Universidad Nacional Autónoma de México

Facultad de Ingeniería

SISTEMA PARA CONTROL Y SIMULACION DE UN MODEM, SCPC.

T E S I S

Que para obtener el título de:

INGENIERO MECANICO ELECTRICISTA

P r e s e n t a :

Norma Angélica Sánchez Paz

Director de Tesis: M.C. Luis Marcial Hernández Ortega

México. D. F. 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

RESUMEN

Dada la necesidad de enlace entre computadora Maestra y Remota en una red de comunicaciones, el objetivo fundamental de la presente tesis fué el Diseño de un Sistema para Configurar y Controlar al Modem (CM 101 Digital Modem de ComStream), que es uno de los elementos fundamentales de la red de comunicación.

Para resolver el problema de Configurar y Controlar al Modem se desarrolló una función que cumple con este objetivo. Para propósitos de prueba se desarrolló un programa que accesa a la función que permite Configurar y Controlar al Modem, este programa interactúa directamente con la función y esta hecho en base a menús que permiten la selección de comandos y la lectura de datos de forma sencilla, para ser enviados al modem. Adicionalmente se hizo un programa capaz de simular el funcionamiento del Modem, en su parte de comunicación con una terminal.

Se mencionan antecedentes, definición y propósito de las redes de datos; la importancia que tiene el modelo de referencia ISO-OSI; los elementos de un sistema de comunicaciones; las características de los Modems y algunos conceptos manejados en análisis de señales.

Se comenta la importancia que tiene la simulación en el diseño de sistemas.

Se plantea el problema, la solución propuesta y la forma como se Configura y Controla al Modem.

Finalmente se incluyen listados del "seudocódigo" y programas que forman al sistema.

1 INTRODUCCION

1.1 ANTECEDENTES SOBRE REDES DE DATOS

Uno de los factores fundamentales que dio origen al surgimiento y desarrollo de las redes de transmisión de datos fue la necesidad de comunicación de información.

El soporte básico de la comunicación de datos lo han sido las redes públicas de telecomunicación, cuyo servicio consiste en efectuar el proceso de transmisión de datos a través de enlaces telefónicos conmutados, o a través de circuitos punto a punto dedicados permanentemente al enlace entre los equipos del usuario.

El desarrollo y avance en computadoras, comunicación y redes de datos se debe a la tecnología de Estado Sólido, y en particular a la Gran Escala de Integración de los circuitos electrónicos: "Very Large Scale Integration" (VLSI). Con el desarrollo de microprocesadores cada vez mas poderosos ha habido cambios en los sistemas de cómputo al aumentar la velocidad de procesamiento de información.

Los avances en las tecnologías de comunicación digital, han impulsado el éxito de los sistemas distribuidos, situados a distancias relativamente cortas (10 m. a 10 km.), comunicados entre sí, en un rango que va de los 50 kbps. a los 50 Mbps.

1.2 DEFINICION DEL CONCEPTO RED DE DATOS

Se define como red a la disposición de equipo de cómputo, de comunicación y líneas de transmisión que permiten el Procesamiento de Información.

Una red de comunicación está constituida por un conjunto de nodos (puntos de la red), entre los cuales hay un contante intercambio de información y recursos a través del establecimiento de una conferencia entre equipos de cómputo.

Una red de comunicaciones es operada y controlada por programas residentes en los componentes de la red. El procesador central, puede contener los programas de aplicación, bases de datos, archivos y métodos de acceso para la comunicación.

1.3 PROPOSITO DE LAS REDES

Los objetivos que se pretenden alcanzar con las redes de datos son:

1) Tener acceso local a una fuente de información remota.

2) Proporcionar comunicación entre procesos.

3) Tener administración centralizada y asignación de recursos en los nodos de la red.

4) Tener un máximo rendimiento de equipos y programas.

1.4 MODELO DE REFERENCIA ISO-OSI :

Este modelo de referencia tiene el propósito de presentar conceptos más fáciles de comprender, dando la posibilidad de analizar sistemas más complejos. Los sistemas de comunicación son estructurados en capas que proporcionan una base para la normalización de los elementos y para asegurar :

- Independencia de actividades entre capas (un cambio en una capa sólo afectará esa capa).

- Utilización de servicios comunes compartidos por diferentes aplicaciones.

- Secuencia adecuada de los sucesos en cada capa.

El modelo permite :

- Identificar las áreas que requieren desarrollo adicional.

- Aumentar, sin interrumpir previamente los protocolos definidos y las interconexiones.

- Facilitar la obtención de realizaciones prácticas.

- Estructurar el diseño para un funcionamiento correcto y completo.

La técnica estructural básica para construir la arquitectura del modelo de referencia, consiste en un conjunto jerárquico de capas; definiéndose en modo más formal de la siguiente manera :

- La capa (n), para n >= 2 de la estructura hace uso de los servicios de la capa (n-1) proporcionados por las capas más bajas.

- Las capas del tipo (n), para n >= 2 utilizan los servicios del tipo (n-1) para proporcionar servicios de tipo (n) a la capa (n+1).

NOTA : n va de la capa 1 a 7.

La especificación de la arquitectura en cada capa se refiere a la serie de servicios proporcionados por las capas más bajas y se logra mediante el uso de funciones de acceso. La serie de funciones de acceso a un servicio se considera como un medio de descripción de la estructura lógica de la red. No implica necesariamente, la existencia de la interconexión correspondiente en cualquier realización de una parte de la red.

A continuación se da una breve descripción de cada capa :

1) Capa de control de interconexión física :

Provee las características mecánicas, eléctricas, funcionales y de procedimiento, necesarias para establecer mantener y liberar conexiones físicas entre los equipos que integran la red.

2) Capa de control de enlace de datos :

Provee la conexión lógica a través de la línea, en el direccionamiento, secuenciamiento y recuperación de información en caso de error. Existe una dirección de enlace que identifica la conexión de enlace con ésta capa.

3) Capa de control :

En esta capa se provee el control entre dos nodos adyacentes si se habla de dos conexiones se dice que el enlace es punto a punto, varias conexiones pueden concurrir en un punto de enlace, distinguiéndose por sus direcciones.

4) Capa de control de transporte :

Esta capa proporciona el control entre nodos de usuarios a través de la red. Por lo tanto se debe tener control de dos direcciones, el destino inmediato y el destino final.

5) Capa de control de sesión :

Provee el soporte para interconexiones entre elementos que cooperan en la Capa de Presentación, las funciones de la capa de sesión se pueden dividir en dos categorías :

- Determinación y cancelación de sesión entre dos elementos de la Capa de Presentación (esto es llamado Servicio de Administración de Sesión).

- Control del intercambio de datos, entre los dos elementos, comprendiendo sincronización, delimitación y recuperación de operaciones con los datos (esto es llamado Servicio de Diálogo de Sesión).

Una sesión se distingue por "identificación de destino final".

6) Capa de servicios de presentación :

Proporciona servicios de conversión y de interpretación que la Capa de Aplicación (7) puede seleccionar, para poder interpretar el significado de los datos intercambiados.

7) Capa de aplicación :

Las capas anteriores tienen la función de brindar soporte a ésta. Una aplicación se compone de procesos que se intercomunican mediante el uso de los protocolos definidos para esta capa. Estos procesos de aplicación son la fuente y el destino de los datos transmitidos.

1.5 PRINCIPALES ELEMENTOS DE UN SISTEMA DE COMUNICACION A TRAVES DE REDES DE DATOS :

Como elementos principales se tienen :

A) Adaptadores de comunicaciones.

B) Nodos.

C) Compactadores de datos.

D) Modems.

E) Adaptadores.

F) Concentradores.

G) Repetidores.

H) Controladores.

I) Procesadores de comunicaciones.

J) Protectores de la red (DAA :Data Acces Arrangement).

K) Topologías de redes.

L) Medios físicos de transmisión

1.6 MODEMS Y SUS CARACTERISTICAS :

Los modems son dispositivos utilizados para la conversión de señales digitales en analógicas y viceversa, de tal forma que se pueda enviar la información a largas distancias por medio de las líneas de transmisión. Su nombre proviene de la contracción de modulación y demodulación; también se les conoce con el nombre de "Data Set" asignado por Bell o como "Adaptadores de Línea" (Line Adapters) asignado por IBM.

El modem convierte las señales digitales "bits" en una señal capaz de ser transmitida a través de un canal, esta señal puede provenir de una computadora o una terminal remota. De esta forma, el modem funciona como una interface de línea, aislador o adaptador, modulador, transmisor, demodulador, receptor, iguala y condiciona la señal digital.

La transmisión de pulsos se puede hacer a través de canales no ideales con ancho de banda limitada tales como líneas telefónicas, las cuales además de las no linealidades y/o las imperfecciones en la respuesta en frecuencia del canal, presentan otras fuentes de degradación de la señal, como son el ruido y la interferencia, por lo tanto el objetivo de diseño del transmisor y receptor es el minimizar dichos efectos.

Los modems pueden ser externos o internos, dependiendo de si se encuentran fuera de la computadora o dentro de ella.

Se distinguen por ser síncronos o asíncronos, dependiendo de la forma en que transmita el mensaje. Pueden tener diagnósticos residentes sobre su operación y disponer de mecanismos de detección y corrección de errores, otra clasificación de los modems surge cuando se habla de las líneas de comunicaciones, y en el caso de tener múltiples conexiones, se habla de modems maestros.

Dentro de las principales características de los modems se tienen:

1) La forma de Transmisión :

Los "bits" o datos binarios son comúnmente transferidos con dispositivos electrónicos, si se transfieren por una línea se habla de datos en serie, o por varias lineas al mismo tiempo de datos en paralelo.

Respecto a la forma de transmisión cabe distinguir entre las siguientes alternativas:

- Transferencia por programa :

Es cuando el intercambio de información se realiza vía registros de la Unidad de Proceso Central (CPU) por medio de instrucciones de transferencia de datos con el sistema de Entrada/Salida.

- Transferencia Síncrona :

Este tipo de transferencia puede ser una alternativa de control para el caso de transferencia por programa en la cual un bloque de caracteres de datos contiguos es transmitido como una unidad, la transmisión se realiza por iniciativa del programa y sin consultar el estado del periférico, su utilización estará enfocada en los periféricos que siempre estén dispuestos a aceptar una nueva información (ej. un despliegue alfanumérico).

- Transferencia Asíncrona :

Este tipo de transferencia puede ser otra alternativa de control para el caso de transferencia por programa; es una forma de transmisión a través de líneas telefónicas en la cual cada caracter consta de 10 bits de los cuales, siete son de información, un bit que indica el inicio, otro el fin y otro la paridad; su codificación es en código ASCII. Este tipo de transferencia ocurre al establecer un diálogo (sincronización "handshaking") entre CPU y periférico con el fin de controlar la transferencia en el instante y forma adecuados.

- Transferencia por acceso directo a memoria (DMA) :

Cuando se inhibe a la Unidad de Proceso Central de la tarea de controlar la transferencia, realizándose ésta bajo control de un sistema externo (controlador de DMA), de tal forma que la comunicación se establece directamente entre memoria y periférico.

- Por consulta de estado :

Se pregunta al periférico sobre su disposición para aceptar la transferencia o su necesidad de servicio.

- Por interrupción :

Se delega en el periférico la facultad de avisar cuando está disponible o en servicio.

2) **Modo de Transmisión :**

- **"Full-Duplex" :**

En la transmisión Full-Duplex o Transmisión de dos-caminos simultáneos, los datos fluyen en ambas direcciones simultáneamente. Requiere de 4 hilos, de los cuales se utilizan dos para cada equipo.

- **"Half-Duplex" :**

  En la transmisión Half-Duplex se permite el flujo de información en ambas direcciones pero no simultáneamente.

- **"Simplex" :**

  En la transmisión Simplex, el flujo sólo es en un sentido, sin posibilidad de hacerlo en el opuesto.

3) Tipo de modulación :

Recordando que modulación es un proceso de modificación de alguna característica de la onda portadora de acuerdo con valores puntuales de la información a ser transmitida.

- **Para modulación analógica se tiene :**

  Modulación en Amplitud "AM"
  Modulación en Frecuencia "FM"

- **Para modulación digital se tiene :**

  Modulación en Frecuencia "FSK"
  Modulación en Fase "PSK"
  Modulación AM en Cuadratura "QUAM", etc.

1.7 CONCEPTOS MANEJADOS EN PROCESO DE SEÑALES

A continuación se dan algunos conceptos manejados en procesos de señales :

1) CLASIFICACION DE LAS SEÑALES :

Una señal se define como una función variable en el tiempo que contiene información. En consecuencia, para cada instante (variable independiente) existe un valor único de la función (variable dependiente). Este valor puede ser un número real, en cuyo caso tendremos una señal de valor real, o bien puede ser un número complejo, en cuyo caso se tendrá una señal de valor complejo. En ambos casos, la variable independiente (el tiempo) tendrá un valor real.

El método utilizado para la representación de la señal depende del tipo de señal que se esté considerando y de la característica que interese analizar, pudiéndose agrupar los siguientes modos :

a) Señales periódicas y no periódicas.

b) Señales determinísticas y aleatorias.

c) Señales de energía y de potencia.

d) Señales contínuas y discretas.

2) CAUSAS DE ALTERACION DE LAS SEÑALES :

Las causas de alteración de las señales en un sistema de comunicación son:

a) Ruido aleatorio, blanco o gausiano :

Es la presencia de señales aleatorias originadas dentro del sistema. Cuando estas variaciones se agregan a la señal portadora de la información, la información puede quedar en parte o totalmente oculta. Este ruido se extiende al azar sobre el espectro de frecuencias; la forma de corregir este efecto es ajustando la relación señal/ruido a un nivel suficientemente alto, para que el "ruido de fondo" no sea detectado o pueda ser filtrado con facilidad.

Como generadores de ruido se tienen :

Los componentes del sistema como resistencias y dispositivos electrónicos de estado sólido, etc.

b) Distorsión :

Es una alteración de la forma de la señal que se está transmitiendo debido a las limitaciones en la respuesta a la frecuencia del canal de transmisión.

En algunos casos la distorsión produce atenuación de la señal; hablando de señales analógicas es la pérdida de amplitud de la señal a medida que crece la distancia de propagación; en lo general, estas pérdidas aumentan al aumentar la frecuencia.

c) Interferencia :

Alteración por la adición de señales ajenas a la señal de interés.

d) Eco :

La diferencia en los valores de impedancia en los extremos de la línea (receptor y transmisor) causa que se generen reflexiones, que viajan tornando en dirección opuesta. El eco es tolerable en comunicaciones de voz y datos en circuitos de corta distancia.

3) LOS CARACTERES Y SU CODIFICACION :

En la comunicación de datos, los elementos básicos de información transmitidos son los códigos asociados al juego de caracteres de la unidad de proceso y secuencias binarias representando valores numéricos.

En el caso de los códigos, en dicha comunicación se utilizan 7 "bits" y se denomina código ASCII (American Standard Code for Information Interchange) conocido también como código CCITT No. 5.

Los 7 "bits" del código van acompañados de un octavo "bit" utilizado para controlar la paridad (par o impar) del conjunto, aunque en algunos casos, dicho "bit" suplementario toma permanentemente el valor "0", o bien el valor "1"; el conjunto de los siete "bits" de codificación del caracter y el "bit" adicional forman un total de 8 "bits" (1 byte o caracter), que constituye realmente la unidad de información transmitida.

Existe otro código como el EBCDIC desarrollado por IBM, el cual utiliza 256 combinaciones posibles en una estructura de 8 bits. Su nombre proviene del nombre en inglés "Extended Binary Coded Decimal Information Code" y el código Baudot empleado en telefonía.

4) DETECCION DE ERRORES :

Se debe considerar la posibilidad de que puedan producirse errores en la transmisión de la información, debido a las características del camino físico utilizado. Simultáneamente con las técnicas de transmisión se han desarrollado métodos orientados a la solución de este problema que permiten detectar y en algunos casos corregir los errores que se hayan producido en la transmisión de un bloque de información.

A nivel de caracter, suele utilizarse el método de detección de paridad, para lo que se requiere un "bit" adicional, transmitido con la información útil; este "bit" indica, de acuerdo a la convención seleccionada, la suma modulo 2 de los unos contenidos en el mensaje, lo que permite al receptor efectuar la misma operación y comparar el resultado con el "bit" adicional que el transmisor incluyó en el mensaje para determinar en el caso de ser diferentes que ocurrió un error. Este tipo de códigos falla cuando ocurre un número par de errores.

Los códigos más efectivos en la detección de errores en sistemas de comunicación, con una cantidad mínima de hardware, son los códigos cíclicos (Cyclic Redundancy Checks "CRC").

El Código de Redundancia Cíclica (CRC) es de las técnicas más usadas en la detección de errores. Trabaja a nivel de mensaje, agregando varios caracteres de control al final, siendo lo más común el adicionar 2 o 4 bytes de control.

El algoritmo para la generación de los bytes adicionales es :

Se divide la secuencia de bits a enviar, por un número binario predeterminado. El residuo de la división se adiciona al mensaje como secuencia de control. Por una regla aritmética simple, si el divisor es un número de 16 bits, podemos tener la seguridad de que el residuo siempre podrá almacenarse en dos bytes, de donde, agregando 2 caracteres al mensaje se tendrá el método implantado.

El extremo receptor realiza el mismo cálculo que el emisor y compara el resultado obtenido con la secuencia de control recibida. Si no coinciden, equivale a una indicación de error.

5) VELOCIDAD DE TRANSMISION :

La velocidad de transmisión representa la cantidad de información que puede transmitirse por unidad de tiempo. El CCITT la define como promedio de bits, caracteres o bloques por unidad de tiempo que pasan entre dos equipos correspondientes en un sistema de transmisión de datos.

En los modems, la velocidad puede variar consideradamente desde los de baja velocidad 300 bps., los de velocidad media que son los que transmiten de 1200 bps. a 9600 bps. y los de alta velocidad que transmiten de 9600 bps. en adelante.

2 PAPEL DE LA SIMULACION EN EL DISEÑO

2.1 CONCEPTO DE SIMULACION

En el sentido más general, simulación significa representación de la realidad.

A continuación se da una breve descripción del argumento de algunos autores sobre simulación y diseño.

La simulación en computadora se utiliza como instrumento para analizar conductas individuales. Es impresionante la capacidad necesaria para desarrollar un modelo que imite la toma de decisiones de un individuo; igualmente impresionante son los modelos más generales que resuelven problemas matemáticos o lógicos.

Hay un uso extendido de la simulación de sistemas en las ciencias sociales; pero otro gran campo lo tiene la simulación en las ciencias físicas; los usos de simulación que han tenido gran publicidad son los que se ejecutan en la NASA, especialmente los utilizados para los vuelos espaciales tripulados. Esos programas emplean una gama muy amplia de modelos y técnicas analíticas; pero lo más importante ha sido el empleo de simulaciones llenas de imaginación en las que el hombre y el vehículo interactúan en ambientes creados por las computadoras.

La simulación en computadoras ha llegado a utilizarse cada vez más para estudiar la conducta de sistemas cuyo estado cambia con el tiempo. Las alternativas al uso de la simulación son el análisis matemático, la experimentación con el sistema real o un prototipo de el, o la dependencia de la experiencia y la intuición, sin olvidar que la simulación tiene sus limitaciones y en ocasiones el análisis matemático de sistemas complejos es imposible; la experimentación con sistemas piloto o reales es costoso y requiere mucho tiempo, además de que algunas variables no siempre están sujetas al control.

Algunas personas sostienen que las matemáticas y la simulación son primos empíricos pero, mientras que el matemático se basa en el intelecto para establecer conexiones entre premisas e implicaciones, el simulador después de enunciar sus suposiciones, saca inferencias con la computadora. Es la estrategia para sacar inferencias lógicas lo que distingue a los simuladores del análisis matemático.

En la simulación de un sistema, se dice que el sistema es complejo cuando es grande en función del número de variables, de parámetros, de relaciones y de eventos a los que responde el sistema. La existencia de variables aleatorias se atribuye en la mayoría de las definiciones a la entrada de un sistema.

Para el diseño de programas de simulación, se tienen paquetes y lenguajes con ventajas diferentes.

La simulación por software es un sistema que debe tener un lenguaje de programación adecuado, su función es :

- Aceptar modelos del sistema codificados adecuadamente, junto con instrucciones para controlarla,

- Verificar que los datos de entrada sean correctos,

- Generar reportes de error,

- Contar con facilidades para la edición de datos de datos,

- Realizar pruebas y

- Proporcionar ayuda al usuario cuando lo requiera.

Un módulo de simulación es un enunciado o conjunto de enunciados escritos en un lenguaje de programación que satisfacen las siguientes condiciones :

- Su función es un subconjunto de las funciones del programa de simulación y

- Tiene un dato de entrada y uno de salida.

Ejemplo de simulación :

- Una función que permite simular a un modem pudiendo interactuar con el programa mediante otra computadora.

Para el desarrollo de la simulación :

- En la formulación de un programa de computadora, con el propósito de que ésta conduzca la simulación de experimentos de un modelo, se deben incluir : diagrama de flujo o seudocódigos, el programa de computadora mismo, verificación de errores, datos de entrada, condiciones iniciales, generación de datos y reportes de salida.

- El modelo debe ser validado por comparación entre los datos simulados y los datos reales.

2.2 APLICACIONES DE DISEÑO

En el diseño se tienen diferentes alternativas para conjuntar los elementos del sistema, en la simulación interviene el diseño del sistema a ser simulado.

Dada la especificación del resultado deseado, se busca optimizar el comportamiento del sistema propuesto, así como beneficios, costos, tiempo, utilización de recursos y estabilidad. Se procesa un modelo del sistema, incluyendo cambios sucesivos que correspondan a diseños alternativos. Se traza la influencia del diseño sobre la medida de eficiencia, y a continuación el analista tiene una base para seleccionar el diseño que logre el resultado deseado del sistema.

En el diseño se debe tener la capacidad para predecir el comportamiento del sistema, lo cual implica una secuencia de especificaciones y predicciones interrelacionadas para el logro de los objetivos, se debe tener información actulizada y así poder predecir el resultado.

Algunas veces, a medida que se van desarrollando las etapas del diseño, surgen dificultades imprevistas, razón por la cual el modelo del problema original puede cambiar tan drásticamente que se tenga que replantear de nuevo; es la inestabilidad del problema lo que complica el diseño haciéndolo difícil.

2.3 PRUEBAS DE DISEÑO

El principal objetivo del diseño es el cumplimiento de las especificaciones, las cuales dan un marco de referencia para verificar el funcionamiento del sistema, una parte de las especificaciones son las pruebas de diseño.

Al realizar las pruebas de diseño, el resultado puede ser el esperados o bien, puede ser el no esperado por causa de :

- Mal procedimiento de diseño del sistema,
- Especificación errónea de los resultados,
- Diseño erróneo de las pruebas y
- Realización incorrecta de las pruebas.

Soluciones o alternativas para los casos anteriores :

- Asegurarse de que las pruebas sean correctas,
- Revisar de nuevo el diseño de las pruebas,
- Si se tiene la seguridad de que las pruebas son correctas y se tienen resultados incongruentes, hay que verificar la especificación de los resultados.
- Si están bien especificados los resultados y las pruebas son adecuadas pero los resultados son erróneos, hay que verificar detalladamente el sistema que se desea probar ya que es probable es que tenga errores.

