VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

DISEÑO Y DESARROLLO DE SISTEMAS DE COMUNICACIÓN DE DATOS PARA ESTACIONES REMOTAS

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRA EN INGENIERÍA

ELÉCTRICA - SISTEMAS ELECTRÓNICOS

P R E S E N T A :

ARCELIA BERNAL DÍAZ

DIRECTOR DE TESIS:

M. en I. LAURO SANTIAGO CRUZ

2006

JURADO ASIGNADO:

Presidente:	DR. ESAU VICENTE VIVAS
Secretario:	M. EN I. LUIS ARTURO HARO RUÍZ
Vocal:	M. EN I. LAURO SANTIAGO CRUZ
1er. Suplente:	M. EN I. ROBERTO TOVAR MEDINA
2do. Suplente:	M. EN I. SERGIO QUINTANA THIERRY

Lugar o lugares donde se realizó la tesis:

INSTITUTO DE INGENIERÍA

TUTOR DE TESIS:

M. EN I. LAURO SANTIAGO CRUZ

_____

FIRMA

Dedicatoria

A mis padres Leobardo Bernal Hernández y María de los Ángeles Díaz Gutiérrez, a quienes les debo todo lo que soy. Gracias por sus enseñanzas, confianza, pero sobre todo por su amor.

A mis hermanos Maritza y Joaquín Leobardo, por estar conmigo en las buenas y en las malas. Los quiero mucho.

A José Arturo Hernández Sánchez, por compartir esta gran vida a mi lado.

A mis amigos, que sin ellos no disfrutaría la vida igual.

Agradecimientos

A Dios, por estar siempre conmigo.

A mis padres por ser mi punto de apoyo.

A mi asesor de tesis M. en I. Lauro Santiago Cruz, por instruirme a lo largo de este proyecto y confiar en mí. Gracias.

Al M. en C. Marcelo Pérez Medel, por el apoyo y el énfasis para concluir mi grado.

Al Dr. Daniel Aldama Ávalos, por el ánimo otorgado en el trayecto de mis estudios.

A mis sinodales:
Dr. Esau Vicente Vivas
M. en I. Luis Arturo Haro Ruíz
M. en I. Roberto Tovar Medina
M. en I. Sergio Quintana Thierry

A CONACYT
Por el apoyo otorgado y la confianza depositada en mi desarrollo académico.

Al Instituto de Ingeniería de la UNAM, por el apoyo a lo largo de este proyecto.

A la Universidad Nacional Autónoma de México, por abrirme sus puertas para compartir la infinidad de conocimiento dentro de ella

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

ÍNDICE DE TABLAS

PRÓLOGO

La necesidad de una comunicación a distancia cada vez se vuelve indispensable en los dispositivos encargados de transferir datos; es el caso de las estaciones remotas destinadas a medir y registrar diversos parámetros a distancia; como ejemplo de dichas estaciones tenemos aquellas que contienen instrumentos tales como: termómetro, barómetro, pluviómetro, psicrómetro, piranómetros, anemómetros, radares, sismómetros, acelerómetros, electrocardiogramas, sistemas acústicos de sondeo, etc.

Las estaciones remotas permiten realizar mediciones de diferentes variables (físicas y químicas), además de enviar la información adquirida a estaciones centrales, mediante diversos tipos de comunicación, como lo son: radiofrecuencia, satelital y vía Internet. Los datos adquiridos por las estaciones remotas son muy importantes, ya que son utilizados para la elaboración de predicciones de diversos fenómenos a partir de modelos numéricos.

En la Coordinación de Instrumentación del Instituto de Ingeniería de la UNAM, se desarrolló una estación remota electrónica de medición de niveles de agua, para realizar aforos en ríos (limnígrafo digital). La estación tiene la capacidad de almacenar la información adquirida en memoria no volátil, misma que puede ser enviada a una computadora personal mediante diferentes protocolos de comunicación para el análisis de los datos, destacando entre ellos la comunicación serial vía interfaz RS-232C, SDI-12 e infrarrojo.

Conociendo las necesidades de comunicación de dicha estación remota, se diseñó y construyó un prototipo de comunicaciones de datos alámbrica e inalámbricas para equipar dicha estación. El sistema de comunicaciones desarrollado permite interconectar una estación remota con equipos móviles, con el apoyo de una interfaz amigable para el usuario y segura para el almacenamiento de la información obtenida por la estación remota.

El diseño del sistema de comunicaciones involucra tanto *hardware* como *software,* los cuales interactúan en conjunto para lograr una operación óptima del sistema. Ambas partes fueron desarrolladas en la Coordinación de Instrumentación del Instituto de Ingeniería de la UNAM.

Se pretende con el sistema minimizar los costos de adquisición y soporte técnico, debido al conocimiento que se alcanza sobre ella; además se contribuye de manera significativa el desarrollo de tecnología nacional que puede sustituir a la de importación.

El prototipo de comunicaciones de datos para estaciones remotas está terminado, este ha sido sometido a pruebas de confiabilidad, las cuales resultaron satisfactorias. El presente trabajo escrito muestra el análisis, diseño y desarrollo de cada uno de los módulos de dicho sistema.

Arcelia Bernal Díaz

INTRODUCCIÓN

En la Coordinación de Instrumentación del Instituto de Ingeniería de la Universidad Nacional Autónoma de México se desarrolló un limnímetro electrónico, para realizar aforos en ríos. El dispositivo tiene la capacidad de almacenar información en memoria no volátil, la cual puede ser enviada por medio de tres protocolos de comunicación, dos de ellos mediante un cable de conexión y otro inalámbrico. Los dos primeros se refieren a los estándares RS-232 y SDI-12; en cuanto a la comunicación inalámbrica, es de tipo infrarrojo de bajo nivel.

Con base a lo ya mencionado, la presente tesis comprende el desarrollo de un prototipo de comunicaciones para estaciones remotas, incluyendo *hardware y software*. Consistirá principalmente con medios de comunicación: RS232; USB; MODEM (línea telefónica); sistema inalámbrico de comunicación infrarroja IRDA, de la estación remota a un equipo portátil (LAPTOP y PDA); radiofrecuencia y telefonía celular GSM, de la estación remota a la estación central mediante un protocolo establecido. Los datos adquiridos por el limnímetro serán procesados en una estación central, que contará con el sistema para recepción de datos. Cabe destacar que se tendrán como principales parámetros de desarrollo los siguientes aspectos: tamaño, costo accesible, facilidad de operación y manejo de información eficiente y oportuna.

Este proyecto formará parte del limnímetro antes mencionado, sólo que ahora se equipará con el sistema completo de comunicaciones para conocer los caudales circulantes en cada momento, donde la información a manejar podrá ser en tiempo real.

El trabajo escrito corresponde al prototipo de sistema de comunicaciones de datos para estaciones remotas y está estructurado de la siguiente manera:

Capítulo I, proporciona una introducción de los diferentes tipos de estaciones remotas que existen, para hacer un análisis de los requerimientos de un limnígrafo digital desarrollado por la Coordinación de Instrumentación del Instituto de Ingeniería de la UNAM; se realiza el planteamiento del problema, así como la propuesta de solución al mismo; también se describe la metodología empleada en el prototipo.

Capítulo II, contiene conceptos generales de los microcontroladores, así como la arquitectura empleada en el sistema de comunicaciones de datos; además se describe los estándares de comunicación empleados para las diferentes formas de comunicación.

Capítulo III, habiendo definido los diferentes estándares de comunicación, ahora se explica el diseño y desarrollo de los módulos que integran el sistema de comunicación alámbricas e inalámbricas, tanto *hardware* como *software*, indicando las características operativas de cada uno de ellos.

Capítulo IV, proporciona las pruebas a las que fueron sometidos los diferentes sistemas de transmisión de datos, tanto en el área de *hardware* como *software*, también se presentan los resultados obtenidos junto con las conclusiones a la que se llegó una vez terminado el desarrollo de este trabajo, describiendo las ventajas, desventajas, recomendaciones y expectativas del sistema a futuro para posibles mejoras.

En la sección de bibliografía se listan los diferentes libros consultados y manuales consultados para el desarrollo del trabajo.

Los diversos apéndices proporcionan valioso material de referencia. En el apéndice A se presenta la programación del hardware, mientras que en el apéndice B la programación software del proyecto.

CAPÍTULO I
ANTECEDENTES

En el presente capítulo se presentan algunos sistemas que hacen uso de estaciones remotas en México, destacando el relacionado con la problemática del importante recurso natural que es el agua. También se presenta una descripción somera del limnímetro digital (estación remota) desarrollado por la UNAM, con la finalidad de determinar los requerimientos de dicha estación, planteando las necesidades principales y las soluciones de cada una de ellas. Por último, se describe la metodología a emplear para el desarrollo del prototipo.

I.1. Generalidades

En México se encuentran varios tipos de redes de estaciones remotas, como son: Red de Monitoreo de Partículas PM2.5, Servicio de Meteorología Aeronáutica, Red del Popocatépetl, Diagnóstico a Pacientes y Servicio Meteorológico Nacional. Las características principales de dichas redes son:

- Red de Monitoreo de Partículas $PM_{2.5}$. Esta red está conformada en cuatro zonas en la Ciudad de México, donde se encuentran estaciones automatizadas: Coyoacán (COY), UAM-Iztapalapa (UIZ), Merced (MER), San Juan de Aragón (SJA), Nezahualcóyotl (PER), San Agustín (SAG), Camarones (CAM), Tlalnepantla (TLA).

- Servicio de Meteorología Aeronáutica. Conformado por cincuenta y seis estaciones de observación que están instaladas a lo largo de la República Mexicana. Ahí se realizan observaciones de las condiciones climatológicas que imperan en cada aeropuerto. La información que se genera en estos lugares se concentra en el CAPMA (*Centro de Análisis y Pronósticos Meteorológicos*), que opera en el Aeropuerto Internacional de la Ciudad de México. Las estaciones de observación meteorológica elaboran información que es transmitida mediante el reporte meteorológico horario, y que es utilizado por el CAPMA para la elaboración y edición de los pronósticos meteorológicos de área, ruta y terminal. La transmisión de los datos se difunde a través de la Red AFTN (*Red Fija de Telecomunicaciones Aeronáuticas*).
- Red del Popocatépetl. Consta de 25 estaciones remotas y una estación central de adquisición y procesamiento de datos localizada en las instalaciones del CENAPRED (*Centro Nacional de Prevención de Desastres*).
- Para el diagnóstico de pacientes hay estaciones remotas con comunicación vía satélite en 18 hospitales regionales.
- Servicio Meteorológico Nacional. Organismo encargado de proporcionar información sobre el estado del tiempo a escala local y nacional en nuestro país, depende de la CNA (*Comisión Nacional del Agua*), la cual forma parte de la SEMARNAT (*Secretaría de Medio Ambiente y Recursos Naturales*). Su función primordial es vigilar y emitir información sobre las condiciones atmosféricas del país, así como pronosticar y alertar sobre eventos hidrometeorológicos que puedan ocasionar daños a la población o a las actividades productivas en el territorio nacional. Los sistemas de alerta están basados en un conjunto de estaciones pluviométricas e hidrométricas, ubicadas en las diferentes cuencas hidrológicas en que se dividen las regiones de estudio; miden principalmente la precipitación acumulada, la intensidad de lluvia y los niveles de los cauces; envían la información adquirida por telemetría a una estación central de registro.

Esta última red es de gran importancia, ya que el proyecto está dirigido a la medición de niveles de agua, por lo tanto se abordan algunos aspectos importantes sobre dicho recurso natural.

Los eventos hidrometeorológicos extremos pueden presentarse en dos formas: lluvias intensas (extremo superior de la estadística hidroclimática) y sequías (el extremo inferior de la estadística hidroclimática). En el primer caso, éstas pueden provocar inundaciones, pérdidas humanas y económicas. En el segundo caso, los impactos se ven reflejados directamente en el sector

agrícola, debido a una reducción en el número de días con disponibilidad de agua para los sembradíos, reduciéndose las producciones. Las sequías también afectan al sector urbano en lo que se refiere a la alta demanda de servicios que requieren de agua.

En años recientes, México se ha visto afectado por la ocurrencia de eventos de precipitación extrema, que han puesto en evidencia la vulnerabilidad de la población a inundaciones, principalmente en regiones montañosas cercanas a la cuenca de ríos. Por otra parte, un déficit de lluvia, por ejemplo la sequía ocurrida en 1998, que ha sido una de las más severas que se tenga registro en México, favoreció a una mayor incidencia de incendios forestales ocasionando grandes pérdidas económicas. Actualmente los ríos parecen ser la ampliación del alcantarillado, por la forma en que ahora son utilizados, están provocando daños medioambientales en todo el país, es decir, generaciones de seres humanos hemos alterado los ríos y utilizado su riqueza biológica.

La mayoría de las amenazas provienen directamente de la influencia humana: planes hidroeléctricos, vertidos de residuos, intensificación de la agricultura, deforestación, urbanización, obras de ingeniería y la variación del curso de los ríos constituyen un sinfín de actuaciones sobre un mismo elemento natural; además muchas especies de invertebrados, mamíferos, aves y anfibios dependen de los sistemas fluviales para su supervivencia.

I.2. Planteamiento del problema y propuesta de solución

Ante la problemática planteada, la medición de caudales en ríos desempeña un papel de suma importancia, ya que la información del caudal del agua es muy útil en el caso de inundaciones, así como la aplicación en diversas áreas de estudio, como pueden ser: biológicas, ecológicas, geológicas, hidráulicas e ingenieriles.

Con la intención de participar en la realización de aforos en ríos, la Coordinación de Instrumentación del Instituto de Ingeniería de la UNAM desarrolló un limnímetro electrónico, cuyos elementos principales se muestran en la figura 1.1. Las características más importantes de dicha estación son: rango de medición ±19.999m a ±199.99m; resolución 0.001m a 0.01m; memoria (EEPROM) de 30,000 datos aproximadamente; intervalos de muestreo de 1,2,3,4,5,6,10,12,15,20,30 minutos y 1,2,3,6,8,12,24 horas; voltaje de operación 1x1.5V, batería tamaño C; temperatura de operación -20° C a 70° C; display (1 línea 4½ posiciones); interfaces RS-232, SDI-12 (versión 1.2) e infrarrojo de bajo nivel.

En cuanto a las comunicaciones cuenta con tres protocolos de comunicación: RS-232C (19200 baudios), SDI-12 (1200 baudios, versión 1.2.) y el último, IrDA, es propietario, ya que se realiza una comunicación vía infrarroja de bajo nivel, es decir, el nivel más básico del protocolo que es el físico.

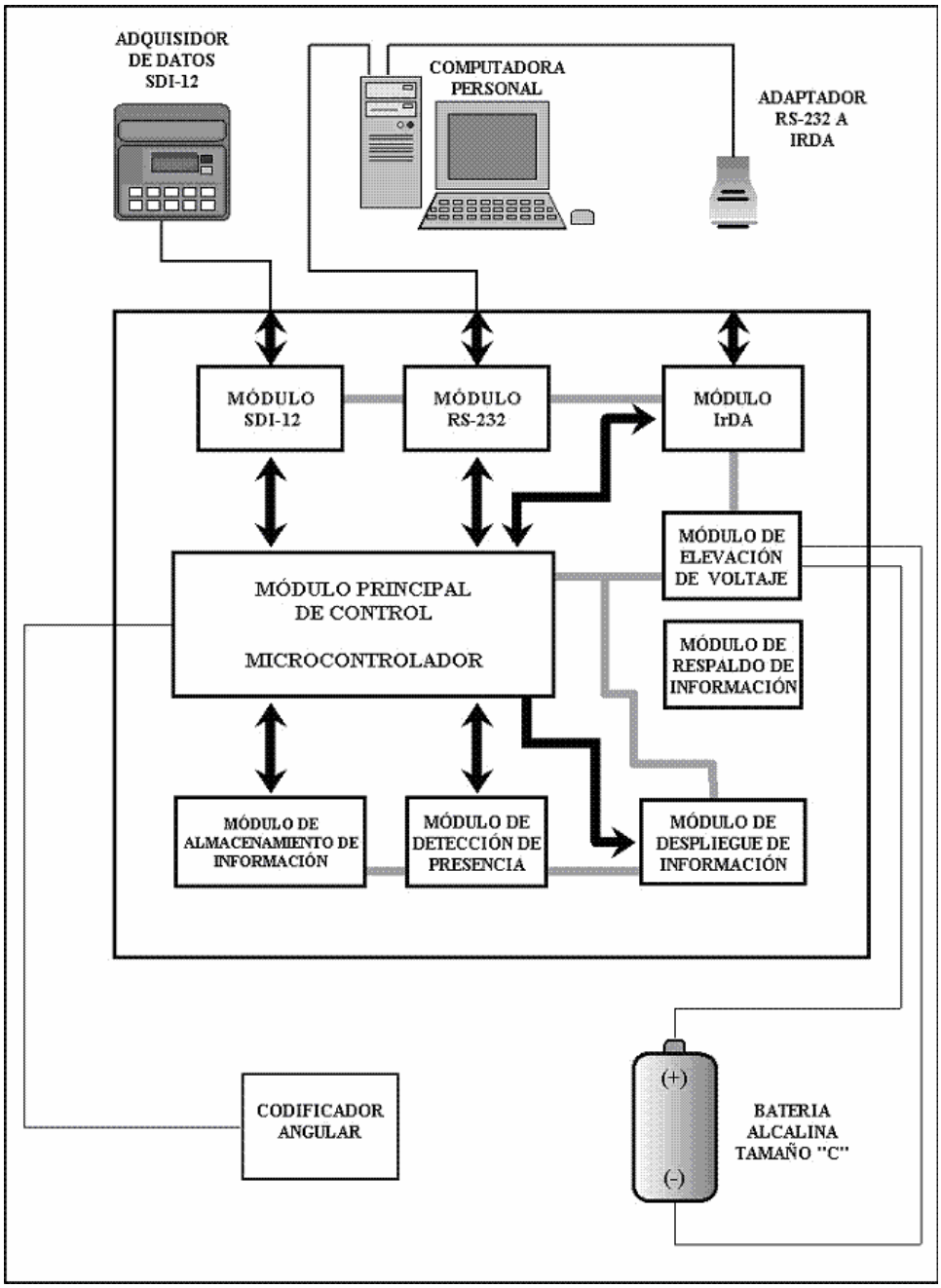


Figura 1.1. Diagrama a bloques del limnímetro digital.

En general, la operación de cualquier estación remota involucra los aspectos de: medición, registro, transmisión, tratamiento y evaluación de datos. Si el equipo cuenta con la integración de las funciones anteriores, se tiene como resultado una excelente herramienta de monitoreo y estudios de los recursos hidrológicos.

Bajo este esquema se realizó un análisis del limnímetro electrónico, en el cual se observó que aunque contaba con algunos elementos de transmisión de datos, se requería de visitas de campo en los lugares donde están situadas las estaciones remotas. Éste es el problema fundamental que se aborda en el presente proyecto.

La solución que se plantea es el desarrollo de un prototipo de comunicaciones de datos, donde éstos puedan ser alámbricos (RS232, USB y MODEM telefónico) e inalámbricos (IrDA, RADIO-MODEM y GSM), varios de ellos con características de inspección remota. Se pretende, sobretodo, con base a estos últimos, se cuente con elementos para la obtención de registros precisos y en tiempo real en una estación central (figura 1.2.); así se tendrá una menor dependencia de que dicho personal realice visitas de campo; además que este tipo de comunicaciones serán fácilmente controladas mediante dispositivos portátiles como pueden ser Laptos y PDA.

Algunas de las características de los tipos de comunicación a implementar son los siguientes:

- RS232 y USB, son puertos de comunicación muy usados en el ámbito de intercambio de datos en dispositivos portátiles, es por eso de la importancia de su implementación.
- En cuanto a la comunicación infrarroja se implementará el protocolo completo IrDA (*Asociación de datos infrarrojos*, *Infrared Data Association*), el cual se describirá más adelante, ya que en el sistema actual la comunicación se realiza en el nivel más básico del protocolo, que corresponde al nivel físico y por lo tanto no es posible comunicarse directamente a un equipo portátil o PDA que posea un puerto infrarrojo con protocolo IrDA.
- MODEM telefónico, para instalar este sistema es necesario un módem conectado a la estación remota y una línea telefónica común.
- RADIO-MODEM, la transmisión se hace a través de ondas de radio. Se trata de un sistema muy barato que únicamente llega a cubrir distancias que oscilan entre 12 y 15 km.

- GSM *(*Sistema Global para las Comunicaciones Móviles*, Global System for Mobile Communications*). Es el sistema más utilizado por sus bajos costos en llamadas, es necesario conectar la unidad GSM a la estación remota y asegurar que exista cobertura de área.

Cada uno de los módulos de comunicación, en general, cumple el siguiente esquema de comunicación:

1. El módulo fuente de información debe activar un camino directo de datos o bien debe proporcionar la identificación del sistema destino deseado.
2. El módulo fuente debe asegurar de que el destino está preparado para recibir datos.
3. La aplicación de transferencia de datos en el origen debe asegurarse de que el programa gestor en el destino está preparado para aceptar y almacenar el archivo para el usuario determinado.

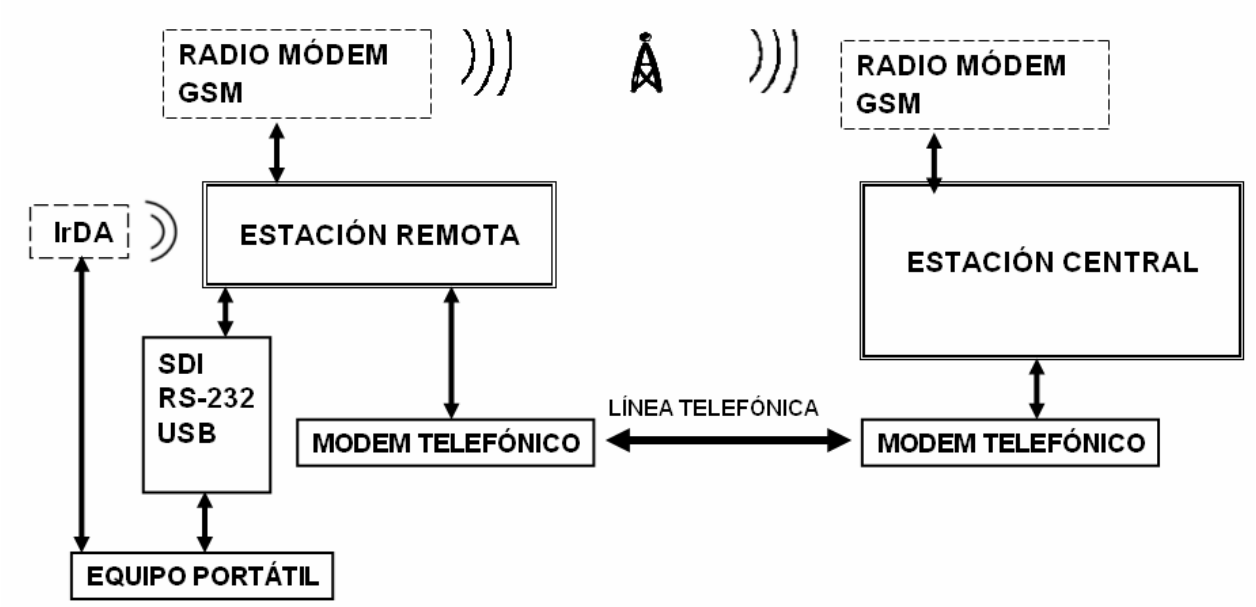


Figura 1.2. Módulos del sistema de comunicaciones alámbricas e inalámbricas.

I.3. Metodología

Para el desarrollo del prototipo de comunicaciones alámbricas e inalámbricas, se hará uso de una metodología bien establecida, la cual contempla dos vertientes: hardware y software.

El proceso del desarrollo del software de la estación central y el hardware/software de la estación remota, se conciben como un ciclo que consta de 4 etapas, las cuales son: diseño, implementación, medición y evaluación. El resultado de cada etapa es la alimentación de la que sigue, incluyendo la última, es decir, los resultados de la etapa de evaluación se toman para re-diseñar la interfaz, implementarla nuevamente, medir, y así sucesivamente. Debido a esa repetición o auto-alimentación se le llama diseño iterativo. En la tabla 1.1 se muestra cada etapa, así como las descripciones de las tareas de cada una de ellas.

ETAPA	**DESCRIPCIÓN DE TAREAS**
DISEÑO	*HARDWARE* ▪ Análisis del limnígrafo electrónico creado por la UNAM. ▪ Determinación de requerimientos del producto. ▪ Propuesta de los diferentes módulos de comunicación para el prototipo de comunicaciones alámbricas e inalámbricas. *SOFTWARE* ▪ Determinación de requerimientos del software. ▪ Organización de tareas. ▪ Conocimiento del usuario.
IMPLEMENTACIÓN	*HARDWARE* ▪ Investigación y estudio de la información correspondiente a los diferentes microcontroladores, así como los dispositivos electrónicos. ▪ Evaluación de lenguajes de programación de bajo y alto nivel para la programación de los microcontroladores.

Tabla 1.1. Metodología empleada en el desarrollo del prototipo de comunicaciones (continúa).

MEDICIÓN	*SOFTWARE* ▪ Determinación de las plataformas a utilizar, para así seleccionar el lenguaje de programación óptimo y seguro. Dependiendo de dicha plataforma, son empleados lenguajes de alto nivel, como son: Visual Basic, Visual C++ y Visual. NET ▪ Generación de prototipos. ▪ Desarrollo de la aplicación. ▪ Aplicación de pruebas (prueba piloto, pruebas con y sin usuarios), tanto al software como al hardware.
EVALUACIÓN	En la etapa de evaluación están considerados tanto el software como el hardware del prototipo. ▪ Análisis de los datos adquiridos tanto en la estación remota como en la estación central. ▪ Verificación de las diferencias. ▪ Generación de nuevas metas.

Tabla 1.1. Metodología empleada en el desarrollo del prototipo de comunicaciones.

En el capítulo siguiente se describirán detalladamente el diseño e implementación de cada módulo de comunicación.

CAPÍTULO II

GENERALIDADES

En este capítulo se revisan de manera breve los aspectos relacionados con los microcontroladores, en general, y las diferentes familias de estos dispositivos; se describe con detalle la familia AVR, que fue empleada para el diseño de la estación remota. Contiene además una descripción de los estándares: RS-232, USB, MODEM, RADIO MODEM, IrDA y GSM; explicando las características y capacidades más sobresalientes de cada una de ellos.

II.1. Microcontroladores

Un microcontrolador es un circuito integrado de alta escala de integración, constituido principalmente por tres unidades básicas: CPU (Unidad Central de Procesamiento, *Central Processing Unit*), para procesar la información; memoria de datos, para almacenar información temporal y la memoria de programa para el almacenamiento de las instrucciones. Además, estos dispositivos cuentan con líneas de E/S (Entrada/Salida) para comunicarse con el exterior y diversos módulos para el control de periféricos (temporizadores, puertos serie y paralelo, CAD (Conversor Analógico Digital, *Analog Digital Converter*), CDA (Conversor Digital Analógico, *Digital Analog Converter*)).

Las ventajas de incorporar un microcontrolador a un sistema son las siguientes:

- *Herramientas mejoradas.* Para facilitar el desarrollo de procesos experimentales e industriales.
- *Aumento de fiabilidad*. Al reemplazar el microcontrolador a un elevado número de elementos, disminuye el riesgo de averías y se precisan menos calibraciones.
- *Reducción de tamaño en el producto acabado*. Los productos tienden a ser de dimensiones pequeñas y se reduce el tiempo de maquilado y del personal necesario en este proceso.
- *Mayor flexibilidad*. La mayoría de los cambios pueden implementarse simplemente reprogramando el microcontrolador, es decir, sólo es necesario cambiar un programa y no el circuito lógico.

II.1.1. Arquitectura

Los Microcontroladores pueden identificarse por su arquitectura, ya sea arquitectura *Von Neumann* o arquitectura *Harvard*.

Arquitectura Von Neumann

Esta arquitectura se caracteriza por tener la *CPU* y la memoria interconectada por un grupo de líneas de direcciones y datos común. Hay aspectos positivos en esta configuración, como lo son los accesos a tablas almacenadas en *ROM* y un conjunto de instrucciones más ortogonal. El grupo de líneas de direcciones es usado para identificar qué localidad de memoria está siendo accesada, mientras que el grupo de líneas de datos es utilizado para trasladar información entre la *CPU* y alguna localidad de memoria o viceversa.

Con un solo grupo de líneas, la arquitectura *Von Neumann* es usada secuencialmente para acceder a instrucciones de la memoria de programa y ejecutarlas bidireccionalmente en la memoria de datos. Esto significa que el ciclo de instrucción no puede traslaparse con algún acceso a la memoria de datos.

La principal ventaja de la arquitectura *Von Neumann* es que se tiene un grupo de líneas de direcciones y de datos uniendo la memoria con la *CPU*. Una desventaja es que el apuntador de programa o algún otro registro se corrompieran y apuntara a la memoria de datos y se tomara ésta momentáneamente como memoria de programa. Consecuentemente se ejecutaría una instrucción no deseada o un error en la decodificación de la instrucción.

Arquitectura Harvard

La arquitectura Harvard se caracteriza por tener grupos de líneas separados para la memoria de programa y la memoria de datos. Una de las ventajas de la arquitectura *Harvard* es que la operación del microcontrolador puede ser controlada fácilmente si se presentara una anomalía en el apuntador de programa, además, el tamaño de las instrucciones puede ser distinto de tamaño de la palabra de datos, consiguiendo que las instrucciones se ejecuten en menos ciclos de reloj, por lo tanto el tamaño de las instrucciones no está relacionado con el de los datos. Existe otra arquitectura que permite accesos a tablas de datos desde la memoria de programa, esta arquitectura es llamada arquitectura *Harvard* modificada.

La arquitectura Harvard modificada es la dominante en los microcontroladores actuales, y tiene como característica principal que las tablas de datos pueden estar en la memoria de programa sin que sean perdidas cada vez que el sistema es apagado. Otra ventaja importante en la arquitectura *Harvard* modificada es que las transferencias de datos pueden ser traslapadas con los ciclos de decodificación de instrucciones. Esto quiere decir que la siguiente instrucción puede ser cargada de la memoria de programa mientras se está ejecutando una instrucción interviniendo la memoria de datos. La desventaja de la arquitectura *Harvard* modificada es que se requieren instrucciones especiales para acceder a los datos en *RAM* y *ROM* haciendo la programación un poco complicada.

II.1.2. Familias de microcontroladores

En la actualidad existe una gran variedad de familias de microcontroladores, estos pueden ser de 4, 8, 16 ó 32 bits. Los microcontroladores de 8 bits dominan el mercado y los de 4 bits se resisten a desaparecer. La razón de esta tendencia es que los microcontroladores de 4 y 8 bits son apropiados para una gran mayoría de aplicaciones que no requieren demasiados recursos, para satisfacer la tarea especificada. Respecto al mercado de los microcontroladores de 16 bits, el crecimiento también ha sido su norma. De los 23 millones de unidades vendidas en 1996, se ha pasado a realizar ventas diez veces mayor en 1998, siendo la informática y las comunicaciones las áreas que absorbieron la mayor parte de estos dispositivos. Los microcontroladores de 32 bits van afianzando sus posiciones en el mercado, siendo las áreas de más interés las siguientes: procesamiento de imágenes, comunicaciones, aplicaciones militares, procesos industriales y control de dispositivos de almacenamiento masivo de datos.

En la actualidad, gran parte de los fabricantes de circuitos integrados disponen de su propia línea de microcontroladores (tabla 2.1). Según el modelo de microcontrolador que se trate, el tamaño y el tipo de memoria pueden variar, así como el número de líneas de E/S y módulos de control de periféricos. La diversificación de modelos permite seleccionar el más adecuado según la aplicación que se trate.

FABRICANTE	MODELOS DE MICROCONTROLADORES
ATMEL	Familia AVR
HITACHI	HD64180
INTEL	8048, 8051, 80C196, 80186, 80188, 80386EX
MICROCHIP	PIC
MOTOROLA	6905, 68HC11, 68HC12, 68HC16, 683XX
NATIONAL SEMICONDUCTOR	COP400, COP800
PHILIPS	GAMA COMPLETA DE CLONES DEL 8051
SGS-THOMSON	ST-62XX
TEXAS INSTRUMENTS	TMS370
TOSHIBA	68HC11
ZILOG	Z8, Z86XX

Tabla 2.1. Fabricantes de microcontroladores con sus modelos más populares.

En el desarrollo del prototipo de sistema de comunicaciones de datos (inalámbricos y alámbricos), se empleará la familia de microcontroladores *AVR* de *ATMEL*, que por sus características demostró ser el dispositivo más adecuado para la transmisión y almacenamiento de información, entre las características principales para la elección del microcontrolador se contemplaron las siguientes: manejo de un bajo consumo de potencia y con modos de bajo consumo adicionales que permitan poner en modo de "*sleep*" al microcontrolador para lograr un ahorro mayor de energía. Los microcontroladores AVR están construidos bajo la arquitectura *Harvard* modificada, lo que permite utilizar varias tablas de valores constantes, las cuales pueden almacenarse en memoria ROM; como ventaja de emplear esta arquitectura es la eliminación de utilizar memorias de programa y de datos externas. El microcontrolador AVR también permite la deshabilitación de los puertos y la CPU. Además puede operar con base en dos relojes basados en cristales de cuarzo, con el de mayor frecuencia opera a máxima velocidad, mientras que con el de menor frecuencia trabaja con muy bajo consumo de potencia.

Lo cual por tener dos temporizadores hace que uno de ellos no dependa de la frecuencia de operación del oscilador del sistema y podrá seguir operando aún cuando los demás subsistemas del microcontrolador se encuentren deshabilitados, esta característica es importante ya que se requiere que el sistema trabaje con una bajo consumo de potencia. Por último, el microcontrolador AVR no divide la frecuencia de operación, es decir, trabaja con la frecuencia total del reloj, mientras que un microcontrolador PIC dividirá la frecuencia de operación entre cuatro.

A continuación se describe dicha arquitectura junto con sus características más sobresalientes, como son: número de terminales, tipos de memoria (FLASH, RAM y EEPROM), frecuencia de operación máxima, temporizadores, entre otros.

II.1.3. Arquitectura AVR

La familia *AVR* es una familia de microcontroladores *RISC* (Computadora de Conjunto de Instrucciones Reducido, *Reduced Instruction Set Computer*) de ocho *bits,* que cuenta hasta el momento con veintidós diferentes dispositivos, disponibles en empaques de ocho a cuarenta y cuatro terminales. La arquitectura *AVR* está optimizada para ser usada con lenguajes de alto nivel como el *C;* se basa en el modelo *Harvard* modificado, lo cual le permite almacenar datos constantes en la memoria de programa además de las instrucciones.

La estructura del procesador es la misma en todos los dispositivos *AVR*. Todos los dispositivos cuentan con memoria de programa *Flash* y ejecutan esencialmente el mismo conjunto de instrucciones. La mayoría de estos microcontroladores posee 118 instrucciones, los modelos avanzados cuentan con un total de 130. Las instrucciones se encuentran integradas en cuatro grupos: aritmética y lógicas, de salto, de transferencia de datos y de prueba bit a bit, además, cuenta con la característica de ser programables dentro del sistema, esto permite la actualización del software sin la necesidad de remover el circuito de la tarjeta de aplicación final.

Algunos de los dispositivos cuentan con un solo temporizador (*timer*) de ocho *bits*, mientras que otros cuentan con temporizadores de 8 y 16 *bits*. Algunos de los dispositivos contienen un conversor analógico digital de 10 *bits* de resolución, y otros cuentan con uno o más puertos de comunicación serial asíncrona (*UART*) y comunicación síncrona. La mayoría de los miembros de esta familia cuenta con dos modos de bajo consumo y algunos otros cuentan con tres. La mayoría de los microcontroladores de esta familia tienen con la característica de poder ser

programarlos dentro del sistema, esto permite la actualización del software sin la necesidad de remover el circuito de la tarjeta de aplicación final. La tabla 2.2 muestra algunas de las características de algunos de los microcontroladores miembros de la familia *AVR*.

Dispositivo	Terminales E/S	*FLASH* (bytes)	RAM (Bytes)	*EEPROM* (Bytes)	Frecuencia de operación máxima (MHz)	Timers 8 *bits*	Timers 16 *bits*	Otro
ATtiny10	8,6	1024	0	64	8	1	0	
ATtiny22	8,6	1024	0	64	1.6	2	0	
ATtiny15	8,6	1024	0	64	1.6	2	0	Conversor A/D 10 *bits*, de 4 canales
AT90S2323	8,3	2048	128	128	10	1	0	
AT90S2343	8,5	2048	128	10	10	1	0	
AT90S1200	20,15	1024	0	64	12	1	0	
AT90S2313	20,15	2048	128	128	10	1	1	UART
AT90S2333	28,20	2048	128	128	8	1	1	UART, Conversor A/D 10 *bits*, 6 canales
AT90S4433	28,20	4096	128	256	8	1	1	UART, Conversor A/D 10 *bits*, 6 canales
AT90S4414	40,32	4096	256	256	8	1	1	UART
AT90S8515	40,32	8192	512	512	8	1	1	UART
AT90S4434	40,32	4096	256	256	8	2	1	UART, Conversor A/D 10 *bits*, 8 canales
AT90S8535	40,32	8192	512	512	8	2	1	UART, Conversor A/D 10 *bits*, 8 canales
ATmega161	40,35	16384	1024	512	8	2	1	2 UART, SPI
ATmega603	64,32	64 k	4 k	2 k	6	2	1	UART, Conversor A/D 10 *bits*, 8 canales
ATmega103	64,32	128 k	4 k	2 k	6	2	1	UART, Conversor A/D 10 *bits*, 8 canales

Tabla 2.2. Características principales de la familia AVR de ATMEL.

II.2. Estándares de comunicación

En la industria de las comunicaciones, desde hace tiempo, se aceptó la necesidad de los estándares para definir las características físicas, mecánicas, eléctricas y de procedimiento de

los equipos de comunicación. Los fabricantes de equipos de cómputo comprendieron que sus equipos deberían en general, interconectarse y comunicarse con equipos desarrollados por terceros. Los diferentes equipos utilizados en las comunicaciones, de diferentes fabricantes, deben comunicarse entre sí y, es más, dada la evolución actual en la normalización de los protocolos, los clientes no admiten ya tener que desarrollar o adquirir software para adaptar protocolos de uso específico. Como consecuencia, en la actualidad *la normalización* se está imponiendo en todas las áreas tecnológicas.

Un *protocolo* de comunicación se define como: un conjunto de normas que están obligados a cumplir todos los dispositivos electrónicos que intervienen en la transferencia de información entre dos entidades.

Un protocolo de comunicación describe los siguientes aspectos:

1. El tiempo relativo al intercambio de mensajes entre dos sistemas de comunicación.
2. El formato que el mensaje debe contener para que el intercambio entre dos computadoras, que emplean protocolos diferentes, puedan establecer comunicación.
3. Las acciones que deben realizarse en el caso de producirse error en la comunicación.
4. Las suposiciones acerca del medio ambiente en el cual se ejecutará el protocolo.

Para el caso de este trabajo describiremos las características principales de los protocolos: RS232, USB, modem, radio modem, IrDA y GSM.

II.2.1. RS-232

El puerto serie RS-232C (*Recomended Standard 232)*, presente en las PC's actuales, es la forma más comúnmente usada para realizar transmisiones de datos entre PC's. El RS-232C es un estándar que constituye la tercera revisión de la antigua norma RS-232, propuesta por la EIA (Asociación de Industrias Electrónicas*, Electronic Industries Alliance*), realizándose posteriormente una versión internacional por el CCITT *(*Comité Consultivo Internacional Telegráfico y Telefónico*, Consultative Committee for International Telegraphy and Telephony),* conocida como V.24. Las diferencias entre ambas son mínimas, por lo que a veces se habla indistintamente de V.24 y de RS-232C (incluso sin el sufijo "C"), refiriéndose siempre al mismo estándar. El estándar RS-232, además de especificar lo niveles de voltaje y la distancia máxima de transmisión de señales digitales de datos, define otras características requeridas para conformar una interfaz de datos, que deberán ser usados cuando una comunicación serial está siendo enviada entre un equipo terminal de datos (DTE), como un computadora,

instrumentos digitales o una terminal y un equipo de comunicación de datos (DCE), como un modem, por ejemplo. Los niveles de voltaje considerados en el protocolo RS-232C están definidos de la siguiente manera:

El protocolo de comunicación RS-232C maneja niveles de voltaje de –15voltios a –3 voltios para representar al uno lógico y de +3 a +15 V para representar el cero lógico. Dependiendo de la funcionalidad del circuito de intercambio de datos, los valores eléctricos se interpretarán como datos binarios o como señales de control. Esta normalización especifica que, respecto a una referencia de tierra común, una tensión más negativa que -3 voltios se interprete como un uno binario, mientras que una tensión mayor de +3 voltios se interprete como un 0 binario (cualquier voltaje entre –3 voltios y +3 voltios tiene un nivel lógico no definido). De acuerdo a esta especificación, la codificación de las señales corresponde a NRZ-L (No Retorno a Nivel Cero, *Nonreturn to Zero-Level*), en donde el nivel del "0" lógico le corresponde un nivel de voltaje alto, mientras que al "1" lógico uno bajo. Típicamente, un sistema RS-232C usa voltajes nominales de –12 V para el uno lógico y +12 V para el cero lógico. Hay que notar que el voltaje más negativo corresponde al uno lógico. Esto es conocido como *lógica negativa*. En la figura 2.1 se muestra lo antes mencionado. Cabe destacar que, de acuerdo a la norma, para la transmisión de datos el cable de transmisión no debe sobrepasar los 15 metros de longitud.

Para las señales de control se aplican los mismos niveles de tensión: una tensión menor de -3 voltios se interpreta como OFF y una tensión mayor de +3 voltios se interpreta como ON.

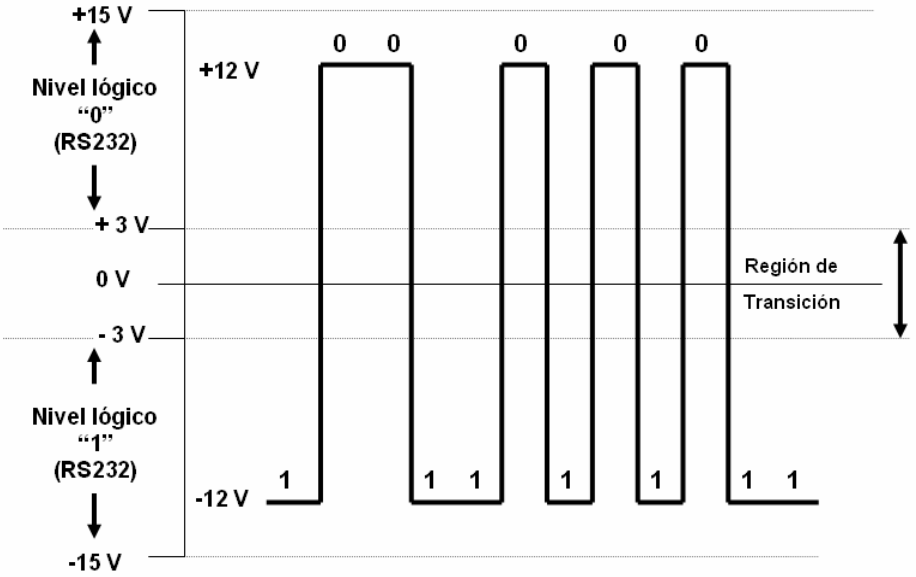


Figura. 2.1. Niveles de tensión lógicos RS232.

En cuanto a la especificación del número de terminales, se tienen conectores de 25 y 9 terminales, conocidos como DB-25 y DB-9. En general, en las comunicaciones bajo el estándar RS232, se utilizan tres terminales para transmitir información: dos son canales portadores de datos (TxD y RxD) y el tercero es un conductor de tierra de señal, que sirve como camino de retorno de la corriente de la señal. Los 22 terminales restantes sirven como terminales de control entre el DTE y el DCE.

En la tabla 2.3 se muestra la distribución de las terminales de acuerdo a la señal y su descripción correspondiente. La figura 2.2 muestra como es físicamente el conector.

Número de Terminal		Señal	Descripción	E/S
DB-25	**DB-9**			
1	1		Physical Earth	-
2	3	TxD	Transmit Data	S
3	2	RxD	Receive Data	E
4	7	RTS	Request To Send	S
5	8	CTS	Clear To Send	E
6	6	DSR	Data Set Ready	E
7	5	SG	Signal Ground	-
8	1	CD/DCD	(Data) Carrier Detect	E
15	-	TxC(*)	Transmit Clock	S
17	-	RxC(*)	Receive Clock	E
20	4	DTR	Data Terminal Ready	S
22	9	RI	Ring Indicador	E
24	-	RTxC(*)	Transmit/Receive Clock	S

Tabla 2.3. Distribución de terminales en los conectores DB25 y DB9.
(*) = Normalmente no conectados en el DB-25.

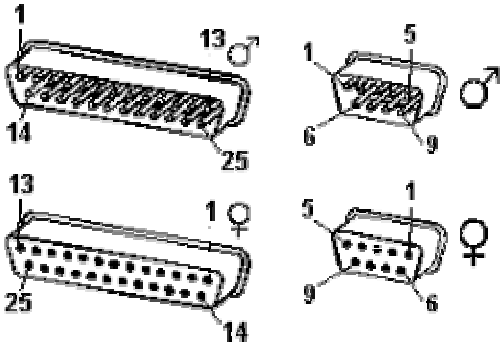


Figura 2.2. Conectores DB25 y DB9.

En sistemas convencionales, los niveles de voltaje lógicos estándar están de acuerdo a los voltajes TTL estándar de 0 V y 5 V. Esos voltajes son empleados también por los dispositivos *MOS* y *CMOS* del sistema. Estos niveles TTL no son compatibles con equipo periférico que utilice la interfaz RS232, y consecuentemente algún tipo de circuito de interfaz es necesario para convertir entre voltajes TTL y RS-232C.

Los diferentes equipos que cuentan con un puerto de comunicación serial, contienen un circuito integrado denominado UART (Transmisor Receptor Asíncrono Universal*, Asynchronous Receiver Transmitter*) o USART (Transmisor Receptor Síncrono Asíncrono Universal*, Universal Synchronous Asynchronous Receiver Transmitter*). La función principal de estos circuitos integrados es transformar el flujo de información en paralelo a un flujo de información en serie. Normalmente se utilizan los siguientes modelos del circuito integrado UART: 8250 (bastante antiguo, con fallos, sólo llega a 9600 baudios), 16450 (versión corregida del 8250, llega hasta 115.200 baudios) y 16550A (con *buffer* de E/S). A partir de la gama Pentium, en las computadoras, la circuitería UART de la placa base son todas de alta velocidad, es decir UART 16550A. De hecho, la mayoría de los modems que son conectados al puerto serie necesitan dicho tipo de UART, incluso algunos juegos para red a través del puerto serie necesitan de este tipo de puerto serie. Las computadoras portátiles suelen llevar circuitos integrados 82510 (con *buffer* especial, emula al 16450) o el 8251 que es un tipo de circuito USART, compatible con los microprocesadores Intel.

Para controlar al puerto serie, la *CPU* emplea direcciones de puertos de E/S y líneas denominadas de interrupción, IRQ (Petición de Interrupción, *Interrupt ReQuest*); las direcciones 3F8h (o 0x3F8) e IRQ 4 se utilizan para el puerto de comunicaciones definido como COM1, y 2F8h e IRQ 3 para el COM2. Al añadir otros puertos serie, se eligieron las direcciones 3E8 y 2E8 para COM3-COM4, pero las IRQ no están especificadas, es decir, las IRQ para estos COM son asignadas por el sistema operativo, dentro del rango de hasta IRQ 15.

Mediante los puertos de E/S se pueden intercambiar datos, controlando este intercambio a través de las IRQ, que producen una interrupción para indicar a la CPU que ha ocurrido un evento, por ejemplo, que ha llegado un dato, o que ha cambiado el estado de algunas señales de entrada. La CPU debe responder a estas interrupciones lo más rápido posible, para que de tiempo a recoger el dato antes de que el siguiente lo sobrescriba. En el caso de la UART 16550A, ésta contiene unos *buffers* de datos de tipo FIFO (primero en entrar primero en salir,

First-In First-Out), dos de 16 bytes, donde se pueden guardar varios datos antes de que la CPU los recoja. Esto también disminuye el número de interrupciones por segundo generadas por el puerto serie.

El RS-232 puede transmitir los datos en grupos de 5, 6, 7 u 8 bits, a velocidades determinadas (normalmente, 9600 bits por segundo o más). Para la transmisión de los datos, se inicia con un bit de comienzo, que corresponde al valor binario 0, le sigue la información a transmitir, después un bit opcional de paridad (indica si el número de bits transmitidos es par o impar, para detectar fallos), y por último 1 o 2 bits de parada (stop). Normalmente, el protocolo utilizado es 8N1 (8 bits de datos, sin paridad y con 1 bit de stop).

En la figura 2.3 se presenta el diagrama de transmisión de un dato con formato 8N1. El receptor indica al emisor que puede enviarle datos activando la salida RTS. El emisor envía un bit de START (nivel alto) antes de los datos y un bit de STOP (nivel bajo) al final de éstos.

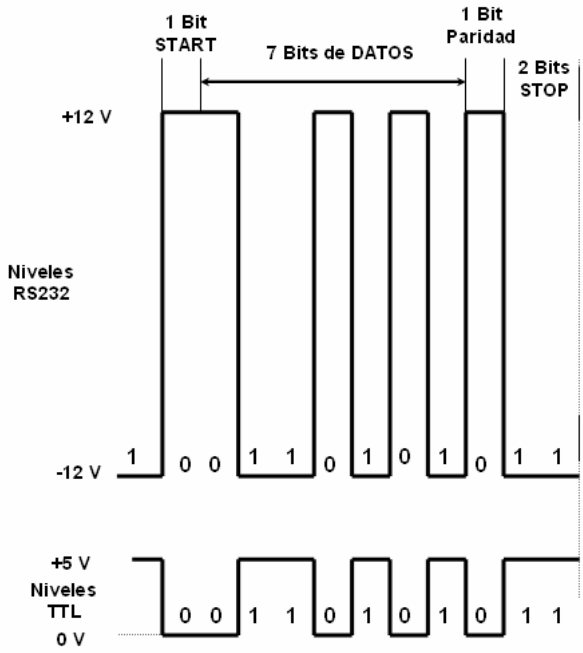


Figura. 2.3. Transmisión de un dato con formato 8N1.

Una vez que ha comenzado la transmisión de un dato, los bits tienen que llegar uno detrás de otro a una velocidad constante y en determinados instantes. Por eso se dice que el RS-232 es asíncrono por carácter y síncrono por bit. Las terminales que portan los datos son RXD y TXD. Las demás terminales se encargan de otros trabajos, por ejemplo: DTR indica que la

computadora está encendido, DSR que el aparato conectado a dicho puerto está encendido, RTS que la computadora puede recibir datos (porque no está ocupado), CTS que el aparato conectado puede recibir datos, y DCD detecta que existe una comunicación (presencia de datos).

Tanto el dispositivo a conectar como la computadora (o el programa terminal) tienen que usar el mismo protocolo serie para comunicarse entre sí. Puesto que el estándar RS-232 no permite indicar en que modo se está trabajando, es el usuario quien tiene que decidirlo y configurar ambas partes. Los parámetros configurables son: protocolo serie (8N1), velocidad del puerto serie y protocolo de control de flujo (*handshaking)*. Este último puede ser por *hardware* (RTS/CTS) o bien por software (XON/XOFF), que son señales que tienen la capacidad de detener o reanudar el flujo de datos. Por ejemplo, para el caso XON/XOFF, si una computadora envía datos a una impresora, más rápido de lo que la impresora puede procesar. La impresora contiene un buffer donde los datos son almacenados para poder recibir los mismos a la velocidad de envío de la computadora. Si dicho buffer se llena, antes de que la impresora haya procesado todos los datos, un pequeño microprocesador en la misma envía una señal XOFF para que pare de enviar datos y evitar perderlos. Cuando el buffer se ha vaciado lo suficiente, entonces la impresora vuelve a enviar una señal XON para que continúe el envío. Cuando lo que se envía son datos binarios, no es recomendable utilizar el tipo de *Handshaking* XON/XOFF*,* ya que es posible que las señales no se reconozcan, puesto que se contienen caracteres codificados.

