.3. Sistemas de comunicación y arquitecturas de red

Un sistema de comunicación es el conjunto completo de entes u objetos que interactúan para el envío o recepción de un mensaje. Está formado por el emisor, el receptor, el mensaje, el canal y el código. Existen modelos de comunicación que sirven de referencia para desarrollar sistemas de comunicación complejos, como el modelo de referencia OSI y la arquitectura de red TCP/IP.

3.3.1 Modelo TCP/IP

El modelo de comunicación TCP/IP divide a los sistemas de comunicación en cuatro capas o niveles: Acceso a la red, Internet, Transporte y Aplicación. Este modelo describe el proceso de comunicación por Internet, desde la fabricación del mensaje por parte del usuario (capa de aplicación) hasta la generación de la señal

eléctrica o luminosa que transportará el mensaje codificado al destino (capa de acceso a la red).

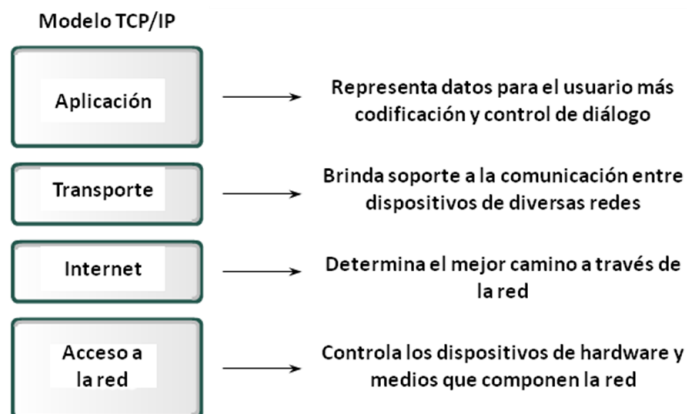


Figura 3.5: Capas del modelo TCP/IP

La Figura 3.5 muestra las capas o niveles del modelo de comunicación TCP/IP, que fuera definido incluso antes que el modelo de referencia OSI. Se puede explicar un proceso de comunicación mediante este modelo como el transporte del mensaje generado en la capa más alta del emisor (aplicación) hacia la capa inferior donde llega al medio de transmisión, para luego realizar el proceso inverso de ascensión hasta la capa más alta del receptor quien espera recibir el mensaje sin errores.

Los switches de comunicación Ethernet comprenden las capas física y de acceso a la red, mientras que los routers (equipos que interconectan redes de computadores) comprenden, además de las 2 anteriores, la capa de red.

3.3.2 Modelo X.25

X.25 es un protocolo de acceso a una red de conmutación de paquetes recomendado por la CCITT para interfaces entre un DTE y un DCE sobre una red pública de datos (PDN). Consta de los 3 primeros niveles del modelo de referencia OSI.

La Figura 3.6 muestra los protocolos del modelo de comunicación X.25 en el marco de referencia del modelo OSI. Se observa que implica un diseño de capas 1, 2 y 3, lo que quiere decir que presenta las características de direccionamiento en una red, enlace de datos y física.

LAPB es un protocolo de capa 2 que se usa para transportar paquetes X.25. El formato de una trama LAPB estándar está formado por flags de inicio y fin, campos de dirección y control, los datos de información y el FCS (Frame Check Sequence). La Figura 3.7 ilustra el contenido de una trama LAPB.

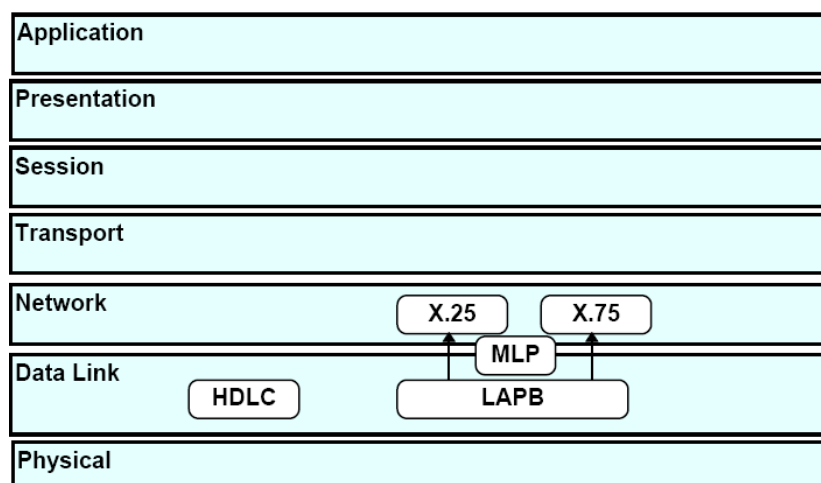


Figura 3.6: Capas del modelo X.25

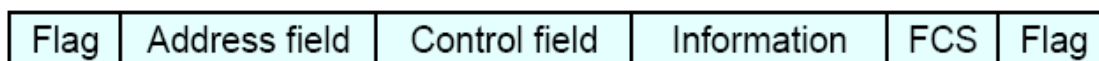


Figura 3.7: Campos de una trama LAPB

El valor del FLAG es siempre 0x7E (01111110). Como la comunicación es serial síncrona, para asegurar que los flag delimitadores no se repitan en el campo de datos de la trama, se usa la técnica denominada Bit Stuffing.

El primer byte después del flag de cabecera es el campo de dirección. En LAPB este campo no tiene significado, porque el protocolo trabaja en modo punto a punto, y la dirección de red del DTE se representa en los paquetes de nivel 3. Por ello se usa de manera distinta: es 0x01 cuando la trama contiene comandos de DTE a DCE o sus respuestas, ó es 0x03 cuando la trama contiene comandos de DCE a DTE o sus respuestas.

El campo de control sirve para identificar los tipos de trama. Incluye números de secuencia, características de control y monitoreo de error de acuerdo al tipo de trama. LAPB trabaja en el modo ABM (Modo Asíncrono Balanceado), es decir, no hay maestro ni esclavo y el DTE y DCE son tratados por igual.

La Secuencia de Chequeo de Trama (FCS) permite verificar la integridad de los paquetes recibidos. La secuencia se calcula por el transmisor a partir de todos los bits de la trama, el receptor realiza la misma operación y compara los valores.

LAPB soporta un tamaño de ventana desde 8 hasta 128, lo que significa que cada 8 a 128 paquetes enviados el transmisor espera un acuse de recibo para seguir enviando o repetir la transmisión de tramas que se han perdido. Por tanto, la estructura de un paquete X.25 puede ser de módulo 8 o 128.

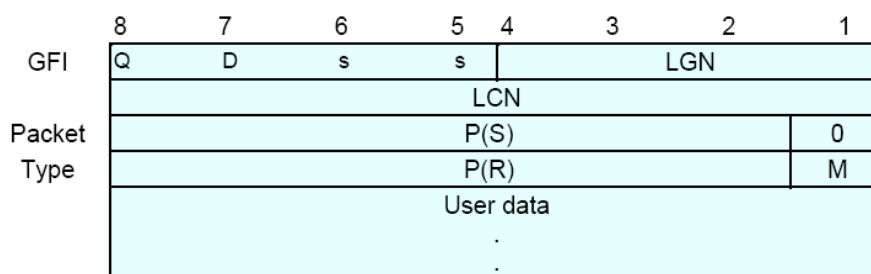


Figura 3.8: Contenido de un paquete X.25

La Figura 3.8 muestra la estructura de un paquete X.25, formado por 4 bytes sin considerar los datos de información que en conjunto pueden hacer un total desde 8 a 128 bytes. Los primeros 4 bytes corresponden a información de control como identificadores, direcciones y número de secuencias.

GFI es el identificador de formato general. Los bits ss indican si la trama es de módulo 8 (ss=01) o de módulo 128 (ss=10). LGN es el número de grupo de canal lógico y es un campo de 4 bits que junto al LCN identifica el número de canal lógico actual. LCN es el número de canal lógico y es un número de 8 bits que identifica el número de canal actual del enlace DTE-DCE.

En módulo 128, 16 bits identifican el tipo de paquete. P(R) denota el número de secuencia de paquete recibido (el cual aparece en paquetes de control de flujo y de datos) o la dirección del DTE en configuración de llamada. P(S) denota el número de secuencia de paquete enviado el cual aparece en paquetes de datos o la dirección del DTE en configuración de llamadas.

3.4. Comunicaciones en pequeños satélites

Para el monitoreo de pequeños satélites, el envío de comandos y la recepción de datos del mismo, se necesita de un ambiente con equipos y dispositivos adecuados; este recibe el nombre de estación terrena. Ésta consiste,

como mínimo, en una antena con controladores de orientación de elevación y azimuth denominados rotores, una radio transceptora (esto quiere decir que funciona como transmisor y receptor), un TNC y una computadora con software de interface para el usuario final.

Una estación terrena se ubica en una zona despejada, donde las antenas puedan tener un mayor ángulo de cobertura sin interferencia. Las estaciones terrenas para satélites de tipo aficionado deben tener el equipamiento necesario para tener capacidad de comunicarse con la mayoría de los satélites puestos en órbita, con una inversión que sea razonable, pues los equipos que utilizan tienen un costo relativamente elevado [8].

3.5. Modulaciones digitales

La modulación es un proceso mediante el cual una información en forma de señal eléctrica es convertida a una forma que pueda ser transmitida con mayores ventajas. Por ejemplo, una señal de audio comprende frecuencias a las cuales es imposible una transmisión a distancias grandes, por la limitación en el tamaño de la antena que sería necesaria para ello. Sin embargo, si se usa una señal con una frecuencia mucho más grande de, y de modo tal que dicha señal de alta frecuencia varíe uno de sus parámetros en función de nuestra señal de audio que contiene la información, se hace posible su transmisión a lugares lejanos.

La modulación de una señal puede ser analógica o digital, dependiendo de la naturaleza de la señal de entrada. Si la información está contenida en una señal analógica hablamos de modulación analógica, tal es el caso de modulación en amplitud (AM), en frecuencia (FM) o en fase (PM).

Si la señal de información es digital, entonces hablamos de formas de modulación digital. También se pueden clasificar, según el parámetro de variación de la señal portadora, que es una señal analógica de onda senoidal, en ASK (Amplitude Shift Keying), FSK (Frequency Shift Keying) y PSK (Phase Shift Keying), denotando un desplazamiento en la amplitud, fase o frecuencia respectivamente, de la señal portadora [9].

3.5.1 Modulación ASK

ASK es un tipo de modulación digital donde la señal portadora es una onda senoidal cuya amplitud varía en forma discreta en función del valor de la señal

digital que contiene la información. Cuando la señal de entrada es binaria (niveles lógicos 0 y 1), la señal modulada en ASK está formada tramos de señal senoidal de amplitud distinta en dos posibles valores, fases y frecuencias iguales. Una ecuación que define una señal ASK se muestra en la ecuación (3.1).

$$s(t) = (A_0 \overline{b[n]} + A_1 b[n]) \text{sen}(2\pi ft + \phi) \quad (3.1)$$

La Figura 3.9 muestra una señal binaria de entrada a un modulador ASK, y la señal de salida correspondiente, que es el producto matemático de la señal binaria y una señal senoidal correspondiente a la portadora.

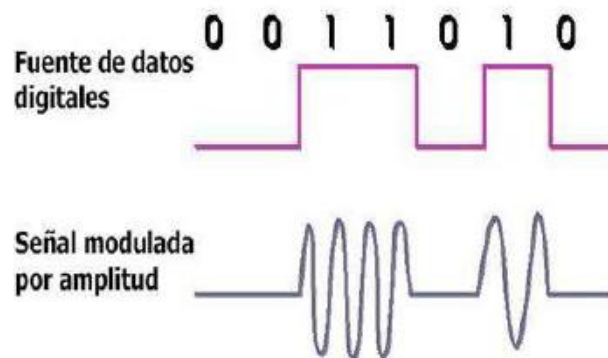


Figura 3.9: Señal modulada en ASK

Un circuito modulador en ASK binario se puede construir con un chip generador de señal senoidal y un interruptor controlado, que conecte y desconecte la señal senoidal generada en función de los datos digitales que ingresen al modulador.

3.5.2 Modulación FSK

FSK es un tipo de modulación digital donde la señal de portadora es una señal senoidal cuya frecuencia varía en forma discreta en función del valor de la señal digital que contiene la información. Cuando la señal de entrada es binaria, la señal modulada en FSK está formada por tramos de dos señales senoidales de distinta frecuencia pero amplitud y fase iguales, como se muestra en la ecuación (3.2).

$$s(t) = A \text{sen}(2\pi (b[n]f_0 + \overline{b[n]}f_1)t + \phi) \quad (3.2)$$

La Figura 3.10 muestra una señal binaria de entrada a un modulador FSK y la señal de salida correspondiente, que es una onda senoidal de frecuencia desplazable en función de la señal de entrada.

Un circuito modulador FSK se puede construir con 2 chips generadores de señal senoidal a frecuencias distintas y un dispositivo que permita seleccionar una u otra señal a la salida en función de la señal digital de entrada.

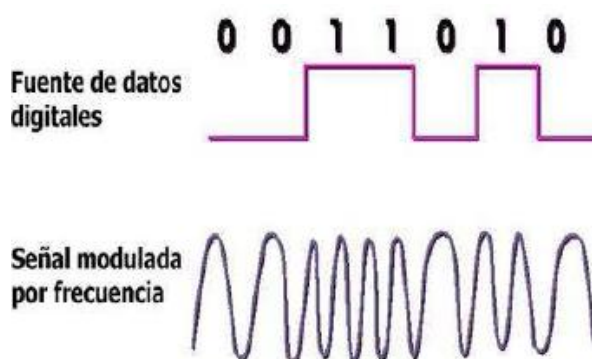


Figura 3.10: Señal modulada en FSK

3.5.3 Modulación PSK

PSK es un tipo de modulación digital donde la señal resultante consiste en tramos de señales senoidales con la misma amplitud y frecuencia, pero con distintas fases en función directa del valor de la señal digital de entrada.

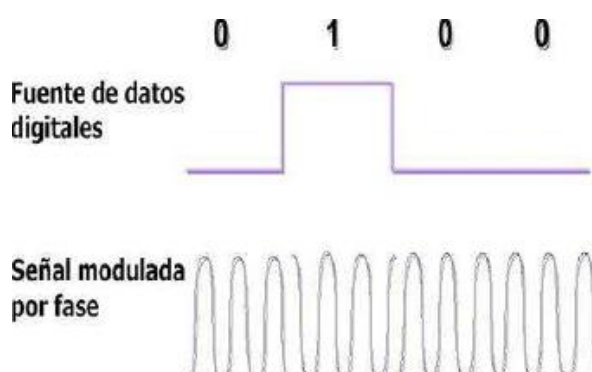


Figura 3.11: Señal modulada en PSK

La Figura 3.11 muestra una señal binaria de entrada a un modulador PSK y la señal de salida correspondiente, que es una onda senoidal de fase desplazable en función de la señal de entrada, y la fórmula está en la ecuación (3.3).

$$s(t) = A \sin(2\pi ft + \phi_0 + \sum_{n=1}^{\infty} \phi_n b[n]) \quad (3.3)$$

Un circuito modulador PSK se puede construir con un chip generador de señal senoidal, un generador de retardo de fase y un dispositivo selector controlado por la señal digital de entrada.

3.5.4 Modulación GMSK

GMSK es un método de modulación digital para enviar datos en alta rapidez a través de canales de radio FM de banda angosta. En su forma más simple consiste en pasar una cadena de bits por un filtro Gaussiano antes de aplicarlo a un modulador de frecuencia.

Un filtro gaussiano es un elemento pasa-bajas, que al ser excitado por entradas de tipo impulso, genera a la salida una respuesta de forma gaussiana. La Figura 3.12 muestra el diagrama de bloques de la modulación GMSK.

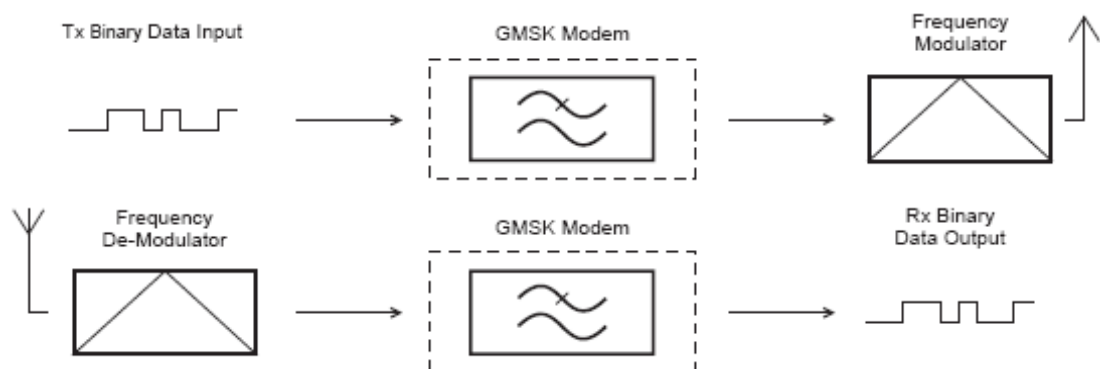


Figura 3.12: Modulación y demodulación GMSK

Para implementar una transmisión GMSK, una señal digital se ingresa a un filtro gaussiano. La salida del filtro gaussiano pasa por un modulador FM con una respuesta en fase continua, con una desviación de pico igual a la mitad de la tasa de bits de los datos digitales de entrada.

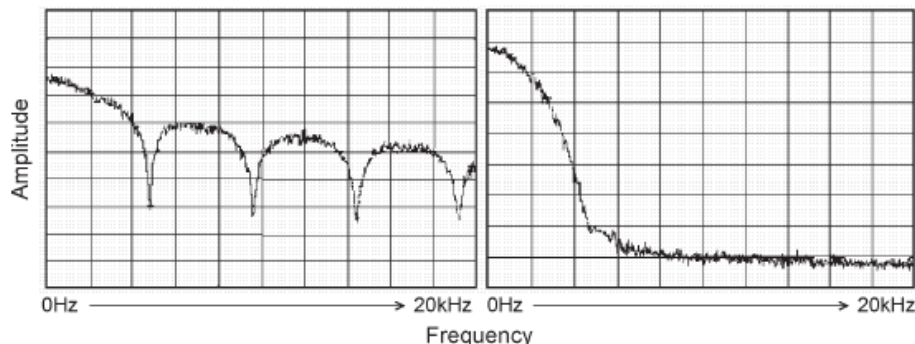


Figura 3.13: Espectro de una señal modulada en GMSK

En la Figura 3.13 se muestran las respuestas en frecuencia de las modulaciones FSK simple y GMSK, donde se observa que el efecto del filtro gaussiano presente en el modulador GMSK es de reducir el ancho de banda requerido para la transmisión.

3.6. Codificaciones usadas en satélites

3.6.1 Codificación Morse

La codificación Morse fue ampliamente utilizada junto con el telégrafo, aunque su uso actual se extiende solo a los radioaficionados y radioaficionados por satélites [10]. La Unión Internacional de Telecomunicaciones (UIT, o ITU por sus siglas en inglés) generó la recomendación ITU-R-M1677 en el 2004, donde actualiza los equivalentes en código Morse de las letras, cifras y otros signos, además de otras recomendaciones. Un resumen de esta recomendación sobre el código Morse se encuentra en el Anexo A.

3.6.2 Codificación SSTV

Slow-Scan Television (SSTV) es un método de transmisión de imágenes usado por radioaficionados, el cual usa un ancho de banda mucho menor que el de la TV convencional. Desafortunadamente, existen muchas variaciones de SSTV y la información referente a ellas es imprecisa, aunque ya se han dado propuestas para estandarizar dichos modos [11].

3.7. Protocolos de comunicación en TNCs

3.7.1 Protocolo AX.25

Amateur X.25 (AX.25) es un protocolo de acceso al medio desarrollado por TAPR con la finalidad de crear una red de radioaficionados para comunicar estaciones terrenas, satélites de investigación e incluso lograr conectividad con Internet [12]. Actualmente se ha reducido su uso al envío de tramas no numeradas por pequeños satélites y al uso en estaciones terrenas como sistema de reconocimiento de posición. En el anexo B se resume la especificación 2.2 del protocolo AX.25, implementado total o parcialmente en todos los TNCs.

3.7.2 Protocolo KISS

KISS (“Keep It Simple, Stupid”) [13] es un protocolo de comunicación entre la PC y el TNC, implementado en la mayoría de TNCs existentes en la actualidad por su simplicidad y utilidad. KISS consiste en una trama asíncrona equivalente a la trama HDLC que reciben los TNCs, sin cabeceras ni bytes de detección de error. Es análogo al protocolo serial IP en el sentido que representa a una trama asíncrona y puede ser usado en PCs que no dispongan del hardware necesario para recibir directamente la trama asíncrona. Asimismo, existen programas de software como UI-View que son compatibles con el protocolo KISS.

3.7.3 Protocolo APRS

El sistema de reporte de posición automático (APRS, por sus siglas en inglés) es un protocolo de comunicación de paquetes para difundir datos a todos los elementos de una red en tiempo real [14]. Su característica más visual es la combinación de radio paquetes con el sistema de posicionamiento global (GPS, por sus siglas en inglés), haciendo posible que los radioaficionados muestren las posiciones de las estaciones de radio y otros objetos sobre mapas en un computador. La estructura de la trama APRS se muestra en la Figura 3.14.

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

Figura 3.14: Trama AX.25 no numerada usada para APRS

3.8. Análisis y estimación del efecto Doppler

Cuando un satélite de radioaficionado orbita alrededor de la tierra, se debe establecer comunicación con él para recibir la señal “beacon”, enviar comandos y/o descargar datos. Sin embargo, la frecuencia de las señales de RF emitidas por el satélite no es la misma para el receptor en tierra, y varía permanentemente durante el tiempo que el satélite es alcanzable desde Tierra. Del mismo modo, cuando la estación terrena transmite hacia el satélite, la frecuencia de las señales RF que llegan al satélite no es igual (para el satélite) a la frecuencia de las que se emiten. Este problema es conocido como el efecto Doppler. Una aplicación directa es implementar un algoritmo dentro del TNC de la estación terrena que compense dicho desplazamiento de frecuencia, o establecer la comunicación de los equipos con un software que realice dicha compensación de frecuencia.

3.8.1 Análisis y simulación del Efecto Doppler

Cuando un emisor de ondas se mueve respecto del receptor, éste último percibe una frecuencia de onda variable dependiendo si el emisor se acerca o se aleja de él. Esta diferencia entre frecuencias de ondas emitida y recibida se debe al fenómeno descrito: el efecto Doppler.

Efecto Doppler en ondas de sonido

Cuando el emisor de ondas de sonido se acerca al receptor, comprime los frentes de onda que van hacia el receptor, ocasionando un aumento en la frecuencia de la onda de sonido percibida por el receptor. De igual manera, cuando la fuente se aleja del receptor, las ondas de sonido que van hacia el receptor se separan ocasionando una disminución en la frecuencia recibida [15]. En la Figura 3.15 se muestra una representación geométrica del efecto Doppler, y las ecuaciones que se deducen de ella son (3.4), (3.5) y (3.6).

a) Fuente móvil y receptor estático

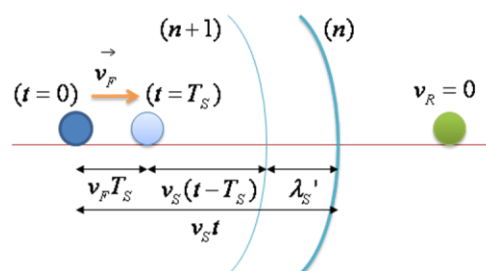


Figura 3.15: Representación geométrica del efecto Doppler

$$f_s' = \frac{v_s}{v_s - v_F} f_s, \quad (3.4)$$

En la ecuación (3.4):

f_s es la frecuencia del sonido

v_s es la rapidez del sonido

v_F es la rapidez de la fuente

v_R es la rapidez del receptor

b) Fuente estática y receptor móvil

$$f_s' = \frac{v_s + v_R}{v_s} f_s \quad (3.5)$$

c) Fuente móvil y receptor móvil

$$f_s' = \frac{v_s + v_R}{v_s - v_F} f_s \quad (3.6)$$

Efecto Doppler en satélites artificiales

Si la fuente es un satélite artificial y el receptor es una estación terrena, entonces $v_r=0$. Las ondas son electromagnéticas, por tanto $v_s=c$ (la rapidez de la luz en el vacío). v_f es la rapidez de traslación del satélite artificial, que se considera mucho menor que la rapidez de la luz, como se muestra en la ecuación (3.7).

$$f' = \frac{c}{c - v} f, \text{ entonces: } \Delta f = \frac{v}{c - v} f \quad (3.7)$$

La ecuación (3.7) es válida cuando la fuente se aproxima al receptor en línea recta, por tanto se requiere de una expresión más general, cuando la fuente se mueve en una trayectoria cualquiera. En este caso, la componente de velocidad en la línea que une la fuente y el receptor determina el desplazamiento de frecuencia por efecto Doppler. La Figura 3.16 muestra una representación vectorial del efecto Doppler, cuya ecuación resultante se observa en (3.8).

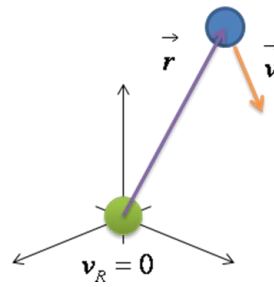


Figura 3.16: Representación vectorial del efecto Doppler

$$\Delta f = - \frac{\vec{v} \cdot \vec{r}}{c |\vec{r}|} f, \quad (3.8)$$

Donde:

f es la frecuencia de la señal emitida por el satélite

v es el vector velocidad de traslación del satélite

r es el vector posición del satélite respecto de la estación terrena

c es la rapidez de la luz en el vacío

Se puede entonces calcular la variación de la frecuencia emitida por un satélite móvil y la recibida por una estación terrena. Se asume que la rapidez de desplazamiento del satélite artificial es aproximadamente constante, y se obtiene una buena aproximación cuando el satélite pasa sobre la estación terrena. La Figura 3.17 muestra una representación gráfica del efecto Doppler en un satélite, y la ecuación (3.9) muestra la fórmula obtenida.

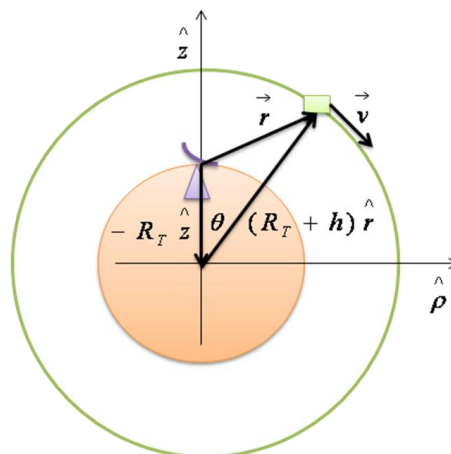


Figura 3.17: Efecto Doppler en un satélite de órbita circular

$$f' = \frac{f}{1 - \frac{v}{c} \frac{R_T \sin \theta}{\sqrt{(R_T + h)^2 + R_T^2 - 2(R_T + h)R_T \cos \theta}}}, \quad (3.9)$$

Donde:

$$\theta = \frac{vt}{R_T + h} < \arccos\left(\frac{R_T}{R_T + h}\right)$$

Simulación del efecto Doppler en la ISS

La estación espacial internacional (ISS) es un satélite artificial multinacional y tripulado que orbita a una altura media de 415Km sobre la tierra. Sus demás parámetros se pueden encontrar en la web.

$$f_0 = 436 \text{ MHz}$$

$$v \approx 27500 \text{ Km / h}$$

$$h \approx 415 \text{ Km}$$

$$c = 300000 \text{ Km / s}$$

$$R_E \approx 6371 \text{ Km}$$

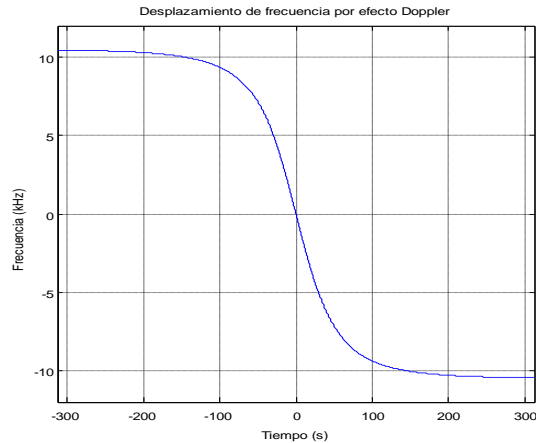


Figura 3.18: Gráfico teórico del desplazamiento de frecuencia Doppler

En la Figura 3.18 se muestra la gráfica teórica. Para contrastar la fórmula hallada se tomaron datos del software Orbitrón, que predice entre otros parámetros la frecuencia de desplazamiento por efecto Doppler en los satélites. En la Figura 3.19 se muestra los datos experimentales (tomados de Orbitrón) en línea de color rojo y los datos analíticos calculados con la fórmula teórica en color azul (el tiempo cero se da cuando el satélite está en su posición más cercana a la estación terrena

para una órbita determinada). Se puede observar que la fórmula teórica es una muy buena aproximación cuando el satélite pasa sobre la estación terrena.

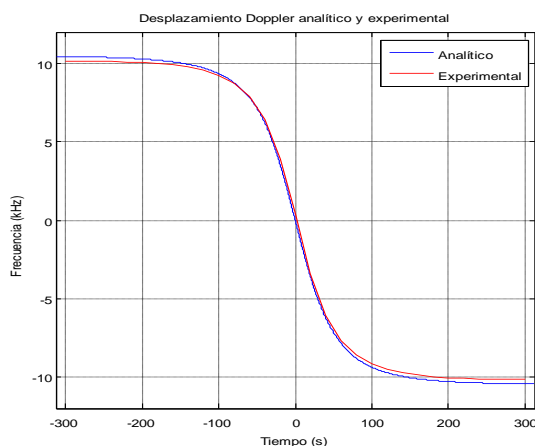


Figura 3.19: Gráficas teórica y experimental del desplazamiento Doppler

3.8.2 Estimación del Efecto Doppler

Otra forma de hallar la ecuación que describe el efecto Doppler en satélites consiste en tomar muestras experimentales y realizar una estimación de una ecuación que tenga error mínimo respecto de los datos de muestra.

El software Orbitrón es libre de uso y sirve para predecir, entre otros parámetros, la frecuencia de recepción considerando el efecto Doppler. Se tomaron datos para varios pasos de la ISS con una precisión de 20 seg por muestra. La Figura 3.20 muestra capturas de pantalla del software Orbitrón.

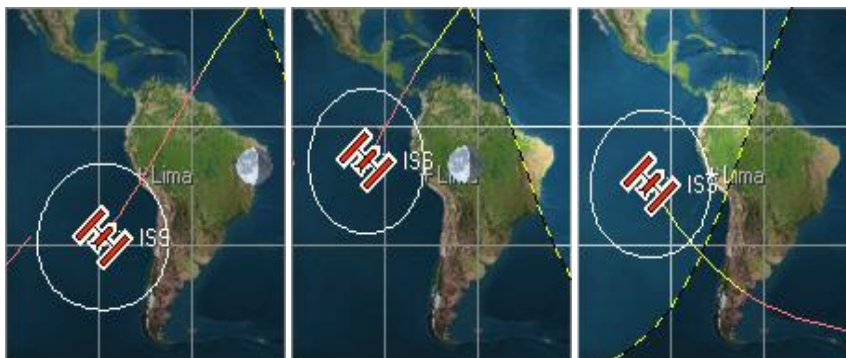


Figura 3.20: "Pasadas" de la ISS muestreadas con Orbitron

El software Eureka es libre de uso y sirve para estimar leyes físicas usando algoritmos genéticos, a partir de datos muestreados. Se ingresaron los datos muestreados y se dejó procesando por 8 horas, de las soluciones se tomó una sencilla y precisa.

En el cálculo de la ecuación se consideraron dos parámetros: tiempo (cero cuando el satélite está más cerca de la estación terrena) y elevación máxima (ya que el satélite no siempre pasa exactamente por encima de la estación terrena). Los resultados se contrastaron con los originales y con otros datos de muestra nuevos, obteniéndose un error menor de 2 kHz.

$$\Delta f = \text{sen}(-0.0235 t) - \text{sen}(0.0116 t) - 0.0713 E \text{sen}(0.0116 t) - 0.0362 t ,$$

Donde:

t: tiempo medido desde ángulo de elevación cero, donde es negativo.

E: ángulo de elevación máximo, cuando el satélite está más cerca.

A continuación se muestran gráficas de los datos muestreados (color azul) con ayuda del software Orbitron y los datos estimados (color rojo) con ayuda del software Eureka. El eje de abscisas está en segundos y el eje de ordenadas está en kHz. Una captura de pantalla del software Eureka se muestra en la Figura 3.21.

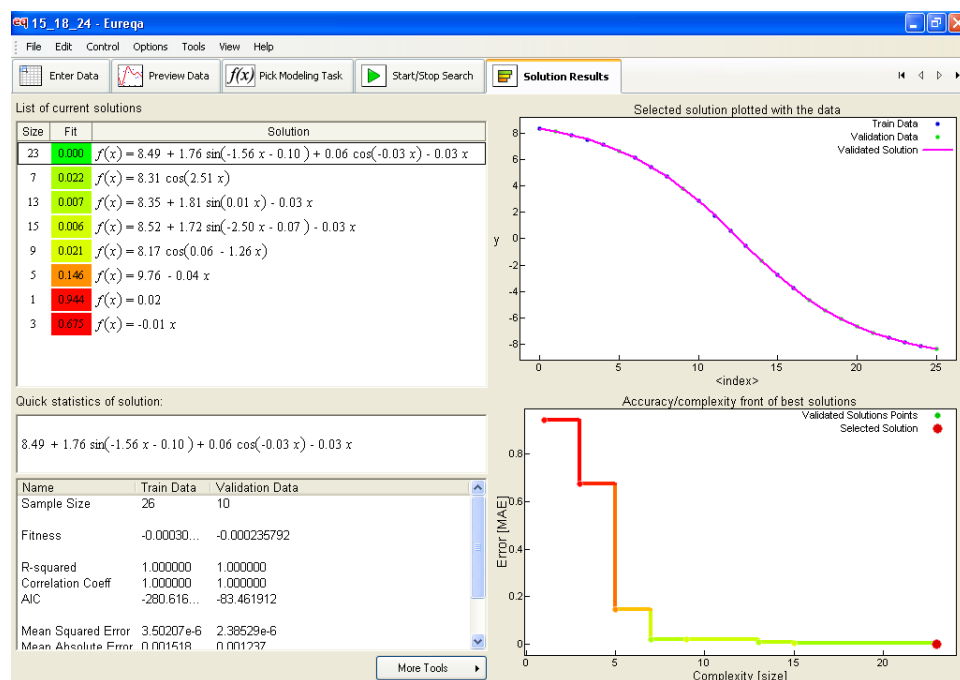


Figura 3.21: Captura de pantalla del software Eureka

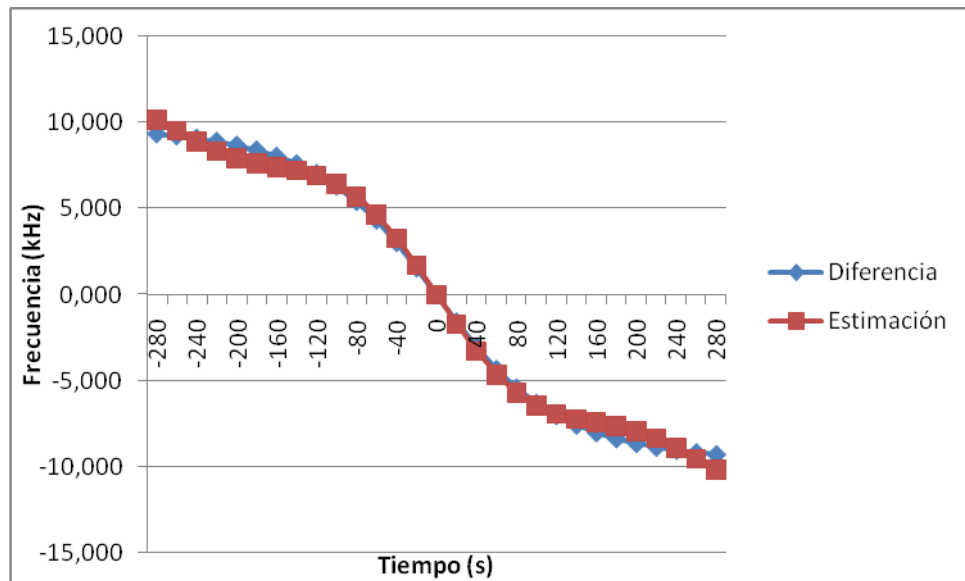


Figura 3.22: Comparación entre datos estimados y experimentales

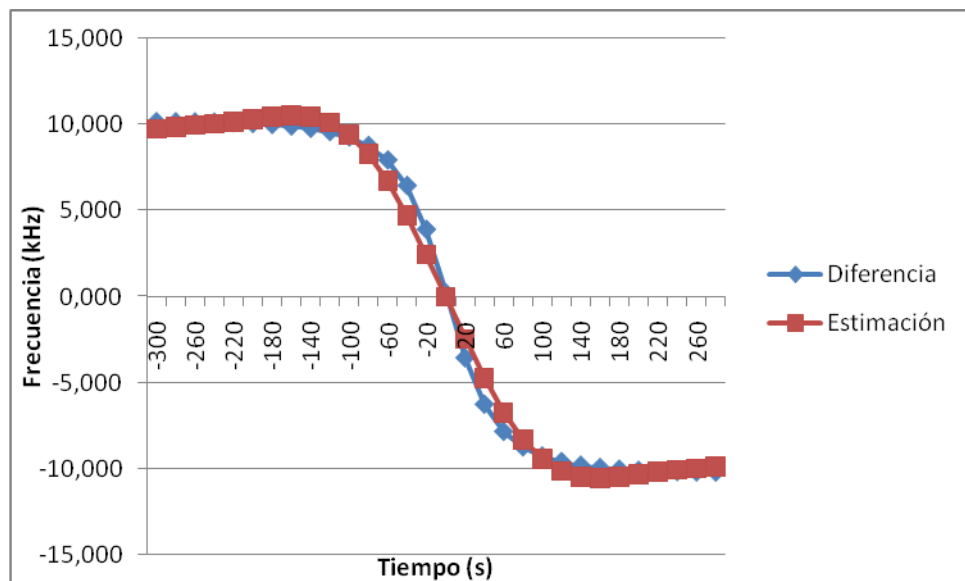


Figura 3.23: Comparación entre datos estimados y experimentales

Para una muestra de menor resolución no considerada en el cálculo:

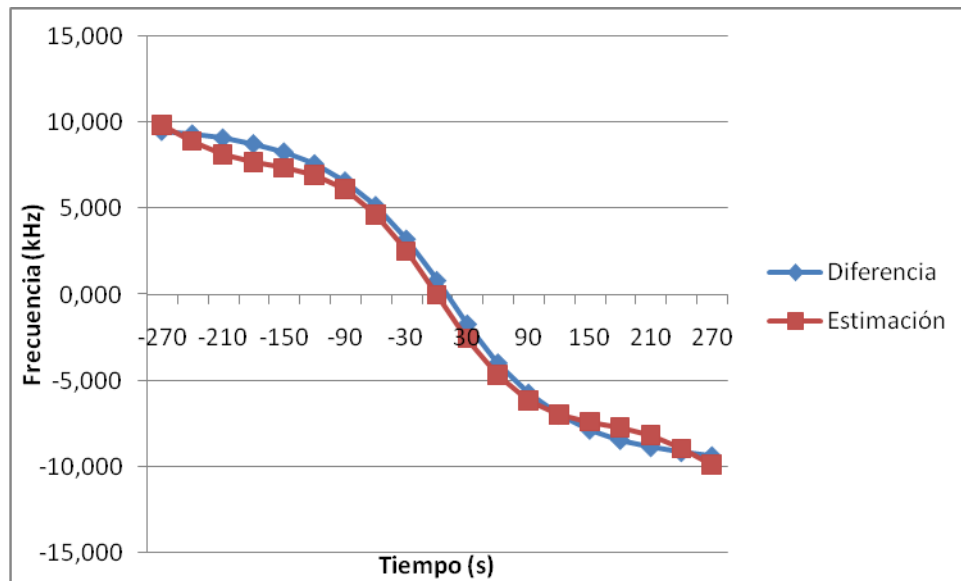


Figura 3.24: Comparación entre datos estimados y experimentales

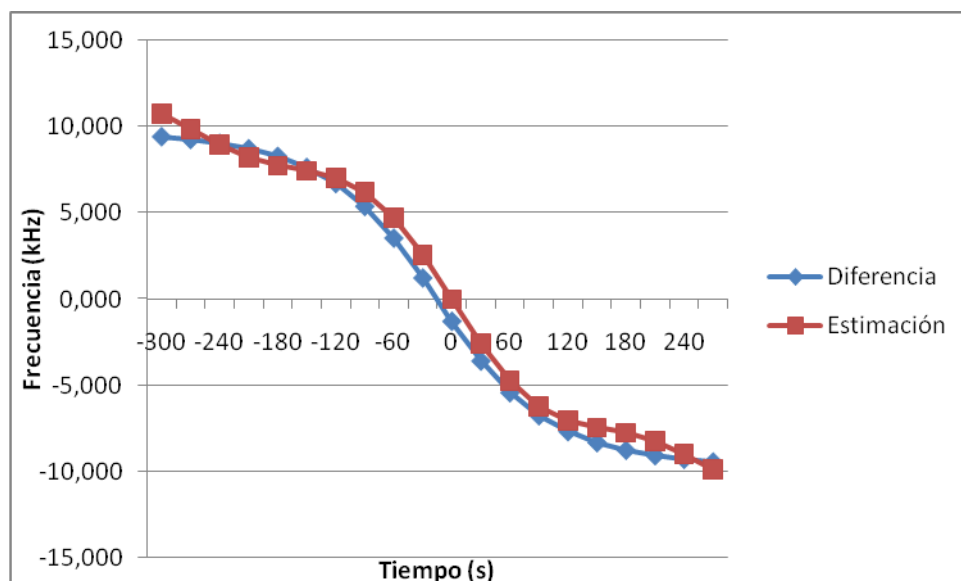


Figura 3.25: Comparación entre datos estimados y experimentales

Las Figuras 3.21, 3.22, 3.23 y 3.24 grafican los resultados obtenidos. En conclusión, se encontró una ecuación que aproxima el comportamiento de la variación de la frecuencia de recepción (y transmisión) de información por efecto Doppler. Por tanto, se puede implementar un mecanismo de compensación del efecto Doppler, a fin de modificar automáticamente la frecuencia de recepción (y transmisión) de la radio, y establecer una comunicación adecuada entre un satélite y la estación terrena. Dicho mecanismo puede ser implementado en código Java

para ser insertado en el software de gestión de la estación terrena, o implementado dentro del TNC multimodulación.

3.9 Órbitas de satélites

Un satélite artificial puede seguir una órbita de tipo LEO (órbita baja, casi circular, a una altura de entre 80 km y 2000 km), MEO (órbita media, entre 2000 km y 35786 km), GEO (órbita geocéntrica, rota junto con la tierra, a una altura de 35786 km) y HEO (órbita alta, de forma elíptica, a una altura de más de 35786 km).

Los ángulos de azimuth y de elevación se usan para ubicar un satélite artificial respecto de una estación terrena en un instante dado. El ángulo de elevación está formado por el plano horizontal donde se encuentra el observador en tierra y la línea de vista que une al observador y el satélite. El ángulo de azimuth está formado por la dirección de referencia Oeste y la proyección de la línea de vista sobre el plano del observador, en el sentido que va desde el Oeste hacia el Sur [7].

La Figura 3.26 muestra un satélite pequeño orbitando alrededor de la Tierra (lado izquierdo) y el satélite pequeño localizado con los ángulos de azimuth y de elevación (lado derecho). De la figura se observa que si el ángulo de elevación es positivo, el satélite está disponible para lograr comunicación con la estación terrena, mientras que el ángulo de azimuth puede variar entre 0° y 359° . También se observa que el ángulo de azimuth puede cambiar de 359° a 0° , debido a que el satélite se desplazará al seguir su órbita baja, a diferencia del ángulo de elevación, que aumenta desde 0° hasta un valor máximo y retorna nuevamente hasta 0° hasta que se pierde la comunicación.

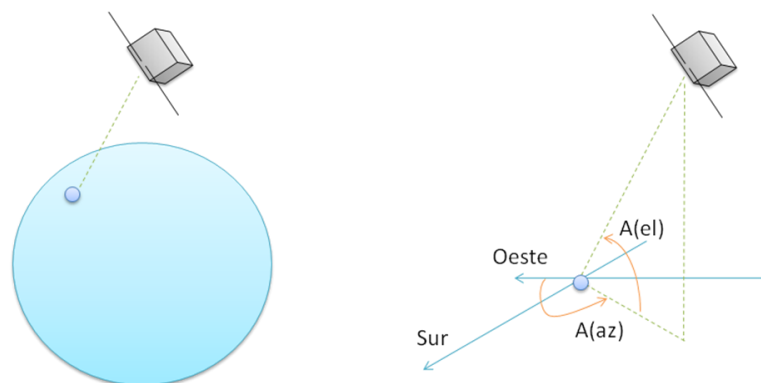


Figura 3.26: Ángulos de azimuth y elevación

CAPÍTULO IV

PLANTEAMIENTO DEL PROBLEMA

4.1. Descripción de la problemática

En la actualidad, muchos satélites de radioaficionados han sido lanzados al espacio y se puede establecer comunicación con ellos, por tanto obtener información que puede ser útil para aplicaciones de interés. Sin embargo, cada uno de estos satélites usa un modo de comunicación en particular, por lo tanto requiere un equipamiento especial si se desea establecer comunicación con él. Se requiere entonces de equipos que soporten la mayor cantidad de modos de comunicación como sea posible.

