Si se considera en las líneas telefónicas una tasa de error de 10^{-5} con el método de detección llevado a cabo este resultado baja a 10^{-10}. Cabe destacar que empleando un código de redundancia ciclíca (CRC) para la detección dicha tasa se reduce hasta 10^{-14}. Los modems empleados en este trabajo cuentan con un protocolo interno para la detección y correción de errores (MNP: *Microcom Networking Protocol*). En dicho protocolo se incluye un CRC para mejorar la seguridad de los datos lo cual representa una protección adicional al método de detección presentado. La corrección de errores se realiza por medio de retransmisiones.

El empleo de líneas telefónicas como medio de transmisión en dicha red, no sólo optimiza los canales de comunicación, sino que también la hace extensible a cualquier punto donde exista una línea telefónica. Sin embargo, esto impide la comunicación simultánea a los diferentes puntos de medición. La comunicación simultánea requiere que en el monitor central se conecte un multiplexor con los modems correspondientes de cada terminal y canales dedicados por cada enlace, los cuales no están disponibles actualmente.

Las características de la red fueron tratadas exhaustivamente en los capítulos anteriores. El protocolo de comunicación propuesto puede ser modificado para adaptarlo a cualquier otra aplicación.

Para el proceso de sincronización de las terminales se propuso el empleo de una tarjeta de sincronía en cada PC las cuales, compartiendo los recursos del mismo protocolo de comunicación, evitarían la compra de costosos sistemas de sincronización.

En el estudio comparativo llevado a cabo en el capítulo V, se concluye el inconveniente de implementar el estándar PROFIBUS para equipos registradores de disturbios. La principal limitación es la longitud del bus ya que no se se permite conexión entre dispositivos a más de 24 Km., empleando fibra optica como medio de transmisión. Además esto también requeriría del uso de *hardware* y *software* especializado, aumentando su complejidad y costo.

6.2 APORTACIONES

Se presenta el diseño de una red de comunicaciones orientada a la monitorización de disturbios electromecánicos en sistemas de potencia. Esta será utilizada en la Red de Distribución del Área Noreste de la Comisión Federal de Electricidad.

Se define y documenta el protocolo de comunicación para el empleo específico del sistema de monitoreo. Se realiza la programación del protocolo utilizando los puertos seriales de la PC como capa física. El programa de comunicaciones permite adaptarse a otro tipo de aplicación requerida.

El programa permite al operador local o remoto realizar cambios de configuración en la tarjeta tales como el umbral de detección, el tiempo de grabación del disturbio y la alta o baja de los canales en operación. Esto evita que el operador local se traslade al nodo remoto para realizar dichos cambios por razones de mantenimiento en las líneas. También se pueden transmitir mensajes de texto entre operadores.

Se ofrece un panorama general sobre el estándar PROFIBUS, definido como un estándar abierto de fieldbus empleado en procesos de automatización.

La implementación de la red propuesta constituye una aportación importante a la tecnología nacional ya que sustituye el empleo de costosas tecnologías extranjeras, liberando al usuario de la dependencia tecnológica en lo que se refiere al mantenimiento y a la modificación de los protocolos.

6.3 RECOMENDACIONES PARA TRABAJOS FUTUROS

- Rediseñar el protocolo propuesto empleando ventanas deslizantes para mejorar la eficiencia del canal de comunicaciones.

- Diseñar e implementar una red tipo estrella que permita un registro paralelo en todas las terminales al presentarse un disturbio. Esto con la finalidad de analizar el comportamiento general del sistema eléctrico en tiempo real.

- Implementar un algoritmo de autocorreción de errores cuando se disponga de una comunicación simultánea entre las diferentes estaciones. Esto con el fin de dividir las tareas de procesamiento en el monitor central.

- Realización de otros estudios en el área de redes de comunicaciones, que proporcionen mejores ventajas que el estándar PROFIBUS.

REFERENCIAS

[1] Marco Antonio Escobar, José Antonio de la O "Red de Comunicaciones para la Monitorización de Fallas en Líneas de Transmisión". Julio 1997. RVP 97, Acapulco, Guerrero, México. Pags. 251-256.

[2] Fred Halsall, "Data Comunications, Computer Networks and Open Systems" Addison-Wesley publishing company, Inc. 1992.

[3] John D. Spragins with Joseph L. Hammond and Krysztof Pawlikowski, "Telecomunicatons Protocols and Design" Addison-Wesley publishing company, Inc. July 1992.

[4] Peter Norton, Peter Aitken, Richard Wilton, "PC Programers's Biblie" Microsoft Press 1985.

[5] Joe Campbell "C programer's Guide to Serial Comunications" Sams Publishing a division of Prentice Hall. 1994.

[6] Peter W. Gofton "Mastering serial Comunications", SYBEX Inc. 1986.

[7] Andrew S. Tanenbaum, "Computers Networks" Prentice Hall, 1991.

[8] Robert G.Winch "Telecomunication Transmission Systems", McGraw hill 1993.

[9] Nestor Gonzalez Sainz "Comunicaciones y redes de procesamiento de datos", Mc Graw Hill . 1985.

[10] Josep Balcells, Francesc Daura, Rafael Esparza y Rámon Pallás . "Interferencias Electromagnéticas en Sistemas Electrónicos", Ed. Alfaomega marcombo. 1992.

[11] R. Jay Murphy R. O. Burnett, Jr. "Protective Relaying Conference" Georgia Institute of Technology Atlanta, Georgia. May 4-6, 1994.

[12] A. G. Phadke, Chairman, B. Pickett, Vice Chairman, M. Adamiak, M. Begovic, G. Benmouyal, R. O. Burnett, "Sincronized Sampling and Phasor Measurements for Relaying and Control" Working Group H-7 of the Relaying Channels Subcommite of the IEEE Power System Relaying Committee.

[13] Arun G. Phadke, James S. Thorp "Computer Relaying for Power Systems" SRP 1988.

[14] A.G. Phadke "Recent Advances in Monitoring, Protection and Control of Power Systems. virginia Tech, Blacksburg, VA, USA.

[15] TrueTime precision Timing Producs. GPS and Synchrnized Clocks. March 1994.

[16] Technical PROFIBUS (Process Field Bus) Overview. Organization PROFIBUS International. March 97.

[17] Measurement and Control PROFIBUS (Process Field Bus) DIN 19245 Part 1 and Part 2. Manuscript English Versión of German Standard April 1991.

[18] Siemens SINEC Industrial Communications Networks. Catalog 1K 10. 1994.

[19] Direct LINK. Published for Customers and Business Associates of S-S Technologies Inc., Winter 1997.

[20] Registrador de fallas REFA-02, Kitron. Instructivo de operación. Mexaltec S.A. de C.V. Febrero 92.

[21] Omnipotencihorimetro OPH-O3/3 v9, Kitron. Instructivo de operacion. GPI Mexicana de alta Tecnologia, S.A. de C.V. Abril 97.

[22] McNamar, J.E. Technical Aspects of Data Comunications. Digital Press, 1978.

[23] Antonio Alabau Muñoz, "Teleinformatica y Redes de Computadores" Marcombo , S.A. Barcelona, España 1987.

APÉNDICE A

REGISTROS DEL UART

Este apéndice describe los 10 registros del circuito integrado 8250 UART, el cual contempla las mismas características de operación con los circuitos 16450 y 16550 cuyas tecnologías sólo difieren en velocidad de procesamiento. Dichos registros son accesibles mediante lecturas o escrituras sobre las direcciones de memoria entrada/salida 3F8-3FF (COM1) ó 2F8-2FF (COM2). Los registros que se pueden leer y escribir se denominan R/W (*read/write*); aquellos que pueden ser sólo para lectura son nombrados como R/O (*read/only*); y aquellos en los que únicamente se puede escribir se listan como W/O (*write/only*).

A.1 Registro de control de la línea (LCR)

El registro LCR (R/W), especifica el formato del caracter y controla el acceso a otros registros. El contenido se muestra en la Tabla A.1.

Tabla A.1 Mapa de bits del LCR

Bits	*Significado*
0	WLS0: Selectores de la longitud de caracter
1	WLS1: Selectores de la longitud de caracter
2	STB: Número de bits de parada
3	PEN: Activación de paridad
4	EPS: Selector de la paridad
5	STICK PARITY: Detector de errores de paridad
6	SET BREAK: Señalizador
7	DLAB: Bit de acceso al cierre del divisor

- Los bits 0 y 1 establecen la longitud de cada caracter. Se requieren al menos 7 bits para los caracteres ASCII.

bit 1	bit 0	Longitud
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

- El bit 2 define el número de bits de parada por caracter; 0 establece un bit parada y 1 establece 2 bits de parada.
- El bit 3 activa la revisión de paridad cuando está en 1, y lo desactiva cuando está en 0.

- El bit 4 selecciona la paridad: 1 si es par y 0 si es impar.
- El bit 5, si está en 1, activa la detección de cualquier error de paridad.
- El bit 6, si es 1, provoca que el módem transmita una señal de interrupción continuamente. Esta señal se detiene cuando el bit está en 0.
- El bit 7, cuando está en 1, permite el acceso a los *registros del divisor de cierre* durante la operación de lectura o de escritura. Si este bit está en 0, activa el acceso a los registros del buffer de recepción, de sostenimiento de transmisión y de habilitación de interrupciones. Para configurar el generador de baudios se definen los siguientes dos registros.

A.2 Registro de cierre del octeto divisor menos significativo (DLL)

El registro DLL (R/W) contiene los 8 bits menos significativos del divisor de razón de baudios utilizado para configurar la razón de baudios. El octeto divisor más significativo está contenido en el siguiente registro (DLH). Juntos, ambos registros deben contener el valor equivalente del divisor para generar la razón de baudios deseada.

A.3 Registro de cierre del octeto divisor mas significativo (DLH)

El registro DLH (R/W), contiene los 8 bits mas significativo del divisor de razón de baudios. Los valores del divisor tanto del DLL como del DLH, se introducen cuando el bit del DLAB en el LCR se encuentra en 1. El valor de los divisores correspondiente a cada velocidad se presentan en la Tabla A.2.

Tabla A.2 Divisores de la razón de baudios bajo la referencia de dos relojes

Baudios deseados Velocidad	Divisor Hexadecimal del Reloj de cristal					
	1.8432 Mhz			3.072 Mhz		
	Divisor	DLH	DLL	Divisor	DLH	DLL
300	180	1	80	280	2	80
600	C0	0	C0	140	1	40
1200	60	0	60	A0	0	A0
2400	30	0	30	50	0	50
4800	18	0	18	28	0	28
9600	C	0	C	14	0	14
14400	8	0	8	--	--	--
19200	6	0	6	A	0	A

El valor divisor se obtiene de la siguiente manera:

$$\text{Divisor} = \frac{\text{Frecuencia del reloj maestro del sistema de la PC}}{(16)\ (\text{velocidad deseada en bps})}$$

A.4 Registro de estado de línea (LSR)

El registro LSR de sólo lectura (R/O), con excepción del bit 0 que permite escritura (R/W), proporciona información sobre el estado de la transferencia de datos y de las condiciones de error asociadas. Suponiendo que las interrupciones están previamente activadas por el Registro de Habilitación de Interrupciones (IER), las condiciones señaladas por los bits 0-5 producen una interrupción. El mapa de bits se muestra en la Tabla A.3.

Tabla A.3 Mapa de bits del LSR

Bits	*Significado*
0	DR: Datos Recibidos
1	OE: Error de estancamiento
2	PE: Error de paridad
3	FE: Error de construcción
4	BREAK:Interrupción de desconexión
5	THRE: Registro de sostenimiento de transmisión vacío
6	TSRE: Registro de desplazamiento de transmisión vacío
7	------- No es usado; se mantiene siempre en 0.

- El bit 0 se pone en 1 para indicar que se ha recibido un caracter y que está contenido en el registro buffer de recepción, cuando el caracter es leído la CPU o el control del programa pone este bit en 0.
- El bit 1 es puesto en 1 cuando manifiesta que el caracter del registro del buffer de recepción no fue leído antes de la recepción del siguiente caracter; el caracter anterior fue destruido. Los bits del 1 al 4 se ponen en 0 siempre que se lee el LSR.
- El bit 2 está en 1 para indicar que la paridad del caracter recibido no concuerda con la especificación dada por el EPS (bit 4) del LCR.
- El bit 3 se mantiene en 1 cuando le falta un bit de parada válido al caracter recibido.

- El bit 4 se fija en 1 para indicar que se ha recibido una señal de desconexión (interrupción); éste se manifiesta cuando existe una condición de espaciamiento mayor que un SDU (*Serial Data Unit*).
- El bit 5 se pone en 1 para indicar que el UART está preparado para aceptar un nuevo caracter para transmisión; se pone en 0, cuando se carga el registro de sostenimiento de transmisión.
- El bit 6 se mantiene en 1 para indicar que el último caracter en el registro de desplazamiento de transmisión (TSR) ha sido transmitido; se pone en 0, cuando se transfiere un caracter del registro de sostenimiento de transmisión (TXR) al TSR.

A.5 Registro de control del módem (MCR)

El MCR es un registro de lectura y escritura (R/W) que maneja la interfase con el módem, los bits se definen en la Tabla A.4. Los bits 4, 5 y 6 no se usan, siempre son cero.

Tabla A.4 Mapa de bits del MCR

Bits	*Significado*
0	DTR: Terminal de datos listo
1	RTS: Petición de emisión
2	OUT 1: Salida 1
3	OUT 2: Salida 2
4	LOOP: Ciclo de prueba

- El bit 0 activa la operación del módem cuando está en 1; cuando está en 0, lo desactiva. En este último caso el módem nunca acepta órdenes ni respuestas automáticas y, si está en línea, se desconecta.
- La acción del bit 1 depende del módem. Si está en 1, confirma la petición de envío.
- Los bits 2 y 3 son salidas definidas por el usuario.
- El bit 4, cuando está en 1, activa la característica de "loopback" (verificación por retroalimentación) para prueba de diagnóstico del UART. Cuando el bit está en 0, el módem trabaja en modo normal.

A.6 Registro del estado en el módem (MSR)

El registro MSR (R/O) proporciona el estado actual de las señales de control del módem tal y como se definen en la Tabla A.5. Siempre que el bit 2 o el 3 están en 1 lógico se genera una interrupción de estado del módem. Esta interrupción la activa el registro de habilitación de interrupciones (IER).

Tabla A.5 Mapa de bits del MSR

Bits	*Significado*
0	DCTS: Delta libre para emitir
1	DDSR: Delta listo para operación
2	DRI: Delta rastro del indicador de timbre
3	DRLSD:Delta rastro detector de señal
4	CTS: Listo para emitir
5	DSR: Listo para operación
6	RI: Indicador de timbre
7	RLSD: Señal detectada en la línea de recepción

Los bits del 0 al 3 indican el reporte de algún cambio en el estado de los *pines* del RS-232/V.24. Los bits del 4 al 7 reportan el estado absoluto de las entradas del RS/232/V.24. Las características especificas se describen a continuación.

- Los bits 0 y 1 se mantienen siempre en 0.
- El bit 2, cuando está en 1, avisa que el indicador de timbre (bit 6) ha cambiado de 1 a 0. El bit 2 se pone en 0 cuando lee el MSR.
- El bit 3 se mantiene en 1 para indicar que el bit de la señal detectada en la línea de recepción ha cambiado de estado. El bit 3 se pone en 0 cuando lee el MSR.
- Los bits 4 y 5 se mantienen siempre en 1.
- El bit 6, si está en 1, indica el timbre de una llamada en la línea telefónica.
- El bit 7, cuando está en 1, indica que el módem detecta una señal de timbre en la línea telefónica. Este bit se pone en 0 cuando hay ausencia de la señal del portador.

A.7 Registro buffer de recepción (RXR)

El registro RXR de sólo lectura contiene el caracter que se acaba de recibir. El caracter es recibido serialmente, llegando primero el bit menos significativo (bit 0). Para acceder a este registro , el DLAB (bit 7) del LCR debe ser 0.

A.8 Registro de sostenimiento de transmisión (TXR)

El registro TXR (W/O) contiene el siguiente caracter a ser transmitido serialmente. El bit menos significativo (bit 0) es el que primero se transmite. Para acceder a este registro, el DLAB del LCR debe estar en 0.

A.9 Registro de habilitación de interrupciones (IER)

El registro IER (R/W) activa las fuentes de interrupción para que puedan activar la señal de interrupción saliente. Para acceder a este registro, el DLAB del LCR debe ser 0. Los bits de este registro se muestran en la Tabla A.6. Los bits 4 al 7 no se usan, siempre so cero.

Tabla A.6 Mapa de bits del IER

Bits	*Significado*
0	RxRDY:Interrupción por dato recibido disponible
1	TBE: Interrupción por registro de sostenimiento de transmisión vacío
2	ELSI: Interrupción por estado de recepción de la línea
3	EDSSI: Interrupción por estado de módem

- El bit 0, cuando está en 1, provoca que el UART genere una interrupción siempre que el DR (bit 0) del registro de estado de la línea (LSR) se ponga en 1.
- El bit 1, cuando está en 1, provoca que el UART genere una interrupción siempre que el THRE (bit 5) del LSR se ponga en 1.

- El bit 2, si está en 1, provoca que el UART genere una interrupción siempre que ocurra en el LSR un error de estancamiento, un error de paridad, un error de construcción o una señal de interrupción.
- El bit 3, cuando está en 1, provoca que el UART genere una interrupción siempre que el RI (bit 6) o el RLSD (bit 7) del MSR se ponga en 1.
-

A.10 Registro de identificación de interrupción (IIR)

El registro IIR (R/O) guarda la información que muestra si una interrupción prioritaria está pendiente. Los bits de este registro se definen en la Tabla A.7. Los bits 3 al 7 no se usan, siempre es cero.

Tabla A.7 Mapa de bits del IIR

Bits	*Significado*
0	Interrupción pendiente
1	Identificador de interrupción
2	Identificador de interrupción

- El bit 0, cuando está en 0, indica que ha ocurrido una condición de interrupción.
- Los bits 1 y 2 indentifican la condición de interrupción y están siempre disponibles incluso aunque la capacidad de interrupción no haya sido activada por el registro IER; las condiciones y prioridades se muestran a continuación en la Tabla A.8.

Tabla A.8 Condiciones y prioridades de interrupción

bit 1	bit 2	Prioridad	Fuente de interrupción
1	1	1	- Tipo de error: de estancamiento, de paridad, de construcción o de interrupción por desconexión.
1	0	2	- Datos listos.
0	1	3	- Registro de sostenimiento de transmisión vacío.
0	0	4	- Indicador de timbre o señal detectada en la línea de recepción.

APÉNDICE B

RECOMENDACIÓN V.32 Bis

La recomendación V.32 bis del CCITT, específica las siguientes características:

B.1 Características generales :

a) Enlace punto a punto para línea conmutada o línea dedicada en modo full-duplex.

b) Separación de canales por técnicas de cancelamiento de eco.

c) Modulación QAM para cada canal con transmisión síncrona.

d) Incluye las facilidades de prueba.

e) Implementa las siguientes tasas de señalización de datos:

- 14400, 12000, 9600, y 7200 bit/s Codificación *Trellis*
- 4800 bit/s No codificado

f) Compatibilidad con módem V.32 operando a 9600 o 4800 bps.

g) Envío automático de la tasa de señalización a partir del establecimiento de la conexión.

h) Truncamiento automático de la tasa de señalización de datos durante una transmisión.

i) Interfaz de conexión RS-232/V.24.

B.2 Características físicas:

Frecuencia de respuesta:

- Portadora de canal inferior: 1200 Hz ± 0,5 Hz.
- Portadora de canal superior: 2400 Hz ± 1,0 Hz.
- Tono de guarda: 1800 Hz ± 20 Hz.
- Tono de respuesta: 2100 Hz ± 15 Hz.

Niveles de transmisión:

- Señal de datos: 0 dBm a -21 dBm en pasos de 3 dBm.
- Tono de guarda: 6 dBm abajo del nivel de datos.

Niveles de detección de portadora:

- Sensibilidades: -43 dBm a -33 dBm.
- Histéresis: mayor que 2 dBm.
- Impedancia: 600 ohms balanceada, resistiva.

APÉNDICE C

LISTADO DEL PROGRAMA DE LA RED DE COMUNICACIONES

C.1 PROGRAMA PRINCIPAL DEL MONITOR CENTRAL (MONITOR.CPP)

```
#include "A0.cpp"
#include "A1.cpp"
#include "A2.cpp"
#include "A3.cpp"

void main(void)
        { unsigned char opcion, nc, salir=0;
          int n;
          PC=1;
          cseg=0;
          getdate(&f);
          cdia=f.da_day;
          clrscr();
          CargarLista();
          VerConecModem();
          do
          { PantallaMonitor();
            nc=0;
            opcion=2;
            do
            { if(kbhit())
                { n=getch();
                  if(n==0)
                    { n=getch();
                      if(n>=59 && n<=63) opcion=1;
                      if(n==64)
                        { puerto_com<<"ATS0=0";      // No contestar ninguna llamada
                          system("intersvr");        // Modo para transferir archivos a otra PC Local
                          puerto_com<<"ATS0=1";      // Auto contestar cualquier llamada
                          PantallaMonitor();
                        }
                      if(n==65) ModiTelTerm();
                      if(n==66) ModiCicloSin();
                    }
                  if(n==27) {  salir=1; goto FIN_SISTEMA; }
                }
              else
                {  gotoxy(75,14);
                   Fecha();
                   Reloj();
                   if(Timbre()) break;
                }
            } while(opcion!=1);
            switch(opcion)
                   { case 1: { nc=Enlace(n-59); break; }
                     case 2: { nc=DetCarrier();  break; }
                   }
            if(!nc) ComunicacionMonitor();
            Colgar();
            FIN_SISTEMA:
          } while(!salir);
        }
```

C.2 PROGRAMA PRINCIPAL DE CADA TERMINAL (TERMINAL.CPP)

```
#include "A0.cpp"
#include "A1.cpp"
#include "A2.cpp"
#include "A3.cpp"
#include "B0.cpp"

void main(void)
        { unsigned char i=0, conec_ok=0;
          char acceso=1;          // 0=Transferir archivo, 1=contestar llamada, 2=Comunicar con CENACE
          PC=0;
          clrscr();
          CargarTelCenace();
          CambiarTelCenace(); // Establecida en el programa de deteccion
          VerConecModem();
          PantallaPrincipal(PC);
          nfpen=0;
          switch(acceso)
                { case 0: { RevisarFallasPen(); break;}
                  case 1: { gotoxy(3,16); cprintf("  LLAMADA ENTRANTE: ESPERANDO PORTADORA");
                            gotoxy(3,17); cprintf("TIEMPO DE CONEXION:");
                            conec_ok=DetCarrier();
                            if(conec_ok==1)
                                  { gotoxy(23,17); cprintf("TEMPORIZADOR VENCIDO...!");
                                    delay(2000);
                                  }
                            break;
                          }
                }
          if(nfpen || (acceso==2))
                {  do { if((conec_ok=Enlace(1))==2) i=LineaOcupada(++i);
                      } while(conec_ok == 2 && i < V_NILL);
                }
          if(i < V_NILL && conec_ok==0) ComunicacionTerminal();
          Colgar();
        }
```

C.3 DECLARACIÓN DE LIBRERIAS, CONSTANTES, VARIABLES Y FUNCIONES, PARA LA COMUNICACIÓN SERIAL (A0.CPP).