La opción de velocidad del puerto especifica la velocidad máxima con la que los programas pueden transmitir datos, aunque, cuando es iniciada la transmisión de datos entre dispositivos éstos deben compartir dicha velocidad, para evitar errores de transmisión. La mayoría de los programas que realizan comunicación de datos a través del puerto serial, establecen su propia velocidad, con lo que se limita la velocidad a la que pueden enviar los datos, un ejemplo es una hiperterminal de datos de Windows.

En el caso de la comunicación entre dos equipos PC, en la figura 2.4 se muestra como se deben conectar las diferentes terminales.

(PC1) RxD <========== TxD (PC2)
(PC1) TxD ==========> RxD (PC2)
(PC1) DTR ==========> DSR (PC2)
(PC1) DSR <========== DTR (PC2)
(PC1) RTS ==========> CTS (PC2)
(PC1) CTS <========== RTS (PC2)
(PC1) TIERRA ========== TIERRA (PC2)

Figura 2.4. Conexión de terminales.

II.2.2. USB

En general, para el manejo de dispositivos externos a una PC, por ejemplo, las tarjetas de adquisición de datos (TAD), se hace uso de las ranuras internas de ésta. Dichas ranuras se conocen como bus ISA (Arquitectura Estándar de la Industria, *Industry Standard Architecture*) y PCI (Interconexión de Componentes Periféricos, *Peripheral Component Interconnect*).

El Puerto ISA fue creado por IBM en 1980, para ser empleado en las computadoras AT (Alta Tecnología) de IBM. La versión original del bus ISA se desarrolló para trabajar en 8 bits. Posteriormente, se creó una extensión de 16 bits aumentando su velocidad (8 MHz). Esta extensión es compatible de forma descendente con el puerto ISA de 8 bits. El ancho de banda máximo del puerto ISA de 16 bits es de 16 MBytes/segundo. Este ancho de banda es insuficiente para las necesidades actuales, tales como tarjetas de vídeo de alta resolución, por lo que el puerto ISA no se emplea en los equipos de cómputo actuales. Este puerto ha sido substituido por el puerto PCI.

El puerto PCI, es un puerto de la computadora que se utiliza para conectar dispositivos (tarjetas) periféricos a la tarjeta “madre” de la computadora. A diferencia de los puertos ISA, el puerto PCI permite la configuración dinámica de un dispositivo periférico (*Plug and Play*). En el encendido de la computadora, las tarjetas PCI y el sistema BIOS interactúan y negocian los recursos que son pedidos por las primeras. Esto permite la asignación de interrupciones y direcciones del puerto para las tarjetas PCI.

La conexión de las tarjetas de datos de E/S en los puertos ISA y PCI requiere que el usuario abra el gabinete de la computadora, inserte la tarjeta, ajuste los interruptores y puenteadores, se asegure de que los cambios no estén en conflicto con otras tarjetas, cierre el gabinete y reinicie el equipo de cómputo. Este proceso es difícil para la mayoría de usuarios que no están familiarizados con el *hardware* del sistema, ya que existe una probabilidad alta de cometer

algún error. Además, el número de ranuras ISA y PCI es muy limitado (dos o tres, por lo regular), por ejemplo, para usuarios que requieren de varias aplicaciones en su equipo de cómputo, como pueden ser tarjetas de: adquisición de datos, video de alta resolución, sonido, modem, entre otros. Las tarjetas *Plug and Play* eliminan los cambios de interruptores y puenteadores, pero, aún así, el usuario tiene la necesidad de abrir la computadora para instalar la tarjeta, además, que el número de ranuras sigue siendo limitado.

Para resolver el problema mencionado, a mediados de la década de los noventa se reunieron representantes de siete compañías (Compaq, DEC, IBM, Intel, Microsoft, NEC y Northern Telecom) para diseñar una mejor manera de conectar dispositivos de E/S de baja velocidad a una computadora. Desde entonces, cientos de compañías más se le han unido. El estándar resultante se llama USB (bus serial universal*, Universal Serial Bus*).

Algunos de los objetivos de las compañías que originalmente concibieron el USB e iniciaron el proyecto son los siguientes:

- Los usuarios no deben tener que ajustar interruptores ni puenteadores en las tarjetas o dispositivos.
- Los usuarios no deben tener que abrir el gabinete para instalar dispositivos de E/S nuevos.
- Sólo debe haber un tipo de cable, que sirva para conectar todos los dispositivos.
- Los dispositivos de E/S deberán obtener su energía del cable.
- Se deberán poder conectar hasta 127 dispositivos a una sola computadora.
- El sistema deberá apoyar dispositivos de tiempo real (sonido, teléfono, entre otros.).
- Los dispositivos deberán poder instalarse mientras la computadora está funcionando.
- No se deberá requerir un reinicio después de instalar un nuevo dispositivo.
- El nuevo bus y sus dispositivos de E/S deberán ser de bajo costo.

La primera especificación comercial de USB 1.1 fue liberada el 23 de septiembre de 1998. Un año después, USB era ya una interfaz común en la mayoría de los equipos de cómputo personal. El objetivo se cumplió, permitir que dispositivos de diversos fabricantes pudieran comunicarse entre sí en una arquitectura abierta, se diseñó para dispositivos de baja velocidad como teclados, ratones, cámaras de foto fija, digitalizadores de imágenes, teléfonos digitales,

entre otros. Las velocidades de transferencia de este dispositivo es de 1.5Mbits/s para dispositivos lentos y velocidad completa de 12Mbits/s, para los dispositivos que necesiten mayor ancho de banda. En abril de 2000 se presentó el USB de alta velocidad o USB 2.0 que llega 480Mbits/s. Un dispositivo de alta velocidad USB 2.0 puede conectarse a un controlador USB 1.1, pero no operará en toda su capacidad. Por su parte, un dispositivo USB 1.1 puede conectarse a un controlador USB 2.0 a un máximo de 12 Mbits/s. Si la computadora tiene USB 1.1 y se desea migrar a USB 2.0, será necesario adquirir e instalar una nueva tarjeta controladora.

Un sistema USB consiste en una estructura jerárquica en el cual el nodo principal se denomina eje raíz, donde éste es conectado a la línea de control y datos principal, es decir como si se conectara a un puerto PCI de la tarjeta madre. Este eje tiene zócalos para cables que se pueden conectar a dispositivos de E/S o a ejes de expansión, a fin de tener más zócalos.

Cuando se conecta un nuevo dispositivo de E/S, el eje raíz detecta este suceso e interrumpe el sistema operativo. Luego éste consulta al dispositivo para averiguar qué es y qué ancho de banda de USB necesita. Si el sistema operativo decide que hay suficiente ancho de banda para el dispositivo, asigna a éste una dirección única (1-127) y coloca esta dirección y otra información en los registros de configuración dentro del dispositivo. De este modo es posible añadir dispositivos nuevos sobre la marcha, sin que el usuario tenga que configurarlos y sin tener que instalar nuevas tarjetas ISA o PCI. El sistema USB puede verse como un conjunto de canales de bit que van del eje raíz a los dispositivos de E/S. Cada dispositivo puede dividir su canal de bits en, cuando más, 16 subcanales para diferentes tipos de datos (audio y video).

Dentro de cada canal o subcanal, fluyen datos del eje raíz al dispositivo o en la otra dirección. No hay tráfico entre dos dispositivos de E/S.

Cada 1.00 $\pm$ 0.05ms, el eje raíz transmite una nueva trama para mantener a todos los dispositivos sincronizados en el tiempo. Una trama se asocia a un canal de bits y consiste en paquetes, el primero de los cuales va del eje raíz al dispositivo. Los paquetes subsecuentes de la trama también podrían enviarse en esa dirección, pero también podrían regresar del dispositivo al eje raíz. En la figura 2.5 se presenta un ejemplo de lo que se acaba de comentar.

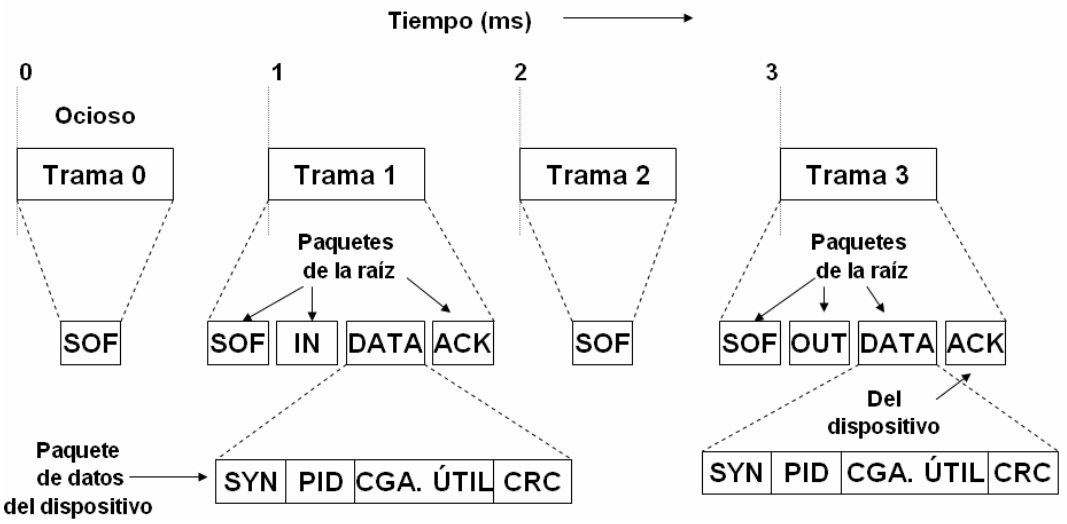


Figura 2.5. Flujo de datos en USB.

En la figura anterior, se puede observar que no existe trabajo que realizar en las tramas 0 y 2, así que todo lo que se necesita es un paquete SOF (inicio de trama, *Start of Frame*). Este paquete siempre se difunde a todos los dispositivos. La trama uno es un escrutinio, por ejemplo, una solicitud a un escáner para que devuelva los bits que ha encontrado en la imagen que está digitalizando. La trama 3 consiste en datos que se envían a algún dispositivo, digamos una impresora.

La interfaz USB reconoce cuatro clases de tramas: control, isocrónicas, volumen e interrupción. Las tramas de control sirven para configurar dispositivos, dar órdenes y preguntar por su estado. Las tramas isocrónicas son para dispositivos de tiempo real, como son: micrófonos, altavoces y teléfonos que necesitan enviar o aceptar datos a intervalos de tiempo precisos. Las tramas de control e isocrónicas tienen un retraso altamente predecible pero no ofrecen retransmisión en caso de ocurrir errores. Las tramas de volumen son para transferencias grandes de datos, hacia/desde dispositivos que no requieren tiempo real, como las impresoras. Por último, las tramas de interrupción son necesarias porque USB no reconoce interrupciones. Por ejemplo, en lugar de hacer que un teclado cause una interrupción cada vez que se pulsa una tecla, el sistema operativo puede escrutarlo cada 50ms para recolectar cualesquier digitalizaciones que estén pendientes.

Una trama consiste en uno o más paquetes de información. Existen cuatro tipos de paquetes de información: testigo, datos, saludo y especiales. Los paquetes de testigo van de la raíz a un

dispositivo periférico y sirven para el control del sistema. Los paquetes SOF, IN y OUT son paquetes del tipo testigo. El paquete SOF es el primero de cada trama y marca su principio. Si no hay trabajo que efectuar, el paquete SOF es el único de la trama. El paquete testigo IN es un escrutinio, que pide al dispositivo periférico devolver ciertos datos. Los campos del paquete IN indican cuál es el canal de bits que se está escrutando para que el dispositivo sepa que datos debe devolver. El paquete de testigo OUT anuncia que acontinuación vienen datos para el dispositivo periférico. Los paquetes de datos, DATA, sirven para transmitir hasta 64 bytes de información en cualquier dirección. En la figura 2.4 se muestra el formato de un paquete de datos (trama1), como respuesta a un paquete testigo (trama SOF); el paquete de datos consiste en un campo de sincronización de 8 bits, un tipo de paquete denominado PID (Campo de identificador de Paquete, *Packet Identifier Field*) de 8 bits, la carga útil y un código para detección de errores, CRC (Código de Redundancia Cíclica). En el caso del paquete de saludo se han definido tres clases de paquetes: ACK (el paquete de datos anterior se recibió correctamente, *Acknowledgement*), NAK (se detectó un error de CRC) y STALL (ocupado). Para el paquete especial se define el paquete PRE (para el tráfico en la línea de control y datos).

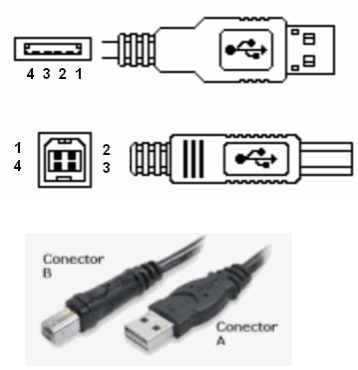
Una descripción a detalle de la figura 2.5 nos indicaría que cada 1.00ms debe enviarse una trama del eje raíz, incluso si no hay trabajo. Las tramas 0 y 2 consisten en un solo paquete SOF, lo que indica no hubo trabajo. La trama 1 es un escrutinio, de modo que principia con paquetes SOF e IN de la computadora al periférico de E/S, seguidos de un paquete DATA del periférico de E/S a la computadora. El paquete ACK le indica al dispositivo que los datos se recibieron correctamente. En caso de haber un error, se devolvería un NAK al dispositivo y el paquete se retransmitiría en el caso de datos de volumen. La trama 3 tiene una estructura similar a la 1, excepto que ahora los datos fluyen de la computadora al dispositivo de E/S.

Desde la perspectiva del usuario, los puertos e interfaces USB son muy sencillos de emplear, comenzando por los cables, cuyos conectores sólo son de dos tipos e imposibles de colocar de manera errónea. Cuando un dispositivo nuevo USB se asocia a una computadora, el sistema operativo detecta su presencia e instala el controlador correspondiente a dicho dispositivo, o bien, puede solicitar al usuario el disco de instalación de ese periférico. Después de trabajar con el dispositivo, el usuario puede desconectarlo directamente del puerto USB, sin riesgo de perder la configuración o dañar el dispositivo.

El cable USB consta de cuatro hilos (Figura 2.6.), los cuales son: dos para datos, uno para alimentación (+5volts) y uno para tierra, dichos hilos están rodeados de una capa de blindaje para evitar interferencias. El sistema de señalización transmite un 0 lógico como una transmisión de voltaje y un 1 lógico como la ausencia de una transmisión de voltaje, de modo que las series largas de ceros generan una serie de pulsaciones regular.

Cuando se deben conectar más dispositivos de los que ya ocupan los puertos USB, es indispensable usar un concentrador USB (Hub). El concentrador amplía la cantidad de puertos disponibles para otros dispositivos. Una sola computadora, combinando cables de no más de cinco metros de longitud cada uno, y concentradores, puede tener asociados hasta 127 dispositivos USB.

El USB utiliza un nuevo tipo de cable pequeño, sencillo, económico y fácil de conectar. Los cables tienen diferentes conectores en el extremo del eje y en el extremo del dispositivo, para evitar que las personas conecten accidentalmente un zócalo de eje a otro; únicamente existe un tipo de cable, pero con dos tipos de conectores diferentes: A y B. Los primeros presentan los cuatro terminales correspondientes a los cuatro hilos alineados en un plano, mientras que los conectores B presentan los contactos distribuidos en dos planos paralelos, dos en cada plano. La longitud del cable no debe exceder de 5 metros, aunque es posible encadenar 5 cables periféricos USB y 4 *hubs* donde se puede obtener una distancia de 25 metros.



Terminal	Nombre	Descripción
1	VBUS	+5 VDC
2	D-	Data -
3	D+	Data +
4	GND	Ground

Figura 2.6. Cable USB y conectores (A y B).

El USB utiliza la decodificación de datos NRZI (*Non Return to Zero, Invert on ones*), se mantiene constante el nivel de tensión durante la duración de un bit. Los datos se codifican mediante la presencia o ausencia de una transición de la señal al principio del intervalo de

duración de un bit. Un uno se codifica mediante la transición (bajo a alto o alto bajo) al principio del intervalo de señalización, mientras que un cero se representa por la ausencia de transición.

NRZI es un ejemplo de codificación diferencial. En la codificación diferencial, en lugar de determinar el valor absoluto, la señal se decodifica en función de los cambios entre los elementos de señales adyacentes. En términos generales, la codificación de cada bit se hace de la siguiente manera: si se trata del valor binario 0, se codifica con la misma señal que el bit anterior; si se trata de un valor binario 1, entonces se codifica con una señal diferente que la utilizada para el bit precedente, como se muestra en la figura 2.7. Una ventaja de este esquema es que en presencia de ruido puede ser más seguro detectar una transición en lugar de comparar un valor con un umbral.

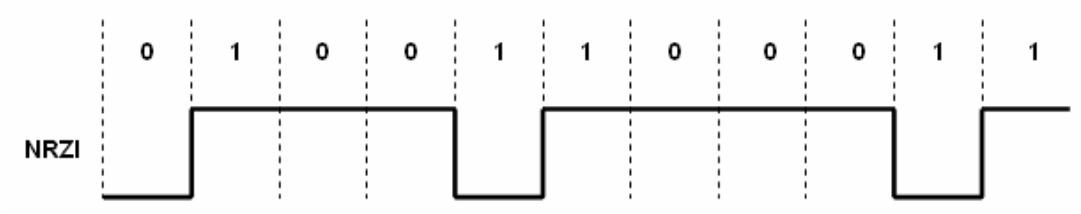


Figura 2.7. Formato de codificación NRZI.

Los voltajes de operación del USB son: 4.75 a 5.25 voltios en el puerto, considerando un nivel bajo menor de 0.8 voltios y un nivel alto mayor de 2.0 voltios.

II.2.3. MODEM

Con el fin de lograr que cualquier computadora se pueda comunicar con otras computadoras, la EIA creó una interfaz estándar denominada RS232. Las características principales de esta norma son: que cualquier computadora que reconozca la interfaz RS232 se pueda comunicar con alguna otra computadora que de igual forma reconozca dicha interfaz; también son definidas las características tales como: el tamaño del cable, la forma física del conector, los niveles de voltajes, así como las terminales correspondientes del conector.

Una limitante de la interfaz RS232 es la distancia de conexión, de acuerdo a la norma es de 15m. Una manera de resolver este problema es emplear una conexión a través del sistema telefónico. Desafortunadamente, el sistema telefónico no es capaz de transmitir las señales requeridas por la norma RS232, y es necesario insertar un dispositivo denominado modem (modulador-demodulador) entre la computadora y el teléfono y también entre la terminal y el

teléfono, con el fin de efectuar la conversión de las señales, es decir, la modulación transforma la señal digital binaria en analógica. La demodulación transforma la señal analógica en digital binaria.

Un aspecto muy importante de un modem es la modulación/demodulación empleada. La modulación es un concepto clave para las comunicaciones, y consiste en la variación de las características de una señal portadora, dependiendo de las características de otra señal denominada moduladora, y que es básicamente la que transporta la información. Algunos objetivos de las modulaciones son las siguientes:

- Reutilización del medio de transmisión, fundamentalmente mediante el uso de técnicas de multiacceso en frecuencia y en código.
- Adaptación de la señal al medio en la transmisión de la señal de comunicaciones, ya que se presentan diferentes características según la frecuencia y entorno (analógico, digital, amplitudes constantes, frecuencia variables, etc.)
- Creación de formas de señal que consigan aspectos como mayores velocidades de transmisión o mayor robustez en la propagación.

Las modulaciones se clasifican en analógicas y digitales, a su vez ambas se pueden subdividir en modulaciones de amplitud, frecuencia y fase. Las modulaciones analógicas son tratadas en relación con su calidad representada en función de la relación señal a ruido (S/N) y las digitales en relación con su tasa de errores de bit (BER)

Entre las modulaciones analógicas tenemos las siguientes:

- Modulación AM (Modulación de Amplitud, *Amplitude Modulation*). Las modulaciones en amplitud consisten en modificar la amplitud de la señal portadora de acuerdo con la amplitud de otra señal denominada moduladora, que es la que transporta la información.
- Modulación PM (Modulación de Fase, *Phase Modulation*). La modulación en fase se fundamenta en variar la fase de la señal portadora según como lo marque la señal moduladora.
- Modulación FM (modulación de frecuencia, Frequency Modulation). La modulación en frecuencia consiste en variar la frecuencia de la señal portadora de a cuerdo con una señal moduladora.

Para la modulación digital, existen tres técnicas básicas de modulación, que transforman los datos digitales en señales analógicas, las cuales son: ASK (Modulación en Amplitud, *Amplitude Shift Keying*), FSK (Modulación en Frecuencia, *Frecuency Shift Keying*) y PSK (Modulación en Fase, *Phase Shift Keying*).

Modulación en Amplitud

En ASK, los valores binarios (0 y 1) se representan mediante dos amplitudes diferentes de la portadora. Es usual que una de las amplitudes sea cero; es decir, uno de los dígitos binarios se representa mediante la presencia de la portadora a amplitud y frecuencia constante y el otro mediante la ausencia de portadora (figura 2.8). La señal transmitida por cada intervalo correspondiente a la duración de un bit es:

$$\text{ASK} \quad s(t) = \begin{cases} A\cos(2\pi f_c t) & 1 \text{ binario} \\ 0 & 0 \text{ binario} \end{cases}$$

portadora = $A\cos(2\pi f_c t)$

f_c = frecuencia portadora

ASK es sensible a los cambios repentinos de la ganancia. Además, es una técnica de modulación ineficaz. En líneas de calidad telefónica, ASK se emplea en el mejor de los casos a 1200bps.

La técnica ASK se usa para la transmisión de datos digitales en fibras ópticas. En los transmisores con LED. Es decir un elemento de señal se representa mediante un pulso de luz, mientras que el otro elemento se representa mediante la ausencia de luz.

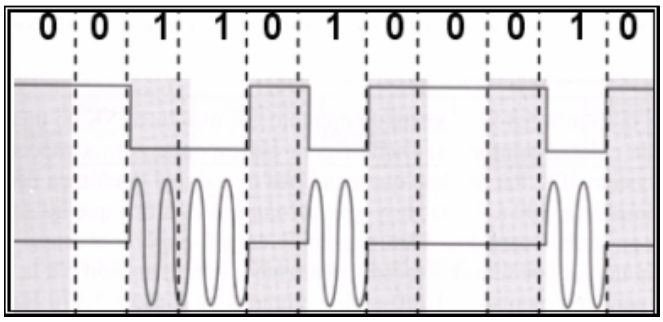


Figura 2.8. Modulación ASK.

Modulación en Frecuencia

El esquema FSK más empleado es el binario, BFSK (*binary FSK*). En este caso, los valores binarios se representan mediante dos frecuencias diferentes, próximas a la frecuencia de la portadora (figura 2.9). La señal transmitida en cada intervalo correspondiente a la duración de un bit será:

$$\text{BFSK} \quad s(t) = \begin{cases} A\cos(2\pi f_1 t) & 1 \text{ binario} \\ A\cos(2\pi f_2 t) & 0 \text{ binario} \end{cases}$$

f_1 y f_2 = frecuencias cuyo valor se define como un desplazamiento de la frecuencia portadora, de igual magnitud, pero en sentidos opuestos.

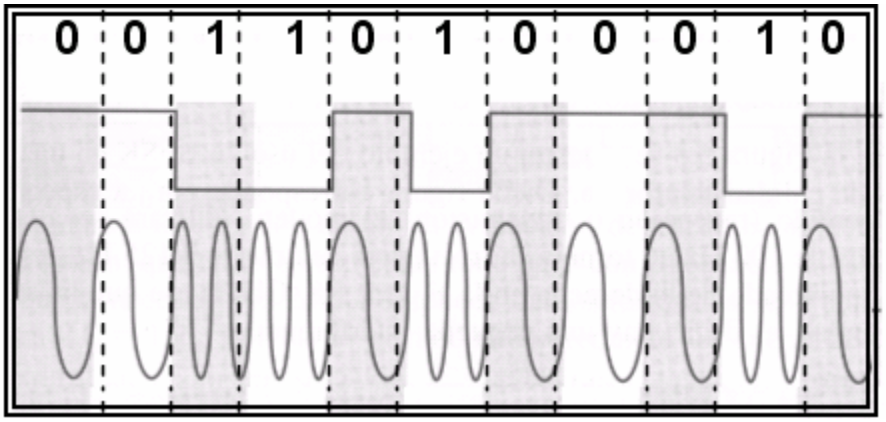


Figura 2.9. Modulación BFSK.

En la figura 2.9 se muestra un ejemplo del uso de la modulación BFSK en una transmisión full duplex para una línea de calidad telefónica. Dicha figura corresponde a la especificación de la serie de un modem *Bell System* 108. Para transmitir en full duplex, el ancho de banda anterior se divide en dos. En uno de los sentidos (correspondiente a la transmisión o recepción) las frecuencias utilizadas para representar al 1 o 0 están centradas en torno a 1170 Hz, con un desplazamiento lateral de 100Hz. El efecto de emplear estas dos frecuencias corresponde a la transmisión de una señal cuyo espectro corresponde con la zona sombreada de la izquierda de la figura 2.10. De igual manera, para el otro sentido (recepción o transmisión) el modem utilizará señales correspondientes a desplazamientos de 100Hz en torno a la frecuencia central de 2125Hz. Estas señales corresponden con el área sombreada de la derecha en la figura 2.10,

donde se puede observar que existe un ligero traslape entre las bandas, esto quiere decir que hay una pequeña interferencia.

BFSK es menos sensible a errores que ASK. En las líneas de calidad telefónica, se utilizan generalmente a velocidades de hasta 1200 bps. También se emplea en transmisiones de radio a más altas frecuencias (de 3 a 30 MHz). También se puede usar incluso a frecuencias superiores en redes de área local que utilicen cable coaxial.

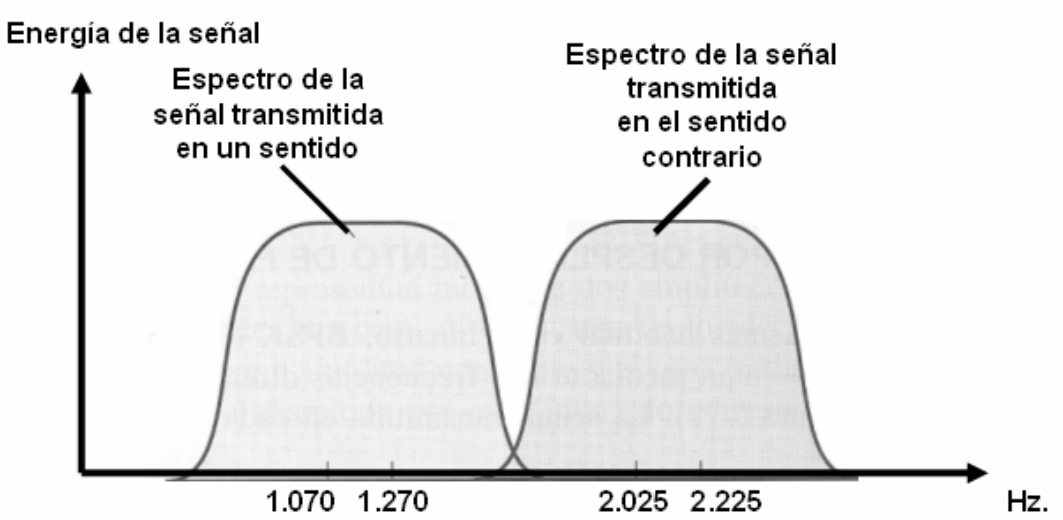


Figura 2.10. Transmisión FSK full duplex en una línea de calidad telefónica.

Modulación en Fase

En el esquema PSK, la fase de la señal portadora se desplaza para representar los datos digitales. El desplazamiento de fase binario (BPSK, *Binary PSK*), utiliza dos fases para representar los dos dígitos binarios, figura 2.11. La señal transmitida resultante durante el intervalo correspondiente a un bit es:

$$\text{BPSK} \quad s(t) = \begin{cases} A\cos(2\pi f_c t) \\ A\cos(2\pi f_c t + \pi) \end{cases} = \begin{cases} A\cos(2\pi f_c t) & \text{1 binario} \\ -A\cos(2\pi f_c t) & \text{0 binario} \end{cases}$$

El término de la derecha de la ecuación anterior, se debe a que un desplazamiento de 180 grados (π), es equivalente a invertir la onda sinusoidal. Si se tiene una secuencia de bits y se define *d(t)* como la función discreta igual a +1 durante la duración de un bit si el bit correspondiente en la secuencia de entrada es 1, e igual a -1 durante la duración de un bit si el

bit correspondiente en la secuencia de entrada es 0, entonces, la señal transmitida se puede definir como:

$$\text{BPSK} \qquad s_d(t) = A\ d(t) \cos(2\pi f_c t)$$

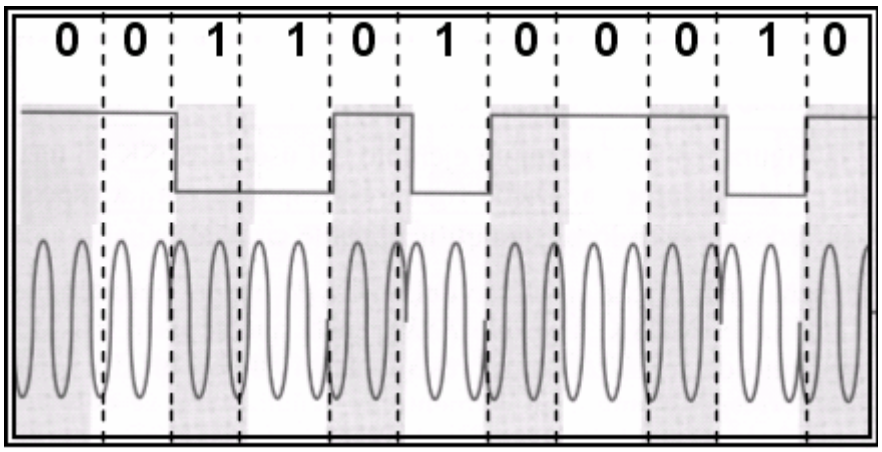


Figura 2.11. Modulación BPSK.

Una alternativa a la modulación PSK de dos niveles es la modulación PSK diferencial (DPSK, *differential PSK)*. En la figura 2.12 se muestra un ejemplo de esta última. En este esquema, un 0 binario se representa enviando un elemento de señal con la misma fase que el elemento anterior transmitido. Un 1 binario se representa enviando un elemento de señalización con fase invertida respecto al anterior elemento transmitido. El término diferencial se refiere al hecho de que el desplazamiento de fase es respecto al bit transmitido anterior, en lugar de ser respecto a una señal de referencia. En la codificación diferencial, la información a transmitir se representa en términos de los cambios introducidos entre los símbolos consecutivos, en lugar de en los elementos de señalización en sí. DPSK evita la necesidad de utilizar en el receptor un oscilador local de fase preciso, el cual debe estar acoplado con el transmisor. Mientras que la fase anterior se reciba correctamente, la referencia de fase será correcta.

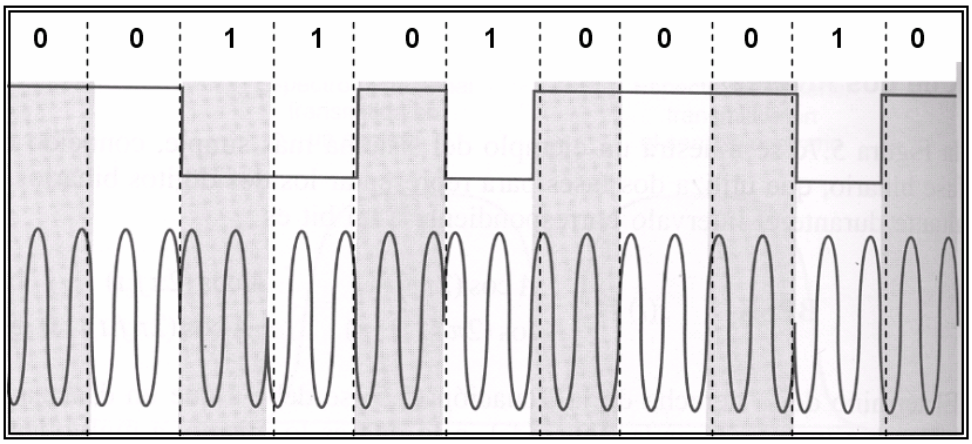


Figura 2.12. Modulación por desplazamiento de fase diferencial (DPSK).

La utilización de varios niveles se puede extender para transmitir más de dos bits a la vez. Por ejemplo, usando ocho ángulos de fase diferentes es posible transmitir tres bits a la vez. En otro caso, cada ángulo puede tener más de una amplitud. Por ejemplo, un modem a 9600 bps emplea 12 ángulos de fase, cuatro de los cuales tienen dos valores de amplitud diferentes, dando lugar a 16 elementos de señalización.

En el caso de las líneas telefónicas analógicas, éstas no pueden trabajar con las señales digitales, utilizadas por las computadoras, conocidas como ETTD (Equipo Terminal de Tratamiento de Datos), por lo que es necesario transformarlas en señales analógicas. El equipo que realiza esta transformación es el modem, conocido como ETCD (Equipo Terminal del Circuito de Datos) y que se utiliza tanto en la transmisión como en la recepción de las señales. La señal emitida por el ETCD a la línea tarda un tiempo en llegar al receptor, el cual es denominado *tiempo de propagación*.

En la comunicación a través de los equipos ETTD y ETCD a cada símbolo (estados de la señal que representan la información binaria) que se envía por la línea se le llama *nivel*. Estos símbolos no tienen por qué corresponderse uno a uno con cada bit, ni tampoco la decodificación de un símbolo depende sólo de su valor. Pueden asociarse varios bits con un nivel y decodificarse cada símbolo en función del precedente (decodificación diferencial). EL ETCD emisor recibe el mensaje de datos del ETTD, lo codifica y modula, mientras que el ETCD receptor lo demodula y decodifica. Hay modems que codifican la información digital binaria en otra también digital, para ser enviada por líneas de comunicación digitales. Estos ETCD reciben el nombre de ETCD Banda Base, y pueden ser síncronos o asíncronos. Este tipo de modems

pueden codificar la señal digital de la computadora de distintas formas, siendo las más comunes: NRZ (*Non Return to Zero*), bifase, bifase diferencial, Miller (modulación retardada), bipolar de orden 1, bipolar de orden 2, bipolar de alta densidad, y de valencia n.

- NRZ. El nivel de tensión se mantiene constante en la duración del bit, es decir, no hay transiciones. Por ejemplo, la ausencia de tensión es representada por un 0 binario, mientras que el nivel constante y positivo de tensión es representado por el 1 binario.
- Bifase. La señal cambia en medio del intervalo del bit, pero no retorno a cero.
- Bifase *diferencial*. Necesita dos cambios de señal para representar el bit '0', pero sólo '1' para representar el bit '1'.
- Miller. A partir de un código bifase, suprime una de cada dos transiciones.
- *Bipolar de orden 1*. Asigna al 0 o 1 puede ser -1 ó +1 dependiendo de su valor anterior.
- *Bipolar de orden 2*. Los bits pares e impares se codifican por separado de acuerdo al principio bipolar de orden 1.
- *Bipolar de alta densidad*. Es un código bipolar que ante secuencias largas de 0, envía secuencias de relleno (+1, -1). Son de orden n, donde n es el número de símbolos que reemplaza.
- *De valencia n*. Es una señal con n valores de tensión, donde cada valor representa uno a más bits. Si es bivalente (V1, V2) cada tensión representa un bit. Si es cuadrivalente (V1, V2, V3, V4) cada tensión representa dos bits.

La codificación digital pretende enviar el número máximo de bits por estado; compactar la banda de frecuencias de la señal; facilitar la sincronización, detección de errores, entre otras.

Los modems operan con tasas de datos entre 28,800 bits/s y 115,200 bits/s, y normalmente a tasas de bauds mucho más bajas. Emplean la combinación de técnicas para enviar varios bits por baud, modulando la amplitud, la frecuencia y la fase. Casi todos ellos son dúplex (full duplex), lo que implica que pueden transmitir en ambas direcciones al mismo tiempo (empleando diferentes frecuencia). Los modems que sólo pueden transmitir en una dirección a la vez se le llaman semiduplex (*half duplex*). Los modems que sólo pueden transmitir en una dirección son simplex.

Para la comunicación entre la computadora y un modem se utiliza la norma RS232C ó V.28, ésta última equiparable a la norma V.24. Para la sincronización entre modems es utilizado el

handshaking, donde puede ser por *hardware* (RTS/CTS) o bien por software (XON/XOFF), descritos anteriormente.

En cuanto a los comandos que definen la operación de los modems, existe un conjunto de estos denominado *AT* (*Attention Command*), *o Hayes*, que se encargan de realizar las diferentes funciones del modem, por ejemplo: ajuste de parámetros de la comunicación, control de la línea de conexión, llamadas, pruebas, entre otros. Los comandos se agrupan en comandos de comunicación y control.

Comandos de comunicación. Se encargan de ajustar los parámetros de trabajo del modem. Algunos ejemplos de este tipo de comando se muestran a continuación:

ATA	El modem queda en espera de una llamada telefónica, comportándose como un receptor.
ATE 0 ó 1	Selecciona eco local activado (1) o desactivado (0). Cuando estamos trabajando en modo full dúplex, el remoto envía un eco de lo que le enviamos. De esta forma si tenemos activado el eco local, veremos en pantalla los caracteres por duplicado (el eco remoto y el eco local).
ATF 0 ó 1	Selecciona el modo half (0) o full (1) Dúplex.
ATH 0 ó 1	Conecta (1) o desconecta (0) la línea telefónica desde el modem.
ATD	Encargado de realizar la llamadas telefónicas.
ATZ	Limpia el *buffer* del modem y restaura los valores originales grabados en RAM.

Comandos de control. Cumplen diversas funciones, por ejemplo: autocomprobación del funcionamiento del modem y lectura/escritura de registros, entre otros. Algunos ejemplos de este tipo de comando se muestran a continuación.

A/	Repite el último comando introducido en el modem.
ATI 0 a 3	Proporciona el número de versión (0), el valor de control de la ROM del modem (1), una comprobación interna (2), o la versión del software (3).
AT&Zn	Graba números de teléfonos en la RAM del modem.
AT&V	Muestra la configuración activa del modem.
AT&M 0 a 3	Establece el modo asíncrono (0) o el modo síncrono (1 a 3).

II.2.4. RADIO MODEM

Como se mencionó anteriormente, un modem se encarga de convertir un flujo de datos digitales en banda base a una señal analógica apropiada para ser transmitida sobre el medio, y viceversa. La principal diferencia entre un radio modem y un modem de cable se refiere a la aplicación a la que se destina. De este modo, los modems de cable están preparados para conectarse a redes de cable como pueden ser la red telefónica conmutada o una red híbrida de fibra óptica y coaxial. Los radio modems están destinados a aplicaciones en las cuales es necesario transmitir la señal vía radio, por ejemplo interconexión de ordenadores a través de LAN (red de área local, *Local Area Network*) o MAN (red de área metropolitana, *Metropolitan Area Network*) inalámbricas, envío y recepción de mensajes, telemetría, localización automática de vehículos, etc.

Los radios modems están preparados para transmitir sobre un entorno más hostil que el cable, a menudo sujeto a desvanecimientos, propagación multicamino (*multipath*) o interferencias. Para la transmisión de información, los radio modems disponibles comercialmente suelen utilizar las bandas ISM (*Industrial, Scientific and Medical*) de 900 MHz (902-928 MHz), 2,4 GHz (2400-2483,5 MHz) y 5,8 GHz (5725-5850 MHz).

Los radio modems actuales están mejorados para funcionar correctamente, en aspectos como: bandas de frecuencia; tolerancia de frecuencia y seguimiento de la portadora; potencia transmitida y margen dinámico; ecualización; efecto multicamino y esquemas de modulación. A continuación se realiza una breve descripción de cada una de ellas:

- *Bandas de frecuencia*. Los radio modems utilizan frecuencias que requieren de licencia para realizar transmisiones inalámbricas. En la figura 2.13. se muestran cuáles son estas bandas de frecuencia. Generalmente es empleado un conversor de frecuencia para colocar los canales de FI del radio modem en estas bandas. Además, son típicos esquemas de multiplexación conjuntos FDMA/TDMA (*Frequency Division Multiple Access/Time Division Multiple Access*), para compartir de forma eficiente el espectro radioeléctrico entre un conjunto de usuarios.

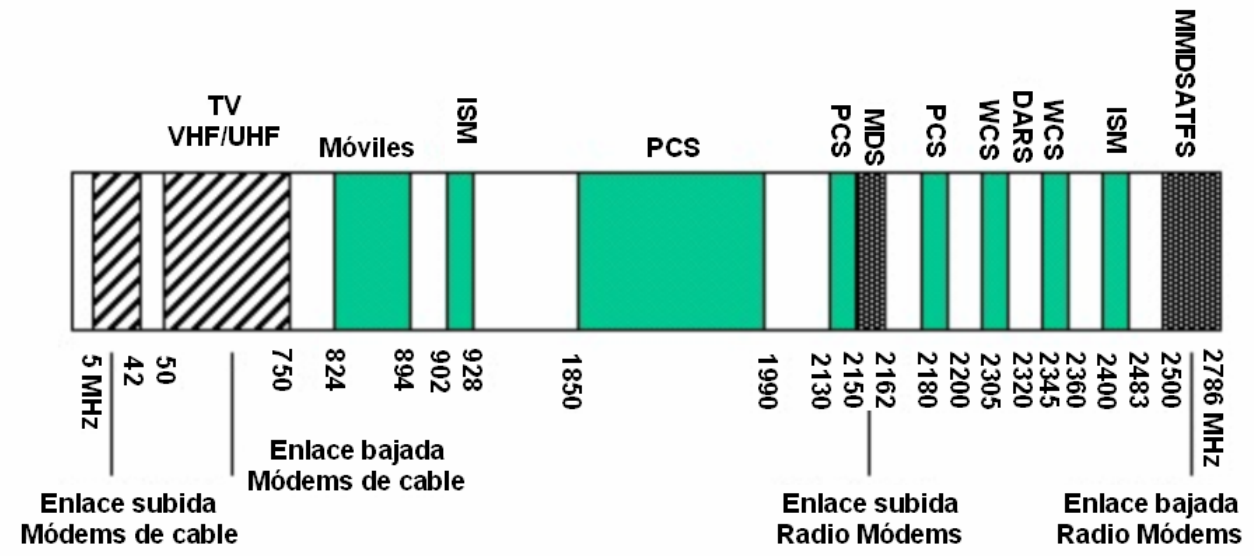


Figura 2.13. Bandas de frecuencia de sistemas inalámbricos.

- *Tolerancia de frecuencia y seguimiento de la portadora*. En un sistema inalámbrico, las frecuencias se convierten a las bandas MDS (Sistema de Datos por Microondas, *Microwave Data Systems*. Los radio modems contienen un mecanismo de búsqueda y seguimiento de la portadora por medio de bucles de enganche de fase, comúnmente conocidos como PLLs, que siguen la señal en rangos de 30 a 150 kHz.

- *Potencia transmitida y margen dinámico*. Cualquier demodulador posee un margen dinámico limitado en el que puede funcionar correctamente. La señal del enlace de subida debe estar contenida dentro del margen dinámico del demodulador de recepción. Esto incluye variaciones en el nivel de potencia de la señal debidas a la ganancia de las antenas, desvanecimientos por vegetación o precipitaciones y efecto multicamino. Los radio modems poseen un margen dinámico superior, típicamente de 20 dB; además, es necesario ejecutar un algoritmo inicial para que el radio modem localice el nivel de potencia adecuado para comenzar a funcionar. Este nivel es muy dependiente de las características del entorno donde vaya a trabajar.

- *Ecualización*. Durante la propagación de la señal de radio, ésta sufre variaciones de amplitud y de fase que es necesario corregir en el receptor. Estos cambios deben corregirse y compensarse dinámicamente. Es por ello que los radio modems disponen de ecualizadores en tiempo real que modifican su ganancia o introducen retardos de forma dinámica en función de las condiciones del medio. Normalmente se implementan por DSPs (procesadores digitales de señal, *Digital Signal Processor*)

- *Efecto multicamino*. La propagación multicamino no existe en los sistemas de cable, sin embargo, en los sistemas de radiocomunicaciones se convierte en uno de los principales problemas. Se produce como consecuencia de reflexiones de la señal que se combinan a la entrada de la antena y que dan lugar a degradaciones en el nivel de potencia o distorsión de la señal. En particular, un camino secundario de la señal ligeramente mayor puede ocasionar la cancelación completa del trayecto principal. En los radio modems esta situación es aun es más perjudicial, puesto que como suelen disponer de movilidad, es posible que en ciertas posiciones se produzca la reflexión en algún obstáculo inesperado.

- *Esquemas de modulación.* Normalmente se utiliza modulación QPSK (modulación por desplazamiento de fase en cuadratura, *Quadrature Phase Shift Keying*) para el enlace de subida y modulaciones 16QAM o 64QAM para el enlace de bajada. Conforme disminuye la complejidad de la modulación, se consigue una mayor inmunidad frente a desvanecimientos y efecto multicamino, aunque a costa de reducirse la tasa de transmisión. Lo mismo ocurre con la velocidad de modulación. En particular, las modulaciones de fase (QPSK y 64QAM) son más adecuadas para la propagación de señales sobre entornos radio.

La técnica de espectro ensanchado (*spread-spectrum*) consiste en la transformación reversible de una señal de forma que su energía se disperse entre una banda de frecuencias mayor que la que ocupaba en un principio. Este tipo de transformación es aplicado en los radio modems. El ensanchamiento de la banda se realiza a partir de una señal pseudoaleatoria, es decir, con una apariencia de ruido. La señal transmitida tendrá características pseudoaleatorias, y sólo podrá ser demodulada, si se es capaz de generar la misma señal pseudoaleatoria utilizada por el transmisor. La mayoría de radio modems comerciales implementan en su interior la técnica de

espectro ensanchado. En particular, suele ser típica en tarjetas de red de PCs que necesitan interconectarse por medio de una LAN.

Las computadoras, terminales o controladores utilizan técnicas de conmutación de paquetes para realizar la transmisión, de tal modo que los mensajes se fragmentan en paquetes que son transmitidos secuencialmente sobre el medio compartido. Esta técnica conduce a una transmisión 100% libre de errores. Cada paquete se comprueba en el receptor y, en caso de recepción incorrecta, se retransmite.

La conexión con el radio modem se realiza a través del puerto serie de un PC. Además, es posible realizar transmisiones punto a punto, punto a multipunto y multipunto a multipunto. Todas estas comunicaciones se pueden realizar en modo Maestro/Esclavo, en donde el dispositivo maestro generalmente se encuentra en la estación central y es el que controlo a los demás dispositivos (esclavos).

Los radio modems incorporan los siguientes elementos: las unidades de modulación y de radio, las baterías y la antena interna o de un conector externo para instalar la antena. En la tabla 2.4 se muestran algunas de las especificaciones técnicas más sobresalientes de un radio modem comercial.

Interfaz	RS-232C, DB25 hembra configurado como DCE.
Tasa de datos del DTE	300 a 38,4 kbit/s, asíncrono, full-dúplex, programable (de fabrica con los siguientes datos 9600, 8, N, 1).
Tasa de datos del enlace y protocolo	76,8 kbit/s A.X. 25, conmutación de paquetes con control de errores.
Contención	CSMA (*Carrier Sense Multiple Access*).
Canales de RF	Ch.1: 907,4 MHz, Ch.2: 914,8 MHz, Ch.3: 921,0 MHz.
Potencia transmitida	20 mW (+13 dBm).
Ancho de banda de canal	+/ 800 kHz. (puntos a 20 dB), +/ 3 MHz (puntos a 50 dB).

Tabla 2.4. Especificaciones técnicas de un radio modem comercial típico. (Continúa).

Modulación	Espectro ensanchado por secuencia directa, DPSK de banda ancha.
Sensibilidad	100 dB como mínimo.
Núm. Repetidores	Máximo 7 por trayecto.
Alcance	100 km en espacio abierto.
Alimentación	12 VDC.

Tabla 2.4. Especificaciones técnicas de un radio modem comercial típico.

II.2.5. IrDA

Los rayos infrarrojos fueron descubiertos en 1800 por William Herschel, astrónomo inglés de origen alemán. El nombre de infrarrojo significa por debajo del rojo, proviene de que fue observada por primera vez al dividir la luz solar en diferentes colores por medio de un prisma que separaba la luz en su espectro, de manera que el rojo era el que estaba más abajo y el violeta el más arriba.

William Hershel colocó un termómetro de mercurio en el espectro obtenido por un prisma de cristal con el fin de medir el calor emitido por cada color. Descubrió que el calor era más fuerte al lado del rojo del espectro y observó que ahí no había luz.

Las ondas producidas por cuerpos calientes y moléculas son absorbidas por la mayor parte de los materiales. La energía infrarroja, absorbida por un cuerpo, aparece como calor, ya que la energía excita a los átomos de éste, aumentando sus vibraciones o movimiento translacional, lo que resulta en un incremento de la temperatura.
La radiación térmica o infrarroja es un tipo de radiación electromagnética de mayor longitud de onda que la luz visible, pero menor que la de las microondas, tienen longitudes de onda que van aproximadamente de 1mm a $7X10^{-7}$m, ésta última corresponde a la longitud de onda más grande de la luz visible. Los infrarrojos se subdividen en infrarrojos cortos (0,7-5 µm), infrarrojos medios (5-30 µm) e infrarrojos largos (30-1000 µm), Figura 2.14.

El infrarrojo es variadamente usado en las ciencias de la astronomía, meteorología, oceanografía, arqueología, medicina, así como en la industria alimenticia, además se utilizan

para la comunicación a corta distancia entre dispositivos móviles, donde estos últimos cumplen con el estándar IrDA (*Infrared Data Association*), el cual se describirá a continuación.

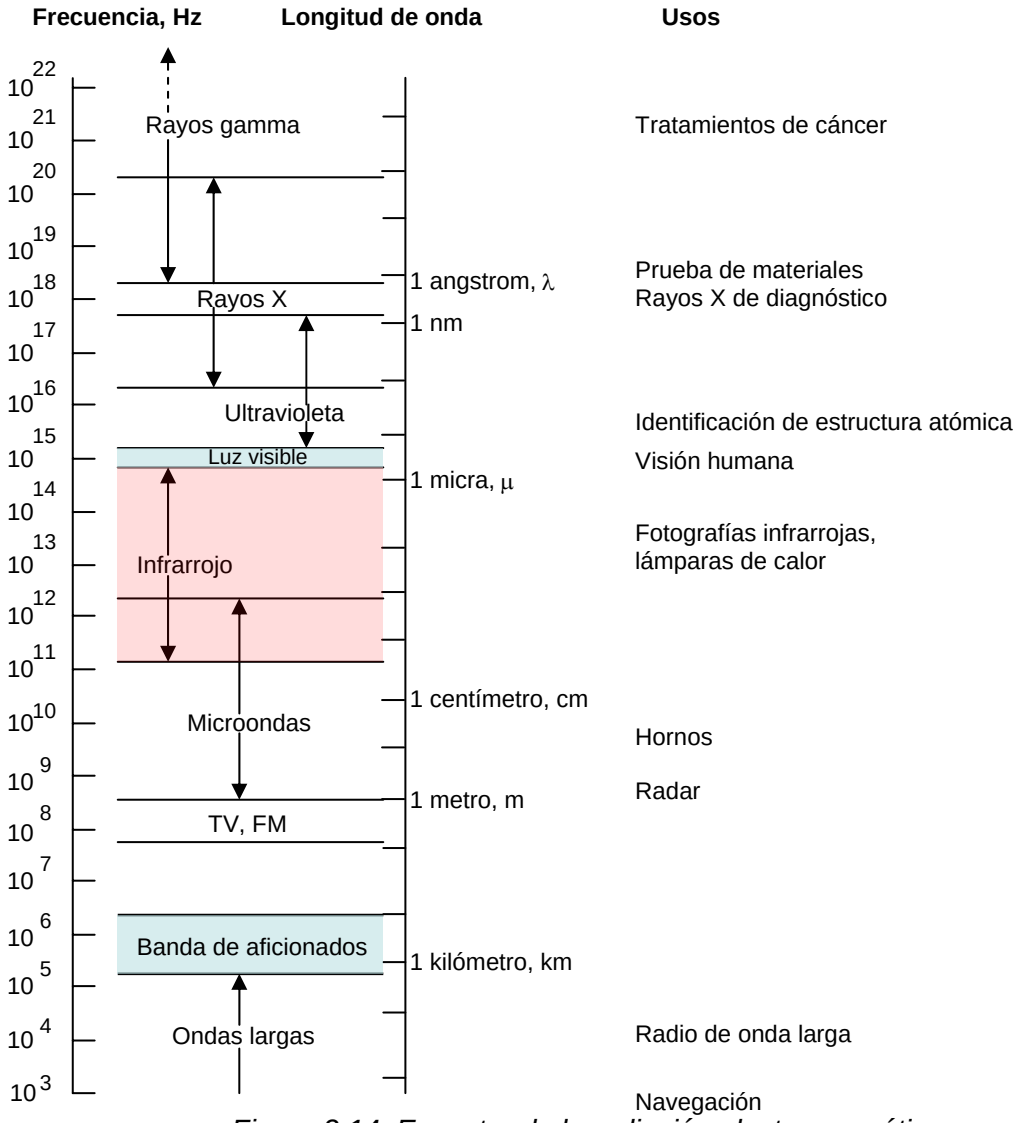


Figura 2.14. Espectro de la radiación electromagnética.