Asimismo, la gran cantidad de satélites de tipo aficionado puestos en órbita ocasiona que, en ciertos momentos, se tenga más de un satélite en la zona de cobertura de la estación terrena. Se requiere por tanto, maximizar la eficiencia de la estación terrena; es decir, que esté en contacto con algún satélite la mayor cantidad de tiempo posible, lo que hace necesario reconfigurar los equipos para poder mantener la obtención de datos. Para poder reconfigurar el modo de comunicación de los equipos, se requiere hacer un envío de comandos por teclado o un ajuste de jumpers en el equipo. Se necesita entonces, que el equipo se reconfigure lo más rápido posible debido al corto tiempo de cobertura de la antena de una estación.

Soluciones existentes

Existen TNCs comerciales hechos por hardware y software, en variedad de capacidades y precios, los cuales están equipados para soportar casi todos los tipos de modulación, tasas de bits y bandas de frecuencia existentes.

Existen también soluciones modernas, que emplean tecnologías actuales como las modulaciones por espectro ensanchado para transmisión segura, los dispositivos FPGA para una gran rapidez de procesamiento, y los dispositivos en banda S para comunicaciones a grandes distancias.

Necesidades actuales

Si se cuentan con TNCs multimodales, es deseable poder monitorear varios satélites amateur, incluso con modos diferentes de comunicación (uno en FSK y el otro en PSK por ejemplo), de manera automática. Sin embargo, el cambio en la

configuración del modo de operación de un TNC es manual, tomando tiempo valioso en que el canal está abierto para comunicación, por lo que una reconfiguración automática (en tiempo real) es muy necesaria.

En las Figuras 4.1 y 4.2 se muestra un escenario donde se requieren las funcionalidades descritas. La estación terrena 1 establece comunicación con el satélite 1 y la mantiene por un determinado tiempo. Antes de que esta comunicación finalice, el satélite 2 ya está en la zona de cobertura de la estación 1.

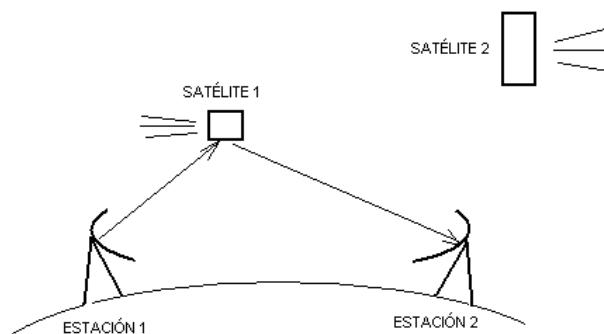


Figura 4.1: Satélite 1 comunicándose con 2 estaciones terrenas en red

Cuando finalice la comunicación de la estación terrena 1 con el satélite 1, se debe establecer inmediatamente comunicación con el satélite 2, porque no se dispone de mucho tiempo hasta que éste salga de la zona de cobertura de la estación terrena 1. Los equipos que conforman la estación terrena 1 entonces deben tener capacidad de selección automática.

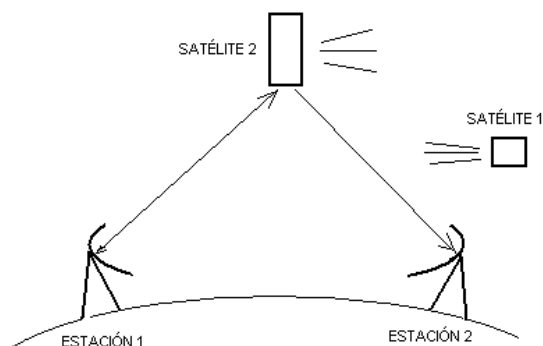


Figura 4.2: Satélite 2 comunicándose después de alejarse el satélite 1

4.2. Alternativas de solución

Existen varias alternativas de solución, según los tipos de equipos que se tomen en cuenta. Así tenemos: solución por software, solución por hardware, o una combinación de éstos (en lo que se denomina solución por firmware). Cada uno de ellos tiene ventajas respecto de los otros dependiendo de la aplicación y del objetivo que se desea alcanzar.

Alternativa por software

Una solución por software consiste en usar la tarjeta de sonido de una PC, y mediante algún programa de código modificable, manejar la tarjeta de sonido para poder trasladar los datos provenientes de la radio, realizar el procesamiento de codificación AX.25 y mostrar los datos en pantalla.

En el sistema operativo Linux se cuenta con librerías que implementan el protocolo AX.25 de modo que se pueden desarrollar programas para controlar la tarjeta de sonido del computador y procesar la información recibida. Es así que esta es una forma de desarrollar un TNC implementado en software.

Entonces, la radio debiera proveer un indicador cuando más de un satélite de tipo aficionado se encuentre en la zona de cobertura de la estación terrena. Este indicador ingresaría por la tarjeta de sonido; el mismo programa detectaría dicho mensaje y en base a algún criterio devolvería el comando necesario a la radio para realizar el cambio de modo de manera automática.

La ventaja de este método es que requiere pocos materiales y poca inversión económica, además de contar con la ideología colaborativa de Linux, por la cual se puede obtener soporte en la red. Sin embargo, por las características inherentes al software, se requiere una inversión de tiempo considerable en la elaboración del código del programa y el procesamiento de la información es lento; como el tiempo de conexión con un satélite no es grande, los retardos que se tendrán son considerables.

Alternativa por hardware

Una solución por hardware consiste en seleccionar dispositivos de hardware que realizan funciones de un TNC y realizar la circuitería necesaria para lograr la compatibilidad en la interconexión de estos dispositivos.

En principio, un TNC está formado por varios chips modulador-demodulador y un codificador-decodificador, más elementos adicionales como interfaces a la radio y la PC. En el caso de selección automática, se debe agregar un circuito que detecte el indicador de la radio de que hay más de un satélite en la zona de cobertura y que envíe la orden necesaria para realizar el cambio en el modo de recepción de la radio.

Este método tiene como ventaja el estar formado por componentes dedicados que realizan sus funciones en tiempos muy cortos, por lo que el procesamiento se hace en tiempo real. Sin embargo, se requiere una gran inversión económica para la adquisición de equipos de los cuales se obtengan los módulos componentes del TNC deseado.

Alternativa híbrida

Una solución híbrida consiste en usar dispositivos programables, tales como microcontroladores o DSP's, donde reside de manera embebida el sistema que cumple las funciones de un TNC. Este firmware es generado por una herramienta de software, y luego es grabado en la memoria del dispositivo de manera permanente. En este tipo de solución se usan los conversores A/D y filtros del dispositivo para modular y demodular la señal; asimismo se usa el procesador y la memoria interna del dispositivo para realizar la codificación y decodificación, y algún otro procesamiento como la selección automática. La ventaja de este método es que se combinan las propiedades de flexibilidad de la solución por software con la rapidez de procesamiento de la solución por hardware.

Una solución enteramente firmware no es conveniente del todo, debido a que hay chips dedicados que realizan mejor el filtrado y la modulación de la señal de entrada. La codificación de los datos es una tarea propia de un sistema embebido, donde se aprovechan todas las potencialidades del firmware. Asimismo, como el circuito electrónico suele conectarse a un computador con software, se pueden implementar ciertas funcionalidades en el software del computador para liberar de carga al dispositivo programable.

CAPÍTULO V

SOLUCIÓN PROPUESTA

Se plantea una solución híbrida entre hardware y software. El TNC multimodulación desarrollado está formado por hasta cuatro módems, selectores analógicos, selectores digitales y dos microcontroladores. La parte hardware consiste en los circuitos módem y la parte software en los algoritmos para cada microcontrolador, además de los algoritmos para la PC que se conectará al TNC para mostrar los datos recibidos y controlar los elementos de la estación.

Para la elección de los cuatro tipos de modulación se hizo una revisión de los pequeños satélites que actualmente se encuentran en órbita y operativos según la página web de AMSAT. De acuerdo con la Tabla 5.1 y la Figura 5.1, las modulaciones más usadas son la ASK (en la forma de OOK y codificada en Morse) y la modulación FSK (en la forma de AFSK y codificada en AX.25). Por tal razón, se decidió implementar circuitos módem para las modulaciones ASK y FSK, mientras que los otros dos otros circuitos módem se dejan para desarrollo futuro. Estos circuitos módem adicionales pueden ser alguno de los siguientes: FSK-9600bps (estándar que será usado en el proyecto GENSO), GMSK-9600bps (usado en RADX y BEESAT), PSK-1200bps (usado en Delfi-C3) u otro.

SATELITE	DOWNLINK	MODULACION	RATE	PROTOCOLO
FASTRAC1	437.345MHz	AFSK (FM)	1200bps	AX.25 (TLM)
FASTRAC2	145.825MHz	AFSK (FM)	1200bps	AX.25 (TLM)
O/OREOS	437.305MHz	AFSK (FM)	1200bps	AX.25 (TLM)
HO-68	435.790MHz	Morse (CW)	15wps	MORSE
BEESAT*	436.000MHz	GMSK	9600bps	SRL
SwissCube	437.505MHz	Morse (CW)	11.8wpm	MORSE
PRISM	437.250MHz	Morse (CW)	50wpm	MORSE
SEEDS-II	437.485MHz	AFSK (FM)	1200bps	AX.25 (TLM)
Delfi-C3	145.870MHz	BPSK	1200bps	AX.25
Cute-1.7	437.475MHz	AFSK	1200bps	AX.25
Genesat-1	437.075MHz	AFSK (FM)	1200bps	AX.25 (TLM)
CO-58	437.345MHz	AFSK	1200bps	AX.25
CO-57	437.490MHz	AFSK	1200bps	AX.25
Cute-1	437.470MHz	AFSK (FM)	1200bps	AX.25/SRL
ISS	145.825MHz	AFSK (FM)	1200bps	AX.25 (APRS)

Tabla 5.2: Modulaciones de pequeños satélites (Agosto de 2012)

Se hizo una recopilación de los pequeños satélites operativos y los tipos de modulación implementados en estos satélites. La Figura 5.1 muestra los tipos de modulación y codificación más usados en pequeños satélites, entre los cuales destacan la modulación ASK codificada en Morse y la modulación FSK codificada en AX.25. Por ello se implementaron y se hicieron pruebas con estos dos tipos de modulación.

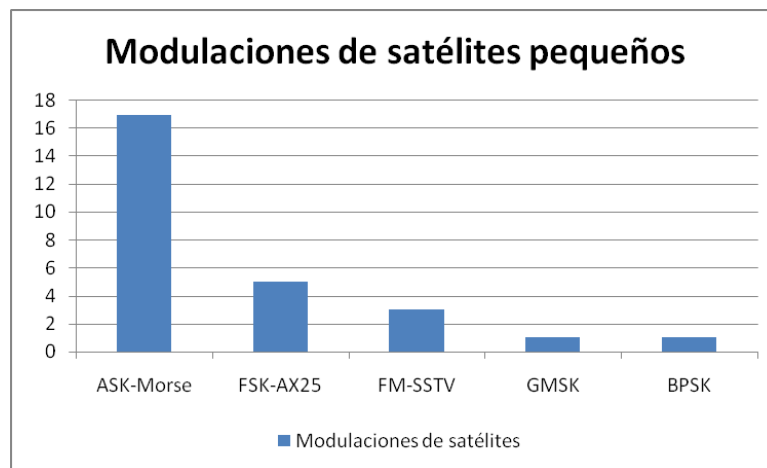


Figura 5.1: Modulaciones usadas en pequeños satélites (Octubre de 2011)

La idea de una solución híbrida que aproveche las ventajas de hardware y software conlleva a repartir convenientemente las funciones entre las dos partes. Por un lado se tienen los circuitos de modulación y demodulación. Por otro lado se tienen los algoritmos de codificación-decodificación y de control de los selectores de módems. La tarea de determinar los parámetros de control de los elementos de la estación terrena en base a simulaciones de las órbitas de los satélites se deja al computador para no recargar el hardware del TNC y usar dispositivos más simples y baratos. La Figura 5.2 muestra el diagrama de bloques general del TNC propuesto con los otros elementos de la estación terrena. Los bloques al lado derecho corresponden al algoritmo de seguimiento de satélites que simula las órbitas de satélites y controla la radio y rotores de antena por medio de interfaces electrónicas.

La Figura 5.3 muestra el diagrama de bloques del hardware del TNC multimodulación. El microcontrolador de control genera las señales de control para los selectores analógicos y digitales de alguno de los cuatro tipos de modulación posibles, a partir de los datos brindados por el computador al cual está conectado.

El microcontrolador códec convierte los datos del módem en caracteres asíncronos y viceversa para el computador. Las señales que ingresan a los circuitos módems mediante el selector analógico provienen de la radio.

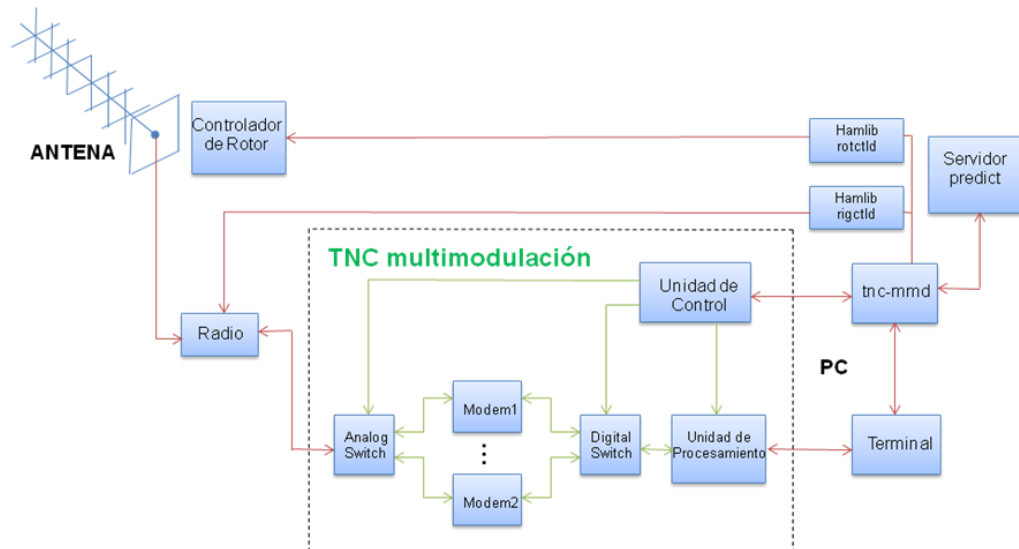


Figura 5.2: Diagrama de bloques del HW/SW del TNC propuesto

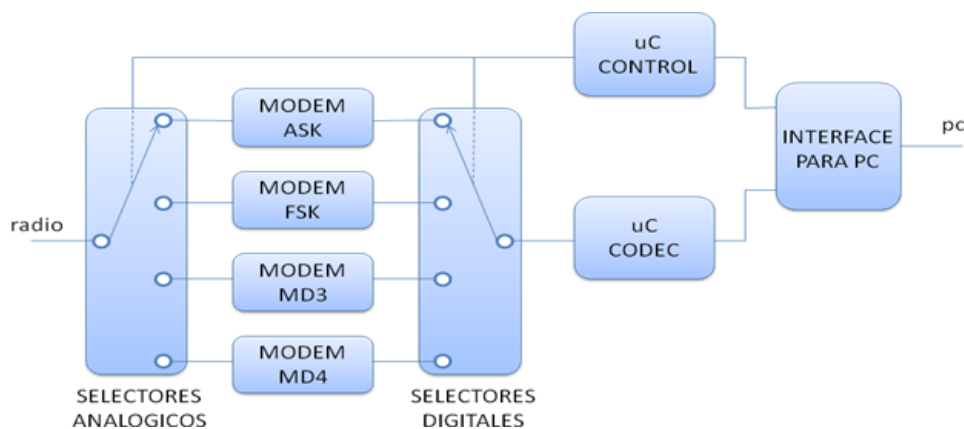


Figura 5.3: Diagrama de Bloques del HW del TNC multimodulación

5.1. Metodología de diseño del Hardware

Selectores analógicos

El TNC multimodulación usa selectores analógicos para conectar la radio a los terminales de audio del módem elegido. Para elegir alguno de los moduladores se requiere de un multiplexor analógico, y para direccionar el demodulador se requiere de un demultiplexor analógico. El circuito integrado CD4052 es un

multiplexor/demultiplexor analógico doble de 4 canales que cumple con los requerimientos para el TNC multimodulación, y se muestra en la Figura 5.4.

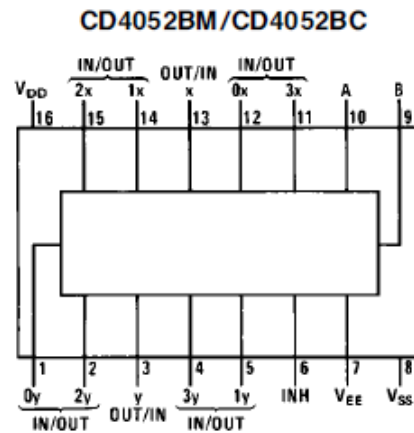


Figura 5.4: Multiplexor/demultiplexor analógico CD4052

Selectores digitales

Para conectar la entrada y salida digital del microcontrolador códec al modulador o demodulador adecuado se necesitan selectores digitales. Un multiplexor digital elige el demodulador, mientras que un demultiplexor digital direcciona la salida digital del microcontrolador códec al modulador adecuado. Los circuitos integrados que cumplen los requisitos son el 74139, demultiplexor doble 1 a 4, y el 74153, multiplexor doble 4 a 1, los que se muestran en la Figura 5.5.

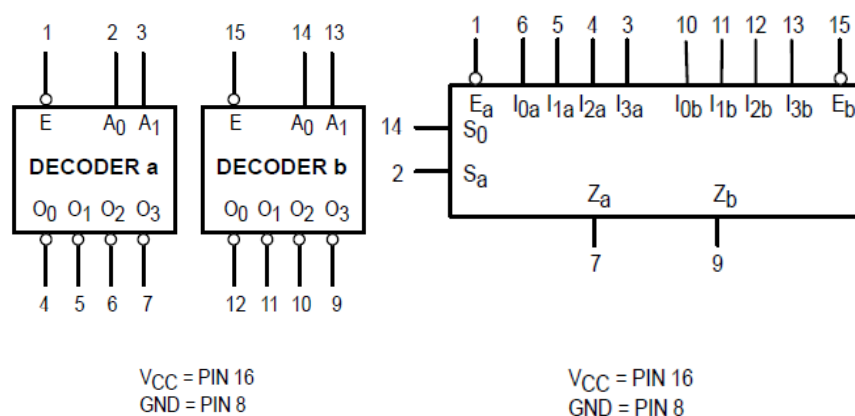


Figura 5.5: Demultiplexor digital 74139 y multiplexor digital 74153

Módem ASK

El TNC multimodulación permite el uso de hasta cuatro módems, que son tarjetas independientes que se insertan en la tarjeta principal. Dos módems, cuyas modulaciones son las más usadas hasta la fecha, han sido construidos para demostrar el principio de selección automática de modulación, mientras que dos conectores quedan disponibles para otros módems que se quieran implementar.

Los dos módems construidos corresponden a modulaciones ASK y FSK respectivamente. Ambos módems usan el circuito integrado XR-2211 de Exar, que es un demodulador FSK y decodificador de tonos basado en un PLL [16]. La modulación ASK que se va a interpretar está codificada en Morse, mientras que la modulación FSK está codificada en AX.25. Las codificaciones Morse y AX.25 han sido implementadas en el microcontrolador de procesamiento de la tarjeta principal del TNC.

Señales ASK son señales digitales moduladas en amplitud, es decir, cada bit (0 ó 1) se representa por una onda senoidal de amplitud determinada (por ejemplo, A_0 ó A_1) como se muestra en la ecuación (5.1). En pequeños satélites es común codificar este tipo de señales digitales usando Morse, donde una de las amplitudes es cero y hay símbolos representados por un número de unos consecutivos.

$$S_{ASK}(t) = (A_0 b[n] + A_1 \overline{b[n]}) \text{sen}(2\pi ft), \text{ donde } b[n]: \text{serie de bits} \quad (5.1)$$

Según la hoja de datos del fabricante del circuito integrado XR-2211, los pasos para el diseño de un decodificador de tonos son los siguientes (ver Figura 5.6).

- Elegir el valor del resistor de sincronización R_0 , que según el fabricante debe estar entre 10k y 50k ohmios (el valor de R_0 es sintonizado finamente con el potenciómetro de precisión R_x).

$$\text{Sean : } R_0' = 22 \text{ k}\Omega \text{ y } R_x' = 10 \text{ k}\Omega \rightarrow R_0 \approx R_0' + \frac{R_x'}{2} = 27 \text{ k}\Omega$$

- Calcular el valor de C_0 de la ecuación de diseño $f_0 = 1/(R_0 * C_0)$, donde f_0 es la frecuencia de oscilación libre del VCO, en este caso el tono que se va a decodificar.

$$\text{Sea : } f_0 = 1.2 \text{ kHz} \rightarrow C_0 = \frac{1}{R_0 f_0} = 30.864 \text{ nF} \approx 22 \text{ nF}$$

- c) Calcular R_1 para establecer una desviación de frecuencia permitida según la ecuación de diseño $\Delta f/f_0=R_0/R_1$, donde Δf es el ancho de banda del tono que se va a decodificar.

$$\text{Sea : } \Delta f = 400 \text{ Hz} \rightarrow R_1 = \frac{2R_0 f_0}{\Delta f} = 162 \text{ k}\Omega \approx 150 \text{ k}\Omega$$

- d) Calcular el valor de C_1 para un valor de factor de amortiguamiento de lazo determinado, cuyo valor recomendado es 0.5, según la ecuación de diseño:

$$C_1 = \frac{1250 C_0}{R_1 \zeta^2} = \frac{1250 \times (22 \times 10^{-9})}{120 \times 10^3 \times (0.5)^2} = 0.917 \text{ nF} \approx 1 \text{ nF}$$

- e) Calcular el valor del capacitor de filtro C_D , según la ecuación de diseño:

$$\text{Sea : } R_D = 470 \text{ k}\Omega \rightarrow C_D > \frac{16}{\Delta F} = \frac{16}{800} = 0.02 \text{ }\mu\text{F} \approx 22 \text{ nF}$$

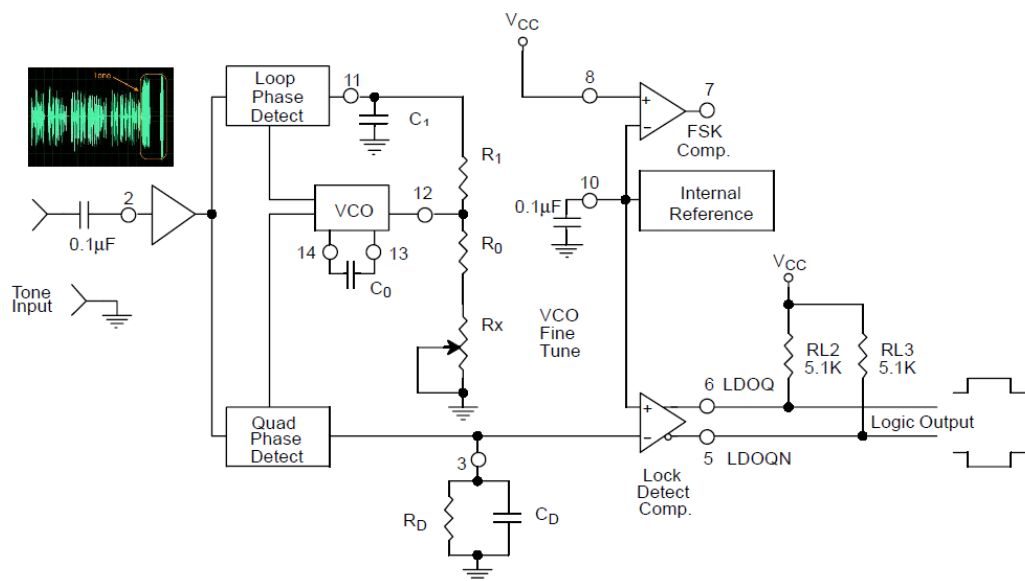


Figura 5.6: Demodulador ASK basado en el CI XR2211

Módem FSK

De manera análoga, señales FSK son señales digitales moduladas en frecuencia, es decir, cada bit (0 ó 1) se representa por una onda senoidal de frecuencia f_0 ó f_1 como se muestra en la ecuación (5.2). En pequeños satélites es común codificar este tipo de señales digitales usando el protocolo AX.25, que agrega una cabecera y una cola a los bytes de datos para formar una trama.

$$S_{FSK}(t) = A\{b[n]\text{sen}(2\pi f_0 t) + \bar{b}[n]\text{sen}(2\pi f_1 t)\}, \text{ donde } b[n]: \text{ serie de bits} \quad (5.2)$$

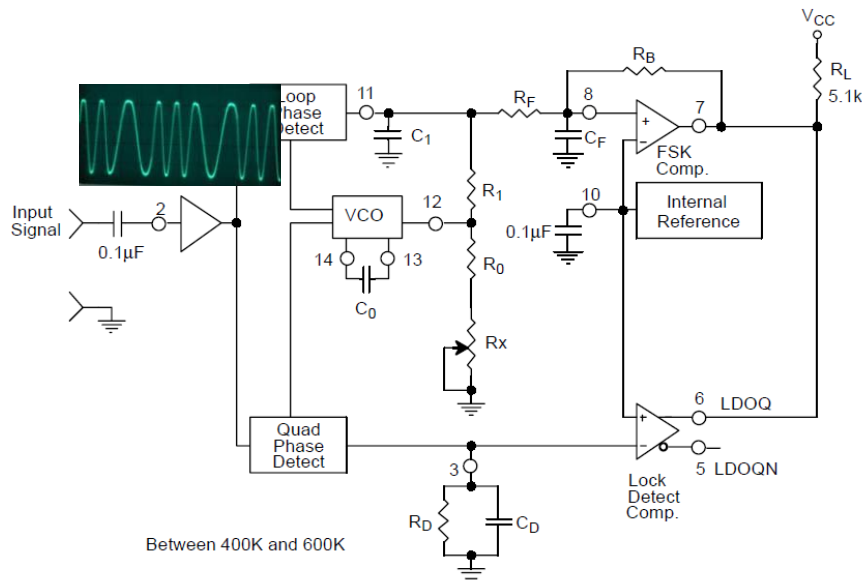


Figura 5.7: Demodulador FSK basado en el CI XR2211

Según la hoja de datos del fabricante del circuito integrado XR-2211, los pasos para el diseño de un demodulador FSK de tipo Bell-202 son los siguientes (ver Figura 5.7):

- a) Calcular la frecuencia central del PLL, f_0 :

$$f_0 = \sqrt{F_1 F_2} \rightarrow f_0 = 1624 \text{ Hz}$$

- b) Elegir el valor del resistor de temporización R_0 (entre 10k y 100k ohmios, finamente sintonizado con el potenciómetro R_x):

$$\text{Sean : } R_0' = 20 \text{ k}\Omega \text{ y } R_x' = 20 \text{ k}\Omega \rightarrow R_0 \approx R_0' + \frac{R_x'}{2} = 30 \text{ k}\Omega$$

- c) Calcular el valor de C_0 de la ecuación de diseño $f_0 = 1/(R_0 \cdot C_0)$, donde f_0 es la frecuencia de oscilación libre del VCO o frecuencia central del PLL:

$$f_0 = 1624 \text{ Hz y } R_0 = 30 \text{ k}\Omega \rightarrow C_0 = \frac{1}{R_0 f_0} = 20.525 \text{ nF} \approx 22 \text{ nF}$$

- d) Calcular R_1 para el ancho de banda deseado:

$$R_1 = \frac{2R_0 f_0}{|F_1 - F_2|} \rightarrow R_1 = 97 \text{ k}\Omega \approx 100 \text{ k}\Omega$$

- e) Calcular C_1 para establecer el factor de amortiguamiento de lazo (0.5 recomendado):

$$C_1 = \frac{1250 C_0}{R_1 \zeta^2} \rightarrow C_1 = 0.917 \text{ nF} \approx 1 \text{ nF}$$

- f) Calcular R_F (por lo menos 5 veces R_1):

$$R_1 = 27 \text{ k}\Omega \rightarrow R_F = 135 \text{ k}\Omega \approx 100 \text{ k}\Omega$$

- g) Calcular R_B (por lo menos 5 veces R_F):

$$R_F = 135 \text{ k}\Omega \rightarrow R_B = 675 \text{ k}\Omega \approx 1 \text{ M}\Omega$$

- h) Calcular R_{SUM} y C_F :

$$R_{SUM} = \frac{(R_F + R_1)R_B}{R_F + R_1 + R_B} = 240 \text{ k}\Omega \rightarrow C_F = \frac{0.25}{R_{SUM} \text{ Baudrate}} = 1 \text{ nF}$$

Microcontrolador códec

Uno de los puntos importantes en el diseño del TNC multimodulación fue la elección de la memoria necesaria para almacenar la trama AX.25 antes de realizar la comprobación de error. En vista que una trama AX.25 es de unos 1024 bytes, se necesitan por lo menos 1.5k bytes para recepción y transmisión de tramas AX.25.

	SRAM ext.	SRAM int.	FRAM	SSD
Capacidad	8kB	2kB	8kB	256MB
Pines	28	0	8	8
Líneas	23	0	3	5
Rapidez	10MHz	10MHz	150kBps	2MBps
Costo	\$1	\$0	\$4	\$10
Disponible	Sí	Sí	No	Sí

Tabla 5.2: Comparación de memorias para el TNC multimodulación

De la Tabla 5.2 se observa que las memorias SRAM son rápidas pero requieren muchos pines I/O para controlarlas, las memorias SSD tienen demasiada capacidad y son de elevado costo, mientras que las FRAM no están disponibles

localmente. Por eso, se decidió usar la memoria SRAM interna del microcontrolador códec, agregándose una restricción al diseño.

Los requerimientos para el microcontrolador códec del TNC multimodulación son los siguientes:

- 22 pines de entrada/salida disponibles.
- Memoria SRAM mayor de 1KB, para evitar uso de memoria externa.
- Frecuencia de trabajo de procesador: 20MHz o mayor.
- Interface hardware UART incluida.
- Temporizadores internos: 3 o más.
- Disponibilidad local.

El microcontrolador PIC18F4550P cumple con las especificaciones en cantidad de pines I/O, memoria SRAM interna, frecuencia de trabajo y disponibilidad. Los recursos de este microcontrolador se muestran en la Figura 5.8.

Device	Program Memory		Data Memory		I/O	10-Bit A/D (ch)	CCP/ECCP (PWM)	SPP	MSSP		EAUSART	Comparators	Timers 8/16-Bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I ² C™			
PIC18F2455	24K	12288	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F2550	32K	16384	2048	256	24	10	2/0	No	Y	Y	1	2	1/3
PIC18F4455	24K	12288	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3
PIC18F4550	32K	16384	2048	256	35	13	1/1	Yes	Y	Y	1	2	1/3

Figura 5.8: Recursos del microcontrolador PIC18F4550P

Microcontrolador de control

Los requerimientos para el microcontrolador de control del TNC multimodulación fueron los siguientes:

- Frecuencia de operación de procesador: 20MHz o mayor.
- Memoria de programa: 12KB o mayor.
- Memoria SRAM interna: 256B o mayor.
- Memoria EEPROM interna: 256B o mayor.
- Número de pines I/O disponibles: 12.
- Interface hardware UART incluida.
- Disponibilidad local.

El microcontrolador PIC16F877A cumple con las especificaciones como se muestra en la Figura 5.9.

Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I ² C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

Figura 5.9: Recursos del microcontrolador PIC16F877A

Interfaz para PC

Los datos procesados por el microcontrolador códec deben intercambiarse con una PC que tiene el software apropiado para gestionar dichos datos. La Tabla 5.3 muestra los protocolos comunes, de los cuales se ha elegido el RS-232 por la disponibilidad en PC comunes y el bajo costo del hardware requerido (UART).

	Paralelo	RS-232	USB	Ethernet
Voltaje	5V	12V	5V	0.85V
Distancia	10m	15m	5m	100m
Líneas	25	3 (9)	4	8
Topología	Bus	Punto-punto	Bus	Red
Rapidez	2MBps	921.6kbps	12Mbps	1Gbps
Costo	\$0	\$1	\$3	\$10
PC común	No	Sí*	Sí	Sí

Tabla 5.3: Comparación de las interfaces para PC comunes

Los caracteres ASCII manejados por el UART del microcontrolador tienen un nivel de voltaje TTL de 0 y 5VDC, mientras que los caracteres ASCII manejados por la PC tienen un nivel de voltaje de 12 y -12VDC. Por ello se necesita de un convertidor de niveles de voltaje de TTL a RS-232 y viceversa, para los microcontroladores de control y códec. El circuito integrado MAX232P es un convertidor de niveles de voltaje doble TTL-RS-232 que cumple con los requerimientos, cuyo diagrama de bloques se muestra en la Figura 5.10..

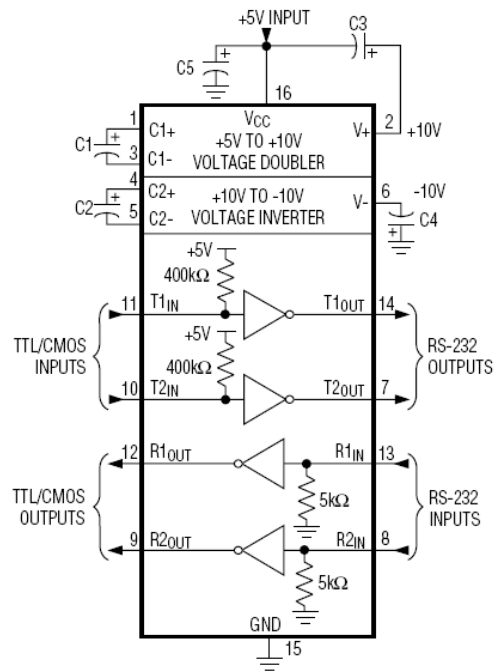


Figura 5.10: Diagrama lógico del convertidor de nivel de voltaje MAX232

Construcción del prototipo

El circuito se construye sobre placas impresas, donde se distribuyen los componentes de los módems y de la tarjeta principal del TNC. Para el diseño de las pistas de conexión de las placas impresas se usó el software Eagle.

Para la implementación de los módems, como los chips XR2206 y XR2211 son integrales, se requiere de poca circuitería adicional. Por ello, el diseño de las pistas se hizo en una sola capa. Para los circuitos integrados se usaron zócalos, de modo que el chip se pueda sacar en cualquier momento y realizar pruebas diferentes.

Las Figuras 5.11 y 5.12 muestran el diagrama esquemático e implementación del primer prototipo, respectivamente, de los circuitos moduladores para ASK y FSK. Se han incluido conectores de 6 pines para las entradas y salidas del circuito, y los potenciómetros son tipo dimmer para circuito impreso. Las Figuras 5.13 y 5.14 muestran el diagrama esquemático e implementación del primer prototipo de los circuitos demoduladores para ASK y FSK. El demodulador ASK usado en el primer prototipo está basado en opamps, luego se cambió por un CI PLL que es más eficiente [17], además de agregar filtro y amplificador.

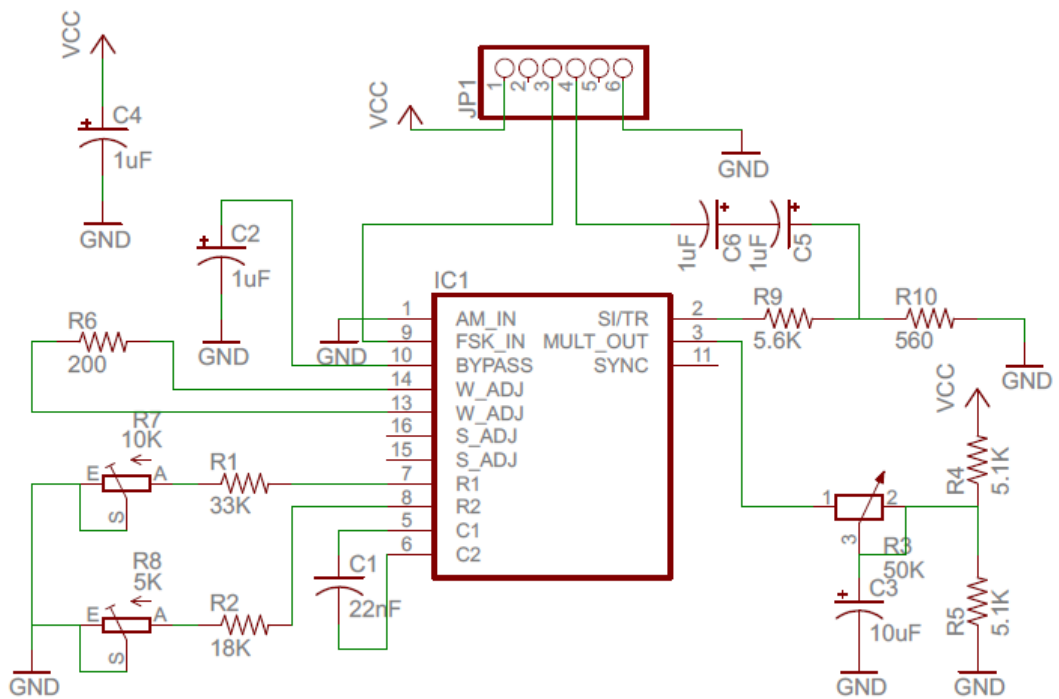


Figura 5.11: Esquema del primer prototipo de modulador ASK/FSK

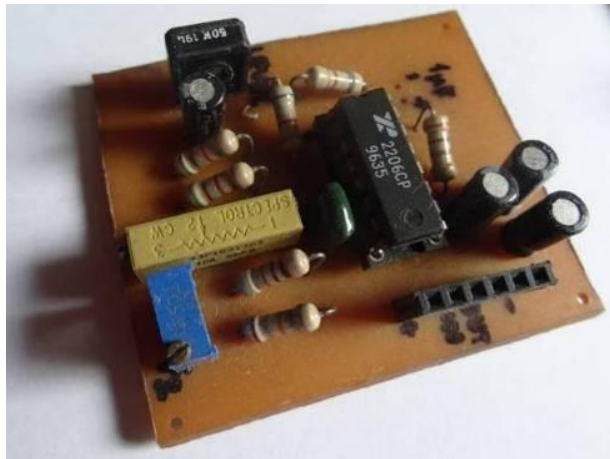


Figura 5.12: Construcción del primer prototipo de modulador ASK/FSK

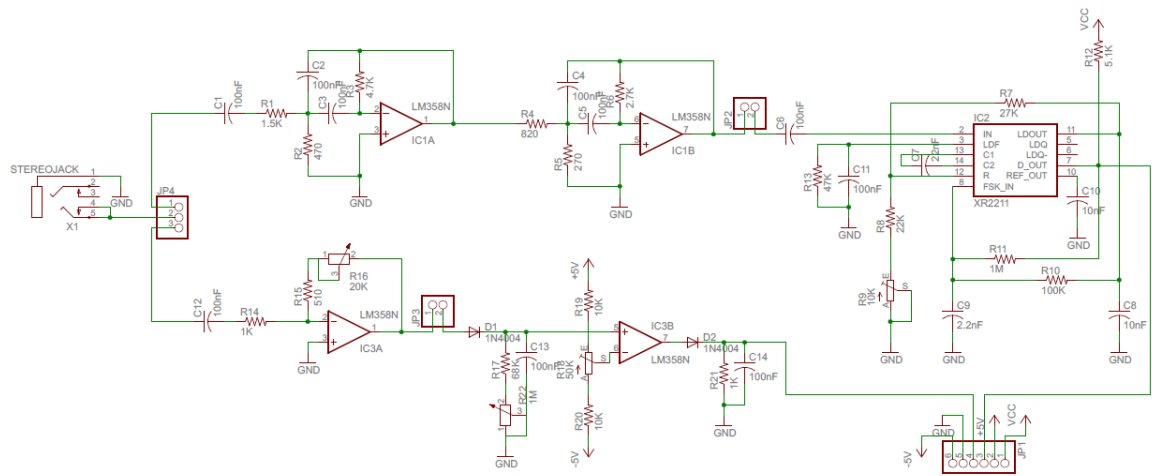


Figura 5.13: Esquema del primer prototipo de demodulador ASK/FSK

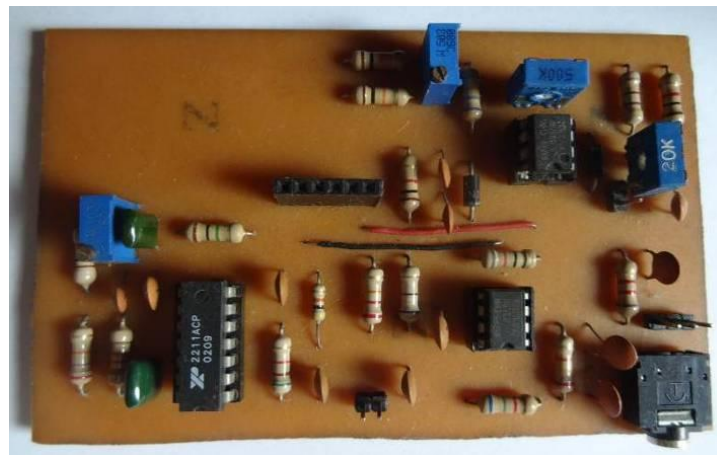


Figura 5.14: Construcción del primer prototipo de demodulador ASK/FSK

La Figura 5.15 muestra la implementación de la tarjeta principal del TNC multimodulación, donde se indican los elementos principales y los puertos de conexión. El TNC se conecta a la radio por medio de los puertos de radio RX y TX, usando un cable fabricado especialmente para la radio con la que se realizarán las pruebas. También se conecta a un computador (PC) por medio de los puertos de control y de datos, usando dos cables USB-serial y un puente USB para ocupar un solo puerto USB del computador o laptop. Los diagramas esquemáticos y diseños de tarjetas impresas de las últimas versiones de las tarjetas se encuentran en los anexos C y D.

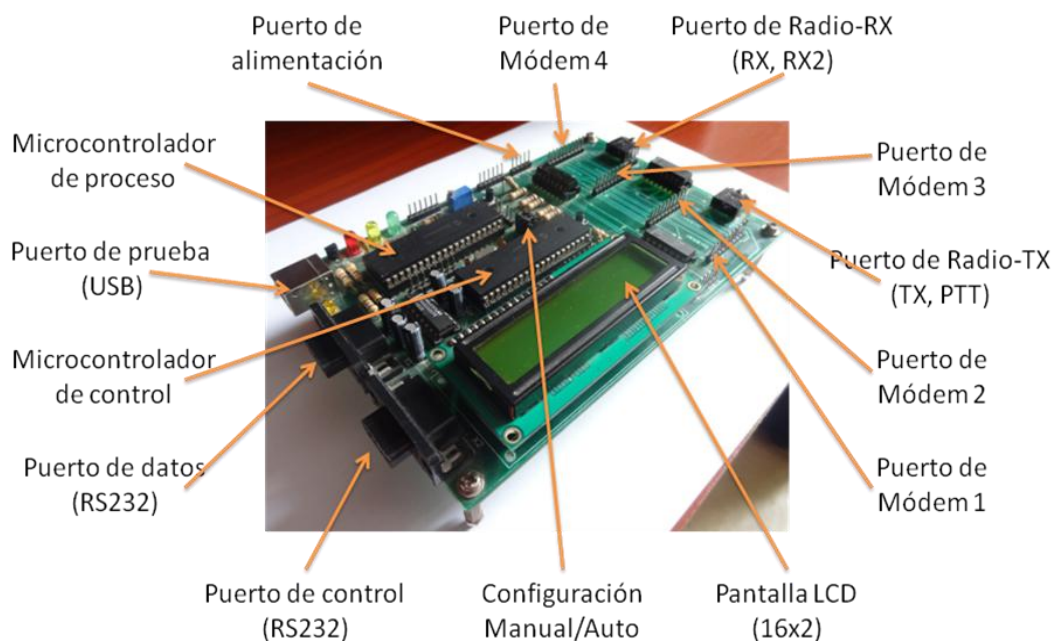


Figura 5.15: Tarjeta principal del TNC propuesto

5.2. Metodología de diseño del Software

El software del TNC multimodulación está destinado principalmente para el microcontrolador códec, para convertir las tramas HDLC enviadas por alguno de los circuitos módem en protocolo KISS para la PC, y para convertir los tonos codificados en Morse en caracteres correspondientes al punto o guión que de igual manera se enviarán a la PC. El software para el microcontrolador de control tiene por finalidad elegir el módem adecuado en función de la información recibida por la PC de control.

El software de gestión para el computador, que interactúa con el TNC multimodulación, también se considera como un elemento de software, y será explicado en la sección siguiente.

Software para el microcontrolador de control

La Figura 5.16 muestra el diagrama de flujo del programa para el microcontrolador de control, compilado con el programa MPLAB X, que consiste en un programa que recibe datos del computador sobre la disponibilidad de algún satélite y el tipo de modulación de éste. Mediante la unidad de control se selecciona el módem adecuado, y se imprime en la pantalla LCD del TNC los parámetros del

satélite si está disponible. Además, el TNC puede ser configurado en modo manual por medio de jumpers conectados al microcontrolador de control, los cuales son revisados al inicio del programa. Se debe quitar el jumper de modo para salir del modo manual.