C.3.1 Declaración de librerias

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

C.3.2 Declaración de constantes

```
#define TBUFFER     260     // Tamaño del arreglo de entrada
#define COM1        1       // No. puerto 1
#define COM2        2       // No. puerto 2
#define COM3        3       // No. puerto 3
#define COM4        4       // No. puerto 4
#define COM1BASE    0x3F8   // Direccion del puerto 1
#define COM2BASE    0x2F8   // Direccion del puerto 2
#define COM3BASE    0x3E8   // Direccion del puerto 1
#define COM4BASE    0x2E8   // Direccion del puerto 2
#define TXR         0       // Registro de trasmisión
#define RXR         0       // Registro de recepción
#define DLL         0       // Registro menos significativo de Divisor de la razón de baudios
#define DLH         1       // Registro mas significativo de Divisor de la razón de baudios
#define IER         1       // Registro de habilitación interrupción
#define IIR         2       // Registro Identificador de Interrupción
#define LCR         3       // Registro  del control de línea
#define MCR         4       // Registro de Control del Módem
#define LSR         5       // Registro de estatus de línea
#define MSR         6       // Registro estatus del Módem
#define NULA        0x00    // Paridad nula
#define IMPAR       0x08    // Paridad impar
#define PAR         0x18    // Paridad par
#define DTR         0x01    // Datos terminales listos
#define RTS         0x02    // Solicitud de emisión
#define MC_INT      0x08    // Salida UART
#define RX_INT      0x01    // Interrupción de datos recibidos
#define RX_ID       0x04    // Interrupcion identificador de datos recibidos
#define RX_MASK     0x07    // Interrupción de recepción de error
#define ICR         0x20    // Interrupción del control del puerto
#define EOI         0x20    // Fin de interrupción
#define IMR         0x21    // Interruptción parcial del puerto
#define IRQ4        0xEF    // COM1
#define IRQ3        0xF7    // COM2
```

C.3.3 Declaración de variables para los parámetros por omisión del puerto serie

```
int puerto   = 1;       // Número de puerto
int velocidad= 9600;    // Velocidad del puerto
int paridad  = NULA;    // Tipo de paridad
int bits     = 8;       // Bits de datos
int bitstop  = 1;       // Bits de stop
```

C.3.4 Declaración de funciones y de variables para utilizar el puerto serie

```
void Desabilitar(void);                 // Apaga los interruptores de comunicación
void interrupt(*desc_vectores[2])(...); // Manejo de vectores
```

```
class serial
        { int Conf_param(int paridad, int bits, int bitstop);// Configura por default: paridad, bits, bitstop
          int Conf_veloc(int velocidad);                     // Configura la velocidad
          int Conf_puerto(int puerto);                       // Configura el número de puerto
          void Ini_serial(void);                             // Habilita el puerto
          void Apagar(void);                                 // Fin de la comunicación
          public:
          serial(int paridad, int bits, int bitstop);        // Config. de los parámetros de comunicación
          void Limpiar(void);                                // Limpia el arreglo de entrada
          serial &operator<<( unsigned char ch );            // Salida de un caracter por el puerto
          serial &operator<<( char *str );                   // Salida de un flujo de caracteres al puerto
          serial &operator>>( unsigned char &ch );           // Entrada de un caracter por el puerto
          ~serial();                                         // Desabilitación del puerto
        };

unsigned int dpuerto, pto=1;                                 // Dirección del puerto
unsigned int ini_buf = 0;                                    // Inicio del arreglo de entrada
unsigned int fin_buf = 0;                                    // Fin del arreglo de entrada
unsigned char bufentrada[TBUFFER];                           // Arreglo de entrada para guardar los datos
                                                               recibidos en el puerto serie
void serial::Limpiar(void)                                   // Limpia el arreglo de entrada
        {ini_buf = fin_buf = 0;}

serial& serial::operator<<(unsigned char x )                 // Salida de un caracter
        { disable();
          while ((inportb(dpuerto + LSR) & 0x20) != 0x20);
          outportb(dpuerto + TXR, x);
          while ((inportb(dpuerto + LSR) & 0x20) != 0x20);
          enable();
          return *this;
        }

serial& serial::operator<<(char *string )                    // Salida de una array de caracteres al puerto
        { while (*string)
                { delay(10);
                  (*this) << *string;
                  delay(10);
                  string++;
                }
          (*this) << 13;
          return *this;
        }

serial &serial::operator>>(unsigned char &ch )               // Obtención del caracter presente en el
        { if (fin_buf == ini_buf)                              arreglo de entrada
                { ch = 255;
                  return *this;
                }
          ch = bufentrada[ini_buf++];
          if(ini_buf==TBUFFER) ini_buf=0;
          return *this;
        }
```

```
void Habilitar()                                    // Inicialización de las interrupciones
        { int c;
          disable();
          c = inportb(dpuerto + MCR) | MC_INT;
          outportb(dpuerto + MCR, c);
          outportb(dpuerto + IER, RX_INT);
          if(pto==COM1 || pto==COM3)  c = inportb(IMR) & IRQ4;
          else if(pto==COM2 || pto==COM4)  c = inportb(IMR) & IRQ3;
          outportb(IMR, c);
          enable();
        }

void Modem(void)                                    // Preparación del módem
        { int c;
          Habilitar();
          c = inportb(dpuerto + MCR) | DTR | RTS;
          outportb(dpuerto + MCR, c);
        }

void interrupt Interrupciones(...)                  // Manejo de interruptores para alojar datos
        { disable();                                  en bufentrada
          if ((inportb(dpuerto + IIR) & RX_MASK) == RX_ID)
                { if(fin_buf == TBUFFER) fin_buf=0;
                  bufentrada[fin_buf++] = inportb(dpuerto + RXR);
                }
          outportb(ICR, EOI);
          enable();
        }

void Conf_func(void)                                // Funciones para activar las interrupciones
        { if(COM1BASE == dpuerto || COM3BASE == dpuerto)
                {desc_vectores[0] = getvect(0x0c);
                 setvect(0x0c, Interrupciones);
                }
          if(COM2BASE == dpuerto || COM4BASE == dpuerto)
                {desc_vectores[1] = getvect(0x0b);
                 setvect(0x0b, Interrupciones);
                }
        }

int serial::Conf_param(int paridad,int bits,int bitstop)    // Configura párametros de comunicación
        { int parametros;
          if(dpuerto == 0) return (-1);
          if(bits < 5 || bits > 8) return (-1);
          if(bitstop != 1 && bitstop != 2) return (-1);
          if(paridad!=NULA && paridad!=IMPAR && paridad!=PAR) return (-1);
          parametros  = bits-5;
          parametros |= ((bitstop == 1) ? 0x00 : 0x04);
          parametros |= paridad;
          disable();
          outportb(dpuerto + LCR, parametros);
          enable();
          return (0);
        }
```

```
void serial::Ini_serial(void)                          // Habilitación del puerto
        { fin_buf = ini_buf = 0;
          Conf_func();
          Modem();
        }

int serial::Conf_veloc(int velocidad)                  // Configura la velocidad
        { char c;
          int divisor;
          if(!(velocidad==600||velocidad==1200||velocidad==2400||velocidad==4800||velocidad==9600))
                velocidad=9600;
          divisor = (int) (115200L/velocidad);         // Tipo de reloj:1.8432 Mhz/16=115200,3.072 Mhz/16=192000
          if (dpuerto == 0) return (-1);
          disable();
          c = inportb(dpuerto + LCR);
          outportb(dpuerto + LCR, (c | 0x80));
          outportb(dpuerto + DLL, (divisor & 0x00FF));
          outportb(dpuerto + DLH, ((divisor >> 8) & 0x00FF));
          outportb(dpuerto + LCR, c);
          enable();
          return (0);
        }

int serial::Conf_puerto(int puerto)                    // Establece el número de puerto a usar
        { switch(puerto)
                { case 1  : dpuerto = COM1BASE;break;
                  case 2  : dpuerto = COM2BASE;break;
                  case 3  : dpuerto = COM3BASE;break;
                  case 4  : dpuerto = COM4BASE;break;
                  default :  dpuerto = COM1BASE;
                }
          return (0);
        }

serial::serial(int paridad, int bits, int bitstop)     // Configura el puerto
        { FILE *Datos;
          char opcion=0;
          char temp=30;                                // Tiempo para modificar los parámetros de comunicación
                                                         antes de establecer los que están por omisión
          int vel[5]={600,1200,2400,4800,9600}; // Velocidades por omisión
          struct Confpuerto{ char puerto[10];      // Parámetros de configuración del puerto
                             char velocidad[10];
                           } config;
          gotoxy(2,2); cprintf(" CONFIGURAR PUERTO:   OPRIMA: ");
          gotoxy(2,3); cprintf("  A)  A CRITERIO      ENTER  ");
          gotoxy(2,4); cprintf("  B)  POR DEFAULT     ESC    ");
          gotoxy(2,5); cprintf(" OPCION:                     ");
          gotoxy(2,6); cprintf(" Tiempo de Espera:           ");
          while(temp)
                { if(kbhit()) opcion=getch();
                  if(opcion==13 || opcion==27) break;
                  gotoxy(22,6); cprintf("%d segs. ",--temp);
                  gotoxy(11,5); delay(1000);
                }
```

```
if(opcion==13)
        { clrscr();
          printf("\n# PUERTO  OPCION  >>\n");
          printf(" COM1     1\n COM2     2\n COM3     3\n COM4     4\n");
          while(!(puerto>='1' && puerto<='4')) { gotoxy(24,2); puerto=getche(); }
          printf("\n\n\n\n\n\n\nVELOCIDAD OPCION  >>\n");
          printf("  %d     1\n  %d     2\n",vel[0],vel[1]);
          printf("  %d     3\n  %d     4\n",vel[2],vel[3]);
          printf("  %d     5",vel[4]);
          while(!(velocidad>='1' && velocidad<='5')) { gotoxy(24,8);velocidad=getche();}
          puerto=puerto-48;
          velocidad= vel[(velocidad-49)];
          itoa(velocidad, config.velocidad, 10);
          itoa(puerto, config.puerto, 10);
          if((Datos=fopen("C:\\DETECTOR\\REGFALLA\\DATPUERT.DAT","w"))==NULL)
             {    clrscr();
                  cprintf("\r\n No se puede Encontrar: DATPUERT.DAT");
                  cprintf("\r\n Verifique que exista tal archivo...! \r\n Presione cualquier tecla  ");
                  getch();
                  system("EDIT /R C:\\DETECTOR\\REGFALLA\\AYUDACOM.dat");
                  exit(1);
             }
          fprintf(Datos,"%s %s",config.puerto,config.velocidad);
          fclose(Datos);
        }
 if(temp==0 || opcion==27)
        { if((Datos=fopen("C:\\DETECTOR\\REGFALLA\\DATPUERT.DAT","r"))==NULL)
             { clrscr();
               cprintf("\r\n No se puede leer  C:\\DETECTOR\\REGFALLA\\DATPUERT.DAT");
               cprintf("\r\n\n Verifique que exista tal archivo...! \r\n Presione cualquier tecla  ");
               getch();
               exit(1);
             }
          while(!feof(Datos))
             { fscanf(Datos,"%s %s",config.puerto,config.velocidad); break; }
          fclose(Datos);
          puerto = atoi(config.puerto);
          velocidad = atoi(config.velocidad);
        }
  if(temp) { printf("\n\n\n\n\n\n\n\n    PUERTO: %d",puerto);
             printf("\n VELOCIDAD: %d\n Presione cualquier tecla para continuar",velocidad);
             getch();
           }
  pto= puerto;
  int cond = 0;
  if (Conf_puerto(puerto)) cond = -1;
  if (Conf_veloc(velocidad)) cond = -1;
  if (Conf_param(paridad, bits, bitstop)) cond = -1;
  if (!cond) Ini_serial();
}

void Desconectar(void)                          // Desactiva los vectores antes de salir del programa
{ if(COM1BASE == dpuerto || COM3BASE == dpuerto)  setvect(0x0c, desc_vectores[0]);
  if(COM2BASE == dpuerto || COM4BASE == dpuerto)  setvect(0x0b, desc_vectores[1]);
}
```

```
void Desabilitar(void)                    // Apagar los interruptores de comunicación
        { int c;
          disable();
          c = inportb(IMR) | ~IRQ3 | ~IRQ4;
          outportb(IMR, c);
          outportb(dpuerto + IER, 0);
          c = inportb(dpuerto + MCR) & ~MC_INT;
          outportb(dpuerto + MCR, c);
          enable();
        }

void serial::Apagar(void)                 // Fin de la comunicación
        { Desabilitar();
          outportb(dpuerto + MCR, 0);
        }

serial::~serial()                         // Desabilitación del puerto
        { Apagar();
          Desconectar();
        }
```

C.4 DECLARACIÓN DE LIBRERIAS, CONSTANTES, VARIABLES Y FUNCIONES, EMPLEADAS EN LA RED (A1.CPP)

C.4.1 Declaración de librerias

```
#include <iostream.h>
#include <process.h>
#include <bios.h>
#include <time.h>
#include <math.h>
#include <errno.h>
#include <fcntl.H>
#include <io.h>
```

C.4.2 Declaración de constantes

```
#define TAM_BUF     260     // Tamaño de los arreglos: bufproces y bufsalida
#define LAP_RTX     4       // Lapso de retransmisiones (segs)
#define NUM_RTX     3       // Número de retransmisiones
#define CLAVE       "MAEV"  // Password para sincronizar terminal remota
#define SONIDO      1000    // Sonido del evento (1000 Hz)
#define LAPSON      100     // Lapso del Sonido del evento (milisegundos)
#define N_TERM      5       // Numero de terminales
```

```
#define M1          0x80    // Tipos de mensajes
#define M2          0x81
#define M3          0x82
#define M4          0x83
#define M5          0x84
#define M6          0x85
#define M7          0x86
#define M8          0x87
#define M9          0x88
#define M10         0x89
#define M11         0x90
#define M12         0x91
#define M13         0x92
#define M14         0x93
#define M15         0x94
#define M16         0x95
#define M17         0x96
```

C.4.3 Declaración variables globales

Para uso general:

```
unsigned char bufsalida[TAM_BUF];          // Arreglo para la salida de datos
unsigned char bufent[TAM_BUF];             // Arreglo para la entrada de datos
unsigned char sal_enlace;                  // Fin de enlace
char PC=1;                                 // Identificación de la PC: 1=Monitor, 0=Terminal
path_fte[40]="C:\\DETECTOR\\FALLAS\\";     // Directorio fuente de los archivos de disturbios
char path_dno[40]="C:\\FALLADES\\";        // Directorio destino de los archivos de disturbios
char Tel_cenace[20];                  // Teléfono del CENACE
char num_nodo[3];                     // Número de nodo remoto
char NdiaSinc[5];                     // Número de días para sincronizar los nodos remotos
char NILL[3];                         // Número de intentos de llamada cuando la línea está ocupada
char LTVM[5];                         // Tiempo (segs) para remarcar cuando está ocupada la línea
int  V_NILL=3;                        // Intentos de llamada por omisión
struct Lis_tel{ char terminal[20];    // Lista de las terminales y sus telefonos
                char telefono[20];
              } lista[N_TERM];
```

Para el manejo de los archivos:

```
char verif_lister[]="C:\\DETECTOR\\REGFALLA\\LISTERMI.DAT";   // Tiene las terminales y sus teléfonos
char verif_telcen[]="C:\\DETECTOR\\REGFALLA\\TELCENAC.DAT";  // Tiene el teléfono del CENACE
char verif_falla[]="C:\\DETECTOR\\REGFALLA\\FALLASPE.DAT";   // Tiene los registros de los disturbios
char arch_com[15]="COMPRESS.ZIP";                            // Usado alojar y descomprimir los datos

FILE *af;                             // Apuntador del archivo de revisión de registros pendientes
FILE *fp1;                            // Apuntador del archivo fuente
FILE *fp2;                            // Apuntador del archivo destino
char Array_temp[250];                 // Lista de los nombres de los archivos de disturbios pendientes
char g_archvs[4];                     // Nombre del archivo de disturbio presente
char path_vi[60];                     // Directorio con el grupo de archivos de disturbios (*.dat)
char path[60];                        // Directorio específico
```

```
int nfpen=0;                    // Cantidad de archivos de disturbios pendientes
int nda;                        // Cantidad de datos en la trama del archivo
float CAPTA=0;                  // Constante porcentual de la transferencia del archivo de disturbio
float pt=0;                     // Porcentaje de transferencia de archivo
char inc_barra=1;               // Incrementador de la barra de porcentaje
long longitud;                  // Longitud del archivo registrador
double NCNAPend;                // Número de caracteres del nombre archivo de disturbio presente
unsigned char EnvArch;          // Envio de archivos pendientes
```

Para la sincronización tomando como fuente de tiempo el sistema de la PC:

```
struct date f;                  // Estructura para obtener la fecha
struct time t;                  // Estructura para obtener el tiempo
time_t t0;                      // Estructura para obtener fecha y hora
union ts{ long sistema;         // Reloj del sistema
          unsigned char t[4];
        } reloj;
long Ndias=1;                   // Número de dias por default para sincronizar las terminales
int diasinc=0;                  // Contador de dias para sincronizar las terminales
char cdia, cseg;                // Cambio de dia y segundo respectivamente.
long t1,t2;                     // Lapsos de tiempos para prueba de sincronia (inicio y fin)
```

Para la configuración de la tarjeta:

```
union confT{ unsigned int T_PostFalla;  // Tiempo de postfalla
             unsigned char t[2];
           } tpf;

union confU{ float Limite_Umbral;       // Sensibilidad de umbral
             unsigned char u[4];
           } lu;

// Las 5 variables siguientes deben estar declaradas en el programa de deteccion !

unsigned char ConfTarjeta = 0;
float UMBRAL_DE_FALLA = 25.10;
unsigned int T_de_F = 10;
int mediciones[16]= {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
int Hora_Ver_Reg_Pend=12;
```

Para los encabezados:

```
char Texto[12][80] = {   "SINCRONIZACION",
                         "TRANSFERENCIA DE TEXTO",
                         "CONFIGURACION DE TARJETA",
                         "MEDICION ACTUAL",
                         "°°±±²²Û  PROCESO DE TRANSFERENCIA DE DATOS DE FALLA  Û²²±±°°",
                         " Aceptar [ENTER]  Cancelar [ESC] ",
                         "                                                    ",
                         "ZONA: NORESTE   CENTRO NACIONAL DE CONTROL DE ENERGIA  C.F.E.   ",
                         "\r\n Presione cualquier tecla para continuar...",
                         "                                     ",
                         "ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ",
                         " ²±±°SISTEMA DE MONITOREO DE FALLAS EN LINEAS DE TRANSMISION°±±² ",
                     };
```

```
char Nodo[13][30]= {    "############",          // Teléfono del nodo 1
                        "############",          // Teléfono del nodo 2
                        "############",          // Teléfono del nodo 3
                        "############",          // Teléfono del nodo 4
                        "############",          // Teléfono del nodo 5
                        "RIO_BRAVO",             // Nombre del nodo 1
                        "RIO_ESCONDIDO",         // Nombre del nodo 2
                        "ALTAMIRA",              // Nombre del nodo 3
                        "SAN_JERONIMO",          // Nombre del nodo 4
                        "MONTERREY",             // Nombre del nodo 5
                        "TRANSFERENCIA DE ARCHIVOS",   // Proceso local 1
                        "MODIFICAR TERMINALES",        // Proceso local 2
                        "MODIFICAR CICLO DE SINCRONIA",// Proceso local 3
                    };
```

C.4.4 Declaración de funciones

Pantallas y encabezados:

```
void Text(int i);                   // Muestra los encabezados
void Marco();                       // Muestra el marco de la pantalla
void PantallaTrasfArchivos();       // Muestra la pantalla para la transferencia de archivos
void PantallaMonitor();             // Muestra la pantalla para el monitor central
void PantallaPrincipal(char lugar); // Muestra la pantalla para accesar a los procesos
void PantallaTexto(char g);         // Muestra la pantalla para la transferencia de texto
void Fecha();                       // Obtiene la fecha del sistema de la PC
void Reloj();                       // Obtiene el tiempo del sistema de la PC
Timbre();                           // Señala la entrada de una llamada
void Sonido(char S,char L );        // Sonido de un evento
void TError(char te);               // Señala el error presente cuando se abre,cierra o lee un archivo
```

Configuración del puerto, conexión y establecimiento del enlace:

```
serial puerto_com(paridad,bits,bitstop); // Configura el puerto
void VerConecModem();                    // Verifica la conexión del módem
void Conf_Hayesmodem();                  // Configura los parámetros por omisión del módem
void CargarLista();                      // Carga el nombre de las terminales, sus teléfonos y los dias
                                            para sincronizarlas
void GuardarParametros();                // Guarda el nombre de las terminales, sus teléfonos y los
                                            dias para sincronizarlas
Enlace(char n);                          // Establece el enlace del monitor central
DetCarrier();                            // Detecta la señal portadora
void Colgar();                           // Cuelga el modém
void ComunicacionMonitor();              // Inicia la comunicación del monitor central
void ProbarLinea();                      // Prueba el estado de la línea
void T_mensajes();                       // Carga datos a los mensajes
void Teclas_F();                         // Teclas de acceso
void ModiTelTerm();                      // Modifica el nombre y teléfono de la terminal
void ModiCicloSin();                     // Modifica el periodo de dias para sincronizar las terminales
```

