UNIVERSIDAD DEL VALLE DE
GUATEMALA

Facultad de Ciencias y Humanidades

Departamento de Ingeniería Electrónica

Transmisión y recepción de datos html en bandas FM de radiodifusión.

LAURA CARLOTA GONZÁLEZ CAJAS

Guatemala 2003

Transmisión y recepción de datos html en bandas FM de radiodifusión.

Vo. Bo.:

Asesor:

(f)_____
Ing. Gonzalo Palaréa Murga

Terna examinadora:

(f)_____
Dr. Ing. Manuel Antonio López Valdéz

(f)_____
Ing. Julio Vásquez

(f)_____
Ing. Gonzalo Palaréa Murga

Fecha de aprobación: 12 de enero 2004

UNIVERSIDAD DEL VALLE DE
GUATEMALA

Facultad de Ciencias y Humanidades

Departamento de Ingeniería Electrónica

Transmisión y recepción de datos html en bandas FM de radiodifusión.

LAURA CARLOTA GONZÁLEZ CAJAS

Trabajo de Graduación presentado para optar al grado académico de
Licenciado en Ingeniería Electrónica.

Guatemala 2003

PREFACIO

Este trabajo tiene como objetivo principal la creación de una aplicación en la cual sea posible la transmisión de datos html (HyperText Markup Language) mediante el uso de las frecuencias de radiodifusión para utilizar con mayor provecho el ancho de banda que poseen las estaciones de radio.

La idea surgió a partir de la inquietud de saber si era posible transmitir por radiofrecuencia algún tipo de información en formato html para poder tener internet de una sola vía accesible a miles de usuarios a la vez.

En el trabajo que se presenta a continuación se encuentra una solución para este medio de transmisión y se detallan los distintos pasos que se llevaron a cabo para la creación de la aplicación.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

I. INTRODUCCIÓN

Hoy en día la importancia de mantenerse informados no importando el lugar o los medios que se posean es una de las preocupaciones principales de la sociedad. La mayoría de aplicaciones desarrolladas implican algún grado de transmisión de datos, ya sea por internet, teléfonos celulares u otros, que ayudan a un individuo a mantenerse en el mayor contacto posible con el medio exterior.

Estas aplicaciones llevan siempre en su diseño el aprovechamiento máximo de los recursos que se posean, tratando de que los costos sean los más bajos posibles para hacerlos rentables.

Entre estos recursos se encuentran las frecuencias VHF (30-300MHz) que son utilizadas para transmisión de datos, pero existe un cuadrante en el espectro radioeléctrico que va de la frecuencia 88.1MHz hasta la frecuencia 107.5MHz que corresponde a la radiodifusión en frecuencia modulada (FM).

La radiodifusión en frecuencia modulada desde su invención (en los años treinta) ha tenido un papel muy importante en la difusión de información, teniendo como ventaja la poca infraestructura que esta necesita, el área que puede cubrir con una sola antena y que todos los que posean un receptor pueden tener acceso a ella. Aún a pesar de toda la tecnología digital con la que hoy se cuenta, la radio sigue funcionando y sigue siendo un medio de comunicación importante en todo el mundo.

En los años ochenta, surgió una tecnología que ofrecía maximizar el uso del espectro con la posibilidad de agregar datos a las transmisiones analógicas en FM en la banda que no se utilizaba por encima del espectro de la señal estereofónica.

Esta tecnología es conocida en Europa como Radio Data System (RDS, sistema de datos radiofónicos). Aquí, una subportadora a 57kHz se modula digitalmente con las señales de datos una vez codificadas. La información que proporcionan estos datos se recogen en receptores especialmente diseñados para RDS con la capacidad de demodular y decodificar la información mostrándola en una pantalla de texto de pocos caracteres. En la Fig. 1 se puede observar el espectro de frecuencia de una estación de radio que transmite señal de datos RDS.