IrDA es una asociación establecida en el año de 1993 por varias empresas, entre ellas HP, IBM y SHARP; el objetivo de esta asociación fue crear las especificaciones y estándares para equipos, además de los protocolos para establecer la comunicación entre ellos.

Para 1995, equipos móviles con puerto IrDA y adaptadores IrDA a PC comenzaron a aparecer en el mercado, al mismo tiempo *Microsoft* presentaba su soporte IrDA para *Windows 95*. Desde entonces la interfaz IrDA ha sido adoptada en la mayoría de los equipos móviles, como son:

laptops, teléfonos celulares, cámaras digitales y PDA. Cualquier equipo que desee obtener la conformidad de IrDA debe cumplir con los protocolos obligatorios; sin embargo, puede omitir alguno o todos los protocolos opcionales, figura 2.15. En cuanto a los protocolos de comunicación, IrDA trabaja con los siguientes niveles: físico, de protocolo y de aplicaciones.

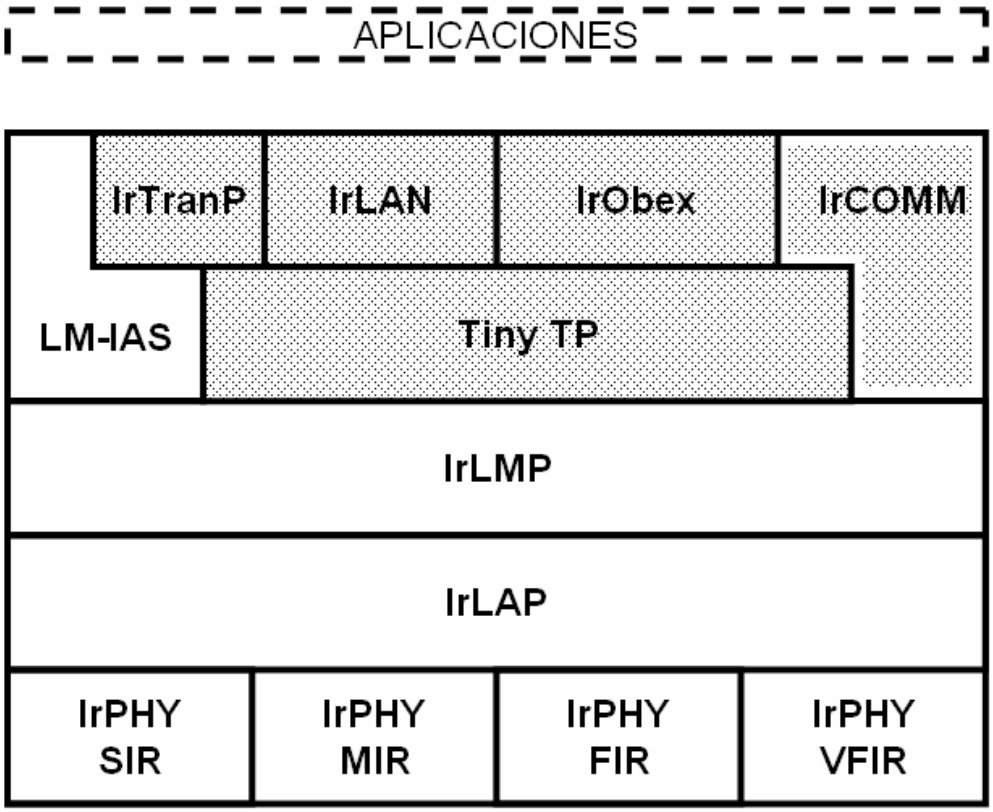


Figura 2.15. Protocolos de datos de IrDA.

Nivel físico

Al nivel físico le corresponde el envío y recepción de cadenas de bits a través del aire, es decir, las características físicas del medio infrarrojo; también es el encargado del entramado de los datos como es el chequeo de redundancia cíclica y la adición de las banderas de inicio y final de la trama; además, especifica el esquema usado en la modulación de los datos.

A nivel físico IrDA considera una UART, un modulador-demodulador y un transmisor receptor compatible con IrDA.

En el estándar IrDA se utiliza la modulación RZI (*Return to Zero Inverted*), para las tasas de transmisión de datos de hasta 1.52Mbps, donde un cero lógico es representado por un pulso de

luz, cuya duración es normalmente $\frac{3}{16}$ de la duración de un bit; para tasas menores o iguales a 115.2 kbps, 576 kbps y 1.152 Mbps, la duración nominal del pulso óptico es ¼ de la duración del bit de la trama.

Las velocidades de comunicación de hasta 115 kbps son referidas como SIR (*Slow infrared*); las de 1.152 Mbps son llamadas MIR (*Medium infrared*); las de 4 Mbps FIR (*Fast infrared*) y la velocidad que pretende alcanzar velocidades de hasta 16 Mbps VFIR (*Very Fast infrared*).

El esquema de codificación RZI, como se muestra en la figura 2.16, se basa en un reloj para manejar la modulación, La frecuencia de reloj es 16 veces la velocidad de transferencia de la comunicación (16xCLK). Por ejemplo, si una comunicación a *115200* bps es requerida, el reloj 16XCLK debe ser establecido a:

$$16 \times 115{,}200 = 1.8432\ MHz$$

Teniendo el *bit* una duración:

$$(1/115{,}200) \times (3/16) = 1.63\mu s$$

Para una comunicación a 4.0 Mbps, el esquema de modulación es conocido como 4PPM (*4 Pulse Position Modulation*). La modulación 4PPM es llevada a cabo definiendo una duración simbólica del dato (*Dt*), generalmente 500 ns, y dividiendo esto en cuatro secciones, llamadas *chips*. Por lo que cada *chip* tendrá una duración de 125 ns.

Debido a que la velocidad de la comunicación es 4 Mbps, cada periodo *Dt* asciende a dos *bits* de datos (*Dt* = 500 ns = 2 x ¼ Mbps). Durante un *Dt* específico sólo un *chip* puede ser un uno lógico.

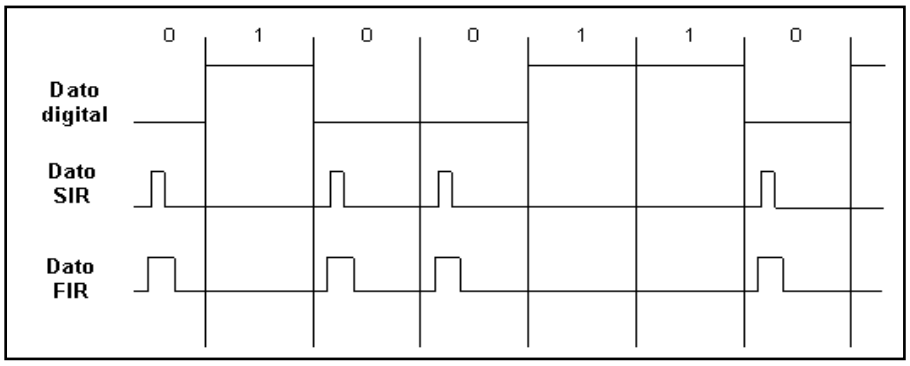


Figura 2.16. Esquemas de modulación IrDA para datos SIR y FIR.

Nivel de protocolo

La capa que se encuentra encima del nivel físico es IrLAP (nivel de acceso de enlace, *Infrared Link Accses Protocol*). Ésta está relacionada con los procesos de control de flujo de datos de bajo nivel, detección de errores de transmisión y petición de retransmisión de los paquetes perdidos.

IrLAP está basada en los protocolos HDLC (*High Data Link Control*) y SDLC (*Synchronous Data Link Control*) con adopciones para las características que se requieren en las transmisiones por infrarrojo y factores del entorno, como pueden ser: las conexiones punto a punto; comunicaciones Half-Duplex, Infrarrojos, interferencia y no detección de colisiones. Las características principales de cada una son:

- *Conexiones punto a punto*. Los dispositivos que se encuentran comunicándose debe estar cara a cara dentro de un margen de más o menos un metro de distancia, para realizar un intercambio de información que los involucra exclusivamente a ellos, es decir, no puede existir un tercer elemento participando en el evento.

- *Comunicaciones Half-Duplex*. Los datos son enviados en uno de los dos sentidos alternándose el turno para transmitir entre los dos dispositivos.

- *Cono angosto de infrarrojos*: La transmisión de infrarrojos es direccional dentro de un ángulo sólido medio de 15 grados, con el objetivo de minimizar las interferencias con dispositivos que se encuentran cercanos.

- *Interferencia*. Son sensibles de las componentes infrarrojos contenidos en luces fluorescentes, el sol e inclusive la luna.

- *No detección de colisiones*: El diseño del hardware es tal, que las colisiones no pueden detectarse. El software utilizado para cada aplicación es quien debe realizar el control de estos inconvenientes.

Las dos componentes de IrLAP que interactúan en una comunicación son: estación primaria y estación secundaria.

- *Estación primaria*. Dispositivo maestro, encargado de enviar los comandos de inicio de conexión y de transferencia, además, garantiza el flujo organizado y controlado de los datos así como el tratamiento de los errores en la transmisión.

- *Estación secundaria*. Dispositivo esclavo, encargado de enviar las respuestas a los requerimientos de inicio de conexión y envío de datos realizadas por el otro extremo.

IrLMP (*IrDA Link Managment Protocol*), pertenece al nivel de protocolo y es el encargado de permitir la multiplexación del flujo de información de diferentes aplicaciones sobre el mismo canal de IrLAP.

LM-IAS, actúa como un directorio que permite determinar para cada tipo de servicio u aplicación disponible un selector de punto de acceso y acceder a información adicional de los servicios.

El nivel de control de flujo, TinyTP (*Tiny Transport Protocol*), es opcional dentro de los niveles de IrDA y tiene asociadas dos funciones: control de flujo sobre las conexiones que se cursan sobre IrLMP y además, la segmentación y reensamblado de los paquetes.

Nivel de aplicaciones

Después de que la interfaz física y de protocolo, el siguiente nivel le corresponde a la interfaz de aplicación, el cual es construir una aplicación de soporte para los dispositivos con los que se requiera comunicar. Los principales métodos de intercambio de datos a través de una conexión infrarroja son: IrCOMM, IrOBEX e IrTRANP.

El estándar de IrCOMM se desarrolló para ofrecer el uso de las interfaces de los dispositivos móviles sobre la tecnología IrDA; el objetivo de IrComm dentro de la arquitectura de IrDA es permitir que las interfaces seriales y paralelas de los dispositivos periféricos puedan operar a través de infrarrojos, sin ningún cambio, es decir, contar con la capacidad de reconocer al dispositivo como un modem. Aunque existen marcadas diferencias en el envío de las señales, ya que en las seriales existe un camino individual para cada señal, mientras que en la interfaz IrDA, tiene un solo haz de luz y todas las señales deben transmitirse a través de este medio, por lo cual es necesario multiplexarlas a través de la capa IrLMP o en la aplicación del usuario.

Existen cuatro tipos de servicios que se definen en IrCOMM.

- *3 Hilos puros*. Emulación de la interfaz serial y paralela para envío de datos únicamente, sin control del canal y soportado enteramente en TinyTP.

- *3 Hilos*. Emulación de la interfaz serial y paralelo con mínimo uso de control del canal y soportado en TinyTP.
- *9 Hilos*. Emulación serial únicamente con control del canal para estado del estándar RS232. Soportado en TinyTP.
- *Centronics*. Emulación paralela únicamente con control del canal para estado de los circuitos de centronics.

El nivel de intercambio de objetos, IrOBEX (*Infrared Objet Exchange*) es opcional. Su función es permitir a dispositivos de diferentes características intercambiar datos y comandos en un modo estandarizado con los recursos presentes en cada uno y, así hacer del intercambio de archivos o mensajes, un procedimiento transparente para la aplicación del usuario.

El protocolo IrTRANP (*Infrared Transfer Picture*) permite al usuario transmitir imágenes entre cámaras, PDA, PC, y aún directamente desde Internet. Sony, Sharp, Casio y algunos otros fabricantes de cámaras digitales que establecieron este protocolo.

IrLAN es el componente de IrDA que permite que los dispositivos con esta tecnología, como computadores, logren acceder a redes de área local.

II.2.6. GSM

La red de telefonía móvil o celular es un sistema telefónico que mediante la combinación de una red de estaciones transmisoras-receptoras de radio (estaciones base) y una serie de centrales telefónicas de conmutación, es posible la comunicación entre terminales de teléfonos portátiles (teléfonos móviles) o entre terminales portátiles (PDA) y teléfonos de la red fija. La esencia de una red celular reside en el uso de múltiples transmisores de baja potencia. El área que necesita ser cubierta se divide en celdas siguiendo un patrón hexagonal que proporciona una cobertura total del área.

En el teléfono móvil, el bucle de abonado se sustituye por un enlace radio; esta técnica fue desarrollada con el fin de incrementar la capacidad de la radio celular. Previamente a la introducción de la radio celular, el servicio de telefonía móvil sobre radio era proporcionado únicamente por un transmisor/receptor de alta potencia. Un sistema típico soportaría en torno a

25 canales con un radio efectivo de alrededor de 80km. La forma de incrementar la capacidad del sistema es utilizar sistemas de baja potencia con un radio más corto y emplear más transmisores/receptores; por lo tanto el dispositivo tiene un equipo de radio capaz de transmitir y recibir información hacia o desde la antena más próxima. Este enlace es el que permite la movilidad del teléfono, permitiendo hacer uso de él desde cualquier punto en el que se reciba señal.

GSM (Sistema Global para las Comunicaciones Móviles, *Global System for Mobile Communications*), formalmente conocida como Grupo Especial Móvil *(*Group Special Mobile*)*, es un estándar mundial para teléfonos móviles digitales. El estándar fue creado por la CEPT (*Conferencia Europea de Administraciones de Correos y Telecomunicaciones, Conférence Européenne des Administrations des Postes et des Télécommunications*) y posteriormente desarrollado por ETSI *(Instituto de Estándares de Telecomunicación Europeos, European Telecommunications Standards Institute*) como un estándar para los teléfonos móviles europeos, con la intención de desarrollar una normativa que fuera adoptada mundialmente. El estándar es abierto, no propietario y evolutivo. Es el estándar predominante en Europa, así como el mayoritario en el resto del mundo (alrededor del 70% de los usuarios de teléfonos móviles del mundo en 2001 usaban GSM).

El celular puede enviar la señal tal cual la recoge, transmisión analógica, pero la capacidad de los sistemas analógicos no es la suficiente para dar servicio al número creciente de abonados. Es ahí donde los sistemas digitales, en general, y el GSM, en particular, proporcionan solución a este problema.

GSM proporciona una serie de servicios añadidos a los de la telefonía fija, tales como: el envío de datos hasta 9.6kbps, sin necesidad de modem externo, ya que contiene una tarjeta PCMCIA para conexión con el puerto serie de la computadora; y el envío de fax grupo 3, gracias a la digitalización de las transmisiones de radio; posibilita, además, la creación de redes privadas virtuales; es compatible con RDSI (Red Digital de Servicios Integrados); permite la identificación de un abonado bajo dos números distintos; ofrece servicio SMS (servicio de mensajes cortos, *Short Message Service*) de hasta 160 caracteres alfanuméricos y toda una completa gama de servicios suplementarios. Además, proporciona los servicios de desvío de la comunicación hacia cualquier otro número de la red móvil, restricción y retención de llamadas, indicación de

llamada en espera, multiconferencia, identificación de la línea llamante, ocultación de la propia identidad, números de marcación fija, restricción de acceso al sistema de comunicaciones móviles Inmarsat[1], consulta a un buzón de voz, indicación del coste de la llamada, fijación del consumo máximo, entre otros. GSM hace uso del espectro radioeléctrico de forma mucho más eficiente, con células más pequeñas y presenta un menor consumo de energía, lo que permite terminales más pequeñas. También ofrece mayor seguridad, al tener acceso por tarjeta inteligente, y permite cifrar todas las conversaciones, para evitar las posibles escuchas en la red. Por otra parte, una facilidad muy útil que ofrece GSM, es el de disponer de una agenda electrónica en la propia terminal, y una lista de hasta 16 redes de operadores extranjeros preferentes.

Arquitectura de una red GSM

La arquitectura de la red GSM está dividida en tres sistemas: (Figura 2.17.)

- Sistema de conmutación
- Estación base
- Operación y mantenimiento

Las funciones relacionadas con el proceso de llamadas y abonados están implementadas en el sistema de conmutación, mientras que las funciones relacionadas con la radio se concentran en el sistema de estación base.

El MS *(*Estación Móvil*, Movil Station),* es la terminal de usuario/teléfono móvil, que se comunica con la red a través de un interfaz radio.

Cada uno de estos sistemas contiene una serie de unidades funcionales operativas, las cuales realizan las operaciones que el sistema GSM es capaz de proporcionar. Dichas unidades son: el sistema de operación y mantenimiento, la estación base, y por último el sistema de conmutación; entre sus características principales se encuentran:

Operación y mantenimiento

- *OSS (*Sistema de Operación y Mantenimiento*, Operations Support System).* Éste proporciona los medios necesarios para poder llevar a cabo una eficiente gestión de la red, tanto de la parte de conmutación como la de radio. Las principales funciones que

[1] Sistema satelital

realiza son: gestión de la red celular, administración de abonados, gestión de averías y medidas de funcionamiento de la red de conmutación y radio.

Estación base

- *BTS (*Estación Transceptora Base*, Base Transceiver Station)*. Ésta contiene los transmisores y receptores para cubrir una determinada área geográfica (una o más celdas).
- *BSC (*Controlador de Estación Base*, Base Station Controller)*. La BSC coordina la transferencia de llamadas entre distintas BTS, con objeto de mantener la continuidad y la potencia con que éstas emiten, para evitar interferencias; también realiza la comunicación con el sistema de conmutación, a través del MSC.

Sistema de conmutación

- *MSC (*Centro de Conmutación de Servicios Móviles*, Mobile Switching Center).* Interconecta a los usuarios de la red fija (RTB (Red Telefónica Básica), RDSI, Iberpac, Internet, RPV (Red Privada Virtual)) con los móviles o a estos entre sí. Mantienen las bases de datos para tratar las peticiones de llamada de los abonados. Las funciones más importantes que realizan son: establecimiento, enrutamiento, control y terminación de la llamadas; gestión de handover (traspaso de llamadas) entre centrales; gestión de servicios suplementarios y recogida de datos de tarificación y contabilidad.
- HLR (Registro de Localización Local*, Home Location Register)*. Este registro es una base de datos donde se almacenan los parámetros de los abonados móviles (información de la suscripción y enrutamiento de llamadas hacia la central donde el móvil está localizado, número internacional de la estación móvil, número del abonado, información sobre teleservicios y servicios portadores, entre otros). Una red GSM puede tener uno o más HLR dependiendo de la capacidad de los equipos y la organización de la red.
- VLR *(*Registro de Posiciones de Visitantes*, Visitor Location Register)*. Este registro almacena toda la información sobre el abonado móvil que entra en su zona de cobertura temporalmente, lo que permite al MSC establecer llamadas tanto terminales como salientes. Cuando un abonado cambia de área de servicio, el nuevo VLR debe actualizar los datos de este abonado y pide a HLR todos los datos necesarios para el establecimiento de llamadas hacia/desde el abonado móvil.

- *AuC (*Centro de Autentificación*, Authentication Center)*. Este centro está asociado al HLR, para proteger la comunicación contra la intrusión y el fraude. Su misión principal es generar tripletas: RAND (número aleatorio), SRES (respuesta) y Kc (clave de cifrado), para el cifrado de la información.
- *EIR (*Registro de Identificación de Equipo*, Equipment Identity Register)*. Es el encargado de controlar el acceso a la red, evitando el empleo de equipos móviles no autorizados.

En la arquitectura GSM, como se puede observar en la Figura 2.17, existen cuatro interfaces de comunicación, las cuales son: MAP, Interfaz A, Interfaz A-bis e Interfaz Um.

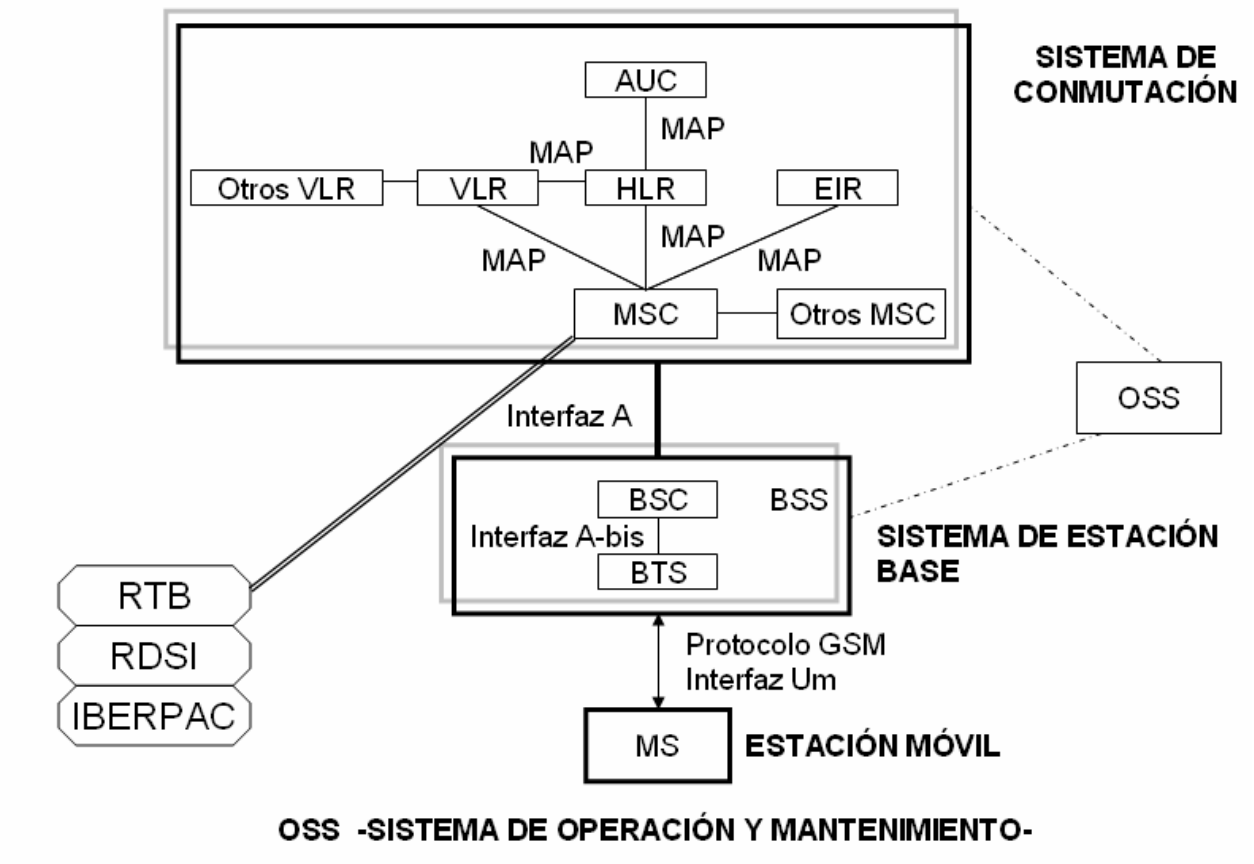


Figura 2.17. Elementos que componen la arquitectura de una red GSM.

MAP

Es un nuevo esquema de señalización desarrollado para redes GSM. La parte de aplicación de móviles (MAP), elaborada por el ETSI, corresponde a los niveles más altos de señalización entre la MSC y las siguientes entidades: los registros de posición (HLR y VLR), el AUC, el EIR y otras MSC. Los niveles de bajos de la señalización son gestionados por la parte de transferencia de mensajes (MTP). Entre MAP y MTP se utilizan la parte de aplicación de las capacidades transaccionales y la parte de control de la conexión de señalización. La señalización debe operar internacionalmente entre las redes GSM y los registros de posición. Esta señalización es adicional al tráfico telefónico convencional, puesto que el GSM a introducido nuevas características, por ejemplo, el seguimiento internacional y la autentificación; también se cuentan con procedimientos como, traspaso de llamada, transferencia de información de abonado y actualización de la posición del abonado móvil en los registros de localización, gestión de los servicios de abonado y transferencia de datos de seguridad.

Interfaz A y A bits

La señalización entre el sistema de conmutación y el sistema de estaciones base (Interfaz A), se realiza según la parte de aplicación del sistema de estación base (BSSAP) dentro del sistema de señalización CCITT N. 7. Algunos procedimientos gestionados por el BSSAP se encuentran los siguientes: asignación y liberación de recursos radio; traspaso de llamada; control de llamada y gestión de la movilidad.

La señalización entre BSC y BTS (Interfaz A-bis) es implementada como un esquema especial de la señalización por canal común para canales PCM de 64kbits/s. Uno de los canales se usa como canal de señalización transportando información de señalización según el protocolo de acceso de enlace sobre canal-D (LAPD) para GSM.

Interfaz Um o Interfaz radio

La interfaz radio es el nombre con el que se conoce la conexión entre la estación móvil (MS) y la estación base (BTS). El sistema de acceso utilizado es el TDMA (Acceso Múltiple por División en el Tiempo, *Time Division Multiple Access*), con una trama TDMA por cada portadora de radio. Cada trama consta de 8intervalos de tiempo (*time slots*) y cada uno de ellos se conoce con el nombre de canal físico.

A través de la interfaz radio es posible enviar una gran variedad de información (datos de abonado, señalización de control, etc.). Dependiendo del tipo de información transmitida, es decir de diferentes canales lógicos que se envía a través de los canales físicos. En cuanto a las características de la señal de radio, se utiliza modulación GMSK (*Gaussiana Minimum Shift Keying*), utilizándose las frecuencias de enlace ascendente 890-915 MHz (de MS a la BTS) y enlace descendente 935-960 MHz (de la BTS a la MS).

GSM cuenta con el servicio SMS donde éste permite el envío de una cadena de caracteres entre las estaciones móviles y una nueva entidad que surge en el sistema con el nombre de Centro SMS. El servicio surge como forma de aprovechar los canales de señalización en los momentos en que no tienen tráfico. Los aspectos que caracterizan al servicio de SMS, son:

- El centro SMS implementa un servicio de almacenamiento y reenvío.
- La transmisión de mensajes cortos requiere las mismas funciones que la transmisión de voz o datos (establecimiento de canal de señalización, autentificación, etc.)
- Ante la indisponibilidad de la estación móvil, el mensaje es almacenado en el Centro SMS para ser reenviado cuando el móvil este disponible.

Los servicios que ofrece SMS, se pueden llevar a cabo punto a punto o mediante Broadcasting.

- Punto a Punto
    - Mensajes MT (*Mobile Terminated*). Servicio de entrega de un mensaje desde el SC (Centro de Servicio, *Service Center*) hasta una MS, obteniéndose un informe sobre lo ocurrido, este tipo de mensaje pueden ser enviados de forma masiva. Los mensajes MT son también denominados "*Push",* por ser el móvil el que recibe la información, figura 2.18.
    - Mensajes MO *(Mobile Originated).* Son los mensajes que se originan del MS, para ser enviados a un SC, obteniéndose un informe sobre lo ocurrido. Los mensajes MO son conocidos también como servicios "*Pull*" por que es el usuario el que solicita el servicio, figura 2.18.
    - La comunicación punto a punto permite la comunicación de mensajes de hasta 160 bytes (alfanumérico y binario).

- Mediante el servicio de broadcasting se hace comunicación de datos desde la red a un conjunto de móviles, permite mensajes de hasta 93 bytes.

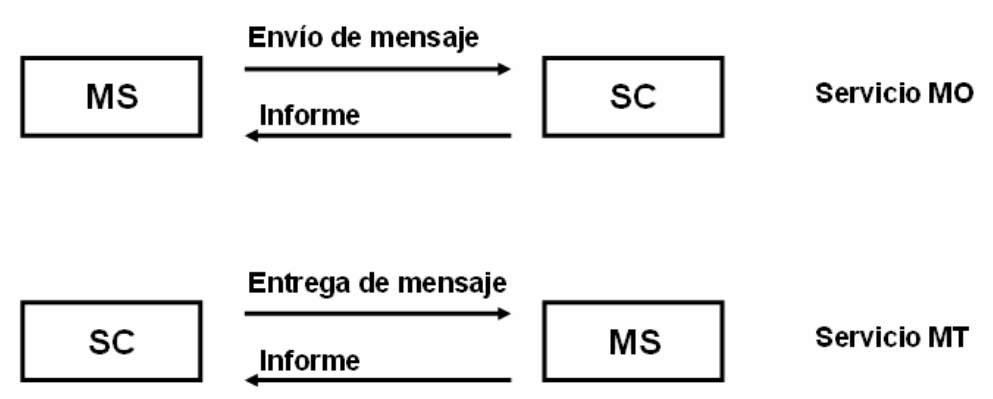


Figura 2.18. Servicios MO y MT.

Las entidades que conforman la arquitectura de la red SMS son las siguientes: SC, MS, MSC, SME (*Short Message Entity*) y el SMSC (*Short Message Service Center*). Este último es el responsable de enviar o almacenar y retransmitir el mensaje desde un SME a un MS; SMS – GMSC (*Gateway Mobile Service Switching Center*), nodo que realiza la transferencia para el servicio SMS –MT; SMS – IWMSC (*Interworking MSC For Short Message Service*) nodo que realiza la transferencia para el servicio SMS –MO y por último HLR y VLR, figura 2.19.

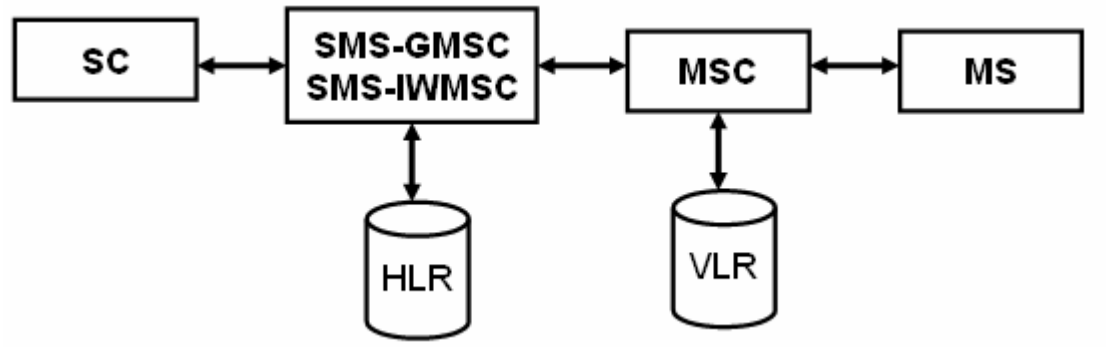


Figura 2.19. Estructura básica de la red de transferencia de SMS.

En la figura 2.20 se muestran las cuatro capas pertenecientes a la arquitectura de la red SMS, las cuales son: *SM-AL (Short Message Application Layer), SM-TL (Short Message Transfer*

Layer), *SM-RL* (*Short Message Relay Layer*) y SM-LL (*Short Message Lower Layers*), las cuales se describen enseguida:

- *SM-AL (Short Message Application Layer).* Corresponde al nivel de aplicación
- *SM-TL (Short Message Transfer Layer)*. Nivel de transferencia el cual su objetivo es la transferencia de un mensaje corto entre un MS y un SC (bidireccional), además de la obtención de los informes de los resultados de la transmisión.
- *SM-RL* (*Short Message Relay Layer*). Corresponde al nivel de repetición y proporciona un servicio que permite enviar un TPDU (*Transfer Protocol Data Units*) a una entidad igual, por lo tanto permite el transporte de mensajes entre varios elementos de la red.
- SM-LL (*Short Message Lower Layers*). Este nivel corresponde al nivel físico del protocolo de datos SMS. Es el encargado del entramado del envío y recepción de cadenas de bits, también es el encargado del entramado de los datos como es el chequeo de redundancia cíclica y la adición de las banderas de inicio y final de la trama.

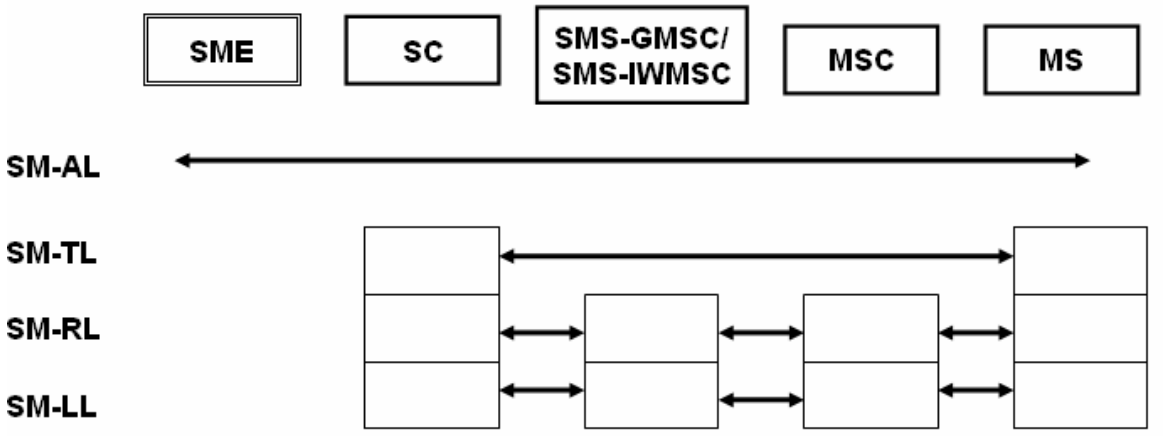


Figura 2.20. Capas de la red SMS.

Por último, cabe destacar que el teléfono móvil se comporta como un modem, y así como el modem cuenta con el lenguaje de control *AT,* descritos anteriormente, también existen comandos para los dispositivos móviles denominados *comandos AT+*. De igual manera son empleados para la realización de las diferentes funciones del dispositivo; también incluyen una tarjeta SIM (*GSM Subscriber Identity Module*) para funcionar, permitiendo gestionar la base de datos de teléfonos, lista de mensajes SMS y envío. Algunos de los comandos *AT+* se presentan enseguida.

AT+CGMI	Identificación del fabricante
AT+CGSN	Obtener número de serie
AT+CIMI	Obtener el IMSI (*Internacional Mobile Subscriber Identity*).
AT+CPAS	Leer estado del MODEM
AT+CREG	Registrarse en una red
AT+CPMS	Seleccionar lugar de almacenamiento de los SMS
AT+CMGF	Seleccionar formato de los mensajes SMS (Modo texto o PDU).

Después de describir los estándares y protocolos de comunicaciones a utilizar en el proyecto, procederemos a realizar el análisis y diseño del prototipo de comunicaciones.

CAPÍTULO III
DISEÑO Y DESARROLLO DE LOS SISTEMAS DE COMUNICACIÓN

En este capítulo se describe el diseño y desarrollo de los diferentes módulos de comunicación que integran al prototipo de comunicaciones, tanto el *hardware* como el *software*. Primero se describen los diseños de *hardware/ software* de los módulos de comunicación alámbricos e inalámbricos. Posteriormente se describe el diseño e implementación del *software* de la estación central, explicando de manera detallada la programación de cada módulo y funcionamiento de dicho *software*.

III.1. Comunicación alámbrica de datos

Las comunicaciones alámbricas son de gran importancia para las estaciones remotas, ya que, gracias a ellas es posible la adquisición, registro y comunicación de datos, en diferentes formas. En particular la comunicación alámbrica de datos se da a través de las interfaces seriales RS232 o USB en distancias relativamente cortas, mencionadas en el capítulo 2. Para distancias muy cortas, del orden de unos cuantos metros, 2 ó 3 ó menor, se hace uso de la comunicación en paralelo, la cual imprime una velocidad de transferencia de datos muy alta (megabytes). Para el caso de la comunicación serial de datos en grandes distancias se hace uso de los

modems telefónicos. Cabe comentar que los dos primeros interfaces se encuentran en equipos PC, Laptops o PDA (Asistente Personal Digital, *Personal Digital Assistant*). El modem telefónico se emplea para al comunicación con una estación central, por medio de una línea telefónica, como se muestra en la figura 3.1.

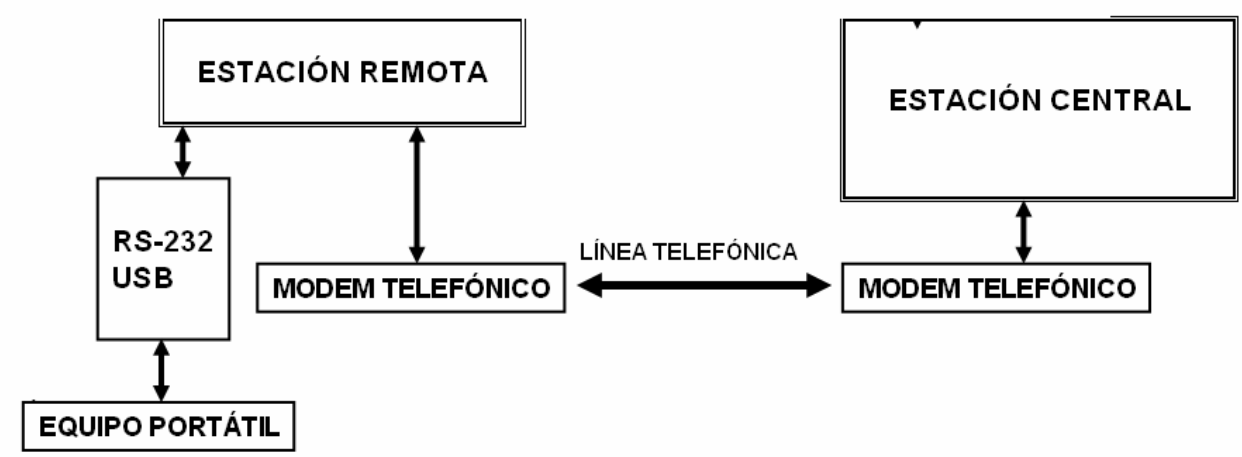


Figura 3.1. Módulos de la arquitectura del sistema de comunicaciones alámbricas.

III.1.1. Interfaz RS232

La interfaz RS232 tiene gran importancia para los módulos de comunicación de la estación remota, ya que son varios los dispositivos que tienen este tipo de conexión, por ejemplo: las PC en general, las Laptops, los teléfonos celulares con tecnología GSM, el modem y el radio modem, como se muestra en la figura 3.2.

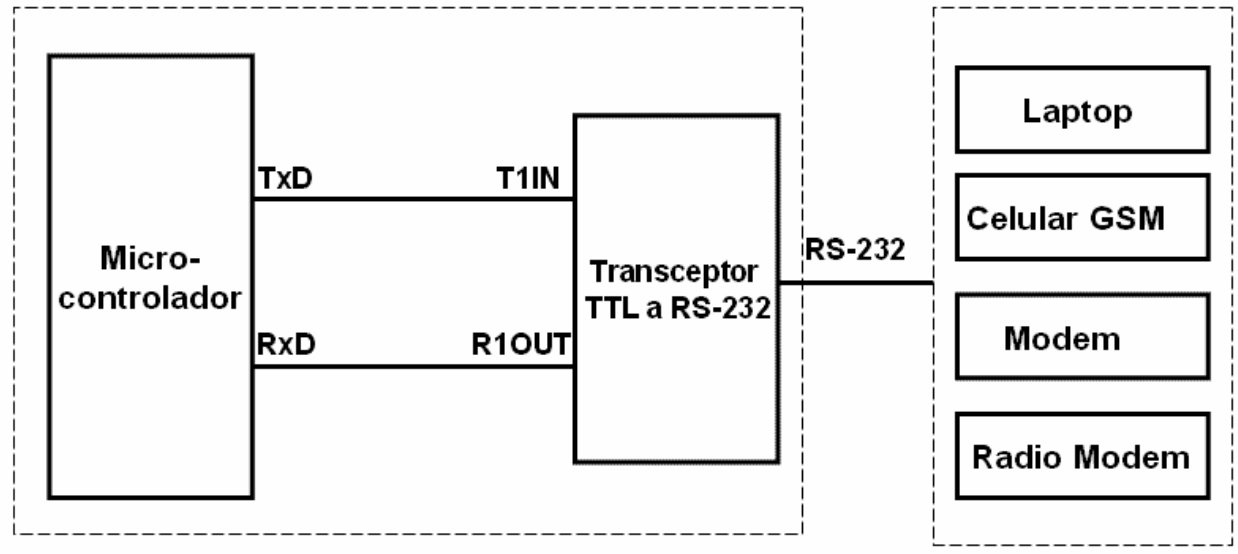


Figura 3.2. Módulos de comunicación que comparte RS232.

En el caso de las estaciones remotas, y en particular para la comunicación de datos, se requerirán principalmente de un microcontrolador y un transceptor; el primero se programará en lenguaje de ensamblador y será capaz de controlar la comunicación serie de la estación remota al dispositivo portátil, con ayuda de un transceptor, el cual convierte los voltajes TTL a RS-232.

Microcontrolador

Para el caso particular del diseño en cuestión, el microcontrolador empleado es el *AT90S8515* de la familia *AVR* de *ATMEL*, el circuito es de tipo *CMOS*, de una familia de microcontroladores RISC de ocho bits, cuenta con una unidad de recepción y transmisión asíncrona UART, mediante la cual es posible implementar la comunicación RS-232. El microcontrolador es el encargado de generar los tiempos necesarios en el ancho de los bits de transmisión de acuerdo a la velocidad deseada. Para el envío de datos se escribe en un registro, y para verificar la información que llega de algún dispositivo externo se lee otro registro de llegada. La velocidad de transmisión se configura en un registro del microcontrolador. De igual forma, la unidad de comunicación serial asíncrona del microcontrolador contará con un sistema de detección de errores de formato y de desbordamiento.

El microcontrolador utilizado cuenta con una memoria interna de datos no volátil de tipo *EEPROM*, en donde se almacenan los datos, por ejemplo: nombre de la estación de operación, datos en ciertos periodos de tiempo, lectura de señales, etc. El espacio de memoria no volátil para datos del microcontrolador permite el almacenamiento de esta información.

El microcontrolador cuenta con el suficiente número de terminales de E/S, para controlar los periféricos (32 líneas de E/S), los cuales se encuentran en cuatro puertos bi-direccionales de E/S (A, B, C y D), cada uno de 8 bits. Para nuestra aplicación son empleados dos puertos, el B y el D.

El puerto B es bi-direccional de 8 *bits* con resistencias de *pull-up* internas. Existen tres localidades de memoria de E/S asociadas al puerto B, los cuales son: registro de datos (PORTB), registro de dirección de datos (DDRB), y registro de entrada en las terminales (PINB). El registro de entrada en las terminales es de sólo lectura, mientras que los registros de datos y de dirección de datos son registros de lectura/escritura. Los registros mencionados son utilizados en el sistema a desarrollar.

Todas las terminales del puerto B poseen resistencias de *pull-up*, las cuales se pueden seleccionar de forma individual mediante *software*; además de que el puerto B tiene funciones alternas, como se muestran en la tabla 3.1.

Terminal	Función Alterna
PB0	T0 (Entrada externa del contador 0)
PB1	T1 (Entrada externa del contador 1)
PB2	AIN0 (Entrada positiva del comparador analógico)
PB3	AIN1 (Entrada negativa del comparador analógico)
PB4	SS (Entrada de selección de esclavo del SPI)
PB5	MOSI (Salida del maestro/entrada del esclavo del bus SPI)
PB6	MISO (Entrada del maestro/salida del esclavo del bus SPI)
PB7	SCK (Reloj del bus serial SPI)

Tabla 3.1. Funciones alternas del puerto B.

Con respecto al puerto D también es un puerto bi-direccional de 8 *bits* con resistencias de *pull-up* internas. Existen tres localidades de memoria de E/S asociadas al puerto D, las cuales son el registro de datos (PORTD), registro de dirección de datos (DDRD), y el registro de entrada en las terminales (PIND). El registro de entrada en las terminales PIND es de sólo lectura, mientras que los registros de datos y de dirección de datos son registros de lectura/escritura.

Todas las terminales del puerto D poseen, igual que el puerto B, resistencias de *pull-up*, las cuales se pueden seleccionar de forma individual mediante *software*. Cuando el puerto D es configurado como puerto digital de E/S, son empleados los registros del puerto D, PORTD, DDRD y PIND. Para llevar acabo la transmisión de datos por medio de la UART son utilizados las terminales PD0 (RXD, terminal de entrada de la UART) y PD1 (TXD, terminal de salida de la UART), éstas son terminales conectadas directamente al transceptor para realizar la transmisión RS-232C. Las configuraciones que se deberán tomar en cuenta en este punto son: velocidad de transmisión/recepción, cantidad de bits por dato, número de bits de parada y si es que se empleará la paridad para detectar errores. Para este módulo de comunicación, la velocidad empleada para el envío y recepción de información es de 19,200 bauds, se emplearán ocho bits de datos, un bit de parada y un bit sin paridad.

El puerto D cuenta con funciones alternas, las cuales se muestran en la tabla 3.2.

Terminal	Función Alterna
PD0	RXD (Terminal de entrada de la UART)
PD1	TXD (Terminal de salida de la UART)
PD2	INT0 (Entrada de la interrupción externa 0)
PD3	INT1 (Entrada de la interrupción externa 1)
PD4	(No tiene otra función)
PD5	OC1A (Salida de comparación A del temp/cont 1)
PD6	WR (Escritura)
PD7	RD (Lectura)

Tabla 3.2. Funciones alternas del puerto D.

La transmisión de datos se iniciará cuando se escriba el dato que será transmitido en el registro de datos de E/S UDR (*UART I/O Data Register*). El dato es transferido desde el registro de datos UDR cuando:

- Un nuevo carácter ha sido escrito hacia el registro de datos UDR, después de que el *bit* de parada del carácter previo ha sido transmitido. El registro de desplazamiento es cargado inmediatamente.
- Un nuevo carácter ha sido escrito al registro de datos UDR, antes de que el *bit* de parada del dato previo haya sido transmitido. El registro de desplazamiento es cargado cuando el *bit* de parada del carácter que está siendo actualmente transmitido es enviado.

Cuando un dato es transmitido desde el registro de datos UDR hacia el registro de desplazamiento, el *bit* UDRE (*UART Data Register Empty*), que indicará que el registro de datos se encuentra vacío, es puesto en un nivel lógico de uno. Cuando este bit se encuentra en nivel lógico uno, la UART se encuentra lista para recibir el siguiente carácter.

En la tabla 3.3 se resumen las características del microcontrolador empleado.

Características	AT90S8515
Memoria de programa *FLASH*	8 k *Bytes*
Memoria de datos *RAM*	512 *Bytes*
Memoria de datos *EEPROM*	512 *Bytes*
Puertos de E/S	32
Conversor analógico/digital	8 canales 10 *bits*

Tabla 3.3. Características del circuito integrado AT90S8515. (Continúa)

Comparadores	1
Temporizadores/WDT	1 de 8 *bits* 1 RTC de 8 *bits* 1 de 16 *bit*s 1 WDT
Entradas salidas seriales	UART I^2C SPI
Velocidad máxima de operación	8 MHz
Multiplicador	9 *bits*, 8x8
Modos de bajo consumo	4
Consumo de corriente @ 3 V 25° C	@ 4 MHz Activo 5mA Modo de espera 1.9 mA Bajo consumo < 1 µA
Voltaje de operación	2.7 V a 6.0 V
Número de instrucciones	118

Tabla 3.3. Características del circuito integrado AT90S8515.

Transceptor TTL a RS232

Para llevar a cabo la transmisión de datos entre el microcontrolador y la computadora se tiene que resolver que los niveles lógicos TTL del microcontrolador no son compatibles con los niveles lógicos RS232 del puerto serie de la computadora. Para dar solución a este problema se empleará el circuito integrado MAX232, el cual es un circuito integrado que tiene como objetivo convertir los niveles RS232 (+12 V y -12V) a voltajes TTL (0 a +5V) y viceversa, con un voltaje de alimentación de +5V. El circuito integrado contiene internamente dos conversores de TTL a RS232 y otros dos de RS232 a TTL, por lo que es posible manejar cuatro señales del puerto de la computadora. Para el funcionamiento del circuito son necesarios cuatro capacitores externos, para poder ser completamente funcional, como se muestra en la figura 3.3.

En la comunicación de datos se utilizan las terminales TX, RX y GND, del conector DB9 de la computadora. La línea TxD (terminal 3 del conector DB9) del puerto serie RS232 transmite información con niveles RS232, por lo tanto se conecta a una de la terminales (R1IN) del MAX232, para convertir estos niveles a TTL y transmitírselos al microcontrolador a través de la terminal (R1OUT). Del mismo modo la información que transmite el microcontrolador con niveles TTL es enviada a la terminal T1IN del MAX2232 para convertirla a niveles RS232 y poder ser recibida por el puerto serie a través de la terminal RxD (terminal 2 del conector DB9).

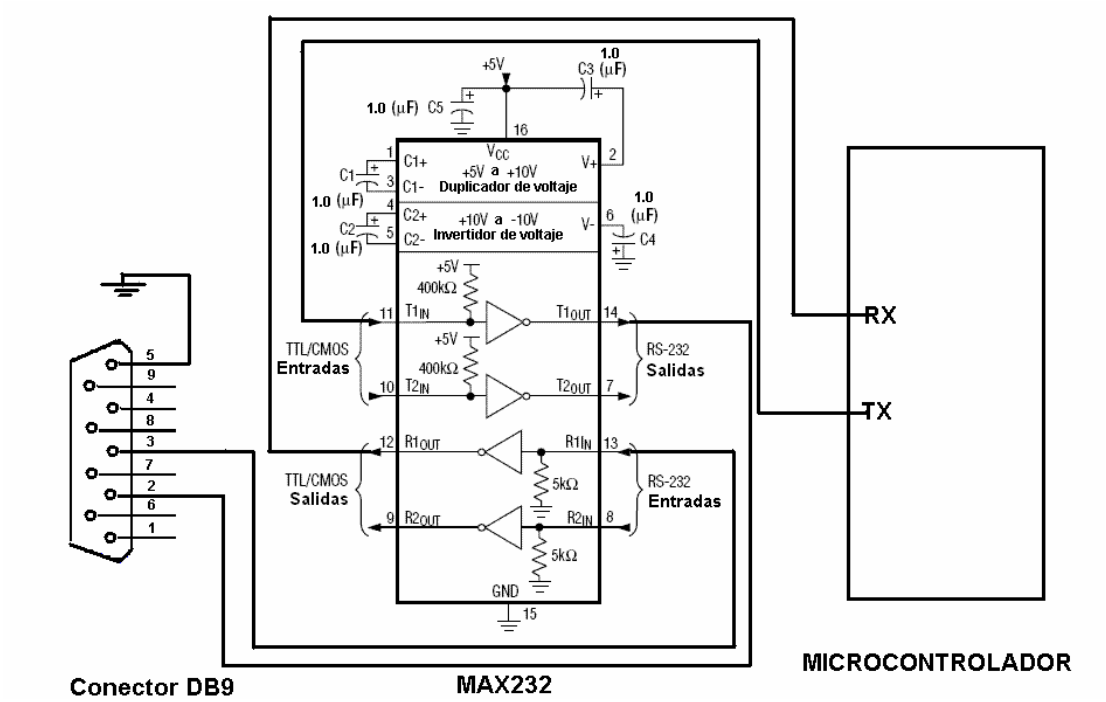


Figura 3.3. Diagrama eléctrico del transceptor con el DB9 y el microcontrolador.

Software de la interfaz

El software del microcontrolador está programado, para que una vez iniciado el sistema configure los puertos correspondientes de entrada y salida, registros necesarios y la UART, para efectuar la comunicación con los diferentes módulos de la estación remota. La comunicación con los módulos serán mediante comandos especiales, como son: RS232 (SR), USB (US), MODEM (MD), RADIO MODEM (RM), IRDA (ID) Y GSM (GS), figura 3.4.