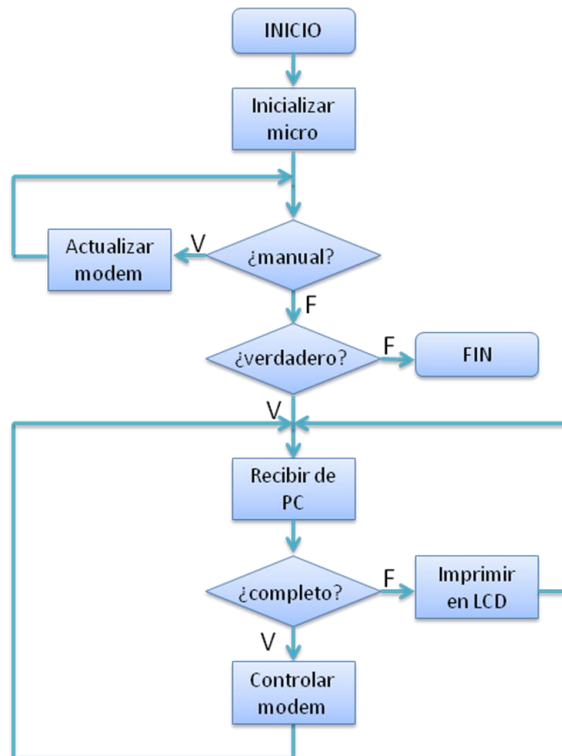


Figura 5.16: Diagrama de flujo del firmware para la unidad de control

Software para el microcontrolador códec

El software para el microcontrolador códec está basado en el código C del TNC-X [18] de John Hansen (W2FS). Para el módem 2 con decodificación AX.25, luego de iniciar el microcontrolador y habilitar las interrupciones, el programa entra a un lazo infinito donde recurre a la función findflag(), que busca la secuencia binaria 0x7E con la cual se inician los paquetes AX.25. Cuando ocurre la secuencia binaria 0x7E, el programa entra a otro lazo donde envía los caracteres ASCII recibidos al puerto serial y guarda los caracteres ingresados por el puerto serial, mientras recibe y almacena los bits provenientes del módem mediante la interrupción rcvbits(), que pone la variable done en 1 y sale de este lazo cuando se recibe y verifica un paquete AX.25 completo. A continuación, si la variable xmitlatch

es 1 se realiza la transmisión de un paquete AX.25 pendiente por medio de la función xmit().

Se agregó al código de John Hansen la funcionalidad de decodificación Morse, que convierte bits demodulados de tonos ASK en secuencias de caracteres punto (.) y guión (-), los cuales son enviados a la PC para ser interpretados usando la tabla Morse. Además, se agregó el método de cambio automático de decodificador, que consiste en consultar los pines de entrada provenientes de la unidad de control cada vez que se entre en un lazo. La Figura 5.17 muestra el diagrama de flujo completo del algoritmo para el microcontrolador de procesamiento, donde el lado izquierdo corresponde al flujo de decodificación Morse y el lado derecho corresponde al flujo de codificación-decodificación AX.25. En ambos casos, la recepción de datos de los módems se da por interrupción, dentro del lazo “¿recibido?”.

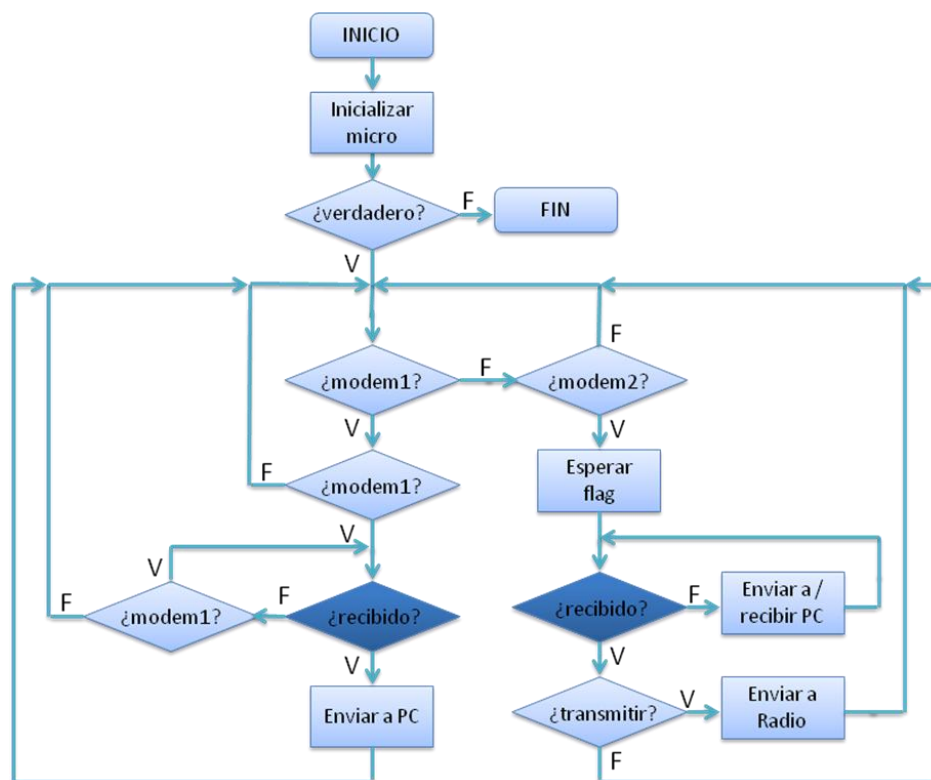


Figura 5.17: Diagrama de flujo del firmware para la unidad códec

5.3. Software de gestión para la PC

Con el aumento rápido de capacidad de las PC comunes a precios razonables, se han podido implementar funciones más complejas a la de los nodos AX.25, entre las cuales se puede mencionar:

-Bases de datos: Antes la PC solo mostraba los datos recibidos en el monitor y servía para enviar los datos escritos por teclado, pero ahora sirve como almacén de la información y gestor de los recursos de hardware y software de la estación terrena.

-Interconexión de redes: Ahora las PCs sirven como interface a la Internet, con todas las ventajas que esto supone, como monitoreo a distancia y distribución de recursos. Cualquier PC con acceso a Internet podría obtener los datos que están llegando a la estación terrena.

-Automatización del manejo de satélites: Los programas en la PC para gestionar la comunicación con satélites son sistemas complejos y robustos que manejan el hardware de la estación terrena, y son programados para una operación autónoma del satélite.

Entre los sistemas de gestión de datos disponibles para aplicaciones de satélites se puede mencionar a Mercury y Genso. Este último es un proyecto que pretende poner en órbita una constelación de pequeños satélites para abarcar todo el planeta, y para ello requiere de la mayor cantidad posible de estaciones terrenas en diferentes ubicaciones, siendo la del CTIC-UNI un posible nodo Genso cuando se concrete el proyecto.

Para operar el TNC multimodulación, aparte del puerto de recepción y envío de datos en el software de gestión, debe haber un servicio adicional que maneje el puerto de control del TNC multimodulación. Por tal motivo, se ha preparado un software en lenguaje C que permite el control de los elementos de la estación terrena: rotores de antena, frecuencia de radio y modulación del TNC. La Figura 5.18 muestra el diagrama de flujo del algoritmo de seguimiento de satélites y control de los elementos de la estación terrena, que se explicará con más detalle en el capítulo 6. Básicamente, consiste en un estado de búsqueda de satélite, donde se halla el satélite con la menor ángulo de elevación positivo, y un estado de seguimiento del satélite, donde se controlan los elementos de la estación terrena hasta que el ángulo de elevación del satélite sea negativo (pérdida de línea de vista con el satélite).

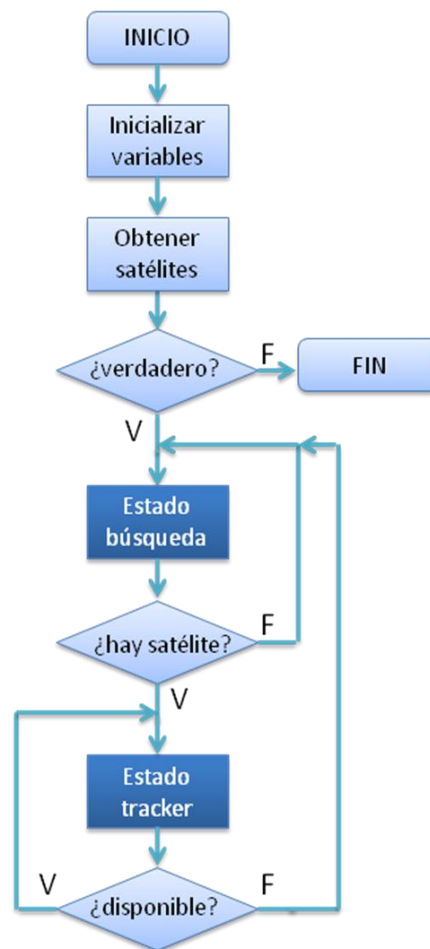


Figura 5.18: Diagrama de flujo del software de control para la PC

5.4. Plan de pruebas del mecanismo de selección

Se probará el TNC multimodulación en dos escenarios: la recepción consecutiva de información de pequeños satélites con diferentes modulaciones usando el algoritmo de seguimiento de satélites, y la decodificación de señales de modulaciones ASK-Morse y FSK-AX.25, con una reconfiguración automática del tipo de modulación del TNC ante el cambio de modulación de uno de los satélites.

La Figura 5.19 muestra la configuración física de conexiones de los elementos de la estación terrena donde se hicieron las pruebas en el primer escenario. Las antenas de VHF y UHF están montadas en un mástil con el rotor instalado para dar movimiento en los ángulos de azimut y elevación. Las antenas se conectan a la radio, la cual se conecta también al TNC multimodulación, y éste a la PC por medio de dos puertos seriales, uno para los datos y el otro para el control

del tipo de modulación. El controlador de rotor se conecta a la PC por puerto paralelo, y la interface de frecuencia de radio se conecta a la PC por puerto USB. La PC tiene un software de gestión demostrativo que accede a programas dentro de la PC para obtener información del satélite actual y manejar la posición de las antenas, la frecuencia de la radio y la modulación del TNC.

La segunda parte de las pruebas consiste en la impresión de los datos en el software de la PC, provenientes de grabaciones de audio de dos diferentes tipos de modulación, donde se demostrará el cambio de circuito demodulador y del algoritmo de decodificación del TNC multimodulación. La descripción de las pruebas realizadas en estos dos escenarios y los resultados obtenidos se explican en el capítulo 7.

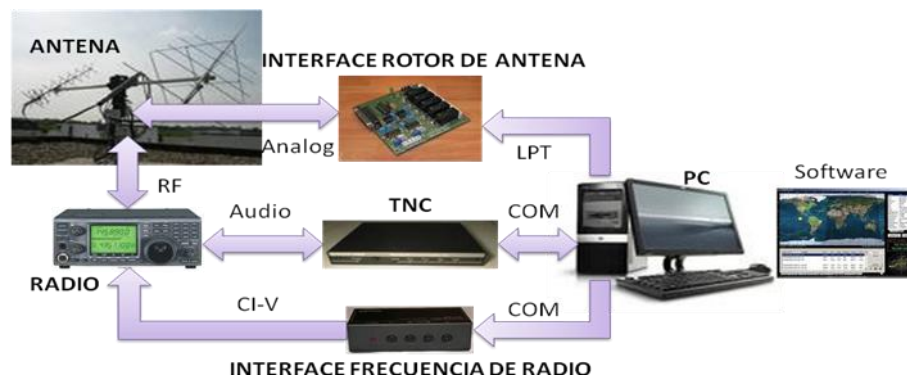


Figura 5.19: Esquema físico de la prueba del mecanismo de selección

CAPÍTULO VI

SIMULACIÓN DEL ALGORITMO DE SEGUIMIENTO DE SATÉLITES

6.1. Introducción

Para probar el mecanismo de selección automática del tipo de modulación se necesitan datos reales de pequeños satélites que orbitan alrededor de la tierra. Existen programas de computador que simulan los parámetros orbitales de satélites en base a los valores de parámetros representativos para cada satélite en un tiempo dado, mediante algoritmos de determinación de órbitas como SGP4 y SPD4 [19]. Estos parámetros representativos de cada satélite se denominan TLE y se obtienen en forma de archivos de servidores web como CelesTrak. Entre los datos suministrados por los programas predictores de órbitas se encuentra el ángulo de elevación, que está directamente relacionado a la disponibilidad del satélite para la estación terrena. En la Figura 6.1 se muestra el entorno de desarrollo para la simulación del algoritmo de seguimiento automático de pequeños satélites. El algoritmo de seguimiento es un programa editado en lenguaje C que se comunica con otros programas de software libre: el software predictor de órbitas “Predict”, la librería de control “Hamlib” y la consola del entorno de desarrollo Eclipse, donde además se prueba el algoritmo de seguimiento. Las interfaces de comunicación de la radio y el rotor se simulan con elementos de la librería Hamlib denominados elementos “dummy”. La interfaz de comunicación del TNC multimodulación se simula con la consola de Eclipse.

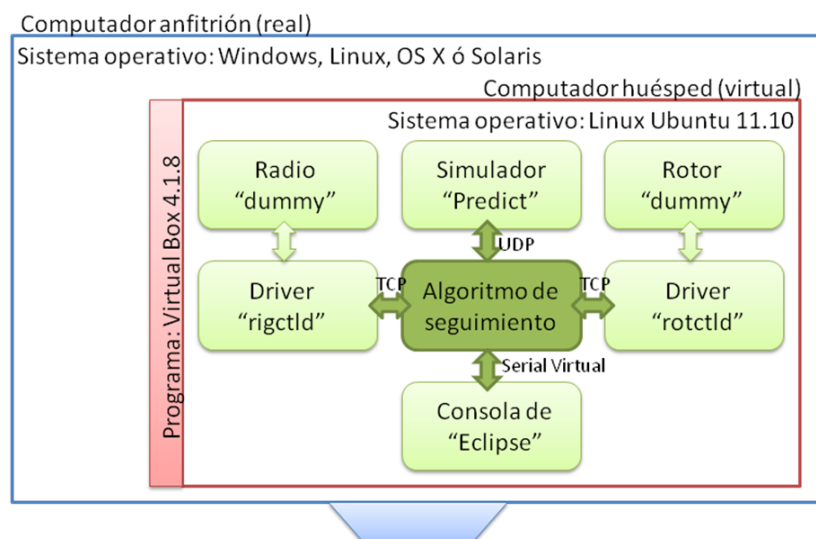


Figura 6.1: Escenario de prueba para las simulaciones

6.2. Programas usados

Para la implementación del algoritmo de seguimiento automático de satélites se usaron los siguientes programas instalados en Linux Ubuntu:

- ✓ Programa predictor de órbitas Predict v2.2.3-3.
- ✓ Librería de control de radios y rotores Hamlib v1.2.14-1.
- ✓ Entorno de desarrollo integrado Eclipse v3.7.0-0.

Instalación de Predict

Se puede instalar predict desde el terminal de comandos, haciendo uso de la herramienta apt-get. También se puede descargar el archivo binario o la última versión de Predict desde su sitio web: <http://www.qsl.net/kd2bd/predict.html>

Para buscar, instalar y ver la versión de Predict desde el terminal de comandos escribir (notar que para instalar en Ubuntu, es necesario establecer un permiso temporal de administrador con el prefijo sudo, por lo que se escribe la contraseña de usuario al inicio):

```

usuario@pc:~$ sudo apt-cache search predict
[sudo] password for usuario: contraseña
usuario@pc:~$ sudo apt-get install predict
usuario@pc:~$ sudo apt-cache show predict

```

Para ver el manual de uso de Predict, en el terminal de comandos escribir: `$man predict` (para salir escribir q). La opción `$predict -s` es de particular interés, porque habilita el servidor UDP de Predict, lo que hace posible el acceso de cualquier programa a los datos del software por el protocolo UDP.

```

predict [-u tle_update_source] [-t tlefile] [-q qthfile] [-a
        serial_port] [-a1 serial_port] [-n network_port] [-f sat_name start -
        ing_date/time ending_date/time] [-p sat_name starting_date/time] [-o
        output_file] [-s] [-east] [-west] [-north] [-south]

```

Al iniciar por primera vez el software Predict, éste solicita los parámetros de ubicación de la estación terrena, es decir su latitud, longitud y altitud (adicionalmente el identificador de la estación terrena). Estos datos son necesarios porque los parámetros orbitales de los satélites se calculan respecto de una

posición fija en tierra. La Figura 6.2 muestra una captura de pantalla del software Predict mostrando los parámetros de una estación terrena.

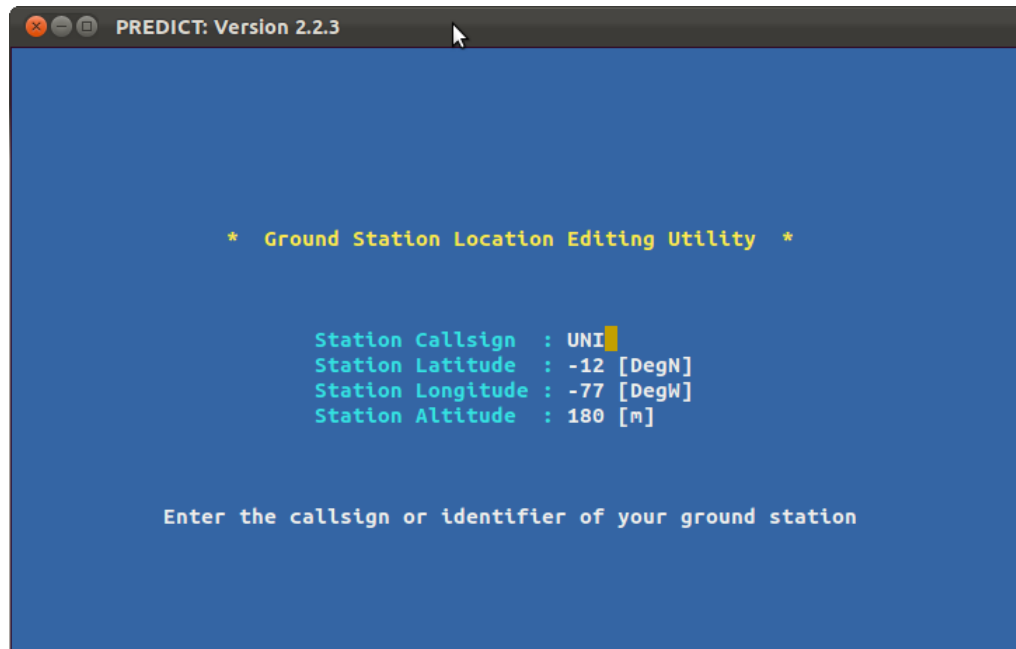


Figura 6.2: Edición de una estación terrena con software Predict

La lista de satélites y los elementos orbitales de los satélites no estarán actualizados luego de instalar. Para actualizar la lista de satélites (y los parámetros orbitales de los mismos) se debe editar el archivo `predict.tle` ubicado en la carpeta `.predict`, que a su vez se encuentra en la carpeta de archivos del usuario. Para editar el archivo `predict.tle` se hace uso de la herramienta `vim`, que quizá tenga que instalarse previamente con `apt-get`. El archivo `predict.tle` contiene una lista de 24 satélites con sus respectivos parámetros TLE, los cuales se pueden modificar con datos actualizados de la página de AMSAT: <http://www.amsat.org/amsat-new/satellites/status.php>

```

usuario@pc:~$ vim home/usuario/.predict/predict.tle
usuario@pc:~$ predict -s
<Presionar la tecla Esc para salir de la aplicación>
  
```

Si solo se quiere actualizar los TLE de los satélites, los cuales se descargan de <http://celestrak.com/NORAD/elements/> por lo menos cada 3 o 5 días, se puede ejecutar un script que descarga los datos de internet en archivos de texto y realiza

la actualización. El script se ejecuta con doble clic en el archivo predictUpdate.sh. La Figura 6.3 muestra una captura de pantalla del software Predict en ejecución.

```
#!/bin/sh
wget -qr www.celestrak.com/NORAD/elements/amateur.txt -O amateur.txt
wget -qr www.celestrak.com/NORAD/elements/visual.txt -O visual.txt
wget -qr www.celestrak.com/NORAD/elements/weather.txt -O weather.txt
/usr/bin/predict -u amateur.txt visual.txt weather.txt
```

The screenshot shows a terminal window titled "PREDICT: Multi-Satellite Tracking Mode". The window displays the following information:

PREDICT Real-Time Multi-Tracking Mode
Current Date/Time: Tue 05Apr11 05:17:51

Satellite	Az	El	LatN	LonW	Range	Satellite	Az	El	LatN	LonW	Range
HOPE-1	16	-64	57	279	12755 D	CO-65	244	-7	-24	109	3748 N
ITUPSAT1	170	-22	-65	58	6263 N	CO-58	82	-68	14	300	12557 D
BEESAT	160	+24	-22	73	1466 N	HAMSAT	144	-46	-49	320	10110 D
SWISSCUBE	206	-48	-56	208	10479 D	ECHO	65	-31	19	12	7912 D
SO-67	200	-25	-63	117	6463 N	RS-22	20	-49	71	343	10498 D
KKS-1	176	-46	-71	270	10012 D	CO-57	264	-41	-6	168	9553 D
STARS	16	-8	21	67	4004 N	CUTE-1	277	-43	7	169	9866 D
PRISM	94	-4	-12	48	3350 N	SO-42	301	+21	-6	86	1374 N
YUBILEINY	117	+42	-17	67	2045 N	PCSAT	344	-11	27	89	4708 N
CO-66	154	-50	-54	303	10534 D	ISS	54	-62	36	313	11610 D
DELFI-C3	151	-53	-48	300	10952 D	SO-33	303	-70	29	222	13046 D
COMPASS-1	160	-45	-64	309	9854 D	GO-32	282	-52	15	183	11010 D

Upcoming Passes

Sun	Moon
-----	-----
165.71 Az	289.17 Az
-83.75 El	-76.09 El

Upcoming passes for ITUPSAT1, KKS-1, and SO-33 are listed with their respective times and UTC offsets.

Figura 6.3: Simulación de parámetros de satélites con Predict

Instalación de Hamlib

La instalación de la librería de control de radios y rotores libhamlib2 se usó la herramienta apt-get. También se pueden descargar los archivos binarios o códigos fuente de la última versión de libhamlib2 de la página web de los desarrolladores de este software libre:

http://sourceforge.net/apps/mediawiki/hamlib/index.php?title=Main_Page

```

usuario@pc:~$ sudo apt-cache search libhamlib
[sudo] password for usuario: contraseña
usuario@pc:~$ sudo apt-get install libhamlib2
usuario@pc:~$ sudo apt-get install libhamlib-dev
usuario@pc:~$ sudo apt-cache show libhamlib2

```

A pesar de que hamlib es una librería en desarrollo, tiene soporte para el control de la mayoría de radios del mercado. El control de la frecuencia y modo de operación de la radio son necesarios para la automatización del seguimiento de satélites y corrección del efecto Doppler. Para ver la lista de controladores de radios que han sido implementados en hamlib y su estado de desarrollo escribir: \$rigctl -l.

Rig#	Mfgr	Model	Vers.	Status
2512	FiFi	FiFi-SDR	0.3	Beta
1504	Winradio	WR-3100	0.6	Untested
352	Optoelectronics	OptoScan535	0.3	Beta
304	Icom	IC-275	0.7.1	Beta
336	Icom	IC-R10	0.7	Untested
368	Icom	IC-9100	0.7	Untested
320	Icom	IC-736	0.7	Untested
208	Kenwood	TS-811	0.8.0.6	Untested
224	Kenwood	TS-680S	0.8.1	Beta
128	Yaesu	FT-950	0.22.1	Stable
2801	Philips/Simoco	PRM8060	0.1	Alpha
2513	AMSAT-UK	FUNcube Dongle	0.2	Beta
2401	RFT	EKD-500	0.4	Alpha
1601	Ten-Tec	TT-550	0.2	Beta
1505	Winradio	WR-3150	0.6	Untested
1105	Racal	RA3702	0.1	Alpha
801	Uniden	BC780xlt	0.3	Untested
513	AOR	AR8600	0.6.1	Beta
401	Icom	IC-PCR1000	0.8	Beta
353	Optoelectronics	OptoScan456	0.3	Beta
337	Icom	IC-R71	0.7	Untested
321	Icom	IC-737	0.7	Untested
209	Kenwood	TS-850	0.8.0	Beta
225	Kenwood	TS-140S	0.8.1	Beta
129	Yaesu	FT-2000	0.22	Stable
113	Yaesu	FT-900	0.1	Untested
1	Hamlib	Dummy	0.5	Beta
1602	Ten-Tec	TT-538 Jupiter	0.4	Beta
1506	Winradio	WR-3500	0.6	Untested
802	Uniden	BC245xlt	0.3	Untested
514	AOR	AR5000A	0.6	Alpha
402	Icom	IC-PCR100	0.8	Beta
354	Icom	IC ID-1	0.7	Untested

306	Icom	IC-471	0.7	Untested
338	Icom	IC-R72	0.7	Untested
322	Icom	IC-738	0.7	Untested
226	Kenwood	TM-D700	0.5	Beta
210	Kenwood	TS-870S	0.8.0	Beta
130	Yaesu	FTDX-9000	0.22	Untested
114	Yaesu	FT-920	2010-08-23	Stable
2	Hamlib	NET rigctl	0.3	Beta
1603	Ten-Tec	RX-320	0.6	Stable
1507	Winradio	WR-3700	0.6	Untested
803	Uniden	BC895xlt	0.3	Untested
515	AOR	AR7030 Plus	0.1	Beta
403	Icom	IC-PCR1500	0.8	Beta
307	Icom	IC-475	0.7.1	Beta
339	Icom	IC-R75	0.7	Beta
323	Icom	IC-746	0.7	Beta
355	Icom	IC-703	0.7	Untested
227	Kenwood	TM-V7	0.5	Beta
211	Kenwood	TS-940S	0.8.0.6	Alpha
131	Yaesu	FT-980	0.1	Alpha
115	Yaesu	FT-890	0.1	Stable
1604	Ten-Tec	RX-340	0.3	Untested
1204	Watkins-Johnson	WJ-8888	0.2	Untested
804	Radio Shack	PRO-2052	0.3	Untested
516	AOR	SR2200	0.1	Beta

Los comandos rigctl y rotctl se usan para el control de radio y rotor respectivamente, cuando dichos equipos están conectados en el computador donde está instalado Hamlib y la aplicación que hace uso de la librería. Esto quiere decir que acceden directamente al puerto serial o paralelo del computador, dependiendo de la interface que tenga el equipo a controlar. Se pueden acceder a los manuales de ayuda de estas herramientas escribiendo en el terminal \$man rigctl o \$man rotctl. Los parámetros más importantes para las herramientas rigctl y rotctl son el identificador de dispositivo y la interfaz de comunicación. Por ejemplo, para un rotor ARS AZ&EL que se comunica por el puerto paralelo:

```
usuario@pc:~$ rotctl -m 1101 -r /dev/parport0
```

Los comandos rigctld y rotctld se usan para controlar la radio y rotor respectivamente, por medio del protocolo TCP, por lo que la aplicación que hace uso de las librerías puede estar en cualquier computador conectado a la red del

computador que está conectado físicamente a los equipos (y donde está instalado Hamlib). Los parámetros de las herramientas rigctld y rotctld son similares a los de sus análogos rigctl y rotctl. Por ejemplo, para una radio ICOM IC-910H que se comunica por puerto serial y cable USB-serial:

```
usuario@pc:~$ rigctld -m 344 -r /dev/ttyUSB0
```

6.3. Pruebas de caja blanca

Pruebas de caja blanca hacen referencia a las pruebas de software de las componentes internas de un programa de software. En efecto, se realizaron pruebas de acceso a los datos calculados por Predict y pruebas de envío de datos a los controladores de rotor y radio por medio de la librería Hamlib.

Acceso a datos de Predict

La Figura 6.4 muestra el esquema de la prueba de lectura de los parametros orbitales de satélites calculados por Predict, via protocolo UDP. El programa ClientePredict está editado en código C, basado en el ejemplo brindado junto con el código fuente de Predict, disponible de manera libre. ClientePredict solicita lista de satélites y parámetros orbitales tales como ángulos de azimuth y elevación, así como desviación de frecuencia por efecto Doppler. Este programa ha sido compilado y ejecutado dentro del entorno Eclipse, en cuya consola se imprimen los datos obtenidos de Predict.

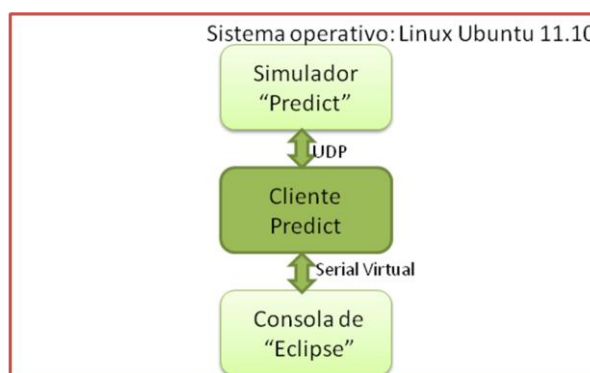


Figura 6.4: Esquema de la prueba de acceso a Predict

Para que el programa ClientePredict pueda acceder a los datos de Predict, éste software se debe estar ejecutando en modo servidor (`$predict -s`); si no es así,

el entorno Eclipse da un error: Connection refused - PREDICT server not running. En la figura se muestra la impresión de los datos obtenidos al enviar el comando GET_LIST. En la Figura 6.5 se observa una captura de pantalla de la ejecución de este programa con Eclipse.

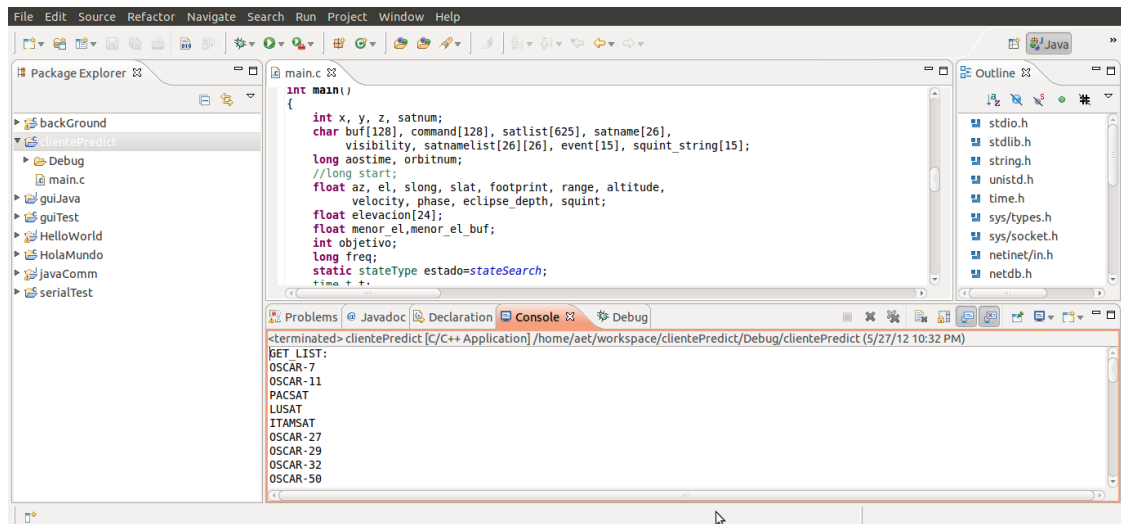


Figura 6.5: Prueba de solicitud de lista de satélites usando Eclipse

Envío de comandos a Hamlib

El envío de comandos (y lectura de respuestas) a los controladores de radio y rotor de Hamlib se hizo de 3 maneras: por el terminal de comandos (para rigctl y rotctl), con la herramienta xdx (para rigctld y rotctld) y con un programa propio editado en lenguaje C (para rigctl, roctl, rigctld o rotctld). La figura muestra el entorno para la prueba del programa ClienteHamlib, que envía comandos a los controladores rigctld y rotctld vía protocolo TCP, además muestra los datos en la consola del entorno Eclipse donde fue compilado y ejecutado. Para evitar el uso de los equipos radio y/o rotor físicos, se usaron los simuladores suministrados por la herramienta Hamlib, denominados radio y rotor tontos, cuyo identificador de dispositivo es el número 1 para ambos. La Figura 6.6 muestra el diagrama de bloques de la prueba de envío de comandos a Hamlib.

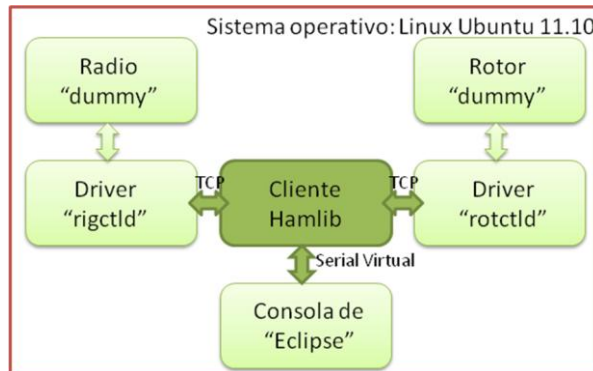


Figura 6.6: Esquema de prueba de envío de comandos a Hamlib

Las pruebas con el terminal de comandos de Ubuntu implican escribir el comando `rigctl` o `rotctl` con sus argumentos correspondientes. Las herramientas `rigctl` y `rotctl` generan un sub-terminal por donde se escriben los comandos. Por ejemplo, para `rigctl` con una radio simulada que se conecta por el puerto serial:

```
usuario@pc:~$ rigctl -m 1 -r /dev/ttyUSB0
```

```
Rig command: F
Frequency: 435000000
```

```
Rig command: f
Frequency: 435000000
```

```
Rig command: F 144000000
Frequency:
Rig command: f
Frequency: 144000000
```

```
Rig command: q
```

Las pruebas con la herramienta `xdx` consistieron en ejecutar `rigctld` y/o `rotctld`, luego enviar comandos y recibir datos vía protocolo TCP, por el puerto 4532 para `rigctld` y por el puerto 4533 para `rotctld`. Para un controlador de rotor simulado que se conecta por el puerto paralelo se escribe en el terminal de Ubuntu: `$rotctld -m 1 -r /dev/parport0` (para salir escribir `Ctrl+c`). Una vez ejecutada la herramienta `rotctld`, se abre el programa `xdx` (previamente instalado con `apt-get`), que es un cliente genérico Linux, luego se realiza la conexión TCP a la dirección 127.0.0.1 (computador local) por el puerto 4533 (`rotctld`). Los comandos importantes son: `p` para obtener los ángulos de azimut y elevación, `P <A.Z> <E.L>` para establecer

ángulos de azimut y elevación (notar que el controlador de rotor simulado muestra un retardo en el establecimiento de los ángulos al leer, lo cual es correcto), y q para salir. Una captura de pantalla de Xdx se muestra en la Figura 6.7.

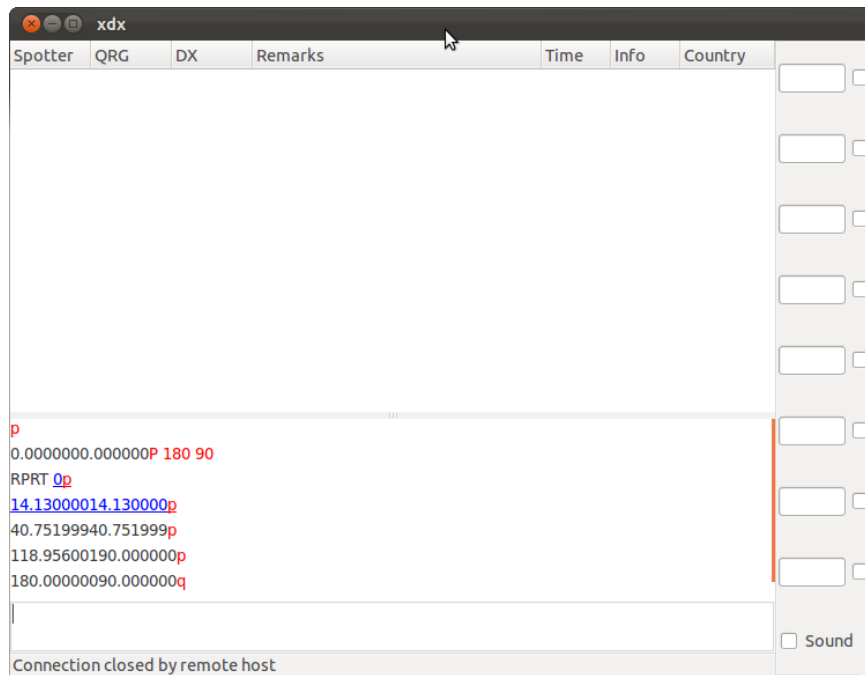


Figura 6.7: Prueba de envío de comandos a Hamlib usando xdx

Para la prueba de ClienteHamlib se ejecutan previamente en el terminal los comandos: `$rigctld -m 1 -r /dev/ttyUSB0 &` (el ampersand para ejecutar en segundo plano), y `$rotctld -m 1 -r /dev/parport0 &`. Si no se ejecutan `rigctld` o `rotctld` el entorno de desarrollo Eclipse da un error de rechazo de conexión, porque alguno de los puertos TCP no está activo. En la consola de Eclipse, se pueden escribir los comandos de radio y rotor, al escribir q se finaliza la conexión TCP con el controlador. La aplicación ClienteHamlib puede estar en un computador conectado en red con el computador donde está instalado Hamlib, que ha de estar conectado físicamente a la radio y/o rotor de la estación terrena. Una captura de pantalla de esta prueba se muestra en la Figura 6.8.

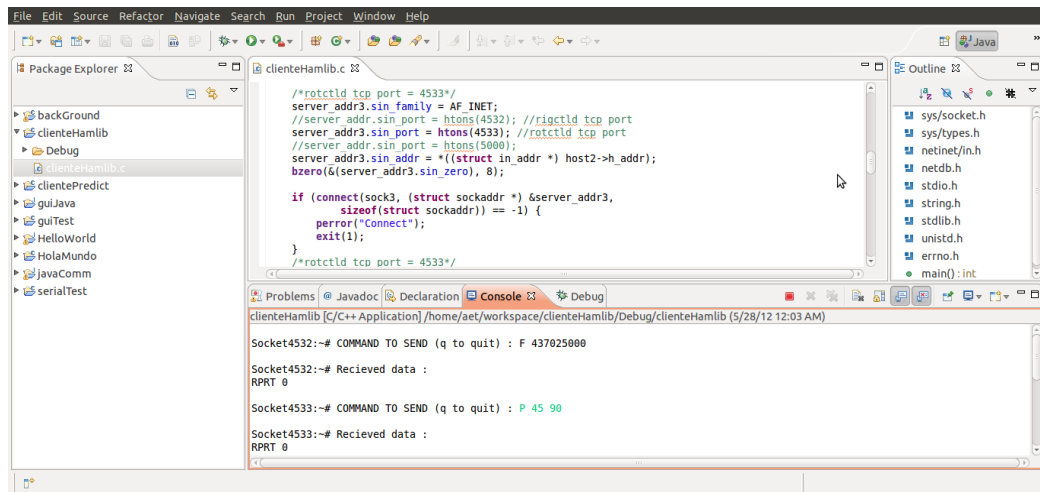


Figura 6.8: Prueba de envío de comandos a Hamlib usando Eclipse

6.4. Pruebas de caja negra

Las pruebas de caja negra son pruebas de software considerando solo los estímulos y las respuestas del software. El programa de seguimiento automático de satélites es la integración de los clientes Predict y Hamlib vistos previamente, con un algoritmo simple de selección de satélite más aprovechable. El algoritmo consiste en iniciar con una comparación de los ángulos de elevación de los satélites disponibles en la lista de Predict, de los cuales elige el menor ángulo, debido a que tendrá más oportunidad de establecer comunicación con la estación terrena. Una vez elegido el satélite, se procede a enviar los ángulos de azimut y elevación, y la frecuencia compensada por efecto Doppler hacia las librerías de controlador de rotor y radio respectivamente, hasta que la elevación del satélite observado sea menor o igual a cero (lo que significa que ya no hay línea de vista). Una vez perdido el contacto con el satélite, se procede nuevamente a una búsqueda de satélite más aprovechable. Una captura de pantalla de esta prueba se muestra en la Figura 6.9.

También se realizaron pruebas de recepción de datos de control por el puerto serial con un terminal serial desarrollado en Java, basado en las librerías Rxtx y WindowBuilder. Este terminal gráfico está incluido en la interfaz gráfica del TNC y ejecuta programa de seguimiento automático de satélites, el cual interactúa con el hardware del TNC multimodulación. La Figura 6.10 muestra los elementos de la interfaz gráfica desarrollada en lenguaje Java.

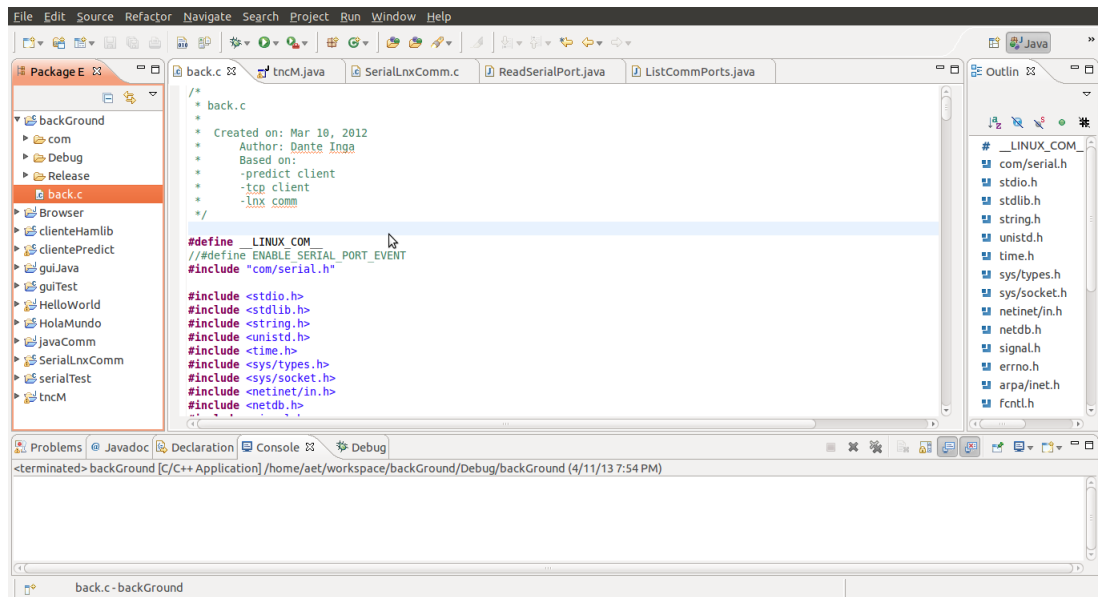


Figura 6.9: Prueba del programa de seguimiento de satélites

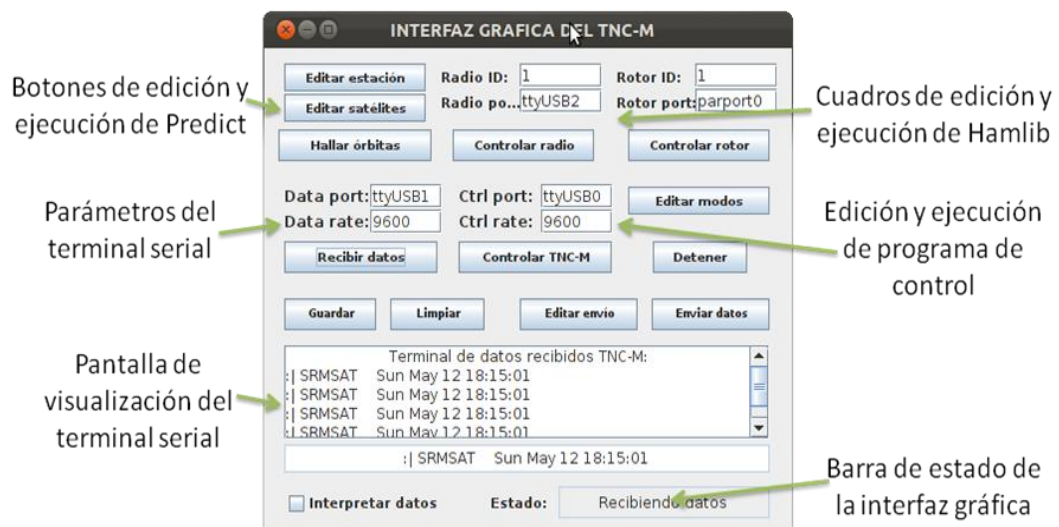


Figura 6.10: Interfaz gráfica con terminal serial

De las pruebas anteriores se encontró que se cumplen los objetivos planteados de los componentes software del programa de seguimiento automático de satélites, de manera individual y conjunta.

CAPÍTULO VII

IMPLEMENTACIÓN Y PRUEBAS DEL TNC PROPUESTO

7.1. Introducción.

Para la implementación del TNC multimodulación se ha elegido un diseño con componentes disponibles de manera local en su totalidad, con la finalidad principal de promover la construcción de circuitos de este tipo en la comunidad de investigación peruana. Esto implica una reducción en el costo y tiempo de adquisición de componentes, al tratarse de materiales disponibles en nuestro medio.

El TNC multimodulación permite la inserción de hasta cuatro tarjetas módem, de las cuales se han implementado dos, correspondientes a los 2 tipos de modulación más comunes a la fecha: ASK-Morse y FSK-AX.25. Las tarjetas módem se insertan en la placa principal del TNC multimodulación, de modo que se pueden intercambiar o reemplazar por tipos de modulaciones nuevas. Además, este TNC tiene conectores de expansión que permiten la adición de periféricos externos. De esta manera se tiene un diseño bastante modular, donde se pueden experimentar con diversas tecnologías de hardware y/o software. Por ejemplo, se puede insertar una pequeña tarjeta de depuración hecha con componentes superficiales al puerto de expansión del microcontrolador de control. La Figura 7.1 muestra el TNC multimodulación implementado.