Convertidor y comparador de datos por entrada del teclado:

```
//      tipo     :Define el tipo dato de entrada (caracteres,números,o una convinación)
//      px,px    :Coordenadas de la posición del cursor
//      Ndatos   :Tamaño del flujo de datos
//      valor    :Valor específico a comparar
//      opcion   :Define la terminal y teléfono correspondiente
```

```
void ConvEntDatos(char tipo,char px,char py,char Ndatos,long valor,char opcion);
```

Acceso y procesamiento de los datos:

```
void mensaje(int m);                    // Carga los datos para el tipo de mensaje
void Transmision();                     // Transmite los mensajes
Recepcion(unsigned char ent);           // Recibe los mensajes
Rev_datos(int NTD);                     // Revisa los mensajes
void MensajeErroneo(char intenrtx);     // Transmite un ack por recibo de un mensaje incorrecto
```

Transferencia del archivo de disturbio:

```
void Abrir_fuente();        // Abre un archivo para guardar y enviar los datos comprimidos del disturbio
Leer_fuente();              // Lee y envía los datos del archivo comprimido
void Ver_fin_arch();        // Lee y envía el fin del archivo comprimido
void Abrir_destino();       // Abre un archivo para recopilar los datos comprimidos del disturbio
void Guardar_destino();     // Recibe y guarda los datos del archivo comprimido
void Cerrar_destino();      // Recibe y guarda el fin del archivo comprimido
void AbortTransfArch();     // Aborta por algún problema en la transferencia del archivo
```

Proceso para efectuar la sincronización:

```
void Sincronizacion();      // Efectúa la sincronización
void Hora_local();          // Obtiene la hora local
void R_hora_local();        // Reconfigura la hora y fecha local
void Hora_remota();         // Obtiene la hora remota
void Sinc_remota();         // Da la opción para sincronizar el nodo remoto
void Resincronizar();       // Sincroniza el nodo remoto
```

Proceso para configurar la tarjeta de adquisión de datos:

```
void ParamActualTarjeta();  // Configura la tarjeta
void ParamModTarjeta();     // Da la opción para configurar los parámetros de la tarjeta
void ReConfTarjeta();       // Reconfigura los parámetros de la tarjeta
```

Procesos para transferir texto y para medir las señales en la línea de transmisión:

```
void Transf_texto();        // Transferiere el texto del usuario
void Medicion_actual();     // Mide las señales de voltaje y de corriente en la línea de transmisión
```

C.5 FUNCIÓN PARA CARGAR Y TRANSMITIR LOS MENSAJES (A2.CPP)

```
void mensaje(int m)
        { int i, check2=0;
          switch(m)
            {    case 1 : { bufsalida[0] = M1 ;              // Envío de prueba de sincronia del monitor
                            bufsalida[1] = 0;
                            t1 = biostime(0,0L);
                            break; }
                 case 2 : { bufsalida[0] = M2;               // Petición de sincronía de la terminal
                            bufsalida[1] = 0;
                            break; }
                 case 3 : { bufsalida[0] = M3;               // Envío de sincronia
                            bufsalida[1] = 4;
                            int lapso;
                            t2 = biostime(0,0L);
                            lapso = floor((t2-t1)/2);
                            reloj.sistema = t2 + lapso;
                            bufsalida[3] = reloj.t[0];
                            bufsalida[4] = reloj.t[1];
                            bufsalida[5] = reloj.t[2];
                            bufsalida[6] = reloj.t[3];
                            break;}
                 case 4 : { bufsalida[0] = M4;               // Envío de la hora
                            bufsalida[1] = 4;
                            reloj.sistema = biostime(0,0L);
                            bufsalida[3] = reloj.t[0];
                            bufsalida[4] = reloj.t[1];
                            bufsalida[5] = reloj.t[2];
                            bufsalida[6] = reloj.t[3];
                            break;}
                 case 5 : { bufsalida[0] = M5;               // Petición de la hora
                            bufsalida[1] = 0;
                            break;}
                 case 6 : { bufsalida[0] = M6;               // Petición de config. de la tarjeta
                            bufsalida[1] = 0;
                            break;}
                 case 7 : { bufsalida[0] = M7;               // Envío de configuración de la tarjeta
                            bufsalida[1] = 8;
                            tpf.T_PostFalla = T_de_F;
                            lu.Limite_Umbral = UMBRAL_DE_FALLA;
                            bufsalida[3] = ConfTarjeta;
                            bufsalida[4] = lu.u[0];
                            bufsalida[5] = lu.u[1];
                            bufsalida[6] = lu.u[2];
                            bufsalida[7] = lu.u[3];
                            bufsalida[8] = tpf.t[0];
                            bufsalida[9] = tpf.t[1];
                            bufsalida[10]= (unsigned char)Hora_Ver_Reg_Pend;
                            break;}
                 case 8 : { bufsalida[0] = M8;               // Envío de reconfiguración de tarjeta
                            bufsalida[1] = 8;
                            bufsalida[3] = ConfTarjeta;
                            bufsalida[4] = lu.u[0];
                            bufsalida[5] = lu.u[1];
                            bufsalida[6] = lu.u[2];
                            bufsalida[7] = lu.u[3];
                            bufsalida[8] = tpf.t[0];
                            bufsalida[9] = tpf.t[1];   break;}
```

```
        case 9 : { bufsalida[0] = M9;              // Petición de medición actual en la línea
                   bufsalida[1] = 0;
                   break;}
        case 10:{ bufsalida[0] = M10;              // Envío de medición actual de fallas
                   bufsalida[1] = 32;
                   int c, n=2;
                   for(c=0;c<16;c++)
                          { bufsalida[++n] = (mediciones[c]>>8)&0xFF;
                            bufsalida[++n] = mediciones[c]&0xFF;
                          }
                   break;}
        case 11:{ bufsalida[0] = M11;              // Petición de transferencia de texto
                   bufsalida[1] = 0;
                   break;}
        case 12:{ bufsalida[0] = M12;              // Envío de aviso del disturbio presente
                   bufsalida[1] = 9;
                   union cte{ float porcen;
                              unsigned char pta[4];    }capta;
                   capta.porcen = CAPTA;
                   bufsalida[3] = capta.pta[0];
                   bufsalida[4] = capta.pta[1];
                   bufsalida[5] = capta.pta[2];
                   bufsalida[6] = capta.pta[3];
                   bufsalida[7] = g_archvs[0];
                   bufsalida[8] = g_archvs[1];
                   bufsalida[9] = g_archvs[2];
                   bufsalida[10] = nfpen;
                   bufsalida[11] = (num_nodo[0]-44)&0xFF;
                   break;}
        case 13:{ bufsalida[0] = M13;              // Envío de datos del archivo de disturbio
                   bufsalida[1] = nda;
                   break;}
        case 14:{ bufsalida[0] = M14;              // Petición de la próxima trama del archivo
                   bufsalida[1] = 0;
                   break;}
        case 15:{ bufsalida[0] = M15;              // Envío de fin del archivo de falla 'N'
                   bufsalida[1] = nda;
                   break;}
        case 16:{ bufsalida[0] = M16;              // Envío de fin de los archivos pendientes
                   bufsalida[1] = nda;
                   break;}
        case 17:{ bufsalida[0] = M17;              // Envío de fin del enlace
                   bufsalida[1] = 0;
                   sal_enlace = 0;
                   break;}
    }
bufsalida[2]=(bufsalida[0]+bufsalida[1])&0xFF;;// Detector de errores de la cabecera:Check 1
check2 = 0;
if(bufsalida[1])                               // Detector de errores del los datos:Check 2
       { for(i=3;i<(bufsalida[1]+3);i++)  check2 = bufsalida[i] + check2;
         bufsalida[bufsalida[1]+3] = check2 & 0xFF;
       }
 Transmision();
}
```

C.6 CONTENIDO DE LAS FUNCIONES DECLARADAS EN C.4.4 (A3.CPP)

La descripción de cada una de las siguientes funciones se encuentra en C.4.4. El contenido de las funciones señaladas con '•' carecen de análisis. Estas son únicamente presentación de pantallas y encabezados.