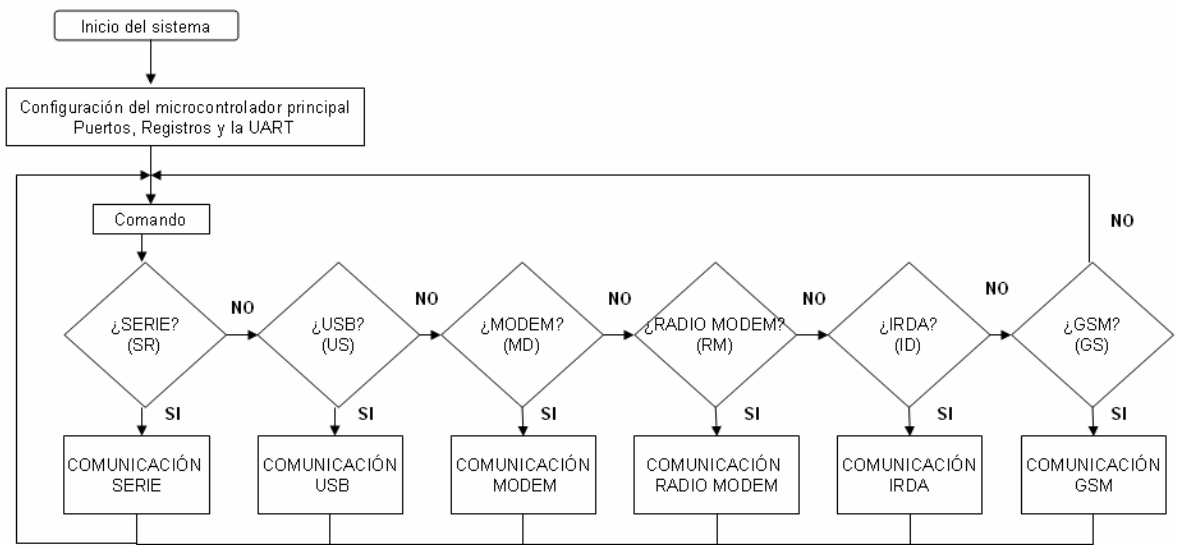


Figura 3.4. Lógica de programación del microcontrolador de la estación base.

En la figura 3.5 presenta el establecimiento de la comunicación serial, la cual realiza una verificación sobre el establecimiento de la comunicación, si ésta aún no se realiza, el sistema espera a realizar dicha conexión. Una vez establecida la comunicación el microcontrolador verificará si el tipo de comunicación se tratará de una transmisión o recepción de información.

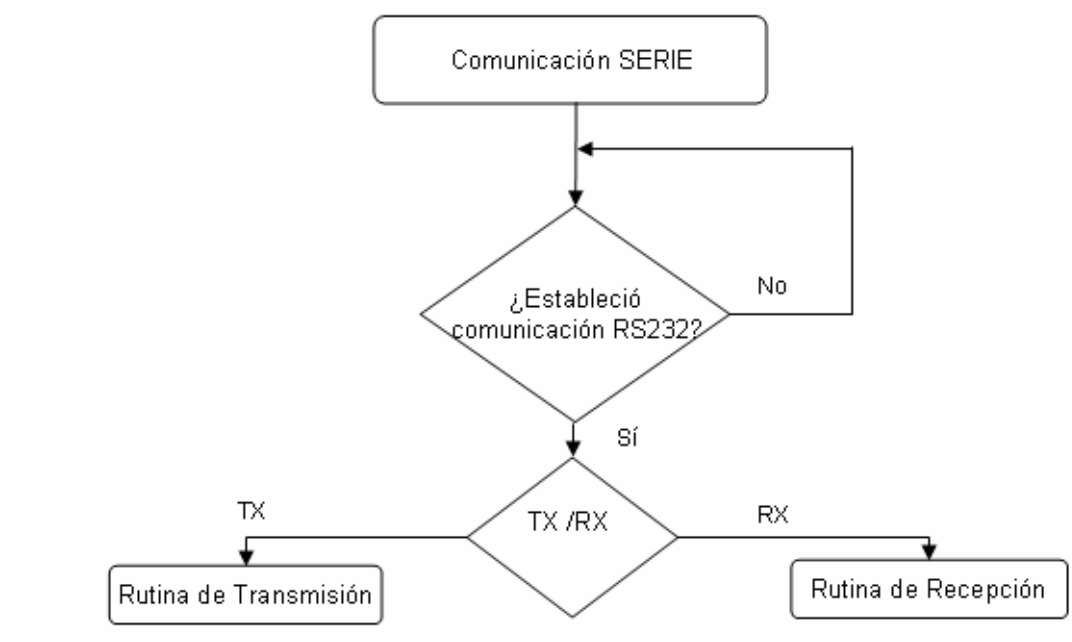


Figura 3.5. Rutina de selección de comunicación RS232.

La figura 3.6 muestra la rutina de transmisión, la cual tendrá como objetivo verificar la comunicación establecida, ya que ésta puede llegar a fallar por cuestiones inesperadas (fallo en el cable o una operación indebida en el sistema), si se llegara a dar este caso el sistema se inicializará. De otra manera se espera una bandera (IN), la cual indica el comienzo de una transmisión válida, en caso de que esta bandera no sea la comentada anteriormente, la rutina vuelve a la verificación de la comunicación, para después volver a preguntar por la bandera (IN). Si la bandera fue válida se lleva a cabo la transmisión del byte de información, esta transmisión continuará hasta que exista una bandera (FN), la cual indicará la finalización de la información, cuando esto suceda la rutina volverá a la rutina de comando, la cual se encontrará lista para recibir el siguiente comando de operación para los diferentes módulos.

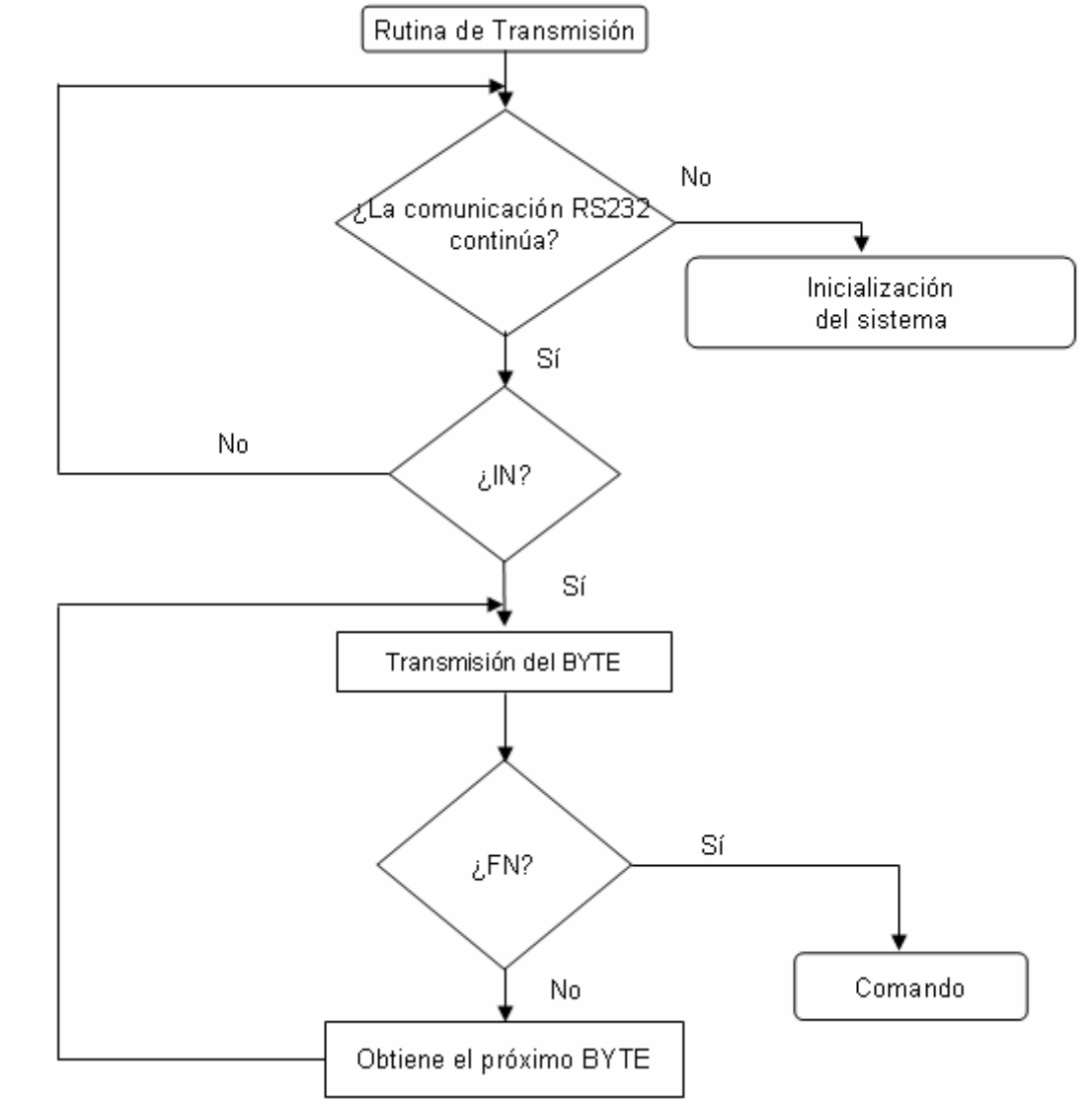


Figura 3.6. Rutina de transmisión RS232.

La figura 3.7 muestra la rutina de recepción, la cual esta diseñada de manera semejante que la rutina anterior, ya que hace manejo de la bandera de inicio (IN), la cual recibida, esperará la información por byte, la cual será almacenada por el microcontrolador; aunque puede suceder que el byte no se reciba por algún error o que llegue información en blanco, en el primer caso el microcontrolador inicializará el sistema, mientras que en el otro caso se volverá a verificar si existe algún otro byte. Existirá otra bandera de fin de recepción (FN), la cual cuando sea afirmativa se regresará a la rutina de comando para preparar la siguiente comunicación.

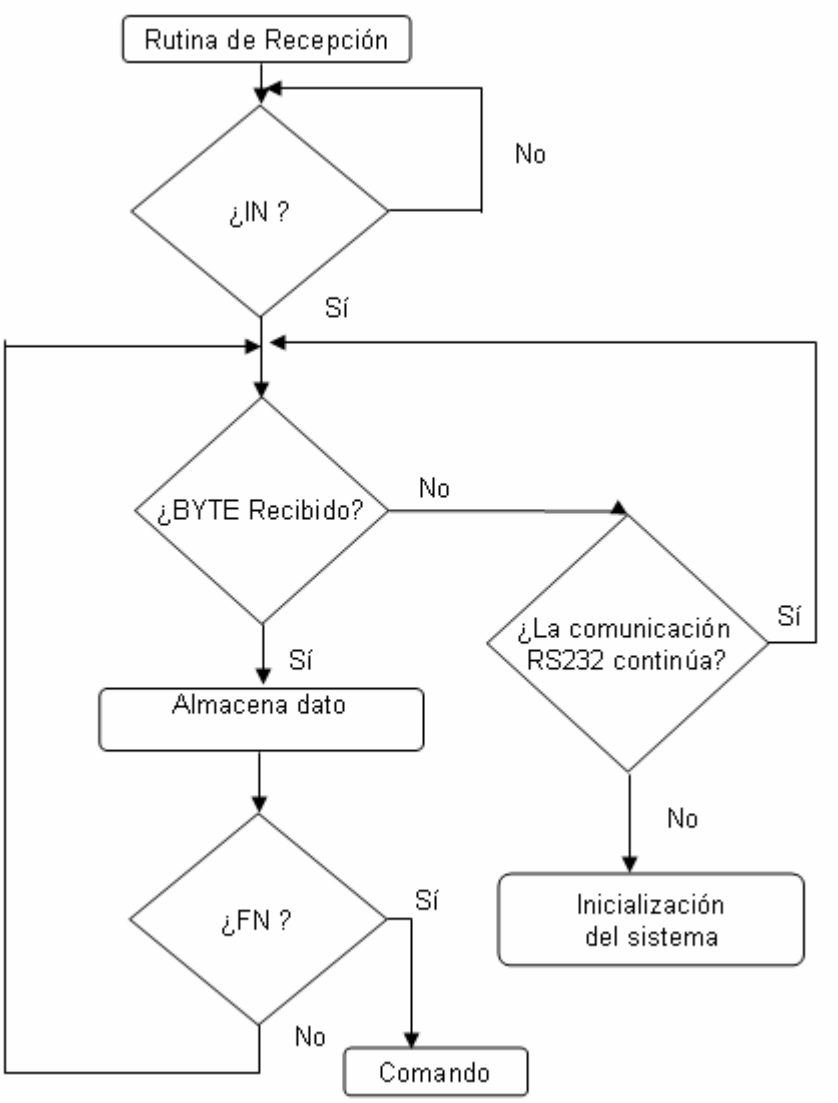


Figura 3.7. Rutina de recepción RS232.

Para el desarrollo y programación del módulo RS232 se utiliza el sistema de desarrollo *AVR STK500,* figura 3.8, el cual es un sistema para realizar pruebas de microcontroladores, que incluye al microcontrolador AT90S8515. La ventaja de utilizar el sistema *AVR STK500,* es que se puede programar el microcontrolador a utilizar en la misma tarjeta, así que evidentemente proporcionará un ahorro de tiempo en el desarrollo del sistema. Las características principales de dicha tarjeta son: contiene la interfaz RS232, tiene sockets de 8, 20, 28 y 40 terminales para dispositivos AVR, 8 botones y 8 leds de propósito general, puertos de I/O, alimentación regulada de 10 a 15V.

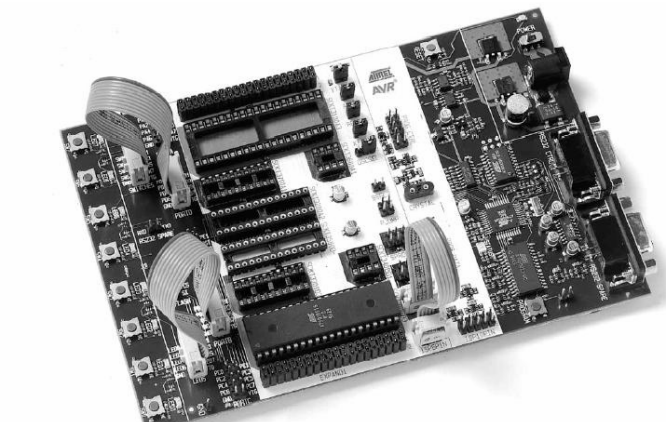


Figura 3.8. Sistema de desarrollo AVR STK500.

En la figura 3.9 se puede observar el diagrama de bloques de *sistema* de desarrollo.

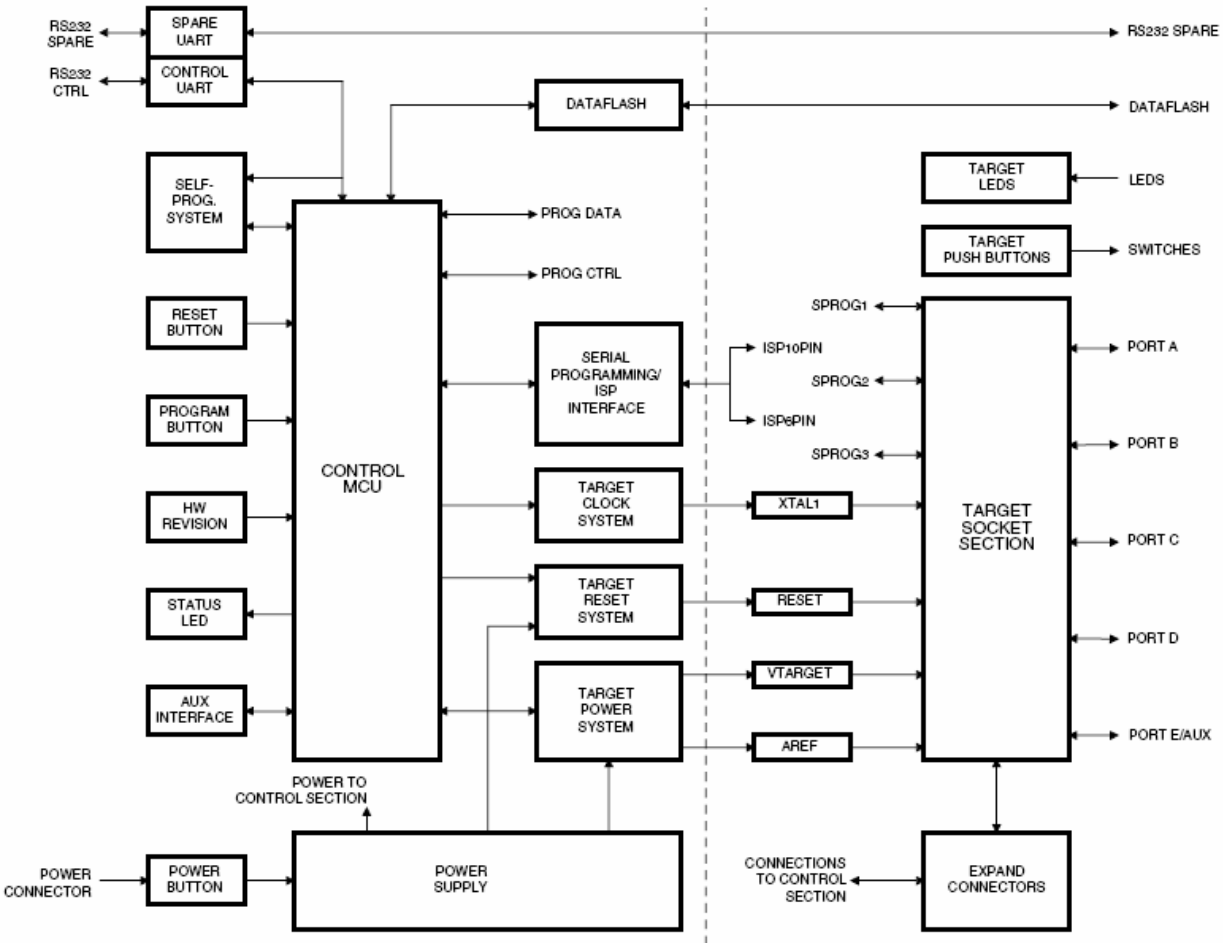


Figura 3.9. Sistema de desarrollo AVR STK500.

En lo que respecta a la programación del microcontrolador, el siguiente código muestra la configuración de los registros, puertos y la UART. Las instrucciones LDI corresponden a las de carga, mientras que OUT corresponden a la de salida.

```
        LDI     R16,LOW(RAMEND)         ;CARGA EL APUNTADOR DE PROGRAMA
        OUT     SPL,R16                 ; VALOR DEL REGISTRO PARA LA
                                        ; CONFIGURACIÓN
        LDI     R16,HIGH(RAMEND)        ;CONFIGURACIÓN DE REGISTRO PARTE ALTA
        LDI     R17,$FF                 ;CARGA VALOR AL REGISTRO R17
        OUT     DDRb,R17                ;CONFIGURA PUERTO COMO SALIDA
        LDI     R17,$00                 ;ENCIENDE LEDS
        OUT     PORTb,R17
        LDI     R17,23
        OUT     UBRR,R17                ;VELOCIDAD = 9600 BAUDS
        LDI     R17,0B00011000
        OUT     UCR,R17                 ;HABILITA RECEPCIÓN Y TRANSMISIÓN
```

El código siguiente muestra la forma de realizar la verificación del las banderas de inicio y fin.

```
; RUTINA DE TRANSMISIÓN DE INFORMACIÓN

        LDI     R17,'I'         ; CARGA EL CARACTER I
        OUT     UDR,R17         ; TRANSMITE LA A
TX_1:   SBIS    USR,TXC         ; ESPERA A QUE SE TRANSMITA LA I
        RJMP    TX_1            ;  MIENTRAS NO SE TRANSMITA VERIFICA
        SBI     USR,TXC
        RCALL   DELAY           ; ESPERA UN TIEMPO
        LDI     R17,'N'         ; CARGA EL CARÁCTER N
        OUT     UDR,R17         ; TRANSMITE LA N
TX_T1:  SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITE LA N
        RJMP    TX_T1           ; MIENTRAS NO SE TRANSMITA VERIFICA

; RUTINA DE VERIFICACIÓN DE BANDERAS

ESPERA_A1:  SBIS    USR,RXC         ; RECIBE CARACTER DE I
            RJMP    ESPERA_A1       ; VERIFICA HASTA QUE SE RECIBA DATO
            IN      R17,UDR         ; ENTRADA POR EL REGISTRO R17
            CPI     R17,'I'         ; COMPARACIÓN DE LA LETRA I
            BRNE    ERROR_AA        ;SI EXISTE UN ERROR BRINCA AL MANEJO DE
                                    ; ERRORES
            RJMP    ESPERA_T1       ; CONTINUA CON LA TRANSMISIÓN
```

III.1.2. Interfaz USB

Para la realización del puerto de comunicación vía USB, se empleará el conversor RS232 a USB con el circuito integrado *TUSB3410,* que suministra una interfaz entre un puerto USB y un

puerto serial asíncrono. En la figura 3.10 se muestra la arquitectura del circuito antes mencionado. El conversor es de tipo bidireccional y contiene la lógica necesaria para comunicarse con una computadora mediante el puerto USB; esto lo realiza por medio de un microcontrolador 8052 de 8 bits, con una memoria RAM de16 kbytes (que puede ser cargada desde la PC) y con un una memoria ROM de 10 kbytes (permiten al microcontrolador configurar el puerto USB al momento de inicialización). El conversor cuenta con las funciones de decodificación de las señales del USB (USB-RS232 y RS232-USB), configuración de la UART y manejo de errores que son administrados por el *firmware (*programa interno del circuito) del microcontrolador. Es posible reutilizar la metodología de la comunicación RS232 en el microcontrolador AT90S8515.

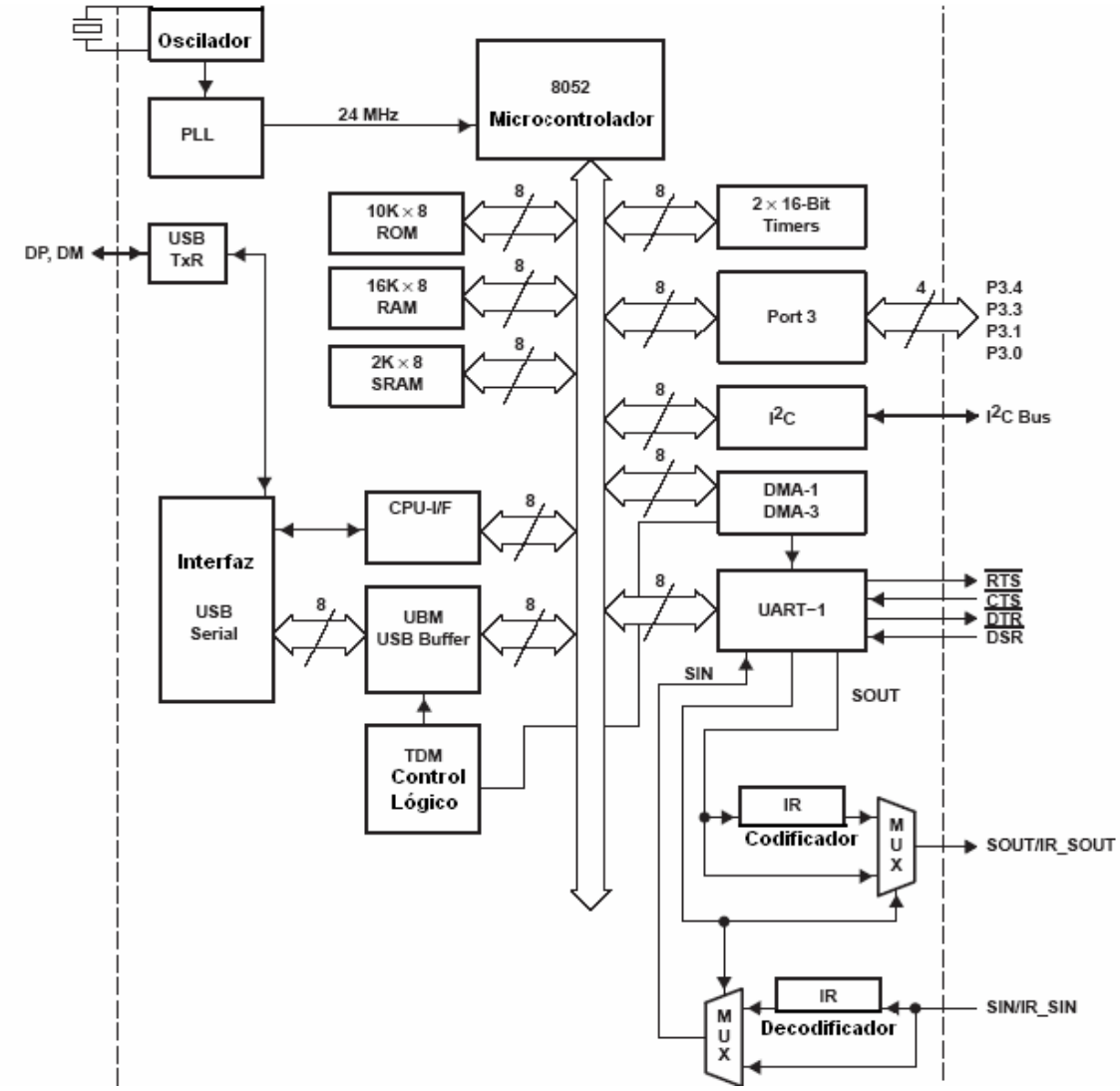
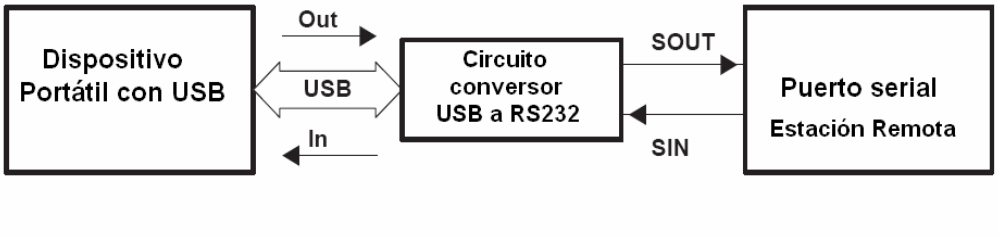


Figura 3.10. Arquitectura del circuito conversor USB a RS232.

Las características principales del circuito integrado son:

- Total compatibilidad con las especificaciones USB 2.0 (12 Mbps).
- Soporte para las operaciones de: suspender, reanudar y reiniciar procesos de manera remota.
- Cuenta con un microcontrolador 8052 de 8 bits.
    - 16K en RAM para espacio de código.
    - 2K en RAM compartida.
    - Cristal a 12MHz.
    - Voltaje de operación de 1.8 voltios a 3.3voltios.
- Flujo de control *Software*/*Hardware*.
    - Programables XON/ XOFF.
    - Auto programable RTS/DTR y CTS/DSR.
- Transferencia de datos en modo IrDA de 115,200bps.
- Baudajes de 50 kbps a 921.6 kbps.
- Funciones de control (CTS, RTS, DSR, DTR, RI, y DCD).
- Rangos de temperatura de operación industriales (−40 °C a 85° C).

El circuito conversor opera de la siguiente manera, figura 3.11(a): el flujo de datos proveniente de la PC, vía su puerto USB, son alimentados a la entrada USB de circuito conversor, y son enviados a través de la línea SOUT al puerto serie de la estación remota. De igual manera, los datos enviados a la PC, a través del circuito conversor, son transferidos desde el puerto serie de la estación remota por la línea SIN. La figura 3.11 (b) muestra la conexión electrónica implementada del conversor en la estación remota.



(a)

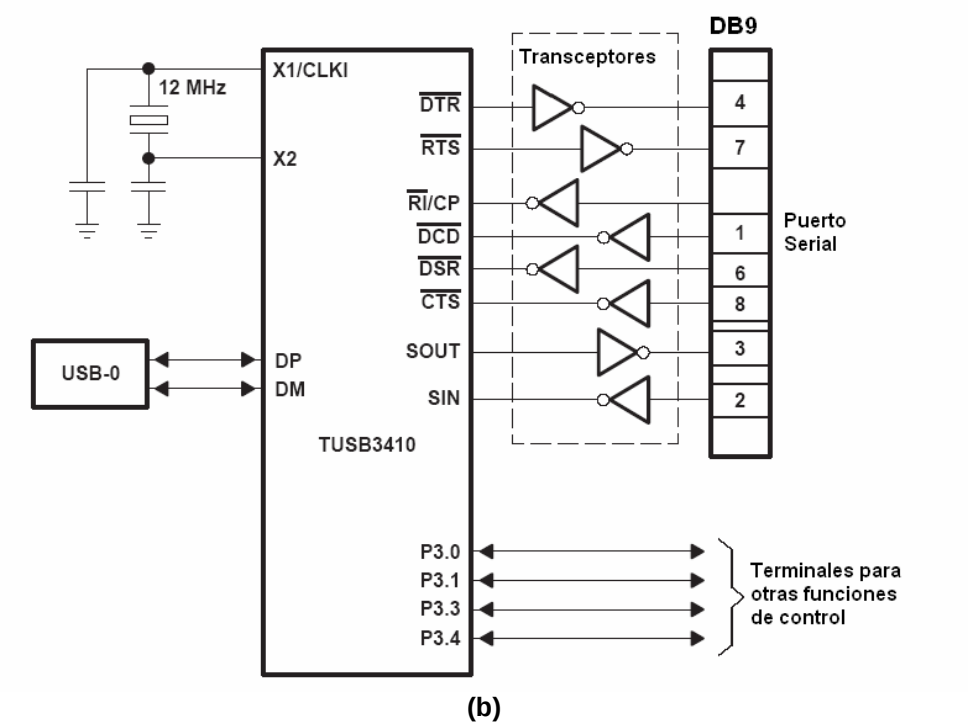


Figura 3.11. (a) Flujo de datos; (b) alambrado electrónico del conversor RS232 a USB.

Los parámetros que necesita el conversor para operar son: RTS, CTS, DTR y DSR, con sus transceptores para los niveles de voltaje, y son configurados por la estación remota.

III.1.3. Interfaz MODEM telefónico

En general, existen básicamente dos tipos de modems telefónicos, los internos y los externos. Los primeros se instalan en las ranuras de expansión libres dentro de la computadora, y los segundos se colocan fuera de ella, conectándose por medio de un cable de comunicación al puerto serial de la computadora.

Para el prototipo de comunicación de datos vía modem telefónico, se empleará un modem externo, ya que se requiere que éste sea transportable de un sistema a otro. En la figura 3.12, se muestra un diagrama a bloques del sistema de comunicaciones vía modem telefónico.

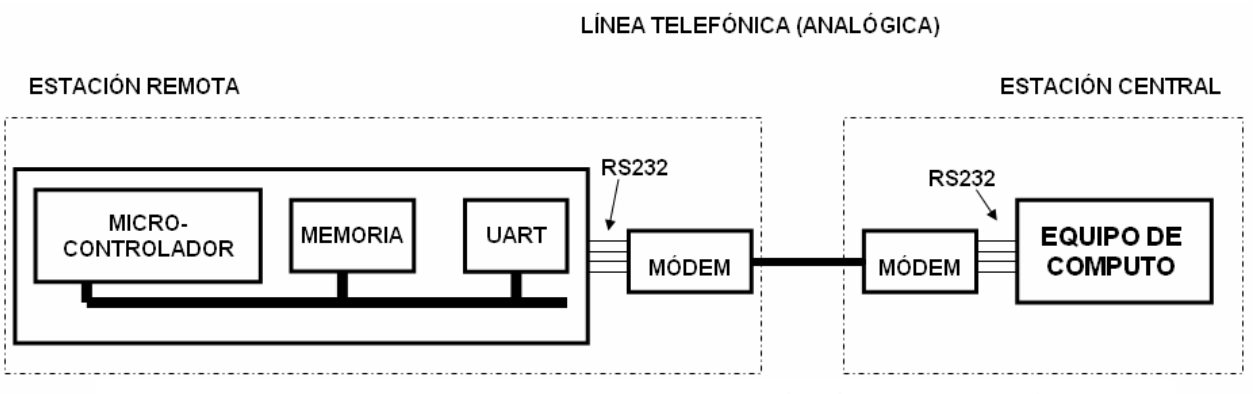


Figura 3.12. Diagrama a bloques de la comunicación vía modem telefónico.

Hardware de la interfaz

El modem externo que se empleó para el desarrollo del diseño antes mencionado fue el modem CN5614XR, figura 3.13. El CN5614XR es un modem externo del alto rendimiento para datos/fax/voz. Está construido con base en el circuito integrado de *Conexant SmartSCM,* que ofrece las últimas innovaciones del diseño y alta velocidad del modem.

Figura 3.13. Modem externo para el módulo de comunicación vía modem telefónico.

En la tabla 3.4 se muestran las características principales de dicho modem, además, de sus especificaciones, tabla 3.5.

Características del modem telefónico
Compatible con Win95/98/2000/NT/ME/XP
Salidas para altavoz y micrófono
Comunicación Full Duplex
56kbps ITU V.90 Data/Fax
Software de comunicación incluido
Estándar de la industria
Cumple c

Tabla 3.4. Características del modem telefónico.

Especificaciones del modem telefónico	
Estándares en modulación de datos	Itu-t V.90, V.34, V.32bis, V.23, V.22bis, V.22a/b, V.21, Bell 212A y 103
Compresión de datos	ITU-V.42bis, Clase 5 De Mnp
Corrección de errores	ITU-V.42 LAPM y MNP2-4
Modulación de fax / estándar de protocolos	V.17, V.29 a 14.000 BPS Clase 1 de EIA
Indicadores del panel delantero	SR, TR, CD, SD, RD, RTS, CTS, OH
Interfaz física	interfaz serial RS-232/V.24
Requisito de Voltaje	Entrada:120 o 230VAC Salida: 9VAC 1000mA

Tabla 3.5. Especificaciones del modem telefónico.

Software de la interfaz

En lo que corresponde a la programación del microcontrolador AVR, se utilizará prácticamente la misma programación que se uso para la comunicación RS232, ya que ambos sistemas comparten el mismo protocolo de comunicación, sólo con la particularidad del empleo de comandos *Hayes* para el manejo del modem telefónico. Por ejemplo, para comenzar la comunicación de un modem telefónico fuente a uno destino, se llevará a cabo la siguiente secuencia: el modem destino verificará su estado, es decir activo o inactivo, este estado puede ser monitoreado mediante el comando *ATA*, el cual será transmitido por una PC o el microcontrolador, según la forma de transmisión que puede ser de estación central a remota ó de estación remota a central. El modem podrá responder de dos maneras: *OK* o *ERROR*, si el dispositivo está activo, la respuesta al comando *ATA* será *OK*, de lo contrario el modem estará inactivo. Cuando el modem está en calidad de activo, se envía el comando *ATH1* para dar línea, si el comando es exitoso, responderá con un *OK*; posteriormente, se marca el número del modem destino con el comando *ATDnúmero*, emitiendo en el teléfono destino un comando *RING*, el cual contestará con el comando *ATA,* para enlazar la comunicación. Una vez que la comunicación es establecida, el modem fuente está configurado con el baudaje de 9600, para llevar a cabo el intercambio de información entre modems. Para concluir con la comunicación se emitirá el comando *+++AT,* el cual concluirá la transferencia de datos. En la figura 3.14 se muestra la secuencia antes descrita.

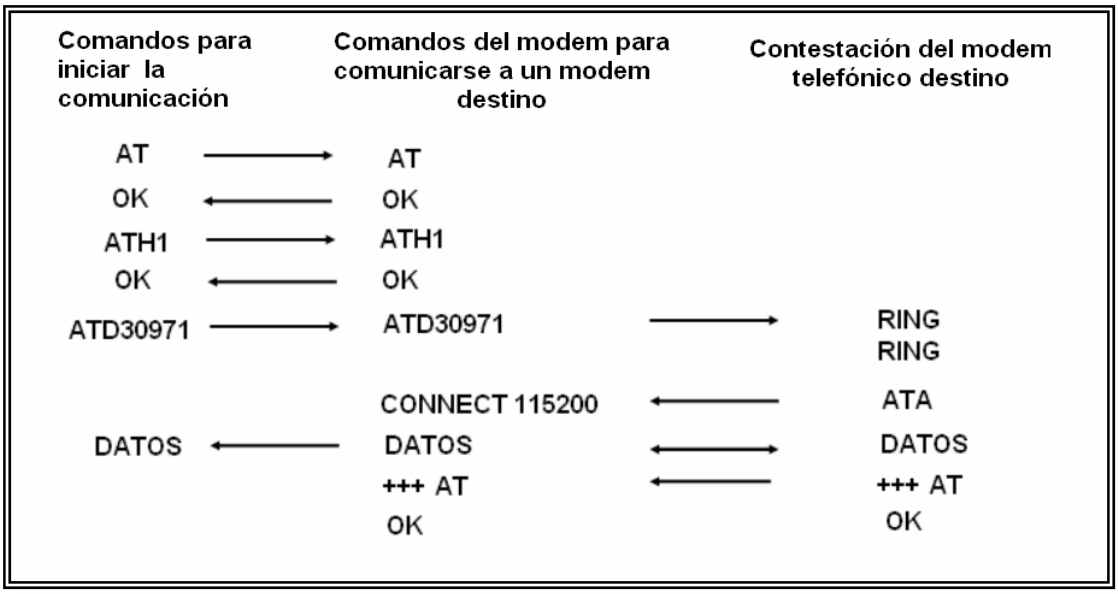


Figura 3.14. Secuencia de pasos para establecer una comunicación vía modem telefónico.

La programación del modem se realizó conforme al diseño de comunicación antes mencionado. En el código que se muestra en seguida se aprecia la secuencia del comando ATA con la verificación de su respectivo OK. En el apéndice B son presentados los códigos completos de la transmisión de datos vía modem telefónico.

```
;*****************************************************************
;COMANDOS TRANSMISTIDOS -ATA PARA ESPERAR LA CONTESTACIÓNL OK-
;*****************************************************************
AGAIN:   RCALL DELAY
                    LDI     R17,'A'
                    OUT     UDR,R17         ;TRANSMITE LA A
TX_A:               SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA A
                    RJMP    TX_A
                    SBI     USR,TXC
                    RCALL   DELAY
                    LDI     R17,'T'
                    OUT     UDR,R17         ;TRANSMITE LA T
TX_T:               SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITE LA T
                    RJMP    TX_T
                    SBI     USR,TXC
LDI                 R17,'   A'
                    OUT     UDR,R17         ;TRANSMITE LA A
TX_T:               SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITE LA T
                    RJMP    TX_T
                    SBI     USR,TXC


;*****************************************************************
;RUTINA DE VERIFICACIÓN DEL COMANDO ATA
;*****************************************************************
ESPERA_A:           SBIS    USR,RXC
                    RJMP    ESPERA_A        ; ESPERA EL CARACTER
                    IN      R17,UDR         ;VERIFICA LO QUE EXISTE EN EL PUERTO DE ENTRADA
```

```
                CPI     R17,'A'         ;COMPARA EL PRIMER CARACTER A
                BRNE    ERROR_A          ;MANEJO DE ERRORES PARA EL CARACTER COMPARADO
ESPERA_T:       SBIS    USR,RXC          ;VERIFICA RX
                RJMP    ESPERA_T        ;ESPERA EL CARACTER
                IN      R17,UDR         ;VERIFICA LO QUE EXISTE EN EL PUERTO
                CPI     R17,'T'         ;COMPARA EL PRIMER CARACTER T
                BRNE    ERROR_T         ;MANEJO DE ERRORES PARA EL CARACTER COMPARADO
ESPERA_A1:      SBIS    USR,RXC          ;VERIFICA RX
                RJMP    ESPERA_A1       ;ESPERA EL CARACTER
                IN      R17,UDR          ;VERIFICA LO QUE EXISTE EN EL PUERTO DE ENTRADA
                CPI     R17,'A'          ;COMPARA EL CARACTER A
                BRNE    ERROR_A         ;MANEJO DE ERRORES PARA EL CARACTER COMPARADO

;*****************************************************************
;RUTINA DE VERIFICACIÓN DE COMANDO OK
;*****************************************************************

ESPERA_O:       SBIS    USR,RXC         ;ESPERA O
                RJMP    ESPERA_O        ;ESPERA EL CARACTER
                IN      R17,UDR         ;VERIFICA LO QUE EXISTE EN EL PUERTO DE ENTRADA
                CPI     R17,'O'         ;COMPARA EL PRIMER CARACTER O
                BRNE    ERROR_O         ;MANEJO DE ERRORES PARA EL CARACTER COMPARADO

ESPERA_K:       SBIS    USR,RXC         ;ESPERA K
                RJMP    ESPERA_K        ;ESPERA EL CARACTER
                IN      R17,UDR         ;VERIFICA LO QUE EXISTE EN EL PUERTO
                CPI     R17,'K'         ;COMPARA EL PRIMER CARACTER K
                BRNE    ERROR_K         ;MANEJO DE ERRORES PARA EL CARACTER COMPARADO
```

III.2. Comunicación inalámbrica de datos

Como ya se mencionó, en la comunicación de datos se contarán con diferentes módulos de comunicaciones inalámbricas para la estación remota, estos módulos de comunicación son: vía infrarroja, radio modem y GSM; cabe destacar que la comunicación se podrá realizar con equipos portátiles y móviles, como lo son Laptops, PDA y teléfonos celulares con tecnología GSM. Figura 3.15.

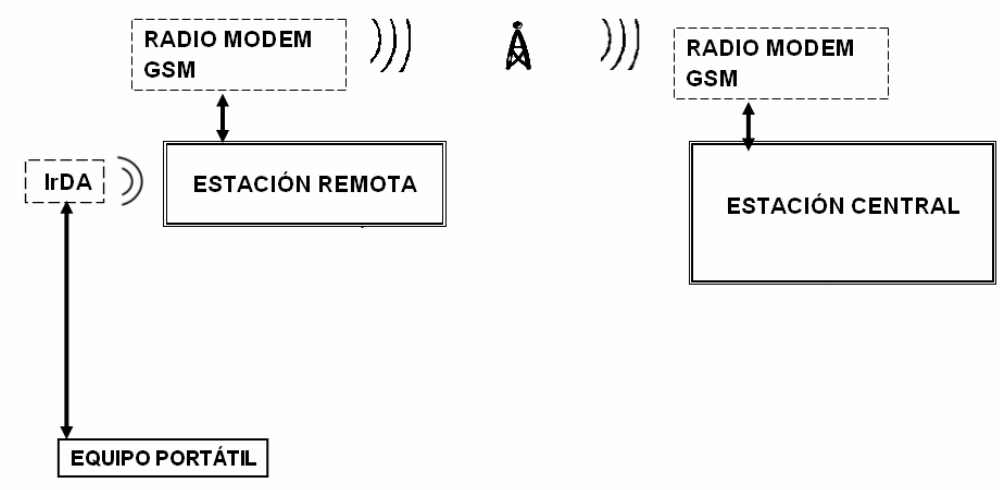


Figura 3.15. Arquitectura del sistema de comunicaciones inalámbricas.

III.2.1. Interfaz IrDA

El prototipo de comunicación vía infrarrojo contendrá elementos de *hardware* y *software*, mediante los cuales permitirá la comunicación de la estación remota a equipos portátiles, Laptop y PDA.

La interfaz IrDA estará constituida principalmente por los siguientes elementos: controlador principal, controlador de comunicación infrarroja y un transceptor óptico, como se muestra en la figura 3.16. Con lo que respecta al controlador principal, éste estará diseñado tanto en hardware como software, mientras que los dos últimos se diseñarán sólo por hardware, por las características propias de los dispositivos.

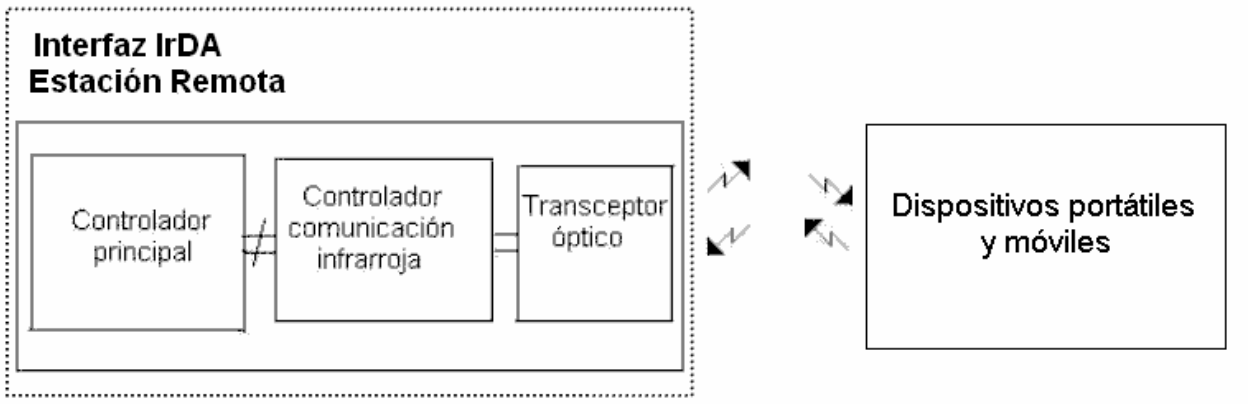


Figura 3.16. Diagrama de bloques de la comunicación vía infrarroja.

Controlador Principal

Como se mencionó anteriormente, el controlador principal se diseñará por hardware y software. Primero se describirá el diseño por hardware, para después describir con detalle la programación del mismo.

Hardware del controlador principal

El *hardware* del controlador principal contendrá un microcontrolador PIC16F877, el cual pertenece a la gama media de la familia de Microchip Las características principales del microcontrolador PIC16F877 se describen en la tabla 3.6.

Características	PIC16F877
Memoria de programa *FLASH*	8 k *Bytes*
Memoria de datos *RAM*	256 *Bytes*
Memoria de datos *EEPROM*	256 *Bytes*
Puertos de E/S	33
Conversor analógico/digital	8 canales 10 *bits*
Comparadores	2
Temporizadores/WDT	1 de 8 *bits* 1 RTC de 8 *bits* 1 de 16 *bit*s 1 WDT
Entradas salidas seriales	UART I^2C SPI
Velocidad máxima de operación	20 MHz
PWM	2
Modos de bajo consumo	4
Consumo de corriente @ 3 V 25° C	@ 4 MHz Activo 5mA Modo de espera 1.9 mA Bajo consumo < 1 µA
Voltaje de operación	2.0 V a 5.5 V
Número de instrucciones	35

Tabla 3.6 Características principales del microcontrolador PIC16F877.

El PIC necesitará para su operación de un reloj de 20MHz, acompañado de dos capacitores cerámicos de 15µF, la alimentación del circuito es de 5V. El microcontrolador se conectará al controlador de comunicaciones infrarrojas, para el control, recepción y transmisión de datos, las terminales empleadas serán las terminales del puerto D, además de las terminales TX y RX para la posible transmisión de datos.

Software del controlador principal

La programación correspondiente que se requiere en la operación del prototipo básico de la estación remota vía infrarroja, se desarrollará en lenguaje ensamblador, al igual que en los otros módulos de comunicaciones. La programación del microcontrolador permitirá controlar el proceso de comunicación asíncrona y el uso de puertos y registros. Además, mediante ésta se controlará el establecimiento, flujo y cierre de la comunicación hacia el circuito de comunicaciones infrarrojas. Entre los parámetros que deben ser configurados se encuentran los siguientes: RTS (*Request to Send*), CTS (*Clear to Send*), DSR (*Data Set Ready*), DTR (*Data Terminal Ready*), CD (*Carrier Detect*) y RI (*Ring Indicador*), estos parámetros son configurados a través de las terminales *del puerto D,* además, el microcontrolador permitirá controlar la

adquisición y almacenamiento de datos, para su posterior envío. En las siguientes líneas de código muestra la programación de la configuración de los puertos necesarios para llevar a cabo la transmisión de datos.

```
;   PORTB
;    Function        LED7  LED6  LED5  LED4  LED3  LED2  LED1  LED0
;    TRIS Direction   O     O     O     O     O     O     O     O
;   PORTC
;    Function        RX    TX    NA    NA    NA    NA    NA    RST2150
;    TRIS Direction   I     I     O     O     O     O     O     O
;
;   PORTD
;    Function        CTS   RTS   DTR   DSR   CD    RI    NA    NA
;    TRIS Direction   I     O     O     I     I     I     I     I
```

En la figura 3.17 presenta el establecimiento de la comunicación IrDA, la cual se activará al presentarse el comando IR en el microcontrolador principal. Antes de establecer la comunicación en su totalidad este medio de comunicación requiere de un tiempo determinado para inicializar el transceptor. Una vez completado el tiempo requerido *se* establecerán las banderas DTR a cero y RTS a uno. Se realiza una verificación sobre el establecimiento de la comunicación, si ésta aún no se realiza el sistema espera a realizar dicha conexión. Una vez establecida la comunicación, el microcontrolador verificará si el tipo de comunicación se tratará de una transmisión o recepción de información.

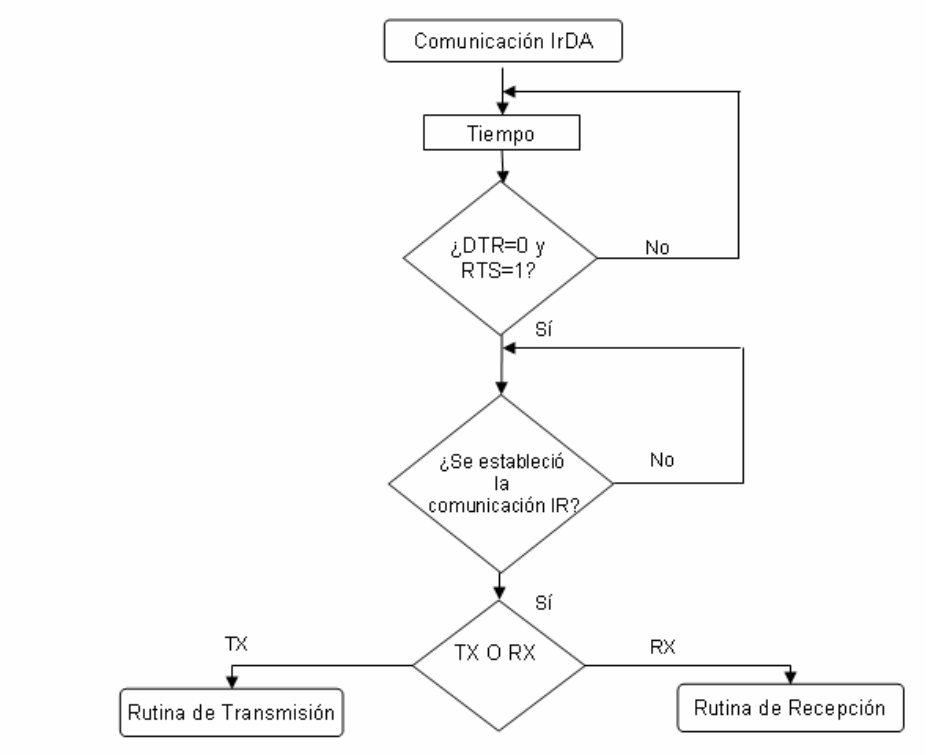


Figura 3.17. Rutina de selección de transmisión o recepción de datos.

En lo que corresponde a la transmisión de datos, figura 3.18, se verifica la comunicación entre la estación remota y el equipo portátil, si se pierde esta comunicación el sistema se inicializa, de otra manera quiere decir que la comunicación continua, en este caso se procede a verificar el estado de la bandera CTS. Si esta bandera se encuentra en estado bajo, nos indica que se está listo para recibir datos, de otra manera se vuelve a verificar si la comunicación continua y realiza nuevamente el proceso antes descrito. Entonces si CTS = 0, comienza la transmisión de datos, los cuales son transmitidos por byte, donde por cada byte se verificará la comunicación del existente entre los dispositivos. Cuando el sistema terminó de transmitir los datos el sistema retorna a la rutina de comando.