El plan de pruebas consiste en pruebas individuales y prueba del sistema. Dentro de las pruebas individuales se tienen las pruebas de los módems y códecs, así como la prueba del software de seguimiento de satélites (explicada en el capítulo anterior). La prueba de funcionamiento del sistema está enmarcada en 2 escenarios: el primero consiste en la captura de señales reales utilizando el programa de control y seguimiento de satélites, y el segundo escenario consiste en la impresión de resultados de la decodificación de señales de dos tipos de modulación diferentes usando la reconfiguración automática, por medio del software de seguimiento de satélites ejecutado con la interfaz gráfica del TNC.

7.2. Pruebas individuales y del sistema

Pruebas del módem ASK

Se realizaron pruebas funcionales y de medición de parámetros, con señales simuladas y señales reales. Los materiales usados en las pruebas del módem ASK (ver Figura 7.2) fueron:

- Tarjeta módem ASK (basada en los circuitos integrados XR2206 y XR2211).
- Fuente de alimentación AEMC AX502.
- Generador de funciones Motech FG710F.
- Osciloscopio digital Tektronix MSO2012.
- Multímetro digital Fluke 289.
- Laptop con archivo de audio ASK ideal.
- Tarjeta principal del TNC multimodulación configurada en modo manual ASK.

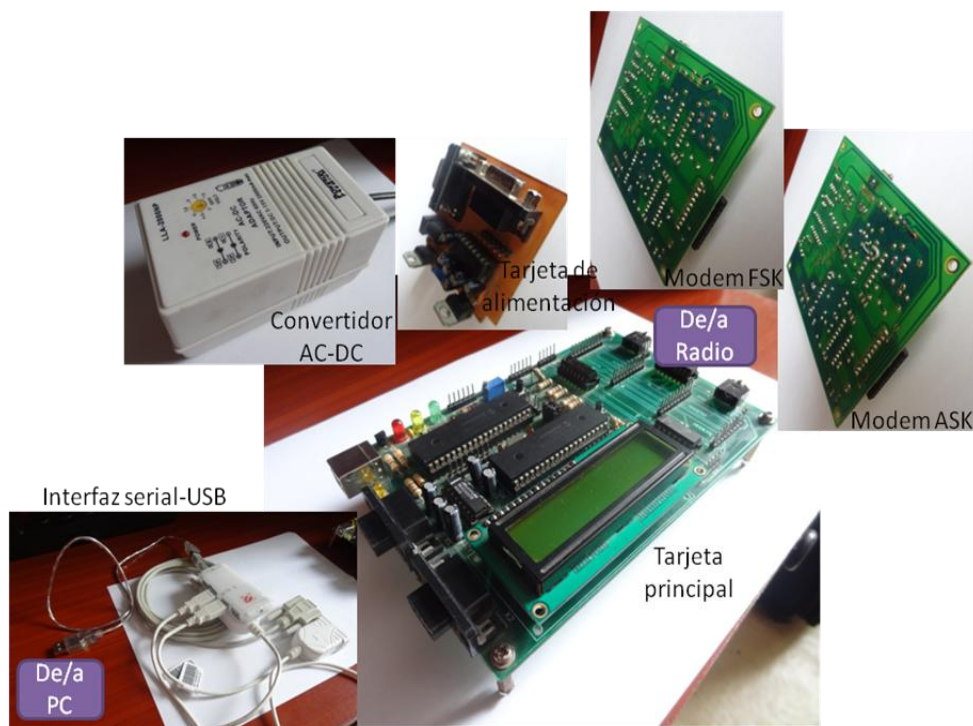


Figura 7.1: TNC multimodulación y sus componentes de hardware

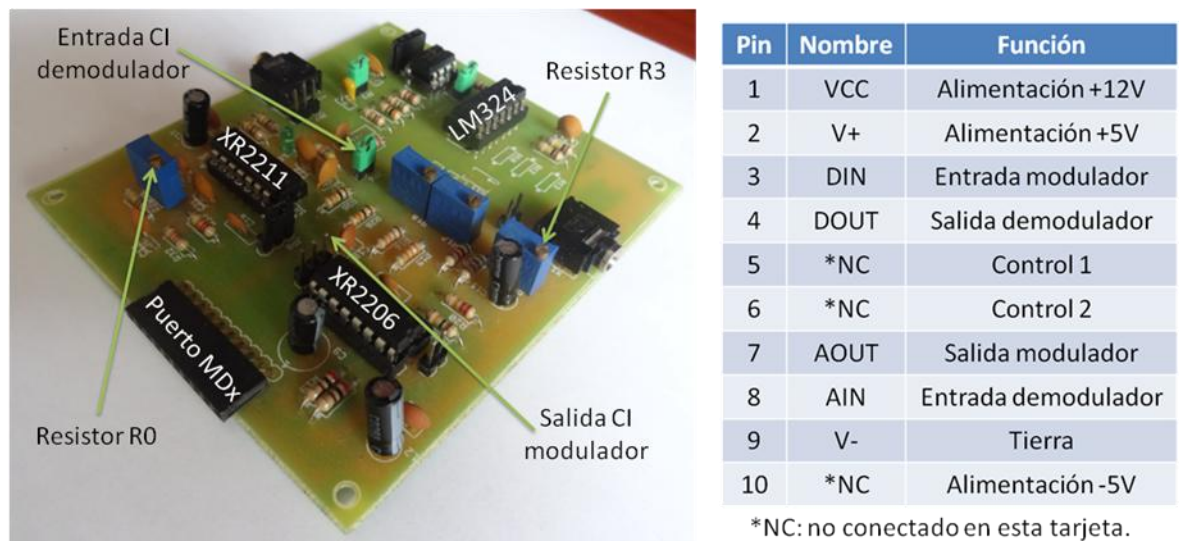


Figura 7.2: Tarjeta módem ASK/FSK y especificación de pines

La primera prueba del módem ASK consistió en ingresar una señal ASK ideal al circuito integrado demodulador XR2211 para verificar la demodulación, ya que la modulación había sido probada en el primer prototipo y no había tenido mayor variación. El procedimiento seguido para la prueba de demodulación de una señal ASK ideal fue:

- Se alimentó la tarjeta módem ASK con la fuente regulada a 12VDC. En caso de observar un consumo de corriente diferente de unos 10mA, se desconecta la alimentación y se revisan las conexiones con el multímetro digital.
- Se conectó la laptop a la tarjeta módem por medio de un cable de audio terminado en conectores tipo espadín macho. Específicamente, se conectó el canal L del cable de audio a la entrada del circuito integrado demodulador y la tierra del cable de audio a la tierra de la tarjeta módem ASK.
- Se ejecutó el archivo de audio ASK ideal en la laptop, creado con el software Audacity.
- Se conectaron los dos canales del osciloscopio a las señales de entrada (canal 1) y salida (canal 2) del circuito integrado demodulador de la tarjeta módem ASK. La Figura 7.3 muestra una captura de pantalla del osciloscopio, donde se observa que la señal decodificada está invertida lógicamente y que hay un retardo en la demodulación.

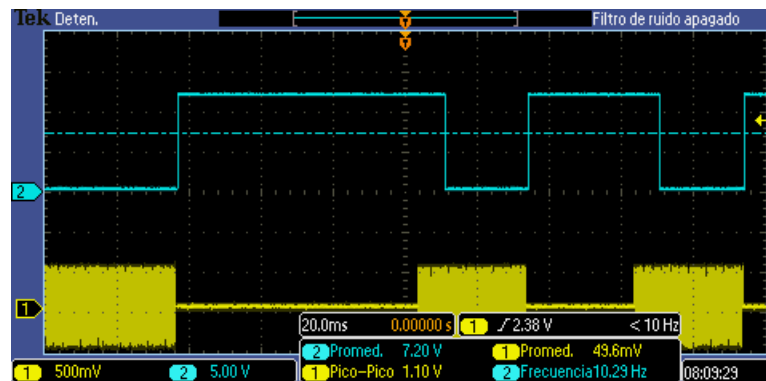


Figura 7.3: Demodulación ASK de señal ideal

La segunda prueba del módem ASK consistió en ingresar una señal ASK real al circuito integrado demodulador para verificar la demodulación con presencia de ruido en la señal de entrada. Los pasos seguidos para esta prueba fueron:

- Se alimentó la tarjeta módem ASK con la fuente regulada a 12VDC. En caso de observar un consumo de corriente diferente de unos 10mA, se desconecta la alimentación y se revisan las conexiones con el multímetro digital.
- Se conectó la laptop a la tarjeta módem por medio de un cable de audio terminado en conectores tipo espadín macho. Específicamente, se conectó el canal L del cable de audio a la entrada del circuito integrado demodulador y la tierra del cable de audio a la tierra de la tarjeta módem ASK.
- Se ejecutó el archivo de audio ASK real en la laptop, adquirido con el sistema de control de seguimiento de satélites.
- Se conectaron los dos canales del osciloscopio a las señales de entrada (canal 1) y salida (canal 2) del circuito integrado demodulador de la tarjeta módem ASK. La Figura 7.4 muestra una captura de pantalla del osciloscopio, donde se observa que la señal decodificada está invertida lógicamente y que hay un retardo en la demodulación.

La tercera prueba consistió en ajustar el valor de la resistencia de sincronización (R_0) del demodulador ASK para lograr una simetría en la demodulación de los bits cero y uno (que corresponden a presencia y ausencia de señal). Para ello se ingresó una señal ASK ideal periódica que cambia cada 50ms

al circuito integrado demodulador y se midió el tiempo de bit 1, que debe ser lo más cercano posible a 50ms, para varios valores de R0. El procedimiento seguido fue:

- Se alimentó la tarjeta módem ASK con la fuente regulada a 12VDC.
- Se conectó la salida de audio de la laptop a la entrada del circuito integrado demodulador de la tarjeta módem ASK, por medio de un cable de audio.
- Se ejecutó un archivo de audio ASK ideal, que conmutaba cada 50ms entre la presencia y ausencia de audio de frecuencia 1200Hz.
- Se midió con ayuda del osciloscopio el tiempo del bit 1 (ausencia de señal de audio) para un valor de R0 determinado, y se tomó nota de estos valores.

Se varió el valor de R0 entre 23.5 kOhmios y 28.5 kOhmios (alrededor del valor teórico), y se tomaron nota de los tiempos de bit 1. Con estos datos se formó la gráfica de la figura 7.5, donde se observa que la mejor respuesta se da para un valor de R0 entre 25 y 25.5 kOhmios, con un tiempo de bit uno de 55.6 ms, lo que se observa en la Figura 7.5.

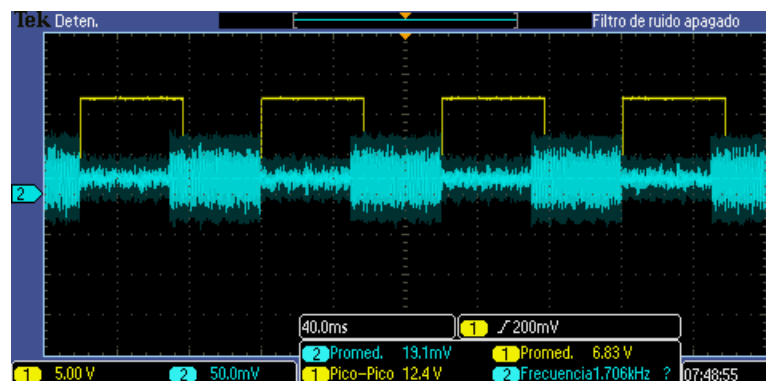


Figura 7.4: Demodulación ASK de señal real

La cuarta prueba consistió en ingresar al circuito integrado demodulador una señal ASK ideal que conmutaba entre presencia y ausencia de señal de audio cada 50 ms, y medir el tiempo de bit uno para diferentes valores de la alimentación de la tarjeta módem. El procedimiento seguido en este caso fue:

- Se alimentó la tarjeta módem ASK con una fuente regulada a 12VDC inicialmente.
- Se conectó la salida de audio de la laptop a la entrada del circuito integrado demodulador de la tarjeta módem ASK, por medio de un cable de audio.

- Se ejecutó un archivo de audio ASK ideal, que conmutaba cada 50ms entre la presencia y ausencia de audio de frecuencia $f_0 = 1200$ Hz.
- Se midió con ayuda del osciloscopio el tiempo del bit uno (ausencia de señal de audio) para un valor de V_{cc} inicial de 12VDC, y se tomó nota de estos valores.
- Se varió el valor de V_{cc} entre 7 V y 12 V (cerca del valor inicial de diseño), y se tomaron nota de los tiempos de bit uno. Con estos datos se formó la gráfica de la figura 7.6, donde se observa que la mejor respuesta se da para un valor de $V_{cc} = 7.5$ V, con un tiempo de bit uno de 52.8 ms, lo que se observa en la Figura 7.6. Sin embargo, según la hoja de datos del circuito integrado modulador XR2206, el valor mínimo de voltaje de alimentación requerido es de 10VDC, por lo que se eligió este valor para la tarjeta módem ASK.

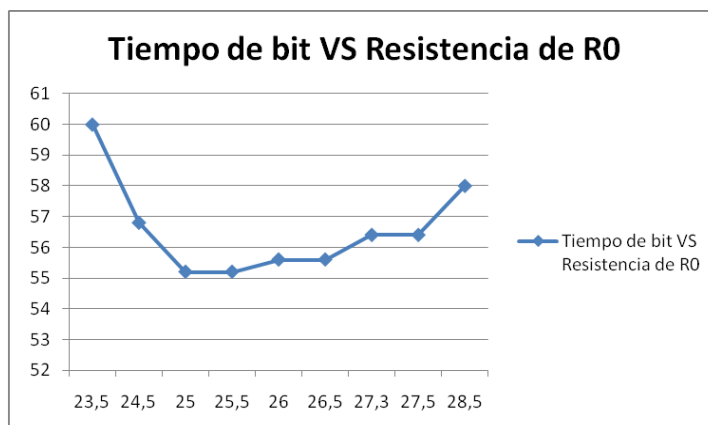


Figura 7.5: Variación del tiempo de bit en función del resistor de timing

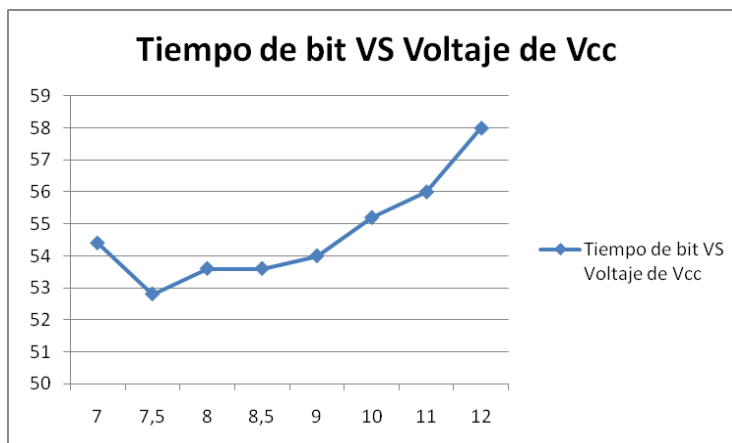


Figura 7.6: Variación del tiempo de bit en función de la alimentación

La quinta prueba consistió en la verificación de funcionamiento del algoritmo de decodificación Morse, específicamente en la medición de los tiempos de señal, en el cual se basa su principio de operación. Para esta prueba se ingresó la señal ASK ideal y simétrica a la entrada del circuito integrado demodulador de la tarjeta módem ASK, y la salida de dicho demodulador se conectó a la tarjeta principal del TNC multimodulación, configurado manualmente en modo ASK-Morse. El procedimiento seguido fue el siguiente:

- Se insertaron las tarjetas módem ASK y de alimentación a la tarjeta principal del TNC multimodulación.
- Se conectó el puerto de datos del TNC multimodulación a la laptop por medio de un cable USB-serial, y la salida de audio de la laptop a la entrada de audio del TNC por medio de un cable de audio estéreo.
- Se alimentó el TNC, programado previamente para medir el tiempo de presencia de señal ASK, con el convertor AC/DC regulado a 10VDC.
- Se ejecutó el audio ASK ideal en la laptop, generado con el software Audacity.
- Se ejecutó el software X-CTU en la laptop, el cual muestra en pantalla los caracteres recibidos por el puerto USB-serial de la laptop. En vista que se programó el TNC para enviar los tiempos de bit uno (corregida la lógica esta vez en la tarjeta módem ASK para corresponder a presencia de señal de audio), se observó en pantalla el tiempo en ms, que es menor al valor ideal de 50ms debido al retardo en la demodulación. Se observa en las Figuras 7.7 y 7.8 que los tiempos medidos son correctos.

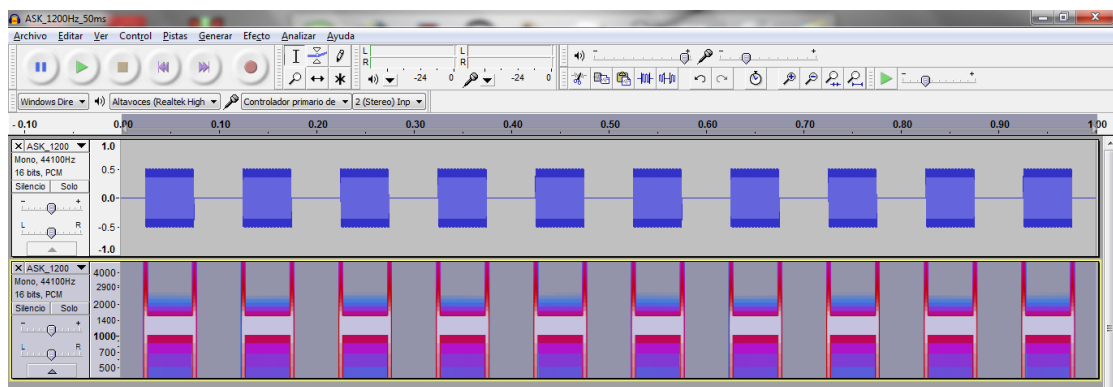


Figura 7.7: Señal ASK ideal en el dominio del tiempo y frecuencia

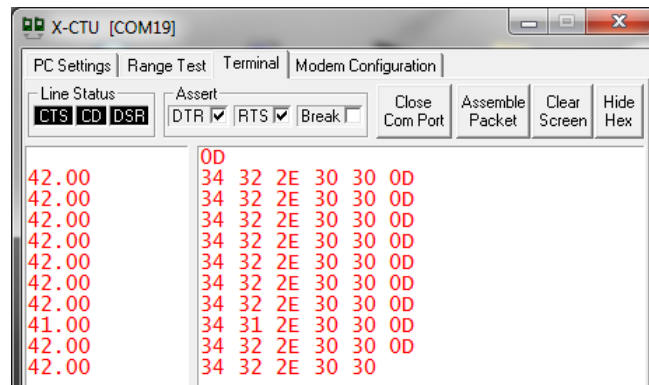


Figura 7.8: Medición de tiempos de señal ASK ideal demodulada

Pruebas del módem FSK

La tarjeta módem FSK tiene el mismo diseño de tarjeta impresa que la tarjeta módem ASK, la cual se muestra en la Figura 7.2; lo que varía entre estas dos tarjetas son los valores de algunos resistores o capacitores, y la configuración de jumpers. Se realizaron pruebas funcionales y de medición de parámetros a la tarjeta módem FSK, con señales simuladas y señales reales. Los materiales usados en las pruebas (ver Figura 7.9) fueron:

- Tarjeta módem FSK (basada en los circuitos integrados XR2206 y XR2211).
- Fuente de alimentación AEMC AX502.
- Generador de funciones Motech FG710F.
- Osciloscopio digital Tektronix MSO2012.
- Multímetro digital Fluke 289.
- Laptop con archivos de audio FSK ideales y reales.
- Tarjeta principal del TNC mutlimodulación configurado en modo manual FSK.



Figura 7.9: Entorno de pruebas de la tarjeta módem FSK/ASK

La primera prueba consistió en generar una señal FSK con el circuito modulador de la tarjeta módem FSK. Para ello se ingresó un tren de pulsos de onda cuadrada de 5V de amplitud y 1200Hz de frecuencia, suministrado por el generador de funciones, a la entrada del modulador de la tarjeta módem FSK (pin 3 del puerto de conexiones MD2). Se capturaron las señales de entrada y salida del circuito integrado modulador XR2206 con un osciloscopio. Dichas señales se muestran en la Figura 7.10, donde se observa que la forma de onda de señal FSK generada (en el canal 2) es continua y suave. Además, el valor pico a pico de la señal FSK generada puede ajustarse con el valor de resistencia R3.

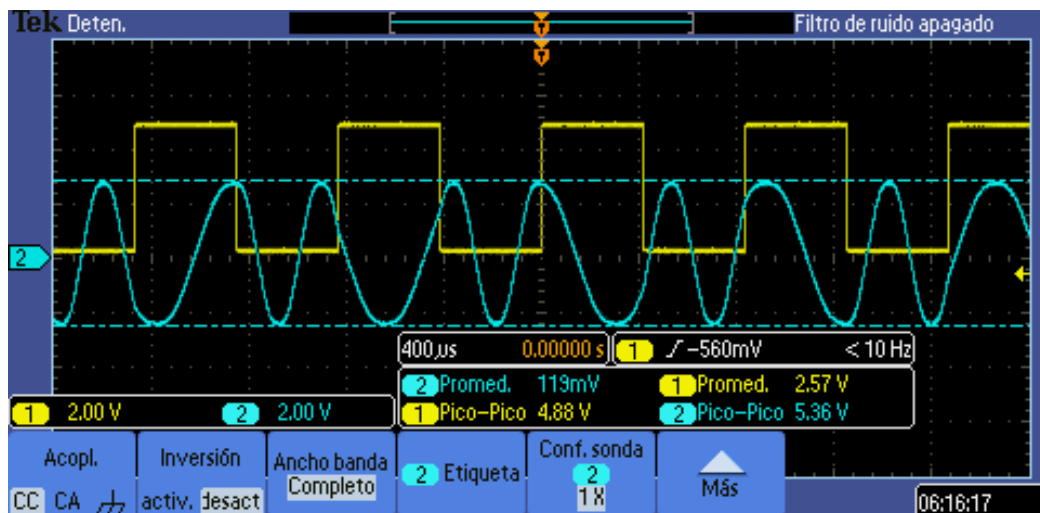


Figura 7.10: Generación de señal modulada en FSK

La segunda prueba consistió en ingresar una señal FSK ideal a la entrada del circuito integrado demodulador XR2211 de la tarjeta módem FSK. Dicha señal FSK ideal fue suministrada por el generador de funciones configurado en modo FSK: $f_1=1200\text{Hz}$, $f_2=2200\text{Hz}$, $T=3\text{ms}$, $A=500\text{ mVpp}$. Las señales de entrada del circuito integrado demodulador (canal 1, color amarillo) y salida del demodulador (canal 2, color celeste) fueron capturadas con el osciloscopio y se muestran en la Figura 7.11. En esta prueba se realizó la calibración del resistor de sincronización R0 de la tarjeta módem FSK, cuyo valor fue ajustado hasta obtener una señal demodulada simétrica en el osciloscopio.

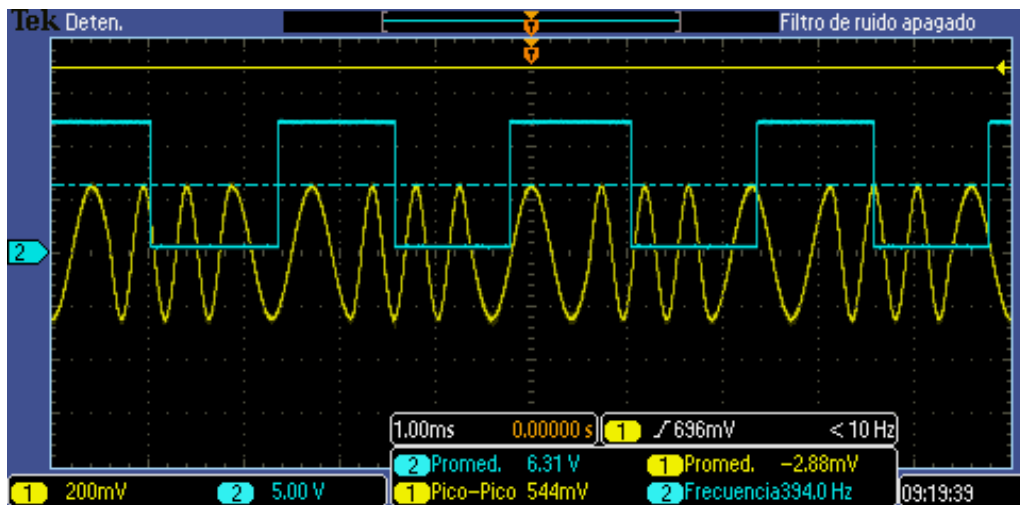


Figura 7.11: Demodulación de señal FSK ideal

La tercera prueba consistió en ingresar una señal FSK real a la entrada del circuito integrado demodulador XR2211 de la tarjeta módem FSK. Esta señal FSK real correspondía a un archivo de audio APRS obtenido de la web, el que fue reproducido con una laptop. Las señales de entrada del circuito integrado demodulador (canal 2, en color celeste) y salida del demodulador (canal 1, en color amarillo) de la tarjeta módem FSK fueron capturadas con un osciloscopio y se muestran en la Figura 7.12, donde se aprecia que la señal FSK real fue correctamente demodulada.

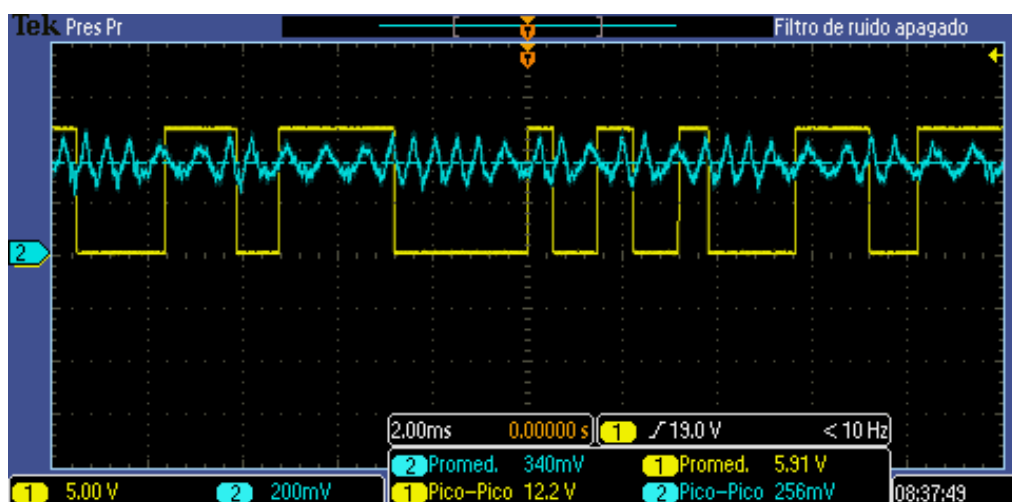


Figura 7.12: Demodulación de señal FSK real

La cuarta prueba de la tarjeta módem FSK corresponde a una prueba de probabilidad de error, que consistió en ingresar una señal digital correspondiente a un tren de pulsos periódicos, conectar la salida del circuito integrado modulador a la entrada del circuito integrado demodulador de la tarjeta módem FSK, y contar el número de pulsos demodulados correctamente para una frecuencia de pulsos de entrada determinada. Por ejemplo, cuando la tasa de rapidez de la señal digital de entrada era de 340bps, el error porcentual (número de bits errados dividido por el número total de bits) en la recuperación de dicha señal digital luego de modular-demodular con la tarjeta módem FSK fue de 1%. Se varió la frecuencia de la señal digital de entrada y se tomaron nota de los porcentajes de error resultantes, y la gráfica resultante se muestra en la Figura 7.13. Se verificó que los circuitos integrados modulador y demodulador operan correctamente para una tasa de rapidez de señal digital de 1200bps (error cercano a 2%) o menor.

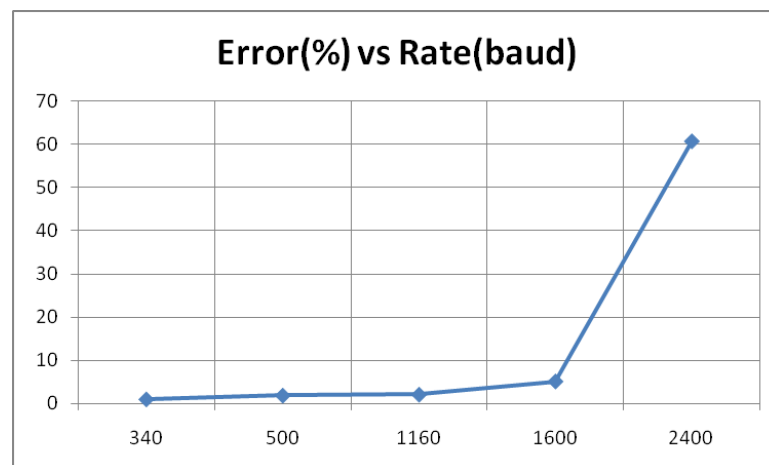



Figura 7.13: Variación del error con la tasa de bits en módem FSK

La quinta prueba de la tarjeta módem FSK consistió en la prueba de integración con el software UI-View. En vista de que el códec AX.25 (microcontrolador de procesamiento de la tarjeta principal) es compatible con el protocolo KISS, se pueden usar softwares comerciales para interpretar señales reales moduladas en FSK y codificadas en AX.25. Es así que se usó un archivo de audio APRS como entrada de la tarjeta módem FSK, la cual fue insertada en la tarjeta principal del TNC multimodulación configurado en modo manual FSK-AX.25. La salida de datos del TNC se conectó a una laptop donde se había instalado y configurado el software UI-View para interpretar las tramas KISS enviadas por el

TNC. Los mensajes decodificados se muestran en la Figura 7.14, donde se aprecia el formato de un mensaje APRS (que es el mismo de un paquete AX.25 no numerado): Emisor>Receptor,Repetidor:Mensaje.



```

04051503.TXT - Bloc de notas
Archivo Edición Formato Ver Ayuda
15:02:46R WA6YLB-4>APRS,N6EX-5* <UI Len=59>:
$ULTW00000000----0000----000086A00001----0000000000000000
15:02:48R KD6FVP-2>APS224,N6EX-1,WIDE1 <UI Len=45>:
>152343z[224]*we know most of your faults!!!
15:02:49R KD6FVP-2>APS224,N6EX-1*,WIDE1 <UI Len=45>:
>152343z[224]*we know most of your faults!!!
15:02:52R KC6HUR-1>S4QVYV,W6SCE-10* <UI R Len=15>:
'.4&l-/k/"7q}
15:02:53R N6XQY-12>GPSLJ,N6EX-4* <UI R Len=71>:
$GPRMC,013641.06,A,3348.1607,N,11807.4631,W,34.0,090.5,231105,13.,E*73
15:02:54R KC6BLF-14>S4PWYS,N6EX-5* <UI R Len=32>:
'-u l{(u/"5\}Lost in the West!
15:02:55R K6KMA-1>GPSLK,N6EX-1* <UI Len=70>:
$GPRMC,013647,A,3350.076,N,11806.996,W,028.3,180.5,231105,013.5,E*69
15:02:57R AE6GR-7>S4PXW,N6EX-1* <UI R Len=15>:
'.-[] tv/"6{
15:02:59R AE6MP>SS5PPQ-2,N6EX-4* <UI Len=13>:
'.](n->>/"4w}
15:02:59R AE6MP>SS5PPQ-2,N6EX-1* <UI Len=13>:
'.](n->>/"4w}
15:03:00R AE6MP>SS5PPQ-1,N6EX-5* <UI Len=13>:
'.](n->>/"4w}
15:03:03R K6LAR-1>APRS,N6EX-1* <UI Len=68>:
$GPGGA,040332,3405.438,N,11801.836,W,1,06,1.1,114.2,M,-31.5,M,,*75
15:03:04R KD6UZM-15>S3UWTS,WB6JAR-10*,WIDE2-1 <UI Len=13>:
'-)01 0v\"r}
15:03:05R KD6UZM-15>S3UWTS,WB6JAR-10*,N6EX-1* <UI Len=13>:
'-)01 0v\"r}
15:03:06R N6QFD-9>GPSLJ,N6EX-5* <UI Len=70>:
$GPRMC,013714,A,3408.6360,N,11812.0716,W,0.0,88.1,231105,13.5,E,D*09

```

Figura 7.14: Interpretación de señal FSK real con UI-VIEW

Prueba 1 del sistema: seguimiento automático de satélites y captura de datos.

La prueba del mecanismo de selección y algoritmo de seguimiento consiste en capturar señales reales de satélites en órbita controlando los elementos necesarios de la estación terrena mediante el programa de seguimiento de satélites, explicado en el capítulo anterior, que entre otras funciones, determina el modo de operación del satélite disponible. Para conseguir una señal adecuada para interpretar, se deben contrarrestar los efectos del movimiento orbital del satélite, y sus consecuencias directas en la transmisión de señal del satélite. Por tanto, se deben direccionar las antenas, controlar la frecuencia de la radio y seleccionar el módem adecuado del TNC multimodulación.

La radio es el equipo electrónico que captura, junto con antenas VHF/UHF, las señales enviadas por satélites de radioaficionados. El computador es el encargado de almacenar la información recibida y ejecutar el algoritmo de seguimiento automático de satélites, por lo tanto controla la frecuencia de la radio y

la orientación de las antenas. El TNC multimodulación convierte las señales de audio obtenidas de la radio, las digitaliza y decodifica para enviar la información obtenida hacia el computador. La Figura 7.15 muestra el escenario 1 de pruebas del TNC multimodulación, donde se emplearon todos los elementos de la estación terrena:

- Radio transceptor ICOM IC-910H.
- Controlador de radio Rig Expert Plus de REU.
- Rotor de antenas Yaesu G-5500.
- Controlador de rotor ARS-SE de EA4TX.
- Antena tipo X-Quad para VHF de Wimo.
- Antena tipo X-Quad para UHF de Wimo.
- TNC multimodulación.
- PC con software de gestión.

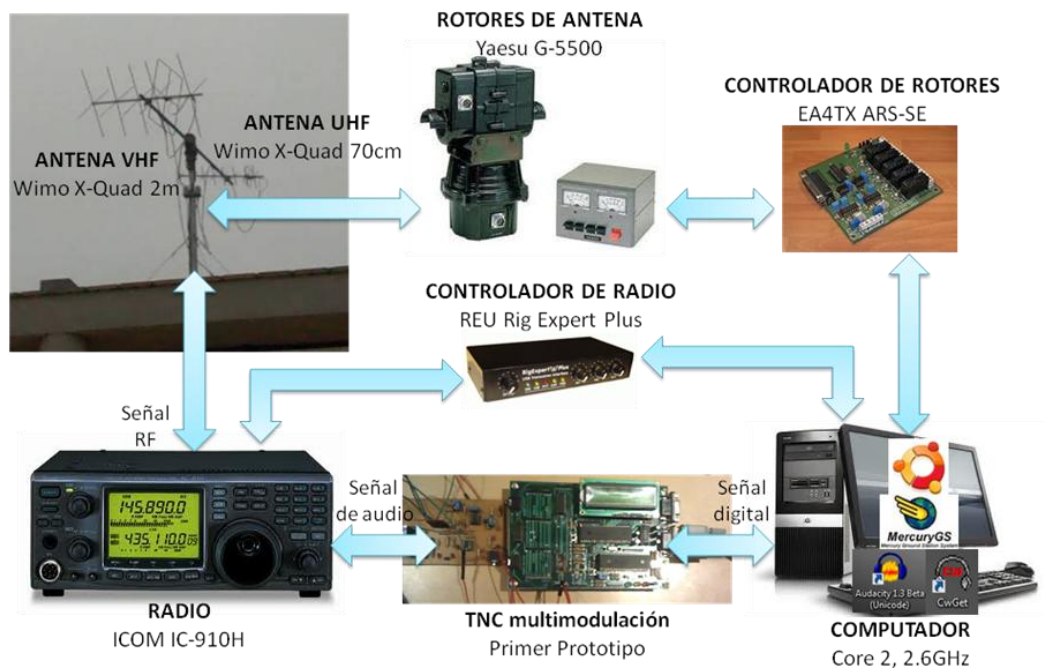


Figura 7.15: Escenario 1 de pruebas del TNC multimodulación

Si la antena no está correctamente direccionada es probable que no se reciba señal alguna; aún si se recibe inicialmente, la dirección cambia constantemente desde que aparece el satélite hasta que queda fuera del alcance. La antena se puede direccionar manualmente o con ayuda de rotores

electromecánicos, los cuales a su vez pueden manejarse manualmente o por medio de un controlador electrónico con interfaz para computador.

Por otra parte, si la frecuencia de recepción de la radio no se actualiza para compensar el efecto Doppler, la frecuencia de la señal de audio recibida cambiará progresivamente hasta llegar al rango inaudible. La frecuencia de la radio se puede variar manualmente o de manera remota por medio de un controlador electrónico con interfaz para computador. La Figura 7.16 muestra una señal de audio modulada en ASK y codificada en Morse conocida como señal Beacon, emitida por el satélite japonés CO-55, la cual ha sido capturada sin el sistema de seguimiento de satélites, con las antenas direccionadas inicialmente hacia el satélite. Se observa que la frecuencia de señal de audio recibida disminuye progresivamente si es que no se resintoniza la radio de forma manual o remota.

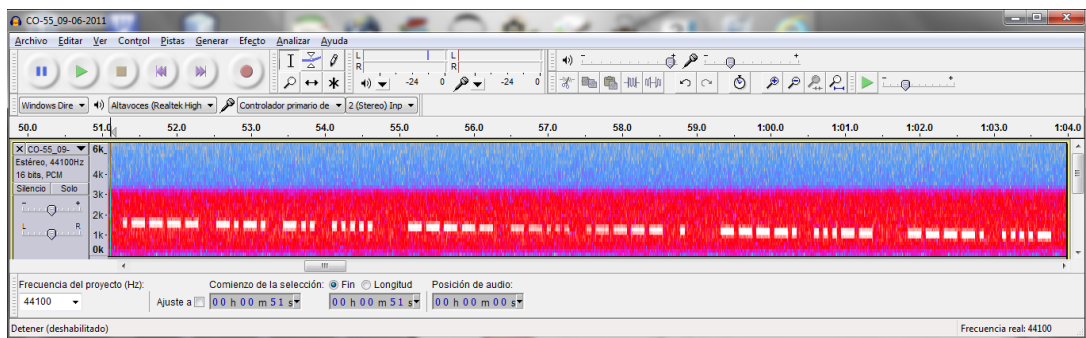


Figura 7.16: Señal ASK de CO-55 recibida sin el sistema de seguimiento

La Figura 7.17 muestra una gráfica en el dominio de la frecuencia de una señal de audio capturada usando el sistema de seguimiento, para el satélite ruso RS-30. La señal de audio se obtiene conectando directamente la radio hacia la entrada de audio del computador y grabando dicha señal con un programa como Audacity. Se observa que la frecuencia de la señal de audio recibida es constante, lo que ayuda a hacer el circuito de filtrado de los módem ASK y FSK más simple en vista que la frecuencia de la señal es constante. Esta frecuencia de señal recibida se calibra cambiando el archivo de modos y frecuencias del software de control del TNC multimodulación, porque la frecuencia de audio recibida es la diferencia de la frecuencia enviada por el satélite y la frecuencia configurada en la radio.

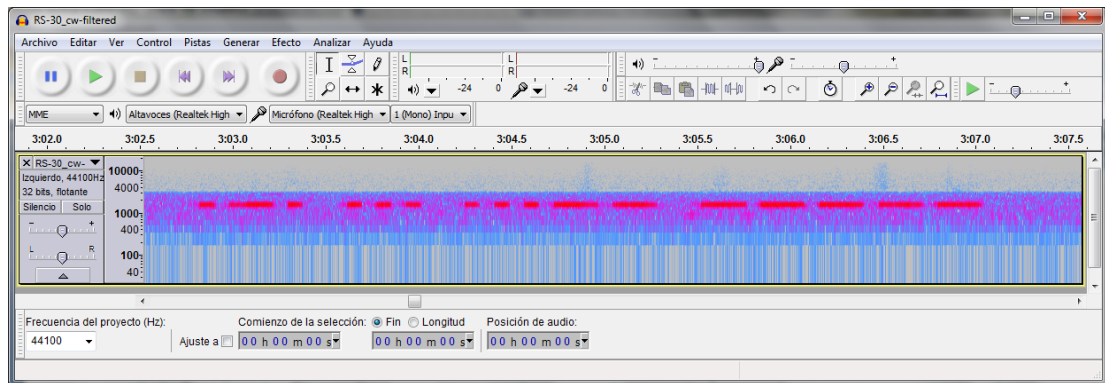


Figura 7.17: Señal ASK de RS-30 recibida con el sistema de seguimiento

A continuación se muestran capturas de señales reales de varios tipos provenientes de satélites en órbita. La Figura 7.18 muestra el programa de seguimiento de satélites, el cual imprime en pantalla los parámetros del satélite CO-57 que estaba disponible, de los cuales los ángulos de azimuth y elevación y la frecuencia compensada son usados para controlar los elementos de la estación terrena. La Figura 7.19 muestra el computador de datos de la estación terrena al momento de recibir un paquete AX.25 proveniente del satélite americano FASTRAC1, el cual fue grabado directamente de la salida de audio de la radio. La Figura 7.20 muestra la manipulación del volumen de la radio ante la recepción de señal de voz proveniente del satélite ruso RadioSkaf2; de igual manera como en los casos anteriores, la frecuencia de la radio y dirección de las antenas eran controladas por el programa de seguimiento de satélites, el cual posteriormente se incluyó en el software de control del TNC multimodulación para la configuración actual.

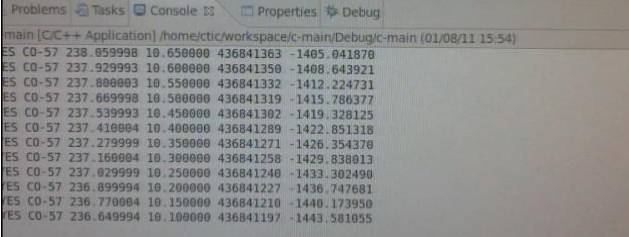
```

#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <errno.h>
#include <arpa/inet.h>

typedef enum {
    stateSearch = 0, stateTrack
} stateType;

const long freqDown[24] = { 437345000, 145825000, 437305000, 435790000, //435795000 (2
437325000, 437485000, 435352000, 435790000, 437305000, 436837500, //437490000
145825000, 436837500, 437325000, 436847500, 436847500, 437505000,

```



```

main [C/C++ Application] /home/ctc/workspace/c-main/Debug/c-main (01/08/11 15:54)
ES CO-57 238.859998 10.650000 436841363 -1405.041870
ES CO-57 237.929993 10.680000 436841350 -1408.643921
ES CO-57 237.800003 10.550000 436841332 -1412.224731
ES CO-57 237.609998 10.500000 436841319 -1415.786377
ES CO-57 237.539993 10.450000 436841302 -1419.328125
ES CO-57 237.410004 10.400000 436841289 -1422.851318
ES CO-57 237.279999 10.350000 436841271 -1426.354370
ES CO-57 237.160004 10.300000 436841258 -1429.838013
ES CO-57 237.029999 10.250000 436841240 -1433.302490
ES CO-57 236.899994 10.200000 436841227 -1436.747681
ES CO-57 236.770004 10.150000 436841210 -1440.173950
ES CO-57 236.649994 10.100000 436841197 -1443.581055

```

Figura 7.18: Recepción ASK de CO-57 con el sistema de seguimiento

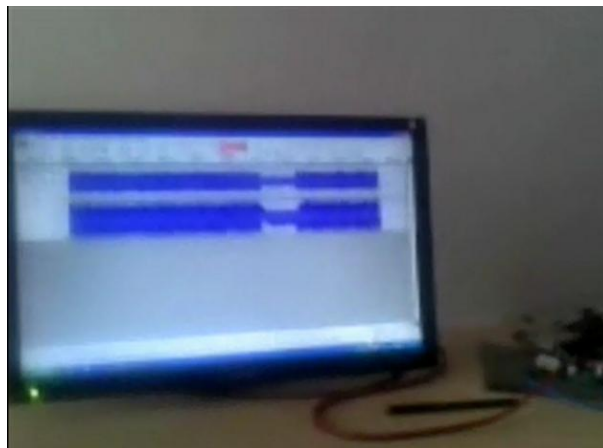


Figura 7.19: Recepción FSK de FASTRAC con el sistema de seguimiento



Figura 7.20: Recepción voz de RadioSkaf2 con el sistema de seguimiento

Prueba 2 del sistema: selección de módem y decodificación automática.

El programa de seguimiento automático de satélites, que genera las señales de control para los rotores de antena, frecuencia de radio y microcontrolador de control del TNC, funciona de manera independiente al algoritmo de decodificación de las señales capturadas con el sistema de seguimiento. Por esta razón, es posible simular la captura de señales de satélites con archivos de audio reales previamente capturados, los cuales se ingresan al TNC multimodulación para probar los algoritmos de decodificación e impresión de datos en un computador. La Figura 7.21 muestra el escenario 2 de pruebas del TNC multimodulación, donde se usan un computador portátil y el TNC multimodulación (la radio, el rotor y la antena con sus interfaces respectivas son simuladas dentro del mismo computador portátil).