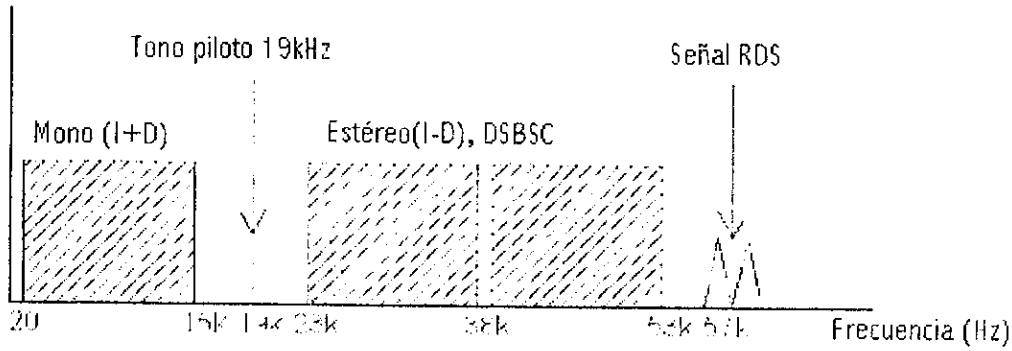


Fig. 1 Espectro frecuencial de una estación de radio

Un ejemplo de la pantalla que se encuentra en los radios con RDS se muestra a continuación en la Fig. 2. En este caso se muestra la identificación de la estación (RNE = Radio Nacional Española, CLAS = Clásica).

Fig. 2 Pantalla típica de un radio con RDS

Los datos se transmiten a una velocidad de 1,187.5 bits por segundo, por lo que este servicio se utiliza actualmente para la identificación de la estación, e informar acerca del contenido de los programas, música y para ayuda en la sintonía automática.

Esta tecnología también se adoptó en los Estados Unidos pero bajo el nombre de Radio Broadcast Data System (RBDS). Ambos sistemas se ajustan a las mismas normas y especificaciones. Difieren prácticamente en los identificadores de los tipos de programas existentes.

Este trabajo pretende crear una nueva aplicación en RDS que consista en la transmisión de paginas html por la subportadora de 57*kHz* de FM. Por ser una transmisión de baja velocidad, la aplicación está diseñada para páginas html cuyo contenido sea mayormente texto y por ser de una vía estará enfocado especialmente para noticias, mensajes de emergencia y avisos.

El trabajo conlleva varias partes que son:

- La realización del protocolo de comunicación entre el transmisor y el receptor, lo que implica agregar a los datos un medio de detección y corrección de errores, así como la división de la información en bloques y paquetes.

- Los programas de software del transmisor y receptor para la interacción con el usuario y con el módulo físico.
- La parte física del transmisor que es la encargada de codificar los datos haciéndolos aptos para el medio de transmisión.
- Y la parte física del receptor que es la encargada de decodificar los datos.

La meta final es tener dos módulos: el primero, un transmisor que tenga como señal de salida, la suma de la señal de audio con la señal de datos, la cual se conectaría al modulador de FM y que se conecte por medio serial (RS232) a la computadora de la cual proviene la información que se codificará pudiendo llevar un control de la información que se desea enviar, el número de repeticiones y la identificación de la estación.

El segundo, un módulo de recepción que sea capaz de sintonizar automáticamente una estación, establecer si posee canal de datos y guardar los datos en el disco duro para poder ser vistos en un explorador por el usuario.

II. OBJETIVOS

A. Generales

- Ampliar la utilización del ancho de banda concedido a las estaciones de radio FM.
- Facilitar el acceso a la información, a través de la radiodifusión.

B. Particulares

- Desarrollar un protocolo de comunicación de datos de una via.
- Construir un generador de RDS
- Construir un receptor de FM que contenga un decodificador RDS.
- Desarrollar el software específico que se encargue de la transmisión, así como el software que se encargue de la recepción de los datos.

III. ANTECEDENTES

A. RDS y RBDS

En los años ochenta, surgió una tecnología que ofrecía maximizar el uso del espectro con la posibilidad de agregar datos a las transmisiones analógicas en FM en la banda que no se utilizaba por encima del espectro de la señal estereofónica.

Esta tecnología es conocida en Europa como Radio Data System (RDS, sistema de datos radiofónicos). Aquí una subportadora a $57kHz$ se modula digitalmente con las señales de datos una vez codificadas. La información que proporcionan estos datos se recogen en receptores especialmente diseñados para RDS con la capacidad de demodular y decodificar la información mostrándola en una pantalla de texto de pocos caracteres.

Esta tecnología también se adoptó en los Estados Unidos pero bajo el nombre de Radio Broadcast Data System (RBDS, sistema de radiodifusión de datos). Ambos sistemas se ajustan a las mismas normas y especificaciones del estándar europeo RDS, pero el RBDS tiene además especificaciones de códigos de área e identificadores de programas para Norteamérica y especificaciones para el servicio de localizadores.