Una vez que el sistema a probar da resultados esperados, se tiene la posibilidad de mejorarlo, siempre y cuando se cuente con tiempo suficiente para hacerlo.

3 GENERALIDADES DEL PROBLEMA

3.1 INTRODUCCION

Se pretende el establecimiento de la Red para lograr el enlace entre diferentes puntos de la República Mexicana mediante comunicación vía satélite, con topología tipo estrella y acceso por encuesta (polling).

Teniendo como objetivos los siguientes :

- Llevar un control de inventario,
- Tener registrados los artículos vendidos,
- Programar abastecimiento de inventarios,
- Contar con precios actualizados,
- Conocer el monto de las ventas y
- Centralizar el control del personal.

3.2 DESCRIPCION DE LA RED

Su operación es "Centralizada", por lo que toda transacción tiene origen en el nodo central; en caso de que algún nodo remoto requiera de algún servicio, éste deberá solicitarlo en su oportunidad al referido nodo central.

Al establecerse el enlace de comunicaciones se realiza el intercambio de información, la cual se almacena en los controladores para después transferirse a la computadora local asociada a cada controlador de comunicaciones.

A continuación se describen los cuatro elementos que integran la Red, realizan funciones de almacenamiento y control de información desde el punto de vista operativo, estos son:

1) COMPUTADORA MAESTRA :

Es la computadora anfitrión (Host) asociada al nodo central.

2) COMPUTADORA REMOTA :

Es la computadora anfitrión asociada a cualquier nodo remoto.

3) CONTROLADOR DE COMUNICACIONES MAESTRO :

Es el controlador a cargo del nodo central.

4) CONTROLADOR DE COMUNICACIONES REMOTO :

Es el controlador a cargo de cualquier nodo remoto.

En general, el controlador de comunicaciones habilita un canal para que las computadoras anfitrionas se enlacen de acuerdo a los protocolos y procedimientos que le sean propios.

La computadora maestra envía archivos y solicitudes de archivos al controlador de comunicaciones, este almacena localmente la información y la transfiere durante un enlace de comunicaciones que se realiza periódicamente de acuerdo a un programa de barrido de controladores remotos.

Las funciones de la Red se dividen en dos clases; aquellas independientes de las computadoras maestra o remota y las que dependen de ellas; estas últimas se consideran como opcionales.

CONTROLADOR DE COMUNICACIONES MAESTRO :

A) DESCRIPCION :

El Controlador de Comunicaciones Maestro es un equipo que puede operar en forma manual o automática. Proporciona una interface hombre/máquina que puede realizar las siguientes funciones :

- Instalación y servicio.
- Supervisar la operación automática.
- Operar el controlador de comunicaciones en forma manual.

Normalmente el controlador debe estar en modo automático, interrogando a cada uno de los controladores remotos para enviar o recibir transacciones (ver apéndice B)

B) TRANSFERENCIA DE ARCHIVOS :

- RECEPCION DE INFORMACION :

La recepción de archivos en el controlador maestro es por solicitud explícita del mismo controlador, el cual envía una solicitud de información en base a la lista de transacciones del controlador remoto que tiene el controlador maestro.

- TRANSMISION DE INFORMACION :

El controlador maestro envía archivos a un controlador remoto, con una transacción de comandos con datos. El comando contiene la información de identificación de la fuente y destino del archivo.

En caso de que el controlador maestro requiera un archivo de un controlador remoto y no esta contenido en la lista de transacciones, el controlador maestro envía una solicitud del archivo, el cual es procesado por el controlador remoto reconociendo la recepción de la solicitud y enviándola a la computadora remota; si la transferencia se puede realizar, la computadora remoto envía el archivo al controlador remoto, de lo contrario, notifica que la solicitud no puede ser procesada y la transacción termina con error.

C) MODOS DE OPERACION :

Tiene cuatro modos de operación, entendiendo como modo de operación a la forma en que trabaja dicho controlador. La modificación del modo de operación es una acción exclusiva para el área de mantenimiento, ya que cambiar de modo de operación altera las condiciones de funcionamiento de toda la Red.

1) MODO MANUAL :

Este permite la ejecución manual de cada una de las transacciones que se encuentren en el controlador.

En este modo se tienen las siguientes opciones :

a) enlace :

Una vez seleccionado el enlace en modo manual, se indica en la zona de acciones del satélite, presentándose información que permite la supervisión del procedimiento del enlace.

El enlace permite establecer comunicación con un controlador remoto. El comando presenta la lista de nombres de cada controlador.

b) cancelación de enlace :

Un enlace que esta en proceso se puede cancelar durante la operación en modo manual y el controlador queda en reposo.

c) envío :

Una vez establecido el enlace en modo manual con el controlador remoto, se presenta la pantalla con las transacciones pendientes.

Al establecer el enlace, el controlador maestro obtiene la información de las transacciones pendientes que tiene el controlador remoto.

La operación manual hace mas lenta la comunicación con el satélite, pero deja activa la recepción de información desde la computadora maestra.

Al activarse el comando, se presenta la pantalla de transacciones pendientes de envío o recepción almacenadas en el controlador, seleccionándose la transacción que se desea procesar.

Se identifica la dirección de la estación remota a la que se va a enviar la transacción, estableciéndose el enlace con el controlador remoto, una vez establecido se realiza la transacción; presentándose la información del estado de la misma.

Se mantiene el enlace abierto por el lapso de tiempo definido para cada comunicación con la remota, con la posibilidad de realizar otra transacción con la misma remota.

Durante la ejecución de una transacción, se puede cancelar su operación.

Si no se envía ninguna transacción adicional en el período de tiempo definido, el enlace se cancela y la Red queda en reposo.

2) MODO PARO :

En este modo, se detiene la operación de barrido de estaciones remotas en forma automática, quedando el equipo en estado de reposo, suspendiendo temporalmente la operación.

3) MODO AUTOMATICO :

Modo en el cual se realiza el barrido de cada estación remota de acuerdo a la secuencia definida en la pantalla de secuencia de barrido.

Al iniciar la operación, el controlador se enlaza con la remota indicada en la secuencia de barrido. Si el controlador estaba operando y se detuvo el barrido por un cambio de operación, se reinicia la operación con la remota que sigue en la secuencia de barrido.

Al encender o reestablecer (Reset) el equipo, este se activa en modo automático.

4) MODO SERVICIO :

El modo servicio suspende el barrido de las remotas, dejando el equipo listo para realizar acciones del menú de servicio, el cual cuenta con utilerías para diagnóstico y mantenimiento (ver apéndice B).

D) UTILERIAS DEL CONTROLADOR :

- **Estadística Local.**

Presenta información estadística al controlador maestro y se puede restaurar en modo de mantenimiento.

- **Estadística Remota.**

  Presenta información estadística al controlador remoto y se puede restaurar en modo de servicio.

- **Cancelación de Enlace.**

  Cancela el enlace en proceso, permite terminar el enlace actual, en cuyo caso la estación maestra continuará el barrido con el siguiente elemento de la lista. Sólo tiene efecto en los modos manual y de servicio.

- **Transacciones en Proceso.**

  Presenta las transacciones en proceso de envío y recepción que ocurren en el controlador de comunicaciones; pudiéndose eliminar de la lista una transacción pendiente.

- **Transacciones Realizadas.**

  Este comando presenta la lista de transacciones realizadas por el controlador. La información de estas transacciones se almacenan en disco.

- **Correo Electrónico.**

  El sistema de mensajería del controlador permite el intercambio de mensajes cortos entre el controlador maestro y cualquiera de los remotos.

  El sistema tiene capacidad para almacenar 50 mensajes a enviar; si se envía correctamente se borra de la lista. En cuanto a los mensajes se tienen las siguientes opciones :

a) **Estado de los mensajes en cuanto al envío.**

   Presenta en la pantalla el título de los últimos 50 mensajes enviados o por enviar.

b) **Edición de Mensajes.**

   Al activar el comando, se puede crear un mensaje; editándose en la zona de operación de comandos. El mensaje consta de 15 líneas.

c) **Estado de los mensajes en cuanto a recepción.**

   Presenta en pantalla el título de los últimos 50 mensajes recibidos; la zona de operación de comandos presenta el número consecutivo del mensaje y el título.

d) **Recepción de Mensajes.**

Comando que puede seleccionar uno de los mensajes recibidos y presentarlo en pantalla; al terminar el comando retorna a la pantalla de estado de mensajes.

- **Generación de Reportes.**

En la generación de reportes se tienen las siguientes opciones:

a) **Enlaces.**

Genera un reporte que contiene la información de todos los enlaces que ha intentado establecer el controlador maestro. El reporte describe por remota en cada enlace el resultado de la operación, el reporte se envía a la impresora.

b) **Transacciones.**

Genera un reporte que contiene la información de cada una de las transacciones realizadas con el controlador maestro; en el reporte se incluye el número de veces que se realizó un reintento y describe por remota en cada enlace el resultado de la operación, el reporte se envía a la impresora.

c) **Errores.**

Crea un reporte que contiene la lista de errores de operación detectados por el controlador maestro, el reporte se envía a la impresora.

d) **Bitácora.**

Crea un reporte que contiene el registro de transacciones realizadas. presenta la información de cada uno de los principales comandos ejecutados. Al generarse el reporte, se borra el archivo de bitácora residente en disco duro. El reporte se envía a la impresora.

CONTROLADOR DE COMUNICACIONES REMOTO :

A) MODOS DE OPERACION :

Son tres modos en los que opera el Controlador Remoto.

1) PARO.

Detiene la operación de barrido; deja de ejecutar en forma automática el barrido de estaciones remotas; quedando el equipo en estado de reposo para poder suspender temporalmente la operación.

Opcionalmente, en este modo, el controlador continúa recibiendo transacciones de la computadora maestra, almacenándolas mientras haya espacio en disco.

2) AUTOMATICO.

Responde al barrido programado en la estación maestra de acuerdo a lo definido por el operador de la Red en la pantalla de secuencia de barrido. Al poner en funcionamiento el equipo se activa en modo automático.

3) SERVICIO.

Este modo suspende el barrido de las remotas y deja el equipo listo para realizar acciones del menú de servicio. Este modo al igual que en el controlador maestro tiene utilerías de diagnóstico y mantenimiento (ver apéndice B).

B) UTILERIAS :

- **Estadística Local.**

Presenta información sobre el controlador maestro. La información estadística se puede restaurar en modo de mantenimiento.

Los siguientes casos funcionan de la misma forma que para el Controlador Maestro.

- **Cancelación de Enlace.**
- **Transacciones en Proceso.**
- **Correo Electrónico.**
- **Generación de Reportes.**

COMUNICACION ENTRE COMPUTADORAS MAESTRA Y REMOTA.

La comunicación entre las computadoras maestra y remota se establece por medio de los controladores de comunicaciones a través del puerto de control, este puerto es un canal serie asíncrono.

Las funciones asociadas a este enlace son :

1) Recibir peticiones de servicio de la computadora y del controlador maestro.

2) Informar a las computadoras del estado del enlace virtual punto a punto.

3) Proporcionar al controlador los parámetros de transferencia de archivos.

Para la utilización de los servicios proporcionados por la Red, se cuenta con cuatro comandos en los controladores de comunicaciones y son :

1) **Envío de un archivo.**

2) **Recepción de un archivo.**

3) **Consulta del Estado de la Red.**

4) **Cancelación de una transacción.**

3.3 PLANTEAMIENTO DEL PROBLEMA

La Red responde a las necesidades de transmisión de archivos binarios entre las estaciones o diferentes puntos y en aplicaciones en las que el número de estaciones conectadas a cada nodo es de 2 a 8.

Para llevar a cabo la transmisión de la información, el satélite se comunica directamente con un Modem de Radio Frecuencia (CM101 PSK Digital Modem).

El Modem de Radio Frecuencia va enlazado directamente al controlador de comunicaciones maestro para el caso de la estación maestra y al multiplexor, el cual combina los flujos de información del propio controlador de comunicaciones y de otros canales de comunicación; cada controlador, tiene comunicación con una fuente de datos.

Lo mismo sucede para el caso de la estación remota con el controlador de comunicaciones remoto.

Esto queda representado esquemáticamente de la siguiente forma:

Este modem requiere de una terminal dedicada para su control, a través de ella se pueden modificar los parámetros del mismo e interpretar las fallas que pudieran ocurrir, así como la supervisión de su operación.

La estructura de la Red no contempla la existencia de la terminal dedicada al modem ya que se sustituye por el controlador de comunicaciones.

Para propósito de prueba y desarrollo, no se tiene la disponibilidad física del Modem de Radio Frecuencia ni la disponibilidad de realizar el enlace con el satélite, razón por la cual el objetivo de la presente tesis es su simulación.

La técnica por encuesta (polling) determina cuál terminal está lista para recibir o enviar un mensaje, esto es, el controlador de comunicaciones maestro envía mensajes interrogatorios a cada estación de acuerdo al criterio establecido para determinar si éstas tienen un mensaje listo para la transmisión, ninguna remota puede enviar mensaje hasta no ser interrogada, la maestra envía códigos de selección (select) para determinar si la terminal receptora puede aceptar o no mensaje.

3.4 DESCRIPCION DEL MODEM

El Modem "CM101 PSK Digital Modem" es usado en comunicaciones vía satélite, en aplicaciones que requieren recepción o transmisión continua; se puede comunicar con otro CM101 o con un CM121 (cuya configuración es punto a punto), por medio de un enlace "SCPC" (una frecuencia para cada enlace).

Puede operar con otros equipos en un rango de 9.6 Kbps a 2.048 Mbps y a frecuencias intermedias "IF" entre 52 MHz. y 88 MHz.

El modem tiene incluido:

a) Programación de la sintetización de frecuencia intermedia en receptor y transmisor.

b) Convolución de la secuencia del codificador.

c) Un ensamblador desensamblador V.35

d) Codificador decodificador diferencial.

e) Microcomputadora Integrada.

CONFIGURACION TIPICA :

La configuración básica del hardware consta de :

1) Una tarjeta con circuitos que constituyen al demodulador,

2) Una tarjeta con circuitos que constituyen al modulador, y

3) Una fuente de poder.

TEORIA DE OPERACION :

El CM101 consta de 3 subsistemas :

1) Una tarjeta con un circuito demodulador.

2) Una tarjeta con un circuito modulador.

3) Un fuente de poder.

El modem puede estar configurado únicamente como receptor o bien como receptor y transmisor.

4 SISTEMA DE CONTROL Y SIMULACION DEL MODEM

4.1 PLANTEAMIENTO DE LA SOLUCION

Debido a que en la estructura de la Red se utilizan microcomputadoras del tipo PC para cada estación, la solución propuesta es la utilización de estas para realizar el control del Modem; aprovechando que este tipo de microcomputadoras soportan una interfaz RS-232 a través de la cual, se puede configurar y programar al Modem, diseñando una función que permite su control desde la PC.

Por otro lado, para resolver el inconveniente de la disponibilidad física del Modem de Radio Frecuencia, se desarrolló un programa que lo simula utilizando una computadora PC para simularlo de acuerdo a sus características y de esta manera poder probar que sea correcta la función que lo controla.

Por lo tanto, la parte de desarrollo de ésta tesis, es la Programación y Simulación del Modem de Radio Frecuencia "ComStream CM101 PSK Digital Modem", para lo cual se cuenta con el manual que especifica las características del Modem, sirviendo de base para su simulación.

El desarrollo está constituido de tres partes :

1) DESARROLLO DE LA FUNCION QUE INTERACTUA CON EL MODEM

Esta función permite tener comunicación entre el Modem de Radio Frecuencia y el programa que realiza la interfaz hombre máquina, por lo que puede ser accesada desde cualquier otro programa; puede enviar comandos de configuración y control al Modem y recibir reporte del estado del mismo.

La función maneja el protocolo para establecer la comunicación con el Modem, emulando una terminal de datos, dicho protocolo determina las características que debe tener la información para llevar a cabo la transmisión de ésta.

2) DESARROLLO DEL PROGRAMA QUE ACCESA A LA FUNCION QUE INTERACTUA CON EL MODEM (INTERFAZ HOMBRE MAQUINA).

Este programa accesa a la función que interactua con el Modem de Radio Frecuencia, por lo tanto, sirve de prueba para simular la interfaz hombre máquina.

3) DESARROLLO DEL SIMULADOR DEL MODEM.

El programa se hizo con el propósito de probar a la función que accesa al Modem de Radio Frecuencia a través de la interfaz hombre máquina.

Con éste programa se simula el funcionamiento del Modem de acuerdo a las características técnicas del manual de referencia.

4.2 ESPECIFICACION DE LA FUNCION QUE ACCESA AL MODEM

DESCRIPCION DE LA FUNCION QUE ACCESA O INTERACTUA CON EL MODEM.

Esta función, programa los parámetros de operación del Modem; puede enviarle comandos de configuración y control y recibir reporte del estado del mismo.

Cuando es llamada por otro programa; se comunica con el Modem a través "del módulo de manejo de puerto serie", que son funciones encargadas del manejo del puerto serie de la PC.

Abre el puerto de comunicaciones en el que está conectado el modem.

Verifica que la instrucción a enviar al modem sea válida codificandola en el formato adecuado.

Envía al modem la instrucción esperando respuesta con un tiempo de 10 segundos.

Una vez que se recibe información del modem se cierra el puerto de comunicaciones.

Las funciones encargadas del puerto serie de la PC son :

a) **Lee configuración (lee_config)**

Función que lee de un archivo los datos de configuración de un puerto. Lee el número de puerto en RAM de acuerdo a la interrupción a la que está conectado; así como número de bits, paridad, velocidad, etc.

b) **Escribe configuración (escribe_config)**

Función que escribe a un archivo los datos de configuración del puerto presentes en memoria.

c) **Abre puerto (abre)**

Esta función se encarga de abrir el puerto de comunicación para lograr la transferencia de información, e inicia las interrupciones de transmisión y/o de recepción.

d) **Envía información (envía)**

Función encargada de enviar la información o conjunto de caracteres a un dispositivo externo (impresora, computadora, etc.) por medio de un puerto de comunicaciones serie (COM1..COM34), usando la interfaz de nivel físico RS-232, en un orden "FCFS" (First-comed, first-severed) primeros en entrar primeros en salir.

e) Recibe información (recibe)

Dicha función recibe un conjunto de bytes del exterior contenidos en el buffer de recepción y que fueron guardados en un orden "FCFS" primeros en entrar, primeros en salir, por medio de interrupciones.

f) Cierra puerto (cierra)

Es la función que cierra el puerto de comunicación y termina las interrupciones de transmisión y de recepción, siendo necesario cerrar un puerto al terminar el programa, pues el no hacerlo puede ocasionar que la computadora pierda datos.

Para los puertos COM3 a COM34, se requiere una tarjeta de expansión de puertos COM/4 ó COM/8 de "Digiboard, Inc.", las cuales traen 4 y 8 puertos respectivamente; o se puede instalar cualquier otra tarjeta de expansión que maneje varios UART's conectados a una línea de interrupciones común.

La transmisión y recepción de caracteres es asíncrona y a través de interrupciones, las cuales mandarán los caracteres recibidos a unos buffers en memoria. Estos buffers en memoria (de transmisión y de recepción), "NO" pueden ser accesados directamente más que a través de las funciones envía y recibe. Estas rutinas facilitan el proceso de comunicación, y sólo necesitan de algunos parámetros para funcionar, los demás datos son obtenidos del archivo de configuración y sólo que sea necesario se cambiará el archivo de configuración.

PROTOTIPO DE LA FUNCION :

Función funamod (inst,param,pto,acopara,astatus)

PARAMETROS QUE REQUIERE LA FUNCION :

Los parámetros de tipo entero necesarios para esta función son :

a) Instrucción (inst)

Este parámetro debe tomar un valor entre 1 y 13 y cada número corresponde a uno de los comandos que se pueden enviar al modem, como se muestra en la siguiente tabla de resumen de comandos :

TABLA DE RESUMEN DE COMANDOS

NUM	COMANDO	PARAMETRO	DEFAULT	SIGNIFICADO
1	AL	1/0	1	Reporte de alarma [5].
2	BE	93-19	17	Umbral de la tasa de error de bit [3].
3	BR	[1]	1200	Velocidad de transmisión.
4	CF	0-16	-	Limpiar el registro de error.
5	EE	1/0	1	Habilita / deshabilita el caracter de eco [5].
6	EM	1/0	[2]	Habilitación del modulador.
7	FL	?	-	Pregunta por el registro de errores.
8	LB	1/0	0	Modo "loop back".
9	PC	1/0	0	Forza la transmisión de la portadora.
10	RB	?	-	Lectura de la tasa de error de bit (BER) [3].
11	RE	-	-	Reestablece el sistema.
12	ST	?	-	Presenta el estado de errores [4]
13	TP	250-50	-	Transmisión de nivel de potencia

Notas :

[1] La velocidad de transmisión está comprendida en el siguiente rango : 300, 1200, 2400, 4800 y 9600 bps.

[2] El estado de default dependerá de la configuración de hardware, el dato será proporcionado al programar al modem.

[3] El parámetro xy implica x veces 10 ^ -y.

[4] En respuesta a la pregunta, envía una palabra codificada en decimal cuya representación binaria corresponde a un mapeo del registro de estados (por ejemplo : 5 = al envío de los bits primero y tercero).

[5] Sólo disponible para la interfaz RS-232.

b) Parámetro (param)

Es el parámetro o valor que toma el comando siempre y cuando éste lo requiera. En alguno de los comandos se tiene un rango de posibles valores de parámetros, en cuyo caso debe seleccionarse sólo uno.

c) Puerto (pto)

Es el parámetro que indica el número del puerto donde está conectado el modem.

VECTORES QUE MANEJA LA FUNCION QUE ACCESA AL MODEM :

a) Apuntador (apst)

Es un vector de tipo entero, en el que se almacena información del comando FL y ST; para el caso de FL va de la posición 0 a 15 y para el caso de ST de 16 a 31, el número de error o falla de estos comandos, se convierte a su equivalente en binario y cada dígito corresponde a una posición del vector, interpretándose que no hay error en el caso del cero y error en el caso del uno.

b) Coparámetro (coparam)

Es el otro vector que se maneja, el cual indica el parámetro que tienen los comandos excepto FL y ST.

VALORES QUE REGRESA LA FUNCION QUE ACCESA AL MODEM :

La función que interactua con el modem regresa al programa que la accesó (proacmol) un valor entero entre 0 y 15 en caso de que haya ocurrido alguna falla, el significado correspondiente a cada valor es el siguiente :

0 Significa que no hay error.
1 Instrucción no válida.
2 Error en el eco.
3 Parámetros de configuración no válidos.
4 El puerto ya estaba abierto.
5 Memoria insuficiente en buffer del puerto.
6 La dirección del UART no es la especificada.
7 El tamaño de buffer no es válido.
8 Problemas con el hardware del puerto.
9 Espacio insuficiente en memoria para transmisión.
10 Error en el número de bytes transmitidos.
11 Hubo problemas al cerrar el puerto.
12 El modem no regresa en el eco el valor correspondiente.
13 Error la instrucción no acepta incógnita.