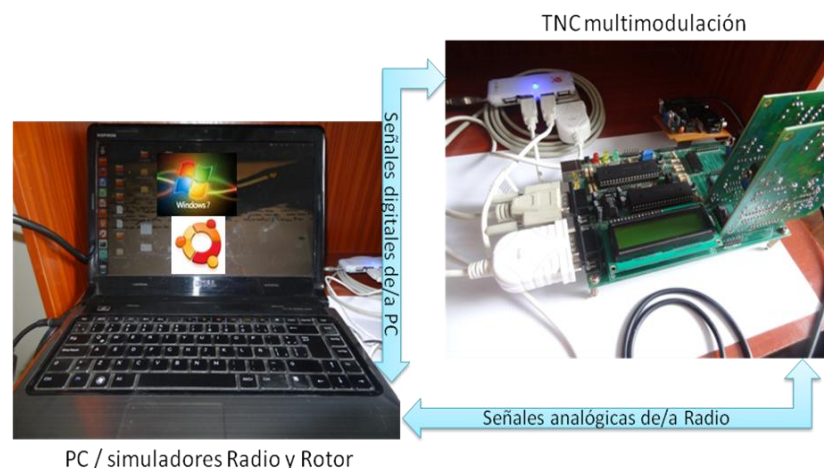


Figura 7.21: Escenario 2 de pruebas del TNC multimodulación

La prueba de selección automática de módem e interpretación de las señales recibidas consistió en dos partes: la prueba del hardware (microcontroladores de control y procesamiento y módems) y la prueba del software (activación de algoritmo de seguimiento e interpretación de datos recibidos). La prueba del hardware consistió en controlar los selectores de modulación mediante las señales de control recibidas por el puerto de control conectado al computador, así como la decodificación de señales recibidas por el puerto de audio y envío de datos decodificados al computador por medio del puerto de datos. La Figura 7.22 muestra el entorno de pruebas, indicación de satélite disponible y satélite siguiente.



Figura 7.22: Prueba de algoritmo de control del TNC multimodulación

La prueba del software consistió en la edición de archivos de configuración, activación de envío de las señales de control e impresión de los datos interpretados correspondientes a dos tipos de modulación. La Figura 7.23 muestra capturas de pantalla de la impresión de datos recibidos en ASK (izquierda, grabación de señal ideal) y de datos recibidos en FSK (derecha, grabación de señal real). Se concluye que el sistema es capaz de interpretar señales de satélites de dos tipos de modulaciones distintas de manera automática.



Figura 7.23: Prueba de decodificación automática de dos modulaciones

7.3. Consumo de recursos

El consumo de recursos se puede dividir en: consumo de recursos en servicios y consumo de recursos de hardware, de firmware y de software. El consumo de recursos en servicios (ver Tabla 7.1) corresponde a los servicios requeridos de fabricación y soldadura de tarjetas. El consumo de hardware (ver Tabla 7.2) corresponde a los costos de los dispositivos de hardware, mientras que el consumo de firmware (ver Tabla 7.3) corresponde a la ocupación de memoria de los dos dispositivos programables usados. El consumo de software (ver Tabla 7.4) corresponde al espacio de memoria ocupado por los archivos ejecutables de la interfaz gráfica del TNC.

Cantidad	Descripción	Código	Referencia	Precio (S/.)
1	Tarjeta impresa	2capas 16x11cm	tnc-mmd	250
1	Tarjeta impresa	1capa 10x9cm	(x2)ask-fsk	70
1	Tarjeta impresa	1capa 6x6cm	tncm_ps	3
4	Soldado de tarjeta	4 tarjetas	4 tarjetas	200
			Total servicios	523

Tabla 7.1: Consumo de recursos en servicios

Cantidad	Descripción	Código	Referencia	Precio (S/.)
1	Circuito Integrado	PIC18F4550	tnc-mmd.IC1	35
1	Circuito Integrado	PIC16F877A	tnc-mmd.IC2	16
1	Circuito Integrado	MAX232	tnc-mmd.IC3	2
1	Circuito Integrado	4052N	tnc-mmd.IC4	1
1	Circuito Integrado	74S139N	tnc-mmd.IC5	2
1	Circuito Integrado	74S153N	tnc-mmd.IC6	2
1	Pantalla LCD	LCD 16x2	tnc-mmd.DIS1	12
5	Capacitor cerámico	22pF(x4), 220nF	tnc-mmd.C1-4,10	1
5	Capacitor electrolítico	1uF(x5)	tnc-mmd.C5-9	1
2	Diodo rectificador	1N4148	tnc-mmd.D1-2	0,4
1	Conector espadín	Macho 40x1	tnc-mmd.JP1-2,4-5	1
4	Diodo LED	LED 5mm	tnc-mmd.LED1-4	1,2
2	Oscilador de cristal	20MHz	tnc-mmd.Q1-2	2
1	Transistor bipolar	BC549C	tnc-mmd.Q3	0,5
17	Resistor	Varios	tnc-mmd.R1-17	1,7
2	Interruptor pulsador	Botón 4 pos	tnc-mmd.S1-2	1
2	Conector espadín	Macho 40x1	tnc-mmd.SV1-10	2

2	Conector serial	DE-9 hembra	tnc-mmd.X1-2	4
2	Conector audio	Estéreo hembra	tnc-mmd.X3-4	2
1	Conector USB	USB-A hembra	tnc-mmd.X5	2
2	Circuito Integrado	LM324N	(x2)ask-fsk.IC1	4
2	Circuito Integrado	MCP41010	(x2)ask-fsk.IC2	16
2	Circuito Integrado	XR-2206	(x2)ask-fsk.IC3	32
2	Circuito Integrado	XR-2211	(x2)ask-fsk.IC4	56
44	Capacitor	Varios	(x2)ask-fsk.C1-22	8,8
2	Conector espadín	Macho 40x1	(x2)ask-fsk.JP1-8	2
2	Diodo LED	LED 3mm	(x2)ask-fsk.LED1	0,4
74	Resistor	Varios	(x2)ask-fsk.R1-37	7,4
2	Conector espadín	Hembra 40x1	(x2)ask-fsk.SV1	2
2	Conector espadín	Hembra 40x1 90°	(x2)ask-fsk.SV2	2
4	Conector audio	Estéreo hembra	(x2)ask-fsk.X1-2	4
1	Circuito Integrado	LM317	tncm_ps.IC1	2
1	Circuito Integrado	7805TV	tncm_ps.IC2	2
1	Circuito Integrado	MAX232	tncm_ps.IC3	2
2	Capacitor cerámico	0.1uF(x2)	tncm_ps.C1,3	0,4
6	Capacitor electrolítico	1uF(x6)	tncm_ps.C2,4-8	1,2
1	Resistor fijo	240 Ohm	tncm_ps.R1	0,1
1	Resistor variable	5k Ohmios	tncm_ps.R2	1
1	Conector espadín	Hembra 40x1	tncm_ps.SV1	1
1	Conector espadín	Macho 40x1	tncm_ps.SV2-3	1
1	Conector DC	Jack DC	tncm_ps.X1	1
2	Conector serial	DE-9 hembra	tncm_ps.X2-3	4
2	Cable USB-serial	Trendnet TU-S9	externo	60
1	Puente USB	ENU-4 USB	externo	12
1	Adaptador AC-DC	Toyoba LLA-220N	externo	30
			Total componentes	342,1

Tabla 7.2: Consumo de recursos de hardware

Dispositivo	Consumo ROM	Consumo RAM
Microcontrolador de control	23 de 368 bytes (6%)	1164 de 8192 bytes (14%)
Microcontrolador de proceso	3k de 32k bytes (11%)	1k de 2k bytes (54-55%)

Tabla 7.3: Consumo de recursos de firmware

Archivo	Función	Peso en Bytes
tncM.jar	Ejecutable Java principal	17.017 MB
backGround	Programa de seguimiento	104 kB
exec.pl	Script para ejecutar programa	1 kB
satelites.tle	Lista de satélites con sus TLE	4 kB
estacion.qth	Ubicación de estación terrena	1 kB
modos.txt	Frecuencia y modo de satélites	1 kB

Tabla 7.4: Consumo de recursos de software

7.4. Comparación con TNCs similares

Para dar una idea del alcance del TNC propuesto, se realiza una comparación con TNCs hardware y software similares, así como la comparación con un software de predicción y control de equipos de la estación terrena, ya que el software del TNC propuesto tiene funcionalidades de control de equipos de la estación terrena. La Tabla 7.5 muestra la comparación de complejidad (se puede medir por el número de componentes usados [20], especialmente de los circuitos integrados), flexibilidad, consumo de potencia, precio, capacidad de control y de reconfiguración con otros similares.

Parámetros	TNC propuesto	KAM-XL	MixW	GPredict
Complejidad	14 CI	28 CI	0	0
Flexibilidad	Hasta 4 modos	15 modos	15 modos	0
Consumo	200mA@10V	150mA@12V	Alto (PC)	Alto (PC)
Precio	350 S/.	470 \$	50 \$	0
Control	TNC-Radio-Rotor	No	Rotor	Radio-Rotor
Reconfiguración	Instantánea	Manual (s)	Manual (s)	No

Tabla 7.5: Comparación del TNC propuesto con otros

7.5. Cronograma de tiempos

El tiempo estimado para la elaboración del presente trabajo fue de un año. Sin embargo, se han tenido retrasos debido a la demora en la adquisición de materiales en un inicio, y luego a la falta de tiempo para el desarrollo del trabajo. El cronograma de tiempos estimados fue realizado con el programa Open Proj y se muestra en la Figura 7.24.

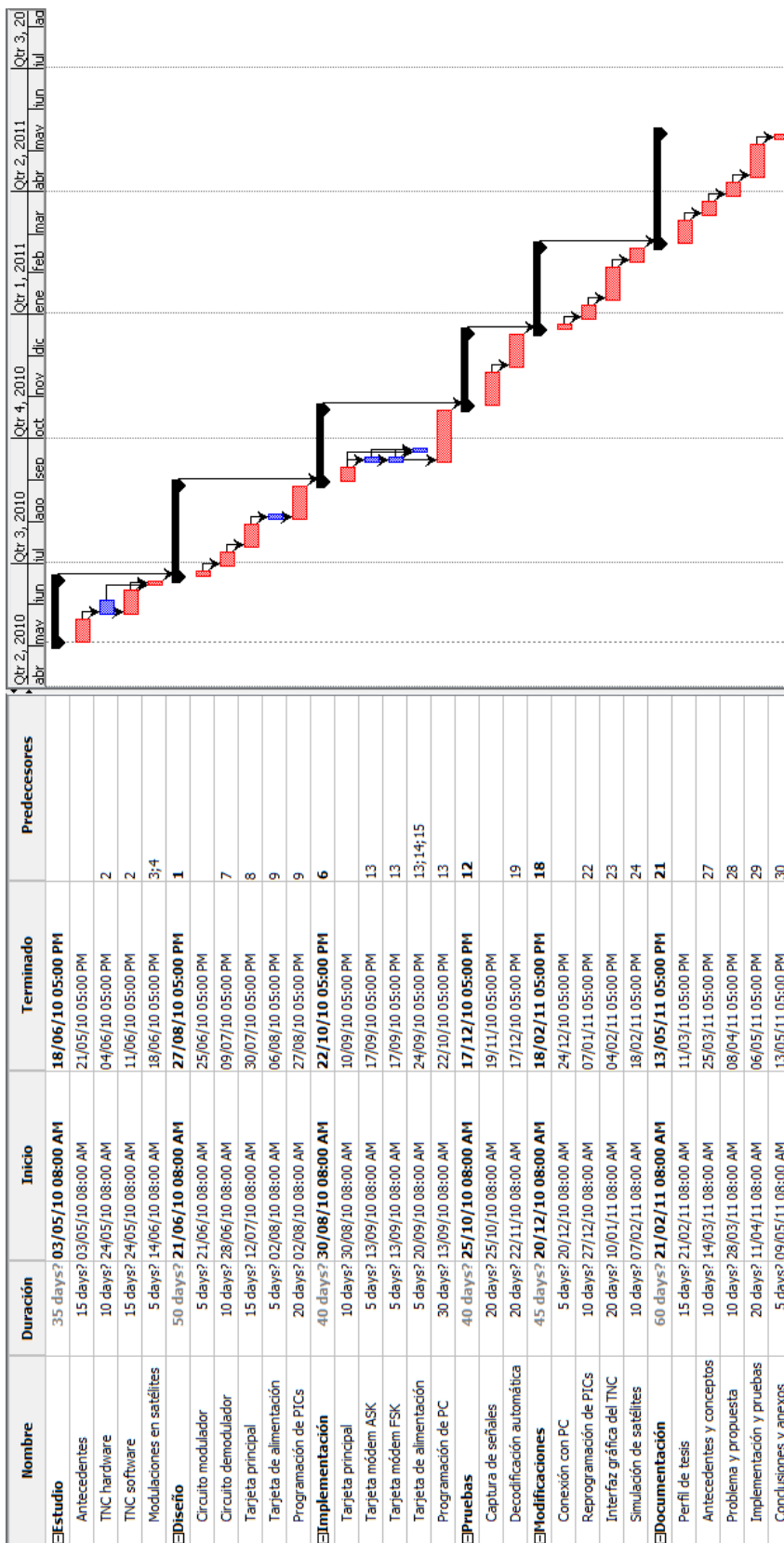


Figura 7.24: Cronograma de tiempos estimados

CONCLUSIONES

- ✓ Se desarrolló un TNC multimodulación con componentes hardware de bajo costo y software de código abierto. El sistema incluye un software para la PC que calcula los datos del satélite artificial disponible, envía señales de control para la frecuencia de la radio y la posición de los rotores de la estación terrena, y envía el tipo de modulación al TNC, que selecciona el módem hardware necesario y envía las señales decodificadas al software de la PC.
- ✓ Se diseñó e implementó un TNC con las dos modulaciones más usadas por los satélites de investigación en la actualidad, con capacidad de hasta cuatro modulaciones a la vez. El TNC multimodulación propuesto es reconfigurable automáticamente por medio de un software de seguimiento de satélites implementado en un computador, que también muestra en una interfaz gráfica los datos decodificados por el TNC. De esta manera, se mejora la eficiencia en tiempo de comunicación de la estación terrena y se hace posible una forma de automatizar el proceso de captura de datos de satélites con fines de investigación.
- ✓ Los materiales usados en el desarrollo de las tarjetas electrónicas han sido de disponibilidad local en su totalidad. Esto mejora la factibilidad de replicar este proyecto y de realizar proyectos similares con un tiempo en adquisición de componentes electrónicos muy pequeño.
- ✓ El consumo de la memoria de programa de los microcontroladores del TNC propuesto es de la mitad o menos para los dos algoritmos de decodificación implementados. Esto quiere decir que se pueden implementar los dos algoritmos para los dos módems restantes. Cabe mencionar que siempre se pueden optimizar los códigos para reducir el consumo de memoria del microcontrolador.
- ✓ El TNC propuesto tiene características adicionales a las de un TNC convencional, ya sea de hardware o software. Esta característica adicional está referida a la capacidad de control de los rotores y la radio de la estación terrena, además del mismo TNC, mediante el software del TNC ejecutado en el computador al que se conecta. Por ello, el sistema completo es comparable a un software de gestión de la estación terrena.

- ✓ El TNC multimodulación propuesto está diseñado para usarse en proyectos modernos como constelaciones de pequeños satélites y redes de estaciones terrenas. Al tener la capacidad de hasta cuatro tipos de modulación permite la compatibilidad con satélites desarrollados por varias instituciones que trabajarán de manera conjunta. Asimismo, al permitir la automatización del proceso de recepción de información de estos satélites hace posible el trabajo de una estación terrena con diferentes satélites de manera coordinada.
- ✓ Los circuitos módems funcionan bien para señales de entrada ideales y reales siempre y cuando las señales reales no tengan un elevado nivel de ruido y tengan las frecuencias convenidas: 1200 Hz para el módem ASK y 1200 – 2200 Hz para el módem FSK. Estas frecuencias se calibran modificando las frecuencias en el archivo de modos de los satélites del software que interactúa con el TNC.
- ✓ El algoritmo de decodificación Morse tiene un porcentaje de error considerable, lo que se puede mejorar calibrando el módem ASK, mejorando el algoritmo de decodificación Morse, o usando un conversor ADC en lugar de la interrupción de cambio de entrada digital usado.
- ✓ Se pueden acoplar nuevas tecnologías al proyecto siempre que los otros elementos soporten las condiciones de operación como rapidez de procesamiento. Por ejemplo, los circuitos módem pueden ser cambiados por otros cualesquiera, fabricados con dispositivos superficiales o no superficiales. Esto incrementa el tiempo de vigencia del TNC propuesto, porque nuevas tecnologías de modulación se pueden agregar a medida que se experimenten con tipos de modulación más convenientes.
- ✓ Se ha encontrado que el mecanismo de selección automática de modulación del TNC propuesto puede probarse en un escenario totalmente simulado en un computador portátil. Para futuros trabajos de optimización de funcionamiento de los circuitos módems en campo se deben considerar las variaciones en amplitud y relación señal a ruido del audio obtenido con la radio, ya que éstos parámetros varían a medida que el satélite se acerca o aleja de la estación terrena.

ANEXO A
CODIFICACIÓN MORSE

ANEXO A

CODIFICACIÓN MORSE

Es una forma de codificación usada desde 1844, que aún se utiliza para algunos servicios de radiocomunicaciones incluidos los de aficionados y de aficionados por satélite y, en menor medida, para los servicios móviles y fijos [10]. La Unión Internacional de Telecomunicaciones (UIT, o ITU por sus siglas en inglés) asimismo recomienda que el código debe actualizarse de vez en cuando para satisfacer las necesidades de los servicios de radiocomunicaciones.

Señales de código Morse

Los caracteres escritos que pueden utilizarse y que corresponden a señales de código Morse son los siguientes:

Letras:

	a	.-	i	..	r	.-.
	b	-...	j	.----	s	...
	c	-.-.	k	-.-	t	-
	d	-..	l	.-..	u	..-
	e	.	m	--	v	...-
acentuada	e	..-..	n	-.	w	.-.-
	f	..-.	o	----	x	-...-
	g	--.	p	.-.-.	y	-.-.-
	h	q	--.-	z	--..

Figura A.1: Letras del código Morse

Cifras:

1	.-----	6	-.....
2	..----	7	--....
3	...--	8	----..
4-	9	----.
5	0	-----

Figura A.2: Cifras del código Morse

Signos de puntuación y signos varios:

Punto	[.]	.-.-.-
Coma	[,]	---.-.-
Dos puntos o signo de división.....	[:]	---...
Interrogación final o petición de repetición de una transmisión no entendida	[?]	..-.-..
Apóstrofo	[']	.-.-.-.
Guión o signo de sustracción	[-]	-....-
Barra de fracción o signo de división	[/]	-.-.-.
Paréntesis izquierdo (abrir)	[(]	-.-.-.
Paréntesis derecho (cerrar)	[)]	-.-.-.-
Comillas (antes y después de las palabras)	[" "]	.-.-.-.
Doble raya	[=]	-....-
Enterado-.
Error (ocho puntos)
Cruz o signo de adición	[+]	.-.-.-.
Invitación a transmitir		-.-
Espera-...
Fin de trabajo-.-
Señal de comienzo (comienzo de toda transmisión)		-.-.-
Signo de multiplicación	[×]	-...-
Arroba comercial ¹	[@]	.-.-.-.

Figura A.3: Signos del código Morse

Separación y longitud de las señales

La longitud de una raya equivale a la de tres puntos.

La separación entre las señales de una misma letra equivale a un punto.

La separación entre dos letras equivale a tres puntos.

La separación entre dos palabras equivale a siete puntos.

Transmisión de signos para los que no hay señal correspondiente en el código Morse

Los signos que no tengan señal correspondiente en el código Morse, pero que sean aceptables para la redacción de los telegramas, se transmitirán como sigue:

- Para el signo de multiplicación, se transmitirá la señal correspondiente a la letra X.
- Para transmitir el signo %, se transmitirán sucesivamente la cifra 0, la barra de fracción y la cifra 0 (esto es, 0/0).
- Un número entero, un número fraccionario o una fracción, seguidos del signo %, se transmitirán uniendo con un solo guión el número entero, el número fraccionario o la fracción y el signo %. Ejemplos: para 2% se transmitirá 2-0/0, para 4½% se transmitirá 4-1/2-0/0.

ANEXO B
CODIFICACIÓN AX.25

ANEXO B

CODIFICACIÓN AX.25

AX.25 es un protocolo de acceso de enlace para radio paquetes de tipo aficionado, estandarizado en su versión 2.2 [12] en Noviembre de 1997 por la Corporación de Radio Paquetes de tipo Aficionado de Tucson. Aparece por primera vez en el libro “AX.25” de John Ackerman, AG9V, en 1992.

Modelo de comunicación AX.25

La organización ISO desarrolló un modelo de comunicación para Interconexión de Sistemas Abiertos (OSI) con la finalidad de tener un modelo de referencia para crear sistemas de comunicación y conectarlos entre sí. El modelo de referencia OSI está constituido por 7 capas o niveles, cada uno destinado a cumplir ciertas funcionalidades en orden de abstracción: Aplicación, Presentación, Sesión, Transporte, Red, Enlace de Datos y Física.

En el modelo de comunicación AX.25, los 2 niveles inferiores del modelo de referencia OSI (capa física y capa de enlace de datos) se subdividen en máquinas de estados finitos que realizan funciones equivalentes, con la finalidad de modelar el enlace con el puerto de una radio.

Layer	Function(s)	
Data Link (2)	Segmenter	Management
	Data Link	Data Link
	Link Multiplexer	
Physical (1)	Physical	
	Silicon/Radio	

Figura B.1: Capas del modelo AX.25

La Máquina de Estados Segmentadora acepta entradas del nivel superior adyacente por medio de un Punto de Acceso al Servicio de Enlace de Datos (DLSAP). Si la unidad de datos a ser enviada excede los límites de tamaño de una trama AX.25, entonces el Segmentador fragmenta las unidades en segmentos más pequeños para hacer posible la transmisión.

La Máquina de Estados de Enlace de Datos es el corazón del protocolo AX.25, pues provee toda la lógica para establecer conexiones entre 2 estaciones, así como intercambiar información sin conexión (tramas UI) o con conexión (tramas I con procesos de recuperación).

La Máquina de Estados de Administración de Enlace de Datos provee la lógica necesaria para realizar la negociación de los parámetros de operación entre 2 estaciones. Al igual que las 2 anteriores subcapas, se debe tener una máquina de estados por cada enlace de datos del puerto de radio.

La Máquina de Estados Multiplexadora de Enlaces hace posible que varios enlaces de datos compartan el mismo canal físico (radio). Una de estas máquinas debe existir por canal físico.

La Máquina de Estados Física controla al transmisor y receptor de la radio. También existe una de éstas máquinas por canal físico. En el estándar se han definido 2 tipos de Máquina de Estados Físicas: Half-dúplex y Full-dúplex.

Estructura de la trama AX.25

En transmisión de radio paquetes de nivel de enlace, la información es enviada en pequeños bloques de datos denominados tramas, que pueden ser de 3 tipos: de información (tramas I), de supervisión (tramas S) y no numeradas (tramas U). A continuación se muestran las formas de las tramas tipo S y U, así como la forma de las tramas tipo I.

Flag	Address	Control	Info	FCS	Flag	
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110	
Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

Figura B.2: Estructura de los 2 tipos de trama AX.25

Procedimientos AX.25

Los procedimientos que se llevan a cabo para iniciar, usar y desconectar un enlace balanceado entre 2 estaciones TNC se pueden agrupar en: operación del campo dirección, procesos del bit P/F, procesos para inicio y desconexión de enlace, procesos para transferencia de información, procesos de restablecimiento y ensamblado/desensamblado.

Todas las tramas transmitidas tienen un campo de dirección que incluye la dirección del dispositivo fuente y del dispositivo destino (la dirección de destino va primero). Esto permite que muchos enlaces compartan el mismo canal RF.

AX.25 implementa la información de comando/respuesta en el campo de dirección en forma de 2 bits que además permiten mantener compatibilidad con versiones anteriores del protocolo. Debido a que toda trama AX.25 (en la última versión) es o bien un comando o bien una respuesta, siempre tiene uno de los bits SSID en uno y el otro en cero.

<u>Frame TypeDest.</u>	<u>SSID C-Bit</u>	<u>Source SSID C-Bit</u>
Previous versions	0	0
Command (V.2.X)	1	0
Response (V.2.X)	0	1
Previous versions	1	1

Figura B.3: Estructura del campo SSID

La trama retornada como respuesta por un TNC depende del comando previo recibido. Por ejemplo, la trama de respuesta a un comando SABM(E) o DISC con el bit "P" (Poll) en 1 es una respuesta UA o DM con el bit "F" (Final) en 1. Cuando no se le usa, el bit P/F es puesto a 0.

La trama de respuesta a una trama I con el bit "P" en 1 (durante el estado de transferencia de información) es una respuesta RR, RNR o REJ con el bit "F" en 1. La trama de respuesta para un comando de supervisión con el bit "P" en 1 (durante una transferencia de información) es una respuesta RR, RNR o REJ con el bit "F" en 1. La trama de respuesta para un comando S o I con el bit "P" en 1 recibido en el estado de desconexión, es una trama DM con el bit "F" en 1.

Para conectarse con un TNC distante, el TNC remitente envía una trama de comando SABM para el TNC distante e inicia el conteo de su temporizador T1. Si el TNC distante existe y acepta el pedido de establecimiento de conexión, envía una trama de respuesta UA hacia el TNC remitente, el cual cancela su temporizador y pone la cuenta en cero. Si el TNC distante no responde antes del tiempo de agotamiento de T1, el TNC remitente envía otra vez la trama SABM e inicia de nuevo su temporizador (se realizan hasta N2 intentos de darse el caso). Si el TNC distante recibe un comando SABM y no puede ingresar al estado indicado, envía una trama DM, con la cual el primer TNC cancela su temporizador y no envía información.

Durante el estado de transferencia de información, alguno de los TNC puede solicitar la desconexión del enlace transmitiendo una trama de comando DISC. El TNC entonces cancela su temporizador T1 e ingresa al estado de desconexión. Si no se recibe respuestas UA o DM antes del tiempo de agotamiento de T1, se envía otra vez la trama DISC y se reinicia el temporizador, hasta N2 intentos de darse el caso, luego de lo cual el TNC entra de todas maneras al estado de desconexión.

En estado de desconexión, un TNC monitorea los comandos recibidos, reacciona al comando SABM como se ha descrito, y transmite una trama DM en respuesta a un comando DISC. En el estado de desconexión, un TNC puede iniciar un enlace respondiendo al comando SABM para establecer una conexión.

Un tipo de operación adicional existe en radioaficionados que no es posible usando conexiones de nivel 2. Este es el modo de operación sin conexión (connectionless) y las tramas empleadas aquí son las de tipo de información no numerada (UI). Sin embargo, este tipo de operación no puede ser implementado por las conexiones del nivel de enlace del protocolo AX.25 actual.

Una vez que la conexión entre 2 TNCs ha sido establecida, cualquiera de los TNCs puede aceptar tramas I, S y U. Cada vez que un TNC tiene una trama I para transmitir, envía una trama I con el N(S) del campo de control igual a su variable de estado de envío actual V(S). Después de enviar la trama I, la variable de estado de envío se incrementa en 1. Si el temporizador T1 no está activo se inicia, y si se está ejecutando entonces es reiniciado. El TNC no transmite ninguna trama si su variable de estado de envío es igual al último N(R) recibido del otro lado del enlace más k. La recepción de tramas I que contienen campos de información de longitud cero es reportado al siguiente nivel. Si el TNC recibe una trama I válida (con un FCS correcto y cuyo número de secuencia enviada es igual a la variable de estado de recepción del receptor) y no está en la condición de ocupado, acepta la trama I, incrementa su variable de estado de recepción y actúa en una de las maneras, si es que tiene una trama I para enviar o no. Si el TNC está en la condición de ocupado, ignora las tramas recibidas sin reportar condición alguna que reportar la condición de ocupado.

Cuando un TNC recibe una trama con un FCS o una dirección incorrectos, el TNC descarta la trama. Cuando una trama S o I es recibida correctamente, aún en una condición de ocupado, el N(R) de la trama recibida se chequea para ver si

incluye una confirmación de recepción de las tramas I anteriormente enviadas. El temporizador 1 es cancelado si la trama recibida confirma la recepción de las tramas anteriormente enviadas.

Si el temporizador del TNC remitente expira mientras espera la confirmación de recepción de una trama I transmitida, el TNC emite reinicia su temporizador y transmite una trama de comando de supervisión apropiada (RR o RNR) con el bit P en 1. Si el temporizador expira antes de recibir una trama de respuesta de supervisión con el bit F en 1, el TNC retransmite la trama de comando de supervisión apropiada (RR o RNR) con el bit P en 1. Si se dan N2 intentos sin éxito, el TNC remitente inicia un procedimiento de restablecimiento de enlace, donde se inicializan ambas direcciones del flujo de datos después de que haya ocurrido un error no recuperable (éste procedimiento solo se usa en el estado de transferencia de información de un enlace AX.25).

El proceso de fragmentación/desfragmentación se activa solo si ambas estaciones usan la versión 2.2 o posterior del protocolo AX.25; esto permite la transmisión de paquetes de longitud mayor de N1 de manera sencilla y clara, y agrega una cabecera adicional de menos de 1% del estándar N1 de 256 bytes. Además agrega la capacidad de de enviar entidades de datos grandes de nivel 3 como datagramas IP sobre entidades simples AX.25.

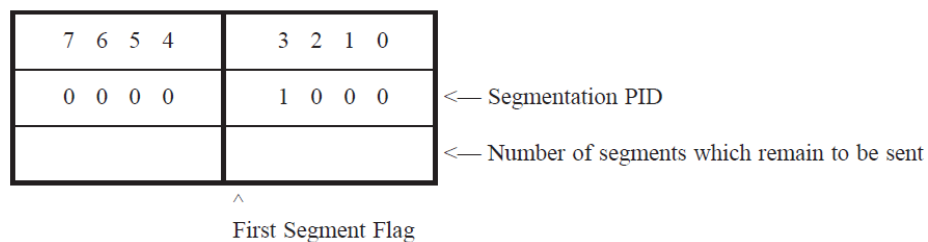


Figura B.4: Estructura del campo PID

La figura muestra un ejemplo de fragmentación de una trama AX.25 de longitud demasiado grande. Se fragmenta la trama en tramas más pequeñas de longitud constante, a las cuales se agrega una cabecera para establecer el orden de los tramos pequeños que serán desfragmentados (vuelto a unir) en el otro lado.

ANEXO C
DIAGRAMAS ESQUEMÁTICOS

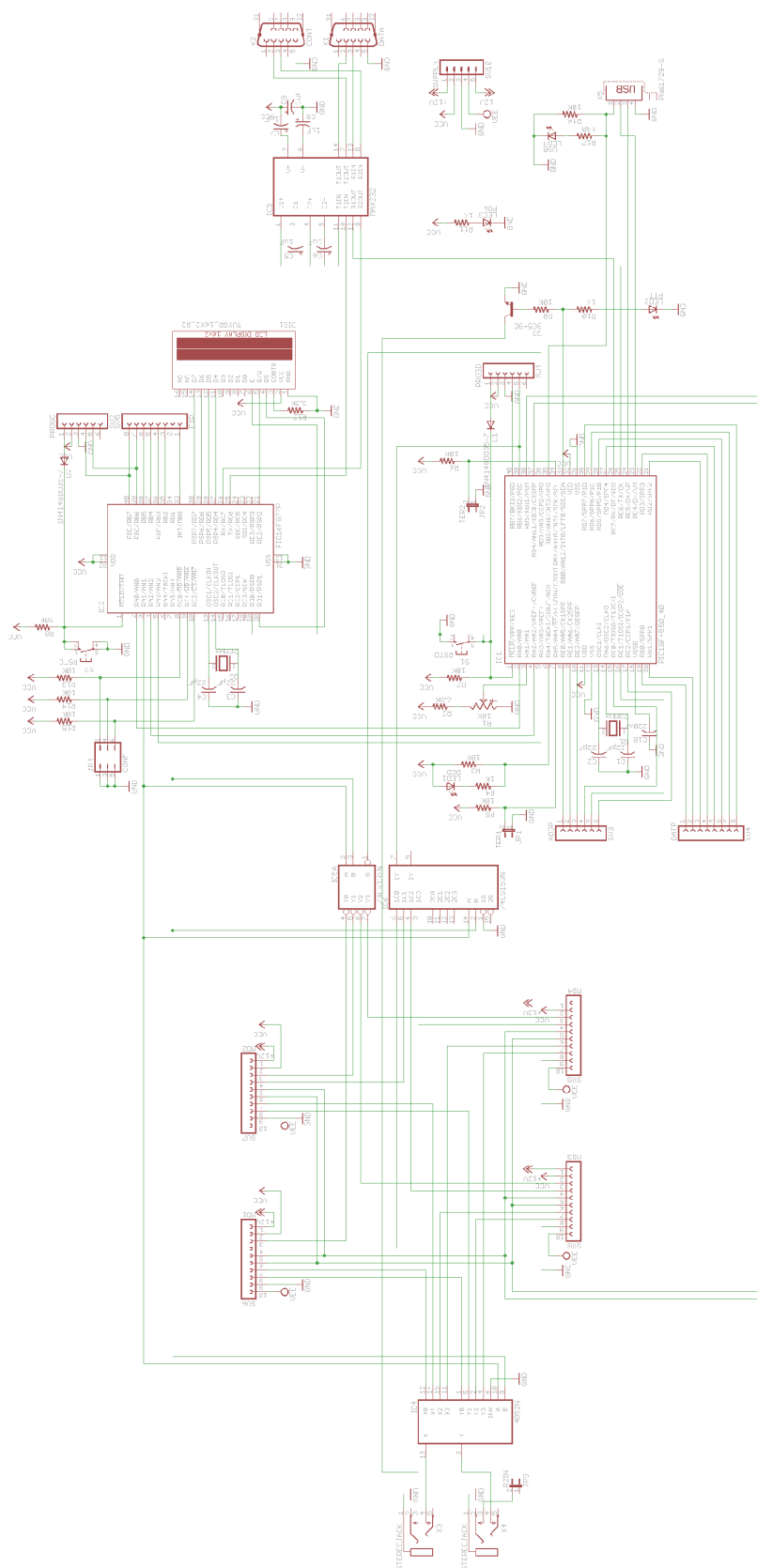


Figura C.1: Diagrama esquemático de la tarjeta principal

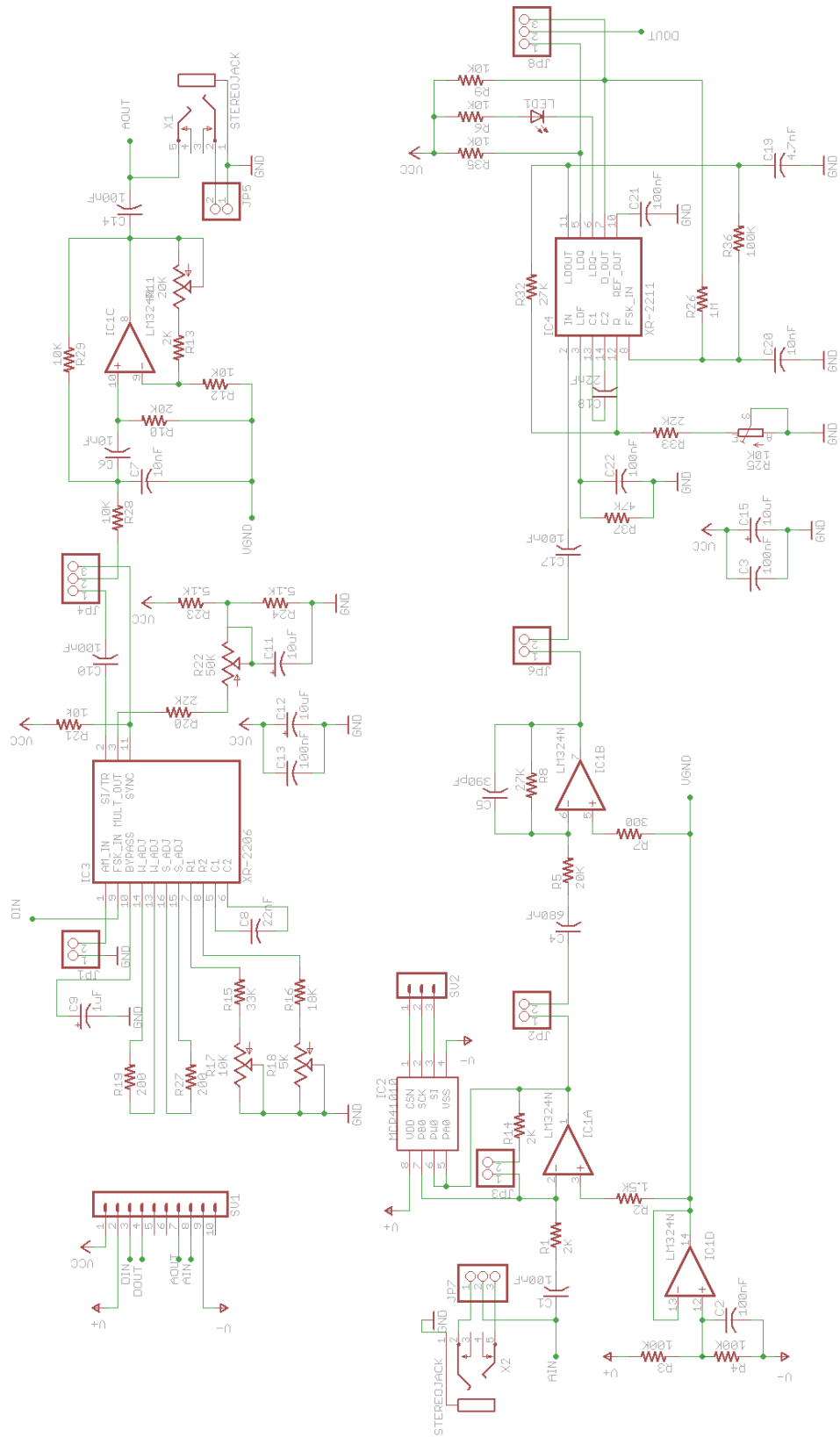


Figura C.2: Diagrama esquemático de las tarjetas módem

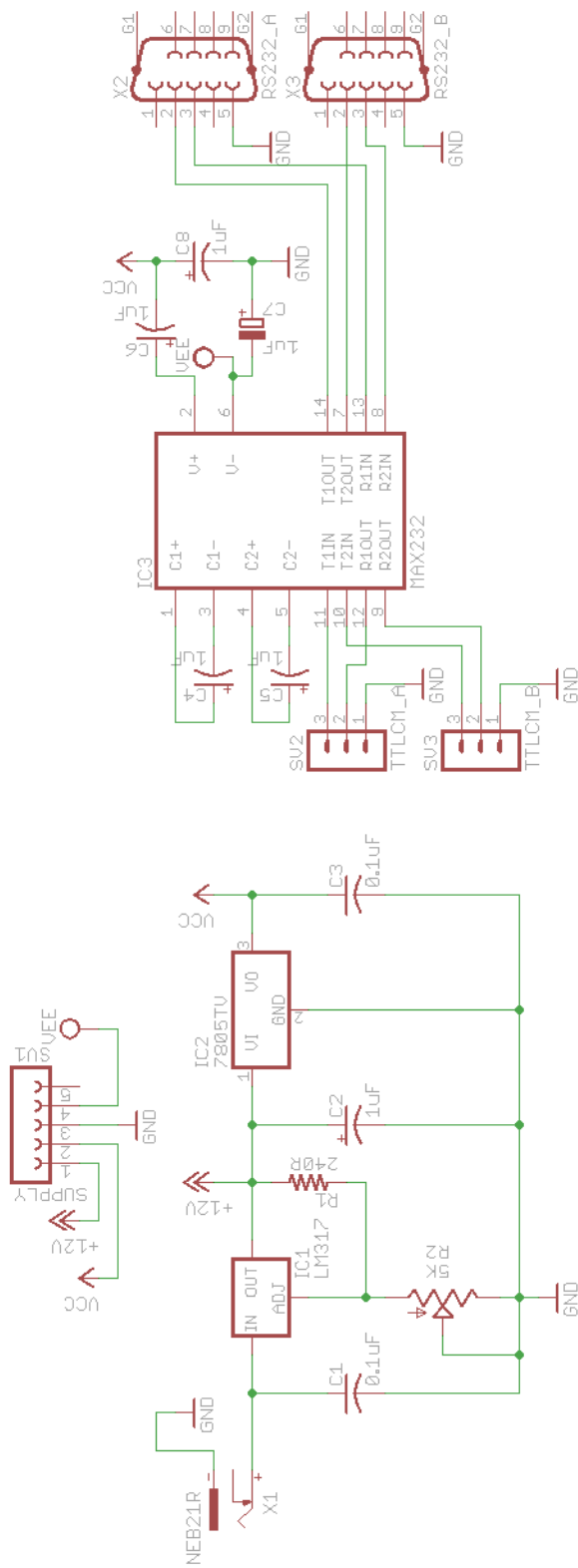
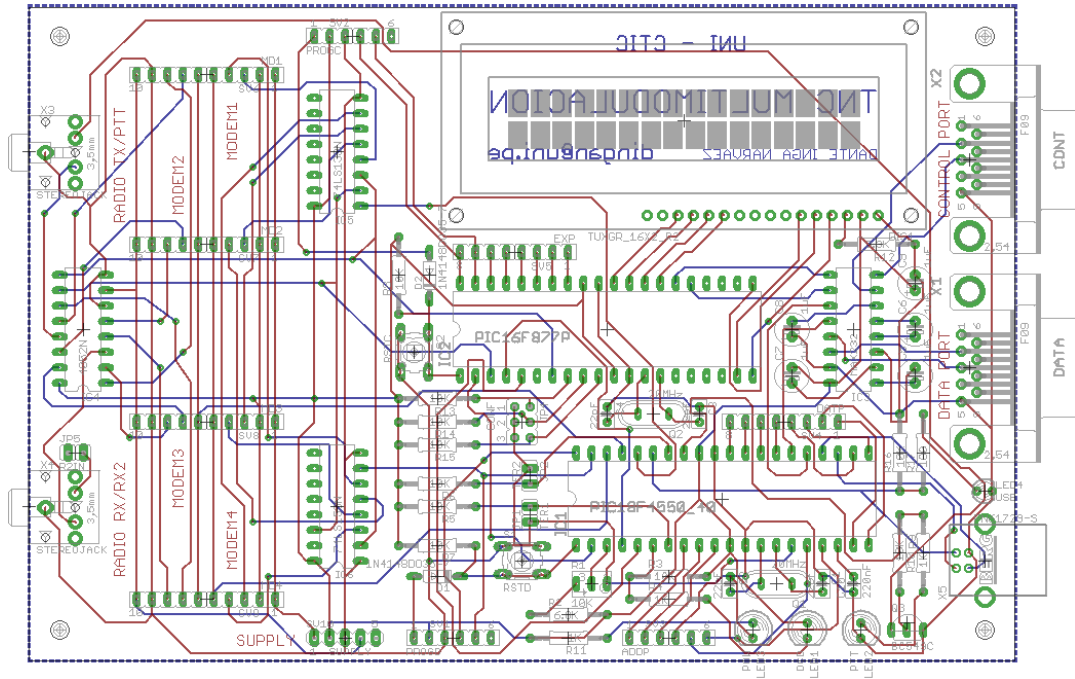


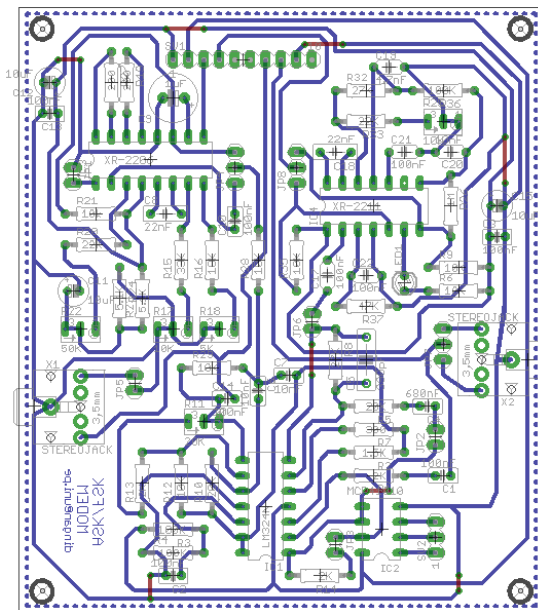
Figura C.3: Diagrama esquemático de la tarjeta de alimentación

ANEXO D
PLACAS IMPRESAS

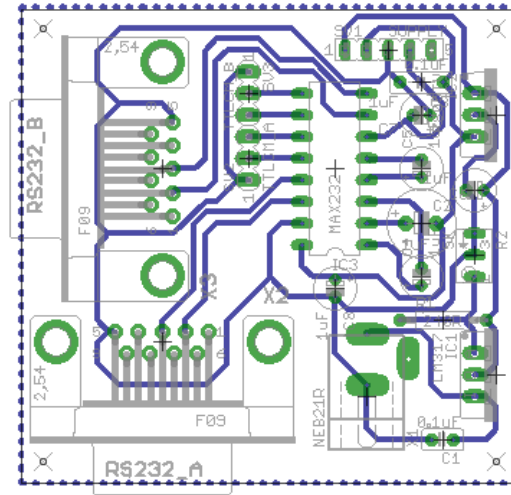
D.1 Diseño de placa impresa de la tarjeta principal