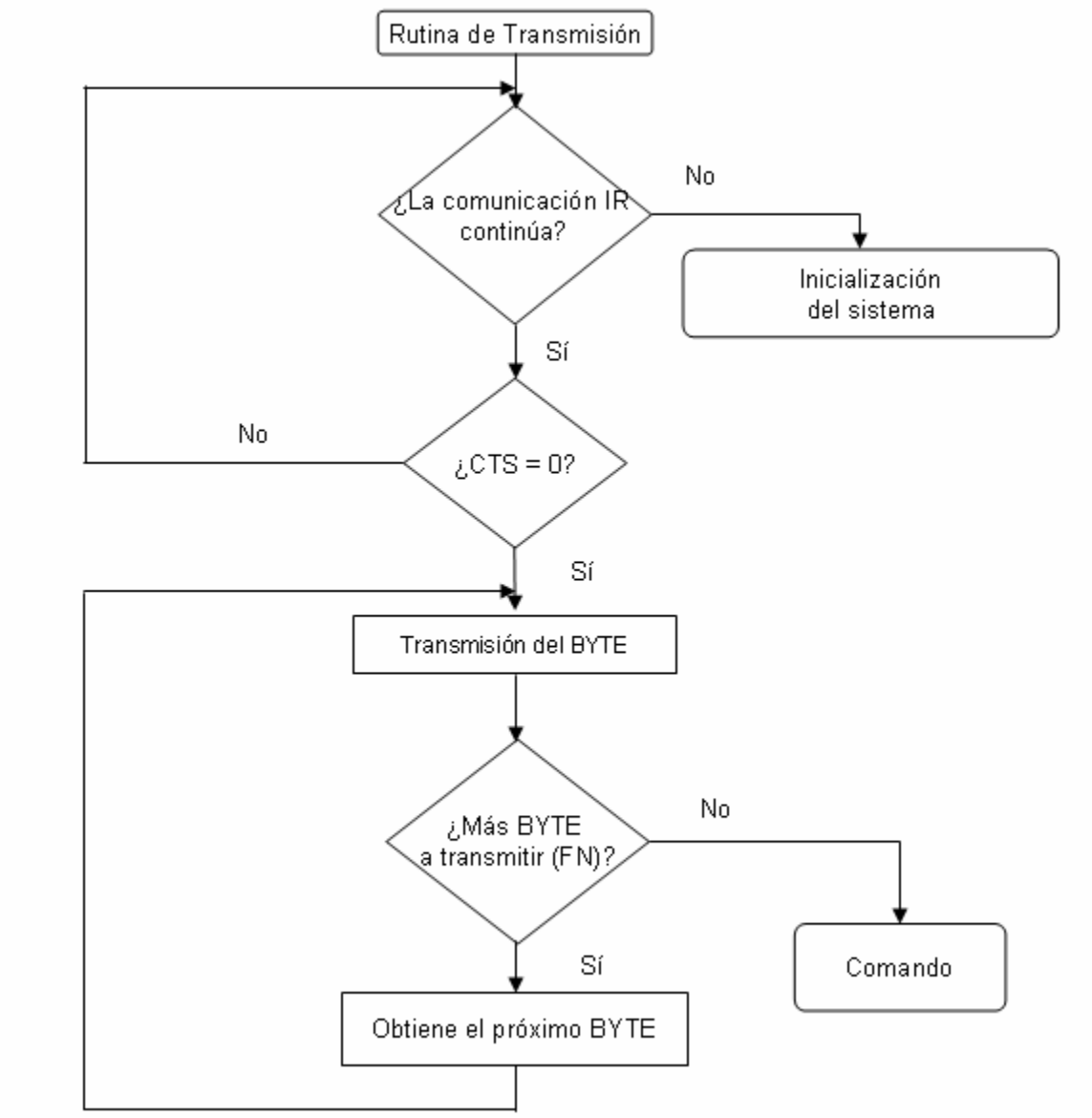


Figura 3.18. Rutina de transmisión.

En el caso de la rutina de recepción, mostrada en la figura 3.19, el estado de la bandera RTS permitirá o no la recepción de datos. Si la bandera cumple la condición RTS=0, se podrán recibir los datos. Posteriormente se preguntará si se ha recibido el primer byte; si es afirmativo es almacenado. Cuando se pregunta por el byte recibido y éste es negativo, se realiza una verificación de conexión entre los dispositivos, el cual puede tomar dos trayectorias: la primera si la comunicación continúa se regresa al ciclo para realizar la pregunta de byte recibido, por si el envió fue caracteres en blanco, en el segundo caso se pierde la comunicación y se inicializa el sistema. Por último, mediante la bandera FN se controla la terminación de la recepción de información; cuando es afirmativa, es decir que terminó la transmisión de datos se regresará a la rutina de comando para iniciar otra comunicación.

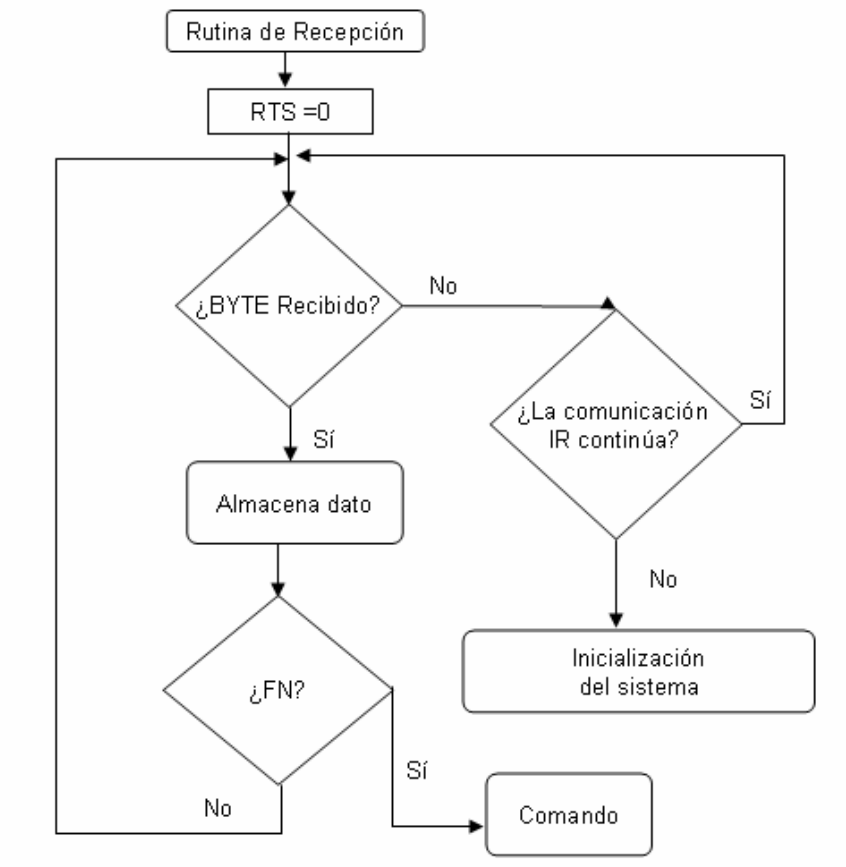


Figura 3.19. Rutina de recepción.

En el siguiente código muestra una parte de la implementación programada del diseño antes mencionado. Se puede apreciar en las líneas de código la configuración de la UART, comparación de banderas (DSR, CD y RTS) para el comienzo de la transmisión de datos.

Enable UART

```
movlw   0x24                    ;CARGA EL VALOR 24 AL ACUMULADOR
movwf   TXSTA                   ; REGISTRO PARA PROGRAMACIÓN DE BAUDAJE
;movlw   B19200at20MHz          ;BAUDAJE DE TRANSMISIÓN
;movlw   B9600at20MHz
movlw   B57600at20MHz           ;ACTIVO  BAUDAJE 57600
movwf   SPBRG                   ; CONFIGURA BAUDAJE ACTUAL AL REGISTRO
clrf    STATUS                  ; BANCO 0
movlw   0x90                    ; ACTIVA EL PUERTO SERIAL
movwf   RCSTA

RESET                           ; SE INICIALIZA EL CONTROLADOR DE COM. INFRARROJAS
bcf     dtr                     ; LA BANDERA DTR = 0
bcf     rst                      ; RUTINA DE INICIALIZACIÓN
bsf     rst

WAIT2150
                btfss   dsr             ;COMPARACIÓN DE LA BANDERA DSR
                goto    WAIT2150         ;ESPERA A QUE CAMBIE LA BANDERA
                goto      MAIN          ;CONTINUA CON ELCÓDIGO

MAIN

WAITCD          btfsc   cd              ; SE CHECA LA BANDERA CD, PARA LA TRANSMISIÓN.
                goto    WAITCD           ; SI CD=1 SE REGRESA A VERIFICAR CAMBIO DE CD

                 bcf    rts             ; SE ACTIVA RTS = 0, LISTO PARA LA TRANSMISIÓN.
RXWAIT1
                btfsc   PIR1, RCIF      ; SE VERIFICA SI EXISTE UN CAMBIO DE BANDERA
                goto    GOTBYTE1        ; COMIENZA LA TRANFERENCIA DE BYTE.
                btfsc   cd              ; VERIFICA LA BANDERA CD, PARA VERIFICAR CONEXIÓN
                goto    MAIN            ;  SI NO EXISTEN MÁS BYTE SE REGRESA A LA RUTINA MAIN
                goto    RXWAIT1         ; SI AUN EXISTE INFORMACIÓN SE VA A RXWAIT1
```

Controlador para comunicaciones infrarrojas

Una vez que el controlador principal de la interfaz IrDA envía los datos al controlador de comunicaciones infrarrojas, éste codifica la cadena asíncrona de datos seriales, conforme al protocolo IrDA y los envía al transceptor óptico. Éste a su vez convierte las señales eléctricas en señales infrarrojas y las envía a los dispositivos con los que desea la comunicación. Las señales infrarrojas recibidas por cualquiera de los dispositivos móviles son decodificadas y tomadas por el receptor del protocolo del dispositivo móvil, para el reconocimiento del dispositivo, el cual también envía bytes de información a la interfaz IrDA, en datos formateados serialmente para la UART, bajo normas requeridas para la transmisión infrarroja IrDA. En el controlador de comunicaciones infrarrojas implementa las siguientes capas del modelo OSI: física, enlace de datos, red, transporte, sesión y presentación.

- *Capa física.* Es la responsable de la definición de las características mecánicas, eléctricas y funcionales de la transmisión y recepción de la información, utilizando un medio de comunicación específico. Entre sus funciones básicas se encuentran la identificación de los datos de los circuitos y el secuenciamiento de los mismos.

- *Capa de enlace de datos*. Mediante esta capa se mantiene la integridad de los datos de una transmisión sobre un canal de comunicaciones, es decir proporciona un canal fiable para la transmisión de datos sobre un medio físico. Entre sus funciones se encuentran las de detección y corrección de errores de transmisión que pudieran ocurrir en el nivel físico.
- *Capa de red*. Esta capa asegura que la información se transmita correctamente a través de la red. Proporciona a las entidades del nivel de transporte una transferencia de datos transparente. En este sentido, libera al nivel de transporte de la necesidad de conocer los mecanismos de transmisión de datos o tecnologías utilizadas para conectar sistemas. Este nivel tiene como funciones la conexión y desconexión de las redes, sincronización y control de flujo de las transferencias y la detección de errores en la transmisión, recuperándolos en caso necesario. En caso de que hubiera más de una red implicada en la transmisión de información, también tiene como función el encaminamiento entre redes. Destacando que si la comunicación se realiza por infrarrojo, esta sólo puede funcionar punto a punto.

- *Capa de transporte.* Proporciona una transferencia de información transparente y fiable de datos entre los puntos finales; además, proporciona procedimientos de recuperación de errores y control de flujo origen-destino.

- *Capa de sesión.* Proporciona el control de la comunicación entre las aplicaciones, establece, gestiona y cierra las conexiones (sesiones) entre las aplicaciones involucradas.

- *Capa de presentación*. Esta capa proporciona a los procesos de aplicación independencia respecto a las diferencias en la representación de los datos (sintaxis).

La operación de estas capas es transparente para el usuario. En la figura 3.20 se muestra el diagrama de bloques del controlador para comunicaciones infrarrojas, así como la conexión que se utiliza en la implementación del sistema.

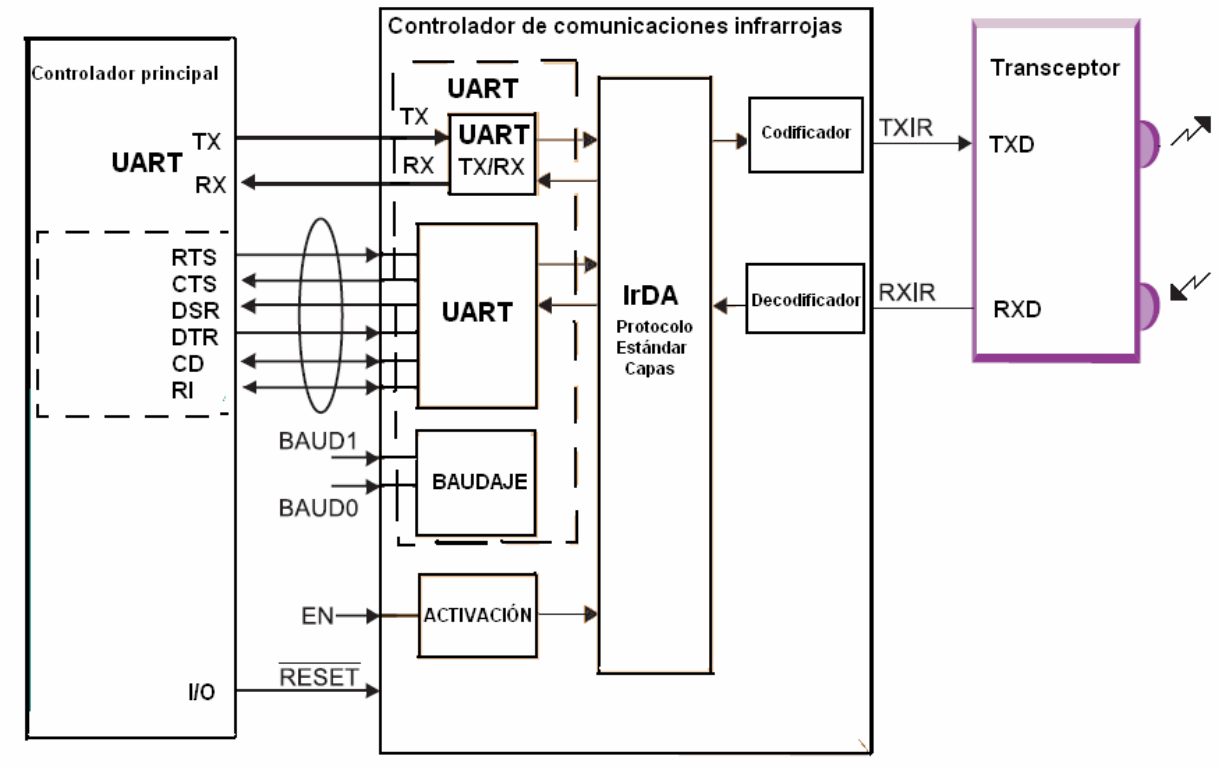


Figura 3.20. Diagrama de bloques del controlador para comunicaciones infrarrojas.

Transceptor Óptico

Para la adecuación de las señales eléctricas a señales infrarrojas se utilizó un dispositivo que cumple con el estándar IrDA, es un modelo fabricado por Agilent Technologies (Figura 3.21). Este dispositivo provee la interfaz entre la lógica y las señales infrarrojas para el establecimiento del envío y recepción de datos en un sistema half duplex, permitiendo una velocidad de hasta 115kb/s, considerando que el microcontrolador y el circuito comunicación infrarroja pueden operar en cuatro diferentes baudajes (9600, 19200, 57600 y 115200). El dispositivo móvil puede operar en cinco diferentes baudajes (9600, 19200, 37400, 57600 y 115200). En lo que respecta a la conexión de las terminales del transceptor y el controlador de comunicaciones infrarrojas, éstas son conectadas con tres terminales del transceptor al controlador de comunicaciones infrarrojas, el cual se encargará de generar las señales para su transmisión y lectura de señales de entrada para decodificar la información.

En cuanto a las características eléctricas del transceptor óptico, éste es un dispositivo que puede trabajar dentro de un rango de voltaje comprendido entre 2.5 y 5.5 V. El circuito contiene un LED de alta velocidad y alta eficiencia, de arseniuro de galio, un fotodiodo y un circuito integrado. Este circuito integrado contiene el manejador del LED, un amplificador y un cuantizador. La siguiente figura muestra el aspecto físico, así como la configuración típica del dispositivo.

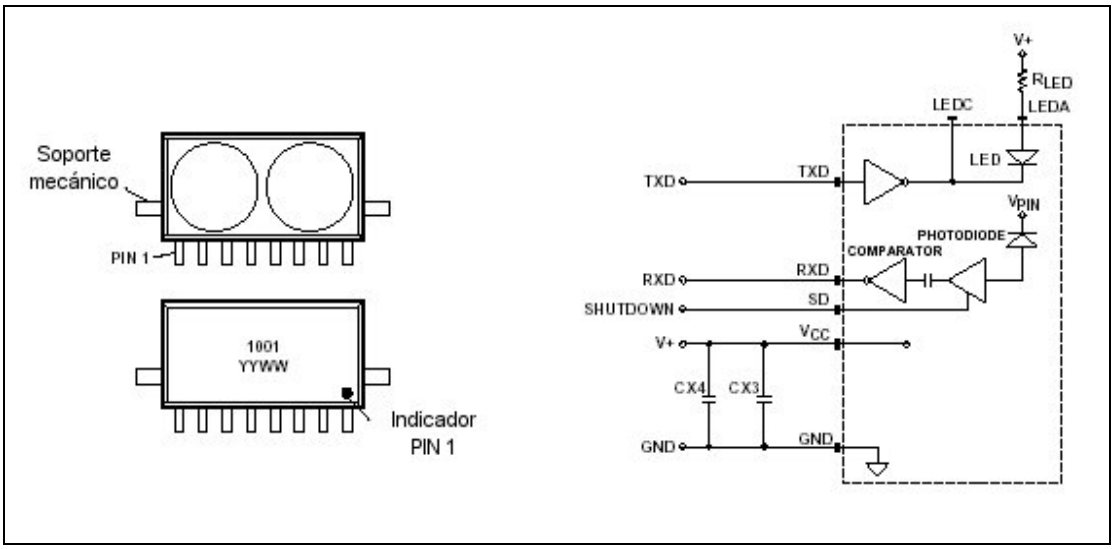


Figura 3.21. Transceptor Óptico.

Para su operación el circuito transceptor requiere solamente de dos capacitores y una resistencia como componentes externos.

Para la transmisión/recepción de datos entre el circuito para las comunicaciones y el microcontrolador, se requieren de este último, tres terminales, una para la recepción de datos (RXD), otra para la transmisión (TXD) y una más para la habilitación del dispositivo (SHUTDOWN), ya que no se encontrará operando en todo momento.

El circuito transceptor cuenta con una terminal de control de la sensibilidad (SC), la cual permite la detección mínima de irradiancia, cuando el nivel lógico colocado en esta terminal sea igual a uno. Cuando el nivel lógico es uno, se incrementa la sensibilidad de las señales infrarrojas y el rango de comunicación de datos aumenta hasta 3 metros. Sin embargo, poner esta terminal en nivel lógico 1, también hace que en transmisor/receptor sea más susceptible de presentar

errores en la transmisión, debido al incremento de la sensibilidad hacia fuentes de luz fluorescente.

Finalmente, en la figura 3.22 se muestran el diagrama esquemático y el circuito impreso diseñados.

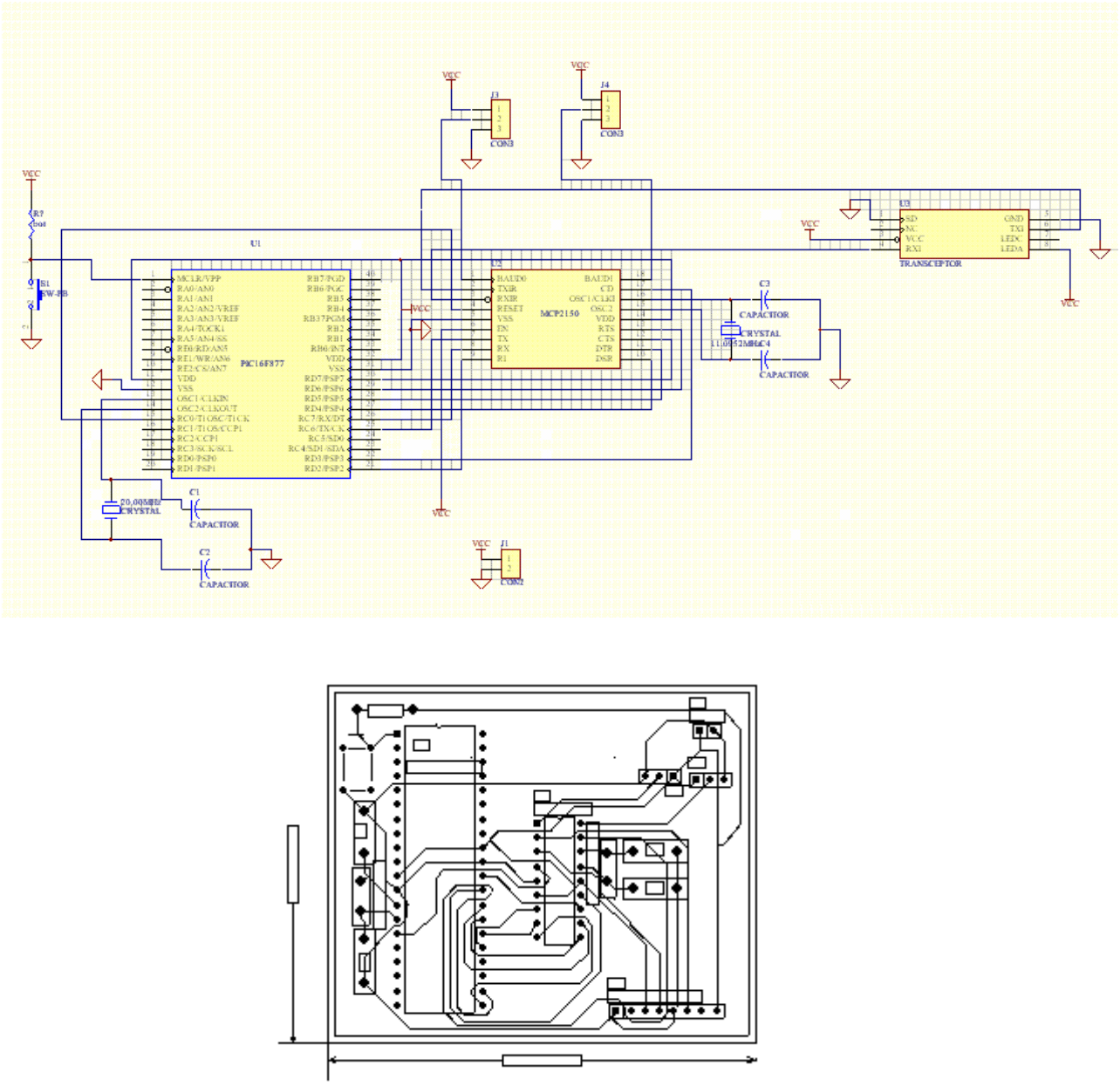


Figura 3.22. Diagrama electrónico y circuito PCB de la comunicación vía infrarrojo.

III.2.2. Interfaz con RADIO MODEM

En lo que corresponde al módulo de la interfaz con radio modem, éste estará constituido por un microcontrolador, transceptor TTL a RS232 y un radio modem, figura 3.23. Para llevar a cabo la comunicación con diferentes estaciones remotas, se utilizarán varios radio modems, los cuales se instalarán en diferentes estaciones remotas con sus respectivos microcontroladores, y serán capaces de enviar/ transmitir información a la estación central.

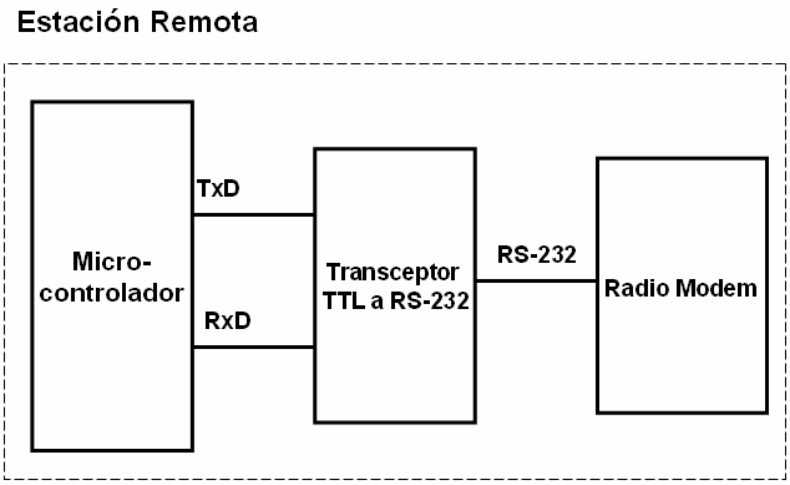


Figura 3.23. Interfaz con radio modem.

Los radios modem pueden configurarse de diferentes maneras, dependiendo el tipo de comunicación que se requiera. Estas configuraciones pueden ser: punto-punto, punto-multipunto, multipunto-multipunto y comunicación por subgrupos. En la figura 3.24 se muestran las diferentes configuraciones para la transmisión de información entre radio modems. Los radio modems podrán ser configurados a través de los comandos *MYID=número* y *TOID=número*. La primera configuración representa el identificador del radio modem, mientras que el comando *TOID=número*, indica entre qué radio modems puede realizarse la comunicación. Para el caso del proyecto se configurarán los radio modem en modo punto a multipunto, es decir, la comunicación entre radio modems se realizará mediante una estación central, la cual podrán transmitir y recibir datos de los diferentes radio modems instalados en varias estaciones remotas.

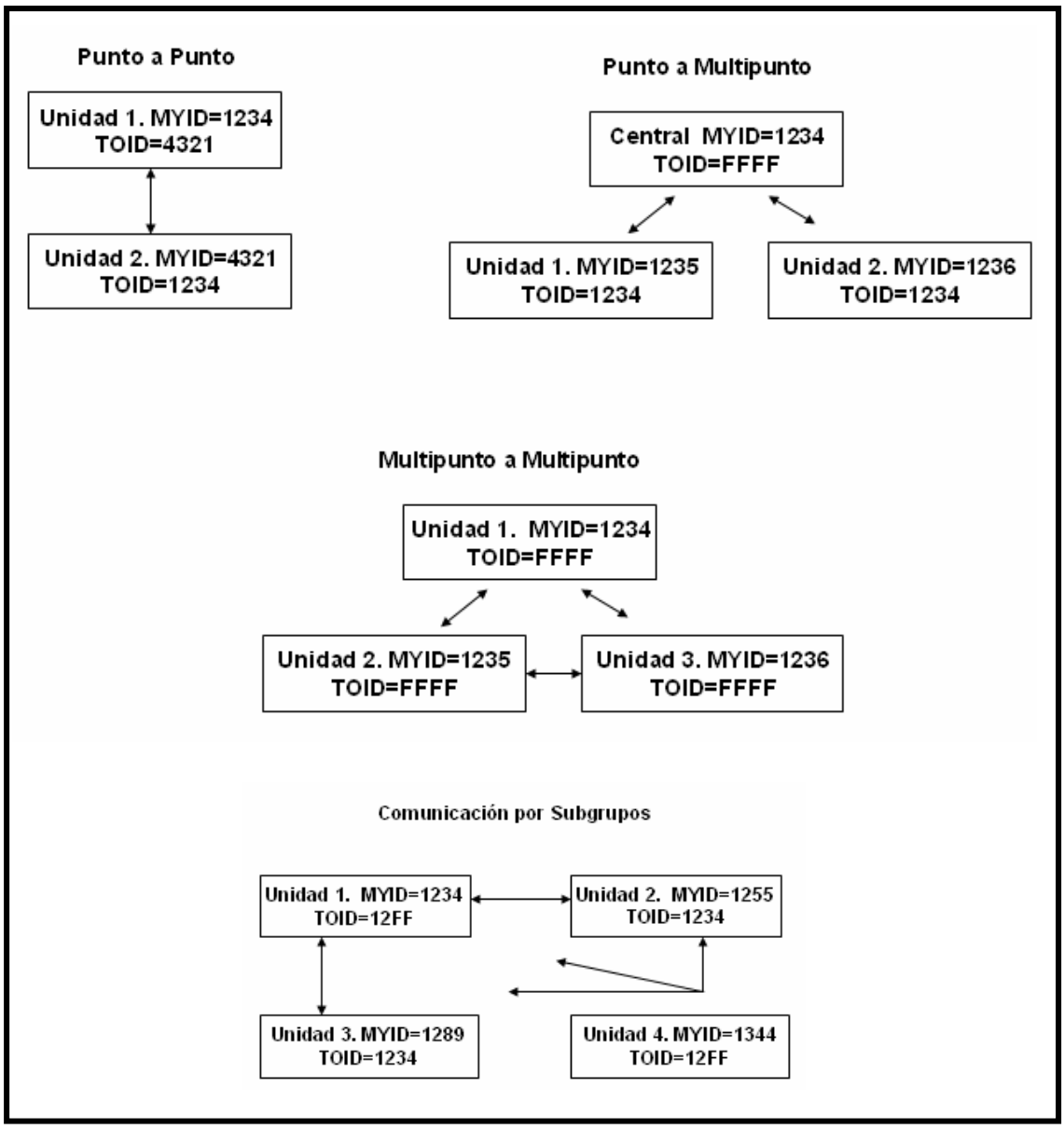


Figura 3.24. Configuraciones para la transmisión de información entre radio modems.

La comunicación del radio modem con el microcontrolador se controlará mediante un programa en lenguaje de máquina, desarrollado para el microcontrolador *AT90S8515* de *AVR* y comentado en el apartado de la comunicación vía modem telefónico. El comando correspondiente para este medio de comunicación será (RM). En el microcontrolador se

empleará el protocolo de comunicaciones RS232 y estará programado para realizar la transmisión y recepción de datos provenientes del radio modem. En la programación para la transferencia de información, mediante el microcontrolador se configurarán los puertos, los registros y la *UART*, para dar inicio de la comunicación radio modem y así poder transmitir la información por bloques de byte.

En la figura 3.25 se muestra la selección que se realizará para el intercambio de información entre diferentes destinos, esta selección puede ser de dos formas, la primera recepción y la segunda transmisión de información.

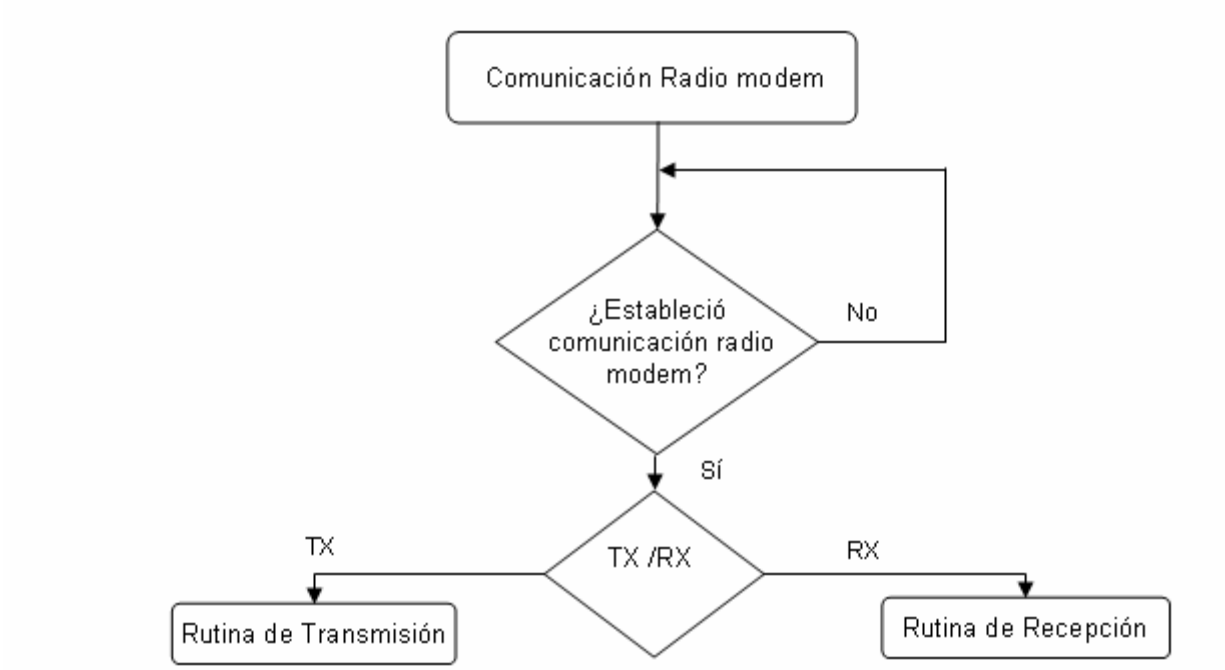


Figura 3.25. Rutina de recepción/ transmisión de datos vía radio modem.

Para llevar a cabo la transmisión de datos se realizará como primer paso la verificación de la conexión con el radio modem, si se pierde el sistema se inicializará, mientras que si continúa se comparará la bandera "IN", la cual indica que es el comienzo de una información válida. Si la bandera no se llegara a presentar, el microcontrolador esperará en un ciclo, verificando la conexión, si la conexión llega a desconectarse se inicializará el sistema del módulo. En caso de que la bandera sea la indicada, el sistema comenzará a transmitir la información, la cual terminará hasta que se presente otra bandera de fin de información "FN". Si es el caso el sistema se regresará a la rutina comando para otra comunicación. Lo anterior se muestra en la figura 3.26.

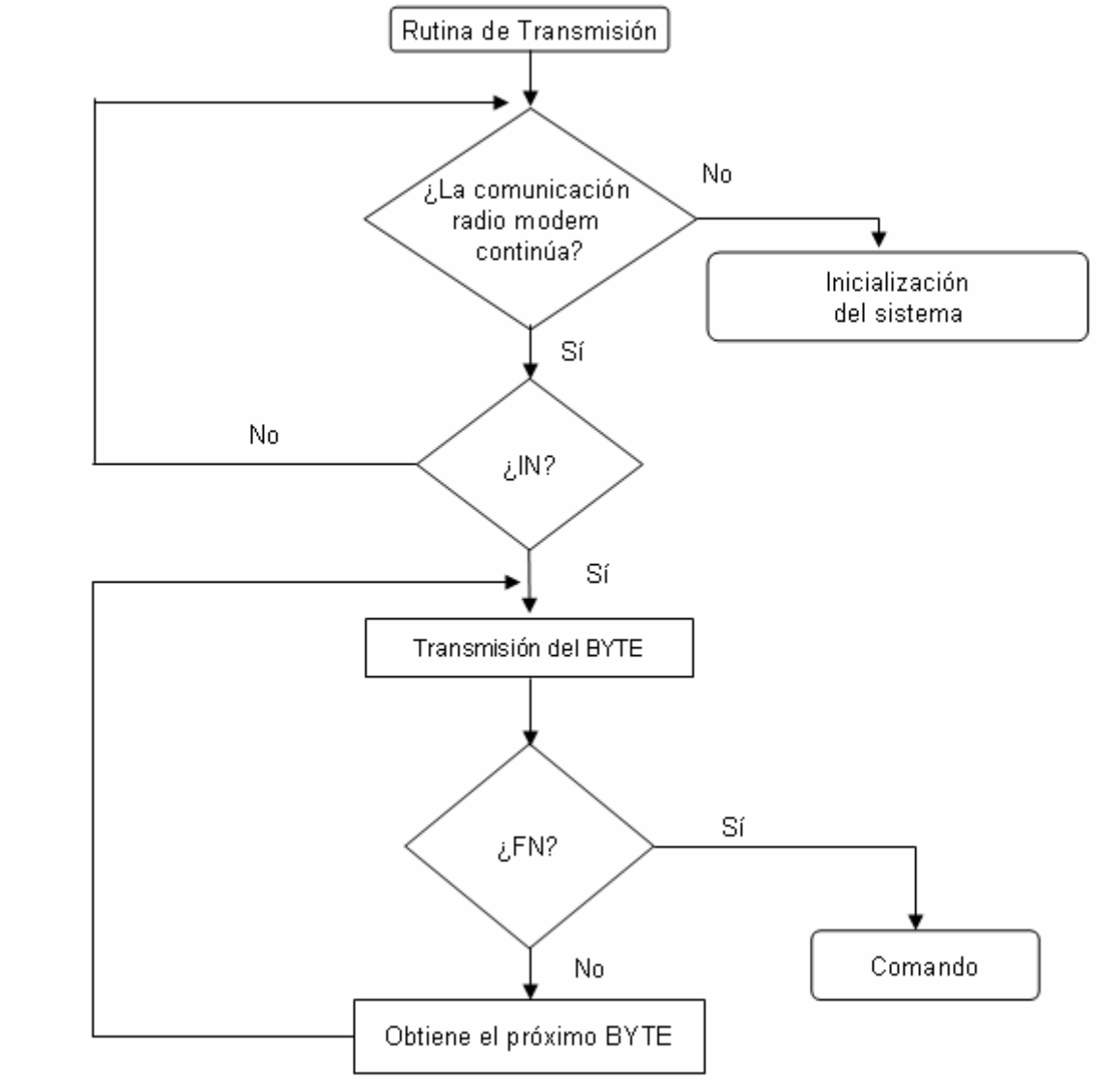


Figura 3.26. Rutina de transmisión de datos vía radio modem.

En el caso de la rutina de recepción, ésta se llevará a cabo de forma análoga a la rutina de transmisión, es decir, la recepción contará con dos banderas, una de ellas para inicializar la recepción (IN) y la segunda para concluir la transmisión de datos (FN), como se muestra en la figura 3.27. Con respecto a la bandera IN, esta esperará en un bucle hasta que se presente dicha bandera, una vez presentada se preguntará por el byte recibido el cual será almacenado en el sistema, pero si no se recibe ningún byte, se preguntará por la existencia de comunicación ya que si esta se pierde, inmediatamente se inicializará el sistema, de otra forma quiere decir

que recibirá más información hasta que se presente la bandera FN de fin de información, que una vez recibida se regresará a la rutina de comando.

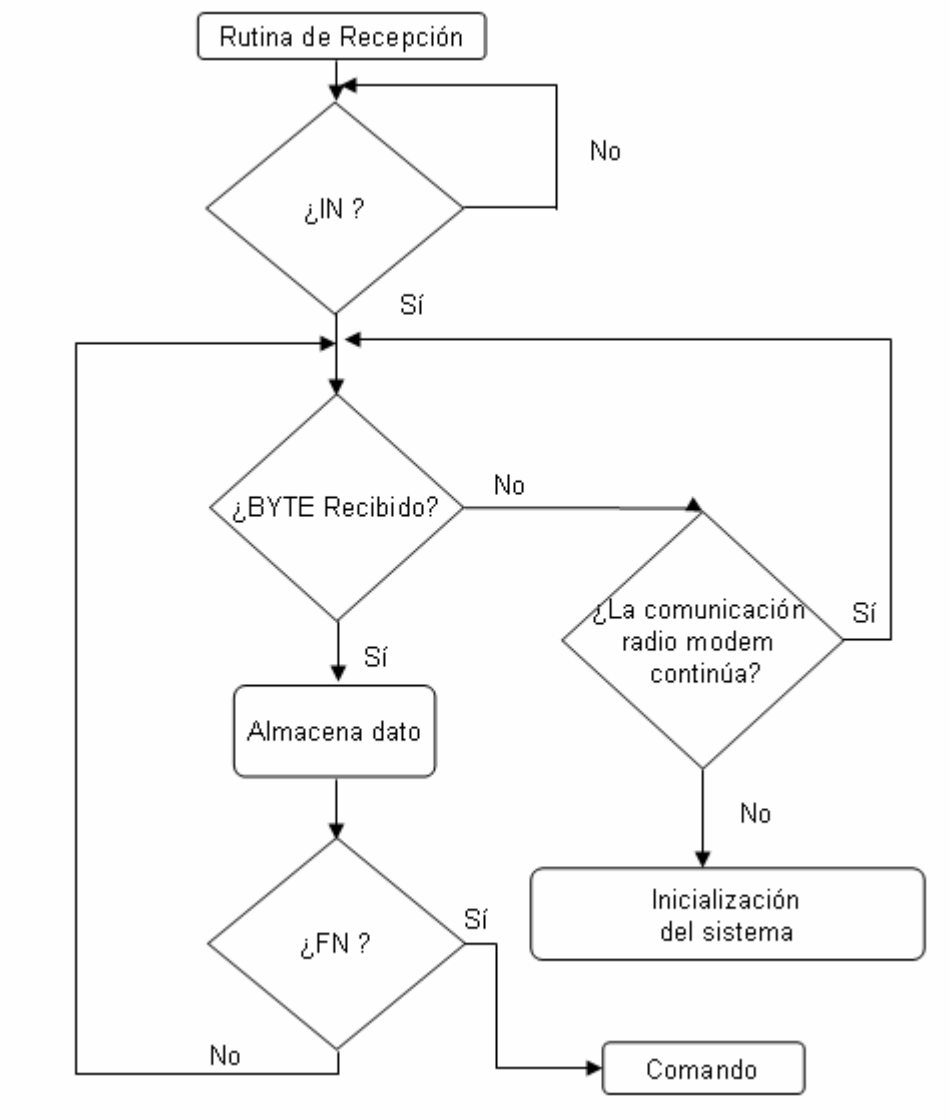


Figura 3.27. Rutina de recepción de datos vía radio modem.

Para el desarrollo del módulo de comunicación vía radio modem es utilizado el modem inalámbrico RF *Neulink RF9600*, el cual por su diseño es de fácil empleo y operación. Las características principales del radio modem empleado se muestran en la tabla 3.7.

Banda de UHF 403-512MHz (en Sub-bandas)
Modulación GFSK
Baudaje máximo 9600bps inalámbricamente
Detección de errores CRC-16
Protocolo RDXPTM para intercambio de datos
Configurable como repetidor
Puerto serial RS-232 configurables
Acepta energía de una batería
Opera en temperaturas: -30° a +60°
Enlace altamente confiable gracias a su protocolo que es sumamente transparente
Viene en un gabinete de acero inoxidable

Tabla 3.7. Características del radio modem.

El radio modem es alimentado con 12 voltios y opera en dos modos: modo comando y modo dato.

En el modo comando, es posible la configuración de los radios modems a través del puerto serial de la PC a una velocidad de transmisión de 57.6 kbps. Existen comandos propios del radio modem que permitirán la configuración del mismo, algunos comandos importantes son: CTS OFF, CTS ON, EXIT, HELP, MODES, MYID nnnn, PROG aa b, RESTART y TOID nnnn.

- *CTS OFF* . Desabilita el CTS (handshaking).
- *CTS ON*. Habilita el CTS (handshaking).
- *EXIT*. Salva los nuevos parámetros.
- *HELP*. Lista todos lo comandos.
- *MODES*. Despliega la configuración actual del radio modem.
- *MYID nnnn.* Determina el identificador de la unidad del radio modem (ID); el ID es un número hexadecimal que va desde 0001(h) a FFFE (h).
- *PROG aa b.* Habilita o deshabilita banderas para las funciones del radio modem.
- *RESTART.* Restablece los parámetros originales del radio modem.
- *TOID nnnn.* Determina el identificador del radio modem que se pretende comunicar.

Los radio modem son configurados en modo punto-multipunto, como se comentó en el diseño de esta comunicación, figura 3.28. En la estación central se encuentra configurado el radio modem que tiene acceso a los radio modems ubicados en las diferentes estaciones remotas. El radio modem de la estación central se identifica con el número “1234” y puede transmitir y

recibir información de 16 diferentes radio modems “000F”. Los radio modems instalados en las diferentes estaciones remotas sólo pueden establecer comunicación con el radio modem de la estación central, por ejemplo, a la estación remota número 1, le corresponde el identificador “0001” y puede comunicarse con el radio modem con el identificador “1234”, figura 3.30. Para poder realizar las pruebas de laboratorio correspondientes, sólo se contó con dos radio modems, el cual uno de ellos se empleó para simular la estación central y el segundo se configuró con diferentes identificadores, simulando las diferentes estaciones remotas.

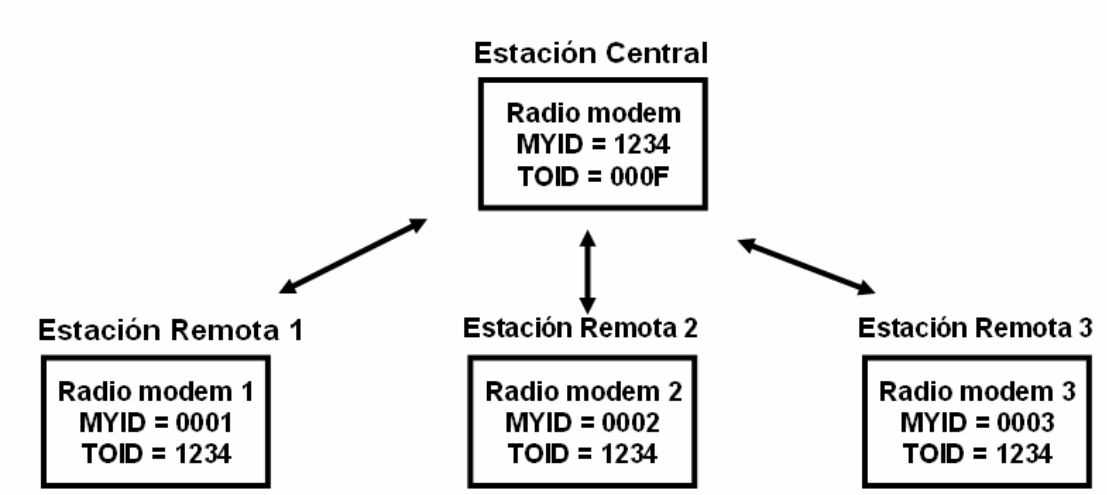


Figura 3.28. Configuración de los radios modems.

Como una muestra de la programación desarrollada, el siguiente código muestra la programación del microcontrolador AVR para las comunicaciones vía modem. Esta programación se desarrolla de la misma forma con la que se programan los dispositivos que comparten el protocolo RS232.

```
LDI     R16,LOW(RAMEND)          ;CARGA EL APUNTADOR DE PROGRAMA
OUT     SPL,R16                  ; VALOR DEL REGISTRO PARA LA
                                 ;CONFIGURACIÓN
LDI     R16,HIGH(RAMEND)         ;CONFIGURACIÓN DE REGISTRO PARTE ALTA
LDI     R17,$FF                  ;CARGA VALOR AL REGISTRO R17
OUT     DDRb,R17                 ;CONFIGURA PUERTO COMO SALIDA
LDI     R17,$00                  ;ENCIENDE LEDS
OUT     PORTb,R17
LDI     R17,23
OUT     UBRR,R17                 ;VELOCIDAD = 9600 BAUDS
LDI     R17,0B00011000
OUT     UCR,R17                  ;HABILITA RECEPCIÓN Y TRANSMISIÓN
```

El código siguiente muestra la forma de realizar la verificación del las banderas de inicio y fin.

```
; RUTINA DE TRANSMISIÓN DE INFORMACIÓN

                LDI             R17,'I'                 ; CARGA EL CARACTER I
                OUT             UDR,R17                 ; TRANSMITE LA A
TX_1:           SBIS            USR,TXC                 ; ESPERA A QUE SE TRANSMITA LA I
                RJMP            TX_1                    ;  MIENTRAS NO SE TRANSMITA VERIFICA
                SBI             USR,TXC
                RCALL           DELAY                   ; ESPERA UN TIEMPO
                LDI             R17,'N'                 ; CARGA EL CARÁCTER N
                OUT             UDR,R17                 ; TRANSMITE LA N
TX_T1:          SBIS            USR,TXC                 ;ESPERA A QUE SE TRANSMITE LA N
                RJMP            TX_T1                   ; MIENTRAS NO SE TRANSMITA VERIFICA

; RUTINA DE VERIFICACIÓN DE BANDERAS

ESPERA_A1:      SBIS            USR,RXC                 ; RECIBE CARACTERER DE I
                RJMP            ESPERA_A1               ; VERIFICA HASTA QUE SE RECIBA DATO
                IN              R17,UDR                 ; SALIDA POR EL REGISTRO R17
                CPI             R17,'I'                 ; COMPARACIÓN DE LA LETRA I
                BRNE            ERROR_AA                ; SI EXISTE UN ERROR BRINCA AL MANEJO DE
                                                        ; ERRORES
                RJMP            ESPERA_T1               ; CONTINUA CON LA TRANSMISIÓN
ERROR_AA:       RJMP
        ERROR_M
```

III.2.3. Interfaz GSM

En el módulo de comunicación vía GSM estará constituido por un microcontrolador *AVR*, un transceptor TTL a RS232 y un teléfono móvil. El microcontrolador se programará en lenguaje ensamblador para la transmisión/recepción de datos; este módulo empleará el transceptor TTL RS232 para la comunicación entre el microcontrolador y el teléfono móvil, como se muestra en la figura 3.29.

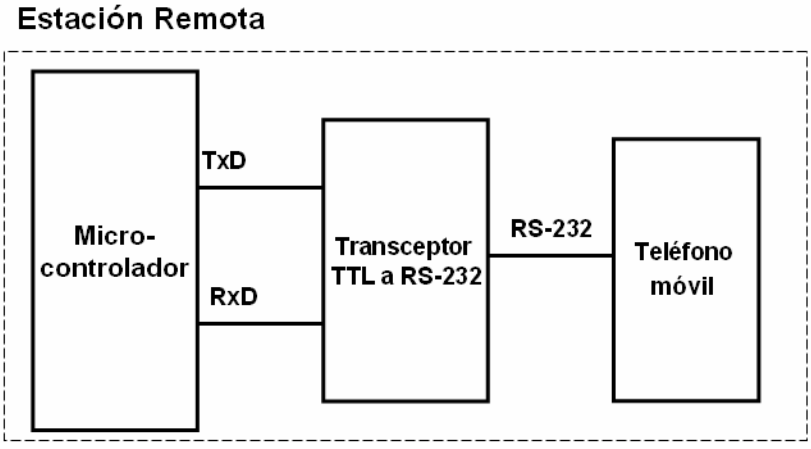


Figura 3.29. Módulo de comunicación vía GSM.

El *software* del microcontrolador se programará con el objetivo de controlar el proceso de comunicación entre el teléfono móvil y el microcontrolador. El microcontrolador realizará el control de la comunicación mediante la *UART*. La comunicación de datos entre el microcontrolador y el dispositivo móvil se realizará a través de los comandos *Hayes AT+,* ya explicados en el capítulo 2.

El diseño del programa del microcontrolador, para la comunicación con el teléfono móvil, deberá recibir el comando GS, para iniciar dicha comunicación. En la figura 3.30 se muestra el diagrama de flujo correspondiente al envío/recepción de datos. Como dato importante en la programación del dispositivo son la programación de los tiempos de espera que se llevarán acabo para establecer la inicialización de la comunicación entre el microcontrolador y el móvil. Además es necesaria la bandera *DTR* con un valor de uno. Con ello el sistema se encontrará listo para llevar a cabo la comunicación GSM, donde se podrá seleccionar la opción de transmisión o recepción de datos.

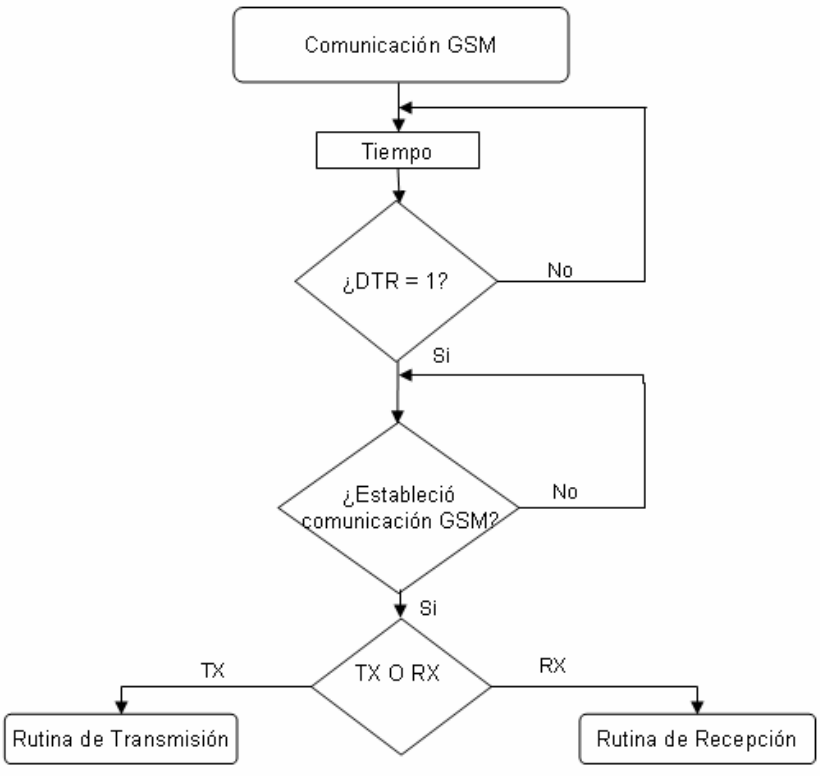


Figura 3.30. Selección de transmisión o recepción de datos GSM.