- Text(int i),
- Marco(),
- PantallaMonitor(),
- PantallaTrasfArchivos(),
- PantallaPrincipal(char lugar),
- PantallaTexto(char g)

```
void Fecha()
        { char espera,ndo,n_int,sinc=1;
          getdate(&f);
          if(f.da_day != cdia)
                { diasinc++;
                  if(diasinc==3000) diasinc=0;
                  if(diasinc==Ndias)                    // Inicio del proceso de sincronizacion
                        { if(PC)
                          for(ndo=0;ndo<N_TERM;ndo++)
                                { n_int=1;
                                  gotoxy(44,16); cprintf("SINCRONIZANDO %s ",Nodo[ndo+5]);
                                  while(sinc)
                                        { if(!(sinc=Enlace(ndo)))
                                                { ComunicacionMonitor();  Colgar(); }
                                          else  { Colgar();
                                                  if(sinc==1)  break;
                                                  if(n_int==V_NILL) break;
                                                  gotoxy(44,18); cprintf("TIEMPO DE ESPERA:");
                                                  espera=60;
                                                  while(espera)
                                                        { gotoxy(62,18);
                                                          cprintf("%d segs ",espera--);
                                                          delay(1000);
                                                          if(kbhit()) break;
                                                        }
                                                  gotoxy(44,17); cprintf("Intento: %d",++n_int);
                                                }
                                        }
                                  delay(2000);
                                  sinc=1;
                                }                                   // Fin del proceso de sincronizacion
                          diasinc=0;
                          if(PC) PantallaMonitor();
                          gotoxy(5,24); cprintf("FECHA: %d/0%d/%d",cdia=f.da_day,f.da_mon,f.da_year);
                        }
                }
        }
```

```
void ConvEntDatos(char tipo,char px,char py,char Ndatos,long valor,char opcion)
        {  int i;
           long convalor=0;
           float valorflot=0;
           char tpch,raninf,ransup,ch,limiteinf=1;
           char numero[15];
           tpch=13;
           raninf='0';
           ransup='9';
           if(tipo==1)  ransup='z';
           if(tipo==2 || tipo==3)  tpch=',';
           if(tipo==4)  tpch='.';
           if(tipo==8) { raninf='1'; ransup='5';}
           ENTDATO:
           gotoxy(px,py);
           for(i=0;i<Ndatos;i++) cprintf(" ");
           gotoxy(px,py);
           i=0;
           for(;;) { ch=getch();
                   if(ch==27)
                          { gotoxy(px,py);
                            switch(tipo)
                                   { case  1: cprintf("%s",Nodo[opcion-44]);   break;
                                     case  2: cprintf("%s",Nodo[opcion-49]);   break;
                                     case  3: cprintf("%s",Tel_cenace);        break;
                                     case  4: cprintf("%.3f",lu.Limite_Umbral); break;
                                     case  5: cprintf("%s dias",NdiaSinc);     break;
                                     case  6: cprintf("%u",tpf.T_PostFalla);   break;
                                     case  7: cprintf("%d",bufent[10]);        break;
                                     case  8: cprintf("%s",num_nodo);          break;
                                     case  9: cprintf("%s",NILL);              break;
                                     case 10: cprintf("%s",LTVM);              break;
                                    }
                             break;
                           }
                   if(ch==13 || ch==tpch || (ch>=raninf && ch<=ransup))
                          { numero[i++]=ch;
                            if(i>=1 && i<=(Ndatos+1))
                              { if((i == (Ndatos+1)) && (ch!=13)) { Sonido(3,2);  goto ENTDATO;}
                                printf("%c",ch);
                                if(ch==13)
                                 { if(i==1) { Sonido(2,2); goto ENTDATO; }
                                   numero[i-1]=0;
                                   if(tipo>=4 && tipo<=10)
                                    { if(tipo==4) convalor = valorflot = atof(numero);
                                      if(tipo>=5 && tipo<=10) convalor=atol(numero);
                                      if((tipo==5||tipo==7||tipo==9||tipo==10)&&convalor==0) limiteinf=0;
                                      if(i>=1 &&  (convalor>=limiteinf) && (convalor<=valor) )
                                        {  switch(tipo)
                                             { case 4: lu.Limite_Umbral = valorflot;          break;
                                               case 5: diasinc=0; Ndias=convalor;
                                                       if(Ndias==0) Ndias = -1;
                                                       strcpy(NdiaSinc,numero);             break;
                                               case 6: tpf.T_PostFalla = convalor;          break;
                                               case 7: bufsal[10]=(unsigned char)convalor;  break;
```

```
                                        case 8: strcpy(num_nodo,numero);        break;
                                        case 9: strcpy(NILL,numero);            break;
                                        case 10:strcpy(LTVM,numero);            break;
                                       }
                                    break;
                                   }
                                  else
                                   { Sonido(1,3);
                                     gotoxy(px,py); printf("VALOR INVALIDO");
                                     delay(3000);
                                     gotoxy(px,py); printf("          ");
                                     goto ENTDATO;
                                   }
                                 }
                                else
                                 { if(tipo==1) strcpy(Nodo[opcion-44],numero);
                                   if(tipo==2) strcpy(Nodo[opcion-49],numero);
                                   if(tipo==3) strcpy(Tel_cenace,numero);
                                   break;
                                 }
                              }
                             else { Sonido(2,2); goto ENTDATO; }
                            }
                    else   { Sonido(3,2);  goto ENTDATO; }
          }
}

void Sonido(char S,char L )
        { sound(SONIDO*S);
          delay(LAPSON*L);
          nosound();
        }

void CargarLista()
        { int loc=0;
          if((fp1=fopen("C:\\DETECTOR\\REGFALLA\\LISTERMI.DAT","r"))==NULL) TError(0);
          while(!feof(fp1))
                { fscanf(fp1,"%s %s",lista[loc].terminal,lista[loc].telefono);
                  strcpy(Nodo[loc],lista[loc].telefono);
                  strcpy(Nodo[loc+5],lista[loc].terminal);
                  if((++loc)==N_TERM) { fscanf(fp1,"%s",NdiaSinc); break;}
                }
          fclose(fp1);
          Ndias=atol(NdiaSinc);
        }

 Timbre()
        {if((inportb(dpuerto+MSR) & 0x40) == 0x40)
                { gotoxy(2,16); cprintf("  LLAMADA ENTRANTE: ESPERANDO PORTADORA");
                  gotoxy(2,17); cprintf("TIEMPO DE CONEXION:");
                  return 1;
                }
          return 0;
        }
```

```
void TError(char te)
        { int min=0;
          cprintf("\n\n No se puede ");
          switch(te)
                { case 0: cprintf("Encontrar... %s ",verif_lister);                    break;
                  case 1: cprintf("Abrir... %s ",verif_falla);                         break;
                  case 2: cprintf("Abrir... %s%s ",path_fte,arch_com);                 break;
                  case 3: cprintf("Abrir... %s%s ",path_dno,arch_com);                 break;
                  case 4: cprintf("Leer... %s%s ",path_fte,arch_com);                  break;
                  case 5: cprintf("Escribir... %s%s ",path_dno,arch_com);              break;
                  case 6: cprintf("Encontrar... %s ",verif_telcen);                    break;
                  case 7: gotoxy(1,2); cprintf(" Error por falta de memoria...!");     break;
                }
          mensaje(17);
          if(te==7) cprintf("\r\n\n SOLUCION: Reinicializar y estar bajo DOS ");
          else      cprintf("\r\n\n SOLUCION: Verifique que el archivo exista...! ");
          cprintf("\r\n\n Oprima cualquier tecla para obtener ayuda     ");
          gettime(&t);
          te=t.ti_min+1;
          for(;;)  { gettime(&t);
                     if(t.ti_min == te)
                       { te=te+1;
                         delay(200);  Sonido(2,1);
                         gotoxy(15,12); cprintf("*** TIEMPO INACTIVO DEL SISTEMA:%d minutos*** ",++min);
                         if(te==60) te=0;
                       }
                     if(kbhit()) { system("edit /r c:\\detector\\regfalla\\ayudacom.dat"); break;}
                   }
          exit(1);
        }

 void VerConecModem()
        { char te, esc=1;
          int min=0;
          if(((inportb(dpuerto+MSR) & 255) != 48))
                { gotoxy(2,16); cprintf(" EL MODEM NO ESTA CONECTADO...!        ");
                  gotoxy(2,17); cprintf(" Si desea continuar as¡, presione ENTER ");
                  gotoxy(2,18); cprintf(" Si necesita AYUDA presione ESC         ");
                  gettime(&t);
                  te=t.ti_min+1;
                  while(((inportb(dpuerto+MSR) & 255) != 48))
                        { if(kbhit()) esc=getch();
                          if(esc==13) break;
                          if(esc==27) { system("EDIT /R C:\\detector\\regfalla\\ayudacom.dat"); exit(1);}
                          gettime(&t);
                          if(t.ti_min == te)
                            { te=te+1;
                              Sonido(2,7);
                              gotoxy(15,20);cprintf("TIEMPO INACTIVO DEL SISTEMA: %d minutos",++min);
                              if(te==60) te=0;
                            }
                        }
                }
           Conf_Hayesmodem();
          }
```

```
void Conf_Hayesmodem()
        { int i;
          unsigned char com_modem;
          char comando[4][10] = { "ATZ0",          // Parámetros por default del modem (profile 0)
                                  "ATE0",          // Ver el eco del comando puesto al modem: No(0) Si(1)
                                  "ATV0",          // Respuestas númericas
                                  "ATS0=1"         // Contestación al primer timbre
                                };
          gotoxy(2,18); printf(" CONFIGURANDO MODEM: ");
          for(i=0;i<4;i++)
                { puerto_com<<comando[i];
                  printf("%s ",comando[i]);
                  delay(1000);
                }
          puerto_com.Limpiar();
        }

Enlace(char n)
        { char telefono[40];
          if(((inportb(dpuerto+MSR) & 255) != 48)) VerConecModem();
          strcpy(telefono,"ATDT");
          if(PC) { puerto_com<<strcat(telefono,Nodo[n]);
                   gotoxy(3,16); cprintf("       TELEFONO: %s",Nodo[n]);
                 }
          else   { puerto_com<<strcat(telefono,Tel_cenace);
                   gotoxy(3,16); cprintf("       TELEFONO: %s",Tel_cenace);
                 }
          puerto_com.Limpiar();
          gotoxy(3,17); cprintf("TIEMPO DE CONEXION: ");
          n=DetCarrier();
          gotoxy(3,18); cprintf("ESTADO DE LA LINEA: ");
          if(n==0) return 0;
          switch(n)  { case '6': cprintf("NO HAY TONO");                break;
                       case '7': cprintf("OCUPADA");                    break;
                       case  1 : cprintf("NO HAY PORTADORA");           break;
                       case  2 : cprintf("MARCACION CANCELADA");        break;
                     }
          delay(2000);
          if(n=='7') return 2;
          return 1;
        }

 DetCarrier()
        { unsigned char edo_linea,canc=0,timer=40;
          while(timer)
            { if((inportb(dpuerto+MSR) & 0x80) == 0x80) return 0;      // Deteccion de portadora
              puerto_com>>edo_linea;
              if(edo_linea == '6' || edo_linea == '7') return edo_linea;  // No hay Tono o linea ocupada
              if(kbhit())  { canc=getch();
                             if(canc==27) return 2;
                           }
              gotoxy(23,17); cprintf("%d segs. ", timer--);
              delay(1000);
            }
          return 1;
        }
```

```
void Colgar()
        { char c;
          disable();
          c = inportb(dpuerto + MCR);
          outportb(dpuerto + MCR, (c ^ 0x1));
          enable();
          gotoxy(5,24); cprintf(" FIN DEL ENLACE...");
          puerto_com<<"ATH0";
          delay(1000);
          disable();
          c = inportb(dpuerto + MCR);
          outportb(dpuerto + MCR, (c ^ 0x1));
          enable();
          delay(1000);
          puerto_com.Limpiar();
        }

void ComunicacionMonitor()
        { unsigned char ent,sal,d,intenrtx=0;
          sal_enlace=1;
          ProbarLinea();
          if(!(diasinc==Ndias)) PantallaPrincipal(PC);
          puerto_com.Limpiar();
          while(sal_enlace)
                { if(kbhit()) { sal = getch();
                                if(sal==27) { AbortTransfArch(); mensaje(17); }      // Salir del enlace
                                if(sal==NULL) Teclas_F();
                              }
                  else        { if(fin_buf!=ini_buf)
                                  { puerto_com >> ent;
                                    if(((inportb(dpuerto+MSR) & 0x80) != 0x80)) AbortTransfArch();
                                    if(ent==21) MensajeErroneo(intenrtx++);
                                    if(ent>=M1 && ent<=M17)
                                        { d=Recepcion(ent);
                                          if(d==6) { intenrtx=0;T_mensajes();}  // Buen mensaje recibido
                                          else  puerto_com<<d;                  // Mal mensaje recibido
                                        }
                                  }
                                gotoxy(61,15); Reloj();
                              }
                  if(diasinc==Ndias) { mensaje(3); delay(2000); mensaje(17); delay(2000); }
                }
        }

void Teclas_F()
        { unsigned char tecla;
          tecla=getch();
          switch(tecla)
                { case 59:  Sincronizacion();    break;     // F1 Sincronización
                  case 60:  PantallaTexto(1);    break;     // F2 Transferencia de texto
                  case 61:  if(PC) mensaje(6);   break;     // F3 Configuración de tarjeta
                  case 62:  if(PC) mensaje(9);   break;     // F4 Medición Actual
                }
        }
```

```
void Reloj()
        { gettime(&t);
          if(t.ti_sec != cseg) {gotoxy(68,24); printf("%2d:%02d:%02d ",t.ti_hour,t.ti_min,cseg=t.ti_sec);}
        }

void ProbarLinea()
        { unsigned char linea,ACK,TST,espera=0;
          ACK=TST=6;
          puerto_com<<TST;                                // Envio de TST para probar la linea
          while(TST)
                { delay(100);
                  espera++;
                  puerto_com>>linea;
                  if(linea==TST) TST=0;                   // Recibo de ack
                  if(espera==40) { sal_enlace=0; break;}  // Limite de espera: 4 segundos
                }
          puerto_com<<ACK;
        }

void T_mensajes()
        { switch((int)bufent[0])
                { case M1 : if(PC) mensaje(3);            // Prueba de sincronia del monitor
                            else  mensaje(1);   break;
                  case M2 : mensaje(1);          break;   // Peticion de sincronia de la terminal
                  case M3 : Resincronizar();     break;   // Envio de sincronia
                  case M4 : Hora_remota();       break;   // Envío de la hora
                  case M5 : mensaje(4);          break;   // Petición de la hora remota
                  case M6 : mensaje(7);          break;   // Petición de config. de la tarjeta
                  case M7 : ParamModTarjeta();   break;   // Envío de configuración de la tarjeta
                  case M8 : ReConfTarjeta();     break;   // Envío de reconfiguración de tarjeta
                  case M9 : mensaje(10);         break;   // Petición de medición actual en la línea
                  case M10: Medicion_actual();   break;   // Envío de medición de las líneas de Tx.
                  case M11: PantallaTexto(0);    break;   // Petición de transferencia de texto
                  case M12: Abrir_destino();     break;   // Envío de aviso de disturbio presente
                  case M13: Guardar_destino();   break;   // Envío de datos del disturbio presente
                  case M14: Ver_fin_arch();      break;   // Petición de la próxima trama
                  case M15: Guardar_destino();   break;   // Fin del archivo de disturbio "N"
                  case M16: Guardar_destino();   break;   // Fin de los archivos de disturbios pend.
                  case M17: sal_enlace=0;        break;   // Fin del enlace
                }
        }

void ModiCicloSin()
        { gotoxy(6,16); cprintf(" ÉÍµ SINCRONIA CON LAS TERMINALES REMOTAS ÆÍ» ");
          gotoxy(6,17); cprintf(" º                                            º ");
          gotoxy(6,18); cprintf(" º           CICLO ACTUAL:                    º ");
          gotoxy(6,19); cprintf(" º CAMBIO DE CICLO (0-3000):                  º ");
          gotoxy(6,20); cprintf(" ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼ ");
          gotoxy(8,21);  cprintf("ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß");
          gotoxy(35,18); cprintf("%s dias",NdiaSinc);
          gotoxy(14,20); cprintf(" Para Cancelar presione ESC ");
          ConvEntDatos(5,35,19,4,3000,0);
          GuardarParametros();
          PantallaMonitor();
        }
```

```
void ModiTelTerm()
        { char telefono[20];
          char terminal[20];
          char opcion=0;
          gotoxy(15,3);  cprintf(" É͸»  ");
          gotoxy(15,4);  cprintf(" º OPCION TERMINALES:      TELEFONOS:          º ");
          gotoxy(15,5);  cprintf(" º                                             º ");
          gotoxy(15,6);  cprintf(" º   1 :                                       º ");
          gotoxy(15,7);  cprintf(" º   2 :                                       º ");
          gotoxy(15,8);  cprintf(" º   3 :                                       º ");
          gotoxy(15,9);  cprintf(" º   4 :                                       º ");
          gotoxy(15,10); cprintf(" º   5 :                                       º ");
          gotoxy(15,11); cprintf(" º                                             º ");
          gotoxy(15,12); cprintf(" º  ESC:  CANCELAR                             º ");
          gotoxy(15,13); cprintf(" º                                             º ");
          gotoxy(15,14); cprintf(" È͸¼  ");
          gotoxy(17,15); cprintf("ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß");
          gotoxy(25,6);  cprintf("%s",Nodo[5]);
          gotoxy(44,6);  cprintf("%s",Nodo[0]);
          gotoxy(25,7);  cprintf("%s",Nodo[6]);
          gotoxy(44,7);  cprintf("%s",Nodo[1]);
          gotoxy(25,8);  cprintf("%s",Nodo[7]);
          gotoxy(44,8);  cprintf("%s",Nodo[2]);
          gotoxy(25,9);  cprintf("%s",Nodo[8]);
          gotoxy(44,9);  cprintf("%s",Nodo[3]);
          gotoxy(25,10); cprintf("%s",Nodo[9]);
          gotoxy(44,10); cprintf("%s",Nodo[4]);
          gotoxy(30,14); cprintf("TERMINAL A MODIFICAR:");
          while(!((opcion==27)||(opcion>='1' && opcion<='5')))    { gotoxy(52,14); opcion=getch(); }
          if(opcion>='1' && opcion<='5')
                { gotoxy(20,16); cprintf(" É͸»  ");
                  gotoxy(20,17); cprintf(" º TERMINAL ACTUAL:                  º ");
                  gotoxy(20,18); cprintf(" º TELEFONO ACTUAL:                  º ");
                  gotoxy(20,19); cprintf(" º                                   º ");
                  gotoxy(20,20); cprintf("  ͸  CAMBIAR POR:  ͸  ");
                  gotoxy(20,21); cprintf(" º                                   º ");
                  gotoxy(20,22); cprintf(" º NOMBRE DE TERMINAL:               º ");
                  gotoxy(20,23); cprintf(" º NUMERO DE TELEFONO:               º ");
                  gotoxy(20,24); cprintf(" È͸¼  ");
                  gotoxy(22,25); cprintf("ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß");
                  gotoxy(43,17); cprintf("%s",Nodo[opcion-44]);
                  gotoxy(43,18); cprintf("%s",Nodo[opcion-49]);
                  ConvEntDatos(1,43,22,13,500,opcion);
                  ConvEntDatos(2,43,23,13,500,opcion);
                  GuardarParametros();
                }
           PantallaMonitor();
        }

 void MensajeErroneo(char intenrtx)
        { if(intenrtx < NUM_RTX)                    // Limite de retransmisiones por recibo de Nacks
                { printf("\r\n Recibo de Nack: Retransmision %d",intenrtx+1);
                  delay(1000); Transmision();
                }
          else  mensaje(17);
        }
```

```
void GuardarParametros()
        { int i;
          if((fp1=fopen("C:\\DETECTOR\\REGFALLA\\LISTERMI.DAT","w"))==NULL) TError(0);
          else  { for(i=0;i<N_TERM;i++)
                        { strcpy(lista[i].telefono,Nodo[i]);
                          strcpy(lista[i].terminal,Nodo[i+5]);
                          fprintf(fp1,"%s %s %s",lista[i].terminal,lista[i].telefono,"\n");
                        }
                  fprintf(fp1,"%s",NdiaSinc);
                  fclose(fp1);
                }
          Ndias=atol(NdiaSinc);
        }


void Transmision()
        { int i,rtx=0;
          RTX:
          for(i=0;i<bufsal[1]+3;i++) puerto_com<<(bufsal[i] & 0XFF);
          if(bufsal[1]) puerto_com<<(bufsal[i] & 0XFF);
          i=0;
          while( fin_buf==ini_buf && (bufent[0]>=M12 && bufent[0]<=M15) )        // Rutina de espera
                { delay(10);
                  i++;
                  if(i == LAP_RTX*100)                                  // Temporizador de entrada: 4 segs
                        { if(++rtx <= NUM_RTX)                          // Contador por agotamiento de espera:3
                              { cprintf("\r\n Temporizador de entrada vencido: Retransmision No %d",rtx);
                                goto RTX;
                              }
                          else { mensaje(17); break;}
                        }
                }
        }


Recepcion(unsigned char ent)
        { int r,rx_datos,NDT=2;
          bufent[0]=ent;
          r=rx_datos=0;
          while(!r)
          if(fin_buf != ini_buf)
                { puerto_com>>ent;
                  bufent[++r]=ent;
                  if(ent) NDT=ent+3;
                }
          while(!rx_datos)
          if(fin_buf != ini_buf)
                { puerto_com>>ent;
                  bufent[++r]=ent;
                  if(r==NDT) rx_datos=Rev_datos(NDT);
                }
          return rx_datos;
        }
```

```
Rev_datos(int NTD)
        { int Rck1,Rck2=0,a=21,i;
          Rck1=(int)(bufent[0]+bufent[1]) & 0xFF;
          if(Rck1 ==(int)bufent[2])
                  { a=6;
                    if((int)bufent[1])
                            { for(i=3;i<NTD;i++) Rck2=(int)bufent[i]+Rck2;
                              Rck2=Rck2 & 0xFF;
                              if(Rck2 != (int)bufent[NTD]) a=21;
                            }
                  }
          return a;
        }

void Ver_fallas_pen()
        { int falla;
          strcpy(path,path_dno);
          strcat(path,arch_com);
          if ((falla = open(path, O_RDONLY))!= -1)
                  { close(falla);
                    strcpy(path,"DEL ");
                    strcat(path,path_dno);
                    strcat(path,arch_com);
                    system(path);
                  }
        }

void Abrir_fuente()
        { char GrupArchFallas[15];
          clrscr();
          strcpy(GrupArchFallas,g_archvs);
          strcat(GrupArchFallas,"*.*");
          strcpy(path_vi,path_fte);
          strcat(path_vi,GrupArchFallas);
          gotoxy(2,4);cprintf("°±² PROCESO DE COMPRESION: %s: ²±°\r\n\n",g_archvs);
          strcpy(path,path_fte);
          strcat(path,arch_com);
          if(spawnlpe(P_WAIT,"pkzip.exe","pkzip.exe",path,path_vi,NULL) == -1) TError(7);
          if ((longitud = open(path, O_RDONLY)) == -1) TError(2);
          CAPTA=(float)((TAM_BUF-5)*100)/filelength(longitud);
          close(longitud);
          if((fp1=fopen(path,"rb"))==NULL) TError(2);
          PantallaTrasfArchivos();
          if(sal_enlace) { EnvArch=0; mensaje(12); }
        }

void Ver_fin_arch()
        { switch(Leer_fuente())
                  { case 0 : mensaje(13);     break;
                    case 1 : mensaje(15);     break;
                    case 2 : mensaje(16);     break;
                    case 3 : Terror(2);       break;
                    case 4 : Terror(1);       break;
                  }
        }
```

```
Leer_fuente()
        { unsigned char caracter;
          int i=0;
          gotoxy(42,12);cprintf("%.1f %",pt=pt+CAPTA);
          if(pt>=(inc_barra*10)) { gotoxy((49+inc_barra++),12);cprintf("²");}
          while(!(feof(fp1)))
                  { caracter=fgetc(fp1);
                    if(ferror(fp1)) return 3;
                    ++i;
                    if(i<TAM_BUF-5) bufsal[i+2]=caracter;
                    else { bufsal[i+2]=caracter; nda=i; return 0; }
                  }
          nda=i;
          fclose(fp1);
          strcpy(path,"DEL ");
          strcat(path,path_fte);
          strcat(path,arch_com);
          system(path);
          if((af=fopen(verif_falla,"w"))==NULL) return 4;
          NCNAPend=NCNAPend- TamNomArch;
          fwrite(&Array_temp[TamNomArch], NCNAPend, 1, af);
          fclose(af);
          –nfpen;
          inc_barra=1;
          Sonido(3,3);
          if(nfpen) { EnvArch=1; return 1; }
          EnvArch=0;
          return 2;
        }

 void AbortTransfArch()
        { if(bufent[0] >=M12 && bufent[0] <= M17)
                  { strcpy(path,"DEL ");
                    if(bufent[0] == M14)  { fclose(fp1); strcat(path,path_fte); }
                    else   { fclose(fp2); strcat(path,path_dno); }
                    strcat(path,arch_com);
                    system(path);
                  }
          mensaje(17);
        }

 void Cerrar_destino()
        { fclose(fp2);
          strcpy(path,path_dno);
          strcat(path,arch_com);
          gotoxy(2,15);printf("°±² PROCESO DE DESCOMPRESION: %s ²±°\n",arch_com);
          spawnlpe(P_WAIT,"pkunzip.exe","pkunzip.exe","-O",path,path_dno,NULL);
          perror("exec error");
          strcpy(path,"DEL ");
          strcat(path,path_dno);
          strcat(path,arch_com);
          system(path);
          if(bufent[0]==M15) puerto_com<<6;
          if(bufent[0]==M16) mensaje(17);
          inc_barra=1;
        }
```

```
void Abrir_destino()
        { if(bufent[0]==M12)
                { union cteent { float porcenent;
                                 unsigned char ptaent[4];  } captaent;
                  captaent.ptaent[0]=bufent[3];
                  captaent.ptaent[1]=bufent[4];
                  captaent.ptaent[2]=bufent[5];
                  captaent.ptaent[3]=bufent[6];
                  CAPTA=(float)captaent.porcenent;
                }
          PantallaTrasfArchivos();
          strcpy(path,path_dno);
          strcat(path,arch_com);
          if((fp2=fopen(path,"wb"))==NULL) TError(3);
          else mensaje(14);
        }

void Guardar_destino()
        { int i=3;
          while(i<bufent[1]+3)
                { if(!feof(fp2)) fputc(bufent[i++],fp2);
                  if(ferror(fp2)) TError(3);
                }
          gotoxy(42,12);cprintf("%.1f %",pt=pt+CAPTA);
          if(pt>=(inc_barra*10)) { gotoxy((49+inc_barra++),12);cprintf("²");}
          if(sal_enlace)
                { if( bufent[0]==M15 || bufent[0]==M16) Cerrar_destino();
                  if( bufent[0]==M13)  { puerto_com.Limpiar(); mensaje(14); }
                }
        }

void ParamActualTarjeta()
        { gotoxy(11,1);  cprintf(" ÉÍÍÍÍµ °°°±±±²²² CONFIGURACION ACTUAL ²²²±±±°°° ÆÍÍÍÍ» ");
          gotoxy(11,2);  cprintf(" º                                        º ");
          gotoxy(11,3);  cprintf(" º                    CANAL 1:            º ");
          gotoxy(11,4);  cprintf(" º                    CANAL 2:            º ");
          gotoxy(11,5);  cprintf(" º    TIEMPO DE GRABACION DE POSFALLA:           º ");
          gotoxy(11,6);  cprintf(" º          UMBRAL DE SENSIBILIDAD:          º ");
          gotoxy(11,7);  cprintf(" º VERIFICACION DE REGISTROS PENDIENTES:              º ");
          gotoxy(11,8);  cprintf(" º                                        º ");
          gotoxy(11,9);  cprintf(" º DESEA CONFIGURAR:                                 º ");
          gotoxy(11,10); cprintf(" º                              ÚÄÄÄÄÄÄÄ½ ");
          gotoxy(11,11); cprintf(" ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¾ OPCION: ");
          gotoxy(13,12);cprintf("ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß"); gotoxy(52,3);
          switch(bufent[3])
                { case 0: cprintf("INACTIVO"); gotoxy(52,4); cprintf("INACTIVO"); break;
                  case 1: cprintf("INACTIVO"); gotoxy(52,4); cprintf("ACTIVO");   break;
                  case 2: cprintf("ACTIVO"); gotoxy(52,4); cprintf("INACTIVO");   break;
                  case 3: cprintf("ACTIVO"); gotoxy(52,4); cprintf("ACTIVO");     break;
                }
          lu.u[0]  = bufent[4];
          lu.u[1]  = bufent[5];
          lu.u[2]  = bufent[6];
          lu.u[3]  = bufent[7];
          tpf.t[0] = bufent[8];
          tpf.t[1] = bufent[9];
          gotoxy(52,5); cprintf("%u segs.",tpf.T_PostFalla);
          gotoxy(52,6); cprintf("%.3f",lu.Limite_Umbral);
          gotoxy(52,7); cprintf("%d Hrs.",bufent[10]);
          gotoxy(32,9); Text(5); gotoxy(72,11);
        }
```

```
void ParamModTarjeta()
      { unsigned char OpCanal=0;
        clrscr(); ParamActualTarjeta();
        OpCanal=getch();
        while(!(OpCanal==13 || OpCanal==27)) { OpCanal=getch(); Sonido(1,2); }
        if(OpCanal==13)
            { gotoxy(6,13); cprintf("CONFIGURACION A DAR DE ALTA");
              gotoxy(6,14); cprintf(" º                                              º ");
              gotoxy(6,15); cprintf(" º                                              º ");
              gotoxy(6,16); cprintf(" º                                              º ");
              gotoxy(6,22); cprintf(" º                                              º ");
              gotoxy(6,23); cprintf(" º                                              º ");
              gotoxy(6,24); cprintf(" ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼ ");
              gotoxy(8,25); cprintf("ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß");
              gotoxy(18,15); Text(5);
              gotoxy(9,16); cprintf("ACTIVAR CANAL 1:");
              OpCanal=getch();
              while(!(OpCanal==13 || OpCanal==27))       { Sonido(1,2); OpCanal=getch(); }
              gotoxy(26,16);
              if(OpCanal==13) { ConfTarjeta=2; cprintf("SI");}
              else cprintf("NO");
              gotoxy(9,17); cprintf("ACTIVAR CANAL 2:");
              OpCanal=getch();
              while(!(OpCanal==13 || OpCanal==27))       { Sonido(1,2); OpCanal=getch(); }
              gotoxy(26,17);
              if(OpCanal==13) { ConfTarjeta=ConfTarjeta+1; cprintf("SI"); }
              else cprintf("NO");
              gotoxy(9,19); cprintf("    UMBRAL DE SENSIBILIDAD (1.000 -  2048.000):");
              gotoxy(9,20); cprintf("      GRABACION DE POSFALLA (1  -  32767) segs:");
              gotoxy(9,21); cprintf("HORA PARA VERIFICAR REGISTROS PENDIENTES (0-23):");
              ConvEntDatos(4,57,19,8, 2048,0);
              ConvEntDatos(6,57,20,5,32767,0);
              ConvEntDatos(7,57,21,2,23,0);
              gotoxy(13,22); cprintf("DAR DE ALTA ESTA CONFIGURACION ?: ");
              gotoxy(18,24); Text(5);
              gotoxy(47,22);
              OpCanal=getch();
              while(!(OpCanal==13 || OpCanal==27))       { Sonido(1,2); OpCanal=getch(); }
              if(OpCanal==13) mensaje(8);
            }
         PantallaPrincipal(PC);
        }

 void ReConfTarjeta()
       { ConfTarjeta=bufent[3];
         lu.u[0]= bufent[4];
         lu.u[1]= bufent[5];
         lu.u[2]= bufent[6];
         lu.u[3]= bufent[7];
         tpf.t[0]=bufent[8];
         tpf.t[1]=bufent[9];
         Hora_Ver_Reg_Pend=bufent[10];
         UMBRAL_DE_FALLA = lu.Limite_Umbral;
         T_de_F = tpf.T_PostFalla;
        }
```

```
void Sincronizacion()
        { char n=0,inten=1;
          clrscr();
          gotoxy(20,3);  cprintf(" ÉÍÍÍÍÍÍÍÍÍµ              ÆÍÍÍÍÍÍÍÍÍ» ");
          gotoxy(20,4);  cprintf(" º                                   º ");
          gotoxy(20,5);  cprintf(" º  OPCION:   LUGAR:                 º ");
          gotoxy(20,6);  cprintf(" º                                   º ");
          gotoxy(20,7);  cprintf(" º    L  :                           º ");
          gotoxy(20,8);  cprintf(" º    R  :                           º ");
          gotoxy(20,9);  cprintf(" º   ESC :   CANCELAR                º ");
          gotoxy(20,10); cprintf(" º                         ÚÄÄÄÄÄÄÄ½ ");
          gotoxy(20,11); cprintf(" ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¾ OPCION: ");
          gotoxy(22,12); cprintf("ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß");
          gotoxy(34,3); cprintf("SINCRONIZACION");
          gotoxy(34,7); cprintf("COMPUTADORA LOCAL");
          gotoxy(34,8); cprintf("TERMINAL REMOTA");
          while( !(n=='L' || n=='l'|| n=='R' || n=='r' || n==27) )
                { gotoxy(63,11); n=getch();
                  if(n=='L' || n=='l'|| n=='R' || n=='r' || n==27) break;
                  Sonido(inten,(inten*2));
                  cprintf("\r\n\n - La eleccion no concuerda !\r\n Elija nuevamente: intento ");
                  cout<<(int)inten;
                  if((inten++)==3) break;
                }
           if(n=='L' || n=='l') Hora_local();
           if(n=='R' || n=='r')
                { if(PC) mensaje(5);
                  else   mensaje(2);
                }
           else PantallaPrincipal(PC);
        }

 void Hora_local()
        { time(&t0);
          gotoxy(2,15);  cprintf("FECHA     HORA     A¥O");
          gotoxy(13,16); cprintf("       ");
          gotoxy(2,16);  printf("%s",ctime(&t0));
          R_hora_local();
        }

 void R_hora_local()
        { char op;
          printf("\n SINCRONIZAR: < Aceptar [ENTER]  Cancelar [ESC] >:");
          op=getch();
          while(!(op==13 || op==27))
                { Sonido(1,2); op=getch(); }
          if(op==13)
                { cout<<"\n\n";
                  system("time");
                  system("date");
                }
        }
```

```
void Hora_remota()
        { double pe,pf;
          unsigned char hra,min,seg,csg;
          clrscr();
          reloj.t[0]=bufent[3];
          reloj.t[1]=bufent[4];
          reloj.t[2]=bufent[5];
          reloj.t[3]=bufent[6];
          pf = reloj.sistema/(3600*CLK_TCK);                    //Proceso de conversión de tiempo
          pf = modf(pf, &pe)*60;  hra = pe;
          pf = modf(pf, &pe)*60;  min = pe;
          pf = modf(pf, &pe)*100; seg = pe;
          csg=pf;
          cprintf("\r\n HORA REMOTA ACTUAL: %02d:%02d:%02d:%02d\n\n",hra,min,seg,csg);
          Sinc_remota();
        }


void Sinc_remota()
        { char pw[10];
          char i;
          cprintf("\r\n Introduzca la CLAVE para sincronizar la terminal remoto:");
          for(i=0;i<8;i++)
                { pw[i]=getch(); putchar('±');
                  if(pw[i]==13) { pw[i]=0; break; }
                }
          if(!strcmpi(pw, CLAVE)) mensaje(1);
          else   { Sonido(1,3);
                   printf("\n\n °±²Û²±°   Su CLAVE no coincidio!  °±²Û²±°");
                   delay(3000);
                 }
          PantallaPrincipal(PC);
        }


void Resincronizar()
        { reloj.t[0]=bufent[3];
          reloj.t[1]=bufent[4];
          reloj.t[2]=bufent[5];
          reloj.t[3]=bufent[6];
          biostime(1,reloj.sistema);
          PantallaPrincipal(PC);
        }


Simultimbre(void)
        { unsigned char t;
          puerto_com >> t;
          if(t=='3') return 1;
          return 0;
        }
```

```
void Transf_texto()
        { unsigned char datoent,datosal,i,j;
          int xl,xr,yl,yr,dl,dr,salida=1;
          xl=xr=1;
          yl=4;yr=15;
          do{ if(kbhit())                                                    // Transmisor
                { datosal = getch();
                  if(datosal==0)
                        { datosal = getch();
                          if(datosal==59){ puerto_com <<(datosal=7);
                                           puerto_com <<"SE ENCUENTRA ALGUIEN...";
                                         }
                        }
                  if(datosal >= 8 && datosal <= 126)
                        { puerto_com << datosal;
                          if(datosal==13) { xl=1; ++yl; }
                          if(datosal==8)
                                { datosal=0;
                                  --xl;
                                  if(xl==0) { --yl;xl=80;
                                              if(yl==3) { yl=4; xl=1; Sonido(4,1); }
                                            }
                                  gotoxy(xl+1,yl);
                                  cprintf("%c",datosal);
                                  goto L;
                                }
                          ++xl;
                          gotoxy(xl,yl); cprintf("%c",datosal);
                          if(xl==80) { ++yl; xl=1; }
                          if(yl==13) { for(dl=4;dl<13;dl++) { gotoxy(1,dl); Text(6); }
                                       gotoxy(xl,yl=4);
                                     }
                          if(datosal==27) salida=0;
                          L;
                        }
                }
            else { puerto_com >> datoent;                                    // Receptor
                   if(((inportb(dpuerto+MSR) & 0x80) != 0x80) || datoent==27) salida=0;
                   if(datoent >=1 && datoent <=126)
                        { if(datoent==13) {xr=1;++yr;}
                          if(datoent==6)  { sound(300);                      // Tono de ocupado
                                            delay(1000);
                                            nosound();
                                          }
                          if(datoent==7) { for (j=0;j<3;j++)                 // Timbrado
                                                { puerto_com << 6;
                                                  for(i=0;i<20;i++){ sound(500); delay(25);
                                                                     nosound(); delay(25);
                                                                     if(kbhit()) break;
                                                                   }
                                                  if(j==2) break;
                                                  for(i=0;i<25;i++){ if(kbhit()) {j=5; break; }
                                                                     delay(100);
                                                                   }
                                                }
```

```
                                    if(j==5) break;
                                    puerto_com <<"USUARIO NO PRESENTE... ";
                                   }
                        if(datoent==8)
                                { --xr; datoent=0;
                                  if(xr==0) { xr=80;--yr;
                                              if(yr==14) { yr=15; xr=1; Sonido(3,1); }
                                            }
                                  gotoxy(xr+1,yr);  cprintf("%c",datoent);
                                  goto R;
                                }
                        ++xr;
                        gotoxy(xr,yr); cprintf("%c",datoent);
                        if(xr==80) { ++yr; xr=1; }
                        if(yr==24)
                                { for(dr=15;dr<24;dr++) { gotoxy(1,dr); Text(6); }
                                  gotoxy(xr,yr=15);
                                }
                        R:
                       }
                }
         } while(salida);
         Sonido(3,3);   PantallaPrincipal(PC);
        }

void Medicion_actual()
        { int med[16];
          int c,i,j=0;
          int n=2;
          float C=(float)400/2048; // 400 ES LA ESCALA USADA
          for(c=0;c<16;c++)
                { med[c]=bufent[++n]<<8;
                  med[c]=med[c]|bufent[++n];
                }
          clrscr();
          cprintf("          °±²Û°±²Û°±²Û  MEDICION ACTUAL  °±²Û°±²Û°±²Û\r\n\n");
          for(i=0;i<10;i=i+8)
              { Text(10); cprintf(" LINEA: CORRIENTES: VOLTAJES:\r\nFase A\r\Fase B\r Fase C\r Neutro");
                gotoxy(8,4+i);  cprintf("%d",++j);
                gotoxy(15,6+i); cprintf("%3.4f",C*med[i+0]);
                gotoxy(36,6+i); cprintf("%3.4f",C*med[i+4]);
                gotoxy(15,7+i); cprintf("%3.4f",C*med[i+1]);
                gotoxy(36,7+i); cprintf("%3.4f",C*med[i+5]);
                gotoxy(15,8+i); cprintf("%3.4f",C*med[i+2]);
                gotoxy(36,8+i); cprintf("%3.4f",C*med[i+6]);
                gotoxy(15,9+i); cprintf("%3.4f",C*med[i+3]);
                gotoxy(36,9+i); cprintf("%3.4f",C*med[i+7]);
                printf("\r\n\n");
              }
           Text(8); getch();
           PantallaPrincipal(PC);
          }
```