14 Error no se encontró el número en la cadena.
15 Error el modem no contesta.

INFORMACION QUE EL MODEM ENVIA A LA FUNCION QUE LO ACCESO

A continuación se describen comandos que solicitan información al modem y la forma en como se reportará ésta al programa que accesó a la función a través del vector apst.

COMANDO SINTAXIS

FL <ESC>FL ?<CR>

El parámetro de este comando es "?", envía el estado de fallas. El modem envía a la función un número en decimal, ésta se encarga de traducirlo a su equivalente en binario, en apst se guarda el valor binario que equivale a la falla y es accesado por el programa que llamó a la función que maneja al modem; por ejemplo; si el modem envía el 5208 en decimal, la rutina lo convierte a su equivalente en "binario" 0000101000011011000, esto significa que existen errores donde hay "unos" y son leídos de derecha a izquierda; por lo tanto hay errores del tipo : 3, 4, 6, 10 y 12 como se muestra en la siguiente tabla y los errores serán accesados por el programa que llamó a la función que maneja al modem a través del vector apst de la siguiente manera:

```
apst [0]  = 0,  apst [1]  = 0,  apst [2]  = 0,  apst [3]  = 1,
apst [4]  = 1,  apst [5]  = 0,  apst [6]  = 1,  apst [7]  = 0,
apst [8]  = 0,  apst [9]  = 0,  apst [10] = 1,  apst [11] = 0,
apst [12] = 1,  apst [13] = 0,  apst [14] = 0,  apst [15] = 0.
```

TABLA DE ERRORES DEL COMANDO FL

NOTA: apst [n] = 0 si no hay falla; apst [n] = 1 falla del tipo n; n tomará cualquier valor entre 0 y 31.

apst [0] = No detecta falla.

apst [1] = Error en la sintetización de la transmisión en la frecuencia intermedia TxIF.

apst [2] = Error en la sintetización de la recepción en la frecuencia intermedia RxIF.

apst [3] = Error en el tiempo VCXO en la transmisión de bit TxBIT.

apst [4] = Error en la recepción del control automático de ganancia.

apst [5] = Falla en el tiempo de amarre de lazo del bit del demodulador.

apst [6] = Falla en el amarre de lazo en la portadora del demodulador.

apst [7] = Falla de lazo en la sincronización del decodificador del demodulador.

apst [8] = Error en la adquisición del demodulador. apst [9] = Error en el rango de recepción de la portadora VCXO.

apst [10] = Error en el rango de transmisión de la portadora VCXO.

apst [11] = Error de bit en el rango de tiempo de recepción VCXO.

apst [12] = Falla en la memoria no volátil del demodulador.

apst [13] = Falla en el microprocesador del demodulador.

COMANDO SINTAXIS

RB <ESC>RB ?<CR>

DESCRIPCION

El parámetro de este comando es "?", el modem reporta la tasa de bits de error a través de este comando, para ser leido por el programa que lo solicitó.

COMANDO SINTAXIS

ST <ESC>ST ?<CR>

DESCRIPCION

Este comando pregunta al modem por los registros de estado de todas las fallas instantáneas. El modem envía a la función un número en decimal, ésta se encargará de traducirlo a su equivalente en binario y posteriormente al número de falla, de la misma manera que FL. En apst a partir de la posición 16 se reportan éstas fallas para ser leídas por el programa que accesó a ésta función. ST reporta fallas instantáneas, como lo indica la siguiente tabla :

TABLA DE ERRORES DEL COMANDO ST

apst [16] = No detecta falla.

apst [17] = Error en la sintetización de la transmisión en la frecuencia intermedia TxIF.

apst [18] = Error en la sintetización de la recepción en la frecuencia intermedia RxIF.

apst [19] = Error en el tiempo VCXO en la transmisión de bit TxBIT.

apst [20] = Error en la recepción del control automático de ganancia RxAGC.

apst [21] = Falla en el tiempo de amarre de lazo del bit del demodulador.

apst [22] = Falla en el amarre de lazo en la portadora del demodulador.

apst [23] = Falla de lazo en la sincronización del decodificador del demodulador.

apst [24] = Error en la adquisición del demodulador.

apst [25] = Error en el rango de recepción de la portadora VCXO.

apst [26] = Error en el rango de transmisión de la portadora VCXO.

apst [27] = Error de bit en el rango de tiempo de recepción VCXO.

PARAMETROS QUE CONFIGURAN AL MODEM :

Información enviada por la función para configurar al modem.

COMANDO SINTAXIS

AL <ESC>AL 1<CR>

Por default tiene el valor de 1, lo cual quiere decir que esta habilitado; reportará las fallas del modem a través del puerto serie RS-232.

COMANDO SINTAXIS

BE <ESC>BE 17<CR>

Por default tiene el valor 17, el parámetro que acepta es de dos dígitos, el primer número puede ser 1,2,4 ó 8 y el segundo puede estar en el rango de 3 a 8 inclusive; el primer número representa la mantisa del umbral de la tasa de los bits de error y el segundo número el exponente negativo; ejemplo si el valor del parámetro es 26 queda representado como 2x10^-6 ó .000002.

COMANDO SINTAXIS

BR <ESC>BR 1200<CR>

DESCRIPCION

Este comando por hardware esta configurado para que el puerto serie del modem opere a una velocidad de transmisión de 1200 bps., pero se puede modificar a los valores de 300 bps., 2400 bps., 4800 bps., o 9600 bps., en caso de que se requiera.

COMANDO SINTAXIS

CF <ESC>CF <CR>

DESCRIPCION

Este comando acepta un valor comprendido en el rango de 0 a 16 inclusive y el valor que se le envíe, corresponde al número de falla que borrará; si se le da el valor cero borrará el registro de todas las fallas activas.

apst [n] = 0 Borrará todas las fallas activas.

COMANDO SINTAXIS

EM <ESC>EM <CR>

DESCRIPCION

Habilita o deshabilita la salida del demodulador, dándole el valor de "1" o de "0" respectivamente; el modulador no puede habilitarse antes de establecer la frecuencia intermedia; el modem enviará al puerto serie mensaje de error cuando haya falla en la interfaz de datos o en el modulador.

COMANDO SINTAXIS

EE <ESC>EE 1<CR>

DESCRIPCION

Habilita o deshabilita el eco del modem; por default está habilitado y es conveniente no deshabilitarlo para asegurar que sea correcta la transferencia de información.

COMANDO SINTAXIS

LB <ESC>LB 0<CR>

DESCRIPCION

Comando que por default está deshabilitado, es decir no está en modo "loop back", para activarse se debe configurar con el valor 1.

COMANDO SINTAXIS

PC <ESC>PC 0<CR>

DESCRIPCION

Este comando por default está deshabilitado o lo que quiere decir que no hay transmisión de la portadora, si se requiere, se debe configurar con el valor 1.

COMANDO SINTAXIS

RE <ESC>RE<CR>

DESCRIPCION

Comando que no tiene default, al enviarse se ejecuta una acción de reset al modem o que es el equivalente a desenergizar y conectar al modem.

COMANDO SINTAXIS

TP <ESC>TP 100<CR>

Este comando tiene por default el valor de 100, que equivale a -10 dB y es la potencia de salida del demodulador.

Los parámetros de los comandos anteriores podrán ser desplegados uno a uno pero en el caso de nuestra aplicación los parámetros no se desplegarán directamente en pantalla, se guardarán en el vector (coparam) que será accesado por el programa que lo solicita a la función que accesa al modem.

coparam [0] = Recibe el parámetro de AL.

coparam [1] = Recibe el parámetro de BE.

coparam [2] = Recibe el parámetro de BR.

coparam [3] = Recibe el parámetro de CF.

coparam [4] = Recibe el parámetro de EE.

coparam [5] = Recibe el parámetro de EM.

coparam [5] = Recibe el parámetro de LB.

coparam [7] = Recibe el parámetro de PC.

coparam [8] = Recibe el parámetro de RB.

coparam [9] = Recibe el parámetro de TP.

DATOS QUE REQUIERE LA FUNCION DEL PROGRAMA QUE LA ACCESA :

Cuando el programa accesa a la función encargada del modem

Función funamod (inst,param,pto,acopara,astatus)

debe indicar en el parámetro "pto" el número de puerto correspondiente para establecer comunicación con el modem; en el parámetro "inst" el número de comando, y en el caso de que se requiera parámetro en "param".

En caso de instrucción no válida se indicará error.

Por ejemplo :

Si para establecer comunicación con el modem se utiliza el puerto 1 y se solicita la instrucción 3 que corresponde al comando BR (de velocidad de transmisión), se podrá modificar su valor (en caso de que se requiera) asignándolo a (param), en este ejemplo 9600 bps; por lo tanto se enviará :

```
pto con 1
inst con 3
param con 9600
vector apst
vector coparam
```

Con los valores de 1, 3 y 9600 la función interpreta que el 1 corresponde al número de puerto, que el 3 corresponde al comando BR y el 9600 a su parámetro, al enviar ésta información al modem, se programa para que opere a esa velocidad, a través del vector apst se solicita información al modem, coparam es el vector que guardará el valor de los parámetros; en este caso aunque se configure al modem, la función que lo accesa, requiere éstos dos vectores.

OPERACION DE LA FUNCION QUE INTERACTUA CON EL MODEM :

La función tiene los comandos y parámetros almacenados en forma matricial, llamada "matriz de comandos", con la estructura adecuada para ser transmitidos al modem, de tal forma que éste los pueda interpretar.

El modem requiere para la interpretación de los comandos que éstos sean enviados de acuerdo con el siguiente protocolo :

1) Espera recibir el caracter <ESC> (ASCII = 27) el cual le indica el inicio de comando.

2) A continuación espera dos letras (xx) que simbolizan al comando.

3) Si el comando requiere de un parámetro, el modem espera un caracter espacio o blanco (ASCII = 32) y a continuación el parámetro, siguiendo la misma filosofía que en el punto anterior. Si el comando no requiere del parámetro, pasa al siguiente punto (pto. 4).

4) Para indicarle fin de comando al modem espera recibir el (CR) "Carriage Return", <CR> (ASCII = 13).

Esto es :

<ESC>XX PARAMETRO<CR>

NOTA: Los caracteres ASCII anteriores son en decimal.

Ejemplo, para cambiar la velocidad de transmisión es:

<ESC>BR 9600<CR>

la rutina que accesa al modem le debe enviar la siguiente secuencia:

ESC
B
R

9
6
0
0
CR

El siguiente vector es el equivalente en ASCII del ejemplo; se puede observar que ésta columna pertenece al renglón de la instrucción 3 de la matriz ASCII.

```
27
66
82
32
57
54
48
48
13
```

La matriz, contiene la secuencia del comando y su correspondiente ASCII en decimal; El parámetro inst indica el número de comando; donde en la tabla aparece el caracter '#' (ASCII 35) se requiere enviar ahí el parámetro que corresponde a ese comando, a través del parámetro param. Para el caso del ejemplo anterior el parámetro correspondiente fue el ASCII en decimal del 9600 (57, 54, 48, 48).

MATRIZ DE COMANDOS

INST						
1	'ESC' (27)	'A' (65)	'L' (76)	' ' (32)	'#' (35)	'CR' (13)
2	'ESC' (27)	'B' (66)	'E' (69)	' ' (32)	'#' (35)	'CR' (13)
3	'ESC' (27)	'B' (66)	'R' (82)	' ' (32)	'#' (35)	'CR' (13)
4	'ESC' (27)	'C' (67)	'F' (70)	' ' (32)	'#' (35)	'CR' (13)
5	'ESC' (27)	'E' (69)	'E' (69)	' ' (32)	'#' (35)	'CR' (13)
6	'ESC' (27)	'E' (69)	'M' (77)	' ' (32)	'#' (35)	'CR' (13)
7	'ESC' (27)	'F' (70)	'L' (76)	' ' (32)	'#' (35)	'CR' (13)
8	'ESC' (27)	'L' (76)	'B' (66)	' ' (32)	'#' (35)	'CR' (13)
9	'ESC' (27)	'P' (80)	'C' (67)	' ' (32)	'#' (35)	'CR' (13)
10	'ESC' (27)	'R' (82)	'B' (66)	' ' (32)	'#' (35)	'CR' (13)
11	'ESC' (27)	'R' (82)	'E' (69)	'CR' (13)		
12	'ESC' (27)	'S' (83)	'T' (84)	' ' (32)	'#' (35)	'CR' (13)
13	'ESC' (27)	'T' (84)	'P' (80)	' ' (32)	'#' (35)	'CR' (13)

NOTA : EL comando 'RE' (inst=11) no requiere parámetro.

De la matriz, una vez definido el renglón correspondiente a la instrucción que se desea enviar al modem, se tiene que interactuar con las funciones encargadas del manejo del puerto serie de la PC.

ESTRUCTURA DE LA FUNCION QUE ACCESA AL MODEM :

Para poder comunicarse con el modem se puede hacer uso de la tarjeta de expansión de puertos ("Digiboard") o bien se puede hacer uso del puerto serie de la PC, se elige una u otra opción dependiendo de las necesidades.

Se tiene la función (funamod) que interactua con varias rutinas para accesar al modem, su estructura es la siguiente :

funamod (inst,param,pto,acoparam,astatus)

Requiere de la variable inst que equivale al número de instrucción la cual le indica que comando es el solicitado, requiere de param que equivale al número del parámetro, del vector acoparam que es donde se registra el valor del parámetro del comando solicitado y del vector astatus en el que se registran las fallas del comando FL y ST; las rutinas con las que interactua son las siguientes:

comunica
gfcm
abrept
cda
cierrapt
enviact
inscinc
instcpa
instspa
leeyesc
pren_eco
rebecpar
rec_flst
recíbect
valinst
verlong
verpar

El programa gfcm.lib es parte de una librería de (green leaf) un paquete de comunicaciones y comunica es la función encargada del manejo de puertos; ya sea COM1 que es el puerto serie de la PC o el manejo de cualquier puerto serie con expansión de tarjeta.

Las rutinas que se desarrollaron como parte de ésta tesis y que interactua con funamod se describen a continuación:

abrept (pto)

Esta rutina accesa a la función "abre", encargada de abrir el puerto de comunicación, en la variable pto se le debe indicar el número de puerto que se comunica con el modem, y se le debe especificar si es puerto de entrada y/o salida; para el caso del modem es puerto de entrada salida, lo que equivale a: "ASINOUT". El formato de la función abre es:

abre (pto,ASINOUT)

La rutina abrept dependiendo de la función abre, regresa 0 si no hubo error o los siguientes valores :

ERROR SIGNIFICADO

3 Parámetros de configuración del puerto inválidos.
4 Si el puerto ya estaba abierto.
5 No hay memoria suficiente en el buffer del puerto.
6 No esta el UART en la dirección especificada.
7 Inválido el tamaño del buffer que se requiere.
8 Problemas con los parámetros de interrupción.

cda (param,apvl,pp)

Esta rutina convierte un número decimal a su equivalente en ASCII. La variable param contiene el parámetro a ser codificado; apvl es un apuntador que contiene la cadena del comando, en esta cadena se va a insertar el parámetro que le corresponde; pp es un apuntador que indica la posición en la cadena del comando, a partir de la cual se va a insertar el ASCII correspondiente del parámetro. Regresa el número de caracteres ASCII que se necesitaron para representar al parámetro.

cierrapt (pto)

Esta rutina cierra el puerto utilizado durante la transmisión, detiene las interrupciones de entrada/salida y libera la memoria utilizada para los buffers de transmisión y recepción. Requiere de la variable pto que indica el número de puerto a cerrar. Regresará un 0 si el puerto se cerró sin problema y un 11 si no existe físicamente el puerto.

enviact (pto,apbuff,vl,EE)

Una vez convertida la información en caracteres ASCII, es enviada al modem en bytes a través de ésta rutina; pto es el puerto a donde está conectado el modem, apbuff es el apuntador que contiene la cadena a enviar al modem; vl indica el número de caracteres a enviar; EE representa el comando de eco, si es igual a uno el eco estará activado, si es igual a cero el eco estará desactivado; en nuestro caso está activado. Esta rutina tiene la capacidad de esperar respuesta del modem en un período de 10 segundos; esto preve que por alguna causa la función se quede en espera por tiempo indefinido. Regresa un 0 si no hubo error en el número de bytes transmitidos o los siguientes errores :

ERROR SIGNIFICADO

2 Error en el eco.

12 El modem no regresa en el eco el valor correspondiente

15 El modem no contesta, la rutina espera 10 segundos a que conteste de lo contrario envía este error.

inscinc (inst,ap)

Esta rutina codifica instrucciones con signo de interrogación y es utilizada cuando se quiere saber la configuración de algún comando, o en el caso de que se pregunte por el registro de errores o por la lectura de la tasa de error de bit. Requiere de la variable inst, que indica el número de instrucción y de ap, apuntador que de acuerdo a la instrucción indica la cadena a formar; regresa en vi la longitud de la cadena.

instcpa (inst,param,ap)

Esta rutina codifica instrucciones con parámetro; inst corresponde a la instrucción, param al parámetro y ap al apuntador a la cadena que tiene la instrucción ya codificada; accesa a la función "cda", que convierte el parámetro a su equivalente ASCII correspondiente para ser enviado al modem. Esta rutina regresa la longitud de la cadena.

instspa (inst,ap)

La rutina instspa codifica instrucciones sin parámetro, requiere de la variable inst que indica el número de instrucción, ésta rutina es utilizada en casos muy especiales, como por ejemplo en el comando que reestablece al sistema; ap corresponde al apuntador de la cadena que tiene codificada la instrucción. Esta rutina regresa la longitud de la cadena.

leeyesc (pto)

Esta rutina busca el archivo de configuración del puerto de comunicaciones de acuerdo con las características del modem, llamado "confmod.prt", si no lo encuentra lo crea; siguiendo la estructura del archivo de configuración "config.prt". En pto se le indica el número de puerto donde esta conectado el modem. Regresa el valor de 0 si los parámetros de configuración del puerto son válidos y 3 si no lo son.

pren_eco (pto)

Esta rutina habilita la función de ECO en el modem, en la variable pto se le indica el número de puerto donde está conectado el modem. Regresa el valor de 0 si envía y recibe adecuadamente la información y el valor correspondiente en f para el caso de que haya error al abrir el puerto o al leer la configuración.

rebecpar (apbuff,nc)

Esta rutina saca un parámetro entero de una cadena; apbuff es un apuntador a la cadena recibida; nc equivale al número de caracteres de la cadena recibida. Regresa en param el parámetro que encontró en la cadena y el error 14 si no encontró el número en la cadena.

rec_flst (apbuff,appst,nc,i)

Esta rutina recibe los errores de los comandos FL y ST para almacenarlos en appst que es un apuntador a un vector donde se indican los errores contenidos; en las primeras 16 localidades aparecen los errores del comando FL y en las 16 restantes los del comando ST; apbuff es un apuntador de la cadena y se va a interpretar el número que contiene para indicar los errores ocurridos; nc contendrá el número de caracteres de la cadena donde va a buscar el número a interpretar; i es quien indica si se trata de FL o ST con i=0 e i=15 respectivamente. Regresa 0 si no hubo error.

recibect (inst,pto,apcopar,apst)

Es una rutina que recibe información del modem, dándole un tiempo de 10 segundos para que conteste; requiere de la variable inst que indica el número de instrucción, de pto que indica el número de puerto donde está conectado el modem; en el apuntador apcopar guarda el parámetro del comando solicitado, donde AL está en la localidad 0, BE en 1, BR en 2, CF en 3, EE en 4, EM en 5, LB en 6, PC en 7, RB en 8, RE en 9 y TP en 10; en el apuntador apst se almacenan los valores de FL en las primeras 16 localidades y en las 16 restantes los valores de ST. Regresa el valor de 0 si no hubo error, un 14 si no encuentra el número en la cadena y 15 si el modem no contesta.

valinst (inst)

Rutina que verifica que la instrucción exista; requiere de la variable inst que equivale al número de instrucción, cuyos valores son del 1 al 13 y corresponden a los comandos del modem. En caso de que la instrucción exista regresa un 0, en caso contrario regresa 1.

verlong (pto,vl)

Esta rutina verifica el espacio disponible en el buffer de transmisión; la variable pto indica el número de puerto donde se tiene conectado el modem y vl indica la longitud de la cadena a enviar. Regresa el valor de 0 si hay espacio disponible en el buffer de transmisión y 9 en caso contrario.

verpar (inst,param)

Esta rutina verifica que el parámetro de la instrucción esté en el rango especificado; requiere de la variable inst que indica el número de instrucción y de param que indica el parámetro a verificar. Regresa 0 si el parámetro es válido y 3 en caso contrario.

El seudocódigo de esta función se encuentra en el apéndice C.

4.3 ESPECIFICACION DEL PROGRAMA QUE CONTROLA AL MODEM

DESCRIPCION DEL PROGRAMA QUE ACCESA A LA FUNCION QUE CONTROLA AL MODEM.

Este programa simula la interfaz hombre máquina, accesa a la función que interactua con el modem.

Al correr el programa, aparecen en pantalla los datos que requiere y algunos dependen del comando solicitado; estos datos son :

a) El número de puerto donde está conectado el modem.

b) El número de instrucción que indica el comando a enviar.

c) Primera opción, si el parámetro a enviar es un número.

d) Segunda opción, si el parámetro a enviar es la incógnita.

En el caso de los comandos que se ejecutan directamente, no se pregunta ninguna de las dos opciones anteriores, uno de éstos comandos es el reset al modem.

El seudocódigo de este programa está en el apéndice D.

4.4 ESPECIFICACION DEL SIMULADOR DEL MODEM

DESCRIPCION DEL PROGRAMA DE PRUEBA QUE SIMULA EL FUNCIONAMIENTO DEL MODEM

El programa que simula al modem, tiene el propósito de simular su funcionamiento y de este modo, poder probar a la función que accesa y programa al modem a través del programa que accesa a dicha función.

Este programa que simula al modem, tiene almacenados los comandos incluyendo a los que tienen parámetro con valor inicial (de default); el programa contempla la posibilidad de que esos valores puedan ser modificados y de volver a tomar su valor inicial al dar "reset".

El seudocódigo de este programa se encuentra en el apéndice E.

4.5 TEORIA DE OPERACION

En la estructura de la función que accesa al modem, se incluyen todas las rutinas que la integran y su codificación está en lenguaje de programación "C".

Para poder probar la función que accesa al modem, se tiene un programa que accesa a esta función también en lenguaje "C".

Se desarrolló otro programa también en el mismo lenguaje, el cual debe estar en otra PC. para poder simular el funcionamiento del modem. este programa debe incluir los siguientes módulos :

abrept
comunica
gfcm

El programa "comunica" contiene el código de las rutinas que llevan a cabo la comunicación, incluyendo el programa "abrept.c" que se encarga de abrir el puerto de comunicación y "gfcm" que forma parte de una librería de un paquete de comunicaciones llamado "green leaf".

El programa que accesa al modem debe incluir las siguientes rutinas:

comunica
gfcm
abrept
cda
cierrapt
enviact
funamod
inscinc
instcpa
instspa
leeyesc
pren_eco
rebecpar
rec_flst
recibect
valinst
verlong
verpar

NOTA :

Los programas en lenguaje "C" fueron hechos con el compilador QuickC versión 1.01 de Microsoft, la versión del sistema operativo 3.3 y el puerto utilizado para hacer las pruebas fue el puerto serie de la PC (COM 1).