B. Historia de la tecnología RDS

La tecnología RDS empezó a desarrollarse en 1974 como una manera de identificar las estaciones de radio y sus respectivos programas. La primera prueba de campo se realizó en Suiza en el año de 1980. Se realizaron más pruebas en Estocolmo en el año de 1982 y en 1983 la tecnología fue adoptada por la European Broadcasting Union (EBU).

La primera presentación de RDS en los Estados Unidos fue en la ciudad de Detroit en el año de 1984. Aquí la compañía Ford empezó el desarrollo de sistemas de radio con RDS para automóviles.

Para el año de 1987, Irlanda, Francia y Suecia introdujeron los primeros receptores de RDS, y Volvo comercializó el primer radio para automóvil con esta tecnología.

En 1988, Austria, Bélgica, Dinamarca, Alemania, Italia e Inglaterra con compañías como Phillips empezaron la producción masiva de radios para automóvil con receptores RDS.

La primera presentación de RDS en la radiodifusión de Asia, Singapur y Sudáfrica se realizó en 1990 y un año más tarde en China.

En 1992 se completó el estándar de RBDS en los Estados Unidos y se publicó el estándar del Comité Europeo de Normalización de Electrotecnia (CENELEC) en Europa. Al año siguiente se forma el foro para el sistema RDS para normar el crecimiento de las aplicaciones .

Las primeras aplicaciones adicionales aparecieron en 1994 y 1995. En 1994 se comienza a utilizar RDS para aplicaciones como el Traffic Message Channel (TMC), sistema de difusión de mensajes sobre el tráfico.

C. Características

El sistema RDS es un medio de transmisión de datos serial sincronizado, diseñado para hacer traslados de información a una velocidad de 1187.5 bits por segundo, utilizando las frecuencias de radiodifusión de la banda FM.

Los datos son divididos en paquetes. Estos paquetes están compuestos de cuatro estructuras llamadas bloques (Bloque A, B, C, D), los cuales poseen 16 bits de información y 10 bits utilizados en la corrección de errores o de paridad.

BloqueA		BloqueB		BloqueC		BloqueD	
16 bits	10 bits	16 bits	10 bits	16 bits	10 bits	16 bits	10 bits

Tabla 1 Estructura bloques RDS

1. Corrección de errores

La corrección de errores que se utiliza consta de 10 bits de paridad con lo que logra las siguientes características:

- Detecta todos los errores de bit simples y dobles de cada bloque.
- Detecta cualquier error que perturbe una sucesión de 10 bits o menos.
- Detecta alrededor del 99.8% de perturbaciones de 11 bits de extensión y aproximadamente el 99.9% de todas las perturbaciones más largas.
- Corrige cualquier perturbación simple de 5 bits de extensión o menos (siempre que el receptor lo disponga).

Para generar los 10 bits de paridad se utiliza el polinomio de grado diez:
$g(x) = x^{10} + x^8 + x^7 + x^5 + x^4 + x^3 + 1$.

Los bits son transmitidos del más significativo al menos significativo.

2. Recuperación de la sincronía

Para recuperar la sincronía, el transmisor agrega a los bits de paridad un error predeterminado, único para cada bloque, lo que permite al receptor identificarlos. A este error se le conoce como desplazamiento. El desplazamiento se agrega a los bits de paridad mediante una suma binaria.

El receptor debe realizar una multiplicación entre los 26 bits recuperados y una matriz de paridad de 26x10 para recuperar un arreglo de 10 bits que se conocen como síndrome. La matriz utilizada se muestra en la Fig. 3.

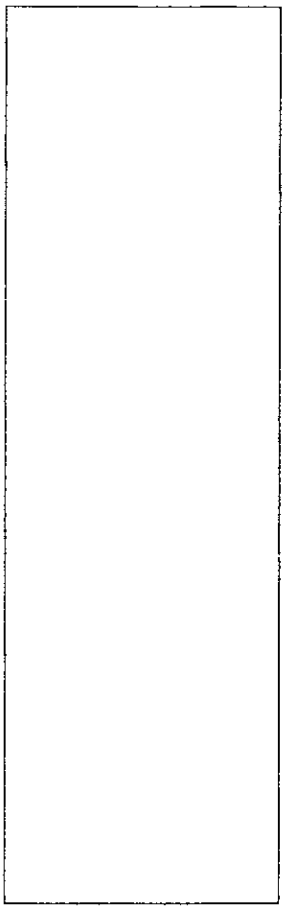


Fig. 3 Matriz de codificación

El síndrome es el que hace posible la identificación de los bloques, ya que existe un síndrome para cada desplazamiento y por ende para cada bloque. En la Tabla 2 se muestran las palabras de desplazamiento y sus respectivos síndromes para cada tipo de bloque.