C.7 CONTENIDO DE LAS FUNCIONES DE LA TERMINAL (TERMINAL.CPP)

```
void RevisarFallasPen()
        { double Nfp;
          if ((longitud = open(verif_falla, O_RDONLY)) == -1) TError(1);
          NCNAPend=filelength(longitud);
          close(longitud);
          modf(NCNAPend/TamNomArch, &Nfp);
          nfpen=Nfp;
        }

void CargarTelCenace()
        { if((fp1=fopen("C:\\DETECTOR\\REGFALLA\\TELCENAC.DAT","r"))==NULL) TError(6);
          else { while(!feof(fp1)) fscanf(fp1,"%s %s %s %s",Tel_cenace,num_nodo,NILL,LTVM);}
          V_NILL=atol(NILL);
          fclose(fp1);
        }

void CambiarTelCenace()
        { int i;
          char ch,x=2,y=1;
          clrscr();
          gotoxy(x,y+1);  cprintf(" ÉÍÍµ  CONFIGURACION LA COMUNICACION LOCAL  ÆÍÍ» ");
          gotoxy(x,y+2);  cprintf(" º                                    º ");
          gotoxy(x,y+3);  cprintf(" º         NODO REMOTO:                       º ");
          gotoxy(x,y+4);  cprintf(" º      TELEFONO REMOTO:                        º ");
          gotoxy(x,y+5);  cprintf(" º        # NODO LOCAL:                     º ");
          gotoxy(x,y+6);  cprintf(" º REINTENTOS DE LLAMADA:                         º ");
          gotoxy(x,y+7);  cprintf(" º      LAPSOS DE ESPERA:    segs.            º ");
          gotoxy(x,y+9);  cprintf(" ÌÍÍÍÍµ CAMBIO DE LA CONFIGURACION LOCAL ÆÍÍÍÍ¹ ");
          gotoxy(x,y+11); cprintf(" º           TELEFONO REMOTO:                 º ");
          gotoxy(x,y+12); cprintf(" º        # NODO LOCAL (1-5):               º ");
          gotoxy(x,y+13); cprintf(" º REINTENTOS DE LLAMADA (0-9):               º ");
          gotoxy(x,y+14); cprintf(" º LAPSOS DE ESPERA (0-60 seg):             º ");
          gotoxy(x,y+15); cprintf(" º                                    º ");
          gotoxy(x,y+16); cprintf(" º                                    º ");
          gotoxy(x,y+17); cprintf(" ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼ ");
          gotoxy(x+1,y+18);cprintf("ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß");
          gotoxy(x+3,y+16); cprintf("Para parametros por default,Presione ESC");
          gotoxy(x+26,y+3); cprintf("CENACE");
          gotoxy(x+26,y+4); cprintf("%s",Tel_cenace);
          gotoxy(x+26,y+5); cprintf("%s",num_nodo);
          gotoxy(x+26,y+6); cprintf("%s",NILL);
          gotoxy(x+26,y+7); cprintf("%s",LTVM);
          ConvEntDatos( 3,x+32, y+11,13,500,0);
          ConvEntDatos( 8,x+32, y+12,1,5,0);
          ConvEntDatos( 9,x+32, y+13,1,9,0);
          ConvEntDatos(10,x+32, y+14,2,60,0);
          if((fp1=fopen("C:\\DETECTOR\\REGFALLA\\TELCENAC.DAT","w"))==NULL) TError(6);
          else  { fprintf(fp1,"%s %s %s %s",Tel_cenace,num_nodo,NILL,LTVM); }
          fclose(fp1);
          V_NILL=atol(NILL);
        }
```

```
LineaOcupada(unsigned char i)
        { unsigned char esp=0;
          int V_LTVM;
          V_LTVM=atol(LTVM);
          esp=V_LTVM;
          if(i==V_NILL) return i;
          gotoxy(40,17); printf("TIEMPO DE ESPERA:");
          while(esp) { gotoxy(58,17); printf("%d segs ",esp--);
                       delay(1000);
                       if(kbhit()) { i=V_NILL; return i;}
                     }
          gotoxy(40,16); cprintf("INTENTO: %d",i+1);
          return i;
        }

void ComunicacionTerminal()
        { unsigned char ent,sal,d,intenrtx=0;
          int cc;
          sal_enlace=1;
          EnvArch=0;
          ProbarLinea();
          puerto_com.Limpiar();
          if(nfpen) EnvArch=1;
          else   PantallaPrincipal(PC);
          while(sal_enlace)
                { if(kbhit())   { sal = getch();
                                  if(sal==27) AbortTransfArch();                    // Salir del enlace
                                  if(sal==NULL) Teclas_F();
                                }
                  if(EnvArch) { if((af=fopen(verif_falla,"r"))==NULL) TError(1);
                                cc=0;
                                while( cc < NCNAPend ){ Array_temp[cc]=fgetc(af);
                                                        if(cc<TamNomArch) g_archvs[cc]=Array_temp[cc];
                                                        if(ferror(af)) TError(1);
                                                        ++cc;
                                                       }
                                fclose(af);
                                g_archvs[TamNomArch]=0;
                                Abrir_fuente();
                               }
                   else        { gotoxy(62,13);
                                 Reloj();
                                 if(fin_buf!=ini_buf)                               // Entradas de datos
                                   { puerto_com >> ent;
                                     if(((inportb(dpuerto+MSR) & 0x80) != 0x80))  AbortTransfArch();
                                     if(ent==21) MensajeErroneo(intenrtx++);
                                     if(ent>=M1 && ent<=M17)
                                             { d=Recepcion(ent);
                                               if(d==6) { intenrtx=0; T_mensajes(); } //Buen mensaje recibido
                                               else puerto_com<<d;                    // Mal mensaje recibido
```

APÉNDICE D

ESTÁNDAR RS-232C/V.24

D.1 INTERFASE RS-232C/V.24

La interfase RS-232C/V.24 cuenta con 25 conductores los cuales por lo regular terminan en los pines de un conector DB-25, o bien en un DB-9 en el cual sólo se utilizan 9 de ellos. Las características de estos conectores no están contempladas en el estándar mencionado anteriormente. Para la tarea de comunicación entre la PC (Monitor o UTR) y el módem, la interfase utiliza solamente nueve de los veinticinco conductores disponibles. La relación entre los números de pines y las señales de la interfase que se utilizan internamente en la PC se definen en la Tabla D.1. El estándar también define los voltajes usados en las señales lógicas de los conductores. Las señales lógicas dentro de la PC se codifican con niveles TTL (*Transmisor Transistor Logic*).

Tabla D.1 Definiciones de los pines y las señales de la interfase RS-232C/V.24.

Pines: DB-9	DB-25	*Señal*		*Descripción*
3	2	Tx	⇒	Salida de datos de la PC.
2	3	Rx	⇒	Entrada de datos.
7	4	RTS	⇒	Petición de emisión (*Request to send*), establecida por la PC cuando quiere emitir datos.
8	5	CTS	⇒	Listo para recibir (*Clear to Send*), recibida por la PC cuando el aparato está listo para recibir datos.
6	6	DSR	⇒	Datos listos para envío (*Data Send Ready*), recibida por la PC cuando el módem esta encendido y conectado.
5	7	GND	⇒	Señal de tierra.
1	8	CD	⇒	Detección de portadora (*Carrier Detect*), Recibida por la PC cuando el módem detecta la señal portadora.
4	20	DTR	⇒	Datos terminales listos (*Data Terminal Ready*), establecida por la PC siempre que la comunicación de datos se activa.
9	22	RI	⇒	Indicador de timbre, recibida por la PC cuando el módem esta recibiendo una señal de timbre.

Las señales más importantes para llevar a cabo la comunicación son la de transmisión de datos (Tx) y la de recepción de datos (Rx), éstas utilizan dos conductores por los que se puede enviar y recibir simultáneamente los datos en forma serial. El resto de las señales, a excepción de la tierra (GND), son señales de control de la PC para dialogar con el módem. Estas señales deben de estar completamente bajo el control del software de comunicaciones.

A continuación se describe la transformación de los datos seriales a paralelos y viceversa que se lleva a cabo en esta interfase.

D.2 PUERTO SERIE

Los detalles específicos del puerto serie se encuentran en [2,3,4,5,6]. En general dicho puerto se compone de una tarjeta adaptadora de comunicación, un circuito integrado, la circuiteria correspondiente y un conector. El puerto serie posee siete localidades de memoria que utiliza para registrar los datos entrada y salida (E/S), y una línea de interrupción de *hardware*. Por medio de estos elementos interactúa con el CPU (*Central Processing Unit*) de la PC. La línea de interrupción indica la presencia de un evento, el cual será identificado por medio de programación a través de una rutina activada por la misma interrupción, y quedará registrado en una de las localidades del puerto serie.

La mayoría de las PC's cuentan con 2 puertos seriales. En este trabajo, para la operación de la red, se utiliza solamente un puerto en cada PC. El puerto designado está conectado a su línea de interrupción de *hardware*. La Tabla D.2 muestra las direcciones de las localidades de E/S e interrupciones para cada puerto serie.

Tabla.D.2 Asignación de los puertos E/S e interrupciones para el correspondiente puerto serie

Puerto serie	Direcciones de E/S (Hex)	Interrupción	Dirección (Hex)
COM1	3F8 - 3FF	IRQ4	C
COM2	2F8 - 2FF	IRQ3	B
COM3	3E8 - 3EF	IRQ4	C
COM4	3E8 - 3EF	IRQ3	B

Como se puede observar en la Tabla D.2 la interrupción IRQ4 afecta tanto al puerto COM1 como al COM3 y la IQR3 a los puertos COM2 y COM4. Esta es una fuerte limitación de las PC´s para el manejo de puertos seriales.

D.2.1 UART

La interfase RS-232C/V.24 utiliza el circuito integrado 8250 conocido como UART (*Universal Asincronous Receiver/Transmiter*) para la conversión de datos de formato paralelo a formato serie y viceversa. La salida de datos se convierte del formato paralelo, usado por la PC, al formato serie, usado por el puerto. En dicha salida, el UART transforma los caracteres de código ASCII en una señal temporal con forma de onda cuadrada, la cual es enviada por el cable.

Sistemas con procesadores mayores que el 80286 usan circuitos integrados compatibles tales como el 16450 ó el 16550, cuyas tecnologías difieren únicamente en la velocidad de procesamiento y de acceso a los puertos de E/S, ya que son más rápidos. Sin embargo, las funciones de operación de estos circuitos son las mismas que las del 8250 [4]. A continuación se describe el funcionamiento del UART con la finalidad de conocer estas funciones para la manipulación de las entradas y salidas de los datos del puerto serie.

D.2.2 Funcionamiento del UART

Las funciones de operación del UART son controladas por 10 registros, clasificados en tres tipos: control, estado y memoria intermedia (*buffer*). Dichos registros son asociados a las localidades de E/S mencionadas en la sección D.2. La dirección de cada registro se obtiene mediante la dirección base del puerto serie correspondiente. El nombre de dichos registros así como sus relativas direcciones se muestran a continuación en la Tabla D.3.

Tabla D.3 Direcciones de los datos y registro de control que manipulan las funciones del UART

Direcciones I/O	*Tipo de registro*	
Base + 0	TXR	Retención de transmisión
Base + 0	RXR	Buffer de Recepción
Base + 0	DLL	Cierre del octeto divisor menos significativo de la razón en baudios
Base + 1	DLH	Cierre del octeto divisor más significativo de la razón en baudios
Base + 1	IER	Habilitación del interruptor
Base + 2	IIR	Identificador de interruptor
Base + 3	LCR	Control de línea
Base + 4	MCR	Control del módem
Base + 5	LSR	Estado de línea
Base + 6	MSR	Estado del módem
Nota: *Base* representa la dirección del puerto utilizado (3F8, 2F8, 3E8 y 2E8 Hex.)		

Los puertos de E/S de la tarjeta están incorporados a la memoria de la PC por lo que pueden ser accesados por el microprocesador. Los datos alojados en dichas localidades pueden ser desplazados dentro de la PC a través del bus de datos. En el proceso de comunicación, el UART modifica los datos contenidos en dichas localidades notificando a la unidad central cuando estos están listos. La Fig. D.1 muestra la disposición de los registros del UART. En el apéndice A se encuentran los detalles específicos de cada uno de ellos.

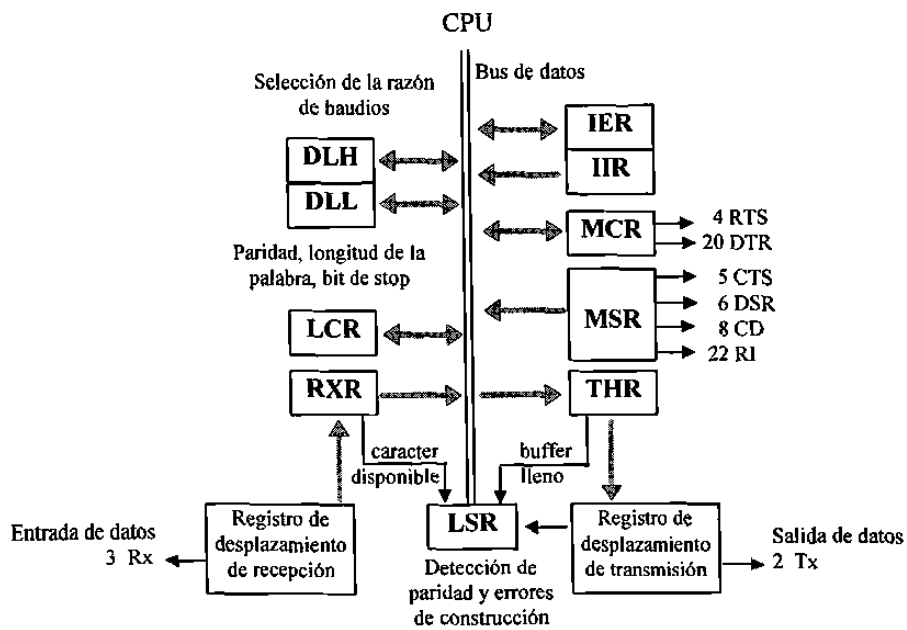


Fig. D.1 Disposición de registros del UART

La CPU accede al UART para realizar tres funciones principales: inicialización, transmisión de datos o recepción de datos. Estas dos últimas funciones son habilitadas e identificadas por medio de la interrupción mencionada en la sección D.2. Para esto, los bits del registro habilitador de interrupciones IER (*Interrupt Enable Register*) habilitan la función requerida por la interrupción, mientras que la rutina de servicio de interrupción examina el registro IIE (*Interrupt Identification Register*) para identificar cual de los dos eventos ha ocurrido. A continuación se explica cada una de las funciones mencionadas anteriormente.

D.2.2.1 Inicialización del UART

La configuración del puerto serie se realiza inicializando el UART. Esta inicialización consiste en declarar los parámetros de comunicaciones y el formato para la transmisión de datos. Los parámetros contemplan el número de puerto y la velocidad de transmisión. Este último se almacena en los registros DLH (*Divisor Latch High Register*) y DLL (*Divisor Latch Low Register*). El formato comprende la longitud del dato, la paridad, y la cantidad de bits terminales. Esta información se guarda en el registro LCR (*Line Control Register*). En la Tabla D.4. se resume la configuración del puerto serie.

Tabla D.4 Configuración del puerto de serie

Parámetros de comunicaciones y formato de los datos en el UART	
Número de puerto	COM1, COM2, COM3 ó COM4
Velocidad de transmisión	600, 1200, 2400, 4800 ó 9600 bps
Paridad	Nula, Par, Impar
Bits de datos	7 ú 8
Bits terminales	1 ó 2

La configuración del UART se realiza al inicio de la activación de la red. Normalmente se utiliza el puerto COM2 y una velocidad de 9600 bps. El formato de los datos se mantiene fijo con ocho bits de datos, paridad nula, y un bit terminal. En caso de una

interrupción parcial de energía, la UART debe de reconfigurarse por lo que los datos de la configuración original son guardados en un archivo, el cual será leído en el momento del arranque para una configuración automática.

D.2.2.2 Función de transmisión en el UART

El transmisor UART consiste de una entrada paralelo, el registro de retención THR (*Transmiter Holding Register*) y un registro de desplazamiento de transmisión TSR (*Transmiter Shift Register*). En la Fig. D.2 se muestra el diagrama a bloques del transmisor UART.