5 CONCLUSIONES, RECOMENDACIONES Y BIBLIOGRAFIA

En el diseño del sistema presentado, para cumplir con el objetivo de Configurar y Controlar al Modem de Radio Frecuencia, una de las actividades centrales fue el análisis a fondo de la operación del propio Modem, detallando su protocolo de comunicación así como la sintaxis y propósito de cada uno de sus comandos. Las tareas de análisis descritas, constituyen la base de diseño del controlador de comunicaciones objeto de la presente tesis. A continuación se detallan los resultados y experiencias mas relevantes, así como aquellos aspectos que se consideran de interés para futuros desarrollos a lo largo de esta línea de trabajo.

El sistema desarrollado, realiza la configuración y control del Modem en forma interactiva y sencilla utilizando una computadora personal; permite su empleo en cualquier otra aplicación donde se requiera modificar la configuración o controlar la operación de un Modem con las características de acceso remoto del CM-101; las estructuras de datos permiten al usuario la flexibilidad necesaria para adaptarlo a sus necesidades independientemente del protocolo de comunicación, o de la sintaxis y estructura de los comandos de operación del Modem.

Adicionalmente, para propósitos de prueba, se implementó una simulación del Modem utilizando la potencialidad del "software" resultado de la aplicación de una metodología "top-down" para el diseño, lo cual permitió manejar programas independientes, con la posibilidad de poderlos interrelacionar para cumplir con un objetivo específico.

Como resultado general de la elaboración de esta tesis, se tiene la validación de recomendaciones metodológicas generalmente aceptadas; en particular, se pueden mencionar tres aspectos determinantes para el desarrollo de sistemas que involucran tanto "hardware" como "software" y su interacción con el usuario:

1) Conocer a fondo el problema a resolver, los elementos disponibles para la solución de éste y las restricciones existentes.

2) Emplear una metodología de programación adecuada, independientemente del lenguaje que se utilice, esto da herramientas para poder desarrollar nuestras ideas en la computadora, además facilita la realización del programa.

3) Conocer a fondo el lenguaje de programación a utilizar, ya que solo así se pueden seleccionar las alternativas mas adecuadas para la realización de las funciones que conforman el "software" del sistema.

En la concepción, desarrollo y aplicación del sistema se ven involucrados los siguientes tres elementos :

- Cliente, que es quien tiene la necesidad y en consecuencia está dispuesto a pagar por el desarrollo del sistema. Se preocupa porque el sistema se adquiera a bajo costo, porque incremente la productividad, por la confiabilidad, flexibilidad y eficiencia del sistema.

- Usuario, que es quien interactúa directamente con el "hardware" y el "software", para la realización de tareas específicas, independientes de la operación interna del sistema. Se preocupa porque el sistema sea funcional, confiable, eficiente, y fácil de operar.

- Mantenimiento, que es quien se encarga de la corrección de errores del sistema, y también en la mayoria de los casos, de las modificaciones y mejoramiento del mismo. Se preocupa porque el sistema no tenga errores, porque tenga una buena documentación y un buen diseño.

Se pudo constatar que el resultado de la correcta aplicación de esta metodología, es un sistema que presenta las siguientes características:

- Confiabilidad, ya que las pruebas al sistema pueden ser exhaustivas y en consecuencia la obtención de los resultados esperados puede garantizarse bajo cualquier condición de operación.

- Flexibilidad, ya que puede tener otras aplicaciones en donde se requiera manejar un Modem con características similares.

- Facilidad de operación, porque por una parte, permite seleccionar el comando que se desea enviar al Modem, así como sus parámetros a través de menús y por otra parte, el programa que accesa al Modem, para ser configurado y controlado, maneja "nemónicos" que permiten recordar de que parámetro se esta hablando.

- Buena documentación, el sistema está ampliamente documentado tanto en los aspecto de definición y diseño, como en los de implementación y operación.

B I B L I O G R A F I A

1. Simon Haykin McMaster University. Communication Systems. Copyright Under The International Copyright Union. Interamericana, 2nd Printing 1985.

2. Ahuja, Vijay. Design and Analysis of Computer Comunication Networks. McGraw-Hill Book Co-Singapore. 2nd Printing 1986.

3. John E. McNamara. Technical Aspects of Data Communication. Digital Equipment Corporation, Bedford, Mass. 2nd Printing 1982.

4. Nestor González Sainz. Comunicaciones y Redes de Procesamiento de Datos. McGraw-Hill 1a. Edición 1987.

5. A. Alabau. Teleinformática y Redes de Computadores. Marcombo 2a. Edición 1987.

6. José Mompín Poblet. Interconexión de Periféricos a Microprocesa_ dores. Marcombo 1983.

7. Thomas H. Naylor. The Design of Computer Simulation Experiments. Duke University Press Durham, N.C. 1969

8. Claude Mc. Millan & Richard F. González. Análisis de Sistemas. Trillas 2da. Reimpresión 1984.

9. J. Chistopher Jones. Design Methods. Seeds of Human Futures. John Wiley & Sons Ltd, Inglaterra 2nd. Printing 1970.

10. Francois E. Cellier. Progressin Modelling and Simulation. Academic Press INC (London) 1982. The Swis Federal Institute of Technology.

11. Manual del Modem. Operation and Instalation, PSK Digital Modem CM101, ComStream. San Diego CA.

12. Manuales del compilador de "C" Microsoft C 5.1

6 APENDICE A

6.1 CONFIGURACION DE LA RED

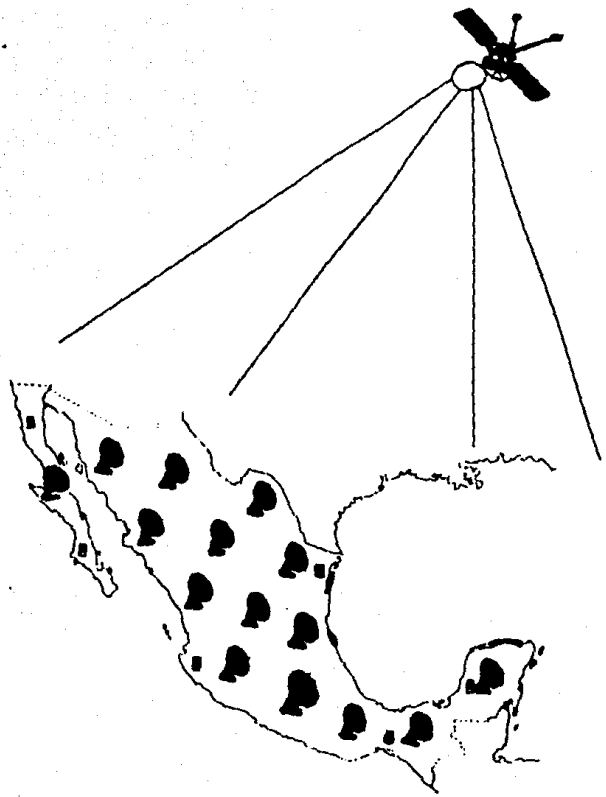
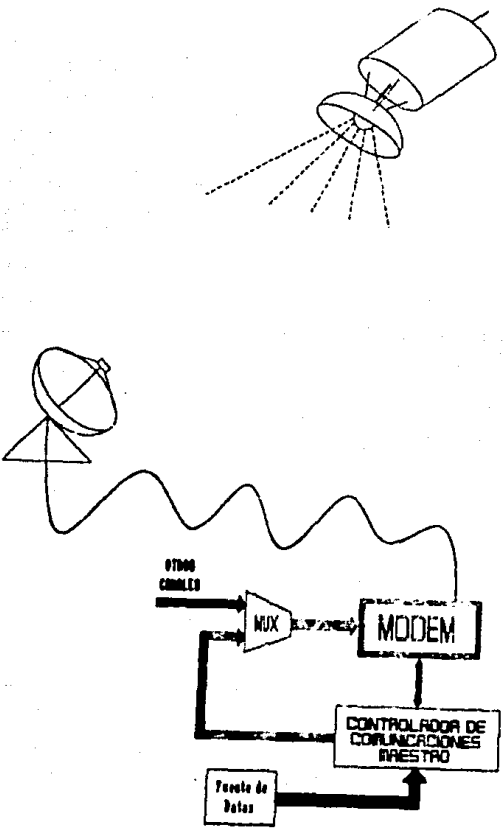


DIAGRAMA DE BLOQUES DEL SISTEMA DE COMUNICACION.



6.2 PROTOCOLO DE COMUNICACION

Se describen las características generales del protocolo que define las interacciones entre la Estación Maestra y las Estaciones Remotas; la descripción se efectúa conforme al modelo ISO ("International Standar Organization") Organización Internacional de Estandarización.

ESTACION MAESTRA A ESTACION REMOTA.

En este caso se trata de la interacción entre dos Controladores de Comunicaciones.

1.- ESTRATO FISICO :

Conforme al estandard RS-232 C. de la EIA.

2.- ESTRATO DE ENLACE :

Protocolo orientado a bit, empleando campos de control con significado por posición.

Sincronización de los bloques y delimitación de los mensajes, mediante patrón de inicio y cuenta de bytes.

Control de errores :

A través del empleo de un esquema para detección de errores mediante verificación de Código Cíclico Redundante (CRC) de dos bytes. La corrección de errores se realiza mediante envío continuo y retransmisión no selectiva del tipo ventana deslizante "sliding window".

Coordinación de la comunicación :

Mediante conmutación de circuito físico (canal satelital); acceso al canal a través de encuesta directa "Roll Call Polling". El establecimiento y la desconexión del enlace están centralizados en la Estación Maestra.

Recuperación de fallas :

A través de solicitudes de retransmisión con número máximo de intentos por sesión y tiempos de espera acotados a nivel de mensaje, paquete y caracter. Bajo cualquier condición de falla la estación remota seleccionada apaga su transmisor.

Control de flujo :

Mediante la utilización de ventanas y bloques del tipo RNR "Receiver Not Ready" Receptor no listo.

3.- ESTRATO DE LA RED :

Conceptualmente la Red es una estrella desde el punto de vista del flujo de las Estaciones Remotas a la Estación Maestra y un Bus desde el punto de vista de la Estación Maestra a las Estaciones Remotas.

Establecimiento de enlaces :

Del tipo punto a punto entre la Estación Maestra y las Remotas, en secuencias preasignadas por el usuario desde la propia Estación Maestra. Se cuenta con medios para el direccionamiento tipo difusión "Broadcast"

Control de flujo :

En este nivel, se efectúa dependiendo del servicio a suministrar, sea por la terminación de la lista de transacciones o por terminación de un intervalo máximo preestablecido por el usuario y fin de la transacción en proceso.

4.- ESTRATO DE TRANSPORTE :

Cada estación remota responde a dos direcciones, una de ellas propia de la estación y otra para recepción de mensajes globales en servicios de difusión.

Establecimiento de la sesión :

A cargo de la Estación Maestra.

Realización de Transacciones :

Entre la Estación Maestra y cualquier Estación Remota una a la vez.

Difusión de información :

Simultáneamente a todas las Estaciones Maestras, con verificación individual de la información transmitida, en la siguiente sesión individual que se establezca.

5.- ESTRATO DE SESION :

Dependerá del tipo de servicio que se solicite a la Red.

Transferencia de archivos :

Full Duplex.

Ejecución de comandos :

Half Duplex.

Correo Electrónico :

Half Duplex con utilización de buzones residentes en la Estación Maestra.

6.- ESTRATO DE PRESENTACION :

Código ASCII en lo general, excepto los archivos que serán considerados como binarios.

7.- ESTRATO DE APLICACION :

El protocolo soporta la transferencia de archivos entre dos nodos cualesquiera.

CONTROLADOR DE COMUNICACIONES A MODEM.

La función básica de este protocolo, es la carga de parámetros al modem y la obtención de datos del estado operativo del mismo.

1.- ESTRATO FISICO :

Conforme al estandard RS-232 C. de la EIA.

2.- ESTRATO DE ENLACE :

Protocolo orientado a caracter.

Control de errores :

Verificación de señal de eco.

Recuperación de fallas :

A través de retransmisión

3.- ESTRATO DE LA RED :

Se trata de una conexión punto a punto.

4.- ESTRATO DE TRANSPORTE :

Establecimiento y término de la sesión a cargo del Controlador de Comunicaciones.

5.- ESTRATO DE SESION :

En modo Half Duplex.

6.- ESTRATO DE PRESENTACION :

Código ASCII.

7.- ESTRATO DE APLICACION :

Envío de caracteres para programación del modem.

CONTROLADOR DE COMUNICACIONES A FUENTE DE DATOS.

1.- ESTRATO FISICO :

Conforme al estandard RS-232 C. de la EIA.

2.- ESTRATO DE ENLACE :

Protocolo orientado a bit, empleando campos de control con significado por posición.

Sincronización de los bloques y delimitación de los mensajes, mediante patrón de inicio y cuenta de bytes.

Control de errores : a través del empleo de un esquema para detección de errores mediante verificación de Código Cíclico Redundante (CRC) de dos bytes. La corrección de errores se realizará mediante retransmisión dentro de un esquema avanza y para "start-stop".

Recuperación de fallas :

A través de solicitudes de retransmisión con número máximo de intentos por sesión y tiempos de espera acotados a nivel de mensaje, paquete y caracter.

Control de flujo :

Mediante la utilización de ventanas y bloques del tipo RNR "Receiver Not Ready" Receptor no listo.

3.- ESTRATO DE LA RED :

Se trata de un enlace punto a punto.

4.- ESTRATO DE TRANSPORTE :

Establecimiento de sesión a cargo de la Estación Maestra.

5.- ESTRATO DE SESION :

De acuerdo al tipo de servicio solicitado a la Red.

Transferencia de archivos :

Half Duplex.

Ejecución de comandos :

Half Duplex.

6.- ESTRATO DE PRESENTACION :

Código ASCII en lo general, excepto los archivos que serán considerados como binarios.

7.- ESTRATO DE APLICACION :

El protocolo soporta los siguientes servicios :

a) Transferencia de archivos.

b) Ejecución remota de comandos en los Controladores de Comunicaciones.

6.3 MODOS DE OPERACION

La programación residente en el Controlador de Comunicaciones soporta los siguientes modos de operación :

1) ENLACE COMPARTIDO :

En este modo, el Controlador de Comunicaciones realiza dos funciones, la primera consiste en el establecimiento automático o manual de un enlace punto a punto entre la Estación Maestra y la Remota seleccionada para la comunicación entre las fuentes de datos respectivas, a través de enlaces síncronos o para canalizar el flujo de señales binarias que representan información de voz o facsímil; la segunda contempla un modo de interacción opcional, en este caso, se establece un canal para la comunicación de archivos de y hacia la Estación Maestra, a través de los Controladores de Comunicaciones, para la transferencia de archivos de datos y para la realización de los servicios de correo electrónico. El tipo de servicio se determina en el diálogo inicial, entre la Estación Maestra y la Remota correspondiente al tiempo del establecimiento del enlace; a continuación se presenta el servicio descrito en función del tipo de información a transmitir.

a) DATOS.

Enlace punto a punto entre Estación Maestra y Estación Remota, con duración típica de 3 min. condicionado al proceso de terminación del enlace y la finalización de la lista de transacciones o al término del intervalo típico y la terminación de la transacción en proceso.

Se puede proporcionar opcionalmente el servicio de almacenamiento temporal de archivos en el Controlador de Comunicaciones, para archivos que en total no excedan de 9 Mbytes. El orden de acceso al canal es mediante asignación a cargo de la Estación Maestra, según una lista preestablecida. El tiempo asignado al enlace se reduce a un minuto en los siguientes casos : la Estación Remota no contesta, la Estación Remota contesta con indicación de no tráfico.

b) VOZ Y FAX.

Enlaces punto a punto entre Estación Maestra y Remota, con duración máxima de 6 min., en este caso no se tiene acceso a la función de almacenamiento temporal de información en el Controlador de Comunicaciones. El orden de acceso al canal es mediante asignación a cargo de la Estación Maestra según una lista preestablecida. El tiempo asignado al enlace se reduce a 1 min. en los siguientes casos: la Estación Remota no contesta. la Estación Remota contesta con indicación de no tráfico.

2) ENLACE DEDICADO :

En este modo, la Estación Maestra establece un enlace de duración indefinida con cualquiera de las Estaciones Remotas. En este modo de operación, la información no circula a través del controlador de Comunicaciones, restringiéndose sus funciones a la transmisión o recepción y ejecución de comandos según se trate de la Estación Maestra o Remota.

3) DIFUSION DE INFORMACION :

En este modo de operación, tiene como propósito el difundir información "Broadcast", por el tipo de archivo de datos, por parámetros del estado de la Red o por comandos de operación genéricos de las Estaciones Remotas.

Se interrogarán secuencialmente a las Estaciones Remotas para determinar si la recepción fue libre de errores; en caso de error, se desplegará la información correspondiente en la Estación Maestra.

4) ADMINISTRACION DE LA RED :

Para facilitar la operación de la Red, se incluyeron funciones de administración y despliegue de información relevante a nivel tanto de Estaciones Remotas como de la Maestra.

a) ESTACION MAESTRA.

Determina el modo de operación de la Red, mantiene un archivo con la bitácora diaria de operación en la que se guardan : comandos, modificación de estados y en su caso, los mensajes correspondientes a errores de operación o comunicación. Para diagnosticar la calidad del enlace "Bit Error Test", se cuenta con la facilidad de establecer lazos "Loop back" con cualquier Estación Remota.

b) ESTACIONES REMOTAS.

Comunican a la Estación Maestra, como parte del diálogo inicial sus necesidades de servicio, con la finalidad de que el operador de la Red pueda tomar las decisiones correspondientes, desplegando de manera permanente el estado de la Red.

7 APENDICE B

7.1 TIPOS DE TRANSACCIONES

Existen los siguientes tipos de transacciones:

COMANDOS :

Consisten en enviar un mensaje que contiene un comando, como puede ser la restauración de estadísticas en un Controlador Remoto o la acción que se desea ejecutar. La respuesta a un comando es el resultado de la operación sin que esto contenga información adicional al código de terminación.

SOLICITUD :

Demanda la ejecución de un comando en el Controlador Remoto y debe responder con información al Controlador Maestro. En la respuesta se incluye un código de terminación como parte del mensaje, en adición a la información que se tiene de respuesta.

CONSULTA :

Solicita la lista de transacciones pendientes de un Controlador Remoto. El control de la secuencia de transferencia corresponde al Controlador Maestro, por lo que al establecer un enlace, se solicita la lista de transacciones que se tienen que realizar en el Controlador Remoto. Esta lista contiene los mensajes, archivos y solicitudes del Controlador Remoto.

COMANDOS Y DATOS :

(Opcional) Transferencia de un archivo. El comando contiene la información de fuente y destino del archivo y los datos corresponden al contenido del archivo a transferir.

MENSAJES :

Envío de un mensaje de Controlador a Controlador de Comunicaciones. Los mensajes son administrados por los Controladores. Se emplean como un medio de operación entre los operadores de cada uno de los centros de cómputo interconectados.

CORREO :

(Opcional) Envío de un mensaje de correo electrónico de Controlador a Controlador. Es un medio para extender las funciones de correo electrónico existente en los nodos interconectados por la Red.

7.2 UTILERIAS DEL MODO SERVICIO (CONTROLADOR MAESTRO)

Utilerías del modo Servicio del Controlador Maestro :

a) Configuración.

Permite la modificación parámetros de operación que tiene cada uno de los puertos de comunicación serie, los parámetros son : el puerto, la velocidad, la paridad, los bits de información y el o los bits de paro "stop". Al aceptar los cambios, se valida la información, si no acepta la configuración, se presenta mensaje de error.

b) Parámetros del Modem.

Presenta la pantalla de configuración del modem del Controlador maestro o remoto, si hubiera error se presenta un mensaje de error y la operación se cancela. La información actualizada, no se almacena automáticamente en disco, lo que permite mantener parámetros en forma temporal durante una sesión de prueba.

c) Configuración de Temporizadores.

Permite la modificación de las constantes empleadas para medir el tiempo de detección de errores y para fijar el tamaño de las ventanas de recepción.

d) Archivos.

En el manejo de archivos se tienen las siguientes opciones :

1) LEER.

Emplea un archivo que contiene la configuración del Controlador de comunicaciones; el archivo puede ser modificado por el menú de servicio.

La información del archivo es utilizada para :

Configurar los puertos de comunicaciones.

Fijar las direcciones de la estación remota.

Establecer la secuencia de barrido en la estación maestra.

Con el comando LEER, se lee de disco la configuración del modem conectado al satélite.

El archivo de configuración contiene la secuencia de cada remota a ser interrogada; una remota puede aparecer cuantas veces se requiera en la lista de barrido, la lista de barrido tiene espacio para 50 elementos.

2) ESCRIBIR.

Escribe en un archivo los parámetros actuales de configuración del puerto serie de comunicación del Controlador.

3) INSTALAR.

Comando que envía al modem los parámetros. En el caso de los Controladores remotos, la instalación de la información se puede en forma local desde el propio Controlador o remotamente desde el Controlador maestro.

4) DIRECTORIO ACTUAL.

El comando solicita la trayectoria (path) del directorio. Fija el directorio presente dentro del sistema operativo, para almacenar y leer los archivos de configuración.

e) Restauración de estadística Local.

Comando que restaura la información estadística contenida en memoria del Controlador de comunicaciones, registrándose en bitácora, hora y fecha en la que se borró la estadística.

f) Generación de Patrones de Prueba.

Con fines de prueba, se tiene la capacidad de enviar mensajes programados que permiten verificar la operación del Sistema de Comunicación de Controlador a Controlador.

Los patrones son capaces de determinar en cual de los extremos del satélite existe problema.

g) Secuencia de barrido.

El comando permite definir la secuencia de barrido de los Controladores remotos. La secuencia de barrido forma parte del archivo de configuración.

h) Nombre de los Controladores remotos.

Comando que permite definir un nombre lógico a cada controlador de comunicación.

7.3 INTERFACE HOMBRE MAQUINA

DESPLIEGUE DEL CONTROLADOR :

Los comandos a cargo del Controlador Maestro son :

1) ZONA DE MENUS.

La zona de menús presenta una lista de los comandos disponibles a cada nivel del árbol de comandos del Controlador de Comunicaciones.

2) ZONA DE MODO DE OPERACION.

El modo de operación presenta el modo actual de operación que puede ser automático, manual, paro o servicio.

3) INDICADOR DE ERROR.

Presenta un mensaje con el último error ocurrido.

4) ACCIONES EN PROCESO CON LA COMPUTADORA MAESTRA.

Presenta mensajes que indican la actividad de los canales de comunicación entre el Controlador y la Computadora Maestra.

5) ACCIONES EN PROCESO CON EL SATELITE.

Presenta mensajes que indican la actividad de los canales de comunicación entre el Controlador de Comunicaciones y el equipo de comunicaciones con el satélite.

6) ZONA DE OPERACION DE COMANDOS.

Es una área libre para la operación de cada uno de los comandos y pantallas de despliegue del Controlador de Comunicaciones.

7.4 UTILERIAS DEL MODO SERVICIO (CONTROLADOR REMOTO)

Utilerías de Servicio del Controlador Remoto :

a) Configuración.