Desplazamiento	**Palabra de desplazamiento**	**Síndrome**
A	0011111100	1111011000
B	0110011000	1111010100
C	0101101000	1001011100
C'	1101010000	1111001100
D	0110110100	1001011000

Tabla 2 Síndromes de Bloques utilizados en RDS

El significado de cada uno de los bloques varía según la aplicación de RDS que se esté utilizando. Por ejemplo para las aplicaciones de radio texto, el bloque A se usa para identificar el tipo de programa, el bloque B para identificar tipo de aplicación RDS, el bloque C para frecuencias alternativas de la estación de radio, el bloque D para identificar los servicios del programa que se está transmitiendo.

3. Codificación de la señal RDS

La señal de datos digitales antes de poder ser sumada con la señal de audio debe pasar un proceso de codificación diferencial, luego por un generador de símbolos bifásicos y por último por una modulación en amplitud con portadora suprimida a una frecuencia de $57kHz$.

El proceso de codificación especificado por el estándar RDS se muestra a continuación en la Fig. 4.

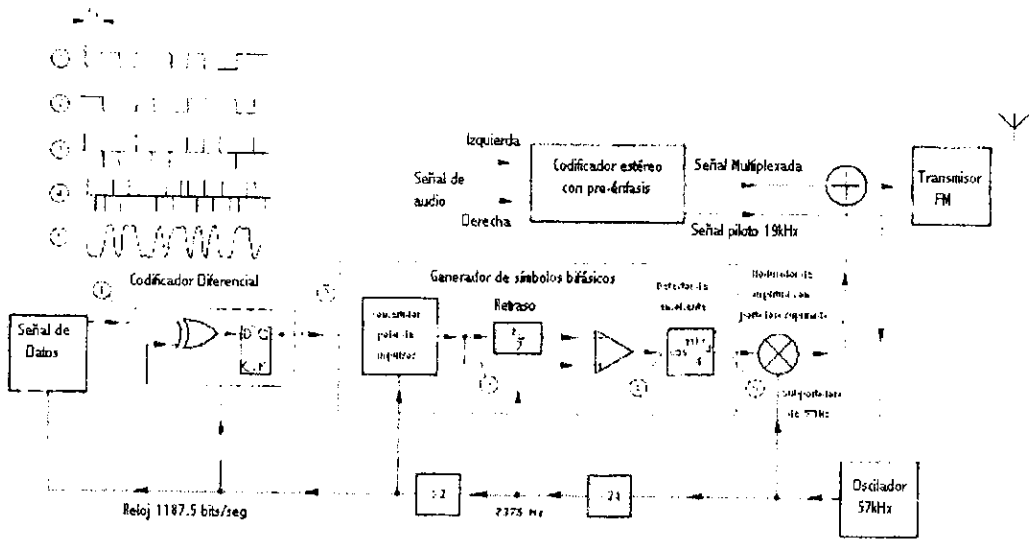


Fig. 4 Esquema de codificador RDS

El espectro de frecuencia de la señal multiplexada que resulta al añadir la señal de datos a la señal de audio antes de ser modulada en FM se muestra en la Fig. 1.

IV. DESARROLLO

El trabajo consiste en tres partes básicas:

- Desarrollo del protocolo
- Realización de los circuitos
- Desarrollo de las aplicaciones de la computadora y de los microcontroladores a utilizar.

El flujo de la información dentro del sistema se representa en la Fig. 5.