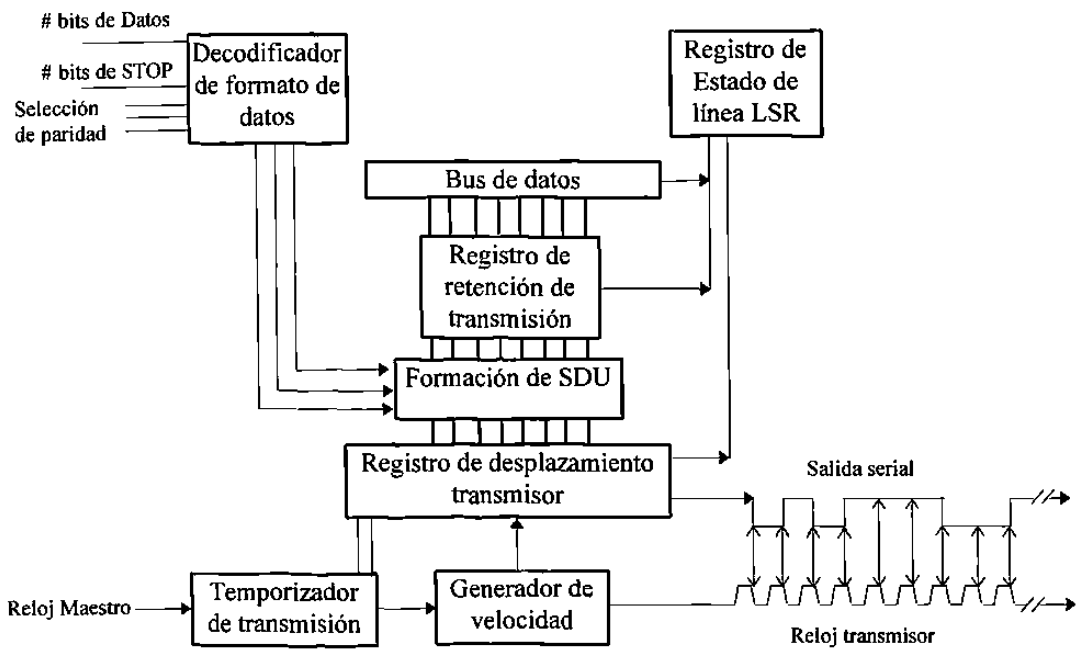


Fig.D.2 Diagrama a bloques del transmisor UART

En la transmisión de datos, la CPU a través del bus de datos consulta el estado de la línea en el registro LSR *(Line Status Register)* para ver si el UART está preparado para transmitir nuevos datos. Si el UART está disponible, la CPU escribe el octeto en el registro THR para que posteriormente sea presentado a la sección de transmisión. Esta sección consiste del registro TSR, el control lógico para cargar el octeto dentro de este registro y una memoria intermedia de transmisión. El control lógico forma, a partir del octeto escrito en el registro THR, en la SDU (*Serial Data Unit*), la agrupación del bit de inicio, los bits de datos, los bits terminales, y el bit de paridad. Posteriormente, esa agrupación binaria se carga dentro del registro TSR para ser desplazada y formar asi la salida serial que será muestreada en las transiciones negativas del reloj. Cuando todos los bits han sido movidos del TSR, se indica al registro LSR que la UART esta disponible para que se cargue el siguiente octeto a transmitir y se repita el mismo proceso.

El registro THR forma un pequeño almacén en el que los bits permanecen cargados temporalmente mientras se espera el término de la serialización. El octeto escrito dentro de este registro permanece inmóvil mientras se elabora la señal de salida.

D.2.2.3 Función de recepción en el UART

Los registros LSR y IIR indican e identifican respectivamente si un dato ha sido recibido bajo la presencia de una interrupción. Cuando esto sucede, la CPU lee el octeto recién alojado en el registro buffer de recepción RXR (*Receiver Buffer Register*) para que el programa principal pueda usarlo posteriormente por medio de una rutina de programación. El diagrama a bloques del receptor del UART se muestra a continuación en la Fig. D.3.

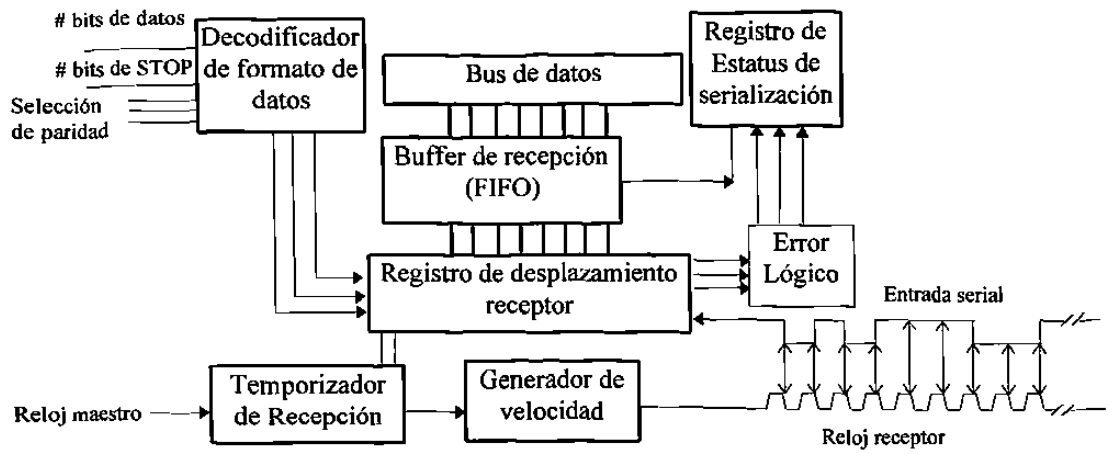


Fig. D.3 Diagrama a bloques del UART receptor.

El receptor monitorea constantemente la entrada de datos por la línea serial, esperando el bit de inicio. Cuando este bit es detectado, los bits recibidos son alojados sucesivamente dentro del registro de desplazamiento de recepción RSR (*Reception Shift Register*), y por medio de un decodificador de formato se extraen los bits de datos de la SDU y se guardan en el registro RXR. Una vez que la información recibida está disponible a través del bus de datos, el UART provoca una interrupción en el microprocesador, la cual lanza automáticamente la subrutina que se encarga de procesarla.

Cabe señalar que en el UART no existe un mecanismo de detección de errores por *hardware* cuando se realizan las acciones de transmisión o recepción. Esto únicamente se puede llevar a cabo por medio de *software*. A continuación se mencionan los virtuales errores que pueden presentarse en UART.

D.2.3 Generación de pulsos en el UART

En la transmisión, el UART genera los pulsos serialmente utilizando los bits almacenados en el registro TSR mencionado anteriormente (Sección D.2.2.2). Los bits de inicio y de parada se añaden según se precisen. El bit menos significativo (de inicio) de este

registro está unido a la línea serie de salida. El registro los va desplazando hacia la derecha, uno por uno según la razón en baudios, como se muestra en la Fig. D.4.

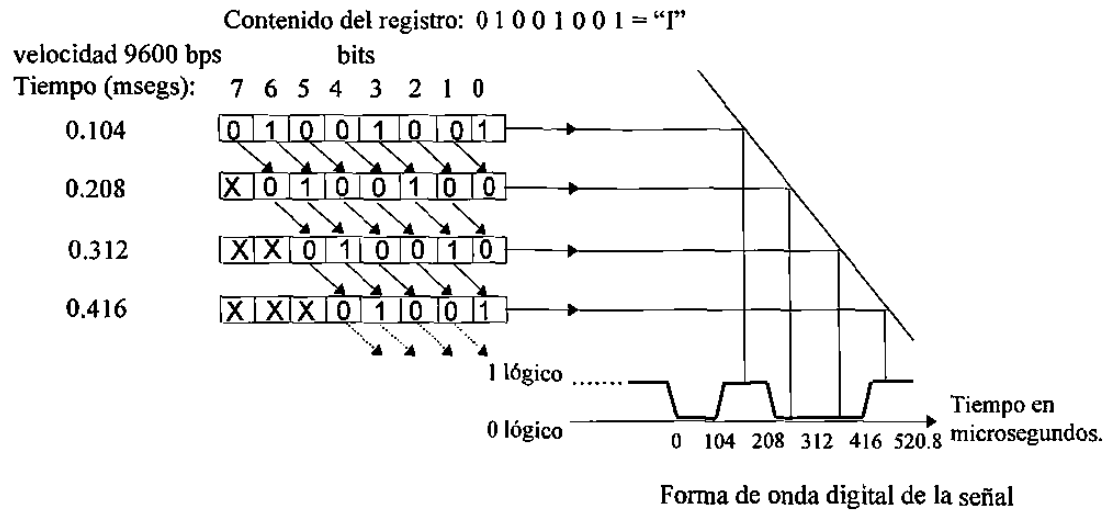


Fig. D.4 Registro de desplazamiento de transmisión del UART.

En la recepción, el proceso comienza por el borde inicial del bit de inicio, como se muestra en la Fig. D.5. El UART espera medio bit y entonces procede a muestrear la entrada a intervalos de un bit. Si el primer muestreo corresponde a una transición, se trata de un falso comienzo y el UART espera al siguiente bit de inicio. Si el décimo muestreo no corresponde a un bit de parada, el UART detecta un error de estructura. Una vez que un bit de parada ha sido detectado, el UART puede empezar inmediatamente a buscar el borde inicial del siguiente bit de inicio.

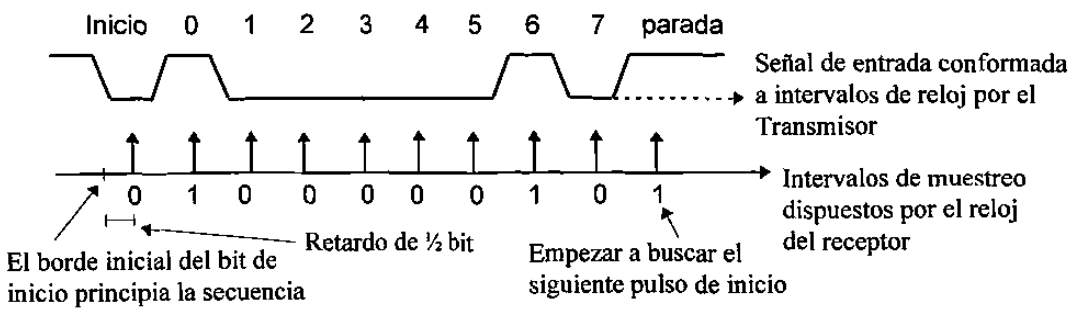


Fig. D.5 Entrada de datos serie

D.2.4 Errores de serialización del UART

Los errores de serialización se producen en la transmisión o recepción del UART. Este tiene la capacidad de detectarlos, pero no la de responder ante ellos. El software es el responsable de manejarlos. Una vez que un bit de inicio se detecta, el UART ensambla los bits recibidos aún cuando se trate de errores. Por esa razón la UART tiene un mecanismo para reportarlos. Los errores de serialización son reportados en registro LSR. A continuación se mencionan cada uno de ellos.

2.2.4.1 Errores de transmisión

Los errores posibles en el proceso de transmisión del UART son muy poco probables, sin embargo deben de tomarse en cuenta. Uno de estos errores, llamado error de sobre-escritura, es el que se produce cuando se escribe en el registro transmisor antes que halla sido vaciado. El error opuesto se da cuando el registro de desplazamiento se vacía, desocupando la línea serial; a esta condición se le conoce como error de retraso en la transmisión, y es irrelevante al menos que el UART esté alimentando un proceso síncrono tal como un módem. Bajo la suposición de que los errores de transmisión son poco importantes o al menos evitados fácilmente, muchos UART's no hacen reporte de esto [4,5].

2.2.4.2 Errores de recepción

A diferencia del proceso de transmisión, son varios los errores que pueden producirse durante la recepción. A continuación se mencionan cuatro de los principales:

- Cuando los octetos llegan más rápido de lo que pueden ser leídos por el programa, estos se almacenan en una memoria intermedia de recepción FIFO. Al llenarse el último

registro de ésta memoria se continua con el primero, repitiendo este proceso de manera cíclica. Esta pérdida de información por sobre-escritura en dicha memoria se conoce como error de rebasamiento en la recepción.

- Si el valor del bit de paridad no concuerda con los datos bajo el formato indicado el decodificado por el programa, un error de paridad es reportado.

- Cuando todos el bits de datos, incluyendo el de paridad, son nulos y se recibe un bit de término inválido es recibido, se asume que la línea serial ha sido interrumpida durante un intervalo de tiempo asignado a la SDU. Este error se reporta en el bit 4 del registro LSR y es el más importante, ya que implica interrupción de línea. Por esa razón el programa lo utiliza sistemáticamente antes de transmitir un octeto.

- Si se recibe un bit de término inválido, el UART determina si el ensamblado SDU contiene al menos un bit igual a uno. De ser así, o bien el bit de inicio fue inválido, o los sucesivos bits fueron dañados durante la transmisión. Dado que no es posible determinar con más precisión la naturaleza de este error, éste se reporta como un error de trama.

Tanto el transmisor como el receptor deben operar con la misma velocidad y formato de los datos. Una discordancia siempre generará un error.

APÉNDICE E

ESTÁNDAR PROFIBUS

E.1 ARQUITECTURA DEL PROTOCOLO PROFIBUS

PROFIBUS esta basado en una variedad existente de estándares nacionales e internacionales. La arquitectura de su protocolo se apoya en el modelo de referencia ISO-OSI (Ref. ISO 7498:1984; DIN ISO 74981B). Según el área de aplicación, PROFIBUS tiene tres protocolos distintos. A continuación en la Fig. E.1 se muestra la arquitectura de cada protocolo y en la Fig. E.2 se presentan las áreas de aplicación.

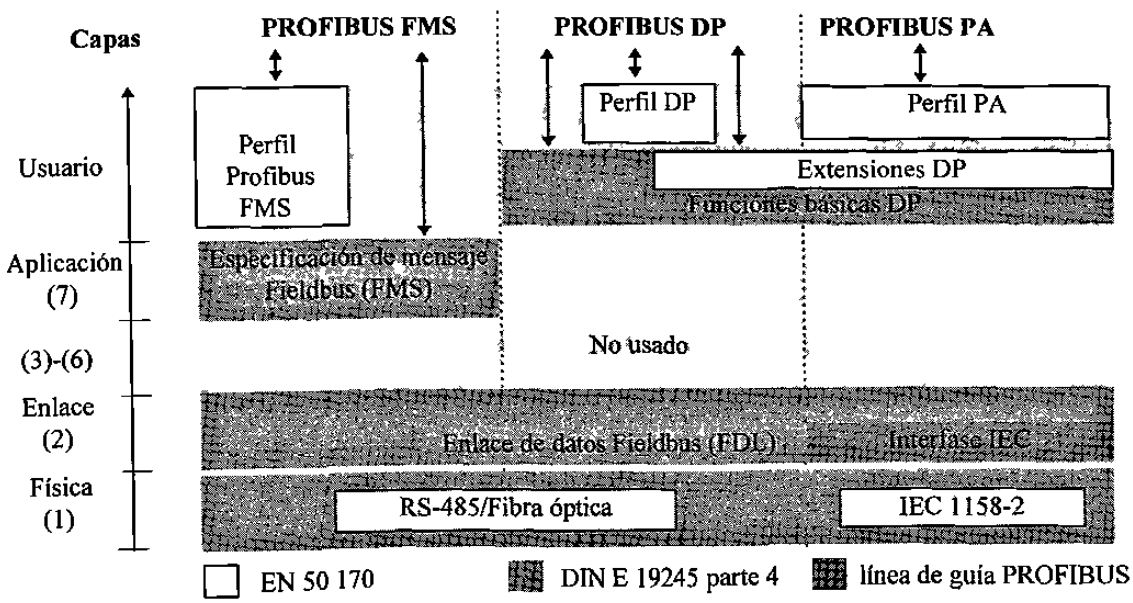


Fig. E.1 Arquitectura del protocolo PROFIBUS

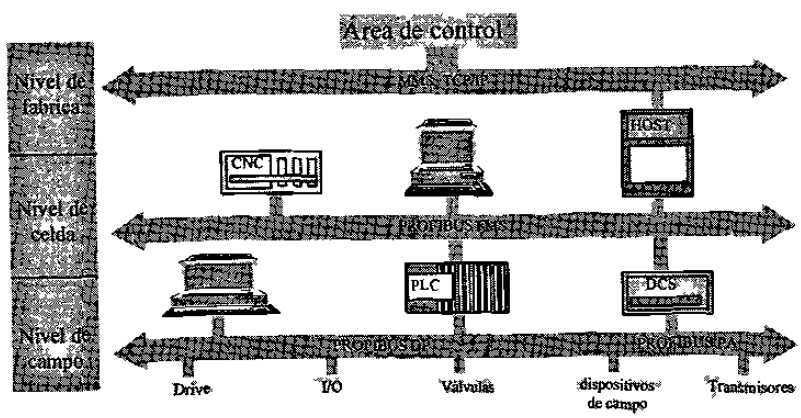


Fig. E.2 Áreas de aplicación de PROFIBUS

Las aplicaciones de cada protocolo se presentan a continuación.

E.1.1 PROFIBUS FMS (*Fieldbus Message Specification*)

Este ámbito es la solución de propósito general para tareas de comunicación entre dispositivos inteligentes y controladores a nivel de celda como se muestra en la Fig. E.2. FMS permite intercambio de información entre dichos dispositivos de forma cíclica y acíclica. En FMS para alcanzar una alta eficiencia así como bajo costo en el hardware y software, las capas 3 a 6 (Red, Transporte, Sesión, Presentación) no se toman en cuenta.

E.1.2 PROFIBUS DP (*Devices Perifericał*)

Este protocolo es optimizada para trabajar a altas velocidades. Específicamente diseñada para comunicación en tiempo crítico entre sistemas de control automático y dispositivos a nivel de campo. En DP, hay un asocie directo de enlace de datos (*DDLM*) para proveer al usuario de un fácil acceso a la Capa 2. DP presenta eficiencia, costo óptimo así como fácil conexión y movimiento de los dispositivos.

E.1.3 PROFIBUS PA (*Process Automation*)

Este ámbito es la solución para conectar sistemas de automatización y descentralizar dispositivos de campo. PA asegura la completa interoperabilidad e intercambiabilidad de los dispositivos de diferentes manufacturas. Esta versión emplea transmisión con seguridad intrínseca para trabajar en áreas con alto riesgo de interferencia. PA permite a la vez la transmisión de datos y la alimentación eléctrica de las estaciones sobre dos hilos.

En DP y PA las capas de la 3 a la 7 no son usadas por la misma razón que en FMS, la capa de Aplicación es omitida para obtener una más rápida y eficiente transmisión en los datos.

La capa física de cada versión usa el mismo protocolo de MAC (*Medium Access Control*) y de transmisión y tiene una interfase común a la capa de Aplicación, incluyendo sus servicios de comunicación. El protocolo de MAC, los servicios de transferencia de datos y la gestión de servicios se definen por el estándar DIN 19 241 (Parte 2), PROWAY C (IEC 955/ 1989), ISO/DIS 8802-2:1987 e ISO/IEC JTC 1/SC 6N 4960.

E.2 PROPIEDADES BÁSICAS

PROFIBUS tiene características técnicas y funcionales de un *Fieldbus* serial para interconectar dispositivos digitales distribuidos o sistemas con baja operación de ejecución tales como PLC's, controladores numéricos, transmisores, actuadores, sensores etc.

Con frecuencia un sistema de control está basado sobre una central de control y una unidad de supervisión a la que se le conectan un determinado número de dispositivos periféricos. En este caso, la transferencia dominante de datos es orientada centralmente y cíclica, de los dispositivos hacia la unidad de procesamiento central. En PROFIBUS dicha transferencia se lleva a través de estaciones maestras y esclavas.

Las estaciones maestras (EM) son las que habilitan el control del Bus. Cuando la estación tiene el derecho de acceso al Bus, denominado *token*, ésta puede transferir mensajes sin petición remota obligatoria. Por otra parte, las estaciones esclavas (EE) son simplemente dispositivos periféricos (sensores, actuadores, transmisores, PLC's, CN's, etc.). Estos no tienen derecho de controlar el Bus, únicamente pueden recibir la solicitud de una maestra, mensajes de reconocimiento o transmitir mensajes a la maestra. Las EM's son llamadas "estaciones maestras" mientras que las EE se nombran "estaciones pasivas".

El *token* circula en un anillo lógico formado por las EM's. Si el sistema contiene únicamente una central maestra de control y una estación de supervisión, el token no es necesario. Una EM con una o más EE representan un sistema puro. La mínima configuración se reduce a una EM con una EE o dos EM's.

E.3 CAPA FÍSICA

PROFIBUS tiene tres medios de transmisión según la aplicación que se requiera. Estos se definen a continuación:

- *EIA RS-485.* Es el estándar empleado por DP y FMS. Se define como la versión base de transmisión para aplicaciones de manufactura, construcción automatizada y controladores. Este estándar usa un par de cable torcido, con protección opcional.
- *IEC 1158-2.* Este estándar lo define PA para transmisión con seguridad intrínseca y alimentación de energía a las estaciones por el mismo Bus. Este estándar cubre los requerimientos de las industrias químicas y petroquímicas.
- *Fibra óptica*: Este medio de transmisión es utilizado para aplicaciones en ambientes con muy alta interferencia electromagnética y para incrementar la longitud del Bus a altas velocidades de transmisión.

Para cubrir la variedad de requerimientos tales como la longitud de la línea, número de estaciones, velocidad de transferencia de los datos y protección respecto a influencias ambientales, la capa Física tiene varias versiones. Únicamente la versión 1 se tratará en este Capítulo, ya que es la más conocida, sencilla y fácil de manipular.

E.3.1 Versión 1

La especificaciones de la versión 1 de PROFIBUS describen la línea de transmisión correspondiente al estándar EIA RS-485 para operar con DP o FMS. La máxima longitud por segmento de Bus es 1.2 Km. para velocidades de 9.6 Kbps. Con velocidades de 12000 Kbps, la distancia se reduce a 100 m.

E.3.1.1 Características eléctricas

- *Topología:* Bus lineal terminado en ambos extremos.

- *Medio físico:* Par torcido con protección (opcional), impedancia entre 100 y 130 ohms, área mínima del conductor 0.22 mm² , capacitancia entre conductores inferior a 60 pF/m.
- *Longitud por segmento del Bus y velocidad de los datos:*

Velocidad (Kbps):	9.6	19.2	93.75	187.5	500	1500	12000
Distancia/seg (mts):	1200	1200	1200	1000	400	200	100

- *Número de estaciones:* 32 en cada segmento sin repetidores y hasta 127 con repetidores.
- *Velocidad de los datos:* entre 9600 bps hasta 12 Mbps.

La longitud y el número de estaciones conectadas pueden ser incrementadas usando repetidores. Hasta siete repetidores pueden ser conectados en línea con velocidades de 500 kbps, y cuatro repetidores con 1500 kbps. Con dos repetidores a una velocidad de 93.75 kbit/s se puede cubrir hasta 92 estaciones sobre una distancia de 3.6 Km. Esta topología se muestra a continuación en la Fig. E.3.