Permite la modificación de los parámetros de operación que tiene cada uno de los canales de comunicación serie. Los parámetros de configuración son : el puerto, la velocidad, la paridad, los bits de información y el o los bits de paro "stop"

b) Parámetros Modem Local.

La información actualizada, no se almacena automáticamente en disco, lo que permite mantener parámetros en forma temporal durante una sesión de prueba.

c) Configuración de Temporizadores.

En esta sección se permite la modificación de las constantes empleadas para medir el tiempo de detección de errores.

d) Archivos.

Para manejo de archivos se tienen las siguientes opciones :

1) LEER.

Se emplea un archivo que contiene la configuración del Controlador de Comunicaciones; el archivo puede ser modificado por medio del menú de servicio.

2) ESCRIBIR.

Escribe en un archivo los parámetros actuales de configuración del puerto de comunicación serie del Controlador.

3) INSTALAR.

Comando que envía al modem los parámetros .

4) DIRECTORIO ACTUAL.

El comando solicita la trayectoria del directorio. Fija el directorio presente dentro del sistema operativo, para almacenar y leer los archivos de configuración.

e) **Restauración de estadística Local.**

Comando que restaura la información estadística contenida en la memoria del Controlador de Comunicaciones, registrandose en bitácora, hora y fecha en la que se borró la estadística.

f) **Generación de Patrones de Prueba.**

Con fines de prueba, se tiene la capacidad de enviar mensajes programados que permiten verificar la operación del Sistema de Comunicación de Controlador a Controlador.

Los patrones son respondidos con información preestablecida, de manera que se puede saber en cual de los extremos del satélite existe problema.

8 APENDICE C

8.1 SEUDOCODIGO DE LA FUNCION QUE ACCESA AL MODEM

Seudocódigo de la función que maneja al modem.

MATRIZ DE COMANDOS

INST						
1	'ESC' (27)	'A' (65)	'L' (76)	' ' (32)	'#' (35)	'CR' (13)
2	'ESC' (27)	'B' (66)	'E' (69)	' ' (32)	'#' (35)	'CR' (13)
3	'ESC' (27)	'B' (66)	'R' (82)	' ' (32)	'#' (35)	'CR' (13)
4	'ESC' (27)	'C' (67)	'F' (70)	' ' (32)	'#' (35)	'CR' (13)
5	'ESC' (27)	'E' (69)	'E' (69)	' ' (32)	'#' (35)	'CR' (13)
6	'ESC' (27)	'E' (69)	'M' (77)	' ' (32)	'#' (35)	'CR' (13)
7	'ESC' (27)	'F' (70)	'L' (76)	' ' (32)	'#' (35)	'CR' (13)
8	'ESC' (27)	'L' (76)	'B' (66)	' ' (32)	'#' (35)	'CR' (13)
9	'ESC' (27)	'P' (80)	'C' (67)	' ' (32)	'#' (35)	'CR' (13)
10	'ESC' (27)	'R' (82)	'B' (66)	' ' (32)	'#' (35)	'CR' (13)
11	'ESC' (27)	'R' (82)	'E' (69)	'CR' (13)		
12	'ESC' (27)	'S' (83)	'T' (84)	' ' (32)	'#' (35)	'CR' (13)
13	'ESC' (27)	'T' (84)	'P' (80)	' ' (32)	'#' (35)	'CR' (13)

NOTA : EL comando 'RE' no requiere parámetro.

```
/*  Función funamod (inst,param,pto,acopara,astatus) */
/*  Esta función es la principal y está estructurada de
```

```
      tal forma que va a interactuar con otras rutinas. */

      f = valinst (inst)
      IF (f = 0) THEN
         f = pren_eco (pto)
       IF (f = 0) THEN
        EN CASO DE QUE inst = 1,2,3,4,5,6,7,8,9,10,12,13:
/*        Instrucciones con parámetro. */
          IF (param =/= 63) THEN
/*           El 63 equivale a la incógnita. */
             vp = verpar (inst,param)
           IF (vp =/= 0) THEN
              cierra (pto)
/*            return (3) */
/*            Parámetro de configuración no válido. */
           ENDIF
             vl = instcpa (inst,param,buff)
/*                Verifica que la instrucción sea con
                  parámetro. */
              f = verlon (vl)
/*                Verifica la longitud en el buffer de
                  transmisión. */
            IF (f = 0) THEN
               f = enviact (pto,buff,vl,EE)
/*                Envía información. */
             IF (f = 0) THEN
                cierra (pto)
/*              Enviada la información se cierra el
                puerto. */
              IF (f = 0) THEN
                 return (0)
              ELSE
                 return (f)
              ENDIF
             ELSE
               return (f)
             ENDIF
            ELSE
              return (f)
             ENDIF
          ELSE
            IF (inst = 4) THEN
               return (13)
/*             Error la instrucción no acepta incógnita. */
            ELSE
             vl = inscinc (inst,buff)
/*                Ejecuta instrucciones con incógnita. */
              f = verlon (vl)
             IF (f = 0) THEN
                enviact (pto,buff,vl,EE)
              IF (f = 0) THEN
                 error = 0
              ELSE
                 error = f
              ENDIF
             ELSE
```

```
                error = f
              ENDIF
            ENDIF
          ENDCASE
          EN CASO DE QUE inst = 11:
/*          Instrucción sin parámetro, el caso del reset. */
            vl = instspa (inst,buff)
             f = verlon (vl)
            IF (f = 0) THEN
               f = envict (pto, buff,vl,EE)
             IF (f = 0)
                f = cierrapt (pto)
              IF (f = 0) THEN
                 return (0)
              ELSE
                return (f)
              ENDIF
             ELSE
               return (f)
             ENDIF
        ELSE
          IF (param = 63 AND error = 0) THEN
/*          Si el parámetro es incógnita. */
            f = recibect (inst,pto,acoparam,astatus)
           IF (f = 0) THEN
              f = cierrapt (pto)
            IF (f = 0) THEN
               return (0)
            ELSE
               return (f)
            ENDIF
           ELSE
             return (f)
           ENDIF
          ELSE
            return (f)
        ENDIF
      ENDCASE
     ELSE
       cierra (pto)
       return (f)
/*     En f regresa el valor correspondiente a prender eco. */
     ENDIF
    ELSE
      return (f)
/*    En f regresa el error que corresponde a instrucción
      no válida. */
    ENDIF
    END
```

```
/*   Rutina abrept (pto) */

/*   Esta rutina  requiere  del parámetro pto,  el cual  debe ser
     especificado por el programa que accesó a la función  modem,
     indicando el número de  puerto de la  pc o de la  tarjeta de
     "Digibord" que será utilizado para manejo del modem. */

    fp = abrept (pto,ASINOUT)
     IF fp = ASSUCCESS THEN
        return(0)

/*      No hubo error al abrir el puerto. */

      ELSE
       IF fp = ASINVPORT THEN
          return(3)

/*        Parámetros de configuración del puerto inválidos */

        ELSE
         IF fp = ASINUSE THEN
            return(4)

/*        El puerto ya estaba abierto. */

         ELSE
          IF fp = ASNOMEMORY THEN
             return(5)

/*         No hay memoria suficiente en el buffer del puerto. */

          ELSE
           IF fp = ASNO8250 THEN
              return(6)

/*          No está el UART en la dirección especificada. */

           ELSE
            IF fp = ASINVBUFSIZE THEN
               return(7)

/*            Inválido el tamaño del buffer requerido. */

            ELSE
             IF fp = ASCONFLICT THEN
                return(8)

/*            Problema con el hardware del puerto. */

             ENDIF
            ENDIF
           ENDIF
          ENDIF
         ENDIF
        ENDIF
       ENDIF
      ENDIF
```

```
/*  Rutina cda (param,apv1,pp) */

/*   La rutina convierte de decimal a ASCII, requiere de la
     variable param donde se indica el parámetro, dado por el
     programa que lo solicitó; apv1 es un apuntador que indica la
     posicisión a partir de la cual se va a insertar el número en
     la cadenala, pp es un apuntador a la cadena; regresa el
     valor de l que indica el número de caracteres ASCII que se
     necesitaron para representar al parámetro. */

    repite = false
    l = 1
    numde = param
    WHILE (repite) DO
      numde = numde/10
       IF numde > 0 THEN
         repite = false
               l = l + 1
       ELSE
          repite = true
       ENDIF
         numde = param
         FOR M = 1 DOWNTO 1
           aux = numde/10 exp(m-1)
           (pp)[apv1-1] = toascii(aux+48)
           numde = numde - aux * 10 exp(m-1)
           apv1 = apv1 + 1

/*           48 es la representación del ascii en decimal del 0. */

         ENDFOR
         return(1)
    ENDWHILE

/*   Rutina cierrapt (pto) */

/*   Esta rutina cierra el puerto y requiere de la variable pto
     que debe ser proporcionada por el programa que accesó a la
     función. */

     f = cierra (pto)
     IF f = ASSUCCES THEN
        return(0)

/*      Se cerró bien el puerto. */

     ELSE
      IF f = ASINVEPORT THEN
         return(11)

/*      No existe físicamente el puerto. */

      ENDIF
     ENDIF
```

```
/*  Rutina enviacl (pto,apbuff,vl,EE) /*

/*   Esta  rutina  envía  información y por  estar activado el
     eco espera recibir lo que envía, tiene un ciclo de espera de
     10 segundos y si el modem no contesta  envía  a la función
     un mensaje de error. */

     if (EE = 1)
       FOR (r = 0) TO (vl-1)
          buffaux[0] = apbuff[r]
          vaux = buffaux[0]
          g = envía(pto,buffaux,1,NULL)
          b = 0
          WHILE (b = 0) DO
            b = recibe(pto,buffaux,1,NULL)
            IF (recibe > 10 segundos) THEN
               return(15)

/*             El modem no contesta. */


            ELSE
            ENDIF
          ENDWHILE
            recbuff[r] = buffaux[0]
            IF (vaux =/= buffaux[0]) THEN
               return(12)

/*             El modem no regresa en el eco el valor
               correspondiente. */

            ENDIF
       ENDFOR
     ELSE
       g = envia(pto,apbuff,vl,NULL)
       IF (g =/= vl) THEN
          return(2)

/*        Error en el eco. */

       ENDIF
     ENDIF
     return(0)
     FIN

/*   Rutina inscinc (inst,ap) */

/*   Esta rutina acepta instrucciones con incógnita, la variable
     inst  indica  el  número  de  instrucción  proporcionada por
     el programa que accesó a ésta función.
     Esta función accesa a la matriz de comandos para leerla. /*

     VL = 1
     WHILE (ascii[inst][vl] =/= toascii(13) DO
       IF (ascii[inst][vl] = toascii(35) THEN
```

```
/*          El 13  de la instrucción anterior equivale al ASCII en
            decimal del  carriage  return,  que  indica  el fin de
            comando; el 35 equivale al signo de interrogación. */

        ELSE
          ap[vl-1] = ascii[inst][vl]

        ENDIF
          VL = VL + 1

      ENDWHILE
        ap[vl-1] = ascii[inst][vl]
        return(vl)

/*    Rutina instcpa (inst,param,ap) */

/*    Esta  rutina ejecuta instrucciones  con parámetro,  requiere
      del  número de instrucción,  del parámetro  y del  apuntador
      ap de la cadena. */

      Esta función lee el ASCII de la matriz de comandos.

      vl=1
       k=1
      WHILE (ascii[inst][k] =/= toascii(13)) DO
        IF (ascii[inst][k] = toascii(35))

/*          el ASCII en decimal 13 equivale al carriage return, y
            el 35 al signo de interrogación. */

          g = cda (param.pvl,pp)

/*            cda es la rutina que convierte de decimal a ASCII. */

          vl = vl + g - 1
        ELSE
          ap[vl-1] = ascii[inst,k]
        ENDIF
          vl = vl+1
           k = k+1
      ENDWHILE
      ap[vl-1] = ascii[inst,k]
      return(vl)

/*  Rutina instspa (inst,ap) */

/*    Esta rutina codifica instrucciones sin parámetro,
      requiere  del  número  de  instrucción  dado  en
      inst y del apuntador ap de la  cadena donde
      almacenará la instrucción codificada.
      Esta función lee el ASCII de la matriz de comandos. */

      l = 1
      WHILE ascii(inst,1) =/= toascii(13) DO

/*      13 es el ASCII en decimal del carriage return. */
```

```
      ap[i-1] = ascii(list,i)
              i = i - 1
    ENDWHILE
      ap[i-1] = ascii(list,i)
      return(1)

/*  Rutina leeyesc (pto)  */

/*  Esta rutina  busca el archivo de configuración de acuerdo  a
    las características del modem, llamado "confmod.prt";  si no
    lo  encuentra  lo crea; siguiendo  la estructura del archivo
    de configuración "config.prt". */

    f = lee_config ("confmod.prt")
    IF f = 0  THEN
       f = lee_config ("config.prt")
    ELSE
      IF f = 1 THEN

/*      No hubo error al leer la configuración. */

        salva = true
        IF parámetros(pto).tx_len =/= 2048 THEN
           parámetros(pto).tx_len = 2048

/*         El puerto debe tener una longitud de 2048 bytes. */

           salva = false
      ENDIF
       IF parámetros(pto).parity =/= 1 THEN
          parámetros(pto).parity = 1

/*       La paridad debe ser impar lo que equivale a 1. */

         salva = false
       ENDIF
        IF parámetros(pto).stb =/= 1 THEN
           parámetros(pto).stb = 1

/*         El bit de stop debe ser 1. */

           salva = false
        ENDIF
         IF parámetros(pto).wl =/= 7 THEN
            parámetros(pto).wl = 7

/*          La longitud de palabra debe ser de 7 bits. */

            salva = false
         ENDIF
          IF parámetros(pto).hand_sake =/= 0 THEN
             parámetros(pto).hand_sake = 0

/*           El tipo de comunicación debe ser 0. */
```

```
            salva = false
          ENDIF
           IF salva = false
              f = escribe_config ("confmod.prt")
              return(0)
/*            Parametros de configuración válidos. */

           ENDIF
      ELSE
       return(3)

/*     Parámetros de configuración no válidos */

      ENDIF
     ENDIF

/*  Rutina pren_eco (pto) */

*/   Esta rutina activa el eco,  para ello  EE = 1; requiere
     del número de puerto donde está conectado el modem. */

    inst = 5
     param = 1
     vl = instcpa(inst,parm,buff)
      f = leeyesc(pto)
     IF f = 0 THEN
        f = abrepto(pto)
         IF f = 0 THEN
           FOR (r = 0) TO (r <= vl - 1)
                buffaux[0] = buff[r]
                g = envia(pto,buffaux,1,NULL)
                f = 0
                WHILE (f = 0) DO
                  f = recibe(pto,buff1,1,NULL)
                ENDWHILE
            ENDFOR
              return(0)
         ELSE
           return(f)

/*         en f regresa los errores manejados al abrir puerto. */

         ENDIF
      ELSE
        return(f)
/*      En f regresa los errores manejados en leeyesc. */
      ENDIF

/*  Rutina rebecpar (apbuff,nc) */

/*   La rutina saca un parámetro entero de una cadena. */

       band = 0
          k = 0
       param = 0
        num = True
```

```
    WHILE (num =/= 0) AND (K <= nc) DO
       IF (apbuff[K] >= '0') AND (apbuff[k] <= '9') THEN

/*        Verifica que el numero decimal este comprendido entre
          1 y 9 . */

       IF band = 0 THEN
          band = 1
       ENDIF
       param = apbuff[K] - 48 + Param * 10

/*      Los números encontrados en la cadena los convierte de
        ASCII a decimal. */

     ELSE
        IF Band = 1 THEN
         num = false
        ENDIF
     ENDIF
     ENDIF
     k = k + 1
   ENDWHILE
     IF band = 0 THEN
        return(14)

/*      No encontró el número en la cadena. */
     ELSE
        return(param)
     ENDIF

/*  Rutina  rec_flst (apbuff,appst,nc,i) */

/*   Esta rutina recibe los errores y los almacena  en el vector
     status. */

    numde = rebecpar(apbuff,nc)
    i = 1
    var = 1
    WHILE (i <= 16) DO
       aux = var AND numde
      IF aux = 0 THEN
         appst[i+1-1] = 0
      ELSE
         appst[i+1-1] = 1
      ENDIF
         i = i + 1
        var = var << 1
    ENDWHILE
        return(0)
```

```
/*  Rutina recibect (inst,pto,apcopar,apst) */

/*   Esta rutina recibect es  una rutina  que recibe información
     del modem, dando un tiempo de 10 segundos para que el modem
     conteste. */

     r = 0
     WHILE (vaux =/= '>') DO
       b = 0
       WHILE (b = 0) DO
         b = recibe (pto,buffaux,1,NULL)
         IF (t > 10) THEN
            return(15)

/*          El modem no contesta. */

       ENDWHILE
         buff[r] = buffaux[0]
            vaux = buffaux[0]
               r = r+1
     ENDWHILE

    EN CASO DE QUE inst = 1,2,3,4,5,6,8,9,10,13:
        p = rebecpar (buff,r)
        IF (p =/= 14) THEN
/*         El 14 es la opción de salida del programa. */

        EN CASO DE QUE inst = 1
           apcopar[0] = p
           return(0)
/*         posición en la que se guarda AL. */
        ENDCASE

        EN CASO DE QUE inst = 2
           apcopar[1] = p
           return(0)
/*         posición en la que se guarda BE. */
        ENDCASE

        EN CASO DE QUE inst = 3
           apcopar[2] = p
           return(0)
/*         posición en la que se guarda BR. */
        ENDCASE

        EN CASO DE QUE inst = 4
           apcopar[3] = p
           return(0)
/*         posición en la que se guarda CF. */
        ENDCASE

        EN CASO DE QUE inst = 5
           apcopar[4] = p
           return(0)
/*         posición en la que se guarda EE. */
        ENDCASE
```

```
        EN CASO DE QUE inst = 6
           apcopar[5] = p
           return(0)
/*         posición en la que se guarda EM. */
        ENDCASE

        EN CASO DE QUE inst = 8
           apcopar[6] = p
           return(0)
/*         posición en la que se guarda LB. */
        ENDCASE

        EN CASO DE QUE inst = 9
           apcopar[7] = p
           return(0)
/*         posición en la que se guarda PC. */
        ENDCASE

        EN CASO DE QUE inst = 10
           apcopar[8] = p
           return(0)
/*         posición en la que se guarda RB. */
        ENDCASE

        EN CASO DE QUE inst = 13
           apcopar[9] = p
           return(0)
/*         posición en la que se guarda TP. */
        ENDCASE

        ELSE
          return(14)
/*        No encontró el número en la cadena. */
        ENDIF

    EN CASO DE QUE inst = 7
       r = rec_flst (buff,apst,r,0)
        return(0)
/*      Posición en la que guarda FL. */
    ENDCASE

    EN CASO DE QUE INST = 12
        r = rec_flst (buff,apst,r,16)
        return(0)
/*      Posición en la que guarda ST. */
     ENDCASE

/*   Rutina   valinst  (inst)  */

/*   Esta rutina verifica que la  instrucción  sea válida. */

     IF inst >= 1 OR inst <= 13 THEN
        return(0)

/*      Al regresar un 0 indica que no hubo error */
```

```
      ELSE
         return(1)

/*       Al regresar 1 indica que la instrucción no es válida */

      ENDIF

/*  Rutina verlon (pto,vl) */

/*   Esta rutina   verifica el espacio disponible en el buffer de
     transmisión. */

      f = long bt(pto)
      f = 2048 - f
      IF f >= vl THEN
         return(0)

/*       Se puede mandar mensaje, espacio disponible en memoria. */
      ELSE
         return(9)

/*       No hay espacio disponible para enviar información. */

      ENDIF

/*  Rutina verpar (inst,param) */

/*   Esta rutina verifica que el parámetro sea correcto. */

      CASE (inst) = 1 THEN
           IF (param >= 0 AND param <= 225) THEN
              return(0)
           ELSE
             return(3)
           ENDIF
      END CASE

      CASE (inst) = 2 THEN
           IF (param >= 19 AND param <= 93) THEN
              return(0)
           ELSE
             return(3)
           ENDIF
      END CASE

      CASE (inst) = 3 THEN
           IF (param =/= 1200) THEN
              return(3)
           ELSE
             return(0)
           ENDIF
      END CASE
```

```
CASE (inst) = 4 THEN
     IF (param >= 0 AND param <= 16) THEN
        return(0)
     ELSE
       return(3)
     ENDIF
END CASE

CASE (inst) = 5 THEN
     return(3)
END CASE

CASE (inst) = 6 THEN
     IF (param =/= 0 AND param =/= 1) THEN
        return(3)
     ELSE
       return(0)
     ENDIF
END CASE

CASE (inst) = 7 THEN
     return(3)
END CASE

CASE (inst) = 8 THEN
     IF (param =/= 0 AND param =/= 1) THEN
        return(3)
     ELSE
       return(0)
     ENDIF
END CASE

CASE (inst) = 9 THEN
     IF (param =/= 0 AND param =/= 1) THEN
        return(3)
     ELSE
       return(0)
     ENDIF
END CASE

CASE (inst) = 10 THEN
     return(3)
END CASE

CASE (inst) = 11 THEN
     return(3)
END CASE

CASE (inst) = 12 THEN
     return(3)
END CASE

CASE (inst) = 13 THEN
     IF (param >= 50 AND param <= 250) THEN
        return(0)
     ELSE
```

```
        return(3)
      ENDIF
END CASE
```