En la figura 3.31 se muestra la rutina de transmisión de datos: el microcontrolador enviará el comando *AT* para saber si el móvil está listo, si es así, éste enviará una contestación de *OK*, de otra forma enviará un mensaje de ERROR. Si el móvil está listo, el microcontrolador le enviará el comando *AT+CMGF=1*, el cual significa que los datos serán transmitidos en modo texto, posteriormente se transmitirá el comando *AT+CMGS ="número de celular",* el cual marcará al número de celular elegido. Si la comunicación es establecida, el móvil enviará el signo ">", para esperar al microcontrolador que le envíe la información y éste a su vez envíe los datos por el protocolo GSM a otro dispositivo móvil. Una vez concluida la transmisión el móvil enviará un *FN* de fin de transmisión para confirmar que los datos fueron enviados correctamente enseguida el sistema regresará a la rutina de comando.

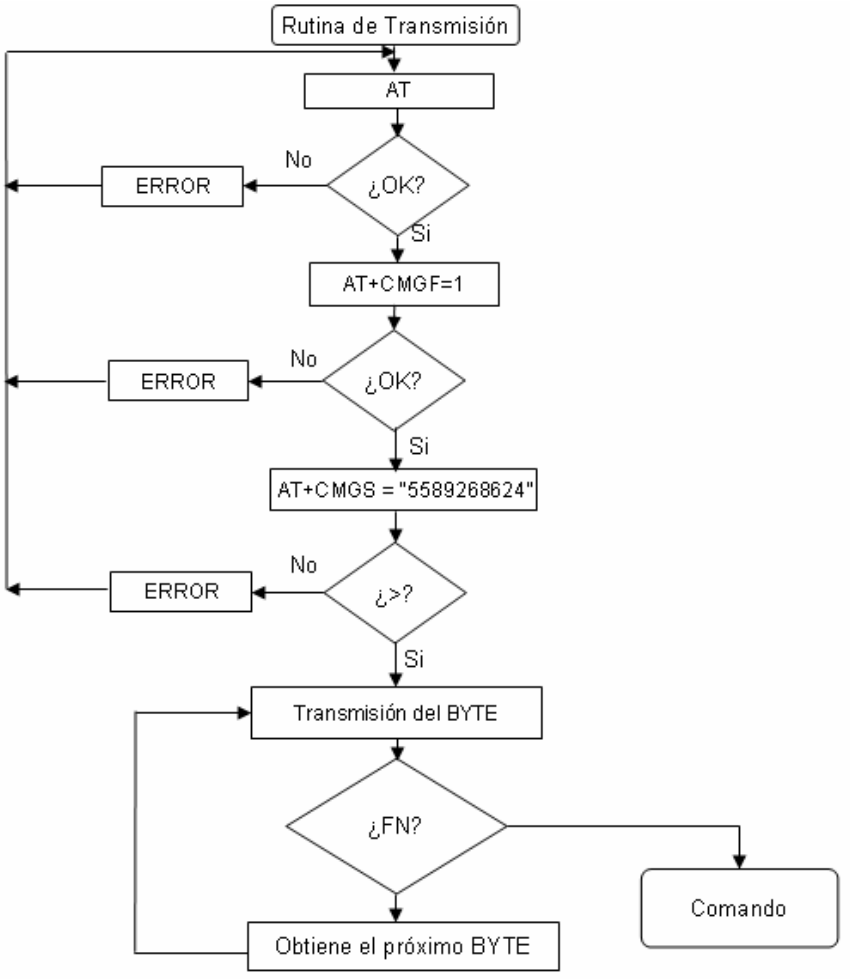


Figura 3.31. Rutina de transmisión de datos vía GSM.

Con respecto a la rutina de recepción de datos, ésta se llevará a cabo con el comando *AT+CMGL*, que es para lectura de datos, es decir, si el microcontrolador además de recibir el comando *AT+CMGF=1,* recibe el comando *AT+CMGL* para lectura de datos, si estas banderas no se cumplen el sistema se quedará en un ciclo hasta encontrar ambas banderas activadas, cuando éstas banderas son afirmativas el microcontrolador comenzará a almacenar la información byte por byte, realizando una verificación de conexión en caso de no encontrar más datos, el sistema se inicializará inmediatamente, de otra forma el sistema regresará a la verificación de los próximos byte hasta recibir la bandera *FN*, la cual indicará el fin de información, una vez recibida dicha bandera el sistema regresará a la rutina comando del microcontrolador, figura 3.32.

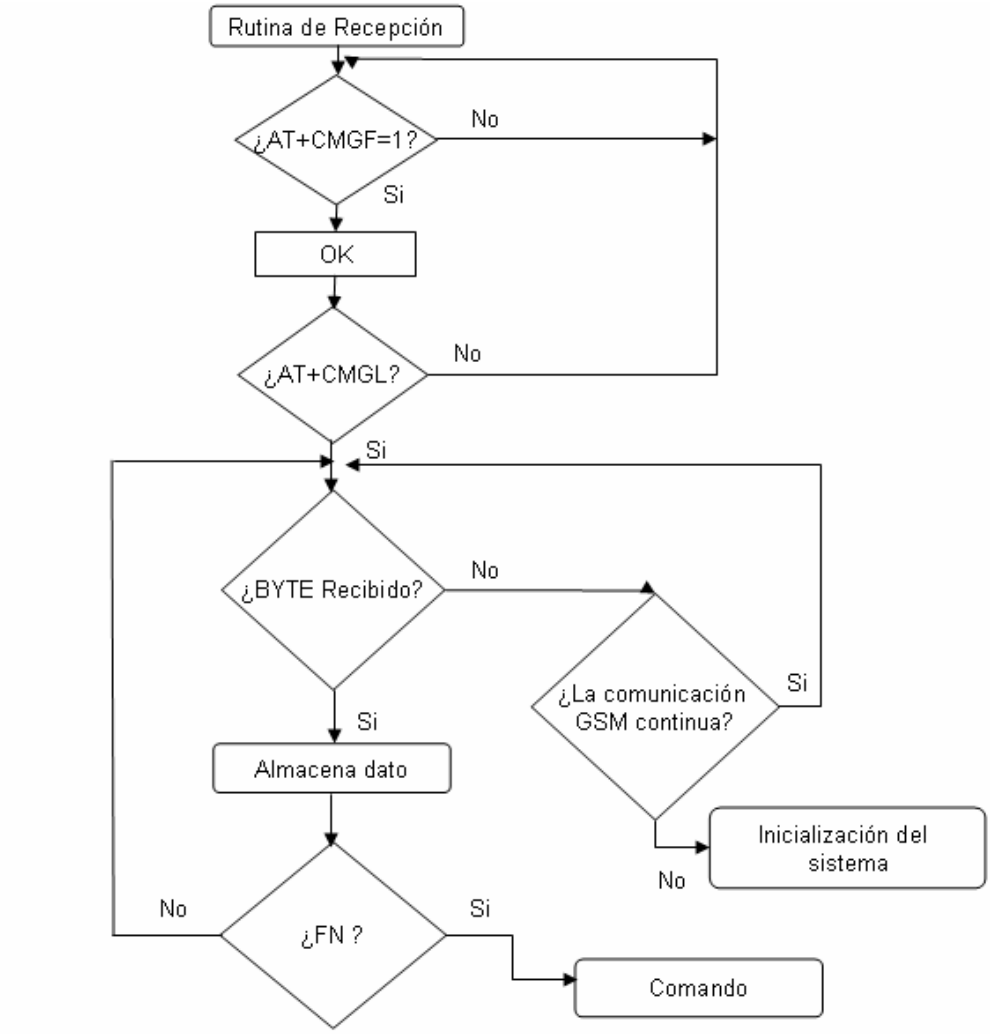


Figura 3.32. Recepción de datos vía GSM.

Para el desarrollo de este módulo se empleó un teléfono celular que cuenta con la comunicación vía GSM, además de manejar la comunicación infrarroja. Se empleó su cable de conexión correspondiente para comunicación RS232, como se muestra en la figura 3.33. Además, para la programación se ocupó el sistema de desarrollo *AVR STK500,* figura 3.3, el cual es un sistema para realizar pruebas de microcontroladores, que incluye al microcontrolador AT90S8515, comentado anteriormente.

Figura 3.33.Teléfono móvil y cable RS232.

El código del microcontrolador se desarrolla de una manera muy similar a los demás módulos de comunicación, sólo con la diferencia de los comandos utilizados, como son: *AT+CMGL, AT+CMGF=1, AT+CMGS =”número de celular”,* etc*.* En el siguiente código se muestra el código para el marcado del número a celular.

```
;*******************************
; MARCADO  DEL NÚMERO
;*******************************
NUMERO:    LDI         R17,'5'         ; CARGA EL NUMERO 5
           OUT         UDR,R17         ;TRANSMITE EL 5

NUM_1:     SBIS        USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO
           RJMP        NUM_1           ; MIENTRAS NO SE TRANSMITA VERIFICA
           SBI         USR,TXC         ; ESPERA A QUE SE TRANSMITA EL DATO
ESPERA5:   SBIS        USR,RXC         ; ESPERA A QUE SE RECIBA EL DATO

           RJMP        ESPERA5         ; MIENTRAS NO SE TRANSMITA VERIFICA
           IN          R17,UDR         ; SALIDA POR EL REGISTRO R17
           CPI         R17,'5'         ;COMPARA EL DATO
           BRNE        ERROR5          ; MANEJO DE ERRORES
           RJMP        NUM_2           ; CONTINUA CON EL SIGUIENTE NÚMERO

ERROR5:    RJMP        ERROR_M         ; MANEJO DE ERRORES
NUM_2:     LDI         R17,'5'         ; CARGA EL NUMERO 5
```

```
            OUT     UDR,R17     ;TRANSMITE EL DATO

TX_NUM2:    SBIS    USR,TXC     ;ESPERA A QUE SE TRANSMITA LA +
            RJMP    TX_NUM2     ; MIENTRAS NO SE TRANSMITA VERIFICA
            SBI     USR,TXC     ; ESPERA A QUE SE TRANSMITA EL DATO

ESPERAN2:   SBIS    USR,RXC     ; ESPERA A QUE SE RECIBA EL DATO
            RJMP    ESPERAN2    ; MIENTRAS NO SE TRANSMITA VERIFICA
            IN      R17,UDR     ; SALIDA POR EL REGISTRO R17
            CPI     R17,'5'     ; COMPARA EL DATO
            BRNE    ERRORN2
            RJMP    NUM_3       ; CONTINUA CON EL SIGUIENTE NÚMERO
ERRORN2:    RJMP    ERROR_M     ; MANEJO DE ERRORES
NUM_3:      LDI     R17,'2'     ; CARGA EL NUMERO 2
            OUT     UDR,R17     ;TRANSMITE EL DATO

TX_3:       SBIS    USR,TXC     ;ESPERA A QUE SE TRANSMITA LA +
            RJMP    TX_3        ; MIENTRAS NO SE TRANSMITA VERIFICA
            SBI     USR,TXC     ; ESPERA A QUE SE TRANSMITA EL DATO

ESPERAN3:   SBIS    USR,RXC     ; ESPERA A QUE SE RECIBA EL DATO
            RJMP    ESPERAN3    ; MIENTRAS NO SE TRANSMITA VERIFICA
            IN      R17,UDR     ; SALIDA POR EL REGISTRO R17
            CPI     R17,'2'     ;COMPARA EL DATO
            BRNE    ERRORN3     ; MANEJO DE ERRORES
            RJMP    NUM_4       ; CONTINUA CON EL SIGUIENTE NÚMERO
```

En el apéndice A se muestra el código completo de este módulo de comunicación. Cabe destacar que para el desarrollo y pruebas de este módulo de comunicación se empleó el emulador ICE200 de ATMEL, figura 3.34, el cual es una tarjeta que es colocada entre la PC y el zócalo de la tarjeta de circuito impreso donde se alojará el microcontrolador definitivo, en este caso es la tarjeta *AVR STK500.* El programa es ejecutado desde la PC, la cual presenta en pantalla los pasos que esta realizando el microcontrolador, teniendo la oportunidad de realizar en cualquier momento una pausa para el análisis del programa.

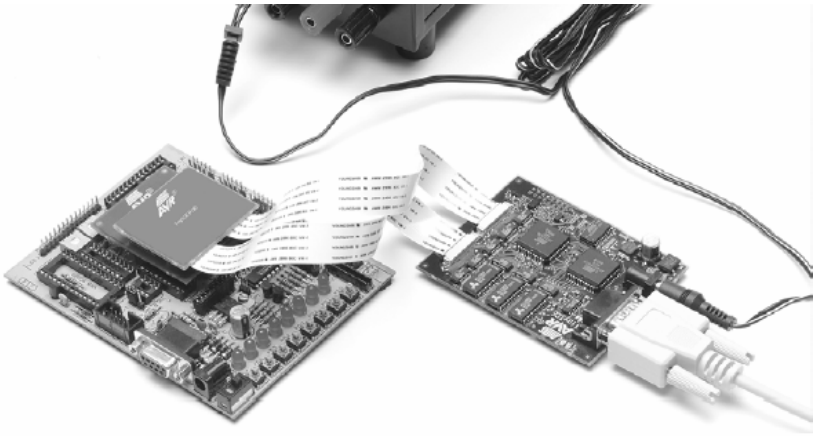


Figura 3.34.Emulador ICE200.

Habiendo terminado de describir las interfaces del *hardware* de cada uno de los módulos que integran el sistema de comunicaciones alámbricas e inalámbricas, se procederá al diseño y desarrollo del *software* de la estación central.

III.3. Software de la Estación Central

El software de la estación central se diseñará para la comunicación de datos con los diferentes módulos de comunicación que se encuentran en las estaciones remotas. Para realizar está comunicación el equipo de cómputo de la estación central contará con uno o más puertos serie.

Para la transmisión de los datos es necesario otro nivel de interpretación. En el lado del sistema operativo, en este caso *Windows,* éste utiliza un controlador de comunicaciones, *Comm.drv*, para enviar y recibir datos mediante las funciones estándar de la API de Windows.

El software para la estación central se desarrollará para dos plataformas, la primera de ella será para *Windows XP*, ésta proporciona características importantes como son: una interfaz novedosa por sus colores llamativos, facilidad de uso, mayor compatibilidad con las nuevas tecnologías en software y hardware, compatibilidad con redes (inalámbricas, infrarrojas, etc.) y controladores de dispositivos actualizados. Las desventajas que presenta este sistema operativo se encuentran las siguientes: la necesidad de un mínimo de memoria en RAM de 128 MB, vulnerabilidad en infecciones de virus o algún programa mal intencionado y además presenta numerosos *bugs* (errores de *software*). La segunda plataforma a utilizar es *Windows CE*, esta plataforma se emplea para dispositivos móviles, como pueden ser PDA. Las ventajas de emplear este sistema operativo se encuentran las siguientes: consume pocas cantidades de RAM (todos los componentes de *Windows CE* se pueden ejecutar en ROM), es independiente del hardware del sistema, es un sistema operativo de plataforma cruzada, multihilos (*threads*), además es posible configurar a los dispositivos móviles para que reinicien el sistema instantáneamente al obtener respuesta por parte de usuario. La desventaja es que es necesaria la licencia para su uso.

Una vez que seleccionadas las plataformas de trabajo, es necesario un lenguaje de programación para el desarrollo del sistema, este lenguaje deberá contemplar las siguientes características: compatibilidad con las plataformas a utilizar e instrucciones para el manejo de comunicaciones seriales. Por lo tanto, los lenguajes de programación que se emplearán serán:

Visual C++, Visual Basic y Visual Studio. NET; ya que las características de cada módulo de comunicación difieren considerablemente al método de programación desarrollado para cada una. En seguida se describen brevemente algunas características importantes de cada lenguaje de programación.

Visual Basic

Es un lenguaje de programación para Windows de *Microsoft*, este lenguaje está diseñado específicamente para crear aplicaciones con interfaz gráfica de forma rápida y sencilla. *Visual Basic* está centrado en dos tipos de objetos, ventanas y controles, que permiten crear una interfaz gráfica para una aplicación dada. Para realizar una aplicación se crean ventanas, llamadas formularios, y sobre ellas se dibujan objetos llamados controles (cajas de texto, botones de órdenes, listas desplegables, etc.). Una vez realizado lo anterior se desarrolla código fuente relacionado con cada objeto (ventanas y controles). Cada objeto está ligado a un código que permanece inactivo hasta que se dé el suceso que lo activa.

Otras ventajas de emplear Visual Basic se encuentran: controles visuales preconstruidos por terceros (controles Visual Basic eXtention VBX), soportes para establecer enlaces con *Windows* y rutinas escritas en otros lenguajes (DLL, Dynamic Link Libraries), visualización y manipulación de datos de otras aplicaciones Windows, utilizando controles OLE (*Objet Linking And Embedding*).

Un último aspecto de importancia en este lenguaje de programación es el control de comunicaciones (*Communications*) que permite una funcionalidad sencilla de comunicaciones del puerto serie, contiene una herramienta de comunicaciones completa controlada por eventos. Además, proporciona una interfaz con un conjunto estándar de comandos de comunicaciones, permitiendo establecer una conexión con un puerto serie, el cual es atendido como un solo proceso. Por lo tanto, como programador en Visual Basic, sólo se tiene que establecer y supervisar las propiedades y eventos del control *Communications*.

Los módulos que se implementarán con el lenguaje de programación Visual Basic serán: modem telefónico, radio modem y GSM.

Visual C++

Cuando se utiliza la plataforma *Windows XP*, es posible la ejecución de varios programas simultáneamente. Esta posibilidad se denomina multitarea. Además de esta posibilidad también es posible la ejecución de procesos independientes que no son aplicaciones completas, denominados hilos. Un hilo es un camino de ejecución a través de un programa. En un programa multihilo, cada hilo tiene su propia pila y funciona de forma independiente a otros hilos que se ejecutan en el mismo programa. Una aplicación puede crear varios hilos, es decir varios flujos de ejecución diferentes y ejecutarlos concurrentemente. Por lo tanto cada aplicación puede ejecutar varias aplicaciones a la vez. Este es el caso de la comunicación infrarroja, el sistema operativo emplea hilos para lograr la transmisión de información con un dispositivo.

En el lenguaje de programación denominado *Visual C++* , además de contar con todas las facilidades que cuenta *Visual Basic*, es posible la programación de los hilos antes mencionados, causa principal para emplear el lenguaje de programación. Cabe destacar que Visual C++, también cuenta con las siguientes ventajas: asistentes para generar código (es posible producir el entorno de una ventana de trabajo en poco tiempo), consta de bibliotecas MFC (Clases Fundamentales de Microsoft, *Microsoft Foundation Classes).*

Los módulos que se programará con Visual C++ serán: SERIE (RS232), USB y para la comunicación infrarroja de alto nivel (IrDA).

Visual Studio. NET

Visual Studio. NET. Es un lenguaje de programación, el cual contiene un conjunto de herramientas de desarrollo para la construcción de aplicaciones Web ASP (*Active Server Pages*), servicios Web XML (Lenguaje de Marcado Ampliable, *Extensible Markup Language*), aplicaciones para escritorio y aplicaciones móviles. Este lenguaje aprovecha las funciones de .NET Framework, que ofrece el acceso a tecnologías clave para simplificar el desarrollo de las aplicaciones.

El entorno de desarrollo integrado de *Visual Studio. NET* incluye herramientas para el desarrollo de aplicaciones para dispositivos inteligentes, como Pocket PC. Mediante las herramientas y .NET Compact Framework, un subconjunto de .NET Framework, puede, crear, generar, depurar e implementar aplicaciones que utilizan .NET Compact Framework, para ejecutarse en asistentes digitales personales (PDA), teléfonos móviles y otros dispositivos.

Dadas las características de este lenguaje de programación, se empleará para la realización de la programación para PDA, cabe destacar que la aplicación sólo se instalará en la PDA y no en la estación central.

Una vez descritos los lenguajes empleados, los módulos de programación serán unificados en un sólo programa desarrollado en Visual Basic, es decir, habrá un solo programa para el control de las diferentes comunicaciones, donde contendrá las diferentes opciones para los diferentes módulos y éstos tendrán a la vez los parámetros configurables para el establecimiento de la comunicación, como pueden son: puerto, baudaje, bit de datos, bit de stop, identificadores, indicadores de conexión, área de marcado de números telefónicos, etc.

En cuanto al almacenamiento de la información, se programarán tres diferentes formas de realizar este proceso, para que más adelante se seleccione la que más convenga. Las forma de almacenar la información son: almacenamiento por parte del usuario eligiendo la ruta o *path*, almacenamiento por parte del usuario con un *path* predeterminado y por último almacenamiento automático en tiempo real.

El manejo de errores del sistema se llevarán a cabo por medio de la programación, tendrán que ser programados opciones como: la recepción de una señal de interrupción, error de trama, desbordamiento del búfer de recepción, espacio insuficiente en el búfer de recepción, error de paridad e intento de colocar un carácter más en la cola mientras el búfer de transmisión se encuentra lleno.

Por último se pretende que el software de la estación central opere de manera autónoma, es decir, que pueda ser programado para el envío y recepción de información en periodos de tiempo largos, ya que los sistemas de estaciones remotas trabajan de esa forma.

Una vez diseñados los parámetros necesarios para al software de la estación central se procede a desarrollar el sistema de la siguiente forma.

Para la ejecución del programa del sistema de comunicaciones de datos para estaciones remotas existe un archivo ejecutable llamado principal, al ejecutar dicho programa, éste desplegará una ventana que contiene los siguientes elementos: nombre del sistema, nombre del desarrollador y versión, figura 3.35. Posteriormente se despliega la ventana principal, la

cual contiene seis opciones a elegir para el tipo de comunicación que se desee: SERIE, IrDA, USB, GSM, MODEM y RADIO MODEM, figura 3.36.

Figura 3.35. Presentación del sistema de comunicaciones de datos para estaciones remotas.

Figura 3.36. Menú principal para los diferentes módulos de comunicación.

Módulos de comunicación vía SERIE, USB e IrDA

Los módulos de comunicación SERIE, USB e IrDA fueron programados en Visual C++, como se comentó anteriormente, estos comparten las ventanas para la transmisión de información entre estación remota y estación central. Para establecer la comunicación en cualquiera de estas opciones, sólo es necesario seleccionar el puerto indicado para cada comunicación, por ejemplo, para el puerto serie y USB se seleccionará el puerto COM1, y para IrDA se seleccionará el COM4 o 6, según la configuración de computadora que sea utilizada.

Considerando lo anterior, a manera de ejemplo se describirá el proceso de comunicación vía IrDA, asumiendo que los módulos anteriores comparten las mismas ventanas.

Para el desarrollo del programa para este tipo de comunicaciones se empleó código libre de *Microsoft*, el cual soporta la comunicación multihilos (*threads*), ya que por la magnitud de la programación esta compañía ofrece libremente dicha programación, por lo tanto se llevó a la tarea de rediseñar el código adaptándolo a las necesidades del módulo de comunicación.

El programa cuenta con archivos fuentes, archivos de cabecera y archivos de recursos, figura 3.37. El programa realiza una verificación de la versión de *Windows32. S*i la versión no es correcta el programa no se ejecutará, enviando una ventana de error la cual indicará el problema presentado. Pero si es correcta la versión, son inicializadas las variables globales, registro de clases y es desplegada la ventana principal, inicializándose los *threads* correspondientes. En el siguiente código se muestra, los valores de los parámetros que deben ser considerados para la verificación de la versión de *Windows* antes mencionada, así como los datos correspondientes a la programación multihilos.

```
VALUE "Comments", "Sistema de Comunicaciones de Datos Sample for the Win32 SDK.\r\nDemonstrates serial communiction using multiple threads.\0"
        VALUE "CompanyName", "Microsoft Corporation\0"
        VALUE "FileDescription", "Sistema de Comunicaciones de Datos Sample for Win32\0"
        VALUE "FileVersion", "4, 0, 0, 0\0"
        VALUE "InternalName", "MTTTY\0"
        VALUE "LegalCopyright", "Copyright © 1995\0"
        VALUE "LegalTrademarks", "\0"
        VALUE "OriginalFilename", "MTTTY.exe\0"
        VALUE "PrivateBuild", "\0"
        VALUE "ProductName", "Microsoft MTTTY Sample\0"
        VALUE "ProductVersion", "4, 0, 0, 0\0"
        VALUE "SpecialBuild", "\0"
```

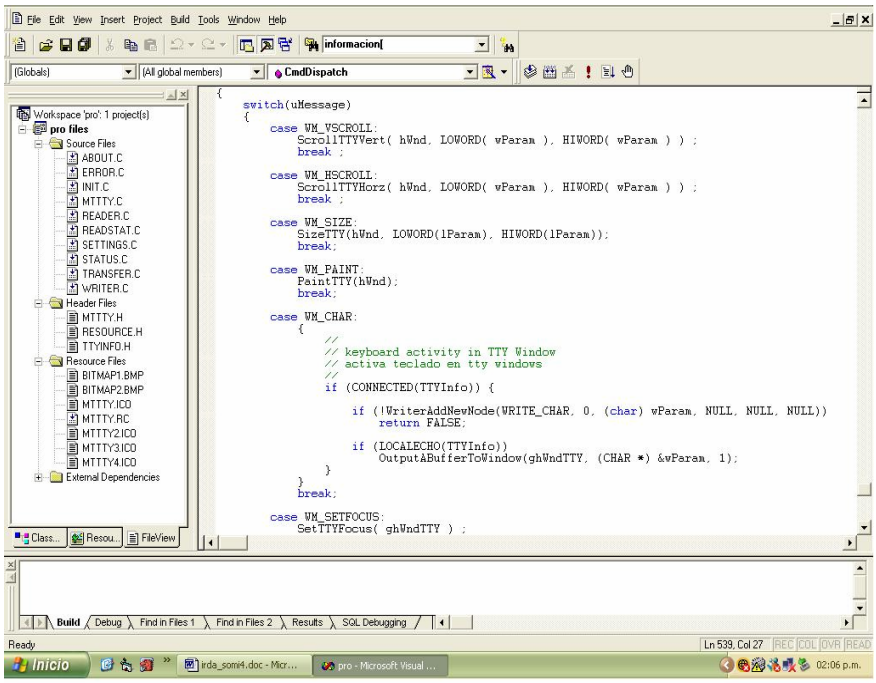


Figura 3.37. Entorno de programación.

Posteriormente, se lleva a cabo la programación de los parámetros a establecer como son: puertos, baudaje, bit de inicio, bit de paridad, bit de datos y bits de stop. Una vez que son configurados los parámetros se envía un comando de apertura de puerto, si se establece la comunicación el programa estará listo para aceptar los datos provenientes de la estación remota, almacenándolos en arreglos. Finalmente, se programa una opción para el almacenamiento de datos en el equipo portátil (Laptop), el cual tiene la capacidad de guardar los datos en extensiones *.TXT*, en cualquier directorio de la plataforma. En caso de no establecer la comunicación, el programa esperará a que se seleccione el puerto correcto.

La forma en que opera la aplicación es la siguiente: al aproximar el equipo portátil, Laptop, a la estación remota, la Laptop reconocerá automáticamente al dispositivo infrarrojo con un icono que aparecerá en la parte inferior derecha de la barra de Windows, figura 3.38. La Laptop asignará el número de puerto que se ocupará para esta aplicación. Posteriormente se ejecuta una segunda ventana que permitirá establecer la comunicación infrarroja, para realizar dicha comunicación es necesario la configuración de los parámetros como son: selección del puerto para la comunicación infrarroja, velocidad de transmisión, bit de inicio, paridad, bit de datos y bit de stop; una vez configurados estos parámetro en el menú de archivo se elige la opción de

conexión y se realizará dicha comunicación. Concluida la comunicación es posible guardar los datos, figura 3.39. Los datos serán almacenados en un archivo con extensión *.TXT*, el cual se podrá almacenar en el directorio que elija el usuario, figura 3.40.

Figura 3.38. Indicación de que un dispositivo infrarrojo está al alcance.

Figura 3.39. Operación de la interfaz IrDA.

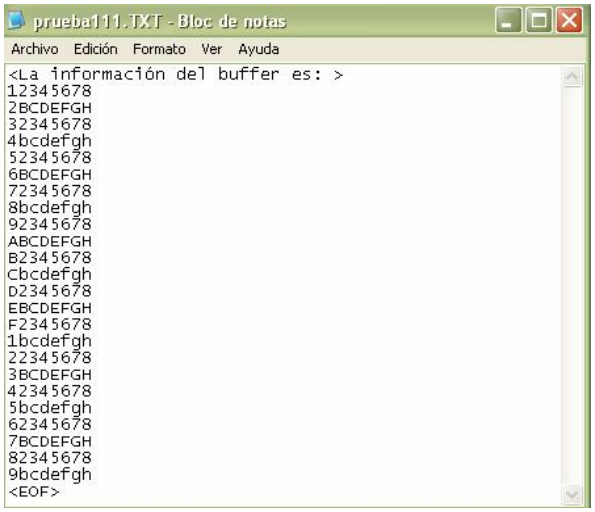


Figura 3.40. Datos almacenados en un archivo con extensión .TXT.

Módulo de comunicación vía MODEM TELEFÓNICO

El programa para controlar las comunicaciones vía modem está desarrollada en *Visual Basic*. Dicho programa está estructurado por un sólo formulario, el cual contiene la secuencia de instrucciones que permiten al modem establecer la comunicación. El primer paso para establecer la comunicación consiste en establecer la conexión con el puerto serie. Para abrir un puerto serie son utilizadas las siguientes propiedades: *CommPort, PortOpen y Settings.*

La propiedad *CommPort* determina el puerto serie que se va a emplear (abrir). Si hay un módem conectado al COM2, se establece el valor a 2 (COM2) y se conecta con el modem. Puede establecer la propiedad *CommPort* a cualquier número entre 1 y 16 (el valor predeterminado es 1.

La propiedad *Settings* permite especificar la velocidad en baudios, la paridad y el número de bits de datos y de parada. De forma predeterminada, la velocidad en baudios es 9600. La paridad sirve para la validación de los datos. Normalmente no se utiliza y se establece a "N". El valor de bits de datos indica el número de bits que representan un bloque de datos. El bit de parada indica cuándo se ha recibido un bloque de datos.

Después de especificar el puerto que se va a abrir y la forma en que se realizará la comunicación de los datos, para establecer la conexión se usa la propiedad *PortOpen,* que es un valor booleano, *True* o *False*. Sin embargo, si el puerto no funciona, si la propiedad

CommPort no se ha establecido correctamente o si el dispositivo no admite la configuración especificada, se producirá un error o puede que el dispositivo externo no funcione correctamente. Si establece la propiedad *PortOpen* a *False*, se cierra el puerto.

Véase las siguientes líneas de código donde se muestra la configuración antes discutida.

```
MSComm1.CommPort = 2
MSComm1.Settings = "9600,N,8,1"
MSComm1.PortOpen = True
```

Una vez establecida la conexión con el puerto serie mediante las propiedades *CommPort, Settings* y *PortOpen*, se emplea la propiedad *Output* para activar el modem e interactuar con él. La propiedad *Output* se utiliza para emitir los comandos que controlan la interacción entre dos modems. Por ejemplo, la siguiente línea de código indica la activación del modem así como el marcado del número telefónico.

```
MSComm1.Output = "ATDT 555-5555" & vbCr
```

El sistema es capaz de realizar una comprobación de la operación del modem mediante el comando *+++,* para saber el estado del dispositivo, que puede ser activo o inactivo. Si está en estado activo se recibirá una contestación *"OK*", para después ejecutar el comando *ATH0,* para proceder a descolgar y así marcar el número indicado por el usuario. El comando *"AT"* inicia la conexión, *"D"* marca el número y *"T"* especifica que el marcado es por tonos (y no por pulsos). Debe incluir un carácter de retorno de carro *(vbCr*) cuando se transmiten datos a una terminal. Cuando un comando se procesa con éxito, se obtendrá el código de resultado "OK", el cual se comprueba para determinar si el comando se ha ejecutado correctamente *(If PtoRead("OK", 30) Then……).* Una vez que se realizó la llamada, se espera la contestación en la terminal, enviando un comando *ATA* para iniciar la transmisión de información.

Cabe destacar que la programación del módulo de comunicación modem contiene un manejo de errores para la transmisión de la información, como se había mencionado en el diseño del sistema del software, estos errores son programados de la siguiente manera:

```
' Controlar cada evento o error escribiendo
' Código en cada instrucción Case

    Case comBreak   ' Se ha recibido una interrupción.
    Case comEventFrame   ' Error de trama
    Case comEventOverrun   ' Datos perdidos.
```

```
Case comEventRxOver   ' Desbordamiento del búfer de recepción.
Case comEventRxParity   ' Error de paridad.
Case comEventTxFull   ' Búfer de transmisión lleno.
Case comEventDCB   ' Error inesperado al recuperar DCB.
Case comEvCD   ' Cambio en la línea CD.
Case comEvCTS   ' Cambio en la línea CTS.
Case comEvDSR   ' Cambio en la línea DSR.
```

La interfaz para el usuario, en la comunicación vía modem consiste en lo siguiente: una vez seleccionada la opción en la ventana principal (figura 3.39.), es desplegada la pantalla que se muestra en la figura 3.41, la cual cuenta con opciones de marcado del número telefónico, restringido a 10 dígitos. Una vez seleccionado el número identificador del modem, que está relacionado con el número de estación remota a monitorear, se pulsa el botón de conectar. En esta opción pueden suceder dos casos, la primera de ella es que la comunicación no se pueda establecer, si esto sucede, se desplegará una ventana con la leyenda *"NO SE HA PODIDO ESTABLECER LA COMUNICACIÓN*", además en la parte esquina superior izquierda aparece la leyenda de desconectado, como se muestra en la figura 3.42. El segundo caso es cuando los modems logran establecer la comunicación, la ventana mostrará dicha conexión en la parte superior izquierda, presentando el número al que se conectó; en este momento la interfaz se encontrará lista para transmitir y recibir datos, como se muestra en la figura 3.43. Una vez que fueron recibidos y transmitidos los datos se tiene la opción de desconectar el puerto, lo que el cual cerrará el puerto de comunicación con el que se trabajó.

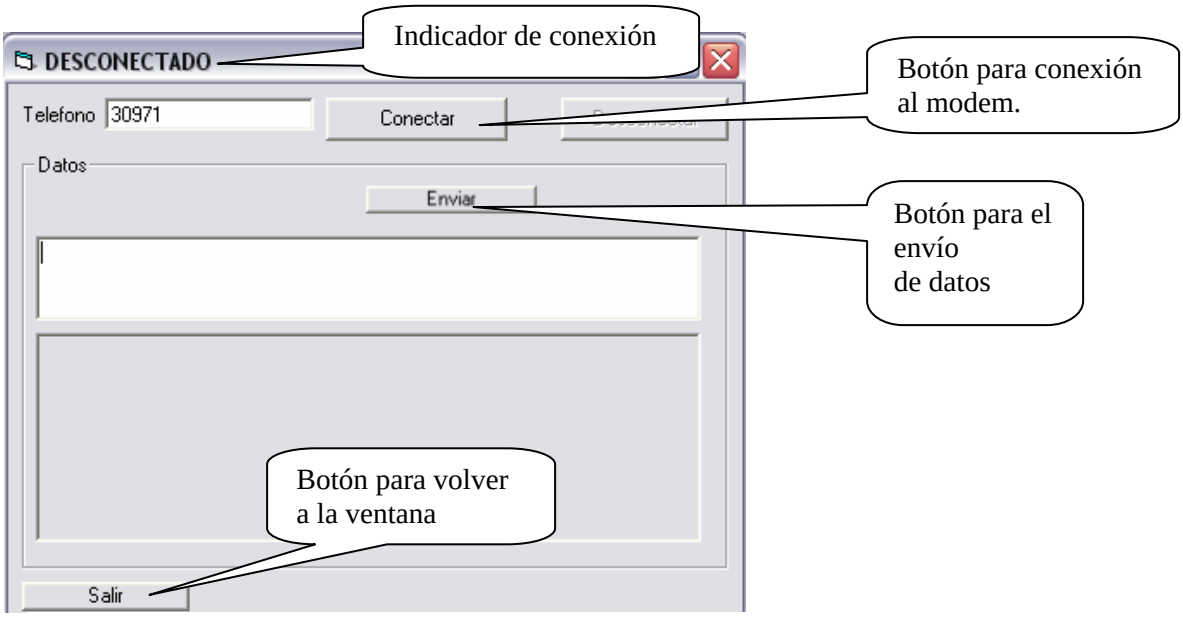


Figura 3.41. Opciones de la interfaz para la comunicación vía modem.

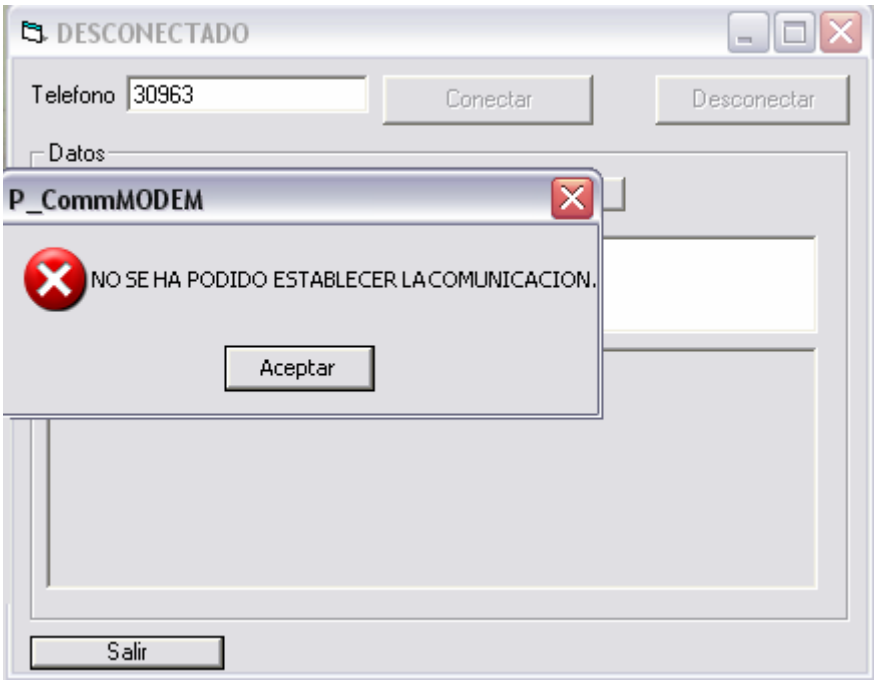


Figura 3.42. Error en el proceso de conexión vía modem.

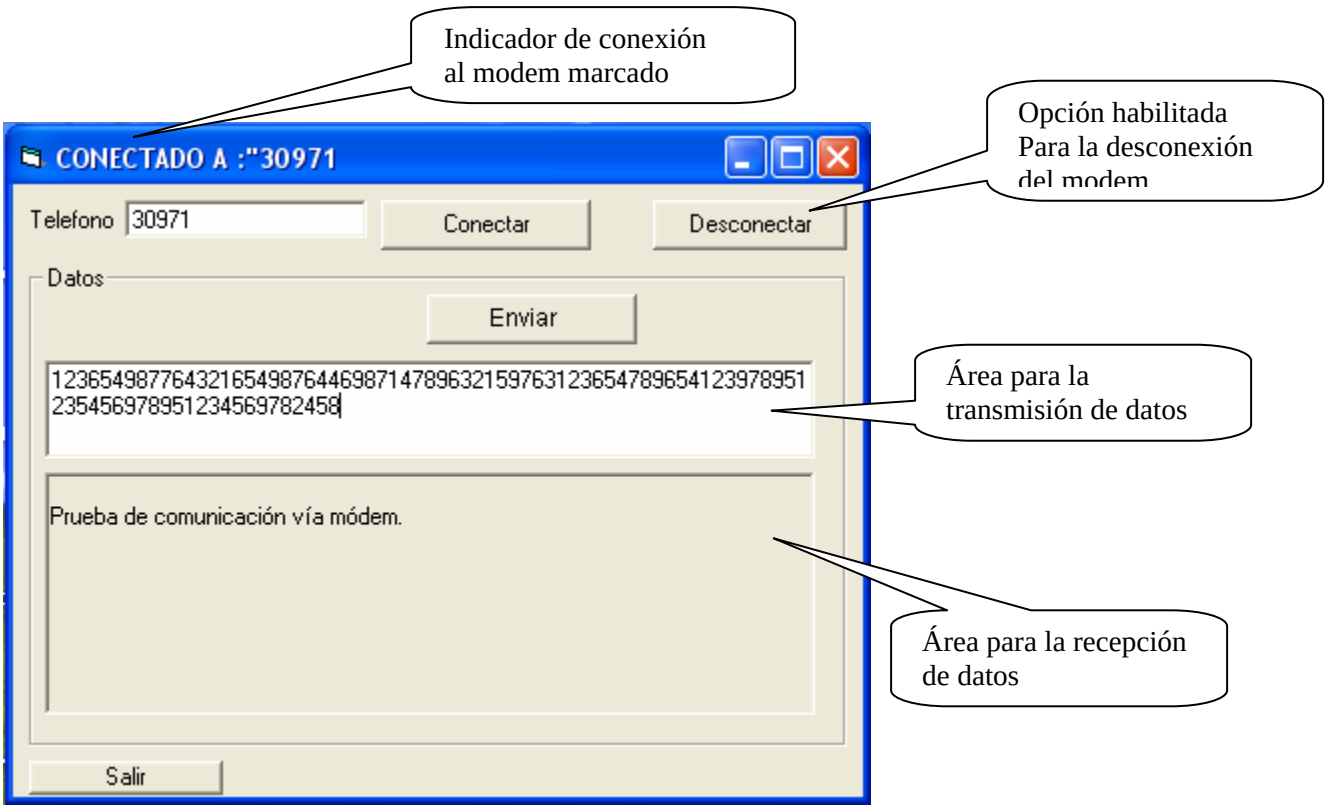


Figura 3.43.Transmisión de datos vía modem.

Los datos adquiridos de la estación remota se podrán almacenar en un archivo con extensión *.TXT*, para esta opción el *path* del archivo está definido en la programación. El nombre del archivo tiene el siguiente formato: día, mes, año, horas, minutos y segundos (ddmmyyhhmmss), figura 3.44.

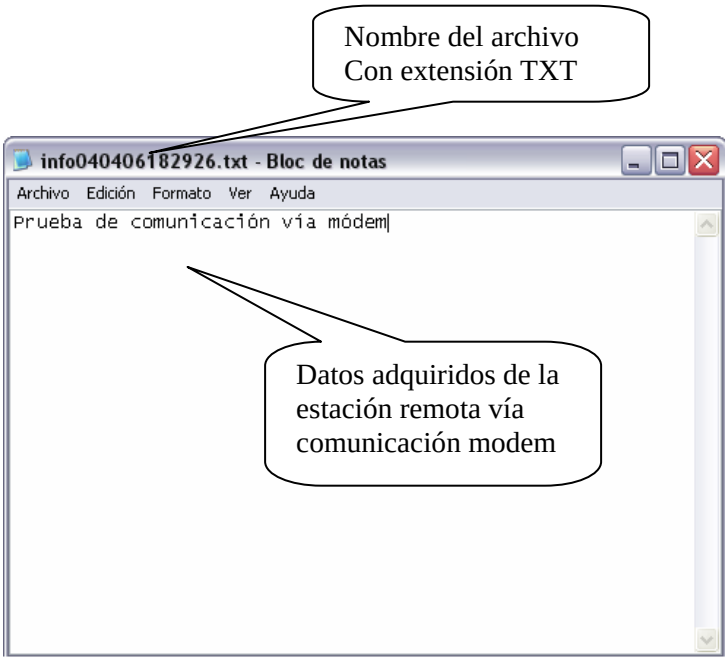


Figura 3.44. Archivo de datos vía comunicación modem.

Módulo de comunicación vía RADIO MODEM

Para el tipo de comunicación vía radio modem se sigue la misma filosofía que la transmisión vía modem telefónico, ya que ambos dispositivos emplean la comunicación RS232, sólo que en este módulo, en vez de marcar el número telefónico se pide al usuario un número identificador, que es el radio modem con el que se desea entablar la comunicación.

El programa está estructurado de igual manera en *Visual Basic*, y consiste de un formulario que contiene de igual forma la propiedades: *CommPort* (determinar el puerto), *Settings* (determina la velocidad en baudios, paridad y número de bits de datos y de parada) y *PortOpen* (para la apertura de puerto).

Los radio modem están configurados de tal manera que puedan realizar una comunicación punto a multipunto, es decir, que los radio modem instalados en las estaciones remotas son capaces de comunicarse con el radio modem que se encuentra en la estación central, de igual

manera, el radio modem de la estación central podrá comunicarse con cada uno de los radio modem instalados en las diferentes estaciones remotas. Por lo tanto, el *software* está desarrollado para que el usuario tenga acceso a los diferentes radios modems y logre transmitir la información requerida en su momento. La forma de identificar la información del radio modem que está transmitiendo es el contenido de etiquetas de empiezo y fin de información de envío (<1> información </1>), las cuales indican el número de radio modem que se está comunicando en ese momento, por ejemplo, si el radio modem de la estación central se comunica con el radio modem número 1, el radio modem número 1 enviará una contestación, como se muestra en la figura 3.45.

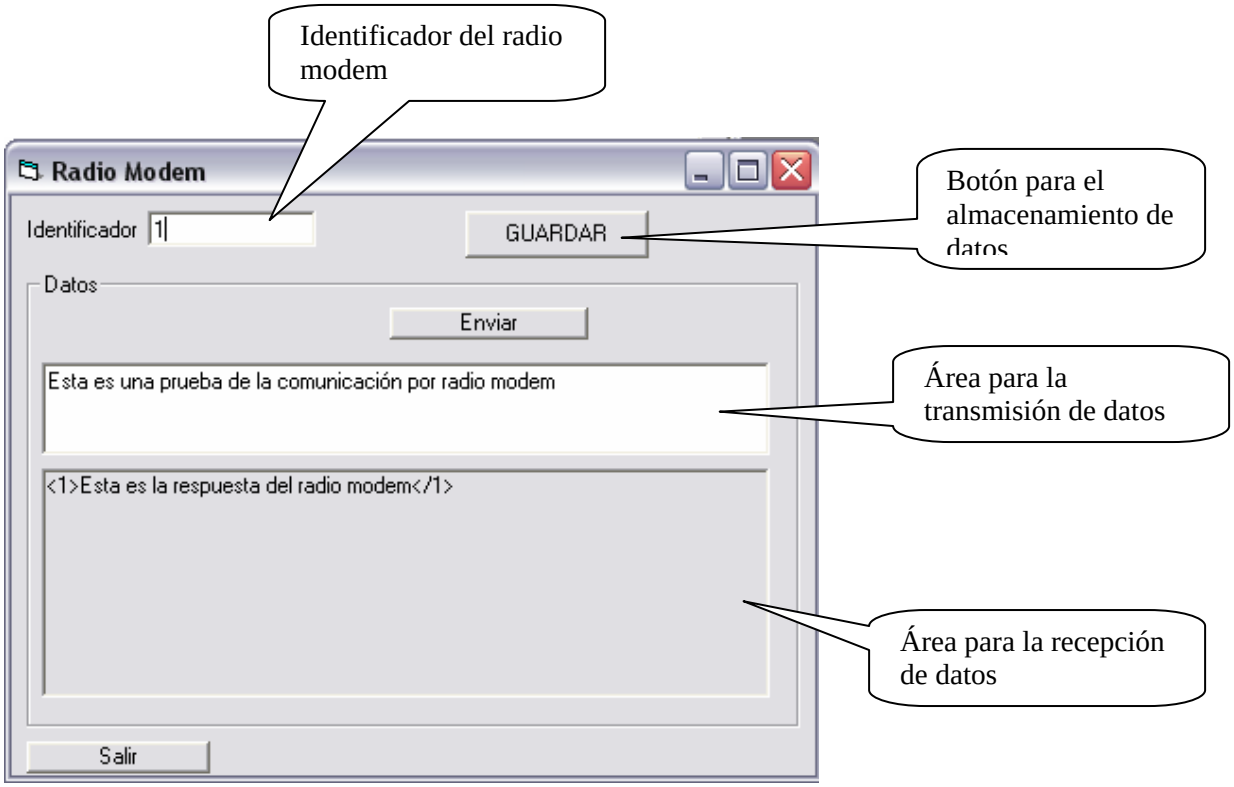


Figura 3.45. Transmisión de datos entre radio modems.

De igual manera que en la comunicación vía modem, es posible guardar los datos obtenidos desde la estación remota, esta datos se podrán almacenar en un archivo con extensión *.TXT*, para esta opción el *path* del archivo está definido en la programación, es decir si el usuario selecciona el botón de guardar, se creará el archivo *.TXT* con el *path* predeterminado, siguiendo con el formato de: día, mes, año, horas, minutos y segundos (ddmmyyhhmmss), figura 3.46.

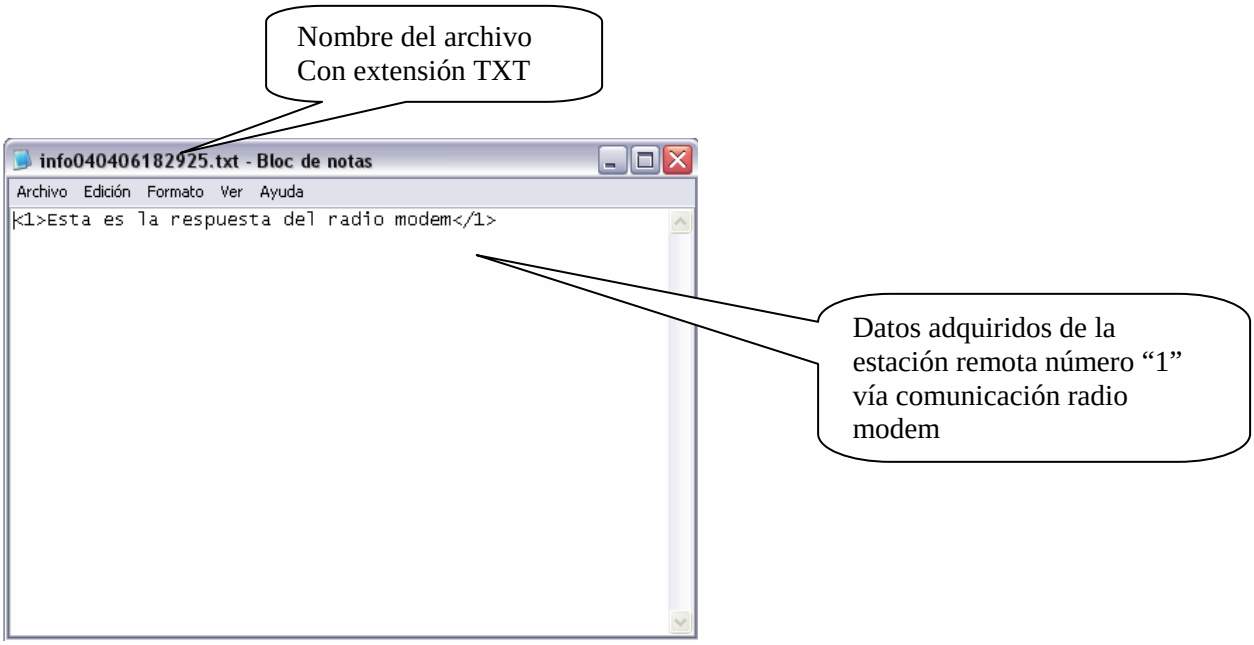


Figura 3.46. Registro de datos.

En cuanto al manejo de errores se programan los mismos que en la comunicación vía modem telefónico.

Módulo de comunicación vía GSM

La interfaz para la comunicación de datos vía GSM se desarrolló en *Visual Basic*, de igual manera emplea RS232, manejo de errores, y utiliza comandos Hayes extendidos.

El código de programación está estructurado en tres formularios: *frmComGSM, frmLeerMsg y frmSendMsg*, los cuales se describen a continuación:

- *frmComGSM*. Este formulario tienen la función de configurar el puerto serie (baudaje, bits de datos, paridad, bits de parada y control de flujo). También se programa el código de control para detección y corrección de errores de transmisión, como son: interrupciones, error de trama, datos perdidos, desbordamiento de búfer, error de paridad, CD, CTS, DSR, entre otros.