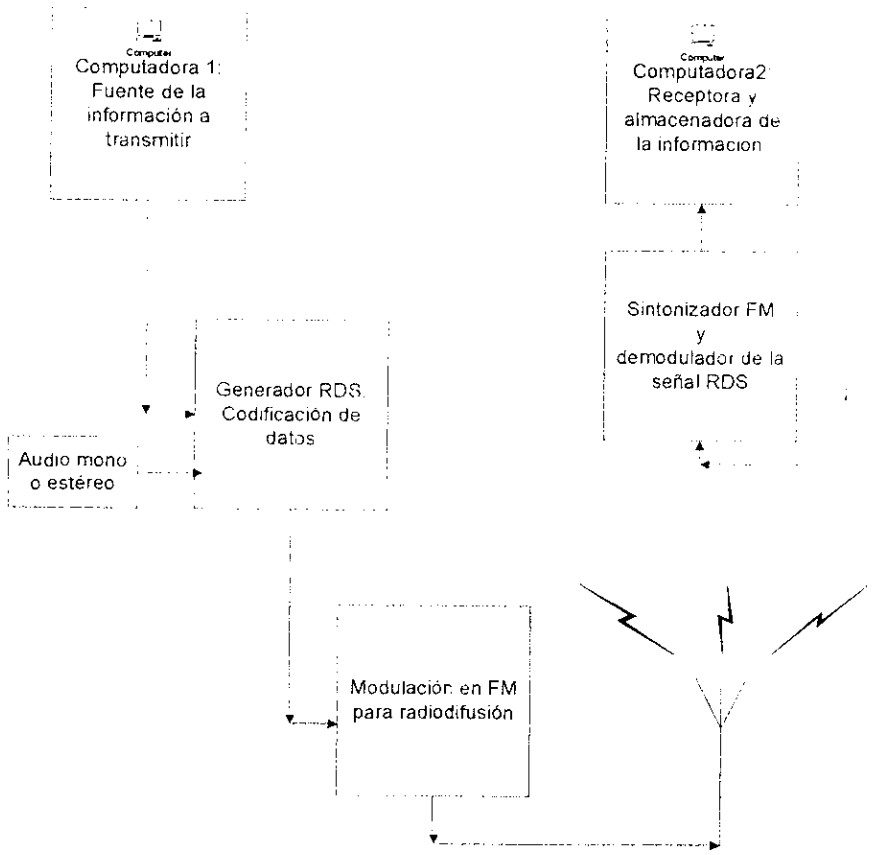


Fig. 5 Diagrama de bloques del flujo de la información

Cada una de las partes serán tratadas en detalle en los siguientes capítulos.

V. PROTOCOLO DE TRANSMISIÓN

A. Desarrollo

El protocolo para la aplicación fue ideado para transmitir archivos html basándose en el sistema RDS.

No se utilizó el mismo protocolo que se utiliza para las aplicaciones existentes de RDS debido a que este protocolo está pensado para transmisiones de información de muy poco contenido (generalmente de unos 16 *bytes*), y para transmitir datos html se necesita transmitir un número considerable de datos que recae en el rango de miles de *bytes*.

Se decidió utilizar el mismo sistema de corrección de errores (codificación cíclica) por sus propiedades de detectar y corregir errores simples (hasta un máximo de cinco errores simples en dos *bytes* de información), así como para hacer esta aplicación de cierta manera compatible con las demás aplicaciones RDS. Por ésto se tiene la misma estructura de bloques donde por cada 16 bits de información se agregan 10 bits de corrección de errores.

Para la recuperación de la sincronía se utiliza la suma de desplazamiento a los bits de corrección de errores de ciertos bloques, cada n número de *bytes* transmitidos.

B. Estructura de la unidad mas pequeña de transmisión

La unidad más pequeña de transmisión es de 26 bits, 16 de información y 10 de corrección de errores, lo cual constituye un bloque en RDS.

Bloque	
16 bits	10 bits

Tabla 3 Estructura bloque de transmisión

C. Empaquetado

En la transmisión de datos html la cantidad de *bytes* transmitidos es considerable, en el rango de miles de *bytes*, y debido a que el medio de transmisión no es 100% libre de perturbaciones es necesario dividir

esta información en paquetes, para poder recuperar paquete por paquete y poder identificar si algún paquete no se ha recibido o se ha recibido con errores no reparables.

El número de *bytes* de información transmitido por paquete se dejó variable para experimentar y poder encontrar qué cantidad es la más adecuada para el medio en que se transmite. Esto se debe a que en un medio limpio se puede transmitir mayor cantidad de información por paquete que en un medio que tenga mucho ruido o interferencia.