8.2 PROGRAMA DE LA FUNCION QUE ACCESA AL MODEM

```
/* FUNAMOD.c .....

   Función principal que permite controlar la operación del modem.
   Inst      es el número del comando a enviar al modem,
   Param     es el parámetro del comando seleccionado,
   Pto       número del puerto al que está conectado el modem,
   Acoparam  vector en el que se reporta el valor de los parámetros
             que tiene programados el modem,
   Astatus   vector en el que se reportan los errores que ocurran en
             la operación del modem. */

#include <stdio.h>

/* Declaración de las variables usadas en la rutina abre puerto
(abrepto) */

int abrepto(int), pto;

/* Declaración de variables usadas en la rutina convierte de decimal
a ASCII (cda) */

int cda(int,int,char[]),param,apvl,pp;

/* Declaración de variables usadas en la rutina cierra puerto
(cierrapt) */

int cierrapt(int),pto;

/* Declaración de variables usadas en la rutina envia información
(enviact) */

int enviact(int,char[],int,int),pto,apbuff,vl,EE;

/* Declaración de variables usadas en la rutina instrucción con
incógnita (inscinc) */

int inscinc(int,char[]),inst,ap;

/* Declaración de variables usadas en la rutina instrucción con
parámetro (instcpa) */

int instcpa(int,int,char[]),inst,param,ap;

/* Declaración de variables usadas en la rutina instrucción sin
parámetro (instspa) */

int instspa(int,char[]),inst,ap;

/* Declaración de variables usadas en la rutina lee y escribe
configuración (leeyesc) */

int leeyesc(int),pto;
```

```
/* Declaración de variables usadas en la rutina prende eco
(pren_eco) */

int pren_eco(int),pto;

/* Declaración de variables usadas en la rutina recibe con parámetro
(rebecpar) */

int rebecpar(char[],int),apbuff,nc;

/* Declaración de variables usadas en la rutina recibe información
(recibect) */

int recibect(int,int,int[],int[]),inst,pto,apcopar,apstatus;

/* Declaración de variables usadas en la rutina recibe el comando FL
Y ST (rec_flst) */

int rec_flst(char[],int[],int,int),apbuff,appst,nc,i;

/* Declaración de variables usadas en la rutina valida instrucción
(valinst) */

int valinst(int),inst;

/* Declaración de variables usadas en la rutina verifica longitud
(verlong) */

int verlong(int),vl;

/* Declaración de variables usadas en la rutina valida parámetro
(valpar) */

int verpar(int,int),inst,param;

/* Declaración de las variables usadas en la rutina función acceso al
modem (funamod) */

int funamod(inst,param,pto,acoparam,astatus)
   int inst,param,pto,(*acoparam)[16],(*astatus)[32];
{
 int  error,f,vp;
 char buff[30];

 f = valinst(inst);
 if (f == 0) {
   f = pren_eco(pto);
   if ( f == 0 ) {
   switch (inst) {
     case (1) :
     case (2) :
     case (3) :
     case (4) :
     case (5) :
     case (6) :
     case (7) :
```

```
case (8) :
case (9) :
case (10) :
case (12) :
case (13) :
/* instrucciones con parámetro. */
 if (param != 63) {
  vp = verpar(inst,param);
    if (vp != 0) {
       cierra(pto);
       return(1);
    }
  strset(buff,'\0');
  vl = instcpa(inst,param,buff);
    /* verifica que la instrucción sea con parámetro. */
  f = verlon(pto,vl);
     /* verifica la longitud en el buffer de transmision. */
  if (f == 0) {
    f = enviact(pto,buff,vl,1);
    /* envia información. */
    if (f == 0) {
    f = cierrapt(pto);
       /* Enviada la información se cierra el puerto. */
    if (f == 0) {
       return(0);
    }
    else {
       return(f);
    }
    }
    else {
    return(f);
    }
  }
  else {
     return(f);
  }
 }
 else {
if ( inst == 4 ) {
   cierrapt(pto);
   return(13);
}
strset(buff,'\0');
vl = inscinc(inst,buff);
   /* Ejecuta instrucción con incognita. */
   f = verlon(pto,vl);
      /* Rutina de verifica longitud en buffer. */
   if (f == 0) {
      f = enviact(pto,buff,vl,1);
       /* Rutina de enviar información.*/
      if (f == 0) {
            error = 0;
         }
      else {
      error = f;
```

```
            }
        }
        else {
            error = f;
        }
    }
    break;

    case(11):
    /* Instrucción sin parámetro, el caso del reset. */
    strset(buff,'\0');
    vl = instspa(inst,buff);
    f = verlon(pto,vl);
        /* Rutina de verifica longitud en buffer. */
    if (f == 0) {
        f = enviact(pto,buff,vl,1);
            /* Rutina de enviar información.*/
        if (f == 0) {
            f = cierrapt(pto);
             /* Ejecuta la rutina de cierra puerto. */
            if (f == 0) {
            return(0);
            }
            else {
            return(f);
            }
        }
        else {
            return(f);
        }
    }
    else {
        return(f);
    }
    break ;
    }
    if (param == 63 && error == 0) {
/* Si el parámetro es incognita: */
    f = recibect(inst,pto,acoparam,astatus);
        /* Recibe información. */
    if (f == 0) {
        f = cierrapt(pto);
         /* Cierra puerto. */
        if (f == 0) {
        return(0);
        }
        else {
        return(f);
        }
    }
    else {
        return(f);
    }
}
else {
  return(f);
```

```
    }
     ]
     else {
    cierra(pto);
    return(f);
    /* error en prende eco */
     }
}
return(f);
}
```

```
/* Abrepto.c ........

Esta rutina abre el puerto; requiere del parámetro pto que debe ser
especificado por el programa que acceso a la función FUNAMOD,
indicando el número de puerto de la tarjeta de "Digibord" que será
utilizado para manejar al modem. */


#include <stdio.h>
#include "comunica.h"


int abrepto(pto)
   int  pto;

{
  int fp;
  fp = abre(pto,ASINOUT);
  if (fp == ASSUCCESS) {
     return(0);
     /* No hubo error al abrir el puerto. */
  }
  else {
    if (fp == ASINVPORT) {
       return(3);
       /* Parámetros de configuración del puerto inválidos. */
    }
    else {
      if (fp == ASINUSE) {
     return(4);
     /* El puerto ya estaba abierto. */
      }
      else {
    if (fp == ASNOMEMORY) {
       return(5);
       /* No hay memoria suficiente en el buffer del puerto. */
    }
    else {
      if (fp == ASNO8250) {
         return(6);
         /* No esta el UART en la dirección especificada. */
      }
      else {
        if (fp == ASINVBUFSIZE) {
           return(7);
           /* Inválido el tamaño del buffer requerido.*/
        }
```

```
        else {
          if (fp == ASCONFLICT) {
          return(8);
          }
        }
      }
    }
      }
    }
  }
}
```

```
/* Cda.c .....

La rutina cda convierte un número decimal a su representación en
ascii. Param es el número a convertir, apvl es la posición en la
cadena (*pp) donde se icluirá el número. */

#include <stdio.h>
#include <ctype.h>
#include <math.h>

 int  cda(param,apvl,pp)
      int param,apvl;
      char (*pp)[30];

{
 int  aux,l,numde,m,repite;

 repite=0;
 l=1;
 numde = param;
 while (repite==0 ){
    numde=numde/10;
    if (numde > 0)  {
       repite=0;
       l=l+1;
    }
    else {
       repite=1;
    }
 }
 numde=param;
 for (m=l;m>=1;m--) {
     aux=numde / pow(10.0,1.0*(m-1));
     (*pp)[apvl-1]=toascii(aux+48);
     numde=numde-aux*pow(10.0,1.0*(m-1));
     apvl=apvl+1;
 }
 return(l);
}
```

```
/* Cierrapt.c .....

Esta rutina cierra el puerto (pto) especificado por el programa que
accesa a la función FUNAMOD. */

#include <stdio.h>
#include "comunica.h"

 int cierrapt(pto)
    int pto;
{

 int f;

 f = cierra(pto);
 if (f == ASSUCCESS) {
    return(0);
    /* Se cerró bien el puerto. */
 }
 else {
   if (f == ASINVPORT) {
      return(11);
      /* No existe físicamente el puerto. */
   }
 }
}
```

```
/* Enviact.c .....

Esta rutina envia una cadena de información.Pto es el número del
puerto por donde se hará la transmisión, apbuff es la cadena de
información a transmitir, vl es la longitud de la cadena y EE
indicador de eco. */

#include <stdio.h>
#include <ctype.h>
#include <time.h>

 int enviact(pto,apbuff,vl,EE)
     char (*apbuff)[30];
     int  pto,vl,EE;
{

 int salida,r,g,b;
 char vaux,buffaux[30],recbuff[30];
 time_t start,finish;
 double t;

 if ( EE == 1 ) {
    for ( r = 0 ; r <= vl-1 ; r++ ) {
       sprintf(buffaux,"");
       buffaux[0] = (*apbuff)[r];
       vaux = buffaux[0];
       g = envia(pto,buffaux,1,NULL);
       b=0;
       time (&start);
       while (b==0) {
/*           printf(" Espera mensaje %d rec %s\r",r,recbuff); */
      sprintf(buffaux,"");
      b = recibe(pto,buffaux,1,NULL);
          time (&finish);
        t = difftime(finish,start);
         if (t > 30.0) {
            return(15);
            /* El modem no contesta. */
         }
       }
       recbuff[r]=buffaux[0];
       if ( vaux != buffaux[0] ) {
      return(12);

       }
    }
 }
 else {
    g = envia ( pto, (*apbuff), vl, NULL);
    if ( g != vl ) {
       return(2);
    }
 }
 return(0);
}
```

```
/* Inscinc.c .....

Esta rutina codifica instrucciones con incógnita, la variable inst
indica el número de instrucción proporcionada por el programa que
acceso a la función FUNAMOD. Ap es la cadena donde se codifica la
instrucción. */

#include <stdio.h>
#include <ctype.h>

int inscinc(inst,ap)
   int inst;
   char (*ap)[30];
{
 int vl;
 char ascii[14][7];

    /* A continuación se introduce la matriz de comandos */

 ascii[1][1]=toascii(27),  ascii[1][2]=toascii(65),
ascii[1][3]=toascii(76);

 ascii[1][4]=toascii(32),  ascii[1][5]=toascii(35),
ascii[1][6]=toascii(13);

 ascii[2][1]=toascii(27),  ascii[2][2]=toascii(66),
ascii[2][3]=toascii(69);

 ascii[2][4]=toascii(32),  ascii[2][5]=toascii(35),
ascii[2][6]=toascii(13);

 ascii[3][1]=toascii(27),  ascii[3][2]=toascii(66),
ascii[3][3]=toascii(82);

 ascii[3][4]=toascii(32),  ascii[3][5]=toascii(35),
ascii[3][6]=toascii(13);

 ascii[4][1]=toascii(27),  ascii[4][2]=toascii(67),
ascii[4][3]=toascii(70);

 ascii[4][4]=toascii(32),  ascii[4][5]=toascii(35),
ascii[4][6]=toascii(13);

 ascii[5][1]=toascii(27),  ascii[5][2]=toascii(69),
ascii[5][3]=toascii(69);

 ascii[5][4]=toascii(32),  ascii[5][5]=toascii(35),
ascii[5][6]=toascii(13);

 ascii[6][1]=toascii(27),  ascii[6][2]=toascii(69),
ascii[6][3]=toascii(77);

 ascii[6][4]=toascii(32),  ascii[6][5]=toascii(35),
ascii[6][6]=toascii(13);
```

```
 ascii[7][1]=toascii(27),  ascii[7][2]=toascii(70),
ascii[7][3]=toascii(76);

 ascii[7][4]=toascii(32),  ascii[7][5]=toascii(35),
ascii[7][6]=toascii(13);

 ascii[8][1]=toascii(27),  ascii[8][2]=toascii(76),
ascii[8][3]=toascii(66);

 ascii[8][4]=toascii(32),  ascii[8][5]=toascii(35),
ascii[8][6]=toascii(13);

 ascii[9][1]=toascii(27),  ascii[9][2]=toascii(80),
ascii[9][3]=toascii(67);

 ascii[9][4]=toascii(32),  ascii[9][5]=toascii(35),
ascii[9][6]=toascii(13);

 ascii[10][1]=toascii(27), ascii[10][2]=toascii(82),
ascii[10][3]=toascii(66);

 ascii[10][4]=toascii(32), ascii[10][5]=toascii(35),
ascii[10][6]=toascii(13);

 ascii[11][1]=toascii(27), ascii[11][2]=toascii(82),
ascii[11][3]=toascii(69);

 ascii[11][4]=toascii(13);

 ascii[12][1]=toascii(27), ascii[12][2]=toascii(83),
ascii[12][3]=toascii(84);

 ascii[12][4]=toascii(32), ascii[12][5]=toascii(35),
ascii[12][6]=toascii(13);

 ascii[13][1]=toascii(27), ascii[13][2]=toascii(84),
ascii[13][3]=toascii(80);

 ascii[13][4]=toascii(32), ascii[13][5]=toascii(35),
ascii[13][6]=toascii(13);

 v1 = 1;
 while (ascii[inst][v1] != toascii(13)) {
   if (ascii[inst][v1] == toascii(35)) {
      /* El 35 es el asccii del signo de interrogación. */
      (*ap)[v1-1] = 63;
      /* El 63 es la representación del signo de interrogación. */
   }
   else {
     (*ap)[v1-1] = ascii[inst][v1];
   }
   v1 = v1 + 1;
 }
 (*ap)[v1-1] = ascii[inst][v1];
 return(v1);
}
```

```
/* Instcpa.c .....

Esta rutina  codifica instrucciones  con parámetro,  requiere de  los
parámetros  inst  y param, deben ser  proporcionados por el  programa
que accesa a la función FUNAMOD. Ap es la cadena donde se codifica la
instrucción. */

#include <stdio.h>
#include <ctype.h>

int instcpa(inst,param,ap)
     int inst,param;
     char (*ap)[30];

{
 int g,k,v1;
 char ascii[14][7];

    /* A continuación se introduce la matriz de comandos al igual que
para el programa Inscinc.c (instrucción con incógnita) */


 k=1;
 v1=1;
 while (ascii[inst][k] != toascii(13)) {
   if (ascii[inst][k] == toascii(35)){
      g=cda(param,v1,ap);
      v1 = v1 + g - 1;
/* la rutina cda, convierte los datos del parámetro de decimal a
ascii. */
   }
   else {
     (*ap)[v1-1]=ascii[inst][k];
   }
   v1 = v1 + 1;
   k  = k + 1;
 }
 (*ap)[v1-1]=ascii[inst][k];
 return(v1);
}
```

```
/* Instspa.c .....

Instrucción sin parámetro; esta  rutina requiere de la variable  inst
que indica el número de la  instrucción a codificar, ap es  la cadena
donde se codifica la instrucción. */

#include <stdio.h>
#include <ctype.h>


int instspa(inst,ap)
   int inst;
   char (*ap)[30];
{
 int l;
 char ascii[14][7];

    /* A continuación se introduce la matriz de comandos al igual que
para el programa Inscinc.c (instrucción con incógnita) */


 l = 1;
 while (ascii[inst][l] != toascii(13)) {
   (*ap)[l-1] = ascii[inst][l];
     l = l + 1;
 }
 (*ap)[l-1] = ascii[inst][l];
 return (l);
}
```

```
/* Leeyesc.c .....

Esta rutina, si existe la configuración del puerto la lee, de lo
contrario crea una configuración y la escribe a un archivo. */

#include <stdio.h>
#include "comunica.h"


int leeyesc(pto)
    int pto;

{
 int  f,salva;

 f = lee_config("confmod.prt");
 if ( f == 0) {
    f = lee_config("config.prt");
 }
 if (f == 1) {
    /* No hubo error al leer cionfiguración. */
   salva = 1;
   if (parametros[pto].tx_len != 2048) {
      parametros[pto].tx_len = 2048;
      /* El puerto debe tener una longitud de 2048 bytes. */
      salva = 0;
   }
   if (parametros[pto].parity != 1 ){
      parametros[pto].parity = 1;
      /* Debe ser paridad impar, lo que equivale a 1.*/
      salva = 0;
   }
   if( parametros[pto].stb != 1) {
       parametros[pto].stb = 1;
       /* El bit de stop debe ser 1. */
       salva = 0;
   }
   if (parametros[pto].wl != 7) {
      parametros[pto].wl = 7;
      /* La longitud de palabra debe ser de 7 bits. */
      salva = 0;
   }
   if (parametros[pto].hand_shake != 0) {
      parametros[pto].hand_shake = 0;
      /* El tipo de comunicación debe ser 0. */
      salva = 0;
   }
   if (salva == 0) {
      f = escribe_config("confmod.prt");
      return(0);
      /* Parámetros de configuración del puerto válidos. */
   }
   else {
      return(0);
   }
```

```
 }
 else {
    /* Parámetros de configuración del puerto inválidos. */
    return(3);
 }
}
```

```
/* Pren_eco.c .....

Esta rutina permite activar el eco en el modem. */

#include <stdio.h>

int pren_eco(pto)
    int pto;
{

 int  inst,param,vl,f,j,r,g;
 char buff[30],buffaux[30],vaux,buffl[30];

 inst = 5;
 param = 1;
 strset(buff,'\0');
 strset(buffl,'\0');
 vl = instcpa(inst,param,buff);
 f  = leeyesc(pto);
 if (f == 0) {
   f = abrepto(pto);
   if (f == 0) {
      for ( r = 0 ; r <= vl-1 ; r++ ) {
      strset(buffaux,'\0');
      buffaux[0] = buff[r];
      g = envia(pto,buffaux,1,NULL);
      f = 0;
      while ( f == 0 ) {
        f = recibe(pto,buffl,1,NULL);
      }
      }
      return(0);
   }
   else {
      return(f); /* regresa los errores */
   }
 }
 else {
    return (f);
 }
}
```

```
/* Rebecpar.c .....

Esta rutina interpreta información del modem, cuando se trata de
comandos con parámetro. Apbuff es la cadena recibida, nc es la
longitud de la cadena. */

#include <stdio.h>

int rebecpar(apbuff,nc)
   char (*apbuff)[30];
   int nc;

{

 int  k,band,num,param;

 band=0;
 k = 0;
 param=0;
 num = 1 ;
 while (num != 0 && k <= nc) {
   if (((*apbuff)[k] >= '0') && ((*apbuff)[k] <='9')) {
      /* Verifica que el número decimal este
     comprendido entre 0 y 9 . */
      if (band == 0 ){
     band = 1;
      }
       param=(*apbuff)[k] - 48 + param * 10;
         /* Los números encontrados en la cadena los
         convierte de ascii a decimal */

   }
   else {
    if (band == 1) {
       num = 0;
    }
   }
   k=k+1;
 }
 if ( band == 0 ) {
    return(14);
    /* No encontró el número en la cadena. */
 }
 else{
    return(param);
 }
}
```

```
/* Recibect.c .....

Esta rutina recibe información del modem, requiere de la variable
inst, proporcionada por el programa que accesa a la función, pto es
el número del puerto y almacena la información recibida en los
vectores apcopar y apst. */

#include <stdio.h>
#include <time.h>

int recibect(inst,pto,apcopar,apst)
    int inst,pto;
    int (*apcopar)[16],(*apst)[32];

{

 int p,q,r,f,b;
 char buff[30],buffaux[30];
 int  vaux;
 time_t start, finish;
 double t;

       r=0;
       strset(buff,'\0');
       vaux = '\0';
       while (vaux != '>'){
      b=0;
      time (&start);
      while (b==0) {
/*            printf(" Espera mensaje %d vaux %c\r",r,vaux);*/
        strset(buffaux,'\0');
        b = recibe(pto,buffaux,1,NULL);
         time (&finish);
          t = difftime(finish,start);
           if (t > 10.0) {
              return(15);
              /* El modem no contesta. */
           }
      }
      buff[r]=buffaux[0];
      vaux=buffaux[0];
      r=r+1;
       }
       buff[r]='\0';
      /* printf("\n Se recibio el mensaje de %d \n",r);
       printf(" El mensaje %s \n",buff);*/

 switch(inst) {
   case(1):
   case(2):
   case(3):
   case(4):
   case(5):
   case(6):
   case(8):
```

```
case(9):
case (10):
case (13):
  p=rebecpar(buff,r);

  if (p != 14) {

    switch(inst) {

    case(1):(*apcopar)[0]=p;
        return(0);
        /* posición en la que guarda AL. */
        break;

    case(2):(*apcopar)[1]=p;
        return(0);
        /* posición en la que guarda BE. */
        break;

    case(3):(*apcopar)[2]=p;
        return(0);
        /* posición en la que guarda BR. */
        break;

    case(4):(*apcopar)[3]=p;
        return(0);
        /* posición en la que guarda CF. */
        break;

    case(5):(*apcopar)[4]=p;
        return(0);
        /* posición en la que guarda EE. */
        break;

    case(6):(*apcopar)[5]=p;
        return(0);
        /* posición en la que guarda EM. */
        break;

    case(8):(*apcopar)[6]=p;
        return(0);
        /* posición en la que guarda LB. */
        break;

    case(9):(*apcopar)[7]=p;
        return(0);
        /* posición en la que guarda PC. */
        break;

    case(10):(*apcopar)[8]=p;
        return(0);
        /* posición en la que guarda RB. */
        break;

    case(13):(*apcopar)[9]=p;
      return(0);
```

```
        /* posición en la que guarda TP. */
        break;
      }
    }
    else {
   return(14);
   /* No encontro el numero en la cadena. */
    }
    break;

  case(7):r=rec_flst(buff,apst,r,0);
        /* posición en la que guarda FL. */
        return(0);
        break;

  case(12):r=rec_flst(buff,apst,r,16);
         /* posición en la que guarda ST. */
         return(0);
         break;

 }
}
```

```
/* Reci_be.c .....

Esta rutina interpreta el umbral de  la tasa de error de  bit. apbuff
es la cadena a interpretar, cr número de caracteres en la cadena. */

#include <stdio.h>
#include <math.h>

float reci_be(apbuff,cr)
    int cr;
    char (*apbuff)[30];

{

 int  j,num,aux;
 float parambe;

 num=1;
 j=0;
 while (num != 0 && j <= cr) {
   if ((*apbuff)[j] >= '0' && (*apbuff)[j] <= '9') {
      aux = (*apbuff)[j+1]-48;
      parambe=((*apbuff)[j]-48) * pow(10.0,-aux);
      num = 0;
   }
      j=j+1;
 }
  if (num == 1) {
     return(9999);
  }
  else {
     return(parambe);
  }
}
```

```
/* Rec_flst.c .....

Esta rutina interpreta los errores ocurridos en al operación del
modem, y los reporta en el vector appst. Apbuff es la cadena a
interpretar, appst es el vector donde se reportarán los errores, nc
número de carateres de la cadena, i indicador de fl o st. */

#include <stdio.h>

int rec_flst(apbuff,appst,nc,i)
    int nc,i;
    char (*apbuff)[30];
    int (*appst)[32];

{

 int l,var,aux,numde;

 numde = rebecpar(apbuff,nc);
 l=1;
 var=1;
 while (l <= 16) {
   aux = var & numde;
   if (aux == 0) {
      (*appst)[i+l-1] = 0;
   }
   else {
      (*appst)[i+l-1] = 1;
   }
      l=l+1;
      var = var << 1;
 }
 return(0);
}
```

```
/* Valins.c .....

Esta función valida la instrucción (inst) solicitada por el
usuario. */

#include <stdio.h>

int valinst(inst)
    int inst;

{

 if ((inst>=1) && (inst<=13)) {    /* (inst>=1)and(inst<=13) */
    return (0);
    /* Al regresar un 0 indica que no hubo error. */
 }
 else {
   return (1);
   /* Al regresar un 1 indica que la instrucción es inválida. */
 }
}
```

```
/* Verlon.c .....

Esta rutina verifica si hay espacio para enviar un comando. Vl es el
número de caracteres  a transmitir. */

#include <stdio.h>
#include "comunica.h"

int verlon(pto,vl)
    int pto,vl;

{

 int f;

 f = long_bt(pto);
 f = 2048-f;
 if (f >= vl) {
    return(0);
    /* Lo anterior indica que hay espacio disponible
       en memoria para mandar mensaje. */
 }
 else {
  return(9);
  /* Al regresar un nueve se indica que no hay espacio
     disponible para enviar información. */
 }
}
```

```
/* Verpar.c .....

La rutina verpar verifica la validez del parámetro (param) de la
instrucción (inst). */

#include <stdio.h>

int verpar(inst,param)
    int inst, param;

{

 switch (inst) {

 case(1):
  if (param >= 0 && param <= 1) {
     return(0);
   }
   else {
     return(1);
     /* parámetro no válido. */
   }
   break;

 case(2):
  if (param >= 19 && param <= 93) {
     return(0);
   }
    else {
     return(1);
     /* parámetro no válido. */
   }
   break;

 case(3):
   if (param != 1200) {
      return(1);
      /* parámetro no válido. */
   }
   else {
     return(0);
   }
   break;

 case(4):
   if (param >= 0 && param <= 16) {
      return(0);
   }
   else {
     return(1);
     /* parámetro no válido. */
   }
   break;

 case(5):
```

```
    return(1);
    break;

case(6):
  if (param != 0 && param != 1) {
     return(1);
     /* parámetro no válido. */
  }
  else {
    return(0);
  }
  break;

case(7):
  return(1);
  break;

case(8):
  if (param != 0 && param != 1) {
     return(1);
     /* parámetro no válido. */
  }
  else {
    return(0);
  }
  break;

case(9):
  if (param != 0 && param != 1) {
     return(1);
     /* parámetro no válido. */
  }
  else {
    return(0);
  }
  break;

case(10):
    return(1);
    break;

case(11):
  return(1);
  break;

case(12):
  return(1);
  break;

case(13):
  if (param >= 50 && param <= 250) {
     return(0);
    }
  else {
    return(1);
     /* parámetro no válido. */
```

```
    }
    break;
  }
}
```