- *frmLeerMsg*. Este formulario contiene el código de lectura de información, donde se programa la secuencia de comandos necesarios para hacer dicha aplicación. Se utiliza el comando de lectura *AT+CMGL*, esperando su contestación de "OK", como se muestra en las líneas de código.

```
msCom.Output = "AT+CMGL" & vbCrLf        ‘comando para realizar lectura de datos
If PtoRead(vbCrLf & "OK" & vbCrLf) Then      ‘se espera confirmación de OK
```

- *frmSendMsg.* Este formulario tiene como objetivo transmitir datos; se programó la secuencia para que el usuario transmita los datos de una manera sencilla y rápida. Primero se programa en modo texto con el comando *AT+CMGF=1.* Posteriormente se utiliza el comando *AT+CMGS* para marcar el número identificador del teléfono celular, seguido de un carácter retorno carro <*CR*>. El teléfono móvil responde transmitiendo el carácter “>”, que indica que se encuentra listo para la información que se requiere. En el código siguiente se puede apreciar lo antes comentado.

```
msCom.Output = "AT+CMGF=1" & vbCrLf     ‘configura modo texto
If PtoRead("OK") Then                    ‘espera confirmación OK
   strBuff = ""
   msCom.Output = "AT+CMGS=" & """" & Trim(Me.txtNumCell.Text) & """" & vbCrLf
                                         ‘comando para el marcado del número de celular
   If PtoRead(">") Then                  ‘espera el signo > para estar listo
      strBuff = ""                       ‘limpia el búfer
      msCom.Output = Trim(Me.txtMsg.Text) & Chr(26) & Chr(8) & vbCrLf   ‘captura de información
      If PtoRead("OK") Then                                             ‘confirmacion OK
```

La interfaz de la estación central vía GSM inicia cuando se ejecuta la opción GSM en el menú principal, figura 3.36. Lo primero que realiza este programa es la verificación de la conexión con el dispositivo móvil, figura 3.47. Una vez ejecutada la acción anterior pueden suceder dos cosas: la primera que el dispositivo no se encuentre conectado o que exista un error en la conexión, si es así se desplegará una pantalla donde se indique lo sucedido, figura 3.48. En el segundo caso es cuando el dispositivo logra una conexión exitosa, figura 3.49, la cual activará el menú de herramientas, que esta constitutito por dos opciones, la primera es para el envió de datos y la segunda para lectura de los mismos, figura 3.50.

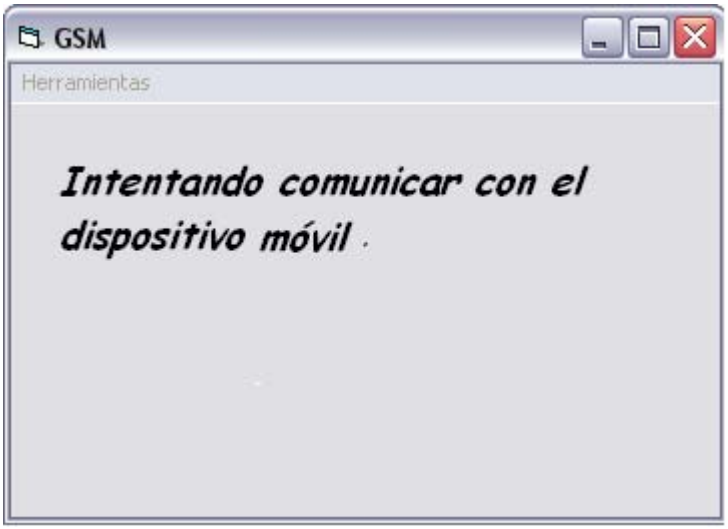


Figura 3.47. Comunicación de datos vía GSM.

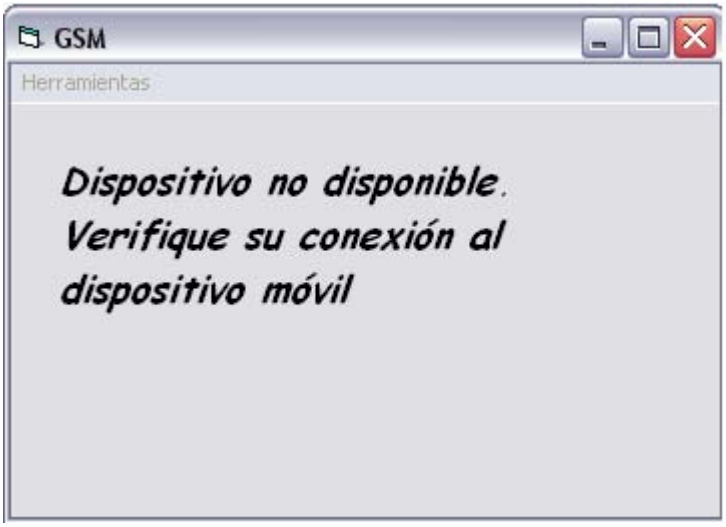


Figura 3.48. Error en la conexión vía GSM.

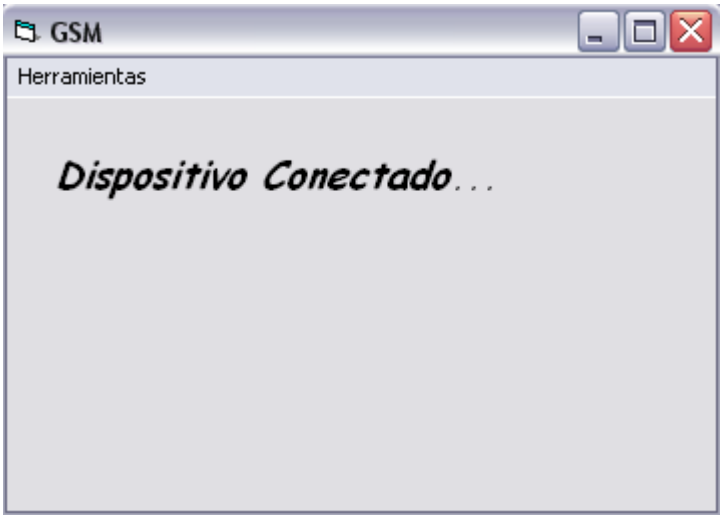


Figura 3.49. El dispositivo se encuentra conectado.

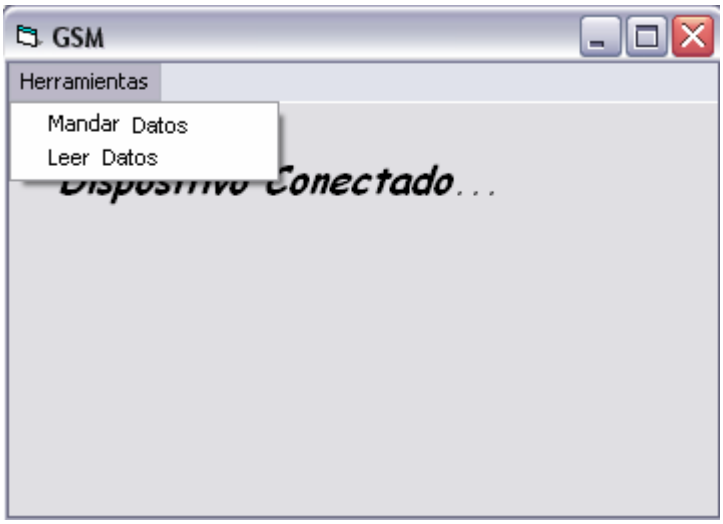


Figura 3.50. Menú de herramientas GSM.

Con respecto a la opción de enviar datos, se desplegará una ventana, solicitando el número de teléfono al que se desea transmitir la información; también se presenta una zona donde el usuario podrá inducir la información, esta información no podrá sobre pasar los 160 caracteres por norma, comentada en el capítulo 2. Una vez que el mensaje se ha escrito, existe un botón de enviar el mensaje, el cual mostrará una ventana con la leyenda de "LOS DATOS HAN SIDO ENVIADOS", figura 3.51. El problema que se puede presentar para la transmisión de datos es cuando no sea posible establecer comunicación con el servidor correspondiente.

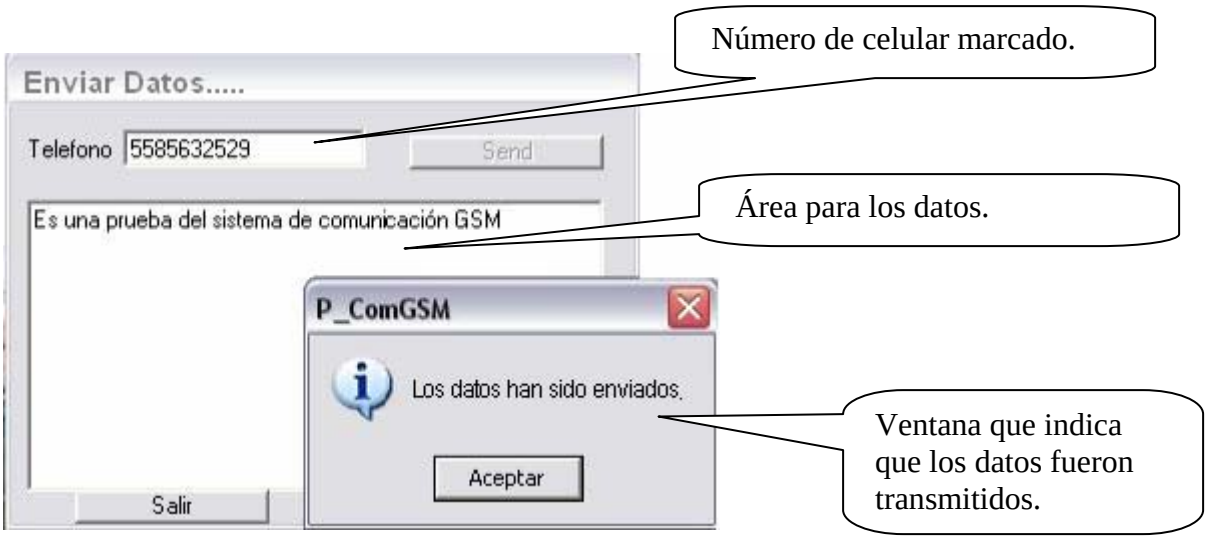


Figura 3.51. Envío de datos GSM.

En la opción de leer datos se desplegará una ventana, la cual muestra la información del dispositivo. Esta ventana incluye los comandos ejecutados *+CMGL*, comando de lectura; enseguida *REC UNREAD*, comando que indica que existen datos nuevos que no se les ha dado lectura, esta opción puede cambiar por los siguientes comandos, "*REC READ*": que indica que hay datos recibidos y leídos. Toda la información muestra el teléfono remitente, fecha y hora recibida, como se muestra en la figura 3.52.

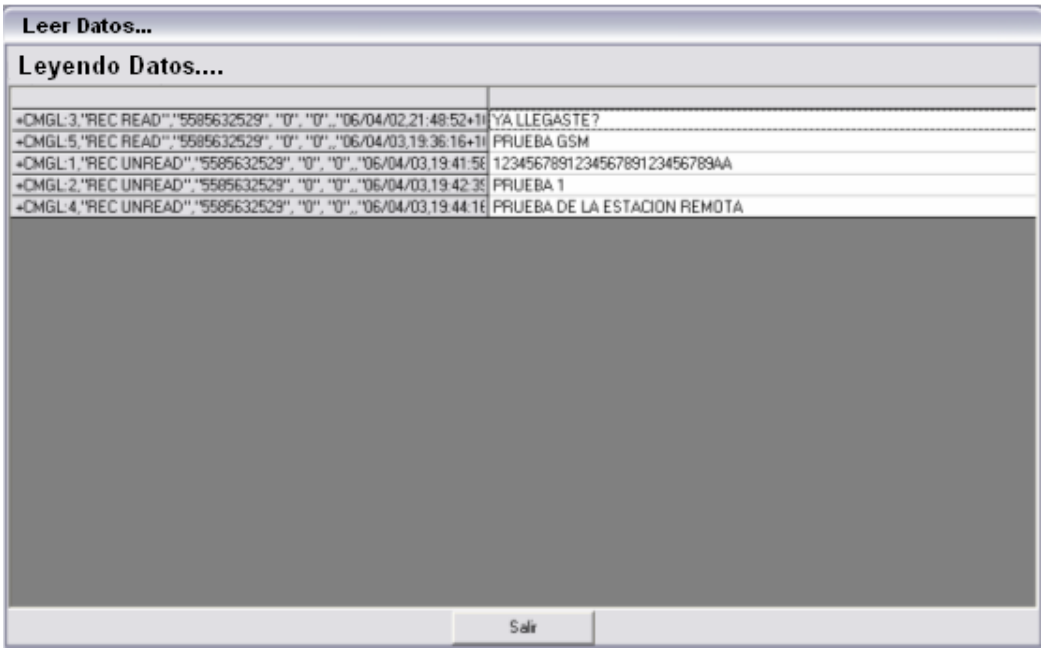


Figura 3.52. Mensajes GSM.

Módulo de comunicación para PDA

La segunda interfaz se realizó bajo la plataforma *Windows CE,* para la PDA, ésta fue desarrolla en lenguaje de alto nivel, *Visual Studio.NET 2003 de Microsoft*, el cual incluye clases para facilitar el desarrollo de aplicaciones para plataformas Pocket PC y Windows CE. La facilidad que da este lenguaje es que tiene tecnología de emulación de *Pocket PC* el cual permite crear y probar las aplicaciones de estos dispositivos.

De manera similar al caso descrito para la Laptop, los datos son recuperados de la estación remota y procesados en el dispositivo móvil. En este caso se empleo la comunicación vía infrarroja para la transmisión de dichos datos. Cabe destacar que cuando se trabaja con una PDA, automáticamente se ejecuta un emulador propio de *Windows CE* para realizar la transmisión de datos, ya que el dispositivo instalado en la estación remota es de alto nivel. Sin embargo, se probó el software libre *Visual Studio. NET,* el cual proporciona varios ejemplos de proyectos completos para dispositivos inteligentes, los cuales se pueden cargar, ejecutar y modificar de acuerdo a las necesidades del propio proyecto. En nuestro caso, se probó un programa que involucra el puerto infrarrojo de la PDA, donde el control está programado en Visual C++ y la aplicación está programada en Visual Basic. En la figura 3.53, se muestra la comunicación entre el dispositivo infrarrojo y una Pocket PC.

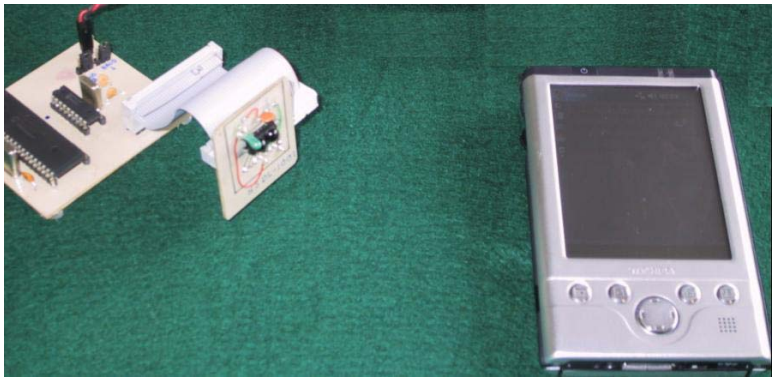


Figura 3.53. Comunicación vía inalámbrica con PDA.

Por último, se presentan las características mínimas que deberá contar el equipo de cómputo para poder ejecutar el sistema de comunicaciones de datos para estaciones remotas.

- Procesador Pentium o superior.
- Windows XP.

- 128 Mega bytes en memoria RAM.
- 30 Mega bytes de espacio disponible en disco duro.
- Puerto serial disponible.
- Puerto infrarrojo.
- Puerto USB.

Una vez realizado el diseño y desarrollo correspondiente a las interfaces de comunicación de datos, procederemos a la evaluación, resultados y conclusiones.

CAPÍTULO IV
EVALUACIÓN, RESULTADOS Y CONCLUSIONES

En el presente y último capítulo se presentan la evaluación, los resultados y conclusiones del prototipo de comunicaciones para estaciones remotas. Al final se hacen algunas recomendaciones para versiones posteriores de dicho sistema, ya que por tratarse de la primera versión, se requiere aún del desarrollo para la integración de los sistemas hasta aquí logrados, al limnígrafo del Instituto de Ingeniería de la UNAM.

IV.1. Evaluación

Una vez concluidos los prototipos de comunicaciones de datos para estaciones remotas, se realizaron pruebas de laboratorio para los diferentes módulos. En las pruebas se comprobó el funcionamiento correcto de cada módulo, tanto en hardware como en software de la estación remota y estación central. Entre las pruebas a las que fueron sometidos los módulos se encuentran las siguientes: la veracidad en los datos transmitidos, el almacenamiento de la información en la plataforma XP, la alineación de los dispositivos inalámbricos con comunicación infrarroja, y la identificación remota de: los módem, radio modem y teléfonos celulares con tecnología GSM; por último, se efectuaron pruebas de los posibles errores que

pueden presentarse durante el proceso de la comunicación. En este último caso se consideraron posibles errores de conexión de cables y dispositivos fuera de operación, así como la existencia de posibles errores de software, como es el caso de los microcontroladores y la estación central. Enseguida se describen con detalle las pruebas antes mencionadas.

- *Veracidad en los datos.* Debido a que no fue el alcance de la tesis la integración de los sistemas de comunicación al limnígrafo, las pruebas realizadas consistieron en simular datos en laboratorio que correspondieran a mediciones que son realizadas por dichos dispositivos. Los datos adquiridos fueron transmitidos por los diferentes módulos de comunicación que, una vez que fueron procesados y presentados, se realizó una verificación de la información obtenida cotejando con datos que fueron tomados como base. Estas pruebas comprobaron el correcto funcionamiento en la recepción de información en los diferentes módulos de información.

- *Almacenamiento de la información en la plataforma XP.* Fueron desarrolladas y probadas tres diferentes formas de almacenamiento de información, las cuales son: la selección por parte del usuario de la ubicación del archivo de datos, ésta puede ser en cualquier *path* o ruta en la plataforma que se está trabajando; la segunda forma de almacenamiento consiste en almacenar la información en un *path* predeterminado por el software y la última forma de realizar este proceso es que conforme llega la información del módulo de comunicación, automáticamente es almacenada en una ruta predeterminada de la plataforma, es decir, no es necesaria la intervención del usuario para el almacenamiento de los datos, mientras que en las dos primeras sí lo es. Las tres formas de almacenar los datos resultaron exitosas, al no existir ningún error en este proceso.

- *Alineación de los dispositivos inalámbricos*. En lo que corresponde al módulo de comunicación vía infrarroja, las pruebas realizadas fueron las siguientes: la consideración del ángulo de la alineación y distancia entre la estación remota y los dispositivos portátiles, así como la compatibilidad de detección entre diferentes equipos. En estas pruebas se comprobó que el ángulo al que puede llegar a transmitir el dispositivo sin ningún problema es de 15 grados con una distancia máxima de tres metros, además, los equipos portátiles presentaron compatibilidad con el sistema de comunicación infrarrojo. De igual forma se verificó el sistema con la presencia de más de

un dispositivo, destacando que la transmisión sólo es posible llevarla a cabo punto a punto.

- *Identificación de dispositivos*. Para los módulos de comunicación correspondientes al modem, GSM y radio modem, fue necesario la implementación de identificadores para los dispositivos, estos fueron implementados por software para la detección de números telefónicos en el caso del modem y GSM. En el caso del radio modem son utilizados solo números para la identificación de dichos dispositivos. Las pruebas consistieron en la realización del marcado a diferentes dispositivos, comprobando que en ningún tipo de comunicación existieron problemas.

- *Pruebas al prototipo de comunicaciones bajo ciertas condiciones no esperadas.* Los sistemas fueron sometidos a pruebas de error a propósito, entre las que destacan: el error en la conexión del cable, la transferencia interrumpida de datos y el error en algún parámetro mal configurado; por ejemplo, la selección del puerto de comunicación. Ante estas situaciones se comprobó que los diferentes sistemas fueron capaces de indicar al usuario el error en el instante que fue detectado, permitiendo a su vez tomar la decisión para la solución al problema y así continuar con el funcionamiento del sistema.

Cabe comentar que todas las pruebas se hicieron a nivel laboratorio y considerando comunicaciones punto a punto.

Una vez comentadas las pruebas a que se sometieron los diferentes módulos de comunicación, son presentados los resultados y conclusiones.

IV.2. Resultados

Para alcanzar los objetivos planteados en el diseño de un prototipo de comunicaciones de datos, se planteó una metodología, consistente de varios pasos. En el primero se realizó el análisis e investigación de las necesidades de comunicación del limnígrafo electrónico desarrollado en la UNAM, esto, para llevar a cabo una determinación de requerimientos del sistema.

La segunda etapa correspondió al diseño y desarrollo de los sistemas, tanto del hardware como del software de la estación remota y estación central. En cuanto al hardware son considerados

el diseño y la selección de dispositivos que integrarían al sistema. Este es un punto importante para el desarrollo del prototipo, puesto que es necesario tomar en cuenta que el prototipo se crea para estaciones remotas, las cuales tienen ciertos parámetros importantes que se deben tomar en cuenta, como lo es la fuente de alimentación y un amplio rango de temperatura de operación. Cabe destacar que también se desarrolló el software necesario para los microcontroladores que se encuentran en las estaciones remotas. Por último, se desarrolló un programa principal para la estación central, el cual conjunta los diferentes programas de cada módulo de comunicación, ya que por sus características, estos no pudieron compartir un solo lenguaje de programación.

La tercera etapa consistió en el desarrollo de las diferentes pruebas de laboratorio que se llevaron a cabo al prototipo de comunicaciones, en el cual fueron evaluados el desempeño del sistema en condiciones óptimas y en condiciones extremas de operación.

Concluyendo la fase de pruebas, se obtuvo una primera versión del prototipo de comunicaciones para estaciones remotas, la cual cumple con los medios de comunicación alámbricos e inalámbricos necesarios en una estación remota y estación central. Entre las características del prototipo de comunicaciones de datos para estaciones remotas, se pueden mencionar las siguientes:

- Contiene seis módulos de comunicación, los cuales son: RS-232, USB, modem, IrDA, radio modem y GSM.
- Los módulos de comunicación pueden operar a temperaturas de -20°C y 70°C requeridos en el limnígrafo digital.
- El sistema responde para equipos de cómputo (PC), equipos portátiles (Laptop) y PDA. Los cuales cuentan con un programa de comunicación que se ejecuta bajo la plataforma *Windows XP* o *Windows CE*, que permiten la adquisición de la información almacenándola en archivos de formato de texto.

IV.3. Conclusiones

Se cumplió el objetivo del proyecto, el cual fue, diseñar e implementar un sistema de comunicación de datos, en el cual están incluidos tanto comunicaciones de tipo alámbricos (RS232, USB y modem telefónico) e inalámbricos (IrDA, radio modem y GSM). El prototipo fue creado para integrarlo al limnígrafo digital desarrollado en la UNAM. El sistema de

comunicación en general trae consigo el ahorro de tiempo en la adquisición de datos, además que en algunos casos no es necesario acercarse hasta los puntos donde son localizados los dispositivos de medición para obtener los datos, por lo tanto la versatilidad de los módulos de comunicación, hace que el usuario elija el medio de comunicación más óptimo, dependiendo de sus necesidades.

La evaluación del prototipo de comunicaciones comprobó que es un sistema confiable para la transmisión de información, además, de contar con los módulos de comunicación necesarios para equipar a las estaciones remotas. Los módulos de comunicación desarrollados presentan ventajas, así como algunas desventajas, ambas se describen a continuación:

- Las comunicaciones vía RS232 y USB son importantes en nuestro diseño, aunque la primera tiende a desaparecer, aún existen en el mercado dispositivos que cuentan con este tipo de conexión. En cuanto a la comunicación USB, ésta tiene la ventaja de que en la actualidad todo dispositivo cuenta con este tipo de conexión.
- La comunicación inalámbrica vía infrarroja (IrDA) fue implementada en alto nivel, con la ventaja de que cualquier dispositivo portátil o móvil pueda tener acceso a la información de la estación remota, con una alta seguridad en los datos. La desventaja de este medio de comunicación es la restricción que impone su ángulo de apertura, para poder establecer comunicación con cualquier otro dispositivo, tienen que estar alineados, además que este medio de comunicación se realiza punto a punto.
- Para el caso de la comunicación vía modem, se debe contar con una línea telefónica. La disponibilidad de las líneas dedicadas es la desventaja principal en este medio de comunicación, más aun cuando se requiere el tratamiento de varios puntos de control. Una ventaja sobre este medio de comunicación es que se puede realizar una comunicación punto-multipunto.
- Para la comunicación vía radio modem se debe contar con la disponibilidad del espectro de frecuencia, para poder realizar dicha comunicación. De igual manera cuenta con la ventaja de que es posible realizar una comunicación punto-multipunto; además de la facilidad de la instalación de los radio modems.
- Para la comunicación vía telefonía celular GSM, se utiliza el servicio SMS, lo único que se debe tener en cuenta es la cobertura del dispositivo y las tarifas de las compañías telefónicas. Esta comunicación es punto-multipunto.

IV.3. Recomendaciones

El prototipo de comunicaciones de datos para estaciones remotas, es una primera versión, así que este puede ser mejorado y optimizado tanto el hardware como el software y lograr un mejor desempeño de dicho prototipo. Por ejemplo, en el software es recomendable hacer que el sistema sea automático, es decir, que en ciertos periodos de tiempo sea capaz de comunicarse la estación central con los diferentes sistemas de las estaciones remotas para establecer la transmisión de información sin intervención del usuario; además de que los datos sean almacenados en una base de datos mediante identificadores de hora y fecha recibidos, contando con las opciones de graficación e impresión de los datos. En cuanto a las mejoras en hardware está la integración del prototipo de comunicaciones al limnígrafo digital es el próximo paso a seguir en este proyecto. Ya que para esta primera versión sólo se diseñaron e implementaron los diferentes módulos de forma aislada, por lo tanto es necesaria la unificación de los módulos de comunicación, diseñando circuitos para la operación óptima de los diferentes módulos de comunicación, para el ahorro de energía en la estación remota.

En cuanto a la transmisión de datos se pueden diseñar algoritmos para la seguridad de los datos, para que la información sea 100% confiable y además pueda ser de carácter exclusivo, si así se requiere.

Con respecto a los módulos de comunicación, es recomendable equipar al sistema de comunicaciones con un módulo de comunicación satelital, para cubrir grandes zonas geográficas, mediante terminales fijas o móviles.

Como el sistema debe estar operando de manera continua, es decir, durante las 24 horas del día y los 365 días del año, los sistemas deberán ser puestos a prueba sin descanso alguno durante un periodo considerable de tiempo, transmitiendo datos a los medios de almacenamientos correspondientes.

Se podría diseñar una página Web que muestre los enlaces de comunicación de las estaciones remotas, mediante el uso de Internet.

APÉNDICE A
PROGRAMACIÓN HARDWARE

PROGRAMACIÓN INTERFAZ IRDA

```
;*******************************************************************************
; Programa del controlador principal para comunicación IrDA, programado en MPLAB
;  PIC  PIC16F877, reloj de 20.00MHz
;  Versión 1.1, última modificación  18 de diciembre del 2005
;*******************************************************************************
          LIST      C=132
          include   P16F877.inc
     ERRORLEVEL -302

#define   reset     H'00'      ; Reset vector
          __CONFIG _CP_OFF & _PWRTE_ON & _HS_OSC & _WDT_OFF ; configuración del microcontrolador
          __IDLOCS H'0010'
;         PORT Bits
#define rxd    PORTC, 7       ; terminal de entrada serial
#define txd    PORTC, 6       ; terminal de salida serial
#define cts    PORTD, 7       ;  CTS
#define rts    PORTD, 6       ;  RTS
#define dtr    PORTD, 5       ;  DTR
#define dsr    PORTD, 4       ;  DSR
#define cd     PORTD, 3       ; CD
#define ri     PORTD, 2       ;  RI
#define rst2150 PORTC,0       ; terminal para reseteo del dispositivo

                              ; Configuraciones para establecer el baudaje
B9600at20MHz   EQU   D'129'
B19200at20MHz  EQU   D'64'
B57600at20MHz  EQU   D'21'
B115200at20MHz EQU   D'10'
;

     org    H'00'       ;  vector de inicio
     goto   START
;*******************************************************************************
; Configuración de  los puertos del microcontrolador
; *****************************************************************************
START  clrf   STATUS
          movlw  0xFF
          movwf  PORTA
          movwf  PORTB
          movwf  PORTC
          movwf  PORTD
          movwf  PORTE
          bsf        STATUS, RP0
          movlw    ddra
          movwf    TRISA
          movlw    ddrb
          movwf    TRISB
          movlw    ddrc
          movwf    TRISC
          movlw    ddrd
          movwf    TRISD
          movlw    ddre
          movwf    TRISE
          movlw    cfgopt
          movwf    OPTION_REG
;*******************************************************************************
; Configuración de la UART  con un baudaje de 9600
;*******************************************************************************
     movlw  0x24                    ; BRGH = 1, 8-bit, TX Enabled, Async.
     movwf  TXSTA
     movlw  B9600at20MHz
```

```
      movwf   SPBRG
      clrf    STATUS              ; Banco 0
      movlw   0x90                ; Enable  para el puerto serial
      movwf   RCSTA
      clrf    PORTB               ; limpia el puerto PORB
;*********************************************************************
;Programación  para las señales de control: Reseteo, DTR , DSR
;*********************************************************************
RESET2150
                  bcf    dtr          ;  DTR en bajo
                  bcf    rst2150      ;  reseteo
                  nop                 ; Tiempo para detección
                  nop
                  bsf    rst2150
                  movlw  H'FF'
                  call   DELAY        ;llamada a una rutina de espera

WAIT2150                              ; Verificación de la bandera DSR
                  btfss  dsr
                  goto   WAIT2150     ; Espera más tiempo
                  goto     MAIN       ; Continua a la etiqueta MAIN
MAIN

WAITCD            btfsc  cd           ;  verificación de la terminal CD, para establecer comunicación
                  goto   WAITCD       ;  Espera para establecer la comunicación
                  bcf    rts          ;  Si se establece la comunicación poner en bajo RTS
RXWAIT1
                  btfsc  PIR1, RCIF   ; Verificación para recepción de byte
                  goto   GOTBYTE1     ;  Si existe información
                  btfsc  cd           ; No,  checa la bandera CD
                  goto   MAIN         ; No, checa si existe comunicación
                  goto   RXWAIT1      ; Si, No recibió byte
GOTBYTE1
                  movf     RCREG, W   ; Obtiene el byte del registro W
                  movwf    PORTB      ;  Despliega los byte por el puerto B
                  call     SENDDATA   ; Envía al menu la cadena
LP4EVER           goto   LP4EVER      ;  Se completa la transmisión

*************************************************************************************
; Rutina de transmisión de datos al circuito integrado (transceptor)
;*************************************************************************************
SENDDATA
                  clrf     MENUCNTR       ; MENU contador = 0
                  call     MENU           ; Se obtiene el próximo byte  de datos desde la tabla MENU
MENU Data Table
                  movwf    MENUBYTES      ;  Número de bytes en el MENU
MENULOOP1
                  incf     MENUCNTR, F    ; Puntero para la próxima localidad del MENU
 MENU
                  call     MENU           ; Llamada a la rutina MENU

MENULP1           btfsc    cd             ; Verificación de CD
                  goto     RESET2150      ; No, comunicación interrumpida
RESET MCP2150
                  btfsc    cts            ; Sí, puede enviar dato
                  goto     MENULP1        ; No, esperar a la lectura de datos
                  call     SERSND         ; Sí, envía datos
                  decf     MENUBYTES, F   ; Decrementa el número bytes disposnibles
                  btfss    STATUS, Z      ; Verificación de la bandera Z del registro STATUS
                  goto     MENULOOP1      ; More of the MENU needs to be sent
                  return                  ; Regresa a la etiqueta main
;*************************************************************************************
```

```
; Rutina para el envío de datos
;*****************************************************************************************
SERSND   bsf      STATUS, RP0              ; Banco 1
SERSLP   btfss    TXSTA, TRMT              ; Verificación de la UART
         goto     SERSLP                   ; No, espera
         bcf      STATUS, RP0              ; Banco  0
SERS1    btfsc    cd                       ; Verificación de la comunicación
         goto     RESET2150                ; No, comunicación interrumpida
         btfsc    cts                      ; Verificación de la bandera CTS
         goto     SERS1                    ; regresa a la etiqueta SERS1
         movf     hostdata,w               ; Obtiene el byte de envío
         movwf    TXREG                    ; Envío del byte
         return
;*****************************************************************************************
; Rutina de consumo de tiempo
;*****************************************************************************************
DELAY    movwf    delreg
DELLP    nop
         decfsz   delreg,f
         goto     DELLP
         retlw    0
;*****************************************************************************************
; Tabla de datos
;*****************************************************************************************
MENU     movlw    HIGH (MENU)              ; Get the upper address bits where this table is located
         movwf    PCLATH                   ;   and load into the PCLATH register
         movf     MENUCNTR, W              ; get the offset
         addwf    PCL,f                    ; add the offset to PC
         DT       D'239'                   ; the first byte is the byte count          ; 1 Characters
         DT       "12345678", 0x0D, 0x0A   ; 10 Characters
         DT       "2BCDEFGH", 0x0D, 0x0A                        ; 10 Characters
         DT       "32345678", 0x0D, 0x0A                       ; 10 Characters
         DT       "4bcdefgh", 0x0D, 0x0A                       ; 10 Characters
         DT       "52345678", 0x0D, 0x0A
         DT       "6BCDEFGH", 0x0D, 0x0A
         DT       "72345678", 0x0D, 0x0A
         DT       "8bcdefgh", 0x0D, 0x0A
         DT       "92345678", 0x0D, 0x0A
         DT       "ABCDEFGH", 0x0D, 0x0A
         DT       "B2345678", 0x0D, 0x0A
         DT       "Cbcdefgh", 0x0D, 0x0A
         DT       "D2345678", 0x0D, 0x0A
         DT       "EBCDEFGH", 0x0D, 0x0A
         DT       "F2345678", 0x0D, 0x0A
         DT       "1bcdefgh", 0x0D, 0x0A
         DT       "22345678", 0x0D, 0x0A
         DT       "3BCDEFGH", 0x0D, 0x0A
         DT       "42345678", 0x0D, 0x0A
         DT       "5bcdefgh", 0x0D, 0x0A
         DT       "62345678", 0x0D, 0x0A
         DT       "7BCDEFGH", 0x0D, 0x0A
         DT       "82345678", 0x0D, 0x0A

    end
```

PROGRAMACIÓN INTERFAZ GSM

```
;*********************************************************************************
; Programa del controlador principal para comunicación IrDA, programado en MPLAB
;  PIC  PIC16F877, reloj de 20.00MHz
;  Versión 1.1, última modificación  18 de diciembre del 2005
;*********************************************************************************

.INCLUDE "8515DEF.INC"

.org    $000                                ;VECTOR DE INICIO
        rjmp    RESET

RESET:  LDI     R16,LOW(RAMEND)             ;CARGA EL APUNTADOR DE PROGRAMA
        OUT     SPL,R16                     ; VALOR DEL REGISTRO PARA LA CONFIGURACIÓN

        LDI R16,HIGH(RAMEND)                ;CONFIGURACIÓN DE REGISTRO PARTE ALTA

        LDI     R17,$FF                     ; CARGA VALOR AL REGISTRO R17
        OUT     DDRb,R17                    ;CONFIGURA PUERTO A COMO SALIDA

        LDI     R17,$00                     ;ENCIENDE LEDS
        OUT     PORTb,R17                   ;SALIDA DE LED´S EN ALTO

        LDI     R17,23                      ; CARGA VALOR AL REGISTRO R17
        OUT     UBRR,R17                    ;VELOCIDAD = 9600 BAUDS

        LDI     R17,0B00011000              ; CARGA VALOR AL REGISTRO R17
        OUT     UCR,R17                     ;HABILITA RECEPCIÓN Y TRANSMISIÓN
;*********************************************************************
;ENVÍA AT PARA RESPONDER EL OK
;*********************************************************************
AGAIN:  RCALL DELAY
        LDI     R17,'A'                 ; CARGA VALOR DE ‘A’ AL REGISTRO R17
        OUT     UDR,R17                 ;TRANSMITE LA A

TX_A:   SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITA EL DATO
        RJMP    TX_A                    ; MIENTRAS NO SE TRANSMITA VERIFICA
        SBI     USR,TXC                 ;ESPERA A QUE SE TRANSMITA
        RCALL   DELAY
        LDI     R17,'T'                 ; CARGA VALOR DE ‘A’ AL REGISTRO R17
        OUT     UDR,R17                 ;TRANSMITE LA T

TX_T:   SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITE LA T
        RJMP    TX_T                    ; MIENTRAS NO SE TRANSMITA VERIFICA
        SBI     USR,TXC

ESPERA_A: SBIS  USR,RXC
        RJMP    ESPERA_A                ; MIENTRAS NO SE TRANSMITA VERIFICA
        IN      R17,UDR
        CPI     R17,'A'
        BRNE    ERROR_A                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

ESPERA_T: SBIS  USR,RXC                 ; RECIBE CARACTER
        RJMP    ESPERA_T                ; MIENTRAS NO SE TRANSMITA VERIFICA
        IN      R17,UDR
        CPI     R17,'T'
        BRNE    ERROR_T                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
        RCALL   DELAY
        RCALL   DELAY

        LDI     R17,$0D                 ; CARGA VALOR DE ‘A’ AL REGISTRO R17
        OUT     UDR,R17                 ;TRANSMITE ENTER

TX_E:   SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITA ENTER
        RJMP    TX_E                    ; MIENTRAS NO SE TRANSMITA VERIFICA
        SBI     USR,TXC                 ;ESPERA A QUE SE TRANSMITA EL DATO
        SBI     PORTB,PB6
```

```
ESPERA_LF:   SBIS   USR,RXC          ;ESPERA LF ESPERA DOS ENTER
             RJMP   ESPERA_LF        ; MIENTRAS NO SE TRANSMITA VERIFICA
             IN     R17,UDR
             SBI    PORTB,PB5
             CPI    R17,$0D          ; COMPARA EL PRIMERO
             BRNE   ERROR_LF         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

ESPERA_LF2:  SBIS   USR,RXC          ;ESPERA SEGUNDO LF
             RJMP   ESPERA_LF2       ; MIENTRAS NO SE TRANSMITA VERIFICA
             IN     R17,UDR
             SBI    PORTB,PB5
             CPI    R17,$0D          ; COMPARA EL SEGUNDO
             BRNE   ERROR_LF2        ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

ESPERA_CR:   SBIS   USR,RXC          ;ESPERA CR EL SALTO DE LINEA LA A
             RJMP   ESPERA_CR        ; MIENTRAS NO SE TRANSMITA VERIFICA
             IN     R17,UDR
             SBI    PORTB,PB4
             CPI    R17,$0A
             BRNE   ERROR_CR         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

ESPERA_O:    SBIS   USR,RXC          ;ESPERA O
             RJMP   ESPERA_O         ; MIENTRAS NO SE TRANSMITA VERIFICA
             IN     R17,UDR
             CBI    PORTB,PB6
             CPI    R17,'O'
             BRNE   ERROR_O

ESPERA_K:    SBIS   USR,RXC          ;ESPERA K
             RJMP   ESPERA_K         ; MIENTRAS NO SE TRANSMITA VERIFICA
             IN     R17,UDR
             CPI    R17,'K'
             BRNE   ERROR_K          ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
             LDI    R20,$AA          ;SI LLEGÓ HASTA AQUÍ TODO ESTUVO BIEN
             OUT    PORTB,R20        ;PRENDE UNO SI Y OTRO NO
             RCALL  DELAY


ESPERA_U:    SBIS   USR,RXC          ;ESPERA SALTO DE LINEA
             RJMP   ESPERA_U         ; MIENTRAS NO SE TRANSMITA VERIFICA
             IN     R17,UDR
             RJMP   CMGF             ; RUTINA CMGF

ERROR_A:     LDI    R21,$01          ; CARGA VALOR DE 'A' AL REGISTRO R17
             OUT    PORTB,R17
             RCALL  DELAY
             RCALL  DELAY
             RCALL  DELAY
             RCALL  DELAY
             RJMP   AGAIN

ERROR_T:     LDI    R21,$02          ; CARGA VALOR DE 'A' AL REGISTRO R17
             OUT    PORTB,R17
             RCALL  DELAY
             RCALL  DELAY
             RCALL  DELAY
             RCALL  DELAY
             RJMP   AGAIN

ERROR_LF:    LDI    R21,$03          ;ERROR AL RECIBIR LF
             OUT    PORTB,R17
             RCALL  DELAY
             RJMP   AGAIN            ; MIENTRAS NO SE TRANSMITA VERIFICA

ERROR_LF2:   LDI    R21,$04          ;ERROR AL RECIBIR LF
             OUT    PORTB,R17
             RCALL  DELAY
             RCALL  DELAY
             RCALL  DELAY
             RCALL  DELAY
```

```
                RJMP    AGAIN                   ; MIENTRAS NO SE TRANSMITA VERIFICA

ERROR_CR:       LDI     R21,$05                 ;ERROR AL RECIBIR CR
                OUT     PORTB,R17
                RCALL   DELAY
                RCALL   DELAY
                RJMP    AGAIN                   ; MIENTRAS NO SE TRANSMITA VERIFICA
ERROR_O:        LDI     R21,$06                 ; CARGA VALOR DE 'A' AL REGISTRO R17
                OUT     PORTB,R17
                RJMP    AGAIN
ERROR_K:        LDI     R21,$07                 ;ERROR AL RECIBIR K
                OUT     PORTB,R17

;*****************************************************************
; PROGRAMACIÓN DEL COMANDO  AT+CMGF=1
;*****************************************************************

CMGF:           LDI     R17,'A'
                OUT     UDR,R17                 ;TRANSMITE LA A

TX_A1:          SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITA LA A
                RJMP    TX_A1                   ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC                 ;ESPERA A QUE SE TRANSMITA DATO
                RCALL   DELAY                   ;LLAMA A LA RUTINA DELAY
                LDI     R17,'T'                 ;CARGA DATO EN EL REGISTRO R17
                OUT     UDR,R17                 ;TRANSMITE LA T
TX_T1:          SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITE LA T
                RJMP    TX_T1                   ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC                 ;ESPERA A QUE SE TRANSMITA EL DATO
ESPERA_A1:      SBIS    USR,RXC
                RJMP    ESPERA_A1               ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'A'                 ;COMPARA DATO
                BRNE    ERROR_AA                ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    ESPERA_T1               ; MIENTRAS NO SE TRANSMITA VERIFICA

ERROR_AA:       RJMP    ERROR_M                 ; MIENTRAS NO SE TRANSMITA VERIFICA
ESPERA_T1:      SBIS    USR,RXC                 ; RECIBE CARACTERER
                RJMP    ESPERA_T1               ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'T'
                BRNE    ERROR_TT
                RJMP    MAS                     ; MIENTRAS NO SE TRANSMITA VERIFICA
ERROR_TT:       RJMP    ERROR_M                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

MAS:            LDI     R17,'+'
                OUT     UDR,R17                 ;TRANSMITE EL +
TX_MAS:         SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITA LA +
                RJMP    TX_MAS                  ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC
ESPERA_MAS:     SBIS    USR,RXC
                RJMP    ESPERA_MAS              ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'+'
                BRNE    ERROR_MASS
                RJMP    CE                      ; MIENTRAS NO SE TRANSMITA VERIFICA
ERROR_MASS:     RJMP    ERROR_M                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

CE:             LDI     R17,'C'
                OUT     UDR,R17                 ;TRANSMITE LA C

TX_C:           SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITA LA +
                RJMP    TX_C                    ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC                 ;ESPERA A QUE SE TRANSMITA EL DATO

ESPERA_C:       SBIS    USR,RXC                 ; RECIBE CARACTERER
                RJMP    ESPERA_C                ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'C'                 ;COMPARA DATO
                BRNE    ERROR_CC
```

```
                RJMP    EM              ; MIENTRAS NO SE TRANSMITA VERIFICA

ERROR_CC:       RJMP    ERROR_M         ; MIENTRAS NO SE TRANSMITA VERIFICA
EM:             LDI     R17,'M'         ; CARGA DATO EN EL REGISTRO R17
                OUT     UDR,R17         ;TRANSMITE LA C

TX_M:           SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA +
                RJMP    TX_M            ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC

ESPERA_M:       SBIS    USR,RXC         ; RECIBE CARACTERER
                RJMP    ESPERA_M        ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'M'
                BRNE    ERROR_MMM       ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    GE
ERROR_MMM:      RJMP    ERROR_M         ; MIENTRAS NO SE TRANSMITA VERIFICA

GE:             LDI     R17,'G'
                OUT     UDR,R17         ;TRANSMITE LA C

TX_G:           SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA DATO
                RJMP    TX_G            ; MIENTRAS NO SE TRANSMITA VERIFICA

                SBI     USR,TXC         ;ESPERA A QUE SE TRANSMITA DATO

ESPERA_G:       SBIS    USR,RXC         ; RECIBE CARACTERER
                RJMP    ESPERA_G        ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'G'         ; COMPARA DATO
                BRNE    ERROR_GG        ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    EF              ; MIENTRAS NO SE TRANSMITA VERIFICA

ERROR_GG:       RJMP    ERROR_M         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

EF:             LDI     R17,'F'
                OUT     UDR,R17         ;TRANSMITE LA F

TX_F:           SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA F
                RJMP    TX_F            ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO
ESPERA_F:       SBIS    USR,RXC         ; RECIBE CARACTERER
                RJMP    ESPERA_F        ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'F'         ; COMPARA DATO
                BRNE    ERROR_FF        ; SI EXISTE ERROR SALTA A ERROR
                RJMP    IGUAL           ; MIENTRAS NO SE TRANSMITA VERIFICA
ERROR_FF:       RJMP    ERROR_M         ; SI EXISTE ERROR SALTA A ERROR_M
IGUAL:          LDI     R17,'='
                OUT     UDR,R17         ;TRANSMITE LA C

TX_IGUAL:       SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA +
                RJMP    TX_IGUAL        ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC

ESPERA_IGU:     SBIS    USR,RXC         ; RECIBE CARACTERER
                RJMP    ESPERA_IGU      ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'='
                BRNE    ERROR_IGU1      ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    UNO1            ; MIENTRAS NO SE TRANSMITA VERIFICA
ERROR_IGU1:     RJMP   ERROR_M          ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
UNO1:           LDI     R17,'1'
                OUT     UDR,R17         ;TRANSMITE LA C

TX_UNO:         SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO
                RJMP    TX_UNO          ; MIENTRAS NO SE TRANSMITA VERIFICA
```

```
                IN      R17,UDR
                CPI     R17,'K'                 ; COMPARA DATO
                BRNE    ERRORKK                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                LDI     R20,$AA                 ;SI LLEGÓ HASTA AQUÍ TODO ESTUVO BIEN
                OUT     PORTB,R20               ;PRENDE UNO SI Y OTRO NO
                RJMP    ESPERA_UU
ERRORKK:        RJMP    ERROR_M                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

ESPERA_UU:      SBIS    USR,RXC                 ;ESPERA SALTO DE LINEA
                RJMP    ESPERA_UU
                IN      R17,UDR
;*****************************************************************************************************************
;AT+CMGS="5585632529"
;*****************************************************************************************************************
CMGS:           LDI     R17,'A'                 ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17                 ;TRANSMITE LA A

TX_A2:          SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITA LA A
                RJMP    TX_A2                   ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC

                RCALL   DELAY
                LDI     R17,'T'                 ; CARGA VALOR DE 'T' AL REGISTRO R17
                OUT     UDR,R17                 ;TRANSMITE LA T

TX_T2:          SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITE LA T
                RJMP    TX_T2                   ; MIENTRAS NO SE TRANSMITA VERIFICA

                SBI     USR,TXC                 ;ESPERA A QUE SE TRANSMITA EL DATO
ESPERA_A2:      SBIS    USR,RXC
                RJMP    ESPERA_A2               ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'A'                 ; COMPARA DATO
                BRNE    ERRORAA                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    ESPERA_T2

ERRORAA:        RJMP    ERROR_M

ESPERA_T2:      SBIS    USR,RXC                 ; RECIBE CARACTERER
                RJMP    ESPERA_T2               ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'T'
                BRNE    ERRORTT                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    MAS1

ERRORTT:        RJMP    ERROR_M                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

MAS1:           LDI     R17,'+'                 ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17                 ;TRANSMITE LA A

TXMAS:          SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITA LA +
                RJMP    TXMAS                   ; MIENTRAS NO SE TRANSMITA VERIFICA

                SBI     USR,TXC                 ;ESPERA A QUE SE TRANSMITA EL DATO

ESPERAMAS:      SBIS    USR,RXC                 ; RECIBE CARACTERER
                RJMP    ESPERAMAS

                IN      R17,UDR
                CPI     R17,'+'
                BRNE    ERRORMASS               ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    CE1

ERRORMASS:      RJMP    ERROR_M                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

CE1:            LDI     R17,'C'                 ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17                 ;TRANSMITE LA C

TX_C1:          SBIS    USR,TXC                 ;ESPERA A QUE SE TRANSMITA LA +
                RJMP    TX_C1                   ; MIENTRAS NO SE TRANSMITA VERIFICA
```

```
                SBI     USR,TXC                 ;ESPERA A QUE SE TRANSMITA EL DATO

ESPERAC:        SBIS    USR,RXC                 ; RECIBE CARACTERER
                RJMP    ESPERAC                 ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                        CPI     R17,'C'
                        BRNE    ERRORCC
                        RJMP    EM1

ERROR_COMI: RJMP    ERROR_M
;*****************************************************************************************************
; MARCAR EL NÚMERO
;*****************************************************************************************************
NUMERO:         LDI     17,'5'          ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17         ;TRANSMITE LA 5
NUM_1:          SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA +
                RJMP    NUM_1           ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO

ESPERA5:        SBIS    USR,RXC         ; RECIBE CARACTERER
                CPI     R17,'5'         ;COMPARACIÓN DEL NÚMERO
                BRNE    ERROR5          ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP     NUM_2

ERROR5:         RJMP    ERROR_M         ; EXISTIÓ ERROR SALTA
NUM_2:          LDI     R17,'5'         ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17         ;TRANSMITE LA 5

TX_NUM2:        SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA 5
                RJMP    TX_NUM2         ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC

ESPERAN2:       SBIS    USR,RXC         ; RECIBE CARACTERER
                RJMP    ESPERAN2        ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'5'         ; COMPARACIÓN DEL NÚMERO
                BRNE    ERRORN2         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP     NUM_3
ERRORN2:        RJMP    ERROR_M         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

NUM_3:           LDI    R17,'2'         ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17         ;TRANSMITE LA 2
TX_3:           SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA EL 2
ESPERAN3:       SBIS    USR,RXC         ; RECIBE CARACTERER
                RJMP    ESPERAN3

                CPI     R17,'2'         ; COMPARACIÓN DEL NÚMERO
                BRNE    ERRORN3         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP   NUM_4

ERRORN3:        RJMP   ERROR_M          ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

NUM_4:          LDI     R17,'1'         ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17         ;TRANSMITE EL 1

TX_N4:          SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA 1
                RJMP    TX_N4           ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC

ESPERAN4:       SBIS    USR,RXC         ; RECIBE CARACTERER
                RJMP    ESPERAN4        ; MIENTRAS NO SE TRANSMITA VERIFICA
                IN      R17,UDR
                CPI     R17,'1'         ; COMPARACIÓN DEL NÚMERO
                BRNE    ERRORN4         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    NUM_5
ERRORN4:        RJMP    ERROR_M
NUM_5:          LDI     R17,'8'         ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17         ;TRANSMITE LA C
```

```
TXNUM5:     SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA +
            RJMP    TXNUM5          ; MIENTRAS NO SE TRANSMITA VERIFICA
            SBI     USR,TXC

ESPERANUM5: SBIS    USR,RXC         ; RECIBE CARACTERER
            RJMP    ESPERANUM5      ; MIENTRAS NO SE TRANSMITA VERIFICA
            IN      R17,UDR
            CPI     R17,'8'         ; COMPARACIÓN DEL NÚMERO
            BRNE    ERRORN6         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
            RJMP    NUM_6
ERRORN6:    RJMP   ERROR_M          ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

NUM_6:      LDI     R17,'0'         ; CARGA VALOR AL REGISTRO R17
            OUT     UDR,R17         ;TRANSMITE LA C

TX_NUM6:    SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA +
            RJMP    TX_NUM6         ; MIENTRAS NO SE TRANSMITA VERIFICA
            SBI     USR,TXC

ESPERANUM6: SBIS    USR,RXC         ; RECIBE CARACTERER
            RJMP    ESPERANUM6      ; MIENTRAS NO SE TRANSMITA VERIFICA
            IN      R17,UDR
            CPI     R17,'0'         ;COMPARACIÓN DEL NÚMERO
            BRNE    ERRORNUM6
            RJMP    NUM_7
ERRORNUM6:  RJMP    ERROR_M         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

NUM_7:      LDI     R17,'1'         ; CARGA VALOR AL REGISTRO R17
            OUT     UDR,R17         ;TRANSMITE LA C

TX_NUM7:    SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO
            RJMP    TX_NUM7         ; MIENTRAS NO SE TRANSMITA VERIFICA
            SBI     USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO

ESPENUM7:   SBIS    USR,RXC         ; RECIBE CARACTERER
            RJMP    ESPENUM7        ; MIENTRAS NO SE TRANSMITA VERIFICA
            IN      R17,UDR
            CPI     R17,'1'         ; COMPARACIÓN DEL NÚMERO
            BRNE    ERRORNUM7       ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
            RJMP    NUM_8

ERRORNUM7:  RJMP    ERROR_M         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
NUM_8:      LDI     R17,'2'         ; CARGA VALOR AL REGISTRO R17
            OUT     UDR,R17         ;TRANSMITE LA C

TX_NUM8:    SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO
            RJMP    TX_NUM8         ; MIENTRAS NO SE TRANSMITA VERIFICA
            SBI     USR,TXC         ; ESPERA A QUE SE TRANSMITA EL DATO

ESPERAN8:   SBIS    USR,RXC         ; RECIBE CARACTERER
            RJMP    ESPERAN8        ; MIENTRAS NO SE TRANSMITA VERIFICA
            IN      R17,UDR
            CPI     R17,'2'         ; COMPARACIÓN DEL NÚMERO
            BRNE    ERRORN8         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
            RJMP   NUM_9
ERRORN8:    RJMP   ERROR_M          ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

NUM_9:      LDI     R17,'3'         ; CARGA VALOR AL REGISTRO R17
            OUT     UDR,R17         ;TRANSMITE EL NÚMERO

TX_9:       SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA LA +
            RJMP    TX_9            ; MIENTRAS NO SE TRANSMITA VERIFICA
            SBI     USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO

ESPERAN9:   SBIS    USR,RXC         ; RECIBE CARACTERER
            RJMP    ESPERAN9        ; MIENTRAS NO SE TRANSMITA VERIFICA
            IN      R17,UDR
            CPI     R17,'3'         ; COMPARACIÓN DEL NÚMERO
            BRNE    ERRORN9         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORE
```

```
ERRORN9:        RJMP   ERROR_M          ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

NUM_10:         LDI     R17,'2'         ; CARGA VALOR AL REGISTRO R17
                OUT     UDR,R17         ;TRANSMITE LA EL 2

TX_N10:         SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA EL NÚMERO
                RJMP    TX_N10          ; MIENTRAS NO SE TRANSMITA VERIFICA
                SBI     USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO

ESPERAN10:      SBIS    USR,RXC         ; RECIBE CARACTERER
                RJMP    ESPERAN10       ; MIENTRAS NO SE TRANSMITA VERIFICA

                IN      R17,UDR
                CPI     R17,'2'         ; COMPARACIÓN DEL NÚMERO
                BRNE    ERRORN10        ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
                RJMP    COMILLAS1
ERRORN10:       RJMP    ERROR_M         ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES

;*****************************************************************************************************************
BIEN:     LDI     R17,$80               ; CARGA VALOR AL REGISTRO R17
          OUT     PORTB,R17
FIN:       RJMP   FIN                   ;  FIN DE LA TRANSMISIÓN
ERROR_MAS:      RJMP    ERROR_M         ;  SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
ERROR_IGU:      RJMP    ERROR_M
ERROR_F:        RJMP    ERROR_M
ERROR_G:        RJMP    ERROR_M
ERROR_MM:       RJMP    ERROR_M
ERROR_C:        RJMP    ERROR_M
ERROR_UNO:      RJMP    ERROR_M
;*****************************************************************************************************************
                                        ; RUTINA DE TIEMPO
DELAY:  LDI     R18,$FF                 ; CARGA VALOR AL REGISTRO R18
DEL2:   LDI     R19,$FF                 ; CARGA VALOR AL REGISTRO R19
DEL1:   DEC     R19
        BRNE    DEL1
        DEC     R18
        BRNE    DEL2

; ******************SUBRUTINA PARA COMPARAR EL MISMO CARACTER*********************************************

MISMO:  SBIS    USR,RXC
        RJMP    ESPERA_A
        IN      R0,UDR
        CP      R17,R0
        BRNE    ERROR_M                 ; SI EXISTE UN ERROR BRINCA AL MANEJO DE ERRORES
        RET
;*****************************************************************************************************************
ERROR_M:        LDI     R21,$FF                 ;ERROR AL RECIBIR EL MISMO
                OUT     PORTB,R17
                RCALL   DELAY
                RCALL   DELAY
                RJMP    AGAIN
;*****************************************************************************************************************
DESPL_SL:       LPM                     ;CARGA MEMORIA DE PROGRAMA
                MOV     R24,R0          ;PASA EL DATO A R24 PARA TRABAJAR
                OUT     UDR, R24        ;MANDA A LA UART
TX_EX:          SBIS    USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO
                RJMP    TX_EX
                SBI     USR,TXC         ;ESPERA A QUE SE TRANSMITA EL DATO
                ADIW    ZL,$01          ;SIGUIENTE DATO
                DEC     R18
                BRNE    DESPL_SL        ; ¿TERMINASTE DE DESPLEGAR?
                RET
;*****************************************************************************************************************

;*******************************TABLA DE DATOS A TRANSMITIR ***************************************************

MENSAJE_1A: .DB "ESTOS SON "
MENSAJE_1B: .DB "LOS DATOS "
MENSAJE_1C: .DB "DEL      "
```

```
MENSAJE_1D: .DB "GSM       "
MENSAJE_2A: .DB "AAAAAAAA  "
MENSAJE_6 : .DB "12345678  "
MENSAJE_7A: .DB "BBBBBBBB  "
MENSAJE_7B: .DB "12345678  "
MENSAJE_9 : .DB "CCCCCCCC  "
MENSAJE_10: .DB "ERROR "
MENSAJE_11A:.DB "12345678  "
MENSAJE_11B:.DB "DDDDDDDD  "
MENSAJE_13 :.DB "11111111  "
MENSAJE_14 :.DB "22222222  "
MENSAJE_16A:.DB "33333333  "
MENSAJE_16B:.DB "44444444  "
MENSAJE_17 :.DB "55555555  "
MENSAJE_22A:.DB "66666666  "
MENSAJE_22B:.DB "77777777  "
MENSAJE_24: .DB "88888888  "
MENSAJE_25A:.DB "UNAM  "
MENSAJE_25B:.DB "INSTITUTO   "
MENSAJE_26A:.DB "DE  "
MENSAJE_26B:.DB "INGENIERIA  "
MENSAJE_32 :.DB "ARCELIA   "
MENSAJE_33A:.DB "BERNAL  "
MENSAJE_33B:.DB "DIAZ  "
MENSAJE_35:         .DB "VERSION1.1  "
```

Nota: para la programación de serie, modem telefónico y radio modem, se emplea la misma programación del GSM, solo es necesario el cambio de comandos Hayes utilizados.