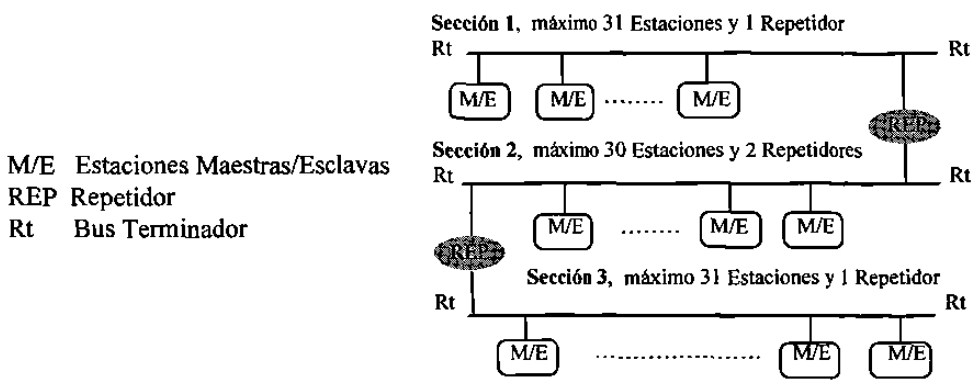


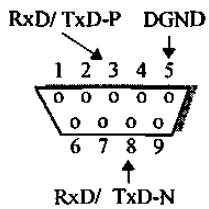
Fig. E.3 Repetidores en topología Bus lineal

E.3.1.2 Técnica de conexión, especificación mecánica y eléctrica

- *Bus conector.* Cada estación se acopla al Bus con conectores DB-9. El lado del conector hembra se localiza en la estación, mientras que el lado macho es montado al cable del Bus.

Las características eléctricas y mecánicas están especificadas en ISO 4902-1980 (DIN 41652, Parte 1). Para la conexión entre la sección del Bus y las estaciones, se emplean conectores tipo "T", con dos conectores machos y uno hembra. Estos conectores permiten la conexión y desconexión de estaciones sin cortar el cable y sin interrumpir la operación.

- *Designación de contactos.* Los pines asignados al conector se muestran en la Fig. E.4.



Designación de contactos

Pin No	RS-485 Ref.	Nombre de la señal	Significado
1		Tierra	Protección, Tierra de protección
2		RP	Reservado para energía
3	B/B'	RxD/TxD-P	Receptor/Transmisor de datos P
4		CNTR-P	Control-P
5	C/C'	DGND	Tierra de los datos
6		VP	Voltaje positivo
7		RP	Reservado para energía
8	A/A'	RxD/TxD-N	Receptor/Transmisor de datos N
9		CNTR-N	Control-N

Fig. E.4 Pines de salida, vista frontal del conector macho y vista trasera del conector hembra respectivamente.

La tierra de los datos conectada al pin 5 y el voltaje positivo conectado al pin 6, alimentan el Bus terminador. El control de las señales conectada a los pines 4 y 9, soportan el control de dirección cuando son usados repetidores sin capacidad de autocontrol. Los pines 2 y 7 se reservan para separar energía remota de los dispositivos de campo. La definición de esta señalización y la energización relativos a los pines 2, 4, 7 y 9 no están contemplados en este capítulo.

- *Cable del bus.* El medio PROFIBUS (versión 1) es un par de cable torcido. Las características eléctricas se mostraron en la sección E.3.1.1. Si no existen severas interferencias electromagnéticas (*EMI*), puede usarse un par de cable torcido sin protección. La selección del criterio del cable se encuentran en el apéndice de US estándar EIA RS-485. El mínimo alambrado entre estaciones se muestra en la Fig. E.5. Este alambrado permite un modo común de voltaje entre ambas estaciones de al menos 7 V.

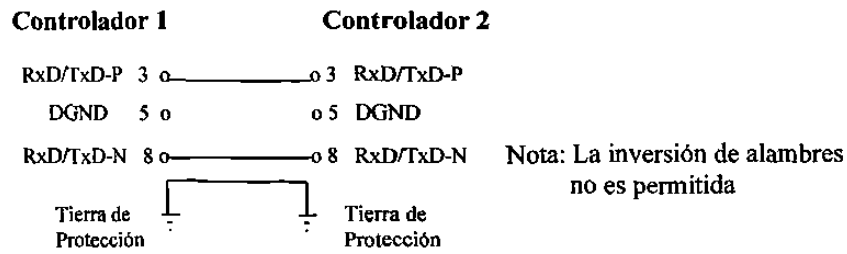


Fig. E.5 Alambrado de interconexión

- *Tierra y protección.* En el par de cable torcido con protección se recomienda conectar ésta a la tierra de protección de ambos extremos del cable obteniendo baja impedancia en las conexiones. Esto genera una razonable compatibilidad electromagnética (*EMC*). La tierra y protección se basan de acuerdo a DIN/VDE 0160 y DIN 57 899/VDE 0800.

- *Bus terminador.* El cable del Bus termina en ambos extremos como se muestra en la Fig. E.6. El resistor de terminación Rt especificado en EIA RS-485 puede ser complementado por una resistencia de acoplamiento negativo Rd, conectado a la tierra de los datos DGND, y por una resistencia de acoplamiento positivo Ru, conectado al Voltaje positivo VP. Este complemento fuerza un modo de voltaje diferencial entre conductores así como también un valor definido cuando las estaciones no están transmitiendo (periodos inactivos). Cada estación, la cual es destinada para terminar la línea, alimenta un voltaje positivo de + 5 V ± 5% al pin 6 del Bus conector. Para este voltaje se recomiendan los valores de los siguientes resistores: Rt = 150 ohms ± 2%, min. 1/4 W; Ru = Rd = 390 ohms ± 2%, min. 1/4 W. La fuente de energía que alimenta el pin 6 (VP) deberá liberar una corriente de al menos 10 mA con las tolerancias de voltaje específico.

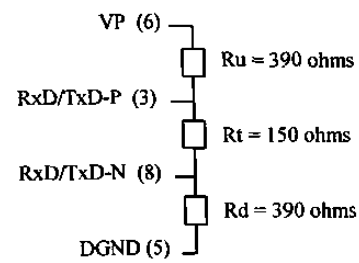


Fig. E.6 Bus Terminador

E.3.1.3 Método de transmisión

- *Codificación del bit:* El dato se codifica con señal NRZ (*No Return to zero*) y se transmite a través del par de cable torcido. El "1" binario se representa por una constante diferencial de voltaje positivo entre el pin 3 y el pin 8 del Bus del conector. El "0" binario se representa por una constante diferencial de voltaje negativo.

  La estructura del octeto (caracter) tiene el formato UART FT 1.2 (transmisión asincrona con bit de sincronización start-stop) empleado para equipo de telecontrol y de sistemas, el cual se especifica en IEC 870-5-1 (DIN 19 244 parte 101). Las definiciones del protocolo de transmisión se especifican en el estándar IEC 870-5-22, especificado por IEC TC 57 (DIN 19244 parte 521).

- *Control del transreceptor:* Cuando una estación no está transmitiendo, la salida del transmisor es deshabilitada. Esto presenta una alta impedancia en la línea, la cual se controla por la señal "*Request to Send*" (*RTS*). Durante periodos inactivos, la señal en la línea se representa por un "1" binario. De esta manera el bus terminador fuerza un voltaje diferencial positivo entre los pines 3 y 8 del conector cuando todos los transmisores se deshabilitan. Los receptores de línea están siempre habilitados, por tanto durante periodos inactivos la señal de "1" binario es recibida por varias estaciones.

E.3.2 Interfases entre la Capa física (*PHY*), el medio de acceso y el protocolo de transmisión (*FDL*).

En esta parte se incluye una descripción concisa de los servicios de datos que la capa PHY provee a la Capa FDL. Ésta soporta recepción y transmisión de símbolos FDL, los cuales son elementos de un caracter UART. A continuación en la Fig. E.7 muestra la relación de ambas capas.

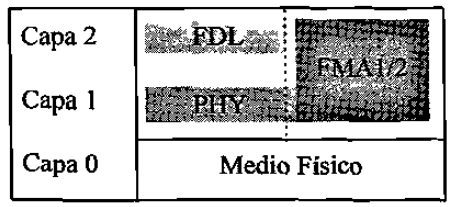


Fig. E.7 Interfase entre PHY y FDL en relación al modelo OSI

Los servicios de datos PHY incluyen dos primitivas de servicio: una petición, usada para solicitar un servicio por el controlador FDL y una indicación, para avisar una recepción del controlador FDL. Estas primitivas son: PHY_DATA.request y PHY_DATA.indication.

E.3.2.1 Especificación de los servicios e interacción

En esta sección se describe el servicio de las primitivas y la relación de los parámetros correspondientes a éstas. Los parámetros contienen información necesaria por la entidad de la Capa Física. El parámetro FDL_symbol de la primitiva PHY_DATA.request (*FDL_symbol*) puede tener alguno de los siguientes valores:

- ZERO. Corresponde a "0" binario
- ONE. Corresponde a "1" binario.
- SILENCE. Deshabilita el transmisor cuando es transmitido un inválido símbolo FDL.

La primitiva PHY_DATA.request se envía de la Capa FDL a la Capa PHY para solicitar que el símbolo FDL se transmita al medio Fieldbus. La recepción de esta primitiva causa que la Capa PHY codifique y transmita dicho símbolo usando la señalización correspondiente a la Capa PHY. La Capa PHY asegura esta primitiva con una confirmación primitiva definida localmente.

La primitiva PHY_DATA.indication se envía de la Capa PHY a la Capa FDL para indicar que un símbolo FDL fue recibido del medio Fieldbus. El valor del parámetro FDL_symbol de la primitiva PHY_DATA.indication (*FDL_symbol*) puede ser ZERO o ONE.

E.3.2.2 Requerimientos eléctricos y codificación

Las características físicas de la interfase entre las capas PHY y FDL como el tipo de conector, la asignación de los pines, los controladores (*drivers*), características del receptor etc., no se especifican. Sin embargo, experiencias muestran que al menos se requieren las señales de Transmisión de datos (*TxD*), Recepción de datos (*RxD*) y habilitación de transmisión (*Request to Send, RTS*) [17]. Un ejemplo de la señal se muestra en la Fig. E.8.

Codificación	PHY_DATA.request Transmitida PHY		PHY_DATA.indication Recibida PHY
Valor de los parámetros	TxD	RTS	RxD
ZERO	0	1	0
ONE	1	1	1
SILENCE	0/1	0	

Fig. E.8 Codificación de las señales TxD, RxD y RTS.

La Capa FDL genera el tiempo de trama para el periodo del símbolo FDL. Consecuentemente ningún tiempo de información es necesitado de la Capa PHY. Una señal adicional de control es pasada de la Capa FDL a la Capa PHY para ser avisada (*CTS, Clear to Send*).

E.3.2.3 Redundancia de la capa física y el medio (opcional)

El uso redundante de capas PHY se aplica para mejorar la seguridad del Fieldbus. Cuando se implementa redundancia en la Capa PHY, ésta contiene dos medios separados (Bus A y Bus B) y dos transreceptores (transmisor/receptor) completos por estación. Los principios básicos de la arquitectura de esta redundancia se muestran a continuación en la Fig. E.9.

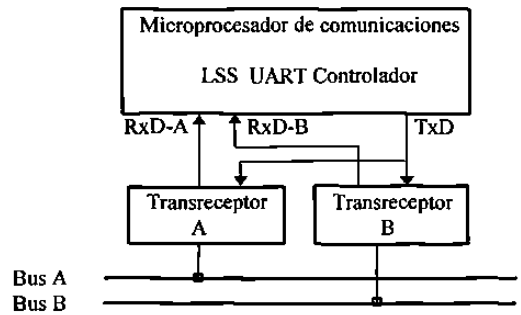


Fig E.9 Redundancia de la Capa Física y el Medio

Los datos se envían simultáneamente por los transreceptores sobre ambos medios (Bus A y Bus B). Mientras que cada estación solamente recibe uno del medio (Bus A o bus B). El canal receptor se selecciona por el switch selector de línea (*LSS*), el cual se localiza entre ambos transreceptores y el controlador UART, o si se usan dos controladores UART entonces entre los controladores y el microprocesador de comunicaciones. El switch selector de línea se controla completamente por la Capa FDL. Para esto el microprocesador de cada estación monitorea la acción del medio, independiente de cualquier otra estación. Las principales condiciones de conmutación para las EM y EE son las siguientes:

- Vencimiento de un temporizador.
- Ningún periodo de espera se detecta durante un intervalo de tiempo de sincronización.
- Recibo de dos o más tramas inválidas sucesivas, formato incorrecto del caracter UART, el bit de paridad o verificación incorrecta en la revisión de la secuencia de la trama detectada.

La selección del canal receptor, ya sea el A (primario) o el B (alternativo), se notifica al administrador Fieldbus, el cual provee esta información al usuario local a través de la interfase FMA1/2. No hay preferencia en el canal receptor después que la inicialización del sistema termina.

E.4 CAPA DE ENLACE DE DATOS

Las tres versiones de PROFIBUS usan el mismo protocolo de acceso al bus. Este protocolo lo implementa la capa 2 del modelo de referencia OSI. En PROFIBUS esta capa es llamada Enlace de Datos Fieldbus (*FDL*). El MAC especifica la operación cuando una estación puede transmitir datos, asegurando que únicamente tenga el derecho de transmitirlos en un lapso de tiempo [2,3,7]. El protocolo PROFIBUS toma en consideración dos requerimientos esenciales para el MAC:

- En el caso de una comunicación entre EM's con igual derecho de acceso, el protocolo tiene que asegurar que cada una de las estaciones tenga oportunidad de ejecutar labores de comunicación dentro del preciso intervalo de tiempo.

- Para la comunicación entre una EM y una EE (periférico) se realiza un intercambio cíclico de datos en tiempo real tan simple y rápido como sea posible.

PROFIBUS controla el acceso al medio por un método híbrido. Para la comunicación entre EM's emplea el método *"token passing"*, mientras que para la comunicación entre EM y EE utiliza el método *"maestra-esclava"*. El MAC puede ser realizado por cada EM. Las EE's actúan neutralmente con respecto al MAC, estas no pueden realizar transmisiones independientemente, únicamente peticiones.

El método *"token passing"* asegura, por medio del token, la asignación de derecho en el bus dentro de un intervalo de tiempo definido. El token es un telegrama especial que otorga el derecho para transmitir de una EM a la próxima, a través de un anillo lógico. El token circula en un tiempo de rotación máximo (configurable) entre todas las EM. El control de la rotación del token se maneja por cada estación conociendo a su predecesora (*PS*: Estación de la cual se recibe el token). Sucesivamente cada estación conoce su sucesora (*NS*: estación a la cual se transmite el token) y ésta conoce su propia dirección (*TS*: estación

actual). Después de inicializar los parámetros de operación por primera vez, cada EM determina las direcciones PS y NS de manera dinámica.

El método "maestra-esclava" permite a la EM que tiene el propio derecho de transmisión comunicarse con las EE. Cada EM tiene capacidad de obtener datos de las EE y transmitir datos a estas. Con el método híbrido de MAC de PROFIBUS es posible implementar sistemas puros tales como maestra-esclava, maestra-maestra, o un sistema combinado. Características adicionales en el MAC de PROFIBUS son detectar y reconocer:

- Defectos en el medio de transmisión o en los transreceptores,
- Errores en el direccionamiento de estaciones (duplicación),
- La adición y eliminación de estaciones durante operación,
- Errores en el *token* passing (multiplicación, pérdidas o el paso),
- Cualquier combinación de estaciones maestras y esclavas.

E.4.1 Procesos de transmisión y el controlador FDL

El intercambio de mensajes se lleva a cabo por ciclos. Un ciclo de mensaje consiste en accesar la trama de la estación maestra (petición o envío) y su trama de respuesta o su reconocimiento, bien sea de una EM o una EE. Los datos se transmiten en la trama activa (envío) o en la trama de respuesta. La trama de reconocimiento carece de datos.

El ciclo completo del mensaje se interrumpe únicamente por la transmisión del token y por la transmisión de datos sin reconocimiento. En ambos modos de operación no hay reconocimiento. En difusión de mensajes, una EM (iniciadora) direcciona a todas las otras estaciones al mismo tiempo por medio de un direccionamiento global.

La estaciones reconocen o responden únicamente cuando éstas son direccionadas. El reconocimiento o respuesta llega dentro de un tiempo predefinido, de lo contrario el iniciador

repite nuevamente la petición. Un reintento no se realiza por el mismo iniciador antes de expirar un periodo de espera (tiempo inactivo). Si el contestador no reconoce o responde después del número predefinido de intentos, éste es señalado como "No operacional" y se suspenden las repeticiones. Los modos de operación de la transmisión definen la secuencia de tiempo en los ciclos del mensaje. Cuatro tipos de modos se distinguen:

- Manipulación del Token
- Petición acíclica o envío / petición de operación
- Envío cíclico / petición de operación, sondeo
- Registro de estaciones

E.4.1.1 Procedimientos del token

- **Rotación del token.** El token se pasa de EM a EM en forma ascendente de las direcciones de las estaciones mediante una trama. Cuando se cierra el anillo lógico, la estación con la dirección más alta pasa el token a la estación con la dirección más baja, ver la Fig. E.10.

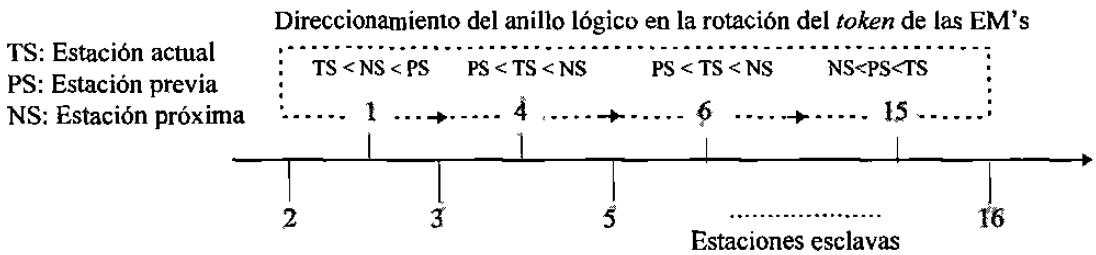


Fig. E.10 Rotación del token sobre el anillo lógico

Recepción del token. Si una estación maestra (TS) recibe una trama de token direccionada a sí misma de una estación registrada como previa (PS) en la lista de estaciones activas (*LAS*), este token ejecuta mensajes cíclicos. Si el transmisor del token no registra la dirección de la PS, el destinatario asume un error y no acepta el token. Únicamente un

subsecuente reintento de la misma PS se acepta asumiendo que ahora el anillo lógico ha cambiado. Este reemplaza la dirección original PS grabada en la LAS por la nueva.

Transmisión del token. Cuando terminan los ciclos de los mensajes, el token se pasa al sucesor (NS). Si después de transmitir el token, se recibe una trama válida, la estación receptora ejecuta los ciclos del mensaje. Si la trama es inválida, ésta asume que otra estación está transmitiendo. Y si no se recibe ninguna trama dentro del periodo definido, ésta nuevamente retransmite el token hasta recibir la trama correspondiente.

- **Adición y eliminación de estaciones.** En cualquier momento las estaciones pueden ser conectadas o desconectadas del medio de transmisión. En el anillo lógico cada EM es responsable de la adición o eliminación de estaciones. Para esto, las direcciones se contemplan en el rango de la propia estación TS a la próxima estación NS. Este rango de direcciones se llama GAP y se representa en una lista (*GAPL*). Cada EM revisa periódicamente el GAP bajo un intervalo de tiempo de actualización (*TGUD*) para cambios concernientes a estaciones. Esto se efectúa examinando una dirección por recibo de token, usando una trama denominada "*Request FDL Status*". Las direcciones son examinadas en orden ascendente, excepto el GAP que sobrepasa la dirección de la estación mas alta, en este caso el proceso continúa nuevamente de la dirección inicial.

  Si una estación se reconoce con el estado "EM no lista" o "EE", ésto se registra en la GAPL y la próxima dirección se revisa. Si una estación contesta con "Preparada para incorporar al anillo lógico", el poseedor del token modifica la GAPL y pasa el token a la nueva NS. La estación que ha sido admitida al anillo lógico tiene que reconstruir la lista LAS. Las estaciones pasivas que fueron registradas originalmente en la GAPL, si no responden a un repetido "*Request FDL State*", se eliminan de la GAPL y se registran como direcciones de estaciones sin uso.

- **Inicialización del token en el anillo lógico.** Cuando inicia el sistema PROFIBUS, la EM con la dirección más baja lleva a cabo la inicialización. Para informar cuales EM's están presentes en el anillo lógico, se transmiten dos tramas de token a sí mismo. Entonces se transmite la trama "*Request FDL Status*" a cada estación, incrementando la secuencia de direcciones en la manera que se van registrando. Si una estación responde con "EM no lista" o "EE", se registra en la GAPL. La primer estación que conteste con "Preparada para incorporar al anillo lógico" se registra como NS en la LAS y de esta manera se cierra el rango GAP del poseedor del token.

E.4.1.2 Disposición del controlador FDL

El controlador FDL de las EM's contempla 10 estados y transiciones entre ellos. La EE tiene únicamente dos estados. A continuación en la Fig. E.11 se muestra el diagrama combinado de los estados de la EM (estado 0 al 9) y la EE (estado 0 y 10).