D. Encabezado

El encabezado es necesario para poder identificar de qué paquete se trata y si sigue siendo del archivo que se está recuperando o no. Se le asignó un tamaño de 8 *bytes* a esta información para que ocupara un total de cuatro bloques. Estos cuatro bloques llevan la siguiente información:

- Paquetes por archivo (2 *bytes*): el número total de paquetes en los fue dividido el archivo.
- Bytes por paquete (2 *bytes*): el número de bytes por paquete es el mismo para todos los paquetes excepto para el último el cual puede tener un número menor de bytes, según el tamaño del archivo.
- Número de Paquete (2 *bytes*): es el identificador del paquete y dice la posición del paquete en el archivo, al igual que los Paquetes por Archivo, posee un tamaño de dos bytes.
- Identificador del archivo (2 *bytes*): es un número característico, y sirve para diferenciar los distintos archivos html. Consiste de dos *bytes*, el primero indica el número de actualización por día, y el segundo el número que el archivo representa en el grupo de archivos que se transmitan.

E. Sincronía

Debido a que el archivo html se ha divido en paquetes y los paquetes se han dejado con un número de *bytes* variable, es necesario recuperar la sincronía para cada paquete, para esto se le agregaron desplazamientos a la información que forma el encabezado para que sean inconfundibles con el resto de la información que se transmitirá. El encabezado quedó de la siguiente manera Tabla 4.

BloqueA	BloqueB	BloqueC	BloqueD
Paquetes por Archivo	*Bytes* por paquete	Número de paquete	Identificador del archivo

Tabla 4 Bloques que conforman el encabezado de paquete

Como se puede ver cada característica quedó de 16 bits, entonces pueden representar un número entero igual a 2^{15}, este número es suficientemente grande para que se pueda transmitir un archivo que tenga 32768 paquetes lo que se considera suficiente para la aplicación que se busca llevar a cabo.

F. Información transmitida y su orden

Ya se trató en la sección anterior la información que se necesita para recuperar el archivo en el orden adecuado, número de paquetes, etc. Ahora se mostrará la información que se necesita mandar con el archivo html para poderlo reconocer, procesar y guardar del lado del receptor.

- Nombre del archivo: Es el nombre del archivo que se desea transmitir, posee una longitud máxima de diez caracteres contando su respectiva terminación y el punto.
- Estación: Nombre representativo de la estación de radio. Tiene un máximo de 11 caracteres. En el receptor se creará una carpeta con este nombre para guardar aquí los archivos provenientes de esa estación. El nombre de la estación no podrá contener caracteres especiales ya que servirá para crear la carpeta.
- País: Abreviación del país de transmisión. Posee un máximo de tres caracteres. Ej. GUA (Guatemala)
- Frecuencia: Número entero de un *byte* que representa la frecuencia de la radio que está transmitiendo la información.
- Número de archivos relacionados: Número entero de un *byte* que representa la cantidad de archivos que se están transmitiendo en un determinado intervalo de tiempo.

A continuación se muestra un resumen de los componentes que identifican el archivo y su estación.

No.	Característica	Tamaño máximo
1.	Nombre del archivo	10 caracteres
2.	Estación	11 caracteres
3.	País	3 caracteres
4.	Frecuencia	1 *byte*
5.	Número de archivos relacionados	1 *byte*

Tabla 5 Resumen características de la estación y archivo

1. Reglas de la transmisión

Considerando que el medio puede ser bastante ruidoso y que no existe control de flujo porque es una comunicación de una vía, debe considerarse (en la transmisión) un alto número de repeticiones de la información, para que el receptor sea capaz de recuperar todos los archivos que se estén transmitiendo. Se deja como una variable el número de repeticiones que se llevarán acabo.

2. Reglas de la recepción

El receptor debe encargarse de recuperar los paquetes con el menor número de errores. Para esto el programa que recibe debe considerar el error posible o existente de cada paquete y desechar los paquetes que tengan una cantidad de errores mayor o cercano al número máximo de corrección de errores de la codificación utilizada, debiendo esperar a recuperar el paquete erróneo en las repeticiones que se deben transmitir del archivo.

VI. CIRCUITOS

Para la realización del trabajo se escogieron circuitos cuyos elementos estuvieran disponibles en la mayoría de proveedores de elementos electrónicos para realizar los circuitos. Además se usaron circuitos que son de uso común para este tipo de aplicaciones.