D.2 Diseño de placa impresa de las tarjetas módem



D.3 Diseño de placa impresa de la tarjeta de alimentación



ANEXO E
LISTA DE COMPONENTES

E.1 Lista de componentes de la tarjeta principal

Part	Value	Device	Package	Library	Sheet
C1	22pF	C-US025-025X050	C025-025X050	rcl	1
C2	22pF	C-US025-025X050	C025-025X050	rcl	1
C3	22pF	C-US025-025X050	C025-025X050	rcl	1
C4	22pF	C-US025-025X050	C025-025X050	rcl	1
C5	1uF	CPOL-USE2.5-6	E2,5-6	rcl	1
C6	1uF	CPOL-USE2.5-6	E2,5-6	rcl	1
C7	1uF	CPOL-USE2.5-6	E2,5-6	rcl	1
C8	1uF	CPOL-USE2.5-6	E2,5-6	rcl	1
C9	1uF	CPOL-USE2.5-6	E2,5-6	rcl	1
C10	220nF	C-US025-025X050	C025-025X050	rcl	1
D1	1N4148DO35-7	1N4148DO35-7	DO35-7	diode	1
D2	1N4148DO35-7	1N4148DO35-7	DO35-7	diode	1
DIS1	TUXGR_16X2_R2	TUXGR_16X2_R2	TUXGR_16X2_R2	display-lcd	1
IC1	PIC18F4550_40	PIC18F4550_40	DIL40	microchip	1
IC2	PIC16F877P	PIC16F877P	DIL40	microchip	1
IC3	MAX232	MAX232	DIL16	maxim	1
IC4	4052N	4052N	DIL16	40xx	1
IC5	74LS139N	74LS139N	DIL16	74xx-us	1
IC6	74LS153N	74LS153N	DIL16	74xx-us	1
JP1	TER1	JP1E	JP1	jumper	1
JP2	TER2	JP1E	JP1	jumper	1
JP4	CONF	JP3Q	JP3Q	jumper	1
JP5	R2IN	JP1E	JP1	jumper	1
LED1	DCD	LED5MM	LED5MM	led	1
LED2	PTT	LED5MM	LED5MM	led	1
LED3	POW	LED5MM	LED5MM	led	1
LED4	USB	LED3MM	LED3MM	led	1
Q1	20MHz	XTAL/S	QS	special	1
Q2	20MHz	XTAL/S	QS	special	1
Q3	BC549C	BC549C	TO92-EBC	transistor-npn	1
R1	10K	TRIM_US-B64Y	B64Y	pot	1
R2	6.8K	R-US_0207/12	0207/12	resistor	1
R3	10K	R-US_0207/12	0207/12	resistor	1
R4	1K	R-US_0207/12	0207/12	resistor	1
R5	10K	R-US_0207/12	0207/12	resistor	1
R6	10K	R-US_0207/12	0207/12	resistor	1
R7	10K	R-US_0207/12	0207/12	resistor	1
R8	10K	R-US_0207/12	0207/12	resistor	1
R9	10K	R-US_0207/12	0207/12	resistor	1
R10	1K	R-US_0207/12	0207/12	resistor	1
R11	1K	R-US_0207/12	0207/12	resistor	1
R12	3.3K	R-US_0207/12	0207/12	resistor	1
R13	10K	R-US_0207/12	0207/12	resistor	1
R14	10K	R-US_0207/12	0207/12	resistor	1
R15	10K	R-US_0207/12	0207/12	resistor	1
R16	10K	R-US_0207/12	0207/12	resistor	1
R17	180	R-US_0207/12	0207/12	resistor	1
S1	RSTD	10-XX	B3F-10XX	switch-omron	1
S2	RSTC	10-XX	B3F-10XX	switch-omron	1

SV1	PROGD	FE06-1	FE06	con-lsta	1
SV2	PROGC	FE06-1	FE06	con-lsta	1
SV3	ADDP	FE06-1	FE06	con-lsta	1
SV4	DATP	FE08-1	FE08	con-lsta	1
SV5	EXP	FE08-1	FE08	con-lsta	1
SV6	MD1	FE10-1	FE10	con-lsta	1
SV7	MD2	FE10-1	FE10	con-lsta	1
SV8	MD3	FE10-1	FE10	con-lsta	1
SV9	MD4	FE10-1	FE10	con-lsta	1
SV10	SUPPLY	MA05-1	MA05-1	con-lstb	1
X1	DATA	F09HP	F09HP	con-subd	1
X2	CONT	F09HP	F09HP	con-subd	1
X3	STEREOJACK	STEREOJACK	STX3100	adafruit	1
X4	STEREOJACK	STEREOJACK	STX3100	adafruit	1
X5	PN61729-S	PN61729-S	PN61729-S	con-berg	1

E.2 Lista de componentes del modem ASK

Part	Value	Package	Library	Position (inch)	Orientation
C1	100nF	C025-025X050	rcl	(2.75 0.9)	R180
C2	100nF	C025-025X050	rcl	(1 0.25)	R180
C3	100nF	C025-025X050	rcl	(3.1 2.45)	R0
C4	680nF	C025-025X050	rcl	(2.65 1.35)	R180
C5	390pF	C075-032X103	rcl	(2.25 1.65)	R90
C6	10nF	C025-025X050	rcl	(1.55 1.45)	R270
C7	10nF	C025-025X050	rcl	(1.75 1.55)	R0
C8	22nF	C050-025X075	rcl	(0.95 2.6)	R0
C9	1uF	E3,5-8	rcl	(1 3.35)	R180
C10	100nF	C025-025X050	rcl	(1.4 2.55)	R90
C11	10uF	E2-5	rcl	(0.35 2.1)	R0
C12	10uF	E2-5	rcl	(0.2 3.45)	R180
C13	100nF	C025-025X050	rcl	(0.2 3.25)	R180
C14	100nF	C025-025X050	rcl	(1.35 1.4)	R0
C15	10uF	E2-5	rcl	(3.1 2.65)	R0
C17	100nF	C050-024X044	rcl	(2 2.2)	R90
C18	22nF	C050-025X075	rcl	(2.05 3)	R180
C19	4.7nF	C050-024X044	rcl	(2.4 3.55)	R0
C20	10nF	C050-024X044	rcl	(2.8 3)	R180
C21	100nF	C050-024X044	rcl	(2.5 3)	R0
C22	100nF	C050-024X044	rcl	(2.25 2.2)	R0
IC1	LM324N	DIL14	linear	(1.6 0.7)	R90
IC2	MCP41010	DIL8	mcp41xxx	(2.35 0.55)	R90
IC3	XR-2206	DIL16	exar	(0.85 2.95)	R0
IC4	XR-2211	DIL14	exar	(2.3 2.65)	R0
JP1		1X02	pinhead	(0.35 2.85)	R270
JP2		1X02	pinhead	(2.7 1.15)	R90
JP3		1X02	pinhead	(2.05 0.45)	R270
JP4		1X03	pinhead	(1.4 2.9)	R90
JP5		1X02	pinhead	(0.75 1.5)	R90
JP6		1X02	pinhead	(1.9 1.9)	R90
JP7		1X03	pinhead	(2.75 1.75)	R90
JP8		1X03	pinhead	(1.8 2.9)	R90
LED1		LED3MM	led	(2.5 2.15)	R90

R1	2K	0207/10	rcl	(2.3 0.9)	R180
R2	1.5K	0207/10	rcl	(2.3 1.05)	R180
R3	100K	0207/10	rcl	(1 0.55)	R180
R4	100K	0207/10	rcl	(1 0.4)	R0
R5	20K	0207/10	rcl	(2.3 1.35)	R180
R6	10K	0207/10	rcl	(2.85 2.1)	R0
R7	300	0207/10	rcl	(2.3 1.2)	R180
R8	27K	0207/10	rcl	(2.05 1.65)	R270
R9	10K	0207/10	rcl	(2.85 2.25)	R0
R10	20K	0207/10	rcl	(1.25 0.85)	R90
R11	20K	B64Y	pot	(1.2 1.25)	R0
R12	10K	0207/10	rcl	(1.05 0.85)	R90
R13	2K	0207/10	rcl	(0.8 0.85)	R90
R14	2K	0207/10	rcl	(1.95 0.25)	R180
R15	33K	0207/10	rcl	(1.05 2.3)	R90
R16	18K	0207/10	rcl	(1.25 2.3)	R90
R17	10K	B64Y	pot	(1 1.85)	R0
R18	5K	B64Y	pot	(1.3 1.85)	R0
R19	200	0207/10	rcl	(0.7 3.45)	R270
R20	22K	0207/10	rcl	(0.5 2.4)	R0
R21	10k	0207/10	rcl	(0.5 2.6)	R0
R22	50K	B64Y	pot	(0.4 1.85)	R0
R23	5.1K	0207/10	rcl	(0.8 2.05)	R90
R24	5.1K	0207/10	rcl	(0.65 2.05)	R270
R25	10K	B64Y	pot	(2.75 3.2)	R0
R26	1M	0207/10	rcl	(2.8 2.65)	R270
R27	200	0207/10	rcl	(0.6 3.45)	R270
R28	10K	0207/10	rcl	(1.55 2.3)	R90
R29	10K	0207/10	rcl	(1.25 1.6)	R0
R32	27K	0207/10	rcl	(2.25 3.4)	R0
R33	22K	0207/10	rcl	(2.25 3.2)	R180
R35	10K	0207/10	rcl	(1.8 2.3)	R90
R36	100K	0207/10	rcl	(2.75 3.4)	R180
R37	47K	0207/10	rcl	(2.25 2)	R180
SV1		MA10-1	con-lstb	(1.45 3.6)	R0
SV2		MA03-1	con-lstb	(2.7 0.5)	R90
X1	STEREOJACK	STX3100	adafruit	(0.1 1.35)	R0
X2	STEREOJACK	STX3100	adafruit	(3.3 1.65)	R180

E.3 Lista de componentes del modem FSK

Part	Value	Package	Library	Position (inch)	Orientation
C1	100nF	C025-025X050	rcl	(2.75 0.9)	R180
C2	100nF	C025-025X050	rcl	(1 0.25)	R180
C3	100nF	C025-025X050	rcl	(3.1 2.45)	R0
C4	680nF	C025-025X050	rcl	(2.65 1.35)	R180
C5	390pF	C075-032X103	rcl	(2.25 1.65)	R90
C6	10nF	C025-025X050	rcl	(1.55 1.45)	R270
C7	10nF	C025-025X050	rcl	(1.75 1.55)	R0
C8	22nF	C050-025X075	rcl	(0.95 2.6)	R0
C9	1uF	E3,5-8	rcl	(1 3.35)	R180
C10	100nF	C025-025X050	rcl	(1.4 2.55)	R90
C11	10uF	E2-5	rcl	(0.35 2.1)	R0
C12	10uF	E2-5	rcl	(0.2 3.45)	R180

C13	100nF	C025-025X050	rcl	(0.2 3.25)	R180
C14	100nF	C025-025X050	rcl	(1.35 1.4)	R0
C15	10uF	E2-5	rcl	(3.1 2.65)	R0
C17	100nF	C050-024X044	rcl	(2 2.2)	R90
C18	22nF	C050-025X075	rcl	(2.05 3)	R180
C19	4.7nF	C050-024X044	rcl	(2.4 3.55)	R0
C20	10nF	C050-024X044	rcl	(2.8 3)	R180
C21	100nF	C050-024X044	rcl	(2.5 3)	R0
C22	100nF	C050-024X044	rcl	(2.25 2.2)	R0
IC1	LM324N	DIL14	linear	(1.6 0.7)	R90
IC2	MCP41010	DIL8	mcp41xxx	(2.35 0.55)	R90
IC3	XR-2206	DIL16	exar	(0.85 2.95)	R0
IC4	XR-2211	DIL14	exar	(2.3 2.65)	R0
JP1		1X02	pinhead	(0.35 2.85)	R270
JP2		1X02	pinhead	(2.7 1.15)	R90
JP3		1X02	pinhead	(2.05 0.45)	R270
JP4		1X03	pinhead	(1.4 2.9)	R90
JP5		1X02	pinhead	(0.75 1.5)	R90
JP6		1X02	pinhead	(1.9 1.9)	R90
JP7		1X03	pinhead	(2.75 1.75)	R90
JP8		1X03	pinhead	(1.8 2.9)	R90
LED1		LED3MM	led	(2.5 2.15)	R90
R1	2K	0207/10	rcl	(2.3 0.9)	R180
R2	1.5K	0207/10	rcl	(2.3 1.05)	R180
R3	100K	0207/10	rcl	(1 0.55)	R180
R4	100K	0207/10	rcl	(1 0.4)	R0
R5	20K	0207/10	rcl	(2.3 1.35)	R180
R6	10K	0207/10	rcl	(2.85 2.1)	R0
R7	300	0207/10	rcl	(2.3 1.2)	R180
R8	27K	0207/10	rcl	(2.05 1.65)	R270
R9	10K	0207/10	rcl	(2.85 2.25)	R0
R10	20K	0207/10	rcl	(1.25 0.85)	R90
R11	20K	B64Y	pot	(1.2 1.25)	R0
R12	10K	0207/10	rcl	(1.05 0.85)	R90
R13	2K	0207/10	rcl	(0.8 0.85)	R90
R14	2K	0207/10	rcl	(1.95 0.25)	R180
R15	33K	0207/10	rcl	(1.05 2.3)	R90
R16	18K	0207/10	rcl	(1.25 2.3)	R90
R17	10K	B64Y	pot	(1 1.85)	R0
R18	5K	B64Y	pot	(1.3 1.85)	R0
R19	200	0207/10	rcl	(0.7 3.45)	R270
R20	22K	0207/10	rcl	(0.5 2.4)	R0
R21	10k	0207/10	rcl	(0.5 2.6)	R0
R22	50K	B64Y	pot	(0.4 1.85)	R0
R23	5.1K	0207/10	rcl	(0.8 2.05)	R90
R24	5.1K	0207/10	rcl	(0.65 2.05)	R270
R25	10K	B64Y	pot	(2.75 3.2)	R0
R26	1M	0207/10	rcl	(2.8 2.65)	R270
R27	200	0207/10	rcl	(0.6 3.45)	R270
R28	10K	0207/10	rcl	(1.55 2.3)	R90
R29	10K	0207/10	rcl	(1.25 1.6)	R0
R32	27K	0207/10	rcl	(2.25 3.4)	R0
R33	22K	0207/10	rcl	(2.25 3.2)	R180
R35	10K	0207/10	rcl	(1.8 2.3)	R90
R36	100K	0207/10	rcl	(2.75 3.4)	R180
R37	47K	0207/10	rcl	(2.25 2)	R180
SV1		MA10-1	con-1stb	(1.45 3.6)	R0

SV2	MA03-1	con-lstb (2.7 0.5)	R90	
X1	STEREOJACK	STX3100	adafruit (0.1 1.35)	R0
X2	STEREOJACK	STX3100	adafruit (3.3 1.65)	R180

E.4 Lista de componentes de la unidad de alimentación

Part	Value	Device	Package	Library	Sheet
C1	0.1uF	C-US025-025X050	C025-025X050	rcl	1
C2	1uF	CPOL-USE2-5	E2-5	rcl	1
C3	0.1uF	C-US050-025X075	C050-025X075	rcl	1
C4	1uF	CPOL-USE2-5	E2-5	rcl	1
C5	1uF	CPOL-USE2-5	E2-5	rcl	1
C6	1uF	CPOL-USE3.5-8	E3,5-8	rcl	1
C7	1uF	CPOL-EUE2-5	E2-5	rcl	1
C8	1uF	CPOL-USE2-5	E2-5	rcl	1
IC1	LM317	LM317TS	317TS	v-reg	1
IC2	7805TV	7805TV	TO220V	linear	1
IC3	MAX232	MAX232	DIL16	maxim	1
R1	240R	R-US_0207/10	0207/10	rcl	1
R2	5K	TRIM_US-B64Y	B64Y	pot	1
SV1	SUPPLY	FE05-1	FE05-1	con-lsta	1
SV2	TTLCM_A	MA03-1	MA03-1	con-lstb	1
SV3	TTLCM_B	MA03-1	MA03-1	con-lstb	1
X1	NEB21R	NEB21R	NEB21R	con-lumberg	1
X2	RS-232_A	F09HP	F09HP	con-subd	1
X3	RS-232_B	F09HP	F09HP	con-subd	1

ANEXO F
CÓDIGOS FUENTE

F.1 Código fuente del microcontrolador códec

*tncm_pu.h

```
//Basado en código de John Hansen (http://tnc-x.com/documentation.htm)

#include <18F4550.h>

#define delay(clock=20000000)
#define fuses HS,NOWDT,PUT,NOLVP, NOBROWNOUT, NOPROTECT, MCLR
// #fuses HS,WDT,PUT,NOLVP, NOBROWNOUT, PROTECT, NOMCLR

#define sel0 PIN_A2 // Selector 0 (PIN_A2)
#define sel1 PIN_A3 // Selector 1 (PIN_A3)
#define serp PIN_A0 // Serie-Paralelo (PIN_A0)

// #define sck 31744 //clock on fram (PIN_A0)
#define A1 31745 //TXDelay pot
// #define dataio 31746 //data on fram
// #define cs 31747 //chip select on fram
#define LED 31748 //DCD LED
#define term1 31749 //serial baudrate (with term2)
#define PTT 31752 //radio PTT (PIN_B0)
#define rxd 31767 //serial receive data (PIN_C7)
#define txd 31766 //serial transmit data (PIN_C6)
#define radtx 31755 //radio transmit data (PIN_B3)
#define m1 31756 //modem control line
#define m0 31757 //modem control line
#define radrx 31758 //radio receive data
#define term2 31759 //serial baudrate (with term1)
#define RS232(baud=19200,parity=N,xmit=txd,rcv=rxd)
// #use RS-232(baud=19200,parity=N,xmit=txd,rcv=rxd,ERRORS)
#define fast_io(A)
#define fast_io(B)

#define status = 0xFD8
#define portb = 0xF81
#define porta = 0xF80
#define intcon = 0xFF2
#define pir1 = 0xF9E
#define cmcon = 0xFB4
#define BSR = 0xFE0

#define T1CON = 0xFCD
#define T3CON = 0xFB1
```

***tncm_pu.c**

```

//Basado en código de John Hansen (http://tnc-x.com/documentation.htm)

#include "tncm_pu.h"

#define DOT_TIME 35 //50msec

unsigned int i,j,k;
long onesTime[6] = {0, 0, 0, 0, 0, 0};
long zeroTime[6] = {0, 0, 0, 0, 0, 0};
//int dataReceived[20];
int1 change=0, newp=0;
//float widp;
long signalTime=0, silenceTime=0;
//float signalTime=0, silenceTime=0;
//short flagMOD0=0, flagSIG=0, flagSIL=0;
short always0=1, always=1;
//short always2=1, always3=1;

int sram_1k[1024]; //512-bytes TX buffer and 512-bytes RX buffer
int minus1, minus2; //maintain a history list of bytes received.
int rcvbyte, bitcount; //the byte being recieved and the num of bits so far
short oldstate; //keeps track of the state of the rcv data pin
int ones; //keeps track of the number of consecutive ones received
short fiveones; //true if we have received 5 consecutive ones
int crclo,crchi; //for calculating CRC
short done; //true if packet reception is done
short isflag = false; //true if the current incoming byte is a flag or noise.
int count; //keeps track of whether it is the beginning or end of a packet on receive
short rmiddle = false; //keeps track of whether we are in the middle of a packet when
sending to serial port (RCV data)
short xmmiddle = false; //keeps track of whether we are in the middle of a packet when
receiving from the serial port (XMIT data)
int receivedpackets; //number of received packets that have not been sent to serial port
int COxmit; //number of packets to transmit = COXMIT
signed long rcvinadd; //address of byte from radio to memory addresses 0 to 1999
signed long rcvoutadd; //address of byte from memory to serial port addresses 0 to 1999
signed long begin; //beginning address of this packet.
signed long txinadd; //address of byte from serial to memory addresses 2048 to 8192
signed long txoutadd; //address of byte from memory to radio (transmit data) addresses
2048 to 8192
int xcrchi, xcrclo; //for transmit CRC calculation
int stuff; //counts the number of ones in a row to see if it is necessary to stuff a bit
short flag, crcflag; //true if we are transmitting flags (flag) or crcbytes (crcflag)

short xmitlatch = false; //enable transmit
byte PERSIST=30; //default 1 in 4 chance of transmitting
byte SLOTTIME=10; //wait 100 ms to check to transmit
int slotcount = 0;

//*****FUNCTION PROTOTYPES FOLLOW*****

```

```

int gettxdelay(); //determine txdelay
void sendbyteserial(); //send a byte over serial port (to terminal)
void rcvbyteserial(); //receive a byte from the serial port (from terminal)
void setParms(int ad);
void sendParms();
void crcbit(byte tbyte); //ongoing CRC calculation for transmitted data
void addcrc(int crbyte); //ongoing CRC calculation for received data
void sendbyte(byte inbyte); //send a byte via radio
void xmit(); //send accumulated data via radio
void findflag(); //look for the beginning of a received packet
void resetclock(); //synchronizes clock on zeros when receiving (Change on B
interrupt)
void rcvbits(); //receive bits from radio (Timer interrupt fires every 833 us)
void resetxmit(); //if something goes horribly wrong during xmit, this allows smooth
recovery...should never be called

//*****SERIAL PORT ROUTINES FOLLOW*****

char rand() { //pseudo random number generator
  char random;
  char sum = 0;
  random = GET_RTCC();
  if(random & 0x80) sum = 1;
  if(random & 0x20) sum ^= 1;
  if(random & 0x10) sum ^= 1;
  if(random & 0x04) sum ^= 1;
  random <<= 1;
  random |= sum;
  return random;
}

void fram_dump(){
long i=0;
int stuff;

  while (i <60){
    stuff = sram_1k[i];
    putc(stuff);
    i++;
  }
}

void sendbyteserial(){
int store;
  if (bit_test(pir1,4)){ //if the port is ready to receive data
    if (receivedpackets > 0){ //if there is data to send
      disable_interrupts(global); //don't allow interrupt of memory operation
      store = sram_1k[rcvoutadd++]; //get a byte from memory to route out serial port
      if (rcvoutadd == 512) rcvoutadd = 0; //memory rollover, if necessary
      enable_interrupts(global); //interrupt ok now.
      if (store == 0xC0){ //if we are on a packet boundary
        if (rmiddle){ //if middle = true we have the ending C0

```



```

        rmiddle = false;          //no longer in the middle
        receivedpackets--;        //decrement the number of received packets.
    }else rmiddle = true;         //if we weren't in the middle, we now are
    }
    putc(store);                  //send the data to the serial port
}
}
}

void rcvbyteserial(){            //process incoming characters from the serial port
int inbyt;
    if (bit_test(pir1,5)){        //if there is a byte in the buffer
        inbyt = getc();           //get the incoming byte
        if (inbyt == 0xC0) {      //do this if the byte is a C0 (begins or ends KISS
frame
            disable_interrupts(global); //don't allow interrupt of memory operation
            sram_1k[txinadd++] = inbyt; //write byte to memory
            if (txinadd == 1024) txinadd = 512; //rollover memory space if necessary
            enable_interrupts(global); //interrupt ok now.
            if (xmmiddle) c0xmit++;
            xmmiddle = !xmmiddle; //flip middle indicator
        }else{                    //if we are in the middle of a KISS frame or in between
KISS frames (shouldn't happen)
            if (xmmiddle) {        //confirm that we are in the middle
                disable_interrupts(global); //don't allow interrupt of memory operation
                sram_1k[txinadd++] = inbyt; //write byte to memory
                if (txinadd == 1024) txinadd = 512; //rollover memory space
                enable_interrupts(global); //interrupt ok now.
            }//end of if
        }//end of else
    }//end of if
}

void setParms(int ad){
    int bt;

    bt = sram_1k[txoutadd++]; //reads the command byte
    if (txoutadd == 1024) txoutadd = 512; //rollover if necessary
    if ((ad == 2) & (bt < 1)) bt = 1;
    WRITE_EEPROM(ad,bt); //write the value to EEPROM memory
    if (ad == 2) PERSIST = bt; //change the value currently used
    if (ad == 3) SLOTTIME = bt;
}

void sendParms(){
    int p,s;
    p = READ_EEPROM(2);
    s = READ_EEPROM(3);
    putc(p);
    putc(s);
}

//*****TRANSMIT ROUTINES FOLLOW*****
void resetxmit(){
int test, rep;
    output_low(PTT); //unkey PTT
    output_low(radtx); //leave transmit pin low
}

```

```

        output_low(M0);          //MX 614 to recv mode
        c0xmit = 0;              //reset xmit values
        txoutadd = txinadd = 512;
        xmiddle = false;
        for (rep = 0; rep < 10; rep++){
            delay_ms(10);
            restart_wdt();
        }
        while (bit_test(pir1,5)){ //wait until the stream of incoming data has ended
            test = getc();
            for (rep=0; rep < 25; rep++){
                delay_ms(10);
                restart_wdt();
            }
        }
    }

int gettxdelay(){                //read the TXdelay pot
int level;
    setup_comparator(A0_VR_A1_VR); //set up to use analog comparators on A0/A1
    for (level = 0; level < 16; level++){ //run the internal volt ref through each of its 16
settings.
        setup_vref(vref_low | level); //set the vref for the next step
        if (bit_test(cmcon,7)) break; //if the reference rises above the voltage on A1 we
have txdelay
    }
    setup_comparator(NC_NC_NC_NC); //pins all digital again
    return 5 * level;             //multiply value that was read by 5 to get txdelay (range 0
to abt 500 ms)
}

void crcbit(byte tbyte){        //accumulates the CRC for transmit
#asm
    //clr BSR
    BCF status,0
    RRCF xcrchi,F //RRF xcrchi,F //rotates the entire 16 bits to the right
    RRCF xcrclo,F //RRF xcrclo,F
#endasm
    if (((status & 0x01)^(tbyte)) ==0x01){
        xcrchi = xcrchi^0x84;
        xcrclo = xcrclo^0x08;
    }
}

void sendbyte (byte inbyte){    //send a byte over the radio link
int k, bt;
    restart_wdt();
    for (k=0;k<8;k++){
        bt = inbyte & 0x01; //strip off the rightmost bit
        while (get_rtcc() < 32) rcvbyteserial(); //wait until 832 us have elapsed
        delay_us(11); //timer resolution not good enough
        set_rtcc(0); //start timer back at 0
        if (!(crcflag) && (!flag)) (crcbit(bt)); //do crc, but only if this is not a flag or crc byte
        if (!bt) //if this is a zero
            stuff = 0; //reset stuff and send a 0
        portb = portb^0b00001000; //flip the radtx pin
    }
}

```

```

}else { //otherwise it is a one
stuff++; //increment 1's count
if (!(flag) && (stuff == 5)){ //stuff an extra 0, if 5 1's in a row
while(get_rtcc() < 32) rcvbyteserial(); //wait until 832 us have elapsed
delay_us(11); //timer resolution not good enough
set_rtcc(0); //start timer back at 0
stuff = 0; //reset stuff and send a 0
portb = portb^0b00001000; //flip the radtx pin
} //end of if
} //end of else
rotate_right(&inbyte,1); //get ready to process the next bit
} //end of for
}

void xmit(){ //transmits any waiting data
int c,outbyte,txdelay;
short DB = false;
short firstpack = true; //this is the first frame
while (c0xmit > 0){ //do this if there is any data to transmit
if(!input(sel0)) break;
stuff = 0; //keeps track of number of ones
crcflag = FALSE; //we aren't transmitting the CRC now
xcrcl0=xcrchi=0xFF; //reset CRC values
output_high(m0); //set up the MX614 to transmit
outbyte = sram_1k[txoutadd++]; //reads the C0 byte
if (txoutadd == 1024) txoutadd = 512; //rollover if necessary
outbyte = sram_1k[txoutadd++]; //reads the next byte (00 if it's data, 02 or 03
for command)
if (txoutadd == 1024) txoutadd = 512; //rollover if necessary
if (outbyte == 2) setParms(2); //set persist value
if (outbyte == 3) setParms(3); //set slottime value
if (outbyte ==6) sendParms(); //send Persist and Slottime
if (outbyte == 0) output_high(PTT); //key the transmitter, but only if data, not
KISS command
if (outbyte > 6){ //need to reset... this is not a valid kiss frame
output_low(PTT);
resetxmit();
return;
}
if (firstpack){ //got to do the txdelay if this is the first frame
txdelay = gettxdelay(); //read the txdelay pot
flag = TRUE; //start with some flags (no bit stuffing)
set_rtcc(0); //set the timer to zero to begin timing
for (c=0;c<txdelay;c++) (sendbyte(0)); //send nulls while TXDelay
sendbyte(0x7E); //send flag
sendbyte(0x7E); //each byte takes approx 6.7 ms
flag = FALSE; //done with flags
firstpack = false; //subsequent frames in this transmission don't need
txdelay
}
outbyte = sram_1k[txoutadd++]; //reads the first address byte.
if (txoutadd == 1024) txoutadd = 512; //rollover if necessary
While (outbyte != 0xC0){ //does the kiss substitutions and transmits byte
if(!input(sel0)) break;
switch (outbyte){ //do KISS substitutions, if necessary
case 0xDB : db = TRUE; break;
case 0xDC : if (db == TRUE){

```

```

        sendbyte(0xC0);
        db = FALSE;
    }else sendbyte(0xDC);
    break;
    case 0xDD : if (db == TRUE){
        sendbyte(0xDB);
        db = FALSE;
    }else sendbyte(0xDD);
    break;
    default : sendbyte(outbyte);
} //end of switch
outbyte = sram_1k[txoutadd++];
if (txoutadd == 1024) txoutadd = 512;
} //end of while
crcflag = TRUE; //now sending CRC... don't do crc calc on these bytes
xrclo = xrclo^0xff; //finish CRC calculation
xcrchi = xcrchi^0xff;
sendbyte(xrclo); //send the low byte of crc
sendbyte(xcrchi); //send the high byte of crc
FLAG = TRUE; //going to send flags now (no bit stuffing)
crcflag = false; //done sending CRC
sendbyte(0x7e);sendbyte(0x7e); // Send flags to end packet
flag = false; //done with flags
c0xmit--; // decrement the number of frames waiting.
} //end of while c0xmit
output_low(PTT); //unkey PTT
output_low(radtx); //leave transmit pin low
output_low(M0); //MX 614 to recv mode
disable_interrups(INT_TIMER2); //reset slottime/persist
slotcount = 0;
xmitlatch = false;
}

//*****RECEIVE ROUTINES FOLLOW*****

void addcrc(int crbyte) //ongoing computation of received CRC
int k, bt;
for(k=0; k<8; k++){
    bt = crbyte & 0x01;
    #asm
        //BCF 03,0
        BCF status,0
        RRCF crchi,F //RRF crchi,F //rotates the entire 16 bits to the right
        RRCF crclo,F //RRF crclo,F
    #endasm
    if (((status & 0x01)^(bt)) == 0x01){
        crchi = crchi^0x84;
        crclo = crclo^0x08;
    }
    rotate_right(&crbyte,1);
}
}

#int_rb //interrupt routine for change on B
void resetclock(){ //resyncs the bit timing on a zero
    if(!input(sel0)){ //Mode 0: ASK-Morse
        if(input(radrx) == 1){ //Change to signal

```

```

disable_interrupts(INT_TIMER3);
bit_clear(T3CON, 0); //disable timer3
signalTime = 0;
enable_interrupts(INT_TIMER1);
set_timer1(60536);
bit_set(T1CON, 0); //enable timer1
//zeroTime[1] = zeroTime[0];
if(change == 0) zeroTime[0] = silenceTime;
else change = 0;
//output_toggle(LED);
//output_low(LED);
}
else{ //Change to silence (logic 0)
disable_interrupts(INT_TIMER1);
bit_clear(T1CON, 0); //disable timer1
silenceTime = 0;
enable_interrupts(INT_TIMER3);
set_timer3(60536);
bit_set(T3CON, 0); //enable timer3
//onesTime[1] = onesTime[0];
if(signalTime > 0){ //1msec debounce
onesTime[0] = signalTime;
newp = 1;
}
else change = 1; //bounce detected
//output_high(LED);
}
//enable_interrupts(INT_RB);
}

else{ //Mode 1: FSK-AX.25
if (input(radrx) != oldstate){ //double check just to make sure the pin has really changed
disable_interrupts(INT_RB); //no more change on B interrupts until the next bit period
set_rtcc(240); //13 ticks (416 us) to next bit reading
bit_clear(intcon,2); //don't let it cause an interrupt immediately
}
}
}

#int_timer1
void count1ms(void){
signalTime++;
set_timer1(60536);
}

#int_timer3
void every1ms(void){
silenceTime++;
set_timer3(60536);
}

#INT_RTCC //interrupt routine for timer int
void rcvbits(){ //fires every 832 us
int rbit;

//output_high(PIN_D0);
set_rtcc(224); //reset the clock

```

```

bit_clear(intcon,2);          //don't let it generate an immediate interrupt
if (ones == 7){
    done = true;              //seven ones -- noise from the MX-614... time to start over
    rcvinadd = begin - 2;     //go back to before the "noise" data
    if (rcvinadd == -2) rcvinadd = 510;
    if (rcvinadd == -1) rcvinadd = 511;
    return;                   //we're done receiving, so leave
}
if (!fiveones){              //do this unless we had 5 ones on last bit
    rbit = 1;
    if (input(radrx) != oldstate){ //if this is a zero
        rbit = ones = 0;        //reset ones since this was a zero
        oldstate = !oldstate;   //flip oldstate variable to reflect the change
    }else ones++;              //else it must be a one so increment ones
    shift_right(&rcvbyte,1,rbit); //shift the new bit in on the right
    bitcount++;                //increment number of bits received in this byte
}
if (fiveones) {              //if this is the bit after 5 consecutive ones
    fiveones = false;          //turn it off, since we are processing it here
    if (input(radrx) != oldstate){ //if bit is a zero, drop it (no shift right)
        ones = 0;              //reset ones since this was a zero
        oldstate = !oldstate;   //flip oldstate variable to reflect the change
    }else{                     //otherwise this is another 1
        ones++;                //this is the sixth one ... must be a flag or noise
        shift_right(&rcvbyte,1,1); //if it is a one, process it.
        bitcount++;            //increment number of bits in this byte.
        isflag = true;
    } //end of else
} // end of if (fiveones)
if (ones == 5) fiveones = true; // if this is the fifth one, next bit gets
specialprocessing
if (bitcount == 8){          //do this if the byte is done
    if (count < 13) count++; //stop at 13 because we don't want this
rollover with long packets
    if ((rcvbyte == 0x7E) && (count < 3)) count = 0; //if this is a flag, don't start saving
bytes yet.

    if ((rcvbyte == 0x7E) && (isflag) && (count > 11)){ //do this when you get to the flag at
the frame end
        crchi = crchi ^ 0xFF; //finish the CRC calculation
        crclo = crclo ^ 0xFF;
        if ((crcli != minus2) || (crchi != minus1)){ //if CRC fails
            rcvinadd = begin; //revert to original address
        }else{ //if crc checks
            receivedpackets++; //increment the number of received packets that
haven't been sent out the serial port
            sram_1k[rcvinadd++] = 0xC0; //write C0 to finish fram
            if (rcvinadd == 512) rcvinadd = 0; //rollover if necessary
            sram_1k[rcvinadd++] = 0xC0; //write C0 to start next frame
            if (rcvinadd == 512) rcvinadd = 0; //rollover if necessary
            sram_1k[rcvinadd++] = 0x00; //write 00 to indicate data in this frame
            if (rcvinadd == 512) rcvinadd = 0; //rollover if necessary
            begin = rcvinadd; //move begin up since this was a good frame
        } //end of else
        crchi = crcli = 0xFF; //prepare for next frame
        count = 0; //prepare for next frame
    }
}

```

```

        if (!((rcvbyte == 0x7E) && (isflag))) { //do this if it is not a flag
            if (count > 2) { //if we are at byte 3 or higher, send the data, calc
                crc
                switch(minus2){ //process the byte, do KISS substitutions if
                    necessary
                    case 0xC0: sram_1k[rcvinadd++]=0xDB;if (rcvinadd == 512) rcvinadd = 0;
                    sram_1k[rcvinadd++]=0xDC;if (rcvinadd == 512) rcvinadd = 0;break;
                    case 0xDB: sram_1k[rcvinadd++]=0xDB;if (rcvinadd == 512) rcvinadd = 0;
                    sram_1k[rcvinadd++]=0xDD;if (rcvinadd == 512) rcvinadd = 0;break;
                    default: sram_1k[rcvinadd++] = minus2;if (rcvinadd == 512) rcvinadd = 0;
                }
                addcrc(minus2); //add to ongoing crc calculation
            }
            minus2 = minus1; //bump the history list
            minus1 = rcvbyte;
        }
        rcvbyte = bitcount = 0; //prepare for next byte
        isflag = false;
    } //end of if (bitcount==8)
    enable_interrupts(INT_RB ); //turn the change on b interrupt back on.
    //output_low(PIN_D0);
}

#INT_TIMER2
void t2int(){
    slotcount++; //increments the variable slotcount every 10 ms
    if (slotcount > SLOTTIME){ //if elapsed time is greater than SLOTTIME x 10 ms
        if (PERSIST > rand()) xmitlatch = true; //if persist passes, allow transmit
        slotcount = 0; //reset the elapsed time counter.
    }
    bit_clear(PIR1,1);
}

void findflag(){

    int time, countlegal = 0;
    //int time, countlegal, freetime = 0;

    oldstate = input(radrx); //remember the state of input pin
    while (input(radrx) == oldstate){ //wait for a change on the input pin
        if(!input(sel0)) break;
        sendbyteserial(); //in the meantime, process data to/from serial
    }
    port
    rcvbyteserial();
    restart_wdt();
}
set_rtcc(0); //set clock to 0
oldstate = !oldstate; //since the state changed, flip oldstate
do{ //keep listening until you get 7 legal interals
    if(!input(sel0)) break;
    if (COXMIT>0) enable_interrupts(INT_TIMER2 ); //enable timer2 interrupt if there is
data to transmit
    while (input(radrx) == oldstate){ //wait for portb to change again.
        if(!input(sel0)) break;
        sendbyteserial(); //flush serial outgoing buffer
        rcvbyteserial(); //grab bytes that show up on serial incoming
    }
}
buffer

```

```

restart_wdt();
if (xmitlatch) xmit();           //if there is data to send, transmit it
}
time = get_rtcc();              //time = how long it took.
set_rtcc(0);                   //reset clock
oldstate = !oldstate;          //update laststate to current

if ((time > 29) && (time < 35)) countlegal++;           //looking for a zero (a legal interval)
else if ((time > 221) && (time < 234)) countlegal+=3; //looking for 6 ones (a flag... legal
interval) Transmission will idle on 0x00 or 0x7E
else countlegal = 0;           //it wasn't a either of the above, start over.
}while(countlegal < 5);        //this is packet, not noise, now look for a flag
disable_interrupts(INT_TIMER2); //don't transmit while data is being received.
slotcount = 0;
xmitlatch = false;
do{
if(!input(sel0)) break;
while (input(radrx) == oldstate){ //looking for 6 1's in a row ... a flag
if(!input(sel0)) break;
sendbyteserial();
rcvbyteserial();
restart_wdt();
}
time = get_rtcc();              //how much time has elapsed?
set_rtcc(0);                   //start measuring time again
oldstate = !oldstate;          //the pin state has flipped so flip oldstate
variable
}while (time < 210);           //we are now at the end of the first flag
}

//*****MAIN PROGRAM LOOP FOLLOWS*****

void main(){
//int temp;
//short always=1;
setup_comparator(7); //all digital I/O ... no analog
set_tris_a(0b101111); //set up I/O on port a
//set_tris_a(0b100000); //set up I/O on port a
set_tris_b(0b11000010); //set up I/O on port b
output_high(LED); //make sure LED is off
output_low(PTT); //make sure PTT is off
output_high(m0);
output_high(m1); //bring up in 614 in zero power mode
//output_high(cs); //get fram ready
delay_ms(10); //wait for everything to settle
output_low(m1); //614 in rcv mode
output_low(m0);
port_b_pullups(true); //save one resistor on B&
//following determines serial port speed from jumpers.
if ((input(term1) & (input(term2)) == 0) set_uart_speed(1200); // both low = 1200
baud
if ((input(term1) && (!input(term2))) set_uart_speed(4800); //A5 high and B7 low
4800 baud
if (!(input(term1)) && (input(term2))) set_uart_speed(9600); //B7 high and A5 low
9600 baud
if ((input(term1) & (input(term2)) == 1) set_uart_speed(19200); // both high = 19200

```



```

baud
  setup_timer_2(T2_DIV_BY_16, 0xC3, 16);      //when turned on, will interrupt every 10
ms
  receivedpackets = c0xmit = 0;              //no waiting received or transmit packets at
powerup
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_128|RTCC_8_BIT);
  //setup_counters(RTCC_INTERNAL,RTCC_DIV_128); //now 32 ticks = 819 us
  rcvinadd = rcvoutadd = 0;      //start receive at the beginning of the fram space
  begin = 2;                      //begin is the start of data after C0 00
  txinadd = txoutadd = 512;      //2048 is start of transmit buffer
  resetxmit();
  //PERSIST = READ_EEPROM(2);    //read Persist from memory
  //SLOTTIME = READ_EEPROM(3);   //read SLOTTIME from memory
  enable_interrups(GLOBAL);

  setup_timer_1(T1_INTERNAL|T1_DIV_BY_1); //5000 ticks = 1msec (from 60536)
  setup_timer_3(T3_INTERNAL|T3_DIV_BY_1);
  change = 0;
  i=0;
  j=0;

  //main loop
  while(TRUE){

  while(always0 == 1){
    if(input(sel0)) break;
    enable_interrups(INT_RB);
    while(newp == 0){ //wait for new pulse, filter bounces
      if(input(sel0)) break;
    }
    //if(newp == 1){
    disable_interrups(INT_RB);

    if(zeroTime[0] > 7*DOT_TIME) printf("\r"); //end of word
    else if(zeroTime[0] > 3*DOT_TIME) printf(" "); //end of letter
    if(onesTime[0] < 2*DOT_TIME) printf(".");
    else printf("-");

    //printf("\r%lu %lu", zeroTime[0], onesTime[0]);
    newp = 0;
  //}
}

while (always == 1){
  if(!input(sel0)) break;
  crclo = crchi = 0xFF;          //set up next packet to be received
  done = fiveones = false;
  ones= rcvbyte = bitcount= count = 0;
  sram_1k[rcvinadd++] = 0xC0;if (rcvinadd == 512) rcvinadd = 0; //start KISS frame
  sram_1k[rcvinadd++] = 0x00;if (rcvinadd == 512) rcvinadd = 0;
  findflag();                    //go find a flag
  output_low(LED);               //packet is here, turn on DCD LED
  set_rtcc(207);                 //delay 1.5 bit intervals to get to the middle of the
next bit.
  bit_clear(intcon,2);           //don't get an interrupt right away
  enable_interrups(INT_RTCC | INT_RB ); //enable interrups to begin to collect
data

```

```

while (!done) { //Do this while data is being received.. stop when
there is noise
    if(!input(sel0)) break;
    sendbyteserial(); //send received data out serial port
    rcvbyteserial(); //take in data to transmit from serial port
    restart_wdt();
}
output_high(LED); //packet is over, turn off DCD LED
if (SLOTTIME > 0){ //reset transmit holdoff counter unless SLOTTIME
== 0)
    xmitlatch = false;
    set_timer2(0);
    slotcount = 0;
}
disable_interrups(INT_RTCC |INT_RB ); //disable interrups to prepare for
next findflag
xmit();
if(!input(sel0)) break;
}

} //end of main loop
}

```

F.2 Código fuente del microcontrolador de control

*tnc_cu.c

```

/*
 * File: tnc_cu.c
 * Author: Dante Inga
 *
 * Created on 9 de febrero de 2013, 12:03 PM
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <pic16f877a.h>
#include "Init.h"
#include "xlcd.h"
#include "usart.h"

#pragma config BOREN = OFF
#pragma config CPD = OFF //No EE protection
#pragma config WRT = OFF
#pragma config FOSC = HS //High speed Osc
#pragma config WDTE = OFF //No Watch Dog Timer
#pragma config CP = OFF

```

```

#pragma config LVP = OFF //No low voltage prgming, B3(PIC16) used for I/O
#pragma config PWRTE = OFF

#define FOSC 2000000L /* target device system clock frequency */
#define FCYC (FOSC/4L) /* target device instruction clock frequency */
#define _XTAL_FREQ FOSC /* required for __delay_ms, __delay_us macros */

unsigned char buff_UART;
unsigned short flag_UART, flag_LCD;
unsigned char LCD_row, LCD_col;

int main(void) {
    GPIO_Init();
    USART_Init();
    XLCDInit();

    B1_GND();
    IND_OFF();
    XLCDClear();
    XLCDL1home();
    Delay(60000);
    IND_ON();
    XLCDPutRamString("...BIENVENIDO...");
    XLCDL2home();
    XLCDPutRamString("*****@TNC-M*****");
    Delay(60000);
    Delay(60000);
    Delay(60000);
    Delay(60000);
    IND_OFF();
    XLCDClear();
    XLCDL1home();
    LCD_row = 1;
    LCD_col = 1;
    Delay(60000);

    if(PORTEbits.RE2 == 0){ //manual mode
        IND_ON();
        XLCDClear();
        XLCDL1home();
        XLCDPutRamString("*****@TNC-M*****");
        //PORTAbits.RA3 = 1; //enable tx
        while(PORTEbits.RE2 == 0){
            PORTAbits.RA1 = PORTEbits.RE0; //modem lsb
            PORTAbits.RA2 = PORTEbits.RE0; //codec lsb
            PORTAbits.RA0 = PORTEbits.RE1; //codec msb
            PORTAbits.RA5 = PORTEbits.RE1; //modem msb
            XLCDL2home();
            if(PORTEbits.RE1 == 0){
                if(PORTEbits.RE0 == 0) XLCDPutRamString("Modo Manual: MD1");
                else if(PORTEbits.RE0 == 1) XLCDPutRamString("Modo Manual: MD2");
            }
            else if(PORTEbits.RE1 == 1){
                if(PORTEbits.RE0 == 0) XLCDPutRamString("Modo Manual: MD3");
                else if(PORTEbits.RE0 == 1) XLCDPutRamString("Modo Manual: MD4");
            }
        }
    }
}