9 APENDICE D

9.1 SEUDOCODIGO DE FUNCION QUE ACCESA A LA FUNCION QUE CONTROLA AL MODEM

Seudocódigo del programa proacmol que accesa la función que interactua con el modem.

```
repite = false
 lee número de puerto donde está conectado el modem
WHILE (repite =/= 14) DO
/*        El 14 es la opción de salir del programa. */
  escribe lista de los comandos a enviar al modem:

    1  AL Reporte de alarma.
    2  BE Umbral de la tasa de error de bit.
    3  BR Velocidad de transmisión.
    4  CF Limpiar el registro de error.
    5  EE Habilita/deshabilita el caracter de eco.
    6  EM Habilitación del modulador.
    7  FL Preguntar por el registro de errores.
    8  LB Modo 'loop back'
    9  PC Forza la transmisión de la portadora.
    10 RB Lectura de la tasa de error de bit.
    11 RE Reestablece el sistema.
    12 ST Pregunta por fallas actuales.
    13 TP Transmisión del nivel de potencia.
    14    Salir del programa.

    lee inst
    IF (inst>=1 AND inst<=13 AND pto>=0 AND pto<=34) THEN
       EN CASO DE QUE inst = 1,2,3,4,5,6,8,9,10,13
          escribe 1 El parámetro a enviar es un número.
          escribe 2 El parámetro a enviar es la incógnita.
          lee opción
          IF (opción = 1) then
            lee param
            pregunta = false
          ELSE
            param = '?'
            pregunta = true
          ENDIF
       ENDCASE
         EN CASO DE QUE inst = 7,12
            param = '?'
            pregunta = true
         ENDCASE
           EN CASO DE QUE inst = 11
              param = 0
              pregunta = false
         ENDCASE
           ff = funamod (inst,param,pto,coparam,apst)
         IF (ff = 0) THEN
            escribe no hubo error
           IF (pregunta =/= 0) THEN
             EN CASO DE inst = 1
                escribe "Parámetro que tiene AL" = coparam[0]
```

```
ENDCASE

EN CASO DE inst = 2
   escribe "Parámetro que tiene BE" = coparam[1]
ENDCASE

EN CASO DE inst = 3
   escribe "Parámetro que tiene BR" = coparam[2]
ENDCASE

EN CASO DE inst = 4
   escribe "Parámetro que tiene CF" = coparam[3]
ENDCASE

EN CASO DE inst = 5
   escribe "Parámetro que tiene EE" = coparam[4]
ENDCASE

EN CASO DE inst = 6
   escribe "Parámetro que tiene EM" = coparam[5]
ENDCASE

EN CASO DE inst = 7
   i = 0
   WHILE (i <= 15) DO
     lee i
     escribe "Errores acumulados FL = apst[i]"
     i = i+1
   ENDWHILE
ENDCASE

EN CASO DE inst = 8
   escribe "Parámetro que tiene LB" = coparam[6]
ENDCASE

EN CASO DE inst = 9
   escribe "Parámetro que tiene PC" = coparam[7]
ENDCASE

EN CASO DE inst = 10
   escribe "Parámetro que tiene RB" = coparam[8]
ENDCASE

EN CASO DE inst = 12
   i = 16
   WHILE (i <= 31) DO
     lee i
     escribe "Errores acumulados ST = apst[i]"
     i = i+1
   ENDWHILE
ENDCASE
```

```
    EN CASO DE inst = 13
       escribe "Parámetro que tiene TP" = coparam[9]
    ENDCASE
      pregunta = 0

  ENDIF
    repite = true
ELSE
  escribe Ocurrió error: ff
  EN CASO DE QUE ff = 1
   escribe "Instrucción no válida."
  ENDCASE

  EN CASO DE QUE ff = 2
   escribe "Error en el eco."
  ENDCASE

  EN CASO DE QUE ff = 3
   escribe "Parámetros de configuración no válidos."
  ENDCASE

  EN CASO DE QUE ff = 4
   escribe "El puerto ya estaba abierto."
  ENDCASE

  EN CASO DE QUE ff = 5
   escribe "Memoria insuficiente en el buffer del
            puerto."
  ENDCASE

  EN CASO DE QUE ff = 6
   escribe "La dirección del UART no es la
   especificada"
  ENDCASE

  EN CASO DE QUE ff = 7
   escribe "El tamaño del buffer no es válido."
  ENDCASE

  EN CASO DE QUE ff = 8
   escribe "Problema con el hardware del puerto."
  ENDCASE

  EN CASO DE QUE ff = 9
   escribe "Espacio insuficiente en memoria para
            transmisión."
  ENDCASE

  EN CASO DE QUE ff = 10
   escribe "Error en el número de bytes
   transmitidos."
  ENDCASE
```

```
          EN CASO DE QUE ff = 11
           escribe "Problema al cerrar el puerto."
          ENDCASE

          EN CASO DE QUE ff = 12
           escribe "El modem no regresó el valor
                    correspondiente."
          ENDCASE

          EN CASO DE QUE ff = 13
            escribe "Error la instrucción no acepta
            incógnita."
          ENDCASE

          EN CASO DE QUE ff = 14
           escribe "No encontró el número en la cadena."
          ENDCASE

          EN CASO DE QUE ff = 15
           escribe "Error el modem no contesta."
          ENDCASE

        ENDIF
    ELSE
      IF (inst = 14) THEN
        repite = 14
      ENDIF
    ENDIF
ENDWHILE
END
```

9.2 PROGRAMA DE LA FUNCION QUE ACCESA A LA FUNCION QUE CONTROLA AL MODEM

```
/* Proacmol.c .....

   Programa de prueba que accesa a la función FUNAMOD. */

#include <stdio.h>
#include <conio.h>
#include <cscape.h>
#include <popdec1.h>

 char *opciones[] =
             ( " AL Reporte de Alarma                          ",
               " BE Umbral de la tasa de error de bit          ",
               " BR Velocidad de transmision                   ",
               " CF Limpiar el registro de error               ",
               " EE Habilita/deshabilita el caracter de eco    ",
               " EM Habilitacion del modulador                 ",
               " FL Pregunta por el registro de errores        ",
               " LB Modo 'loop back'                           ",
               " PC Forza la transmision de la portadora       ",
               " RB Lectura de la tasa de error de bit         ",
               " RE Reestablece el sistema                     ",
               " ST Pregunta por fallas actuales               ",
               " TP Transmision del nivel de potencia          ",
               " Salir del programa                            ",
              NULL );
 char *paramop[] =  ( " Número          ",
                      " Incógnita ?     ",
                      NULL );

    menu_type       menu,menu1,menu2;
    sed_type        sed,sed1,sed2;
    int             pto,inst,param;
    int             coparam[16],status[32];
    int             c;

DespParam(){
   menu1 = menu_Open();
   menu_Printf(menu1, "@p[0,0]@c[\x10]");
   menu_Printf(menu1, "@p[1,0]@c[\x1b]%01d@c[\x10]",pto);
   menu_Printf(menu1, "@p[3,1]@c[\x1b]%01d@c[\x10]",coparam[0]);
   menu_Printf(menu1, "@p[4,1]@c[\x1b]%02d@c[\x10]",coparam[1]);
   menu_Printf(menu1, "@p[5,1]@c[\x1b]%04d@c[\x10]",coparam[2]);
   menu_Printf(menu1, "@p[6,1]@c[\x1b]%03d@c[\x10]",coparam[3]);
   menu_Printf(menu1, "@p[7,1]@c[\x1b]%01d@c[\x10]",coparam[4]);
   menu_Printf(menu1, "@p[8,1]@c[\x1b]%01d@c[\x10]",coparam[5]);
   menu_Printf(menu1, "@p[10,1]@c[\x1b]%01d@c[\x10]",coparam[6]);
   menu_Printf(menu1, "@p[11,1]@c[\x1b]%01d@c[\x10]",coparam[7]);
   menu_Printf(menu1, "@p[12,1]@c[\x1b]%01d@c[\x10]",coparam[8]);
   menu_Printf(menu1, "@p[15,1]@c[\x1b]%01d@c[\x10]",coparam[9]);
   menu_Flush(menu1);
   sed1=sed_Open(menu1);
   sed_SetColors(sed1, '\x1e', '\x1e', '\xe0');
```

```
    sed_SetPosition(sed1, 3, 66);
    sed_BorderPrompt(sed1,"");
    sed_Repaint(sed1);
    sed_Go(sed1);
    menu_Close(menu1);
    sed_Close(sed1);
}

DespFLST(){
    int i;

    menu2 = menu_Open();
    menu_Printf(menu2, "@p[0,0]@c[\x70] \n");
    menu_Printf(menu2, "@p[0,0]");
    if ( inst == 7 ){
      i = 0;
      do {
         if (status[i]==1) {
           switch (i) {
       case(0)  : menu_Printf(menu2, "@c[\x70] No detecta Falla
                  @c[\x70]\n");
                  break;
       case(1)  : menu_Printf(menu2, "@c[\x70] Error en la
                  sintetización de la transmisión en la frecuencia \n
                  intermedia TXIF @c[\x70]\n");
                  break;
       case(2)  : menu_Printf(menu2, "@c[\x70] Error en la
                  sintetización de la recepción en la frecuencia \n
                  intermedia @c[\x70]\n");
                  break;
       case(3)  : menu_Printf(menu2, "@c[\x70] Error en el tiempo
                  VCXO en la transmisión de bit TXBIT @c[\x70]\n");
                  break;
       case(4)  : menu_Printf(menu2, "@c[\x70] Error en la recepción
                  del control automático de ganancia @c[\x70]\n");
                  break;
       case(5)  : menu_Printf(menu2, "@c[\x70] Falla en el tiempo de
                   amarre de lazo del bit del demodulador @c[\x70]\n");
                  break;
       case(6)  : menu_Printf(menu2, "@c[\x70] Falla en el amarre de
                  lazo en la portadora del demodulador @c[\x70]\n");
                  break;
       case(7)  : menu_Printf(menu2, "@c[\x70] Falla de lazo en la
                  sincronización del decodificador del demodulador
                  @c[\x70]\n");
                  break;
       case(8) : menu_Printf(menu2, "@c[\x70] Error en la dquisición
                   del demodulador @c[\x70]\n");
                  break;
       case(9) : menu_Printf(menu2, "@c[\x70] Error en el rango de
                   recepción de la portadora VCXO @c[\x70]\n");
                  break;
       case(10) : menu_Printf(menu2, "@c[\x70] Error en el rango de
                   transmisión de la portadora VCXO @c[\x70]\n");
                  break;
       case(11) : menu_Printf(menu2, "@c[\x70] Error de bit en el
```

```
                rango de tiempo de recepción VCXO @c[\x70]\n");
              break;

    case(12) : menu_Printf(menu2, "@c[\x70] Falla en la memoria
                no volátil del demodulador @c[\x70]\n");
              break;
    case(13) : menu_Printf(menu2, "@c[\x70] Falla en el
                microprocesador del demodulador @c[\x70]\n");
              break;
          )
        )
       i = i + 1;
  ) while(i<=15);
  menu_Printf(menu2, "@c[\x74]\n Oprima una tecla para continuar
               @c[\x70]");
}
if ( inst == 12 ){
  i = 16;
  do {
     if (status[i]==1) {
       switch (i) {
    case(16)  : menu_Printf(menu2, "@c[\x70] No detecta falla
                @c[\x70]\n");
              break;
    case(17)  : menu_Printf(menu2, "@c[\x70] Error en la
                sintetización de la transmisión en la frecuencia
                \n intermedia TXIF @c[\x70]\n");
              break;
    case(18)  : menu_Printf(menu2, "@c[\x70] Error en la
                sintetización de la recepción en la frecuencia \n
                intermedia RXIF @c[\x70]\n");
              break;
    case(19)  : menu_Printf(menu2, "@c[\x70] Error en el tiempo
                VCXO en la transmisión de bit TXBIT @c[\x70]\n");
              break;
    case(20)  : menu_Printf(menu2, "@c[\x70] Error en la
                recepción del control automático de ganancia RXAGC
                @c[\x70]\n");
              break;
    case(21)  : menu_Printf(menu2, "@c[\x70] Falla en el tiempo
                de amarre de lazo del bit del demodulador
                @c[\x70]\n");
              break;
    case(22)  : menu_Printf(menu2, "@c[\x70] Falla en el amarre
                de lazo en la portadora del demodulador
                @c[\x70]\n");
              break;
    case(23)  : menu_Printf(menu2, "@c[\x70] Falla de lazo en la
                sincronización del decodificador del demodulador
                @c[\x70]\n");
              break;
    case(24) : menu_Printf(menu2, "@c[\x70] Error en la
               adquisición del demodulador @c[\x70]\n");
              break;
    case(25) : menu_Printf(menu2, "@c[\x70] Error en el rango de
                recepción de la portadora VCXO @c[\x70]\n");
```

```
                break;

        case(26) : menu_Printf(menu2, "@c[\x70] Error en el rango de
                   transmisión de la portadora @c[\x70]\n");
                break;
        case(27) : menu_Printf(menu2,  "@c[\x70] Error de bit en el
                   rango de tiempo de recepción VCXO @c[\x70]\n");
                break;
            }
          }
         i = i + 1;
      } while(i <= 31);
      menu_Printf(menu2, "@c[\x74]\n Oprima una tecla para continuar
                  @c[\x70]");
    }
    menu_Flush(menu2);
    sed2=sed_Open(menu2);
    sed_SetColors(sed2, '\x0e', '\x74', '\xe4');
    sed_SetBorder(sed2,bd_title);
    if (inst == 7) {
      sed_SetPosition(sed2, 5, 7); }
    else {
      sed_SetPosition(sed2, 5, 5);
    }
/*    sed_BorderPrompt(sed2," Oprima una tecla para continuar ");  */
    if (inst==7) {
       sed_SetBorderTitle(sed2,"   Reporte de Errores (FL) ");
    }
    if (inst==12) {
       sed_SetBorderTitle(sed2,"   Errores Instantáneos (ST) ");
    }
    sed_Push(sed2);
    sed_Repaint(sed2);
    sed_Go(sed2);
    while(kbhit()==0);
    c=getch();
    sed_Pop(sed2);
    menu_Close(menu2);
    sed_Close(sed2);
}

main()
{
 int repite,pregunta;
 int opc,ff,i;

 repite = 0;
 cls();
 pto = 0;
 coparam[0]=1;
 coparam[1]=17;
 coparam[2]=1200;
 coparam[3]=0;
 coparam[4]=1;
 coparam[5]=0;
 coparam[6]=0;
```

```
coparam[7]=0;
coparam[8]=19;
coparam[9]=100;
menu = menu_Open();
menu_Printf(menu, "@p[0,0]@c[\x10]     ");
menu_Printf(menu, "@p[1,0] @c[\x1e] Número del puerto donde está
            conectado el modem : @c[\x07]@fd[####]@c[\x10]       ",
            &pto, &int_funcs,"       Teclee el número del puerto");
menu_Printf(menu, "@p[3,1] @c[\x1e] AL Reporte de Alarma @c[\x10]
            @c[\x1e]  AL =  @c[\x10]");
menu_Printf(menu, "@p[4,1] @c[\x1e] BE Umbral de la tasa de error de
            bit @c[\x10] @c[\x1e]  BE =  @c[\x10]");
menu_Printf(menu, "@p[5,1] @c[\x1e] BR Velocidad de transmision
            @c[\x10] @c[\x1e]  BR =  @c[\x10]");
menu_Printf(menu, "@p[6,1] @c[\x1e] CF Limpiar el registro de error
            @c[\x10] @c[\x1e]  CF =  @c[\x10]");
menu_Printf(menu, "@p[7,1] @c[\x1e] EE Habilita/deshabilita el
            caracter de eco@c[\x10] @c[\x1e]  EE =  @c[\x10]");
menu_Printf(menu, "@p[8,1] @c[\x1e] EM Habilitacion del modulador
            @c[\x10] @c[\x1e]  EM =  @c[\x10]");
menu_Printf(menu, "@p[9,1] @c[\x1e] FL Pregunta por el registro de
            errores @c[\x10] @c[\x1e]  FL    @c[\x10]");
menu_Printf(menu, "@p[10,1] @c[\x1e] LB Modo 'loop back' @c[\x10]
            @c[\x1e]  LB =  @c[\x10]");
menu_Printf(menu, "@p[11,1] @c[\x1e] PC Forza la transmision de la
            portadora @c[\x10] @c[\x1e]  PC =  @c[\x10]");
menu_Printf(menu, "@p[12,1] @c[\x1e] RB Lectura de la tasa de error
            de bit @c[\x10] @c[\x1e]  RB =  @c[\x10]");
menu_Printf(menu, "@p[13,1] @c[\x1e] RE Reestablece el sistema
            @c[\x10] @c[\x1e] RE @c[\x10]");
menu_Printf(menu, "@p[14,1] @c[\x1e] ST Pregunta por fallas actuales
            @c[\x10] @c[\x1e] ST @c[\x10]");
menu_Printf(menu, "@p[15,1] @c[\x1e] TP Transmision del nivel de
            potencia @c[\x10] @c[\x1e]  TP =  @c[\x10]");
menu_Printf(menu, "@p[16,1] @c[\x1e] Salir del programa @c[\x10] ");
menu_Printf(menu, "@p[18,1]");
menu_Flush(menu);
sed = sed_Open(menu);
sed_SetColors(sed, '\x1e', '\x1e', '\xe0');
sed_SetPosition(sed, 3, 1);
sed_SetBorder(sed,bd_std);
sed_SetBorderTitle(sed," SISTEMA DE CONTROL DE MODEM");
sed_Push(sed);
sed_Repaint(sed);
sed_Go(sed);
DespParam();
while (repite != 14) {
  inst = pop_Menu(" Seleccione el comando a enviar
:",opciones,3,6,14,44,(char)'\x1e',0,bd_title);
  if (inst >= 1 && inst <= 13 && pto >=0 && pto <= 34) {
   switch (inst) {
   /*case (1,2,3,4,5,6,8,9,10,13):*/
     case (1) :
     case (2) :
     case (3) :
     case (4) :
```

```
   case (5) :
   case (6) :
   case (8) :
   case (9) :
   case (10) :
   case (13) :
     opc = pop_Menu(" El parámetro a enviar es :",paramop,10,10,
                    2,28,(char)'\x30',0,bd_title);
     if (opc == 1) {
        param=0;
        menu1 = menu_Open();
        menu_Printf(menu1,"@c[\x4b] Valor del parámetro ? :
                    @c[\xb4]@f[######]@c[\x4b] ",&param,
                    &int_funcs);
        menu_Flush(menu1);
        sed1 = sed_Open(menu1);
        sed_SetColors(sed1, '\x47', '\x4b', '\xb4');
        sed_SetPosition(sed1,13,10);
        sed_SetBorder(sed1,bd_title);
        sed_Push(sed1);
        sed_Repaint(sed1);
        sed_Go(sed1);
        sed_Pop(sed1);
        sed_Close(sed1);
        menu_Close(menu1);
        pregunta = 0;
     }
     else {
        param = '?';
        pregunta = 1;
     }
     break;
   case (7) :
   case (12):
     param = '?';
     pregunta = 1;
     break;
   case (11) :
     param = 0;
     pregunta = 0;
     break;
}
ff = funamod(inst,param,pto,coparam,status);
if (ff == 0) {
   if (pregunta != 0) {
     DespParam();
     if ((inst==7)||(inst==12)) {
       DespFLST();
     }
   pregunta = 0;
   }
   repite = 1;
}
else {
 switch (ff) {
   case (1):
```

```
  pop_Prompt(" Error 1 \n @c[\x31]Oprima ESC para continuar
              @c[\x34]",20,10,2,56,(char)'\x34',bd_2);
  break;
case (2):
  pop_Prompt(" Error 2   Error en el eco.\n @c[\x31]Oprima
              ESC para continuar@c[\x34]",20,10,2,56,
              (char)'\x34',bd_2);
  break;
case (3):
  pop_Prompt(" Error 3   Parámetro de configuración inválido.
              \n @c[\x31]Oprima ESC para continuar@c[\x34]",
              20,10,2,56,(char)'\x34',bd_2);
  break;
case (4):
  pop_Prompt(" Error 4   El puerto ya estaba abierto.\n
              @c[\x31]Oprima ESC para continuar@c[\x34]",20,
              10,2,56,(char)'\x34',bd_2);
  break;
case (5):
  pop_Prompt(" Error 5   Memoria insuficiente en el buffer
              del puerto.\n @c[\x31]Oprima ESC para continuar
              @c[\x34]",20,10,2,56,(char)'\x34',bd_2);
  break;
case (6):
  pop_Prompt(" Error 6   La dirección del UART es inválida.\n
              @c[\x31]Oprima ESC para continuar@c[\x34]",20,
              10,2,56,(char)'\x34',bd_2);
  break;
case (7):
  pop_Prompt(" Error 7   El tamaño del buffer es inválido.\n
              @c[\x31]Oprima ESC para continuar@c[\x34]",20,
              10,2,56,(char)'\x34',bd_2);
  break;
case (8):
  pop_Prompt(" Error 8   Problemas con el Hardware del
              puerto.\n@c[\x31]Oprima ESC para continuar
              @c[\x34]",20,10,2,56,(char)'\x34',bd_2);
  break;
case (9):
  pop_Prompt(" Error 9   Memoria insuficiente para
              transmitir.\n@c[\x31]Oprima ESC para continuar
              @c[\x34]",20,10,2,56,(char)'\x34',bd_2);
  break;
case (10):
  pop_Prompt(" Error 10  Error en el número de bytes
              transmitidos.\n@c[\x31]Oprima ESC para
              continuar@c[\x34]",20,10,2,56,
              (char)'\x34',bd_2);
  break;
case (11):
  pop_Prompt(" Error 11  Problemas al cerrar el puerto.\n
              @c[\x31]Oprima ESC para continuar@c[\x34]",
              20,10,2,56,(char)'\x34',bd_2);
  break;
case (12):
  pop_Prompt(" Error 12  El modem no regresó el valor
```

```
                    correspondiente.\n@c[\x31]Oprima ESC para
                    continuar@c[\x34]",20,10,2,56,
                    (char)'\x34',bd_2);
        break;
      case (13):
        pop_Prompt(" Error 13  La instrucción no acepta incógnita.
                    \n@c[\x31]Oprima ESC para continuar@c[\x34]",
                    20,10,2,56,(char)'\x34',bd_2);
        break;
      case (14):
        pop_Prompt(" Error 14  No encontró el número en la cadena.
                    \n@c[\x31]Oprima ESC para continuar@c[\x34]",
                    20,10,2,56,(char)'\x34',bd_2);
        break;
      case (15):
        pop_Prompt(" Error 15  El modem no contesta.\n @c[\x31]
                    Oprima ESC para continuar@c[\x34]",20,
                    10,2,56,(char)'\x34',bd_2);
        break;
    }
   }
  }
  else {
   if (inst == 14) {
     repite = 14;
   }
  }
 }
 sed_Pop(sed);
 menu_Close(menu);
 sed_Close(sed);
}
```