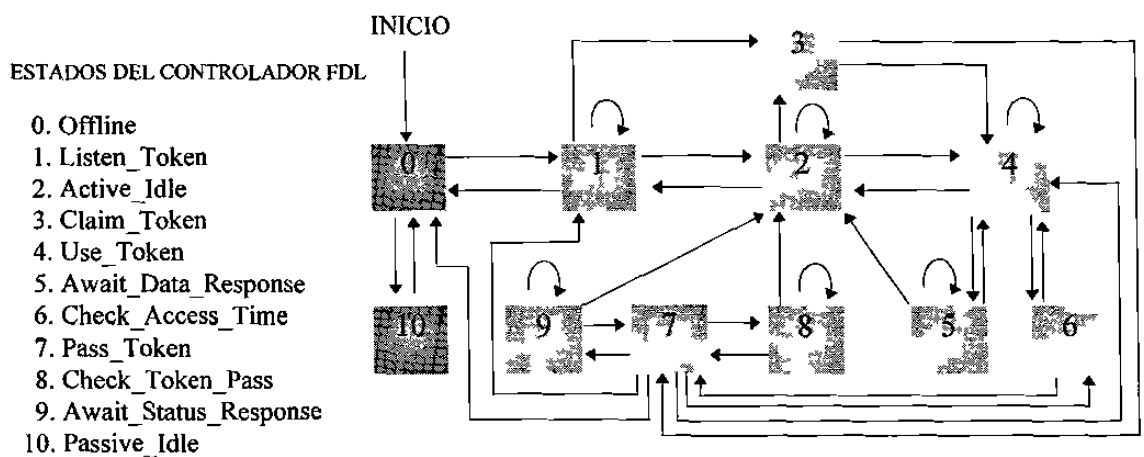


Fig. E.11 Diagrama del estado FDL

- **Inicialización del FDL.** Después de encender el controlador FDL de las EM's y EE's, se activa el estado "*Offline*". Dentro de este estado el controlador no recibe o transmite ninguna señal (tramas) del o al Bus repectivamente. En dicho estado el controlador FDL cambiará al estado "*Passive_Idle*" o "*Listen_Token*" únicamente si los parámetros de

operación establecidos manipulan el protocolo correcto. La Tabla E.1 muestra los parámetros de operación que pueden ser provistos por el administrador (*FMA1/2*).

Tabla E.1 Parámetros de operación del administrador FMA1/2

1	Dirección de la estación TS
2	Velocidad de señalización de datos(kbit/s)
3	Medio disponible: Simple/Redundante
4	Liberación de Hardware y Software
5	Tiempo de trama TSL
6	Mínimo tiempo de retardo de la estación TSDR
7	Máximo tiempo de retardo de la estación TSDR
8	Baja del transmisor/Tiempo de conmutación del repetidor TQUI
9	Configuración de tiempo TSET
10	Tiempo de ejecución de rotación TSET
11	Factor G del respaldo GAP
12	Alta/Baja de estaciones maestras del anillo lógico
13	Dirección más alta de estaciones (HSA)
14	Máximo número de reintentos (max_retry_limit)

Nota: Únicamente los parámetros del 7 al 14 se aplican a las EE's.

E.4.1.3 Tiempos de operación del Bus

Para operación del Bus, se definen los siguientes tiempos medidos en bits:

- *Tiempo de bit (tBIT).* Duración de un bit, igual al inverso de la velocidad de transmisión.
- *Tiempo de ejecución de rotación (TTR).* Es el predeterminado tiempo para la vuelta del token sobre el bus, incluyendo altas y bajas de estaciones, transacciones de prioridades, errores y mantenimiento del GAP.
- *Tiempo de sincronía (TSYN).* Intervalo mínimo durante el cual cada estación recibe el estado inactivo del medio de transmisión antes de aceptar el inicio de una trama o token.
- *Intervalo de tiempo de sincronía (TSYNI).* Mide el lapso máximo de tiempo permitido entre dos TSYN consecutivos.
- *Tiempo de retardo de la estación (TSDx).* Lapso entre la transmisión o recepción del último bit de trama hasta la recepción o transmisión del primer bit de la siguiente trama respectivamente.

- *Tiempo activo (TRDY).* Lapso durante el cual la EM debe estar lista para recibir un reconocimiento o respuesta después de la transmisión de una petición.
- *Actualización de tiempo (TSET).* Define el tiempo para la ocurrencia de un evento.
- *Tiempo pasivo (TQUI).* Periodo de tiempo que una estación transmisora espera después del fin de una trama antes de habilitar el receptor.
- *Tiempo de seguridad (TSM).* Intervalo de tiempo definido como margen de seguridad, donde: TSM = 2 bits + 2(TSET) + TQUI.
- *Tiempo inactivo (TID).* Lapso entre la transmisión del último bit de una trama, el cual no debe ser un reconocimiento y la transmisión del primer bit de la trama siguiente.
- *Retardo de transmisión (TTD).* Máximo lapso de tiempo en el medio de transmisión entre el transmisor y receptor cuando una trama se envía.
- *Tiempo de trama (TSL).* Lapso que espera el iniciador para recibir el primer caracter del inmediato reconocimiento ó respuesta, después de enviar el ultimo bit de la trama.
- *Tiempo de exceso (TTO).* Este tiempo sirve para supervisar la actividad del Bus y el tiempo TID en las estaciones. La supervisión inicia cuando se presenta el estado "*Listen_Token*" o "*Passive_Idle*" o después de recibir el ultimo bit de una trama. Este finaliza después de recibir el primer bit de la siguiente trama. Si el periodo inactivo TID excede el TTO, el Bus se declara inactivo. El valor es: $TTO = TSL(6 + 2n)$, donde n es de 0 a 126 direcciones para EM's y 130 para EE's, independiente de estas direcciones.
- *Tiempo de respaldo del GAP (TGUD).* Lapso de tiempo que la EM espera para el mantenimiento del GAP.

E.4.2 Temporizadores y contadores

El tiempo de rotación del token se mide y supervisa por los siguientes temporizadores:

- *Temporizador de rotación del token.* Cuando una EM recibe el token, este temporizador carga el TTR y lo decrementa cada tBIT. Cuando la estación recibe otro token antes de agotarse el TTR, el temporizador nuevamente lo restablece con el TTR.

- *Temporizador de estado inactivo.* Supervisa el estado en reposo de la estación en el Bus. Las estaciones "sin token" cargan este temporizador con TSYN después de la transmisión o recepción del ultimo bit de las tramas y entonces se decrementa cada tBIT. El receptor se habilita inmediatamente después que este tiempo se agota.

- *Temporizador de trama.* En una EM este temporizador, después una petición o paso del token, monitorea si la estación receptora responde o se activa dentro del tiempo TSL. Este tiempo se carga al temporizador, después de transmitir el último bit de una trama y se decrementa cada tBIT tan pronto como el receptor se habilite. Si el temporizador se agota antes que el primer bit de la trama se reciba, un error se indica. Entonces un reintento o un nuevo ciclo de mensaje se inicia.

- *Temporizador de exceso.* Después de transmitir o recibir el último bit de una trama, este temporizador se carga con un múltiplo de el TSL y decrece cada tBIT hasta que la nueva trama se recibe. Si el temporizador se agota un error fatal ocurrió.

- *Temporizador de intervalo de SYN.* Este temporizador supervisa el medio de transmisión cuando ocurre una sincronización en el receptor dentro del periodo TSYNI. Cada vez que el receptor se sincroniza, este temporizador se carga con TSYNI y desde el inicio de una trama se decrementa cada tBIT hasta que el nuevo TSYN se detecta. Si el temporizador se agota, un error ocurrió en el medio de transmisión.

- *Temporizador de mantenimiento del GAP.* Establece un lapso para realizar la revisión completa del GAP. Para esto, el temporizador se carga con un múltiplo del tiempo de rotación TTR. Únicamente las EM's usan este temporizador.

Cuando una estación maestra pasa al estado "*Listen_token*", el temporizador de estado inactivo se carga con TSYN, el temporizador de exceso con TTO, el temporizador de intervalo de sincronía con TSYNI y los otros temporizadores se restablecen. Si la estación

pasa al estado "*Passive_Idle*", el temporizador de exceso se carga con TTO y el temporizador de intervalo de sincronía con TSYNI. Para instalación y mantenimiento de los temporizadores, se definen los contadores siguientes:

1. Contador para recibir delimitadores válidos de inicio (*SD_count*)
2. Contador para recibir delimitadores inválidos de inicio (*SD_error _count*)
3. Contador para transmisión de reintentos (*Retry_count*)
4. Contador para transmir tramas (*frame_sent_count*), excepto para servicios SDN y FDL.

Cuando una estación esta en el estado "*Listen_token*" o "*Passive_Idle*", los contadores se habilitan e inicializan. Los contadores 3 y 4 se emplean exclusivamente por las EM's.

E.4.3 Estructura de las tramas

Cada trama consiste de un número caracteres UART, los cuales se basan sobre los estándares ISO 1177:1985, ISO 2022:1986 y CCITT V.4. Cada carácter contempla 11 bits: Un bit de inicio, 8 bits de información, un bit de paridad y un bit de *stop*. La reglas de transmisión para intercambiar tramas es el siguiente:

- El estado *idle* (inactivo) corresponde a un nivel de señalización de 1 binario.
- Cada trama activa se precede por al menos 33 bits idle (*TSYN*).
- Ningún estado *idle* se permite entre tramas de caracter UART.
- El receptor revisa por caracter el bit de inicio, el bit de parada y la paridad, y por trama el delimitador de inicio, DA, SA, FCS, el delimitador de fin y el SYN.

A continuación en la Fig. E.12 se muestra el formato y contenido de cada una de las tramas.

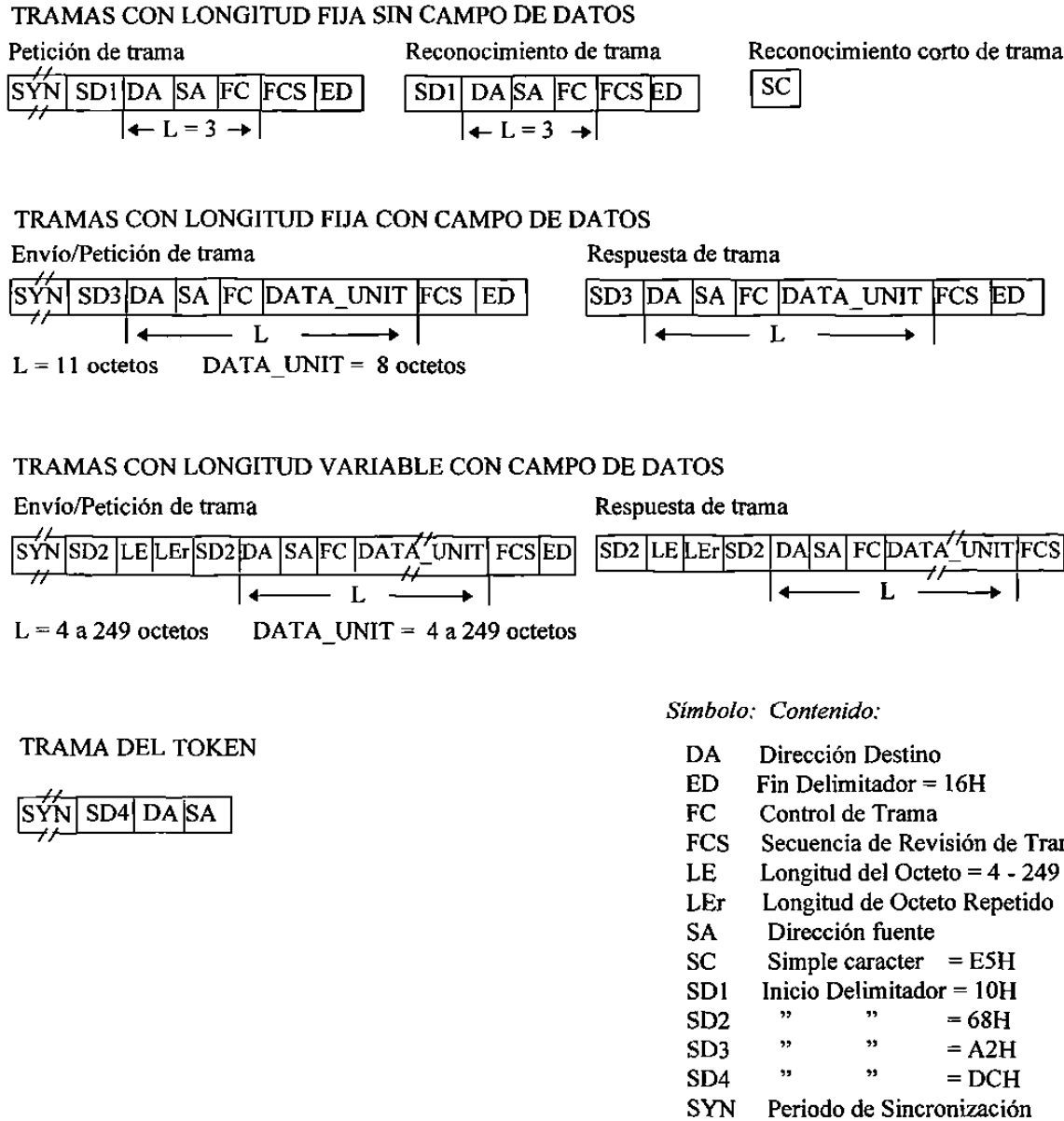


Fig. E.12 Formato y estructuras de las tramas

E.5 INTERFASE PROFIBUS DE LA CAPA 2 Y SU GESTIONAMIENTO (*FMA1/2*)

En las subsecciones siguientes se describe de manera concreta la transferencia de datos (*FDL, Fieldbus Data Link*) y la gestión de servicios de PROFIBUS (*FMA, Fieldbus*

Magnament) . Los servicios FDL son disponibles al usuario a través de la Capa 2. Los servicios FMA1/2 son disponibles a través de la gestión FMA1/2 asociada con las Capas 1 y 2.

E.5.1 Usuario FDL y la Interfase FDL

En esta sección se describen los servicios de transferencia de datos disponibles al usuario FDL de la capa 2 en una EM, denominado usuario local. Estos servicios son:

- **Envío de datos con reconocimiento (*SDA*).** Este servicio permite al usuario local, enviar datos (*L_sdu, Link service data unit*) a una estación remota. En ésta, al recibir el L_sdu libre de error, se pasa por el FDL al usuario remoto. El usuario local recibe una confirmación correspondiente al recibo o no recibo de los datos del usuario. Si un error ocurre durante la transferencia, el FDL del usuario local repite la transferencia de datos.

- **Envío de datos sin reconocimiento (*SDN*).** Este servicio concede al usuario local transferir datos a una estación remota, a varias o a todas las estaciones remotas al mismo tiempo. Este usuario recibe una confirmación de reconocimiento del fin de la transferencia, sin que los datos se reciban adecuadamente. En las estaciones remotas cuando el L_sdu se recibe libre de error, se pasa al usuario remoto de lo contrario este se ignora. Ante cualquier caso, no hay confirmación de que tal transferencia ha tomado lugar.

- **Envío y petición de datos con replica (*SRD*).** Permite al usuario local transferir datos a una estación remota y a la vez solicitar datos disponibles del usuario remoto en ese instante. En la estación remota cuando el L_sdu se recibe sin error, este se pasa al usuario remoto. El servicio también permite a un usuario local solicitar datos de un usuario remoto sin enviar datos a éste. El usuario local recibe el dato solicitado, una indicación que los datos no estuvieron disponibles o una confirmación de que los datos transmitidos no se recibieron. Las primeras dos reacciones también confirman el recibo de los datos transferidos. Si un error ocurre durante la transferencia, el usuario local repite la transferencia de datos con su solicitud.

- **Envío cíclico y petición de datos con réplica (*CSDR*).** Este servicio concede al usuario local transferir datos cíclicamente a una estación remota y al mismo tiempo solicitar datos de esta. En la estación remota el recibo del dato sin error se pasa cíclicamente al usuario remoto. El servicio también permite solicitar datos cíclicamente de la estación remota sin enviar datos a ésta. El usuario local puede recibir cíclicamente el dato requerido, una indicación de que los datos no están disponibles o una confirmación de no recibir los datos transmitidos. Las primeras dos respuestas también confirman la recepción de los datos transferidos. Si ocurre un error durante la transferencia, el FDL del usuario local repite la transferencia de datos junto con su solicitud. La selección de estaciones remotas, el número y la secuencia de transferencia de los datos con los requeridos datos por el modo cíclico se definen por el usuario local en la lista de sondeo (*Poll list*).

E.5.2 Interfase FMA1/2 con el usuario FMA1/2

Esta sección describe la gestión de los servicios provistos al usuario FMA1/2 y las primitivas de servicio asociadas. La interfase de servicios entre el usuario FMA1/2 y el FMA1/2 tiene las siguientes funciones:

- Restablecer las Capas 1 y 2 (local).
- Solicitar y modificar los parámetros de operación actuales de FDL, PHY y de los contadores (local).
- Notificar eventos inesperados, errores y cambios de estado (local y remoto).
- Solicitar para identificar y para configurar la LSAP de las estaciones (local y remoto).
- Activación y desactivación de las LSAP's locales.
- Solicitar para actualizar la lista de estaciones activas (Live List) por medio del estado FDL (remoto).

El FMA1/2 provee los siguientes servicios al usuario FMA1/2:

- **Restablecimiento del FMA1/2 (obligatorio).** Permite al usuario que el FMA1/2 restaure la Capa 1 (PHY), la Capa 2 (FDL) y a si mismo. Esto es equivalente a encender nuevamente el sistema. El usuario recibe una confirmación de esto.
- **Configuración de valores FMA1/2 (opcional).** Otorga al usuario asignar un nuevo valor a las variables de las Capas 1 o 2, recibiendo una confirmación si la variable específica cambio el nuevo valor.
- **Lectura de valores FMA1/2 (opcional).** Este servicio permite habilitar la lectura de las variables FMA1/2 de la Capa 1 y 2.
- **Eventos FMA1/2 (obligatorio).** Este servicio sirve para informar al usuario sobre ciertos eventos o errores en las Capas 1 y 2.
- **Identificador FMA1/2 (opcional).** Permite al usuario de la EE o EM determinar la versión de los datos del FDL local y del *hardware* y *software* FMA1/2. En el caso de emplear este servicio en la EM, el usuario adicionalmente puede solicitar el mismo tipo de información de una estación remota.
- **Estado de la LSAP (opcional).** Informa al usuario sobre la configuración de los servicios de puntos de acceso del FDL local o, si es una estación remota, sobre sus servicios FDL.
- **Lista activa (opcional).** Otorga al usuario de la EM configurar la lista de todas las estaciones que están funcionando en el Bus.
- **Activación del SAP (opcional).** Habilita al usuario para activar y configurar un enlace de servicio de punto de acceso para los individuales servicios FDL. Se excluye de este servicio la función de respuesta para servicios de replica (*SRD y CSRD*). El usuario recibe una confirmación sobre la ejecución de los servicios del FMA1/2.
- **Activación del RSAP (opcional).** Permite al usuario activar una LSAP local para la función de respuesta de los servicios de replica. El usuario recibe una confirmación de la ejecución del FMA1/2.
- **Desactivación del SAP (opcional).** Permite al usuario causar que el FMA1/2 desactive la LSAP local. El administrador regresa una confirmación de esto.

E.6 GESTIONAMIENTO (FMA1/2)

La gestión para las Capas 1 y 2 (*FMA1/2*) administra la inicialización, la supervisión y el manejo de los errores entre el usuario FMA1/2 y las funciones lógicas en PHY y FDL. La funciones de FMA1/2 para las Capas 1 y 2 son las siguientes:

- Restablecer las capas 1 y 2 (local).
- Leer y configurar parámetros (local).
- Activar, configurar y desactivar las LSAP's (local).
- Eventos y mensajes de error (local).
- Determinar la configuración LSAP (local).
- Identificar la versión (local y remota).
- Solicitar la configuración LSAP (remota).
- Determinar la lista activa (remota).

El FMA1/2 es un mediador entre el usuario local FMA1/2 y las capas 1 y 2. Las solicitudes del usuario, para modificar como para requerir, se transfieren al control FDL y PHY respectivamente por el FMA1/2. Estas se reconocen con una confirmación al usuario local FMA1/2. Además el FMA1/2 inmediatamente recibe indicaciones de las capas si ocurre un cambio de estado dentro de ellas.

E.6.1 Interfase entre FDL y FMA1/2

El FDL provee los siguientes servicios a FMA1/2:

- **Restablecimiento del FDL (obligatorio).** El FMA1/2 usa este servicio para restaurar el FDL. Después de la ejecución el FMA1/2 recibe una confirmación.
- **Configuración del valor FDL (obligatorio).** Permite al FMA1/2 configurar valores a desear del FDL. Una confirmación informa si el valor configurado es valido.

- **Lectura de valores FDL (opcional).** Permite habilitar la lectura de parámetros FDL. El valor presente se transfiere al FMA1/2 en la respuesta del FDL.
- **Errores FDL (obligatorio).** El FDL usa este servicio para informar al FMA1/2 un evento o condiciones de error.

E.6.2 Interfase entre PHY y FMA1/2

La capa PHY ofrece los siguientes servicios a FMA1/2:

- **Restablecimiento de la capa PHY (obligatorio).** Permite al FMA1/2 restaurar directamente la capa PHY. La gestión recibe una confirmación.
- **Configuración de valores PHY (opcional).** Habilita la configuración de ciertas variables en la Capa PHY. PHY da una confirmación a FMA1/2 cuando el valor es valido.
- **Lectura de valores PHY (opcional).** Con este servicio la capa FMA1/2 habilita la lectura de ciertas variables a PHY. En la respuesta el presente valor de la variable seleccionada es transferido a el FMA1/2.
- **Eventos PHY (opcional).** Este servicio informa al FMA1/2 sobre los cambios en el valor de ciertas variables.

RESUMEN AUTOBIOGRÁFICO

Marco Antonio Escobar Vera nació en México D.F. en 1972, sus padres el Sr. Eduardo Escobar Nakamura y la Sra. Guillermina Vera Hernández. En Enero de 1995 recibió el título de Ingeniero en Electrónica y Comunicaciones en la Facultad de Ingeniería Mecánica y Eléctrica de la U.A.N.L donde también obtuvo un reconocimiento al mérito Academico. Entre 1993 y 1995 trabajó en Sistemas Especializados en Teleproceso y Eléctronica S.A. como Auxiliar de Soporte a equipo de Telecomunicaciones. En Febrero de 1995 inició sus estudios de Postgrado en la misma Facultad. La tesis desarrollada para obtener el grado de Maestro en Ciencias de la Ingeniería en Telecomunicaciones fue "Diseño de una Red de Comunicaciones para la Monitorización de Disturbios Eléctricos en Sistemas de Potencia". En mayo de 1998 ingresó al Departamento de Diseño de AXA YAZAKI Planta 1 en Monterrey donde actualmente es Supervisor del area de CAE (Computing Adding Engineering).