```

```

XLCDL2home();
XLCDPutRamString("Modo Automatico!");
}

while(1){
  while(!flag_UART);
  flag_UART = 0;
  if(buff_UART == '\r'){
    XLCDClear();
    XLCDL1home();
    LCD_row = 1;
    LCD_col = 1;
  }else{
    //Control modems and codecs
    if((LCD_row == 2) & (LCD_col == 16)){
      if(buff_UART == '1'){
        PORTAbits.RA0 = 0; //codec msb
        PORTAbits.RA2 = 0; //codec lsb
        PORTAbits.RA5 = 0; //modem msb
        PORTAbits.RA1 = 0; //modem lsb
      }
      else if(buff_UART == '2'){
        PORTAbits.RA0 = 0;
        PORTAbits.RA2 = 1;
        PORTAbits.RA5 = 0;
        PORTAbits.RA1 = 1;
      }
      else if(buff_UART == '3'){
        PORTAbits.RA0 = 1;
        PORTAbits.RA2 = 0;
        PORTAbits.RA5 = 1;
        PORTAbits.RA1 = 0;
      }
      else if(buff_UART == '4'){
        PORTAbits.RA0 = 1;
        PORTAbits.RA2 = 1;
        PORTAbits.RA5 = 1;
        PORTAbits.RA1 = 1;
      }
    }
    //Print information received
    XLCDPut(buff_UART);
    LCD_col++;
    if(LCD_col == 17){
      LCD_col = 1;
      LCD_row++;
      XLCDL2home();
    }
    if(LCD_row == 3){
      LCD_row = 1;
      IND_TOG();
      XLCDL1home();
    }
  }
}
//return 0;
}

```

```

void interrupt isr(void){
    //Service Serial interrupt
    if(PIR1bits.RCIF){
        //PIR1bits.RCIF = 0; //read-only bit, cleared by hardware
        buff_UART = RCREG;
        flag_UART = 1;
    }
    //Service external interrupt
    if(INTCONbits.INTF){

    }
    //Service Timer0 interrupt
    if(INTCONbits.T0IF){

    }
}
}

```

*Init.h

```

/*
 * File: Init.h
 * Author: Dante Inga
 *
 * Created on 9 de febrero de 2013, 12:09 PM
 */

#ifndef INIT_H
#define INIT_H

#include <pic16f877a.h>
#include <xc.h>

#define IND_ON() PORTBbits.RB0 = 1
#define IND_OFF() PORTBbits.RB0 = 0
#define IND_TOG() PORTB ^= 0x01
#define B1_GND() PORTBbits.RB1 = 0
#define B1_VCC() PORTBbits.RB1 = 1

void Delay(unsigned int MAX);
void GPIO_Init(void);

#endif /* INIT_H */

```

*Init.c

```

/*
 * File: Init.c
 * Author: Dante Inga
 *

```

```

* Created on 9 de febrero de 2013, 12:10 PM
*/

#include "Init.h"

void Delay(unsigned int MAX){
    unsigned int i;
    for(i=0; i<MAX; i++){
        ;
    }
}

void GPIO_Init(void){
    PORTA = 0x00; //Initialize PORTA by clearing output data latches
    ADCON1 = 0x06; //Configure PortA as digital inputs
    TRISA = 0b00010000; //TRISA<7:6> are always read as '0'
    //TRISA = 0b011000; //PIN_A3 of process unit?
    TRISB = 0b11111100;
    TRISC = 0b10111111;
    TRISD = 0b00001000;
    TRISEbits.TRISE0 = 1;
    TRISEbits.TRISE1 = 1;
    TRISEbits.TRISE2 = 1;
    TRISEbits.PSPMODE = 0;
}

```

*xlcDef.h

```

//Basado en librería de Danny Ng (http://www.bitguides.com/pic18-Guides/lcd)

////////////////////////////////////
// LCD Settings
////////////////////////////////////

/* DATA_PORT defines the port to which the LCD data lines are connected */

#define XLCD_DATAPORT PORTD
#define XLCD_DATAPORT_TRIS TRISD
// #define XLCD_DATAPORT PORTB
// #define XLCD_DATAPORT_TRIS TRISB
#define XLCD_4BIT // For 4 bit communication
#define XLCD_UPPER // Upper Nibble of 4 bit used for data
// #define XLCD_LOWER // Lower Nibble of 4 bit used for data
// #define XLCD_8BIT // For 8 bit communication

/* CTRL_PORT defines the port where the control lines are connected.
 * These are just samples, change to match your application.
 */

#define XLCD_RWPIN PORTDbits.RD2
#define XLCD_RWPIN_TRIS TRISDbits.TRISD2
#define XLCD_RSPIN PORTDbits.RD1
#define XLCD_RSPIN_TRIS TRISDbits.TRISD1

```

```

#define XLCD_ENPIN    PORTDbits.RD0
#define XLCD_ENPIN_TRIS TRISDbits.TRISD0
//#define XLCD_RWPIN    LATCbits.LATC1
//#define XLCD_RWPIN_TRIS TRISCbits.TRISC1
//#define XLCD_RSPIN    LATCbits.LATC0
//#define XLCD_RSPIN_TRIS TRISCbits.TRISC0
//#define XLCD_ENPIN    LATCbits.LATC2
//#define XLCD_ENPIN_TRIS TRISCbits.TRISC2

/* LCD command setting */
#define XLCD_READBFMODE // Check Busy flag
//#define XLCD_DELAYMODE // Use Delay
//#define XLCD_RW_GROUND // If Delay mode and only write is needed

/* LCD function Setting */
#define XLCD_2LINE
#define XLCD_FONT5x8
#define XLCD_BLOCK
#define XLCD_DISPLAYON
#define XLCD_CURSOROFF
#define XLCD_BLINKOFF
#define XLCD_CURSOR_INCREMENT
#define XLCD_DISPLAY_NOSHIFT

```

*xlcd.h

```

/*****
 *
 *          Modified LCD access routines for XC8
 *
 *****/
 *
 * Compiler:    XC8
 *
 * Author      Date      Comment
 * ~~~~~~
 * Danny Ng    6/Aug/12  1. Removed XLCDPutRomString
 *                  2. Port Setting in xlcdDef.h
 *****/

/*****
 *
 *          External LCD access routines defs
 *
 *****/
 * FileName:    XLCD.h
 * Dependencies: compiler.h
 * Processor:   PIC18
 * Compiler:    MCC18 v1.00.50 or higher
 *              HITECH PICC-18 V8.10PL1 or higher
 * Company:     Microchip Technology, Inc.
 *
 * Software License Agreement

```

```

*
* The software supplied herewith by Microchip Technology Incorporated
* (the ?Company?) for its PICmicro? Microcontroller is intended and
* supplied to you, the Company?s customer, for use solely and
* exclusively on Microchip PICmicro Microcontroller products. The
* software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are reserved.
* Any use in violation of the foregoing restrictions may subject the
* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN ?AS IS? CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
* Author          Date   Comment
* ~~~~~
* Naveen Raj      6/9/03  Original    (Rev 1.0)
*****/
#endif __XLCD_H
#define __XLCD_H
#define AddFile ///ADD_PROC_INC_FILE
#include <xc.h>
#include "xlcdDef.h"

void XLCDInit(void);           //to initialise the LCD
void XLCDPut(char data);      //to put data to be displayed
void XLCDPutRamString(char *string); //to display data string in RAM
char XLCDIsBusy(void);        //to check Busy flag
void XLCDCommand(unsigned char cmd); //to send commands to LCD
unsigned char XLCDGetAddr(void);
char XLCDGet(void);

#define XLCDL1home() XLCDCommand(0x80)
#define XLCDL2home() XLCDCommand(0xC0)
#define XLCDClear() XLCDCommand(0x01)
#define XLCDReturnHome() XLCDCommand(0x02)

void XLCDDelay15ms(void);
void XLCDDelay4ms(void);
void XLCDDelay100us(void);
void XLCD_Delay500ns(void);
void XLCDDelay(void);

#endif

```

***xlcd.c**

```

/*****

```



```

*
*           Modified LCD access routines for XC8
*
*****
*
* Compiler:   XC8
*
* Author      Date      Comment
* ~~~~~
* Danny Ng    6/Aug/12  1. Removed XLCDPutRomString
*              2. Delay Setting Included in xlcd.c
* ~~~~~/
/*****
*
*           External LCD access routines
*
*****
* FileName:   XLCD.c
* Dependencies: xlcd.h
* Processor:  PIC18
* Compiler:   MCC18 v1.00.50 or higher
*             HITECH PICC-18 V8.10PL1 or higher
* Company:    Microchip Technology, Inc.
*
* Software License Agreement
* The software supplied herewith by Microchip Technology Incorporated
* (the "Company") for its PICmicro? Microcontroller is intended and
* supplied to you, the Company's customer, for use solely and
* exclusively on Microchip PICmicro Microcontroller products. The
* software is owned by the Company and/or its supplier, and is
* protected under applicable copyright laws. All rights are reserved.
* Any use in violation of the foregoing restrictions may subject the
* user to criminal sanctions under applicable laws, as well as to
* civil liability for the breach of the terms and conditions of this
* license.
*
* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
*
* HiTech PICC18 Compiler Options excluding device selection:
* -FAKELOCAL -G -E -C
*
* Author      Date      Comment
* ~~~~~
* Naveen Raj   6/9/03    Original    (Rev 1.0)
* ~~~~~/
#include "delays.h"
#include "xlcd.h"
char _vXLCDreg =0;      //Used as a flag to check if from XLCDInit()
/*****
* Function      : void XLCDInit(void)

```

```

* PreCondition   : None
* Input         : None
* Output        : None
* Side Effects   : None
* Overview      : LCD is initialized
* Note         : This function will work with all Hitachi HD447780
*              : LCD controller.
*              : *****/
void XLCDInit(void)
{
/*This par of the code is initialization by instruction*/
_vXLCDreg=1;

//PORT initialization
#ifdef XLCD_8BIT //8-bit mode, use whole port
    XLCD_DATAPORT_TRIS = 0x00; //make DATAPORT output
    XLCD_DATAPORT = 0;
#endif

#ifdef XLCD_4BIT //4bit mode
    #ifdef XLCD_UPPER //Upper 4-bits of the DATAPORT output
        XLCD_DATAPORT_TRIS &= 0x0f;
        XLCD_DATAPORT &= 0x0f;
    #else //Lower 4-bits of the DATAPORT output
        XLCD_DATAPORT_TRIS &= 0xf0;
        XLCD_DATAPORT &= 0xf0;
    #endif
#endif
//end of data port initialization

//control port initialization
XLCD_RSPIN_TRIS =0; //make control ports output
XLCD_ENPIN_TRIS =0;
#ifdef XLCD_RW_GROUND
XLCD_RWPIN_TRIS =0; //if RW pin grounded
#endif

XLCD_RSPIN =0; //clear control ports
XLCD_ENPIN =0;
#ifdef XLCD_RW_GROUND
XLCD_RWPIN=0; //if RW pin grounded
#endif

//initialization by instruction

XLCDDelay15ms();
#ifdef XLCD_8BIT // 8-bit mode interface
    XLCD_DATAPORT = 0b00110000; // Function set cmd(8-bit interface)
#endif

#ifdef XLCD_4BIT
    #ifdef XLCD_UPPER // Upper nibble interface
        XLCD_DATAPORT &= 0x0f; // Clear upper port
        XLCD_DATAPORT |= 0b00110000;
    #endif

```

```

#else // Lower nibble interface
XLCD_DATAPORT  &= 0xf0; // Clear lower port
XLCD_DATAPORT |= 0b00000011; // Function set cmd(4-bit interface)
#endif
#endif
XLCD_ENPIN = 1; // Clock the cmd in
XLCD_Delay500ns();
XLCD_ENPIN = 0;
/////////////////////////////////////////////////////////////////
XLCDDelay4ms();
#ifdef XLCD_8BIT // 8-bit mode interface
XLCD_DATAPORT = 0b00110000; // Function set cmd(8-bit interface)
#endif

#ifdef XLCD_4BIT
#ifdef XLCD_UPPER // Upper nibble interface
XLCD_DATAPORT &= 0x0f; // Clear upper port
XLCD_DATAPORT |= 0b00110000;
#else // Lower nibble interface
XLCD_DATAPORT &= 0xf0; // Clear lower port
XLCD_DATAPORT |= 0b00000011; // Function set cmd(4-bit interface)
#endif
#endif
XLCD_ENPIN = 1; // Clock the cmd in
XLCD_Delay500ns();
XLCD_ENPIN = 0;
/////////////////////////////////////////////////////////////////
// XLCDDelay100us();
XLCDDelay4ms();
#ifdef XLCD_8BIT // 8-bit mode interface
XLCD_DATAPORT = 0b00110000; // Function set cmd(8-bit interface)
#endif

#ifdef XLCD_4BIT
#ifdef XLCD_UPPER // Upper nibble interface
XLCD_DATAPORT &= 0x0f; // Clear upper port
XLCD_DATAPORT |= 0b00110000;
#else // Lower nibble interface
XLCD_DATAPORT &= 0xf0; // Clear lower port
XLCD_DATAPORT |= 0b00000011; // Function set cmd(4-bit interface)
#endif
#endif

XLCD_ENPIN = 1; // Clock the cmd in
XLCD_Delay500ns();
XLCD_ENPIN = 0;

//required only for 4 bit interface as per LCDdatasheet
#ifdef XLCD_4BIT
XLCDDelay4ms();
#ifdef XLCD_UPPER // Upper nibble interface
XLCD_DATAPORT &= 0x0f; // Clear upper port
XLCD_DATAPORT |= 0b00110000;
#else // Lower nibble interface
XLCD_DATAPORT &= 0xf0; // Clear lower port
XLCD_DATAPORT |= 0b00000010; // Function set cmd(4-bit interface)

```

```

#endif

    XLCD_ENPIN = 1;          // Clock the cmd in
    XLCD_Delay500ns();
    XLCD_ENPIN = 0;
#endif
//-----
//function set command "0 0 1 DL N F X X"
//-----

#ifdef XLCD_8BIT          // if 8bit

    #ifdef XLCD_1LINE
        #ifdef XLCD_FONT5x8
            XLCDCommand(0b00110000); //if 1Line 5x8
        #else
            XLCDCommand(0b00110100); //if 1Line 5x10
        #endif
    #endif

    #ifdef XLCD_2LINE
        #ifdef XLCD_FONT5x8
            XLCDCommand(0b00111000); //if 2Line 5x8
        #else
            XLCDCommand(0b00111100); //if 2Line 5x10
        #endif
    #endif
#endif

#ifdef XLCD_4BIT          //if 4bit
    #ifdef XLCD_1LINE
        #ifdef XLCD_FONT5x8
            XLCDCommand(0b00100000); //if 1Line 5x8
        #else
            XLCDCommand(0b00100100); //if 1Line 5x10
        #endif
    #else
        #ifdef XLCD_FONT5x8
            XLCDCommand(0b00101000); //if 2Line 5x8
        #else
            XLCDCommand(0b00101100); //if 2Line 5x10
        #endif
    #endif
#endif

XLCDCommand(0b00001000); //display off
XLCDCommand(0b00000001); //display clear
/////////////////////////////////////////////////////////////////
//Entry mode setting
/////////////////////////////////////////////////////////////////
//Entry mode command " 0 0 0 0 1 ID S "
//ID =0 no cursor increment during read and write
//ID =1 cursor increment during read and write
//S =0 no display during read and write
//S =1 display shift

#ifdef XLCD_CURSOR_INCREMENT
    #ifdef XLCD_DISPLAY_SHIFT

```

```

    XLCDCommand(0b00000111); //if cursor inc and display shift
  #endif
  #ifdef XLCD_DISPLAY_NOSHIFT
    XLCDCommand(0b00000110); //if cursor inc and no display shift
  #endif
#endif

#ifdef XLCD_CURSOR_NOINCREMENT
  #ifdef XLCD_DISPLAY_SHIFT
    XLCDCommand(0b00000101); //if no cursor increment, but with display shift
  #endif
  #ifdef XLCD_DISPLAY_NOSHIFT
    XLCDCommand(0b00000100); //if no cursor increment, and no display shift
  #endif
#endif

////////////////////////////////////
//Display on off ,Blink ,cursor command set
// //////////////////////////////////////
//"0 0 0 0 1 D C B "
//D=1 display on, C=1 cursor on, B=1 blink on

#ifdef XLCD_DISPLAYON
  #ifdef XLCD_CURSORON
    #ifdef XLCD_BLINKON
      XLCDCommand(0b00001111); //display on cursor on blink on
    #else
      XLCDCommand(0b00001110); //display on cursor on blink off
    #endif
  #endif
#endif

  #ifdef XLCD_CURSOROFF
    #ifdef XLCD_BLINKON
      //XLCDCommand(0b00001001); //display on cursor off blink on
    #else
      XLCDCommand(0b00001100); // display on cursor off blink off
    #endif
  #endif
#endif

#ifdef XLCD_DISPLAYOFF
  XLCDCommand(0b00001000); //display off
#endif
_vXLCDreg=0;
// end of initialization
  return;
}

/*****
* Function      : void XLCDCommand(unsigned char cmd)
* PreCondition  : None
* Input        : cmd - Command to be set to LCD.
* Output       : None
* Side Effects  : None
* Overview     : None
* Note         : None
*****/

```

```

void XLCDCommand(unsigned char cmd)
{
if(_vXLCDreg==1)                //if called from XLCDinit routine is always Blocking
{
    #ifdef XLCD_DELAYMODE
    XLCDDelay();
    #endif
    #ifdef XLCD_READBFMODE
    XLCDIsBusy();
    #endif
}

if(_vXLCDreg==0)                //if not called from XLCDinit routine
{                                //if NON Block the user need to call XLCDIsBusy
    #ifdef XLCD_BLOCK            //and check the w reg status to check if the
    #ifdef XLCD_DELAYMODE        //module is free
    XLCDDelay();
    #endif
    #ifdef XLCD_READBFMODE
    XLCDIsBusy();
    #endif
    #endif
}

XLCD_RSPIN=0;
XLCD_ENPIN=0;
#ifdef XLCD_RW_GROUND
XLCD_RWPIN=0;
#endif

#ifdef XLCD_8BIT
    XLCD_DATAPORT = cmd;        // Write command to data port
    XLCD_ENPIN = 1;            // Clock the cmd in
    XLCD_Delay500ns();
    XLCD_ENPIN = 0;
#endif

#ifdef XLCD_4BIT
    #ifdef XLCD_UPPER
        XLCD_DATAPORT &=0x0f;    //clear port
        XLCD_DATAPORT |= cmd&0xf0; //write upper nibble to port
        XLCD_ENPIN = 1;        // Clock the cmd in
        XLCD_Delay500ns();
        XLCD_ENPIN = 0;

        XLCD_DATAPORT &= 0x0f;    //clear port
        XLCD_DATAPORT |= (cmd<<4)&0xf0; //shift left 4 times
        XLCD_ENPIN = 1;
        XLCD_Delay500ns();
        XLCD_ENPIN = 0;
    #endif

    #ifdef XLCD_LOWER
        XLCD_DATAPORT &=0xF0;    //clear port
        XLCD_DATAPORT |=((cmd>>4)&0x0f);
        XLCD_ENPIN = 1;        // Clock the cmd in
    #endif
#endif
}

```

```

    XLCD_Delay500ns();
    XLCD_ENPIN = 0;

    XLCD_DATAPORT &= 0xF0;          //clear port
    XLCD_DATAPORT |= cmd&0x0f;     //shift left 4 times
    XLCD_ENPIN = 1;
    XLCD_Delay500ns();
    XLCD_ENPIN = 0;
#endif

#endif

return;
}
/*****
 * Function      :XLCDPut()
 * PreCondition  :None
 * Input        :cmd - Command to be set to LCD.
 * Output       :None
 * Side Effects  :None
 * Overview     :None
 * Note         :None
 *****/
void XLCDPut(char data)
{
#ifdef XLCD_BLOCK
    #ifdef XLCD_DELAYMODE
        XLCDDelay();
    #endif
    #ifdef XLCD_READBFMODE
        XLCDIsBusy();
    #endif
#endif

#ifndef XLCD_RW_GROUND
    XLCD_RWPIN=0;
#endif
    XLCD_RSPIN=1;
    XLCD_ENPIN=0;
#ifdef XLCD_8BIT
    XLCD_DATAPORT=data;
    XLCD_ENPIN = 1;
    XLCD_Delay500ns();
    XLCD_ENPIN = 0;
#endif
#ifdef XLCD_4BIT
    #ifdef XLCD_UPPER
        XLCD_DATAPORT &=0x0f;          //clear port
        XLCD_DATAPORT |= data&0xf0;   //write upper nibble to port
        XLCD_ENPIN = 1;                // Clock the cmd in
        XLCD_Delay500ns();
        XLCD_ENPIN = 0;

        XLCD_DATAPORT &= 0x0f;          //clear port
        XLCD_DATAPORT |= (data<<4)&0xf0; //shift left 4 times
    #endif
#endif
}

```

```

    XLCD_ENPIN = 1;
    XLCD_Delay500ns();
    XLCD_ENPIN = 0;
#endif
#ifdef XLCD_LOWER
    XLCD_DATAPORT &=0xF0;          //clear port
    XLCD_DATAPORT |=((data>>4)&0x0f);
    XLCD_ENPIN = 1;                // Clock the cmd in
    XLCD_Delay500ns();
    XLCD_ENPIN = 0;

    XLCD_DATAPORT &= 0xF0;        //clear port
    XLCD_DATAPORT |= data&0x0f ;  //shift left 4 times
    XLCD_ENPIN = 1;
    XLCD_Delay500ns();
    XLCD_ENPIN = 0;
#endif

#endif

return;
}

#ifndef XLCD_RW_GROUND //need not compile any read command if RWpin grounded
/*****
* Function      :char XLCDIsBusy(void)
* PreCondition  :None
* Input        :None
* Output       :non-zero if LCD controller is ready to accept new
*              data or commandzero otherwise.
* Side Effects  :None
* Overview     :None
* Note         :None
*****/
char XLCDIsBusy(void)
{
    XLCD_RSPIN=0;
    XLCD_RWPIN=1;
    XLCD_ENPIN=0;

#ifdef XLCD_8BIT
    XLCD_DATAPORT_TRIS=0xFF;      //make port input
    XLCD_DATAPORT=0;
    XLCD_ENPIN=1;
    XLCD_Delay500ns();

    if(_vXLCDreg==1)              //will execute only if called from XLCDInit
    {
        while(XLCD_DATAPORT&0x80);
        XLCD_ENPIN=0;
        XLCD_DATAPORT_TRIS=0x00;  //make port output
    }
    return;
#endif
}

```



```

}

#ifdef XLCD_BLOCK
if(_vXLCDreg==0)           // will execute only if not called from XLCDInit
{
while(XLCD_DATAPORT&0x80);
XLCD_ENPIN=0;
XLCD_DATAPORT_TRIS=0x00;    //make port input
return;
}
#endif

#ifdef XLCD_NONBLOCK
if(_vXLCDreg==0)           //will execute only if not called from XLCDInit
{
if(XLCD_DATAPORT&0x80)     //Read bit 7 (busy bit)
{
XLCD_ENPIN=0;
XLCD_DATAPORT_TRIS=0x00;    //make port op
return 1;                    //Return TRUE
}
else
{
XLCD_ENPIN=0;
XLCD_DATAPORT_TRIS=0x00;    //make port op
return 0;                    //Return FALSE
}
}
}
#endif

#ifdef XLCD_4BIT

#ifdef XLCD_UPPER
XLCD_DATAPORT_TRIS|=0xF0;    //make upper port input
XLCD_DATAPORT &=0x0F;
XLCD_ENPIN=1;
XLCD_Delay500ns();
if(_vXLCDreg==1)           // will execute only if called from XLCDInit
{
while(XLCD_DATAPORT&0x80);
XLCD_ENPIN=0;
XLCD_Delay500ns();
XLCD_ENPIN=1;
XLCD_Delay500ns();
XLCD_ENPIN=0;
XLCD_DATAPORT_TRIS&=0x0F;    //make upper port output
return 0;
}
}

#ifdef XLCD_BLOCK
if(_vXLCDreg==0)           // When not called from the XLCDInit
{
while(XLCD_DATAPORT&0x80);
XLCD_ENPIN=0;

```

```

    XLCD_Delay500ns();
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    XLCD_ENPIN=0;
    XLCD_DATAPORT_TRIS&=0x0F;    //make upper port output
    return 0;
}
#endif

#ifdef XLCD_NONBLOCK
if(_vXLCDreg==0)    // will execute only if not called from XLCDInit
{
if(XLCD_DATAPORT&0x80)
{
    XLCD_ENPIN=0;
    XLCD_Delay500ns();
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    XLCD_ENPIN=0;
    XLCD_DATAPORT_TRIS&=0x0F;    //make port output
    return 1;    //Return TRUE
}
else
{
    XLCD_ENPIN=0;
    XLCD_Delay500ns();    // If high
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    XLCD_ENPIN=0;
    XLCD_DATAPORT_TRIS&=0x0F;    //make port input
    return 0;    // Return FALSE
}
}
#endif
#endif

#ifdef XLCD_LOWER
XLCD_DATAPORT_TRIS|=0x0F;    //make lower port input
XLCD_DATAPORT &=0xF0;
XLCD_ENPIN=1;
XLCD_Delay500ns();
if(_vXLCDreg==1)    // will execute only if called from XLCDInit
{
    while(XLCD_DATAPORT&0x08);
    XLCD_ENPIN=0;
    XLCD_Delay500ns();
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    XLCD_ENPIN=0;
    XLCD_DATAPORT_TRIS&=0xF0;    //make port output
    return 0;
}
#endif
#ifdef XLCD_BLOCK
if(_vXLCDreg==0)    //will execute only if called from XLCDInit
{

```

```

while(XLCD_DATAPORT&0x08);
XLCD_ENPIN=0;
XLCD_Delay500ns();
XLCD_ENPIN=1;
XLCD_Delay500ns();
XLCD_ENPIN=0;
XLCD_DATAPORT_TRIS&=0xF0;    //make port output
return 0;
}
#endif

#ifdef XLCD_NONBLOCK
if(!_vXLCDreg==0)           // will execute only if called from XLCDInit
{
    if(XLCD_DATAPORT&0x08)
    {
        XLCD_ENPIN=0;
        XLCD_Delay500ns();
        XLCD_ENPIN=1;
        XLCD_Delay500ns();
        XLCD_ENPIN=0;
        XLCD_DATAPORT_TRIS=0x00;    //make port input
        return 1;                // Return TRUE
    }
    else
    {
        XLCD_ENPIN=0;
        XLCD_Delay500ns();
        XLCD_ENPIN=1;
        XLCD_Delay500ns();
        XLCD_ENPIN=0;
        XLCD_DATAPORT_TRIS=0x00;    //make port input
        return 0;                // Return FALSE
    }
}
#endif
#endif
return 0;
}

/*****
* Function      :unsigned char XLCDGetAddr(void)
* PreCondition  :None
* Input         :None
* Output        :Current address byte from LCD
* Side Effects  :None
* Overview      :None
* Note          :The address is read from the character generator
*               RAM or display RAM depending on current setup.
*****/

unsigned char XLCDGetAddr(void)
{
char addr =0;

```

```

#ifdef XLCD_BLOCK
  #ifdef XLCD_DELAYMODE
    XLCDDelay();
  #endif
  #ifdef XLCD_READBFMODE
    XLCDIsBusy();
  #endif
#endif

XLCD_RSPIN=0;
XLCD_RWPIN=1;
XLCD_ENPIN=0;

#ifdef XLCD_8BIT
  XLCD_DATAPORT_TRIS=0xFF;           //make port input
  XLCD_ENPIN=1;
  XLCD_Delay500ns();
  addr=XLCD_DATAPORT;
  XLCD_ENPIN=0;
  XLCD_DATAPORT_TRIS=0x00;         //make port input
  return(addr&0x7F);
#endif
#ifdef XLCD_4BIT
  #ifdef XLCD_UPPER
    XLCD_DATAPORT_TRIS|=0xF0;       //make upper port input
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    addr=XLCD_DATAPORT&0xF0;
    XLCD_ENPIN=0;
    XLCD_Delay500ns();
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    addr|=(XLCD_DATAPORT>>4)&0x0F;
    XLCD_ENPIN=0;
    XLCD_DATAPORT_TRIS&=0x0F;      //make upper port output
    return(addr&0x7F);
  #endif

  #ifdef XLCD_LOWER
    XLCD_DATAPORT_TRIS|=0x0F;       //make lower port input
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    addr=(XLCD_DATAPORT<<4)&0xF0;
    XLCD_ENPIN=0;
    XLCD_Delay500ns();
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    addr|=XLCD_DATAPORT&0x0F;
    XLCD_ENPIN=0;
    XLCD_DATAPORT_TRIS&=0xF0;     //make port output
    return(addr&0x7F);
  #endif
#endif
}

```

```

/*****
* Function      :char XLCDGet(void)
* PreCondition  :None
* Input         :None
* Output        :Current data byte from LCD
* Side Effects  :None
* Overview      :None
* Note          :The data is read from the character generator
*               :RAM or display RAM depending on current setup.
*****/
char XLCDGet(void)
{
    char data=0;
#ifdef XLCD_BLOCK
    #ifdef XLCD_DELAYMODE
        XLCDDelay();
    #endif
    #ifdef XLCD_READBFMODE
        XLCDIsBusy();
    #endif
#endif
    XLCD_RSPIN=1;
    XLCD_RWPIN=1;
    XLCD_ENPIN=0;

#ifdef XLCD_8BIT
    XLCD_DATAPORT_TRIS=0xFF;
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    data=XLCD_DATAPORT;
    XLCD_ENPIN=0;
    XLCD_DATAPORT_TRIS=0x00;
    return(data);
#endif
#ifdef XLCD_4BIT
    #ifdef XLCD_UPPER
        XLCD_DATAPORT_TRIS |=0xf0;           //make upper input
        XLCD_ENPIN=1;
        XLCD_Delay500ns();
        data = XLCD_DATAPORT&0xf0;         //Read the upper nibble of data
        XLCD_ENPIN=0;
        XLCD_Delay500ns();
        XLCD_ENPIN=1;
        XLCD_Delay500ns();
        data |= ((XLCD_DATAPORT>>4)&0x0f); //Read the upper nibble of data
        XLCD_ENPIN=0;
        XLCD_DATAPORT_TRIS &=0x0f;        //make output
        return(data);
    #endif

    #ifdef XLCD_LOWER
        XLCD_DATAPORT_TRIS |=0x0F;         //make input
        XLCD_ENPIN=1;
        XLCD_Delay500ns();
        data = (XLCD_DATAPORT<<4)&0xf0;   //Read the upper nibble of data
        XLCD_ENPIN=0;
    #endif
#endif
}

```

```

    XLCD_Delay500ns();
    XLCD_ENPIN=1;
    XLCD_Delay500ns();
    data |= XLCD_DATAPORT&0x0f;          //Read the upper nibble of data
    XLCD_ENPIN=0;
    XLCD_DATAPORT_TRIS &=0xf0;          //make output
    return(data);
#endif
#endif

}

#endif      //end of #ifndef XLCD_RW_GROUND(all read commands)

/*****
 * Function      :XLCDPutRomString(rom char *string)
 * PreCondition  :None
 * Input        :None
 * Output       :Displays string in Program memory
 * Side Effects  :None
 * Overview     :None
 * Note         :is lways blocking till the string is written fully
 *****/
void XLCDPutRamString(char *string)
{
    while(*string)          // Write data to LCD up to null
    {
        #ifdef XLCD_NONBLOCK
            while(XLCDIsBusy());
        #endif

        XLCDPut(*string);    // Write character to LCD
        string++;            // Increment buffer
    }
    return;
}

/*****
 * Functions    : void XLCDDelay15ms(void)
 *              void XLCDDelay4ms (void)
 *              void XLCD_Delay500ns(void)
 * Note        : These function are needed to generate delays
 *              necessary to get the LCD functioning.
 *              Adjust the functions according to the oscillator
 *              that is used in the project
 *****/
void Delay1KTCYx(char times){
    char i, j;
    for(i=0; i<times; i++){
        //for(j=0; j<; j++){
        ;
        //}
    }
}
void XLCDDelay15ms (void)
{

```

```

    Delay1KTCYx(30);
}
void XLCDDelay4ms (void)
{
    Delay1KTCYx(8);
}
void XLCD_Delay500ns(void)
{
    Delay1KTCYx(1);
    //Nop();
    //Nop();
    //Nop();
}

/*****
 * Functions      : void XLCDDelay(void)
 * Note          : For LCD Blocking method. Delay for LCD
 *****/

void XLCDDelay(void){
    Delay1KTCYx(3);
}

```

*usart.h

```

/*
 * File: usart.h
 * Author: Dante Inga
 *
 * Created on 2 de febrero de 2013, 12:29 AM
 */

#ifndef USART_H
#define USART_H

#include <xc.h>

#define ENA_UART_INT() PIE1bits.RCIE = 1
#define DIS_UART_INT() PIE1bits.RCIE = 0

void send_Char(unsigned char ascii);
void send_String(unsigned char cadena[]); //null ended
void USART_Init(void);

#endif /* USART_H */

```

*usart.c

```

/*
 * File: usart.c

```

```

* Author: Dante Inga
*
* Created on 2 de febrero de 2013, 12:30 AM
*/

#include "usart.h"

void USART_Init(void){
    TXSTA = 0x26; //USART asynchronous mode
    RCSTA = 0x90; //Serial Port Enabled
    TRISCbits.TRISC6 = 0; //RC6 is output TXD
    TRISCbits.TRISC7 = 1; //RC7 is input RXD
    SPBRG = 129; // (FOSC/16/9600)-1 = 129.21
    //For interrupts
    INTCONbits.GIE = 1;
    INTCONbits.PEIE = 1;
    PIE1bits.RCIE = 1; //Enable interrupt
}

void send_Char(unsigned char ascii){
    while(PIR1bits.TXIF == 0); //wait while TX buffer is full
    TXREG = ascii;
}

void send_String(unsigned char cadena[]){ //null ended
    unsigned char i;
    i = 0;
    while(cadena[i] != '\0'){
        send_Char(cadena[i]);
        i++;
    }
}

```

F.3 Código fuente del software para la PC

*tncM.java

```

/*
* tncM
* Author: Dante Inga
* Used libraries:
* -Window Builder (http://www.eclipse.org/windowbuilder/)
* -RXTX serial (http://rxtx.qbang.org/wiki/index.php/Main\_Page)
*/

package my.tncM;

import java.io.IOException;
import java.io.InputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Enumeration;

```



```

import gnu.io.CommPort;
import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import java.awt.Font;
import javax.swing.SwingConstants;
import java.awt.Color;
import javax.swing.ScrollPaneConstants;

public class tncM extends JFrame {

    public class ReadSerialPort {

        public ReadSerialPort() {
            super();
        }

        public void connect (String portName) throws Exception {
            CommPortIdentifier portIdentifier =
CommPortIdentifier.getPortIdentifier(portName);
            if ( portIdentifier.isCurrentlyOwned() ){
                System.out.println("Error: Port is currently in use");
            } else {
                CommPort commPort = portIdentifier.open(this.getClass().getName(),2000);
                tncM.this.textField.setText("Recibiendo datos");
                if ( commPort instanceof SerialPort ) {
                    SerialPort serialPort = (SerialPort) commPort;

serialPort.setSerialPortParams(19200,SerialPort.DATABITS_8,SerialPort.STOPBITS_1,Seri
alPort.FLOWCONTROL_NONE);

                    InputStream in = serialPort.getInputStream();
                    (new Thread(new SerialReader(in))).start();

                } else {
                    System.out.println("Error: Only serial ports are handled by this example.");
                }
            }
        }
    }

    /** */

```

```

//public static class SerialReader implements Runnable {
public class SerialReader implements Runnable {
    InputStream in;
    BufferedReader reader;
    private final static String newline = "\n";
    private final static char fend = 0xC0; //'\u00C0';

    public SerialReader ( InputStream in ) {
        this.in = in;
        this.reader = new BufferedReader(new InputStreamReader(in));
    }

    public void run () {
        String line = null;
        String deco = null;
        //byte[] lineB;
        int i, f;
        try{
            while ((line = reader.readLine()) != null) {
                //System.out.println("Read line with " + line.length() + " characters: \"" +
line + "\"");
                if(line.startsWith(".") || line.startsWith("-")){ //ASK-Morse
                    deco = "";
                    i=0; f=0;
                    if(line.length() != 0){
                        //System.out.print("\n");
                        while(f < line.length()-1){
                            i = f;
                            while(!line.substring(f, f+1).equals(" ")){
                                f++;
                                if(f == line.length()) break;
                            }
                            if(line.substring(i, f).equals("-")) deco += "B";
//System.out.print("A");
                            if(line.substring(i, f).equals("-...")) deco += "B";
//System.out.print("B");
                            if(line.substring(i, f).equals("-.-")) deco += "C";
//System.out.print("C");
                            if(line.substring(i, f).equals("-..")) deco += "D";
//System.out.print("D");
                            if(line.substring(i, f).equals(".")) deco += "E";
//System.out.print("E");
                            if(line.substring(i, f).equals("..-")) deco += "F";
//System.out.print("F");
                            if(line.substring(i, f).equals("--")) deco += "G";
//System.out.print("G");
                            if(line.substring(i, f).equals("...")) deco += "H";
//System.out.print("H");
                            if(line.substring(i, f).equals("..")) deco += "I";
//System.out.print("I");
                            if(line.substring(i, f).equals(".-")) deco += "J";
//System.out.print("J");
                            if(line.substring(i, f).equals("-.-")) deco += "K";
//System.out.print("K");
                            if(line.substring(i, f).equals("-..")) deco += "L";
//System.out.print("L");
                            if(line.substring(i, f).equals("--")) deco += "M";

```

```

//System.out.print("M");
//System.out.print("N");
//System.out.print("O");
//System.out.print("P");
//System.out.print("Q");
//System.out.print("R");
//System.out.print("S");
//System.out.print("T");
//System.out.print("U");
//System.out.print("V");
//System.out.print("W");
//System.out.print("X");
//System.out.print("Y");
//System.out.print("Z");

if(line.substring(i, f).equals("-.")) deco += "N";
if(line.substring(i, f).equals("----")) deco += "O";
if(line.substring(i, f).equals("-.")) deco += "P";
if(line.substring(i, f).equals("---")) deco += "Q";
if(line.substring(i, f).equals("-.")) deco += "R";
if(line.substring(i, f).equals("...")) deco += "S";
if(line.substring(i, f).equals("-")) deco += "T";
if(line.substring(i, f).equals("..")) deco += "U";
if(line.substring(i, f).equals("...-")) deco += "V";
if(line.substring(i, f).equals("-.-")) deco += "W";
if(line.substring(i, f).equals("-..")) deco += "X";
if(line.substring(i, f).equals("-.-")) deco += "Y";
if(line.substring(i, f).equals("---")) deco += "Z";

f++;
}
}
//System.out.print(line.length() + " " + i + " " + f + "\t");
tncM.this.textRX.setText(deco);
tncM.this.textArea_1.append(deco + newline);
}
else{ //FSK-AX.25
deco = "";
i = 0;
//lineB = null;
if(line.length() != 0){
//lineB = line.getBytes("ASCII");
//System.out.println(line.length());
//System.out.println(line.charAt(3*line.length()/4));
/*
while((int)line.charAt(i) != (int)fend){ //start
System.out.printf("%d ", (int)line.charAt(i));
i++;
if(i == line.length()) break;
}
*/

//if(i > 0){
i = i + 3; //FEND|FEND|PORT
deco = deco + "> " + line.substring(i, i+6);

//destination

i = i + 7;
deco = line.substring(i, i+6) + " " + deco; //source
i = i + 7;
}
}
}

```

```

//while(((int) line.charAt(i+6)) % 2 == 0){
    deco = deco + " " + line.substring(i, i+6);

    i = i + 7;
//}
deco = deco + ":\n" + line.substring(i) + "\n"; //data
i = 0;
//}

}
tncM.this.textRX.setText(line);
tncM.this.textArea_1.append(line + newline);
}
//tncM.this.textRX.setText(deco);
//tncM.this.textArea_1.append(deco + newline);
//tncM.this.textRX.setText(line);
//tncM.this.textArea_1.append(line + newline);
}
}
catch ( IOException e ) {
    e.printStackTrace();
}
}
}
}

class StreamGobbler extends Thread
{
    InputStream is;
    String type;

    StreamGobbler(InputStream is, String type)
    {
        this.is = is;
        this.type = type;
    }

    public void run()
    {
        try
        {
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            String line=null;
            while ( (line = br.readLine()) != null)
                System.out.println(type + ">" + line);
        } catch (IOException ioe)
        {
            ioe.printStackTrace();
        }
    }
}

private static final long serialVersionUID = 1L;
private int COLS = 1000;
private String host;
private String cmd;

```

```

private String dir;

public void execJava (String host, String classPath, String dir, String _class, String args)
throws Exception {
    //public static void execJava (String host, String classPath, String dir, String _class, String
args) throws Exception {

    String cmd = "java -cp ." + classPath + " " + _class + " " + args;

    exec (host, dir, cmd);
}

public void exec (String cmd) throws Exception {
//public static void exec (String cmd) throws Exception {
exec ("localhost", null, cmd);
}

public void exec (String host, String dir, String cmd) throws Exception {
//public static void exec (String host, String dir, String cmd) throws Exception {

this.host = host;
this.cmd = cmd;
this.dir = dir;

if (!host.equals ("localhost"))
    cmd = " ssh " + host + " 'cd " + dir + "/src; " + cmd + "'";

//System.out.println (host);
//System.out.println (dir);
//System.out.println (cmd);

//Runtime.getRuntime().exec("./exec " + cmd + " &");
//Runtime.getRuntime().exec("./exec.pl " + cmd + " &");
Runtime.getRuntime().exec("perl /home/aet/Desktop/tncM/Srcs/exec.pl " + cmd + " &");

System.out.println (cmd);
}

public void restart () throws Exception {
//public static void restart () throws Exception {
String cmd = this.cmd;

if (!host.equals ("localhost"))
    cmd = " ssh " + host + " 'cd " + dir + "/src; " + cmd + "'";

//Runtime.getRuntime().exec ("./exec " + cmd);
Runtime.getRuntime().exec ("perl /home/aet/Desktop/tncM/Srcs/exec.pl " + cmd);
}

public void destroy () throws Exception {
//public static void destroy () throws Exception {
    //System.out.print(cmd);
    String killCmd =
" ps -eo pid,args --cols " + COLS + " | grep " + cmd +
"" | grep -v 'grep' | awk '{ print $1 }' " +
" | xargs echo -9 " +
" | xargs kill";

```

```

//System.out.print(killCmd);

if (!host.equals ("localhost"))
    killCmd = " ssh " + host + " \"" + killCmd + "\"";

//Runtime.getRuntime().exec (".exec " + killCmd);
Runtime.getRuntime().exec ("perl /home/aet/Desktop/tncM/Srcs/exec.pl" + killCmd);
}

private JPanel contentPane;
private JTextField txtIdRadio;
private JTextField txtPortRadio;
private JTextField txtIdRotor;
private JTextField txtPortRotor;
private JTextField txtCtrlPort;
private JTextField txtCtrlRate;
private JTextField txtDataPort;
private JTextField txtDataRate;
private JTextField textField;
private JTextField textRX;
private JTextArea textArea_1;

/**
 * Launch the application.
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                tncM frame = new tncM();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Create the frame.
 */
public tncM() {
    setTitle("\tINTERFAZ GRAFICA DEL TNC-M");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 410);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JButton btnEstacion = new JButton("Editar estación");
    btnEstacion.setFont(new Font("Dialog", Font.BOLD, 10));
    btnEstacion.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            //String cmd = "ls -l"; // this is the command to execute in the

```

Unix shell

```

//String cmd = "gedit
/home/aet/workspace/tncM/src/estacion.qth &"; // this is the command to execute in the Unix
shell

//String cmd = "gedit estacion.qth &";
String cmd = "gedit
/home/aet/Desktop/tncM/Srcs/estacion.qth &";
// create a process for the shell
ProcessBuilder pb = new ProcessBuilder("bash", "-c", cmd);
pb.redirectErrorStream(true); // use this to capture
messages sent to stderr

try{
Process shell = pb.start();
InputStream shellIn = shell.getInputStream(); // this captures
the output from the command

int shellExitStatus = shell.waitFor(); // wait for the shell to
finish and get the return code

//System.out.println(shellExitStatus);
// at this point you can process the output issued by the
command

// for instance, this reads the output and writes it to
System.out:

int c;
while ((c = shellIn.read()) != -1) {System.out.write(c);}
// close the stream
try {shellIn.close();} catch (IOException ignoreMe) {}
}catch (InterruptedException ev) {
throw new RuntimeException(ev);
}
}
catch (IOException ev) {
throw new RuntimeException(ev);
}
});
btnEstacion.setBounds(17, 12, 120, 22);
contentPane.add(btnEstacion);

JButton btnSatelites = new JButton("Editar satélites");
btnSatelites.setFont(new Font("Dialog", Font.BOLD, 10));
btnSatelites.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
//String cmd = "gedit
/home/aet/workspace/tncM/src/satelites.tle &"; // this is the command to execute in the Unix
shell

String cmd = "gedit
/home/aet/Desktop/tncM/Srcs/satelites.tle &";
// create a process for the shell
ProcessBuilder pb = new ProcessBuilder("bash", "-c", cmd);
pb.redirectErrorStream(true); // use this to capture
messages sent to stderr

try{
Process shell = pb.start();
InputStream shellIn = shell.getInputStream(); // this captures
the output from the command

int shellExitStatus = shell.waitFor(); // wait for the shell to
finish and get the return code

//System.out.println(shellExitStatus);
// at this point you can process the output issued by the