APÉNDICE B
PROGRAMACIÓN SOFTWARE

PROGRAMACIÓN DEL SOFTWARE DEL PROTOTIPO DE COMUNICACIONES DE DATOS

```
' ********************************************************************************************************************
Programa que unifica a los programas de los diferentes módulos de comunicación, con archivos ejecutables
Programadora Arcelia Bernal Díaz
Versión 1.1.
Última modificación 18 de Marzo del 2006
Visual Basic
' ********************************************************************************************************************

Option Explicit

Private id_rs232 As Double

Private Sub Command1_Click()
On Error GoTo TrErr
AppActivate id_rs232
Me.SetFocus
MsgBox "La aplicacion correspondiente ya ha sido inicializada", vbInformation, ""
Exit Sub
TrErr:
If Err.Number = 5 Then
   id_rs232 = Shell("C:\Documents and Settings\ARCELIA\Mis
documentos\tesis\MAESTRIA\aplicacion\IRDA\Debug\PRO.exe", vbNormalFocus)
Else
   MsgBox Err.Number & vbCrLf & Err.Description
End If
End Sub

Private Sub Command2_Click()
On Error GoTo TrErr
AppActivate id_rs232
Me.SetFocus
MsgBox "La aplicacion correspondiente ya ha sido inicializada", vbInformation, ""
Exit Sub
TrErr:
If Err.Number = 5 Then
   id_rs232 = Shell("C:\Documents and Settings\ARCELIA\Mis documentos\tesis\MAESTRIA\aplicacion\modem1\modem.exe",
vbNormalFocus)    ' llamada al archivo ejecutable modem.exe
Else
   MsgBox Err.Number & vbCrLf & Err.Description
End If
End Sub

Private Sub Command3_Click()
On Error GoTo TrErr
AppActivate id_rs232
Me.SetFocus
MsgBox "La aplicacion correspondiente ya ha sido inicializada", vbInformation, ""
Exit Sub
TrErr:
If Err.Number = 5 Then
   id_rs232 = Shell("C:\Documents and Settings\ARCELIA\Mis
documentos\tesis\MAESTRIA\aplicacion\IRDA\Debug\PRO.exe", vbNormalFocus)
Else
   MsgBox Err.Number & vbCrLf & Err.Description
End If
End Sub

Private Sub Command4_Click()
On Error GoTo TrErr
AppActivate id_rs232
Me.SetFocus
MsgBox "La aplicacion correspondiente ya ha sido inicializada", vbInformation, ""
Exit Sub
TrErr:
```

```
If Err.Number = 5 Then
   id_rs232 = Shell("C:\Documents and Settings\ARCELIA\Mis documentos\tesis\MAESTRIA\aplicacion\IRDA\Debug\PRO.exe", vbNormalFocus)
Else
   MsgBox Err.Number & vbCrLf & Err.Description
End If
End Sub

Private Sub Command5_Click()
Shell "C:\Documents and Settings\ARCELIA\Mis documentos\tesis\MAESTRIA\aplicacion\ComGsm\GSM.exe", vbNormalFocus

End Sub

Private Sub Command6_Click()
On Error GoTo TrErr
AppActivate id_rs232
Me.SetFocus
MsgBox "La aplicacion correspondiente ya ha sido inicializada", vbInformation, ""
Exit Sub
TrErr:
If Err.Number = 5 Then
   id_rs232 = Shell("C:\Documents and Settings\ARCELIA\Mis documentos\tesis\MAESTRIA\aplicacion\IRDA\Debug\PRO.exe", vbNormalFocus)
Else
   MsgBox Err.Number & vbCrLf & Err.Description
End If
End Sub


Private Sub Form_Load()
id_rs232 = 0
End Sub


' ************************************************************************************************************************
' Programa de la estación base para la comunicación serie e IrDA
' Versión 1.1
' Programadores: Arcelia Bernal Díaz y Código de ejemplos de Microsoft
' Última modificación 18 de Marzo del 2006
' Visual C++
' ************************************************************************************************************************

#include <stdio.h>
#include <windows.h>
#include "mttty.h"

BOOL InitializeApp( HINSTANCE, int);
int WINAPI MTTTYWndProc( HWND, UINT, WPARAM, LPARAM );
int WINAPI TTYChildProc( HWND, UINT, WPARAM, LPARAM );
void CmdDispatch( int, HWND, LPARAM );
void OpenTTYChildWindow( HWND );
BOOL ScrollTTYVert( HWND, WORD, WORD );
BOOL ScrollTTYHorz( HWND, WORD, WORD );
BOOL VersionCheck();
BOOL PaintTTY( HWND );
BOOL CALLBACK SplashDlgProc( HWND, UINT, WPARAM, LPARAM );

char informacion[4000];
int contC = 0;

/************************************************************************************************************************
Manejo de la aplicación y los mensajes de  Windows con la función
FUNCTION: WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
************************************************************************************************************************/
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
   MSG msg;
```

```
   if (!VersionCheck()) {
      MessageBox(NULL, "MTTTY can't run on this version of Windows.", NULL, MB_OK);
      return 0;
   }
            //--------------------------------------------------------------------
            //VENTANA DE BIENVENIDA INACTIVA
            //--------------------------------------------------------------------
            //          DialogBox          ( hInstance,
            //                                                MAKEINTRESOURCE(IDD_SPLASH),
            //                                                NULL,
            //                                                SplashDlgProc );
            //--------------------------------------------------------------------
            //--------------------------------------------------------------------


   if (!InitializeApp(hInstance, nShowCmd)) {
      MessageBox(NULL, "MTTTY couldn't start!", NULL, MB_OK);
      return 0;
   }

   while (GetMessage(&msg, NULL, 0, 0)) {
      if (!TranslateAccelerator( ghwndMain, ghAccel, &msg )) {
         TranslateMessage( &msg ) ;
         DispatchMessage( &msg ) ;
      }
   }

   return 1;
}

BOOL VersionCheck()
{
   gOSV.dwOSVersionInfoSize = sizeof(gOSV);
   if (!GetVersionEx(&gOSV))
      return FALSE;

   if (gOSV.dwPlatformId == VER_PLATFORM_WIN32s)
      return FALSE;

   return TRUE ;
}

BOOL InitializeApp(HINSTANCE hInst, int nShowCmd)
{
   WNDCLASS wc = {0};

   GlobalInitialize();     // get all global variables initialized to defaults

   wc.lpfnWndProc      = (WNDPROC) MTTTYWndProc;
   wc.hInstance        = hInst;
   wc.hIcon            = LoadIcon(hInst,MAKEINTRESOURCE(IDI_APPICON));
   wc.hCursor          = LoadCursor(NULL, IDC_ARROW);
   wc.lpszMenuName     = MAKEINTRESOURCE(IDR_MTTTYMENU);
   wc.hbrBackground    = (HBRUSH) (COLOR_WINDOW + 1) ;
   wc.lpszClassName    = "MTTTYClass";

   if (!RegisterClass(&wc)) {
      GlobalCleanup();
      return FALSE;
   }
```

```
/*****************************************************************************************************
Configuración del programa multihilos de la clase de Windows
*****************************************************************************************************/

   wc.lpfnWndProc     = (WNDPROC) TTYChildProc;
   wc.hInstance       = hInst;
   wc.hCursor         = LoadCursor(NULL, IDC_IBEAM);
   wc.hbrBackground   = (HBRUSH) (COLOR_WINDOW + 1);
   wc.lpszClassName   = "MTTTYChildClass";
   wc.lpszMenuName    = NULL;
   wc.hIcon           = NULL;

   if (!RegisterClass(&wc)) {
      GlobalCleanup();
      return FALSE;
   }

/*****************************************************************************************************
Configuración de la ventana principal de Windows
*****************************************************************************************************/
   ghwndMain = CreateWindow("MTTTYClass", "Sistema de Comunicaciones de Datos",
         WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
         STARTXWINDOW, STARTYWINDOW,
         MAXXWINDOW, MAXYWINDOW,
         NULL, NULL, hInst, NULL);

   if (ghwndMain == NULL) {
      GlobalCleanup();
      return FALSE;
   }

   ShowWindow( ghwndMain, nShowCmd ) ;
   UpdateWindow( ghwndMain ) ;

   ghInst = hInst;
   ghAccel = LoadAccelerators( hInst, MAKEINTRESOURCE( IDR_MTTTYACCELERATOR) ) ;

   return TRUE;
}

int WINAPI MTTTYWndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   switch (message)
   {
      case WM_CREATE:

         InitTTYInfo();
         OpenTTYChildWindow(hwnd);
         OpenSettingsToolbar(hwnd);
         OpenStatusToolbar(hwnd);
         ChangeConnection(hwnd, CONNECTED(TTYInfo));
         break;

      case WM_DESTROY:
         DestroyTTYInfo();
         DestroyWindow(ghWndToolbarDlg);
         DestroyWindow(ghWndStatusDlg);
         DestroyWindow(ghWndTTY);

         GlobalCleanup();
         PostQuitMessage(0);
         break;
```

```
      case WM_GETMINMAXINFO:
         {

            LPMINMAXINFO lpTemp;
            POINT        ptTemp;

            lpTemp = (LPMINMAXINFO) lParam;

            ptTemp.x = (long) lpTemp->ptMinTrackSize.x;
            ptTemp.y = (long) gcyMinimumWindowHeight;

            lpTemp->ptMinTrackSize = ptTemp;
         }

         break;

      case WM_SIZE:
         {
            WORD wTop;
            WORD wHeight;
            WORD wWidth = LOWORD(lParam);

            wHeight = SETTINGSFACTOR*gwBaseY;
            wTop = 0;
            MoveWindow(ghWndToolbarDlg, 0,wTop, wWidth, wHeight, TRUE);

            wHeight = STATUSFACTOR*gwBaseY;
            wTop = HIWORD(lParam) - wHeight;
            MoveWindow(ghWndStatusDlg, 0, wTop, wWidth, wHeight, TRUE);

           wHeight = HIWORD(lParam) - ((STATUSFACTOR + SETTINGSFACTOR)*gwBaseY);
            wTop = SETTINGSFACTOR*gwBaseY;
            MoveWindow(ghWndTTY, 0, wTop, wWidth, wHeight, TRUE);
         }

         break;

      case WM_COMMAND:
         CmdDispatch(LOWORD(wParam), hwnd, lParam);
         break;

      case WM_CHAR:
         SetFocus(ghWndTTY);
         SendMessage(ghWndTTY, WM_CHAR, wParam, lParam);
         break;

      case WM_CLOSE:
         if (DisconnectOK()) {
            if (CONNECTED(TTYInfo)) {
               if (TRANSFERRING(TTYInfo))
                  TransferFileTextEnd();
               BreakDownCommPort();
            }
            DestroyWindow(hwnd);
         }
         break;

      default:
         return DefWindowProc(hwnd, message, wParam, lParam);
   }

   return 0L;
}
```

```
/*****************************************************************************************************************
Confuguración del menú principal
*****************************************************************************************************************/
void CmdDispatch(int iMenuChoice, HWND hwnd, LPARAM lParam)
{
            FILE *pFile;
   static char szFileName[MAX_PATH] = {0};

   switch (iMenuChoice)
   {
      case ID_HELP_ABOUTMTTTY:
         CmdAbout(hwnd);
         break;

      case ID_TRANSFER_SENDFILETEXT:
         {
            char * szFilter = "Text Files\0*.TXT\0";
            OPENFILENAME ofn = {0};

            ofn.lStructSize = sizeof(OPENFILENAME);
            ofn.hwndOwner = hwnd;
            ofn.lpstrFilter = szFilter;
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = MAX_PATH;
            ofn.lpstrTitle = "Send File";
            ofn.Flags = OFN_FILEMUSTEXIST;

            if (!GetOpenFileName(&ofn))
               break;

            if (TRUE)
               TransferFileTextStart(szFileName);
         }
         break;

      case ID_TRANSFER_RECEIVEFILETEXT:
         {
            char * szFilter = "Text Files\0*.TXT\0";
            OPENFILENAME ofn = {0};

            ofn.lStructSize = sizeof(OPENFILENAME);
            ofn.hwndOwner = hwnd;
            ofn.lpstrFilter = szFilter;
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = MAX_PATH;
            ofn.lpstrTitle = "Receive File";
            ofn.Flags = OFN_OVERWRITEPROMPT;

            if (!GetSaveFileName(&ofn))
               break;
            ReceiveFileText(szFileName);
         }
         break;
           TransferFileTextEnd();
         break;
      case ID_TRANSFER_SENDREPEATEDLY:
         {
            DWORD dwFreq;
            char * szFilter = "Text Files\0*.TXT\0";
            OPENFILENAME ofn = {0};

            ofn.lStructSize = sizeof(OPENFILENAME);
```

```
            ofn.hwndOwner = hwnd;
            ofn.lpstrFilter = szFilter;
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = MAX_PATH;
            ofn.lpstrTitle = "Send File Repeatedly";
            ofn.Flags = OFN_FILEMUSTEXIST;

            if (!GetOpenFileName(&ofn))
               break;

            dwFreq = GetAFrequency();

            TransferRepeatCreate(szFileName, dwFreq);
         }
         break;

      case ID_TRANSFER_ABORTREPEATEDSENDING:
         TransferRepeatDestroy();
         break;

      case ID_TTY_CLEAR:
         ClearTTYContents();
         InvalidateRect(ghWndTTY, NULL, TRUE);
         break;
      case IDC_FONTBTN:
      case IDC_COMMEVENTSBTN:
      case IDC_FLOWCONTROLBTN:
      case IDC_TIMEOUTSBTN:
         SendMessage(ghWndToolbarDlg, WM_COMMAND, (WPARAM) iMenuChoice, (LPARAM)
GetDlgItem(ghWndToolbarDlg, iMenuChoice));
         break;

      case ID_FILE_CONNECT:
         if (SetupCommPort() != NULL)
            ChangeConnection(hwnd, CONNECTED(TTYInfo));
         break;
      case ID_FILE_DISCONNECT:
         if (BreakDownCommPort())
            ChangeConnection(hwnd, CONNECTED(TTYInfo));
         break;

      case ID_FILE_EXIT:
         PostMessage(hwnd, WM_CLOSE, 0, 0);
         break;
/*****************************************************************************************************************
Programación para el almacenamiento de información
*****************************************************************************************************************/
      case ID_ARCHIVO_GUARDAR:


          pFile = fopen ("mmmm.txt","wt");
          if (pFile!=NULL)
                                {
             fprintf(pFile, "La información del buffer es: %s\n",TTYInfo.Screen);
                                        fprintf(pFile, "<EOF>");
             fclose (pFile);
                                }

                                break;
      case ID_ARCHIVO_GUARDARCOMO:
         {
            char * szFilter = "Text Files\0*.TXT\0";
                                        char * szDefExt = "txt";
```

```
            OPENFILENAME ofn = {0};

            ofn.lStructSize = sizeof(OPENFILENAME);
            ofn.hwndOwner = hwnd;
            ofn.lpstrFilter = szFilter;
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = MAX_PATH;
                                        ofn.lpstrDefExt = szDefExt;
            ofn.lpstrTitle = "Send File";
            ofn.Flags = OFN_EXPLORER | OFN_PATHMUSTEXIST;
            if (GetSaveFileName(&ofn))
                                        {
                                                    informacion[contC]= '\0';
                                                    pFile = fopen (ofn.lpstrFile,"wt");
                                                    if (pFile!=NULL)
                                                    {
             fprintf(pFile, "<La información del buffer es: >\n%s\n",informacion);
                                                     fprintf(pFile, "<EOF>");
             fclose (pFile);
                                                    }
                                        }
          }
                                    break;

    }
    return;
}
/*****************************************************************************************************************
Creación de ventanas hijas
*****************************************************************************************************************/
FUNCTION: OpenTTYChildWindow(HWND)

PURPOSE: Creates the TTY Child Window
        crea el tty ventana hija

PARAMETERS:
    hWnd - parent window handle of TTY child window
                indicador del padre de la ventana de la tty ventana hija

COMMENTS: This window is actually the TTY Screen
         esta ventana es actualizada en pantalla TTY

HISTORY:   Date:      Author:     Comment:
          10/27/95   AllenD      Wrote it

BOOL NEAR ScrollTTYVert( HWND hWnd, WORD wScrollCmd, WORD wScrollPos )
{
  int  nScrollAmt ;

  switch (wScrollCmd)
  {
    case SB_TOP:
      nScrollAmt = -YOFFSET( TTYInfo ) ;
      break ;

    case SB_BOTTOM:
      nScrollAmt = YSCROLL( TTYInfo ) - YOFFSET( TTYInfo ) ;
      break ;

    case SB_PAGEUP:
      nScrollAmt = -YSIZE( TTYInfo ) ;
      break ;
```

```
    case SB_PAGEDOWN:
     nScrollAmt = YSIZE( TTYInfo ) ;
      break ;

    case SB_LINEUP:
      nScrollAmt = -YCHAR( TTYInfo ) ;
      break ;

    case SB_LINEDOWN:
      nScrollAmt = YCHAR( TTYInfo ) ;
      break ;

    case SB_THUMBPOSITION:
      nScrollAmt = wScrollPos - YOFFSET( TTYInfo ) ;
      break ;

    default:
      return ( FALSE ) ;
  }

  if ((YOFFSET( TTYInfo ) + nScrollAmt) > YSCROLL( TTYInfo ))
    nScrollAmt = YSCROLL( TTYInfo ) - YOFFSET( TTYInfo ) ;

  if ((YOFFSET( TTYInfo ) + nScrollAmt) < 0)
    nScrollAmt = -YOFFSET( TTYInfo ) ;

  ScrollWindowEx( hWnd, 0, -nScrollAmt, NULL, NULL, NULL, NULL, SW_INVALIDATE | SW_ERASE) ;

  YOFFSET( TTYInfo ) = YOFFSET( TTYInfo ) + nScrollAmt ;

  SetScrollPos( hWnd, SB_VERT, YOFFSET( TTYInfo ), TRUE ) ;

  return ( TRUE ) ;

} // end of ScrollTTYVert()

BOOL NEAR ScrollTTYHorz( HWND hWnd, WORD wScrollCmd, WORD wScrollPos )
{
  int  nScrollAmt ;

  switch (wScrollCmd)
  {
    case SB_TOP:
      nScrollAmt = -XOFFSET( TTYInfo ) ;
      break ;

    case SB_BOTTOM:
      nScrollAmt = XSCROLL( TTYInfo ) - XOFFSET( TTYInfo ) ;
      break ;

    case SB_PAGEUP:
      nScrollAmt = -XSIZE( TTYInfo ) ;
      break ;

    case SB_PAGEDOWN:
      nScrollAmt = XSIZE( TTYInfo ) ;
      break ;

    case SB_LINEUP:
      nScrollAmt = -XCHAR( TTYInfo ) ;
      break ;

    case SB_LINEDOWN:
```

```
        nScrollAmt = XCHAR( TTYInfo ) ;
        break ;

      case SB_THUMBPOSITION:
        nScrollAmt = wScrollPos - XOFFSET( TTYInfo ) ;
        break ;

      default:
        return ( FALSE ) ;
    }
    if ((XOFFSET( TTYInfo ) + nScrollAmt) > XSCROLL( TTYInfo ))
      nScrollAmt = XSCROLL( TTYInfo ) - XOFFSET( TTYInfo ) ;
    if ((XOFFSET( TTYInfo ) + nScrollAmt) < 0)
      nScrollAmt = -XOFFSET( TTYInfo ) ;
    ScrollWindowEx( hWnd, -nScrollAmt, 0, NULL, NULL, NULL, NULL, SW_INVALIDATE | SW_ERASE) ;
    XOFFSET( TTYInfo ) = XOFFSET( TTYInfo ) + nScrollAmt ;
    SetScrollPos( hWnd, SB_HORZ, XOFFSET( TTYInfo ), TRUE ) ;

    return ( TRUE ) ;

} // end of ScrollTTYHorz()

BOOL NEAR PaintTTY( HWND hWnd )
{
    PAINTSTRUCT  ps ;
    HFONT        hOldFont ;
    RECT         rect ;
    HDC          hDC ;
    int          nRow, nCol, nEndRow, nEndCol;
    int          nCount, nHorzPos, nVertPos ;

    hDC = BeginPaint( hWnd, &ps ) ;
    hOldFont = SelectObject( hDC, HTTYFONT( TTYInfo ) ) ;
    SetTextColor( hDC, FGCOLOR( TTYInfo ) ) ;
    SetBkColor( hDC, GetSysColor( COLOR_WINDOW ) ) ;
    rect = ps.rcPaint ;
    nRow =
      min( MAXROWS - 1,
          max( 0, (rect.top + YOFFSET( TTYInfo )) / YCHAR( TTYInfo ) ) ) ;
    nEndRow =
      min( MAXROWS - 1,
          ((rect.bottom + YOFFSET( TTYInfo ) - 1) / YCHAR( TTYInfo ) ) ) ;
    nCol =
      min( MAXCOLS - 1,
          max( 0, (rect.left + XOFFSET( TTYInfo )) / XCHAR( TTYInfo ) ) ) ;
    nEndCol =
      min( MAXCOLS - 1,
          ((rect.right + XOFFSET( TTYInfo ) - 1) / XCHAR( TTYInfo ) ) ) ;
    nCount = nEndCol - nCol + 1 ;
    for (; nRow <= nEndRow; nRow++)
    {
      nVertPos = (nRow * YCHAR( TTYInfo )) - YOFFSET( TTYInfo ) ;
      nHorzPos = (nCol * XCHAR( TTYInfo )) - XOFFSET( TTYInfo ) ;
      rect.top = nVertPos ;
      rect.bottom = nVertPos + YCHAR( TTYInfo ) ;
      rect.left = nHorzPos ;
      rect.right = nHorzPos + XCHAR( TTYInfo ) * nCount ;
      SetBkMode( hDC, OPAQUE ) ;
      ExtTextOut( hDC, nHorzPos, nVertPos, ETO_OPAQUE | ETO_CLIPPED, &rect,
                (LPSTR)( SCREEN( TTYInfo ) + nRow * MAXCOLS + nCol ),
                nCount, NULL ) ;
    }
    SelectObject( hDC, hOldFont ) ;
```

```
  EndPaint( hWnd, &ps ) ;
  MoveTTYCursor( hWnd ) ;
  return ( TRUE ) ;

}
BOOL NEAR MoveTTYCursor( HWND hWnd )
{
  if (CONNECTED( TTYInfo ) && (CURSORSTATE( TTYInfo ) & CS_SHOW))
    SetCaretPos( (COLUMN( TTYInfo ) * XCHAR( TTYInfo )) -
              XOFFSET( TTYInfo ),
              (ROW( TTYInfo ) * YCHAR( TTYInfo )) -
              YOFFSET( TTYInfo ) ) ;

  return ( TRUE ) ;

}

BOOL NEAR SetTTYFocus( HWND hWnd )
{
  if (CONNECTED(TTYInfo) && (CURSORSTATE( TTYInfo ) != CS_SHOW) )
  {
    CreateCaret( hWnd, NULL, XCHAR( TTYInfo ), YCHAR( TTYInfo ) ) ;
    ShowCaret( hWnd ) ;
    CURSORSTATE( TTYInfo ) = CS_SHOW ;
  }

  MoveTTYCursor( hWnd ) ;
  return ( TRUE ) ;

}

BOOL NEAR KillTTYFocus( HWND hWnd )
{
  if (CURSORSTATE( TTYInfo ) != CS_HIDE)
  {
    HideCaret( hWnd ) ;
    DestroyCaret() ;
    CURSORSTATE( TTYInfo ) = CS_HIDE ;
  }
  return ( TRUE ) ;

}

BOOL NEAR SizeTTY( HWND hWnd, WORD wWidth, WORD wHeight )
{
  int nScrollAmt ;

  YSIZE( TTYInfo ) = (int) wHeight ;
  YSCROLL( TTYInfo ) = max( 0, (MAXROWS * YCHAR( TTYInfo )) -
                    YSIZE( TTYInfo ) ) ;
  nScrollAmt = min( YSCROLL( TTYInfo ), YOFFSET( TTYInfo ) ) -
               YOFFSET( TTYInfo ) ;
  ScrollWindow( hWnd, 0, -nScrollAmt, NULL, NULL ) ;

  YOFFSET( TTYInfo ) = YOFFSET( TTYInfo ) + nScrollAmt ;
  SetScrollPos( hWnd, SB_VERT, YOFFSET( TTYInfo ), FALSE ) ;
  SetScrollRange( hWnd, SB_VERT, 0, YSCROLL( TTYInfo ), TRUE ) ;

  XSIZE( TTYInfo ) = (int) wHeight ;
  XSCROLL( TTYInfo ) = max( 0, (MAXCOLS * XCHAR( TTYInfo )) -
                    XSIZE( TTYInfo ) ) ;
  nScrollAmt = min( XSCROLL( TTYInfo ), XOFFSET( TTYInfo )) -
               XOFFSET( TTYInfo ) ;
```

```
  ScrollWindow( hWnd, nScrollAmt, 0, NULL, NULL );
  XOFFSET( TTYInfo ) = XOFFSET( TTYInfo ) + nScrollAmt ;
  SetScrollRange( hWnd, SB_HORZ, 0, XSCROLL( TTYInfo ), FALSE ) ;
  SetScrollPos( hWnd, SB_HORZ, XOFFSET( TTYInfo ), TRUE ) ;

  InvalidateRect( hWnd, NULL, FALSE ) ;   // redraw entire window

  return ( TRUE ) ;

}

int WINAPI TTYChildProc(HWND hWnd, UINT uMessage, WPARAM wParam, LPARAM lParam)
{
   switch(uMessage)
   {
      case WM_VSCROLL:
         ScrollTTYVert( hWnd, LOWORD( wParam ), HIWORD( wParam ) ) ;
         break ;

      case WM_HSCROLL:
         ScrollTTYHorz( hWnd, LOWORD( wParam ), HIWORD( wParam ) ) ;
         break ;

      case WM_SIZE:
         SizeTTY(hWnd, LOWORD(lParam), HIWORD(lParam));
         break;

      case WM_PAINT:
         PaintTTY(hWnd);
         break;

      case WM_CHAR:
         {
            if (CONNECTED(TTYInfo)) {

               if (!WriterAddNewNode(WRITE_CHAR, 0, (char) wParam, NULL, NULL, NULL))
                  return FALSE;

               if (LOCALECHO(TTYInfo))
                  OutputABufferToWindow(ghWndTTY, (CHAR *) &wParam, 1);
            }
         }
         break;

      case WM_SETFOCUS:
         SetTTYFocus( ghWndTTY ) ;
         break ;

      case WM_KILLFOCUS:
         KillTTYFocus( ghWndTTY ) ;
         break ;
      case WM_MOUSEACTIVATE:
         SetFocus(hWnd);
         return MA_ACTIVATE;
         break;
      default:
         return DefWindowProc(hWnd, uMessage, wParam, lParam);
   }
   return 0L;
}
/*****************************************************************************************************************
Programación de los diálogos de las ventanas
*****************************************************************************************************************/
```

```
BOOL CALLBACK SplashDlgProc(HWND hdlg, UINT uMessage, WPARAM wparam, LPARAM lparam)
{
  static UINT uTimer;

  switch(uMessage)
  {
    case WM_INITDIALOG:
                        uTimer = SetTimer(hdlg, 1, 3000, NULL);
                        if (uTimer == 0)
                                EndDialog(hdlg, TRUE);
      break;

    case WM_TIMER:
                        KillTimer(hdlg, uTimer);
                        EndDialog(hdlg, TRUE);
      break;
  }

  return FALSE;
}
```

PROGRAMACIÓN SOFTWARE MODEM TELEFÓNICO

```
' ************************************************************************************************************
Programa del módulo de comunicaciones modem telefónico
Programadora Arcelia Bernal Díaz
Versión 1.1.
Última modificación 18 de Marzo del 2006
Visual Basic
' ************************************************************************************************************
Option Explicit

Private strBuff As String               'Variables
Private flgConn_Local As Boolean
Private flgRecibiendo As Boolean
Private flgEdo_Escucha As Boolean


Private Sub cmdConectar_Click()
Me.cmdConectar.Enabled = False

'************************************************************************************************************
Configuración de comandos Hayes para el modem telefónico
************************************************************************************************************/
msCom.Output = "AT" & Chr(13)
flgEdo_Escucha = False
If PtoRead("OK", 30) Then
   msCom.Output = "ATH1" & Chr(13)
  If PtoRead("OK", 30) Then
    msCom.Output = "ATD" & Me.txtNumCell.Text & Chr(13)
    If PtoRead("CONNECT ", 120) Then
          Else
      MsgBox "NO HA PODIDO ESTABLECER COMUNICACION.", vbCritical
      Me.cmdConectar.Enabled = True
      flgEdo_Escucha = False
    End If
  Else
    MsgBox "NO HA PODIDO ESTABLECER COMUNICACION.", vbCritical
    Me.cmdConectar.Enabled = True
    flgEdo_Escucha = False
  End If
```

```
Else
   MsgBox "NO HA PODIDO ESTABLECER COMUNICACION.", vbCritical
   Me.cmdConectar.Enabled = True
   flgEdo_Escucha = False
End If
End Sub

'*************************************************************************************************************
Desconexión del modem
*************************************************************************************************************/

Private Sub cmdDesconectar_Click()
Me.cmdDesconectar.Enabled = False
strBuff = ""
msCom.Output = "+++"
If PtoRead("OK", 30) Then
   strBuff = ""
   msCom.Output = "ATH0" & vbCrLf
   If PtoRead("OK", 30) Then
      Me.cmdDesconectar.Enabled = False
      Me.cmdConectar.Enabled = True
      Me.Caption = "DESCONECTADO"
      flgRecibiendo = False
      guardar_info
      flgConn_Local = False
   Else
      MsgBox "NO RESPONDE EL MODEM.", vbCritical
      Me.cmdDesconectar.Enabled = True
   End If
Else
   MsgBox "NO RESPONDE EL MODEM.", vbCritical
   Me.cmdDesconectar.Enabled = True
End If
flgEdo_Escucha = True
End Sub

Private Sub cmdSalir_Click()
Unload Me
End Sub

Private Sub cmdSend_Click()
Me.cmdSend.Enabled = False
msCom.Output = Trim(Me.txtMsg.Text)
Me.cmdSend.Enabled = True
End Sub

Private Sub Form_Load()
Me.msCom.PortOpen = True
flgRecibiendo = False
flgEdo_Escucha = True
flgConn_Local = False
End Sub

'*************************************************************************************************************
Programación del manejo de errores
*************************************************************************************************************/
Private Sub msCom_OnComm()
  Select Case msCom.CommEvent
  ' Controlar cada evento o error escribiendo
  ' código en cada instrucción Case

  ' Errores
    Case comBreak   ' Se ha recibido una interrupción.
```

```
    Case comEventFrame   ' Error de trama
    Case comEventOverrun   ' Datos perdidos.
    Case comEventRxOver   ' Desbordamiento del búfer
                ' de recepción.
    Case comEventRxParity   ' Error de paridad.
    Case comEventTxFull   ' Búfer de transmisión lleno.
    Case comEventDCB   ' Error inesperado al recuperar DCB.

  ' Eventos
    Case comEvCD   ' Cambio en la línea CD.
    Case comEvCTS   ' Cambio en la línea CTS.
    Case comEvDSR   ' Cambio en la línea DSR.
    Case comEvRing   ' Cambio en el indicador de  llamadas.
    Case comEvReceive   ' Recibido nº SThreshold de  caracteres.
                If flgRecibiendo Then
                   strBuff = strBuff & msCom.Input
                   Me.lblRcv.Caption = strBuff
                   DESCONECTADO
                Else
                   strBuff = strBuff & msCom.Input
                   Debug.Print "strBuff =" & strBuff & vbCrLf
                   If flgEdo_Escucha Then llamada
                End If
  End Select
End Sub
'*************************************************************************************************************
Programación para el almacenamiento de datos
*************************************************************************************************************'
Private Function PtoRead(ByRef strCad As String, ByVal espera As Integer) As Boolean
Dim tm As Single

tm = Timer
PtoRead = True
Do Until InStr(1, strBuff, strCad) > 0 Or InStr(1, strBuff, "ERROR") > 0
   If tm + espera < Timer Then
      PtoRead = False
      Exit Do
   End If
   strBuff = strBuff & Me.msCom.Input
   'DoEvents
Loop

End Function

Private Function llamada() As Boolean
If PtoRead("RING", 1) Then
   strBuff = ""
   msCom.Output = "ATA" & Chr(13)
   If PtoRead("CONNECT ", 60) Then
      Me.Caption = "CONECTADO A :" & """" & Me.txtNumCell.Text
      Me.cmdDesconectar.Enabled = True
      Me.txtMsg.Text = ""
      flgRecibiendo = True
      flgEdo_Escucha = False
      strBuff = ""
    Else
         MsgBox "NO HA PODIDO ESTABLECER COMUNICACION.", vbCritical
         Me.cmdConectar.Enabled = True
   End If
Else
   MsgBox "NO HA PODIDO ESTABLECER COMUNICACION.", vbCritical
   Me.cmdConectar.Enabled = True
End If
```

```
End Function

Private Sub DESCONECTADO()
If InStr(strBuff, "NO CARRIER" & vbCrLf) > 0 Then
   Else
      MsgBox "NO HA COLGADO CORRECTAMENTE.", vbCritical
   End If
End If
End Sub

Private Function guardar_info() As Boolean
Dim f As String
Dim numFile As Integer
numFile = FreeFile()
f = App.Path & "\info" & Format(Date, "ddmmyy") & Format(Time, "hhmmss") & ".txt"

Open f For Output As #numFile
Print #numFile, Me.lblRcv.Caption
Close #numFile

End Function
```

PROGRAMACIÓN SOFTWARE RADIO MODEM

```
' ************************************************************************************************************************
Programa del módulo de comunicaciones radio modem
Programadora Arcelia Bernal Díaz
Versión 1.1.
Última modificación 18 de Marzo del 2006
' ************************************************************************************************************************
Option Explicit

Private strBuff As String
Private Sub cmdGuardar_Click()
guardar_info
End Sub
Private Sub cmdSalir_Click()
Unload Me
End Sub

'************************************************************************************************************************
Programación para el marcado del número telefónico
************************************************************************************************************************/
Private Sub cmdSend_Click()
Me.cmdSend.Enabled = False
msCom.Output = "<" & Me.txtNumCell.Text & ">" & _
            Trim(Me.txtMsg.Text) & _
            "</" & Me.txtNumCell.Text & ">"

End Sub
Private Sub Form_Load()
Me.msCom.PortOpen = True
End Sub
'************************************************************************************************************************
Programación del manejo de errores
************************************************************************************************************************/
Private Sub msCom_OnComm()
  Select Case msCom.CommEvent
  ' Controlar cada evento o error escribiendo
  ' código en cada instrucción Case

  ' Errores
```

```
    Case comBreak   ' Se ha recibido una interrupción.
    Case comEventFrame   ' Error de trama
    Case comEventOverrun   ' Datos perdidos.
    Case comEventRxOver   ' Desbordamiento del búfer de recepción.
    Case comEventRxParity   ' Error de paridad.
    Case comEventTxFull   ' Búfer de transmisión lleno.
    Case comEventDCB   ' Error inesperado al recuperar DCB.
    Case comEvCD   ' Cambio en la línea CD.
    Case comEvCTS   ' Cambio en la línea CTS.
    Case comEvDSR   ' Cambio en la línea DSR.
    Case comEvRing   ' Cambio en el indicador de llamadas.

  End Select
End Sub
'*********************************************************************************************************
Programación para lectura del puerto
*********************************************************************************************************/
Private Function PtoRead(ByRef strCad As String, ByVal espera As Integer) As Boolean
Dim tm As Single
Do Until InStr(1, strBuff, strCad) > 0 Or InStr(1, strBuff, "ERROR") > 0
   If tm + espera < Timer Then
      PtoRead = False
      Exit Do
   End If
   strBuff = strBuff & Me.msCom.Input
Loop

End Function


'*********************************************************************************************************
Programación para el almacenamiento de información con su respectivo formato
*********************************************************************************************************/
Private Function guardar_info() As Boolean
Dim f As String
Dim numFile As Integer
numFile = FreeFile()
f = App.Path & "\info" & Format(Date, "ddmmyy") & Format(Time, "hhmmss") & ".txt"

Open f For Output As #numFile
Print #numFile, Me.lblRcv.Caption
End Function
```

PROGRAMACIÓN SOFTWARE GSM

```
' *********************************************************************************************************
Programa del módulo de comunicaciones radio modem
Programadora Arcelia Bernal Díaz
Versión 1.1.
Última modificación 18 de Marzo del 2006
' *********************************************************************************************************
Option Explicit

Private Const eCMD As Byte = 1' Conectado
Private Const eSMS As Byte = 2      ' Desconectado
Private Const eRSMS As Byte = 4     ' etc... todo
```

```
Private Const eUNE As Byte = 128   ' Deshabilitado


Private flgStatus As Byte
Private strBuff As String


Private Sub Form_Load()
flgStatus = eUNE
Me.MSComm1.PortOpen = True
Me.mnuHerramientas.Enabled = False
Me.Timer1.Enabled = True
Me.Label1.Caption = "Intentando comunicar con el dispositivo..."
P_Conect
End Sub

Private Sub Form_Unload(Cancel As Integer)
Me.MSComm1.PortOpen = False
End Sub
' **********************************************************************************************************
Programación para la lectura de datos
' **********************************************************************************************************
Private Sub mnuReadMsg_Click()
Load frmLeerMsg
flgStatus = eRSMS
frmLeerMsg.SetCom Me.MSComm1
frmLeerMsg.Show vbModal
flgStatus = eCMD
End Sub

Private Sub mnuSendMsg_Click()
Load frmSendMsg
frmSendMsg.SetCom Me.MSComm1
End Sub


'**********************************************************************************************************
Programación del manejo de errores
**********************************************************************************************************/
Private Sub MSComm1_OnComm()
  Select Case MSComm1.CommEvent

    Case comBreak   ' Se ha recibido una interrupción.
    Case comEventFrame   ' Error de trama
    Case comEventOverrun   ' Datos perdidos.
    Case comEventRxOver   ' Desbordamiento del búfer de recepción.
    Case comEventRxParity   ' Error de paridad.
    Case comEventTxFull   ' Búfer de transmisión lleno.
    Case comEventDCB   ' Error inesperado al recuperar DCB.
    Case comEvCD   ' Cambio en la línea CD.
    Case comEvCTS   ' Cambio en la línea CTS.
    Case comEvDSR   ' Cambio en la línea DSR.
    Case comEvRing   ' Cambio en el indicador de llamadas.
    Case comEvReceive   ' Recibido nº SThreshold de caracteres.
          If flgStatus = eUNE Then
             strBuff = strBuff & Me.MSComm1.Input
          End If

    End Sub

' **********************************************************************************************************
Verificación del puerto
' **********************************************************************************************************
```

```
Private Function P_Conect() As Boolean
Me.MSComm1.Output = "AT" & vbCrLf
End Function

Private Function PtoRead(ByRef strCad As String) As Boolean
Dim tm As Single
tm = Timer
PtoRead = True
Do Until InStr(1, strBuff, strCad) > 0
   If tm + 1 < Timer Or InStr(1, strBuff, "ERROR") > 0 Then
      PtoRead = False
      Exit Do
   End If
   DoEvents
Loop
End Function


Private Sub Timer1_Timer()
Static intentos As Integer
If intentos < 3 Then
   flgStatus = IIf(PtoRead("OK" & vbCrLf), eCMD, eUNE)
Else
   If flgStatus = eUNE Then
      Me.Label1.Caption = "Dispositivo no disponible." & vbCrLf & "Verifique su conexión al dispositivo móvil"
   Else
      Me.Label1.Caption = "Dispositivo Conectado..."
      Me.mnuHerramientas.Enabled = True
   End If
   Me.Timer1.Enabled = False
End If
intentos = intentos + 1
End Sub

' ************************************************************************************************************
Programación para la lectura de datos
' ************************************************************************************************************
Option Explicit

Private WithEvents msCom As MSComm
Private strBuff As String
Private Sub fgMensages_Click()
Clipboard.SetText Me.fgMensages.TextMatrix(Me.fgMensages.Row, Me.fgMensages.Col)
End Sub

Private Sub Form_Load()
Me.Label1.Caption = "Leyendo Mensajes..."
Me.fgMensages.ColWidth(0) = Me.fgMensages.Font.Size * 16 * 40
Me.fgMensages.ColAlignment(0) = flexAlignLeftCenter
Me.fgMensages.ColWidth(1) = Me.fgMensages.Font.Size * 16 * 60
Me.fgMensages.ColAlignment(1) = flexAlignLeftCenter
End Sub

Private Sub Form_Resize()
On Error Resume Next
Me.fgMensages.Top = Me.Label1.Top + Me.Label1.Height
Me.fgMensages.Height = Me.ScaleHeight - Me.fgMensages.Top - Me.cmdSalir.Height
Me.fgMensages.Width = Me.ScaleWidth
Me.cmdSalir.Left = (Me.ScaleWidth - Me.cmdSalir.Width) / 2
Me.cmdSalir.Top = Me.fgMensages.Top + Me.fgMensages.Height
End Sub
```

```
Private Sub cmdSalir_Click()
Unload Me
End Sub

Private Sub Timer1_Timer()
Me.Timer1.Enabled = False
Leer_Mensajes
End Sub

' ************************************************************************************************************
Programación de los comandos GSM
' ************************************************************************************************************
Public Sub SetCom(ByRef Com As MSComm)
Set msCom = Com
End Sub


Private Sub Leer_Mensajes()
Dim iInicio, iFinal As Integer
Dim strId As String, strMsg As String
Me.cmdSalir.Enabled = False
iInicio = 1
iFinal = 1
msCom.Output = "AT+CMGF=1"
If PtoRead(vbCrLf & "OK") Then
   strBuff = ""
   msCom.Output = "AT+CMGL" & vbCrLf
   If PtoRead(vbCrLf & "OK" & vbCrLf) Then
      Do While iFinal > 0 And iInicio > 0
         iInicio = InStr(iFinal, strBuff, "+CMGL:")
         If iInicio = 0 Then Exit Do
         iFinal = InStr(iInicio + 2, strBuff, vbCrLf & vbCrLf)
         strId = Mid(strBuff, iInicio, iFinal - iInicio)

         iInicio = InStr(iFinal, strBuff, vbCrLf & vbCrLf)
         If iInicio = 0 Then Exit Do
      Loop

   End If
End If
Me.cmdSalir.Enabled = True
End Sub
```

BIBLIOGRAFÍA

LIBROS

- Brey Barry, B. *Los Microprocesadores INTEL, Arquitectura, programación e Interfaces*, México, Prentice Hall, 1994.
- Ceballos, Francisco J. *Enciclopedia de Microsoft ® Visual Basic 6*, Alfaomega, 1997.
- Collazo, Javier. *Diccionario enciclopédico de términos técnicos*, McGraw-Hill, 1981.
- Fernández, f. *"Tecnología móvil, aplicaciones GSM, GPRS, UMTS y WIFI"*, Madrid, Anaya Multimedia, 2003.
- Fishbane, Gasiorowicz, Thornton, S., *Física para Ciencias e Ingeniería Volumen II*, México, Prentice Hall, 1994.
- Hall Douglas, V. *Microprocessors and interfacing programming and hardware,* E.E.U.U. Mc Graw Hill, 1992
- Huidobro, J*., Manual de Telefonía, Telefonía Fija y Móvil", Paraninfo*, España 1998.
- *Infrared data transmission*, ELEKTOR ELECTRONICS, April 1996.

- Kaaranen, Ahtiainen, et al *UMTS Networks –Arquitecture, Mobility and Services*, England, Wiley, 2001

- Kate G.,"*Edición Especial Visual C++® 6*, España, Prentice Hall, 1999.

- Mandado, Enrique. *Sistemas electrónico digitales* 7ª edición, México, ALfaomega, 1991.

- Miller, Brian. *My favorite family of micros*, CIRCUIT CELLAR # 34, August 2001.

- Nava, I., *"Diseño y Construcción de un Adquisidor de Datos y un Codificador Angular para Medición de Nivel de Agua"*, México, UNAM, 2004.

- Predko, Michael, *Programming and customizing PICmicro microcontrollers*, Second edition, USA, McGraw-Hill, 2002.

- Ramirez, Daniel. *Optimize your PIC*, CIRCUIT CELLAR # 133, August 2001, p. 46-51.

- Ramachandran, Hari. *IrDA Technology, Part 1: An overview*, CIRCUIT CELLAR #111 October 1999. p. 60-64.

- Ramachandran, Hari. *IrDA Technology, Part 2: Protocol Layers*, CIRCUIT CELLAR #112 November 1999. p. 60-65.

- Sendín, A., *Fundamentos de los sistemas de comunicaciones móviles*, España, McGraw-Hill, 2004.

- Shapiro J., *Visual Basic .NET*, España, Mc Graw Hill, 2003.

- Serway R., *Física Tomo II* , México, Mc Graw Hill, 1993.

- Stallings, W., *Comunicaciones y Redes de Computadores*, España, Pearson Prentice Hall, 2004.

- Tocci, Ronald J. *Microprocessors and microcomputers / Hardware and software*, Fifth edition, USA, Prentice Hall, 2000.

- Walraven, K. *Working with surface mount devices*. ELEKTOR ELECTRONICS, May 2000.

MANUALES

- AVR Risc Microcontroller, ATMEL Corporation, August 1999.

- Microcontroller Data Book, AT89 Series, ATMEL Corporation, December 1997.

- Microcontroller Data Book, MICROCHIP, 2005.
- National Analog and Interface Products Data Book, National Semiconductor, 2002.
- Nonvolatile data memory, ATMEL Corporation, December 1998.