COMANDOS DE COMPILACION

```
PROACMO1.MAK :
#
# Program: Proacmo1
#

.c.obj:
    cl -c  -Wl -Gs -Za -Zi -AM $*.c

OTHER_LIBS=gfcm.lib+m5mcscap.lib

proacmo1.obj : proacmo1.c
abrepto.obj : abrepto.c
cda.obj : cda.c
cierrapt.obj : cierrapt.c
comunica.obj : comunica.c
enviact.obj : enviact.c
funamod.obj : funamod.c
inscinc.obj : inscinc.c
instcpa.obj : instcpa.c
instspa.obj : instspa.c
leeyesc.obj : leeyesc.c
pren_eco.obj : pren_eco.c
rebecpar.obj : rebecpar.c
rec_flst.obj : rec_flst.c
recibect.obj : recibect.c
valins.obj : valins.c
verlon.obj : verlon.c
verpar.obj : verpar.c

Proacmo1.exe : proacmo1.obj abrepto.obj cda.obj cierrapt.obj
               comunica.obj enviact.obj funamod.obj inscinc.obj
               instcpa.obj instspa.obj leeyesc.obj pren_eco.obj
               rebecpar.obj rec_flst.obj recibect.obj valins.obj
               verlon.obj verpar.obj
    link @Proacmo1.lnk /NOI $(LDFLAGS);
```

PROACMO1.LNK :

```
proacmo1.obj+
abrepto.obj+
cda.obj+
cierrapt.obj+
comunica.obj+
enviact.obj+
funamod.obj+
inscinc.obj+
instcpa.obj+
instspa.obj+
leeyesc.obj+
pren_eco.obj+
rebecpar.obj+
rec_flst.obj+
recibect.obj+
valins.obj+
verlon.obj+
verpar.obj /CO
Proacmo1.exe
Proacmo1.map
gfcm.lib+
m5mcscap.lib
```

10 APENDICE E

10.1 SEUDOCODIGO DEL PROGRAMA QUE SIMULA AL MODEM

Seudocódigo:

```
/*   Lista de comandos almacenados en el modem
     y su valor de default. */

     AL = 1
     BE = 17
     BR = 1200
     EE = 1
     EM = 0
     LB = 0
     PC = 0
     TP = 100

/*   Al dar reset, el modem toma los siguientes
     valores de default. */

     ALD = 1
     BED = 17
     BRD = 1200
     EED = 1
     EMD = 0
     LBD = 0
     PCD = 0
     TPD = 100

/*   Las siguientes instrucciones reciben información. */

     salida = 0
     f = lee_config ("confmod.prt")
/*       Se leyó la configuración. */
     f = abre (pto,ASINOUT)
/*       Se abrió el puerto. */
     WHILE (salida = 0) DO
       r = 0
       vaux = NULL
       WHILE (vaux =/= toascii 13) DO
         b = 0
         WHILE (b = 0) DO
/*         Espera mensaje. */
           b = recibe (pto,buffaux,EE)
         ENDWHILE
         buff[r] = buffaux[0]
            vaux = buffaux[0]
               r = r + 1
         IF (EE = 1) THEN
            t = envía (pto,buffaux,EE)
         ENDIF
       ENDWHILE
/*     Se recibió el mensaje. */
       escribe El mensaje es:
/*     Análisis del comando. */
       aux[0] = buff[1]
```

```
aux[1] = buff[2]
aux[2] = NULL

EN CASO DE QUE (aux[0],aux[1] = AL)
   IF (buff = '?') THEN
      escribe (comando AL = buff)
      buff[3] = toascii(AL)
      f = envia(pto,buff,NULL,'>')
   ELSE
     escribe "Desea modificar el valor de AL"
     AL = buff[4]
   ENDIF
ENDCASE

EN CASO DE QUE (aux[0],aux[1] = BE)
    IF (buff = '?') THEN
      escribe (comando BE = buff)
      buff[3] = toascii(BE)
      f = envia(pto,buff,NULL,'>')
   ELSE
      escribe "Desea modificar el valor de BE"
     BE = buff[4]
   ENDIF
ENDCASE

EN CASO DE QUE (aux[0],aux[1] = BR)
    IF (buff = '?') THEN
      escribe (comando BE = buff)
     IF (buff = buff[0],buff[1],buff[2],buff[3]) THEN
       buff[3] = BR[0]
       buff[4] = BR[1]
       buff[5] = BR[2]
       buff[6] = BR[3]
    ELSE
       buff[3] = BR[0]
       buff[4] = BR[1]
       buff[5] = BR[2]
    ENDIF
       f = envia(pto,buff,NULL,'>')
   ELSE
      escribe "Desea modificar el valor de BR"
      IF (buff[7]>=48 & buff[7]<=57) THEN
        BR[0] = buff[4]
        BR[1] = buff[5]
        BR[2] = buff[6]
        BR[3] = buff[7]
        BR[4] = NULL
     ELSE
        BR[0] = buff[4]
        BR[1] = buff[5]
        BR[2] = buff[6]
        BR[3] = NULL
     ENDIF
    BR = buff[4]
    ENDIF
ENDCASE
```

```
EN CASO DE QUE (aux[0],aux[1] = CF)
    IF (buff = '?') THEN
      escribe (comando CF = buff)
      buff[3] = CF
      f = envia(pto,buff,NULL,'>')
   ELSE
      IF (buff[5]>=48 & buff[6]<=57){
         CF = (buff[4],buff[5])
      ELSE
         CF = buff[4]
      ENDIF
   ENDIF
ENDCASE

EN CASO DE QUE (aux[0],aux[1] = EE)
    IF (buff = '?') THEN
      escribe (comando EE = buff)
      buff[3] = toascii(EE)
      f = envia(pto,buff,NULL,'>')
   ELSE
     escribe "Desea modificar el valor de EE"
     EE = buff[4]
   ENDIF
ENDCASE

EN CASO DE QUE (aux[0],aux[1] = EM)
    IF (buff = '?') THEN
      escribe (comando EM = buff)
      buff[3] = toascii(EM)
      f = envia(pto,buff,NULL,'>')
   ELSE
     escribe "Desea modificar el valor de EM"
     EM = buff[4]
   ENDIF
ENDCASE

EN CASO DE QUE (aux[0],aux[1] = FL)
    IF (buff = '?') THEN
      escribe (comando FL = buff)
      buff[3] = FL[0]
      buff[4] = FL[1]
      buff[5] = FL[2]
      buff[6] = FL[3]
      f = envia(pto,buff,NULL,'>')
   ENDIF
ENDCASE

EN CASO DE QUE (aux[0],aux[1] = LB)
    IF (buff = '?') THEN
      escribe (comando LB = buff)
      buff[3] = toascii(LB)
      f = envia(pto,buff,NULL,'>')
   ELSE
     escribe "Desea modificar el valor de LB"
     LB = buff[4]
```

```
   ENDIF
ENDCASE

EN CASO DE QUE (aux[0],aux[1] = PC)
    IF (buff = '?') THEN
      escribe (comando PC = buff)
      buff[3] = toascii(PC)
      f = envia(pto,buff,NULL,'>')
   ELSE
     escribe "Desea modificar el valor de LB"
     PC = buff[4]
   ENDIF
ENDCASE


EN CASO DE QUE (aux[0],aux[1] = RB)
    IF (buff = '?') THEN
      escribe (comando RB = buff)
      buff[3] = toascii(RB)
      f = envia(pto,buff,NULL,'>')
   ENDIF
ENDCASE

EN CASO DE QUE (aux[0],aux[1] = RE)

   AL = ALD
   BE = BED
   BR = BRD
   EE = EED
   EM = EMD
   LB = LBD
   PC = PCD
   TP = TPD

ENDCASE

EN CASO DE QUE (aux[0],aux[1] = ST)
    IF (buff = '?') THEN
      escribe (comando ST = buff)
      buff[3] = ST[0]
      buff[4] = ST[1]
      buff[5] = ST[2]
      buff[6] = ST[3]
      f = envia(pto,buff,NULL,'>')
   ENDIF
ENDCASE


EN CASO DE QUE (aux[0],aux[1] = TP)
    IF (buff = '?') THEN
      escribe (comando TP = buff)
      buff[3] = TP[0]
      buff[4] = TP[1]
      buff[5] = TP[2]
      f = envia(pto,buff,NULL,'>')
   ELSE
```

```
        escribe "Desea modificar el valor de TP"
        TP[0] = buff[4]
        TP[1] = buff[5]
        TP[2] = buff[6]
        TP[3] = NULL
      ENDIF
   ENDCASE
    salida = 1
ENDWHILE
 f = cierra(pto)
 END
```

10.2 PROGRAMA DEL SIMULADOR DEL MODEM

```
/* Programa que simula el funcionamiento del modem. */

#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <cscape.h>
#include <popdec1.h>
#include "comunica.h"

main()
{
 int
AL=1,Be=1,E=7,EE=1,EM=0,ALD=1,BD=1,ED=7,EED=1,EMD=0,l,r,b,f,pto=COM1;
 int   LB=0,LBD=0,PC=0,PCD=0,Rb=1,BB=9;
 int   salida,ch;
 char  BRR[5],BRD[5],CF='\0',FL[5],ST[5],TP[5];
 char  CFD='\0',FLD[5],STD[5],TPD[5],t;
 char  buff[30],buffaux[30],aux[3],message[150];
 char  *st,*apbuff,*apconf,vaux;
 unsigned long i;
 menu_type menu;
 sed_type sed;

 strcpy(BRD,"1200\0");
 strcpy(BRR,"1200\0");
 strcpy(FL,"5208\0");
 strcpy(FLD,"5208\0");
 strcpy(ST,"13\0");
 strcpy(STD,"13\0");
 strcpy(TP,"100\0");
 strcpy(TPD,"100\0");
 salida = 0;
 f=lee_config("confmod.prt");
 /*printf(" Se leyo la configuracion \n");*/
 f = abre ( pto, ASINOUT );
 /*printf(" Se abrio el puerto \n");*/
 while ((salida==0)&&(kbhit()==0)) {
    r=0;
    cls();
    sprintf(message,"");
    menu = menu_Open();
    menu_Printf(menu, "@p[0,0]@c[\x10]     ");
    menu_Printf(menu, "@p[1,1] @c[\x1e] AL Reporte de Alarma @c[\x10]
                @c[\x1e]  AL =  @c[\x1b]%d@c[\x10]          ",AL);
    menu_Printf(menu, "@p[2,1] @c[\x1e] BE Umbral de la Tasa de Bits
                Erróneos      @c[\x10]            @c[\x1e]  BE =
                @c[\x1b]%d%d@c[\x10]",Be,E);
    menu_Printf(menu, "@p[3,1] @c[\x1e] BR Velocidad de Transmisión
                @c[\x10]            @c[\x1e]  BR =
                @c[\x1b]%s@c[\x10]",BRR);
```

```
menu_Printf(menu, "@p[4,1] @c[\x1e] CF Limpiar el Registro de
            Errores          @c[\x10]          @c[\x1e]  CF =
            @c[\x1b]%s@c[\x10]",CF);
menu_Printf(menu, "@p[5,1] @c[\x1e] EE Habilita/Deshabilita el
            Eco              @c[\x10]          @c[\x1e]  EE =
            @c[\x1b]%d@c[\x10]",EE);
menu_Printf(menu, "@p[6,1] @c[\x1e] EM Habilitación del Modulador
            @c[\x10]          @c[\x1e]  EM =
            @c[\x1b]%d@c[\x10]",EM);
menu_Printf(menu, "@p[7,1] @c[\x1e] FL Pregunta por el Registro
            de Errores    @c[\x10]          @c[\x1e]  FL =
            @c[\x1b]%s@c[\x10]",FL);
menu_Printf(menu, "@p[8,1] @c[\x1e] LB Modo 'loop back'@c[\x10]
            @c[\x1e]  LB =  @c[\x1b]%d@c[\x10]",LB);
menu_Printf(menu, "@p[9,1] @c[\x1e] PC Forza la Transmisión de la
            Portadora   @c[\x10]          @c[\x1e]  PC =
            @c[\x1b]%d@c[\x10]",PC);
menu_Printf(menu, "@p[10,1] @c[\x1e] RB Lectura de la Tasa de
            Bits Erróneos    @c[\x10]          @c[\x1e]  RB =
            @c[\x1b]%d%d@c[\x10]",Rb,BB);
menu_Printf(menu, "@p[11,1] @c[\x1e] RE Reestablece el Sistema
            @c[\x10]          @c[\x1e]  RE   @c[\x1b]@c[\x10]");
menu_Printf(menu, "@p[12,1] @c[\x1e] ST Pregunta por Fallas
            Presentes         @c[\x10]          @c[\x1e]  ST =
            @c[\x1b]%s@c[\x10]",ST);
menu_Printf(menu, "@p[13,1] @c[\x1e] TP Transmisión del Nivel de
            Potencia      @c[\x10]          @c[\x1e]  TP =
            @c[\x1b]%s@c[\x10]",TP);
menu_Printf(menu, "@p[17,1] @c[\x1b] Oprima una tecla para
            salir@c[\x10]");
menu_Printf(menu, "@p[18,1]");
menu_Flush(menu);
sed = sed_Open(menu);
sed_SetColors(sed, '\x1e', '\x1e', '\xe0');
sed_SetPosition(sed, 3, 1);
sed_SetBorder(sed,bd_std);
sed_SetBorderTitle(sed,"                          SISTEMA DE
                  SIMULACION DEL MODEM");
sed_Repaint(sed);
sed_Go(sed);
menu_Close(menu);
sed_Close(sed);
sprintf(buff,"");
vaux = '\0';
pop_Message(" Se espera el comando ",19,4,1,22,(char)'\xe1',
            bd_1);
while ((vaux != toascii(13))&&(kbhit()==0)){
   b=0;
   while ((b==0)&&(kbhit()==0)) {
    sprintf(buffaux,"");
    b=recibe(pto,buffaux,1,0);
   }
   buff[r]=buffaux[0];
   vaux=buffaux[0];
   r=r+1;
   if ( EE == 1 ) {
```

```
      t = envia ( pto, buffaux, 1, 0 );
    ]
}
if (kbhit()==0) {
  pop_Message(NULL,19,4,1,22,(char)'\x1e',bd_1);
  buff[r]='\0';
  strcpy(message," Se recibió el comando : \n     \0");
  strcat(message,buff);
  aux[0] = buff[1];
  aux[1] = buff[2];
  aux[3] = '\0';
  switch (aux[0]) {
   case 'A' :
      st = strstr(buff,"?");
      if (st != 0) {
         /*printf(" Comando %s  \n",buff);*/
         strcpy(buff,"AL ?\r\n>");
         buff[3]=toascii(AL+'0');
         /*printf(" Buff = %s",buff);*/
         f = envia(pto,buff,0,'>');
      }
      else {
         strcat(message,"\n\n Desea modificar el valor de
                AL\0");
         /*printf(" Desea modificar el valor de AL \n ");*/
         AL=buff[4]-'0';
         /*printf(" AL = %d \n",AL) ;*/
      }
      break;
   case 'B' :
  switch(aux[1]) {
     case 'E' :
            st = strstr(buff,"?");
            if (st != 0) {
            /*printf(" Comando %s \n",buff);*/
            strcpy(buff,"BE ##\r\n>");
            buff[3]=toascii(Be+'0');
            buff[4]=toascii(E+'0');
            f=envia(pto,buff,0,'>');
            }
            else {
            strcat(message,"\n\n Desea modificar el valor de
                   BE\0");
            /*printf(" Desea modificar el valor de BE \n");*/
            Be=buff[4]-'0';
            E=buff[5]-'0';
            /*printf(" BE = %d X E -%d \n",Be,E);*/
            }
            break;
     case 'R' :
            st = strstr(buff,"?");
            if (st != 0) {
            /*printf(" Comando %s \n",buff);*/
            strcpy(buff,"BR ####\r\n>");
            l=strlen(BRR);
            if (l=4) {
```

```
            buff[3]=BRR[0];
            buff[4]=BRR[1];
            buff[5]=BRR[2];
            buff[6]=BRR[3];
         }
         else {
            buff[3]=BRR[0];
            buff[4]=BRR[1];
            buff[5]=BRR[2];
         }
         f=envia(pto,buff,0,'>');
         }
         else {
         strcat(message,"\n\n Desea modificar el valor de
                BR\0");
         /*printf(" Desea modificar el valor de BR \n");*/
         if (buff[7]>=48 & buff[7]<=57) {
            BRR[0]=buff[4];
            BRR[1]=buff[5];
            BRR[2]=buff[6];
            BRR[3]=buff[7];
            BRR[4]='\0';
         }
         else {
            BRR[0]=buff[4];
            BRR[1]=buff[5];
            BRR[2]=buff[6];
            BRR[3]='\0';
         }
         /*printf(" BR = %s \n",BRR);*/
         }
         break;
   }
   break;
 case 'C' :
st = strstr(buff,"?");
if (st != 0) {
   /*printf(" Comando %s \n",buff);*/
   strcpy(buff,"CF #\n\r>");
   buff[3]=CF;
   f=envia(pto,buff,0,'>');
}
else {
   /*printf("%s,buff",buff);*/
   if (buff[5]>=48 & buff[6]<=57){
   CF=(buff[4],buff[5]);
   }
   else {
   CF=buff[4];
   }
}
break;
 case  'E' :
switch(aux[1]) {
   case 'E' :
   st = strstr(buff,"?");
```

```
    if (st != 0) {
       /*printf(" Comando %s \n",buff);*/
       strcpy(buff,"EE #\n\r>");
       buff[3]=toascii(EE+'0');
       f=envia(pto,buff,0,'>');
    }
    else {
       strcat(message,"\n\n Desea modificar el valor de EE\0");
       /*printf(" Desea modificar el valor de EE \n");*/
       EE=buff[4]-'0';
       /*printf(" EE = %d \n",EE);*/
    }
    break;
    case 'M' :
    st = strstr(buff,"?");
    if (st != 0) {
       /*printf(" Comando %s \n",buff);*/
       strcpy(buff,"EM #\n\r>");
       buff[3]=toascii(EM+'0');
       f=envia(pto,buff,0,'>');
    }
    else {
       strcat(message,"\n\n Desea modificar el valor de EM\0");
       /*printf(" Desea modificar el valor de EM \n");*/
       EM=buff[4]-'0';
       /*printf(" EM = %d \n",EM);*/
    }
    break;
}
break;
 case 'F' :
st = strstr(buff,"?");
if (st != 0) {
   /*printf(" Comando %s",buff);*/
   strcpy(buff,"FL ####\n\r>");
   buff[3]=FL[0];
   buff[4]=FL[1];
   buff[5]=FL[2];
   buff[6]=FL[3];
   f=envia(pto,buff,0,'>');
}
break;
 case 'L' :
st = strstr(buff,"?");
if (st != 0) {
   /*printf(" Comando %s",buff);*/
   strcpy(buff,"LB #\n\r>");
   buff[3]=toascii(LB+'0');
   f=envia(pto,buff,0,'>');
}
else {
   strcat(message,"\n\n Desea modificar el valor de LB\0");
   /*printf(" Desea cambiar el valor de LB \n");*/
   LB=buff[4]-'0';
   /*printf(" LB = %d \n",LB);*/
}
```

```
break;
 case 'P' :
st = strstr(buff,"?");
if (st != 0) {
   /*printf(" Comando %s",buff);*/
   buff[3]=toascii(PC+'0');
   f=envia(pto,buff,0,'>');
}
else {
   strcat(message,"\n\n Desea modificar el valor de PC\0");
   /*printf(" Desea cambiar el valor de PC \n");*/
   PC=buff[4]-'0';
   /*printf(" PC = \n",PC);*/
}
break;
 case 'R' :
switch(aux[1]) {
   case 'B' :
          st = strstr(buff,"?");
          if (st != 0) {
          /*printf(" Comando %s \n",buff);*/
          strcpy(buff,"RB ##\r\n>");
          buff[3]=toascii(Rb+'0');
          buff[4]=toascii(BB+'0');
          f=envia(pto,buff,0,'>');
          }
          break;
   case 'E' :
    /*printf(" Comado %s",buff);*/
    AL=ALD;
    Be=BD;
    E=ED;
    CF=CFD;
    EE=EED;
    EM=EMD;
    strcpy(FL,FLD);
    strcpy(ST,STD);
    LB=LBD;
    PC=PCD;
    Rb=Rb;
    BB=BB;
    strcpy(TP,TPD);
    strcpy(BRR,BRD);
    break;
}
break;
 case 'S' :
st = strstr(buff,"?");
if (st != 0) {
   /*printf(" Comando %s",buff);*/
   strcpy(buff,"ST ####\n\r>");
   buff[3]=ST[0];
   buff[4]=ST[1];
   buff[5]=ST[2];
   buff[6]=ST[3];
   f=envia(pto,buff,0,'>');
```

```
      }
      break;
       case 'T' :
      st = strstr(buff,"?");
      if (st != 0) {
         /*printf(" Comando %s",buff);*/
         strcpy(buff,"TP ###\n\r>");
         buff[3]=TP[0];
         buff[4]=TP[1];
         if (TP[2] == toascii(13)) {
         buff[5] = '\0';
         }
         else {
         buff[5] = TP[2];
         }
         f=envia(pto,buff,0,'>');
      }
      else {
         strcat(message,"\n\n Desea modificar el valor de TP\0");
         /*printf(" Desea cambiar el valor de TP \n");*/
         TP[0] = buff[4];
         TP[1] = buff[5];
         if (buff[6] == toascii(13)) {
         TP[2] = '\0';
         }
         else {
         TP[2] = buff[6];
         TP[3] = '\0';
         }
         /*printf(" TP = %s \n",TP);*/
      }
      break;
       default :
      salida=1;
      break;
    }
     pop_Message(message,-1,-1,4,40,(char)'\x0e',bd_1);
     for(i=1;i<200000;i++);
     pop_Message(NULL,-1,-1,4,40,(char)'\x0e',bd_1);
    }
 }
 cls();
 printf(" F I N \n");
 f = cierra(pto);
}
```

COMANDOS DE COMPILACION

```
SIMUMOD.MAK :
#
# Program: Simumod
#

.c.obj:
    cl -c  -Wl -Ze -Zi -AM $*.c

OTHER_LIBS=gfcm.lib m5mcscap.lib


simumod.obj : simumod.c
comunica.obj : comunica.c

Simumod.exe : simumod.obj comunica.obj
    link @Simumod.lnk /NOI $(LDFLAGS);



SIMUMOD.LNK :

simumod.obj+
comunica.obj /CO
Simumod.exe
Simumod.map
gfcm.lib+
C:\COMPILER\LIB\m5mcscap.lib
```