```

```

command
// for instance, this reads the output and writes it to
System.out:
int c;
while ((c = shellIn.read()) != -1) {System.out.write(c);}
// close the stream
try {shellIn.close();} catch (IOException ignoreMe) {}
}catch (InterruptedException ev) {
throw new RuntimeException(ev);
}
catch (IOException ev) {
throw new RuntimeException(ev);
}
});
btnSatelites.setBounds(17, 36, 120, 22);
contentPane.add(btnSatelites);

JButton btnEjecutarPredict = new JButton("Hallar órbitas");
btnEjecutarPredict.setFont(new Font("Dialog", Font.BOLD, 10));
btnEjecutarPredict.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
// System.out.println("Evento btnEjecutarPredict
detectado...");

// merc.this.exec("gpredict");
try {
// exec("gpredict");
exec("predict -s -t
/home/aet/Desktop/tncM/Srcs/satelites.tle -q /home/aet/Desktop/tncM/Srcs/estacion.qth");
} catch (Exception ev) {
throw new RuntimeException(ev);
}
tncM.this.textField.setText("predict ejecutado");
// System.out.println("Evento btnEjecutarPredict
ejecutado...");
}
});
btnEjecutarPredict.setBounds(12, 64, 130, 25);
contentPane.add(btnEjecutarPredict);

JButton btnRigctld = new JButton("Controlar radio");
btnRigctld.setFont(new Font("Dialog", Font.BOLD, 10));
btnRigctld.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
String comando = "rigctld -m " + txtIdRadio.getText() + " -r
/dev/" + txtPortRadio.getText();
try{
//exec("rigctld -m 1 -r /dev/ttyUSB0");
exec(comando);
}catch (Exception ev) {
throw new RuntimeException(ev);
}
tncM.this.textField.setText("rigctld ejecutado");
}
});
btnRigctld.setBounds(160, 64, 120, 25);

```



```

contentPane.add(btnRigctld);

JButton btnRotctld = new JButton("Controlar rotor");
btnRotctld.setFont(new Font("Dialog", Font.BOLD, 10));
btnRotctld.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String comando = "rotctld -m " + textldRotor.getText() + " -r
/dev/" + txtPortRotor.getText();
        try{
            //exec("rotctld -m 1 -r /dev/parport0");
            exec(comando);
        }catch (Exception ev) {
            throw new RuntimeException(ev);
        }
        tncM.this.textField.setText("rotctld ejecutado");
    }
});
btnRotctld.setBounds(309, 64, 120, 25);
contentPane.add(btnRotctld);

textldRadio = new JTextField();
textldRadio.setText("1");
textldRadio.setBounds(217, 12, 70, 19);
contentPane.add(textldRadio);
textldRadio.setColumns(10);

txtPortRadio = new JTextField();
txtPortRadio.setText("ttyUSB2");
txtPortRadio.setColumns(10);
txtPortRadio.setBounds(217, 32, 70, 19);
contentPane.add(txtPortRadio);

textldRotor = new JTextField();
textldRotor.setText("1");
textldRotor.setColumns(10);
textldRotor.setBounds(366, 12, 70, 19);
contentPane.add(textldRotor);

txtPortRotor = new JTextField();
txtPortRotor.setText("parport0");
txtPortRotor.setColumns(10);
txtPortRotor.setBounds(366, 32, 70, 19);
contentPane.add(txtPortRotor);

JLabel lblIdRadio = new JLabel("Radio ID:");
lblIdRadio.setFont(new Font("Dialog", Font.BOLD, 11));
lblIdRadio.setBounds(150, 16, 68, 15);
contentPane.add(lblIdRadio);

JLabel lblPortRadio = new JLabel("Radio port:");
lblPortRadio.setFont(new Font("Dialog", Font.BOLD, 11));
lblPortRadio.setBounds(150, 36, 68, 15);
contentPane.add(lblPortRadio);

JLabel lblIdRotor = new JLabel("Rotor ID:");
lblIdRotor.setFont(new Font("Dialog", Font.BOLD, 11));
lblIdRotor.setBounds(299, 16, 61, 15);

```

```

contentPane.add(lblDRotor);

JLabel lblPortRotor = new JLabel("Rotor port:");
lblPortRotor.setFont(new Font("Dialog", Font.BOLD, 11));
lblPortRotor.setBounds(299, 36, 75, 15);
contentPane.add(lblPortRotor);

JLabel lblCtrlPort = new JLabel("Ctrl port:");
lblCtrlPort.setBounds(165, 109, 70, 15);
contentPane.add(lblCtrlPort);

JLabel lblCtrlRate = new JLabel("Ctrl rate:");
lblCtrlRate.setBounds(165, 129, 70, 15);
contentPane.add(lblCtrlRate);

txtCtrlPort = new JTextField();
txtCtrlPort.setText("ttyUSB0");
txtCtrlPort.setBounds(235, 107, 61, 19);
contentPane.add(txtCtrlPort);
txtCtrlPort.setColumns(10);

txtCtrlRate = new JTextField();
txtCtrlRate.setText("9600");
txtCtrlRate.setColumns(10);
txtCtrlRate.setBounds(235, 127, 61, 19);
contentPane.add(txtCtrlRate);

JButton btnTnc = new JButton("Controlar TNC-M");
btnTnc.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        /*
        try{
            Process p = Runtime.getRuntime().exec(new
String[]{"bash", "-c", "/home/aet/Desktop/tncM/Srcs/backGround &"});
        } catch (IOException ev) {
            throw new RuntimeException(ev);
        }
        */

        String cmd = "/home/aet/Desktop/tncM/Srcs/backGround &";
// this is the command to execute in the Unix shell
// create a process for the shell
ProcessBuilder pb = new ProcessBuilder("bash", "-c", cmd);
pb.redirectErrorStream(true); // use this to capture
messages sent to stderr

        try{
            Process shell = pb.start();
//InputStream shellIn = shell.getInputStream(); // this
captures the output from the command
//int shellExitStatus = shell.waitFor(); // wait for the
shell to finish and get the return code
// at this point you can process the output issued by
the command
// for instance, this reads the output and writes it to
System.out:

            //int c;
//while ((c = shellIn.read()) != -1)

```

```

{System.out.write(c);}
// close the stream
//try {shellIn.close();} catch (IOException ignoreMe)
{}

//)catch (InterruptedException ev) {
//    throw new RuntimeException(ev);
System.out.println(cmd);
}
catch (IOException ev) {
throw new RuntimeException(ev);
}

/*
try{
    exec("/home/aet/Desktop/tncM/Srcs/backGround");
}catch (Exception ev) {
throw new RuntimeException(ev);
}
*/
tncM.this.textField.setText("controlando TNC-M");
}
});
btnTnc.setFont(new Font("Dialog", Font.BOLD, 10));
btnTnc.setBounds(165, 152, 130, 25);
contentPane.add(btnTnc);

JButton btnDetener = new JButton("Detener");
btnDetener.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //Eliminar el proceso "predict"
        cmd = "predict -s -t
/home/aet/Desktop/tncM/Srcs/satelites.tle -q /home/aet/Desktop/tncM/Srcs/estacion.qth";
        try{
            destroy();
            System.out.println("Proceso predict eliminado...");
        }catch (Exception ev) {
throw new RuntimeException(ev);
        }
        //Eliminar el proceso "rigctld"
        cmd = "rigctld -m 1 -r /dev/ttyUSB2";
        try{
            destroy();
            System.out.println("Proceso rigctld eliminado...");
        }catch (Exception ev) {
throw new RuntimeException(ev);
        }
        //Eliminar el proceso "rotctld"
        cmd = "rotctld -m 1 -r /dev/parport0";
        try{
            destroy();
            System.out.println("Proceso rotctld eliminado...");
        }catch (Exception ev) {
throw new RuntimeException(ev);
        }
        //Eliminar el proceso "backGround"
        cmd = "/home/aet/Desktop/tncM/Srcs/backGround";
        try{

```

```

destroy();
System.out.println("Proceso backGround
eliminado...");
    }catch (Exception ev) {
        throw new RuntimeException(ev);
    }
    //Cerrar el puerto de datos /dev/ttyUSB1
    //ReadSerialPort.SerialReader.serialPort.close();
    //Mostrar mensaje en barra de estado
    tncM.this.textField.setText("procesos terminados");
    }
});
btnDetener.setFont(new Font("Dialog", Font.BOLD, 10));
btnDetener.setBounds(330, 152, 80, 25);
contentPane.add(btnDetener);

JButton btnGuardar = new JButton("Guardar");
btnGuardar.setFont(new Font("Dialog", Font.BOLD, 9));
btnGuardar.setBounds(17, 197, 78, 25);
contentPane.add(btnGuardar);

JButton btnLimpiar = new JButton("Limpiar");
btnLimpiar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        textArea_1.setText("\tTerminal de datos recibidos TNC-
M:\n");
    }
});
btnLimpiar.setFont(new Font("Dialog", Font.BOLD, 9));
btnLimpiar.setBounds(107, 197, 80, 25);
contentPane.add(btnLimpiar);

JLabel lblDataPort = new JLabel("Data port:");
lblDataPort.setBounds(17, 109, 75, 15);
contentPane.add(lblDataPort);

txtDataPort = new JTextField();
txtDataPort.setText("ttyUSB1");
txtDataPort.setColumns(10);
txtDataPort.setBounds(90, 107, 61, 19);
contentPane.add(txtDataPort);

txtDataRate = new JTextField();
txtDataRate.setText("19200");
txtDataRate.setColumns(10);
txtDataRate.setBounds(90, 127, 61, 19);
contentPane.add(txtDataRate);

JLabel lblDataRate = new JLabel("Data rate:");
lblDataRate.setBounds(17, 129, 75, 15);
contentPane.add(lblDataRate);

JButton btnEditarModos = new JButton("Editar modos");
btnEditarModos.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String cmd = "gedit /home/aet/Desktop/tncM/Srcs/modos.txt
&";

```

```

// create a process for the shell
ProcessBuilder pb = new ProcessBuilder("bash", "-c", cmd);
pb.redirectErrorStream(true); // use this to capture
messages sent to stderr

try{
Process shell = pb.start();
InputStream shellIn = shell.getInputStream(); // this captures
the output from the command

int shellExitStatus = shell.waitFor(); // wait for the shell to
finish and get the return code

//System.out.println(shellExitStatus);
// at this point you can process the output issued by the
command

// for instance, this reads the output and writes it to
System.out:

int c;
while ((c = shellIn.read()) != -1) {System.out.write(c);}
// close the stream
try {shellIn.close();} catch (IOException ignoreMe) {}
}catch (InterruptedException ev) {
throw new RuntimeException(ev);
}
}
catch (IOException ev) {
throw new RuntimeException(ev);
}
});
btnEditarModos.setFont(new Font("Dialog", Font.BOLD, 10));
btnEditarModos.setBounds(309, 109, 120, 22);
contentPane.add(btnEditarModos);

JButton btnRecibirDatos = new JButton("Recibir datos");
btnRecibirDatos.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent arg0) {
try {
(new ReadSerialPort()).connect("/dev/ttyUSB1");
} catch (Exception e) {
e.printStackTrace();
}
}
});
btnRecibirDatos.setFont(new Font("Dialog", Font.BOLD, 10));
btnRecibirDatos.setBounds(17, 152, 130, 25);
contentPane.add(btnRecibirDatos);

JButton btnEditarEnvo = new JButton("Editar envío");
btnEditarEnvo.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
String cmd = "gedit
/home/aet/Desktop/tncM/Srcs/datos_a_enviar.txt &";
// create a process for the shell
ProcessBuilder pb = new ProcessBuilder("bash", "-c", cmd);
pb.redirectErrorStream(true); // use this to capture
messages sent to stderr

try{
Process shell = pb.start();
InputStream shellIn = shell.getInputStream(); // this captures

```

the output from the command

finish and get the return code

command

System.out:

```

int shellExitStatus = shell.waitFor(); // wait for the shell to
//System.out.println(shellExitStatus);
// at this point you can process the output issued by the
// for instance, this reads the output and writes it to
int c;
while ((c = shellIn.read()) != -1) {System.out.write(c);}
// close the stream
try {shellIn.close();} catch (IOException ignoreMe) {}
} catch (InterruptedException ev) {
throw new RuntimeException(ev);
}
catch (IOException ev) {
throw new RuntimeException(ev);
}
});
btnEditarEnvo.setFont(new Font("Dialog", Font.BOLD, 9));
btnEditarEnvo.setBounds(217, 197, 100, 25);
contentPane.add(btnEditarEnvo);

JButton btnEnviarDatos = new JButton("Enviar datos");
btnEnviarDatos.setFont(new Font("Dialog", Font.BOLD, 9));
btnEnviarDatos.setBounds(329, 197, 100, 25);
contentPane.add(btnEnviarDatos);

textField = new JTextField();
textField.setHorizontalAlignment(SwingConstants.CENTER);
textField.setEditable(false);
textField.setText("...");
textField.setColumns(10);
textField.setBounds(165, 346, 179, 25);
contentPane.add(textField);

JLabel lblEstado = new JLabel("Estado:");
lblEstado.setFont(new Font("Dialog", Font.BOLD, 11));
lblEstado.setBounds(107, 352, 55, 15);
contentPane.add(lblEstado);

textRX = new JTextField();
textRX.setFont(new Font("Dialog", Font.PLAIN, 12));
textRX.setBackground(Color.WHITE);
textRX.setHorizontalAlignment(SwingConstants.CENTER);
textRX.setEditable(false);
textRX.setColumns(10);
textRX.setBounds(17, 312, 412, 22);
contentPane.add(textRX);

JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(17, 234, 412, 74);
contentPane.add(scrollPane);

//JTextArea textArea_1 = new JTextArea("\tTerminal de datos recibidos
TNC-M:\n",5,40);

```

```

        textArea_1 = new JTextArea();
        scrollPane.setViewportView(textArea_1);
        textArea_1.setEditable(false);
        textArea_1.setLineWrap(true);
        textArea_1.setWrapStyleWord(true);
        textArea_1.append("\tTerminal de datos recibidos TNC-M:\n");
    }
}

```

*back.c

```

/*
 * back.c
 *
 * Created on: Mar 10, 2012
 * Author: Dante Inga
 * Based on:
 * -predict client (http://www.qsl.net/kd2bd/predict.html)
 * -tcp client (http://www.tenouk.com/Module41a.html)
 * -lnx comm (http://lnxcomm.sourceforge.net/)
 */

#define __LINUX_COM__
// #define ENABLE_SERIAL_PORT_EVENT
#include "com/serial.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <errno.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/signal.h>

typedef enum {
    stateSearch = 0, stateTrack
} stateType;

// Actualizado al 12-08-2012
unsigned long freqDown[24] = { 437505000, 437425000, 437275000, 437345000,
    145825000, 437305000, 435790000, 437325000, 436000000, 437505000,
    437250000, 437385000, 435315000, 437485000, 437275000, 436465000,
    145860000, 435352000, 436847500, 436837500, 145825000, 437025000,
    437025000, 437025000 };

// 0: ASK-Morse, 1: FSK-AX.25, 2: MD3, 3: MD4

```

```

unsigned int satMode[24] = {
    1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0
};

char string[625];

void handler() {
    /* This is a function that is called when the response function
       times out. This is in case the server fails to respond. */

    signal(SIGALRM, handler);
}

int connectsock(char *host, char *service, char *protocol) {
    /* This function is used to connect to the server. "host" is the
       name of the computer on which PREDICT is running in server mode.
       "service" is the name of the socket port. "protocol" is the
       socket protocol. It should be set to UDP. */

    struct hostent *phe;
    struct servent *pse;
    struct protoent *ppe;
    struct sockaddr_in sin;

    int s, type;

    bzero((char *) &sin, sizeof(struct sockaddr_in));
    sin.sin_family = AF_INET;

    if ((pse = getservbyname(service, protocol)))
        sin.sin_port = pse->s_port;

    else if ((sin.sin_port = htons((unsigned short) atoi(service))) == 0) {
        printf("Can't get services\n");
        return -1;
    }

    if ((phe = gethostbyname(host))
        bcopy(phe->h_addr, (char *) &sin.sin_addr, phe->h_length);

    else if ((sin.sin_addr.s_addr = inet_addr(host)) == INADDR_NONE) {
        printf("Can't get host: \"%s\".\n", host);
        return -1;
    }

    if ((ppe = getprotobyname(protocol)) == 0)
        return -1;

    if (strcmp(protocol, "udp") == 0)
        type = SOCK_DGRAM;
    else
        type = SOCK_STREAM;

    s = socket(PF_INET, type, ppe->p_proto);

    if (s < 0) {
        printf("Can't get socket.\n");
    }
}

```



```

        return -1;
    }

    if (connect(s, (struct sockaddr *) &sin, sizeof(sin)) < 0) {
        printf("Can't connect to socket.\n");
        return -1;
    }

    return s;
}

void get_response(int sock, char *buf) {
    /* This function gets a response from the
       server in the form of a character string. */

    int n;

    n = read(sock, buf, 625);

    if (n < 0) {
        if (errno == EINTR)
            return;

        if (errno == ECONNREFUSED) {
            fprintf(stderr, "Connection refused - PREDICT server not running\n");
            exit(1);
        }
    }

    buf[n] = '\0';
}

char *send_command(host, command)
char *host, *command; {
    int sk;

    /* This function sends "command" to PREDICT running on
       machine "host", and returns the result of the command
       as a pointer to a character string. */

    /* Open a network socket */
    sk = connectsock(host, "predict", "udp");

    if (sk < 0)
        exit(-1);

    /* Build a command buffer */
    sprintf(string, "%s\n", command);

    /* Send the command to the server */
    write(sk, command, strlen(string));

    /* clear string[] so it can be re-used for the response */
    string[0] = 0;

    /* Get the response */
    get_response(sk, string);
}

```

```

        /* Close the connection */
        close(sk);

        return string;
    }

void send_command2(host, command)
    char *host, *command; {
    int sk;

    /* This function sends "command" to PREDICT running on
       machine "host", and displays the result of the command
       on the screen as text. It reads the data sent back
       from PREDICT until an EOF marker (^Z) is received. */

    /* Open a network socket */
    sk = connectsock(host, "predict", "udp");

    if (sk < 0)
        exit(-1);

    /* Build a command buffer */
    sprintf(string, "%s\n", command);

    /* Send the command to the server */
    write(sk, command, strlen(string));

    /* clear string[] so it can be re-used for the response */
    string[0] = 0;

    /* Read and display the response until a ^Z is received */
    get_response(sk, string);

    while (string[0] != 26) /* Control Z */
    {
        printf("%s", string);
        string[0] = 0;
        get_response(sk, string);
    }

    printf("\n");

    /* Close the connection */
    close(sk);
}

int main() {
    int x, y, z, satnum;
    char buf[128], command[128], satlist[625], visibility,
          satnamelist[26][26], event[15], squint_string[15];
    char satname[26];
    long aostime, orbitnum;
    //long start;
    float az, el, slong, slat, footprint, range, altitude, velocity, phase,
          eclipse_depth, squint;
    float elevacion[24];

```

```

float menor_el, menor_el_buf;
int objetivo;
long freq;
char mod[3];
long tiempo[24]; //Fecha absoluta de siguiente paso de satellite
long menor_aos;
int temp;
static stateType estado = stateSearch;
time_t t;

/*****TNC-M Control Port*****/
HANDLE pid;
pid = Open_Port("/dev/ttyUSB0");
DCB conf_bkp;
conf_bkp = Get_Configure_Port(pid);
DCB conf;
conf = Configure_Port(pid, B9600, "8N1"); //Usar #define
long nbyte_sent, leng;
char info[64];
int ack;
/*****TNC-M Control Port*****/

/*****Loading Satellite Mode*****/
FILE *fp;
char file_name[12];
unsigned long file_freq;
int file_mode, i = 0;
/* open the file */
fp = fopen("/home/aet/Desktop/tncM/Srcs/modos.txt", "r");
if(fp == NULL){
    printf("No es posible abrir el archivo 'modos.txt'");
    exit(0);
}
/* read the file */
while (fscanf(fp, "%s %lu %d\n", &file_name, &file_freq, &file_mode) == 3){
    //printf("%d: %lu %d\n", i+1, file_freq, file_mode);
    freqDown[i] = file_freq;
    satMode[i] = file_mode;
    i++;
}
/* close the file */
fclose(fp);
/*****Loading Satellite Mode*****/

/****rigctl and rotctl variables definition****/
int sock2, bytes_recieved2;
char send_data2[1024], recv_data2[1024];
struct hostent *host2;
struct sockaddr_in server_addr2;
host2 = gethostbyname("127.0.0.1");

if ((sock2 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Socket");
    exit(1);
}

/*rotctl tcp port = 4533*/

```

```

int sock3, bytes_recieved3;
char send_data3[1024], recv_data3[1024];
struct hostent *host3;
struct sockaddr_in server_addr3;
host3 = gethostbyname("127.0.0.1");

if ((sock3 = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("Socket");
    exit(1);
}
/*rotctld tcp port = 4533*/
/****rigctld and rotctld variables definition****/

/****rigctld and rotctld tcp sockets opening****/
server_addr2.sin_family = AF_INET;
server_addr2.sin_port = htons(4532); //rigctld tcp port
server_addr2.sin_addr = *((struct in_addr *) host2->h_addr);
bzero(&(server_addr2.sin_zero), 8);

if (connect(sock2, (struct sockaddr *) &server_addr2,
            sizeof(struct sockaddr)) == -1) {
    perror("Connect");
    exit(1);
}

/*rotctld tcp port = 4533*/
server_addr3.sin_family = AF_INET;
//server_addr.sin_port = htons(4532); //rigctld tcp port
server_addr3.sin_port = htons(4533); //rotctld tcp port
//server_addr.sin_port = htons(5000);
server_addr3.sin_addr = *((struct in_addr *) host2->h_addr);
bzero(&(server_addr3.sin_zero), 8);

if (connect(sock3, (struct sockaddr *) &server_addr3,
            sizeof(struct sockaddr)) == -1) {
    perror("Connect");
    exit(1);
}
/*rotctld tcp port = 4533*/
/****rigctld and rotctld tcp sockets opening****/

/* Get the list of satellite names from PREDICT */
strcpy(satlist, send_command("localhost", "GET_LIST"));

/* Parse the response and place each name
in the character array satnamelist[[]]. */

for (x = 0, y = 0, z = 0; y < strlen(satlist); y++) {
    if (satlist[y] != '\n') {
        satnamelist[z][x] = satlist[y];
        x++;
    }
    else {
        satnamelist[z][x] = 0;
        z++;
        x = 0;
    }
}

```

```

    }
}

satnum = z;
while (1) {
    switch (estado) {

        case stateSearch:
            for (z = 0; z < satnum; z++) {
                sprintf(command, "GET_SAT %s", satnamelist[z]);
                strcpy(buf, send_command("localhost", command));
                //printf("The following string was returned in response to
%s:\n%s\n",command,buf);

                /* Parse the response from GET_SAT */

                /* The first element of the response is the satellite name.
                It is ended with a '\n' character and many contain spaces. */

                for (x = 0; buf[x] != '\n'; x++)
                    satname[x] = buf[x];

                satname[x] = 0;
                x++;

                /* The rest of the data from GET_SAT is numerical, and
                can be parsed using the sscanf() function. First, the
                satellite name is removed from "buf", and then "buf"
                is parsed for numerical data using an sscanf(). */

                for (y = 0; buf[x + y] != 0; y++)
                    buf[y] = buf[x + y];

                buf[y] = 0;

                sscanf(buf, "%f %f %f %f %ld %f %f %f %f %ld %c %f %f
%f",
                    &slong, &slat, &az, &el, &aostime,
                    &altitude, &velocity, &orbitnum, &visibility,
                    &eclipse_depth, &sqint);

                t = (time_t) aostime;

                if (el > 0.0)
                    strcpy(event, "LOS at");

                else
                    strcpy(event, "Next AOS at");

                if (sqint == 360.0)
                    sprintf(sqint_string, "N/A");
                else
                    sprintf(sqint_string, "%.2f degrees", sqint);

                elevacion[z] = el;
            }
        }
    }
}

```

```

        tiempo[z] = aostime;
    }

    /* Calculo del menor angulo de elevacion */
    menor_el = elevacion[0];
    objetivo = 0;
    for (z = 0; z < satnum; z++) {
        if (elevacion[z] > 0) {
            menor_el = elevacion[z];
            objetivo = z + 1;
            break;
        } else {
            if (elevacion[z] > menor_el) {
                menor_el = elevacion[z];
                objetivo = z + 1;
            }
        }
    }
    if (objetivo > 0) {
        for (z = objetivo; z < satnum; z++) {
            if (elevacion[z] > 0 && elevacion[z] < menor_el) {
                menor_el = elevacion[z];
                objetivo = z + 1;
            }
        }
    }
}

/*Impresion de los resultados del calculo*/
if (menor_el > 0) {
    sprintf(command, "GET_DOPPLER %s",
satnamelist[objetivo - 1]);

    strcpy(buf, send_command("localhost", command));
    estado = stateTrack;
} else {
    /* Calculo del tiempo de contacto mas cercano */
    satnum = 21; //No deberia ser necesario?
    menor_aos = tiempo[0];
    temp = 0;
    for (z = 0; z < satnum; z++) {
        if (tiempo[z] < menor_aos) {
            menor_aos = tiempo[z];
            temp = z;
        }
    }
    t = (time_t) menor_aos;
    //printf(":%s %s", satnamelist[temp], asctime(gmtime(&t)));
    leng = sprintf(info, "\r:%s %s", satnamelist[temp],
asctime(gmtime(&t)));

    nbyte_sent = Write_Port(pid, info, leng-6);
    usleep(1000000); //Delay = 1.0seg
    menor_el_buf = menor_el;
    estado = stateSearch;
}
break;

case stateTrack:
    z = objetivo - 1;

```

```

sprintf(command, "GET_SAT %s", satnamelist[z]);
strcpy(buf, send_command("localhost", command));

/* Parse the response from GET_SAT */

/* The first element of the response is the satellite name.
   It is ended with a '\n' character and many contain spaces. */

for (x = 0; buf[x] != '\n'; x++)
    satname[x] = buf[x];

satname[x] = 0;
x++;

/* The rest of the data from GET_SAT is numerical, and
   can be parsed using the sscanf() function. First, the
   satellite name is removed from "buf", and then "buf"
   is parsed for numerical data using an sscanf(). */

for (y = 0; buf[x + y] != 0; y++)
    buf[y] = buf[x + y];

buf[y] = 0;

sscanf(buf, "%f %f %f %f %ld %f %f %f %f %ld %c %f %f %f",
&slong,
                                &slat, &az, &el, &aostime, &footprint, &range,
&altitude,
                                &velocity, &orbitnum, &visibility, &phase,
&eclipse_depth,
                                &squint);

t = (time_t) aostime;

if (el > 0.0)
    strcpy(event, "LOS at");

else
    strcpy(event, "Next AOS at");

if (squint == 360.0)
    sprintf(squint_string, "N/A");
else
    sprintf(squint_string, "%.2f degrees", squint);

elevacion[z] = el;

sprintf(command, "GET_DOPPLER %s", satnamelist[objetivo - 1]);
strcpy(buf, send_command("localhost", command));
freq = freqDown[objetivo - 1] + (long) strtol(buf, NULL, 10);
if (satMode[objetivo-1]==0) sprintf(mod,"ASK"); else
sprintf(mod,"FSK");
//printf(=") %s AZ:%i EL:%i FR:%lu MO:%s\n", satnamelist[objetivo -
1], (int)az, (int)el, freq, mod);
leng = sprintf(info, "\r=) %-9s %-3i%-2i %lu MD%d",
satnamelist[objetivo - 1], (int)az, (int)el, freq, satMode[objetivo-1]+1);
nbyte_sent = Write_Port(pid, info, leng);

```

```

/****rigctld and rotctld control actions*****/
sprintf(send_data2, "F %lu\n", freq);
//printf("Port4532:~# %s", send_data2);
send(sock2, send_data2, strlen(send_data2), 0);

bytes_recieved2 = recv(sock2, recv_data2, 1024, 0);
recv_data2[bytes_recieved2] = '\0';

if (strcmp(recv_data2, "") == 0) //RPRT -1
{
    close(sock2);
    //break;
} else
    //printf("port4532:~# %s", recv_data2);

/*rotctld tcp port = 4533*/
sprintf(send_data3, "P %f %f\n", az, el);
//printf("Port4533:~# %s", send_data3);
send(sock3, send_data3, strlen(send_data3), 0);

bytes_recieved3 = recv(sock3, recv_data3, 1024, 0);
recv_data3[bytes_recieved3] = '\0';

if (strcmp(recv_data3, "") == 0) //error(RPRT -1)
{
    close(sock3);
} else{
    //printf("port4533:~# %s", recv_data3);
}
/*rotctld tcp port = 4533*/
usleep(1000000); //Delay = 1.0seg
/****rigctld and rotctld control actions*****/

if (el > 0)
    estado = stateTrack;
else{
    estado = stateSearch;
}
break;

    } //Fin de switch
} //Fin de while

exit(0);
}

```


ANEXO G
RESULTADOS DERIVADOS

Publicación en IEEE:

The screenshot shows the IEEE Xplore Digital Library search results page. The search criteria are "Authors": "Inga, Dante". One result is returned: "Surrogate modeling with Genetic Programming applied to satellite communication and ground stations" by Rodriguez, Glen D.; Velasquez, Ivan; Cachi, Dane; Inga, Dante. The paper is from the Aerospace Conference, 2012 IEEE, with a Digital Object Identifier of 10.1109/AERO.2012.6187326. The abstract discusses the use of surrogate models in satellite missions, mentioning complex factors like orbital calculation and Doppler shift correction, and the use of Genetic Programming as a simpler alternative to neural networks and support vector machines.

Publicación en ECI:

The screenshot shows the ECI publication page for the paper "Implementación de un Sistema de Seguimiento Automático de Satélites Pequeños para la recepción e Interpretación de Señales Beacon desde la Estación Terrena del CTIC-UNI". The authors are Dante Inga, Iván Velásquez and Dane Cachi. The paper is published by Centro de Tecnologías de Información y Comunicaciones - CTIC UNI, Universidad Nacional de Ingeniería (UNI), Av. Túpac Amaru 210 - Rímac / Lima 25 - Perú. The abstract describes the development of a small satellite tracking system for receiving and interpreting beacon signals from CTIC-UNI's ground station. It mentions the use of hardware like radio, antenna, rotor, and TNC, and software for remote management of equipment and data received. The system is designed for small satellites in low orbit (200-800 km) moving at high speeds relative to the ground.

ANEXO H
LISTA DE FIGURAS, ABREVIATURAS

LISTA DE FIGURAS

Figura 3.1: Satélite artificial y estación terrena.....	5
Figura 3.2: Elementos básicos de una estación terrena.....	6
Figura 3.3: TNC hardware KAM-XL	8
Figura 3.4: TNC software MixW	10
Figura 3.5: Capas del modelo TCP/IP.....	11
Figura 3.6: Capas del modelo X.25.....	12
Figura 3.7: Campos de una trama LAPB	12
Figura 3.8: Contenido de un paquete X.25.....	13
Figura 3.9: Señal modulada en ASK.....	15
Figura 3.10: Señal modulada en FSK	16
Figura 3.11: Señal modulada en PSK.....	16
Figura 3.12: Modulación y demodulación GMSK	17
Figura 3.13: Espectro de una señal modulada en GMSK.....	18
Figura 3.14: Trama AX.25 no numerada usada para APRS.....	19
Figura 3.15: Representación geométrica del efecto Doppler.....	20
Figura 3.16: Representación vectorial del efecto Doppler	22
Figura 3.17: Efecto Doppler en un satélite de órbita circular	22
Figura 3.18: Gráfico teórico del desplazamiento de frecuencia Doppler.....	23
Figura 3.19: Gráficas teórica y experimental del desplazamiento Doppler	24
Figura 3.20: “Pasadas” de la ISS muestreadas con Orbitron	24
Figura 3.21: Captura de pantalla del software Eureka	25
Figura 3.22: Comparación entre datos estimados y experimentales	26
Figura 3.23: Comparación entre datos estimados y experimentales	26
Figura 3.24: Comparación entre datos estimados y experimentales	27
Figura 3.25: Comparación entre datos estimados y experimentales	27
Figura 3.26: Ángulos de azimuth y elevación.....	28
Figura 4.1: Satélite 1 comunicándose con 2 estaciones terrenas en red.....	30
Figura 4.2: Satélite 2 comunicándose después de alejarse el satélite 1.....	30
Figura 5.1: Modulaciones usadas en pequeños satélites (Octubre de 2011)	34
Figura 5.2: Diagrama de bloques del HW/SW del TNC propuesto	35

Figura 5.3: Diagrama de Bloques del HW del TNC multimodulación.....	35
Figura 5.4: Multiplexor/demultiplexor analógico CD4052	36
Figura 5.5: Demultiplexor digital 74139 y multiplexor digital 74153	36
Figura 5.6: Demodulador ASK basado en el CI XR2211	38
Figura 5.7: Demodulador FSK basado en el CI XR2211	39
Figura 5.8: Recursos del microcontrolador PIC18F4550P	41
Figura 5.9: Recursos del microcontrolador PIC16F877A	42
Figura 5.10: Diagrama lógico del convertidor de nivel de voltaje MAX232	43
Figura 5.11: Esquema del primer prototipo de modulador ASK/FSK.....	44
Figura 5.12: Construcción del primer prototipo de modulador ASK/FSK.....	44
Figura 5.13: Esquema del primer prototipo de demodulador ASK/FSK.....	45
Figura 5.14: Construcción del primer prototipo de demodulador ASK/FSK	45
Figura 5.15: Tarjeta principal del TNC propuesto.....	46
Figura 5.16: Diagrama de flujo del firmware para la unidad de control.....	47
Figura 5.18: Diagrama de flujo del software de control para la PC.....	50
Figura 5.19: Esquema físico de la prueba del mecanismo de selección	51
Figura 6.1: Escenario de prueba para las simulaciones	52
Figura 6.2: Edición de una estación terrena con software Predict	54
Figura 6.3: Simulación de parámetros de satélites con Predict	55
Figura 6.4: Esquema de la prueba de acceso a Predict	58
Figura 6.5: Prueba de solicitud de lista de satélites usando Eclipse	59
Figura 6.6: Esquema de prueba de envío de comandos a Hamlib	60
Figura 6.7: Prueba de envío de comandos a Hamlib usando xdx	61
Figura 6.8: Prueba de envío de comandos a Hamlib usando Eclipse.....	62
Figura 6.9: Prueba del programa de seguimiento de satélites	63
Figura 6.10: Interfaz gráfica con terminal serial	63
Figura 7.1: TNC multimodulación y sus componentes de hardware.....	65
Figura 7.2: Tarjeta módem ASK/FSK y especificación de pines.....	66
Figura 7.3: Demodulación ASK de señal ideal	67
Figura 7.4: Demodulación ASK de señal real.....	68
Figura 7.5: Variación del tiempo de bit en función del resistor de timing	69
Figura 7.6: Variación del tiempo de bit en función de la alimentación	69
Figura 7.7: Señal ASK ideal en el dominio del tiempo y frecuencia.....	70
Figura 7.8: Medición de tiempos de señal ASK ideal demodulada.....	71

Figura 7.9: Entorno de pruebas de la tarjeta módem FSK/ASK	71
Figura 7.10: Generación de señal modulada en FSK.....	72
Figura 7.11: Demodulación de señal FSK ideal	73
Figura 7.12: Demodulación de señal FSK real.....	73
Figura 7.13: Variación del error con la tasa de bits en módem FSK.....	74
Figura 7.14: Interpretación de señal FSK real con UI-VIEW	75
Figura 7.15: Escenario 1 de pruebas del TNC multimodulación.....	76
Figura 7.16: Señal ASK de CO-55 recibida sin el sistema de seguimiento	77
Figura 7.17: Señal ASK de RS-30 recibida con el sistema de seguimiento.....	78
Figura 7.18: Recepción ASK de CO-57 con el sistema de seguimiento	79
Figura 7.19: Recepción FSK de FASTRAC con el sistema de seguimiento	79
Figura 7.20: Recepción voz de RadioSkaf2 con el sistema de seguimiento.....	79
Figura 7.21: Escenario 2 de pruebas del TNC multimodulación.....	80
Figura 7.22: Prueba de algoritmo de control del TNC multimodulación.....	81
Figura 7.23: Prueba de decodificación automática de dos modulaciones	81
Figura 7.24: Cronograma de tiempos estimados.....	85
Figura A.1: Letras del código Morse	89
Figura A.2: Cifras del código Morse	89
Figura A.3: Signos del código Morse	90
Figura B.1: Capas del modelo AX.25.....	93
Figura B.2: Estructura de los 2 tipos de trama AX.25.....	94
Figura B.3: Estructura del campo SSID	95
Figura B.4: Estructura del campo PID.....	97
Figura C.1: Diagrama esquemático de la tarjeta principal	99
Figura C.2: Diagrama esquemático de las tarjetas módem.....	100
Figura C.3: Diagrama esquemático de la tarjeta de alimentación	101

ABREVIATURAS

- ADC : Analog to Digital Converter (Convertidor analógico digital).
- AFSK : Audio Frequency Shift Keying (Modulación por desplazamiento de frecuencia de audio).
- AMSAT : Amateur Satellites (Satélites de radioaficionados).
- AMTOR : Amateur Teleprinting Over Radio (Impresión a distancia por radio).
- APRS : Automatic Position Reporting System (Sistema de reporte automático de posición).
- ASCII : American Standard Code for Information Interchange (Código americano estándar para intercambio de información).
- ASK : Amplitude Shift Keying (Modulación por desplazamiento de amplitud).
- AX.25 : Amateur X.25 (Protocolo X.25 para radioaficionados).
- bps : Bits per second (Bits por segundo).
- CI-V : CAT Interface verssion 5 (Interfaz CAT versión 5).
- CW : Continuous wave (Onda continúa).
- DCE : Data Communication Equipment (Equipo de comunicación de datos).
- DE-9 : D-subminiature with 9-pins (Conector serial de 9 pines).
- DIN : Deutsches Institut für Normung (Organización de estandarización nacional de Alemania).
- DSP : Digital Signal Processor (Procesador digital de señales).
- DTE : Data Terminal Equipment (Equipo terminal de datos).
- EPRM : Electrically Programmable Read Only Memory (Memoria de solo lectura programable eléctricamente).
- FCS : Frame Check Sum (Suma de verificación de trama).
- FM : Frequency Modulation (Modulación en frecuencia).
- FPGA : Field Programmable Gate Array (Arreglo de compuertas programables por campo).
- FRAM : Ferromagnetic Random Access Memory (Memoria de acceso aleatorio ferromagnética).
- FSK : Frequency Shift Keying (Modulación por desplazamiento de frecuencia).
- GEO : Geostationary Earth Orbit (Órbita geo-estacionaria).

<i>GMSK</i>	<i>: Gaussian Minimum Shift Keying (Modulación por desplazamiento mínimo gaussiano).</i>
<i>GPS</i>	<i>: Global Positioning System (Sistema de posición global).</i>
<i>HDLC</i>	<i>: High Level Data Link Control (Control de enlace de datos de alto nivel).</i>
<i>HEO</i>	<i>: High Earth Orbit (Órbita altamente elíptica).</i>
<i>HF</i>	<i>: High Frequency (Frecuencia alta).</i>
<i>Hz</i>	<i>: Hertz (Hertzios).</i>
<i>IP</i>	<i>: Internet Protocol (Protocolo de Internet).</i>
<i>ISS</i>	<i>: International Space Station (Estación espacial internacional).</i>
<i>KISS</i>	<i>: Keep It Simple Stupid (Protocolo de comunicación entre PC y TNC).</i>
<i>LAPB</i>	<i>: Link Access Procedure, Balanced (Procedimiento de acceso de enlace, balanceado).</i>
<i>LCD</i>	<i>: Liquid Crystal Display (Pantalla de cristal líquido).</i>
<i>LEO</i>	<i>: Low Earth Orbit (Órbita terrestre baja).</i>
<i>OFDM</i>	<i>: Orthogonal Frequency Division Multiplexing (Multiplexación por división de frecuencias ortogonales).</i>
<i>OSCAR</i>	<i>: Orbiting Satellite Carrying Amateur Radio (Satélite en órbita transportando un radio de aficionados).</i>
<i>OSI</i>	<i>: Open System Interconnection (Interconexión de sistemas abierto).</i>
<i>PC</i>	<i>: Personal Computer (Computador personal).</i>
<i>PIC</i>	<i>: Peripheral Interface Controller (Familia de microcontroladores).</i>
<i>PLL</i>	<i>: Phase-Locked Loop (Lazo de seguimiento de fase).</i>
<i>PM</i>	<i>: Phase Modulation (Modulación de fase).</i>
<i>PSK</i>	<i>: Phase Shift Keying (Modulación por desplazamiento de fase).</i>
<i>RF</i>	<i>: Radio Frequency (Frecuencia de radio).</i>
<i>RTTY</i>	<i>: Radioteletype (Telégrafo por radio).</i>
<i>RS-232</i>	<i>: Recommended Estándar 232 (Estándar recomendado 232).</i>
<i>SABM</i>	<i>: Set Asynchronous Balanced Mode (Inicializa modo balanceado asíncrono).</i>
<i>SRAM</i>	<i>: Static Random Access Memory (Memoria estática de acceso aleatorio).</i>
<i>SRL</i>	<i>: Simple Radio Link Layer (Nivel simple de enlace por radio).</i>
<i>SSB</i>	<i>: Single-Side Band Modulation (Modulación de banda lateral única).</i>

<i>SSD</i>	<i>: Solid-State Drive (Unidad de estado sólido).</i>
<i>TAPR</i>	<i>: Tucson Amateur Packet Radio (Organización educacional de investigación y desarrollo).</i>
<i>TCP</i>	<i>: Transmission Control Protocol (Protocolo de control de transmisión).</i>
<i>TLE</i>	<i>: Two-Line Elements (Parámetros de órbita de satélites).</i>
<i>TLM</i>	<i>: Telemetry (Telemetría – medición a distancia).</i>
<i>TNC</i>	<i>: Terminal Node Controller (Controlador de nodo terminal).</i>
<i>TTL</i>	<i>: Transistor-Transistor Logic.</i>
<i>UART</i>	<i>: Universal Asynchronous Receiver-Transmitter (Transmisor-receptor asíncrono universal).</i>
<i>UHF</i>	<i>: Ultra High Frequency (Frecuencia ultra alta).</i>
<i>UI</i>	<i>: Unnumbered Information frame (Trama de información no numerada).</i>
<i>USB</i>	<i>: Universal Serial Bus (Bus serial universal).</i>
<i>VDC</i>	<i>: Volts of Direct Current (Voltios de corriente directa).</i>
<i>VHF</i>	<i>: Very High Frequency (Frecuencia muy alta).</i>
<i>wps</i>	<i>: Words per second (Palabras por segundo).</i>

BIBLIOGRAFIA

- [1] The CubeSat Program, "CubeSat Design Specifications (CDS)", California Polytechnic State University, 2009. Disponible en:
http://www.cubesat.org/images/developers/cds_rev12.pdf
- [2] AMSAT, "Amateur Satellite Status". Disponible en:
<http://www.amsat.org/amsat-new/satellites/status.php>
- [3] Bruce Elber, "Introduction to Satellite Communication", 2008.
- [4] TAPR, "Terminal Node Controller TNC2", Tucson Amateur Packet Radio, 1992.
- [5] Kantronics, "KAM XL: Getting Started and Reference Manuals", Kantronics, 2001.
- [6] Manual "PK-232 Data Controller", Advanced Electronic Applications, 2004.
- [7] Nick Fedoseev, Denis Nechitailov, "MixW: Overview", 2006.
- [8] Bruce Elber, "The Satellite Communication, Ground Segment and Earth Station", 2001.
- [9] Fuqin Xiong, "Digital Modulation Techniques", 2006.
- [10] Recomendación UIT-R M.1677, "Código Morse Internacional", 2004.
- [11] JL Barber, "Proposal for SSTV Specifications", 2000.
- [12] TAPR, "AX.25 Link Access Protocol for Amateur Packet Radio", Tucson Amateur Packet Radio Ver 2.2, 1998. Disponible en:
<http://www.tapr.org/pdf/AX.25.2.2.pdf>
- [13] Mike Chepponis, Phil Karn, "The KISS TNC: A simple Host-to-TNC communications protocol", 1997. Disponible en:
<http://www.AX.25.net/kiss.aspx>
- [14] The APRS Working Group, "Automatic Position Reporting System, protocol reference", 2000. Disponible en: <http://www.aprs.org/doc/APRS101.PDF>
- [15] Emily Clark, "An Introduction to Amateur Satellites", AMSAT, 2004.
- [16] Roland Best, "Phase-Locked Loops", Fourth Edition.
- [17] John Hansen, "TNC-X, An Expandable Microcontroller-Based Terminal Node Controller", 2003.
- [18] Sergio Franco, "Design with Operational Amplifiers and Analog Integrated Circuits", 2002.

- [19] Space Track Report No. 3, "Models for Propagation of NORAD Elements", 1988.
- [20] Phillip Seabe, "Evaluation of Microcontroller Based Packet Radio Modem", 2007.