En la parte del transmisor se usaron dos circuitos, un codificador estéreo y el codificador RDS. Esto se hizo con la idea de simular una verdadera estación de radio donde se tienen estos equipos por aparte. La salida del codificador RDS se dirige al transmisor FM. en este caso se utilizó un transmisor de $50mW$

El circuito transmisor requiere una entrada de voltaje que puede estar en el rango de $14V$, este voltaje alimenta a un regulador de $12V$ y de este voltaje principal utilizado para dispositivos CMOS (Complementary Metal Oxide Semiconductor, semiconductor complementarios de óxido metálico) se obtienen los $5V$ para los circuitos TTL (Transistor-Transistor Logic, lógica transistor-transistor) mediante un regulador de voltaje, y un voltaje intermedio, que es la mitad del principal con un divisor resistivo de voltaje.

Este voltaje intermedio se utiliza para montar la señal que entra a los amplificadores operacionales de modo que no se necesite un voltaje negativo para alimentarlos.

Para el circuito receptor se utiliza una entrada de voltaje de $14V$ y $-14V$, que se dirigen a reguladores de voltaje de $12V$ y $-12V$, para poder alimentar al convertidor digital análogo . El consumo de corriente de este circuito es de $200mA$.

El consumo de corriente de cada circuito es: para el codificador estéreo alimentado con $14V$ $21mA$ y para el codificador RDS alimentado con $14V$, $46mA$.

El circuito del regulador de voltaje utilizado para el transmisor y el divisor de voltaje se muestran a continuación, Fig.6 y Fig.7 respectivamente:

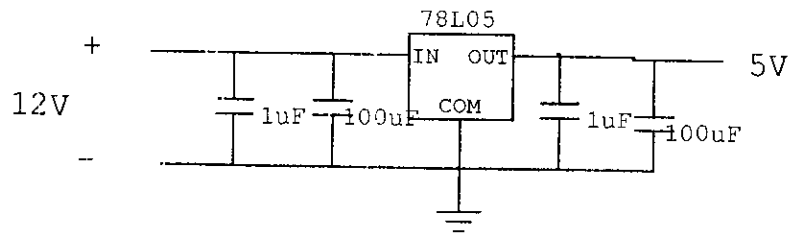


Fig. 6 Circuito regulador de voltaje

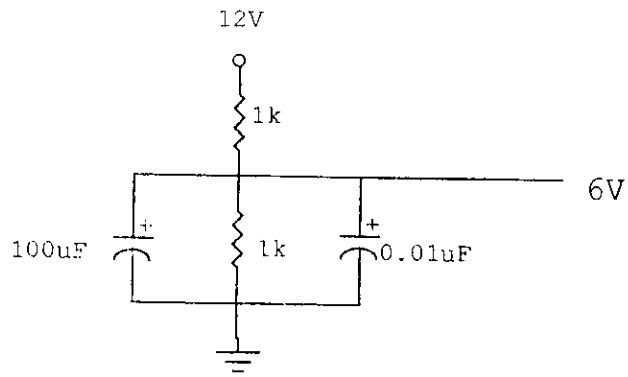


Fig. 7 Circuito divisor de voltaje

A. Transmisor

El circuito del transmisor cuenta con dos partes: el codificador estéreo y el codificador de RDS, el cual utiliza un microcontrolador como enlace y medio para la transmisión de la información de la computadora al circuito.

Debido a que la transmisión RDS es síncrona se decidió utilizar un microcontrolador que realizara la conversión de la transmisión serial asíncrona de la computadora a serial síncrona para la transmisión RDS, debido a la facilidad de manejar los bits y los interruptos de reloj en el microcontrolador, quitándole a la computadora la tarea de transmitir tan sólo un bit a la vez.

El microcontrolador que se escogió fue el PIC16F877 por tener módulo USART y por contar con los medios necesarios para la emulación del mismo.

1. Codificador estereo

El codificador estéreo tiene como entradas los dos canales de audio, derecho e izquierdo y su salida es una señal multiplexada que contiene tres señales: la suma de los dos canales, la resta de los dos canales (la cual es modulada a $38kHz$) y la señal piloto de $19kHz$, la cual indica al receptor que la transmisión es estéreo.

En la Fig.8 se muestra el diagrama de bloques del codificador estéreo utilizado.

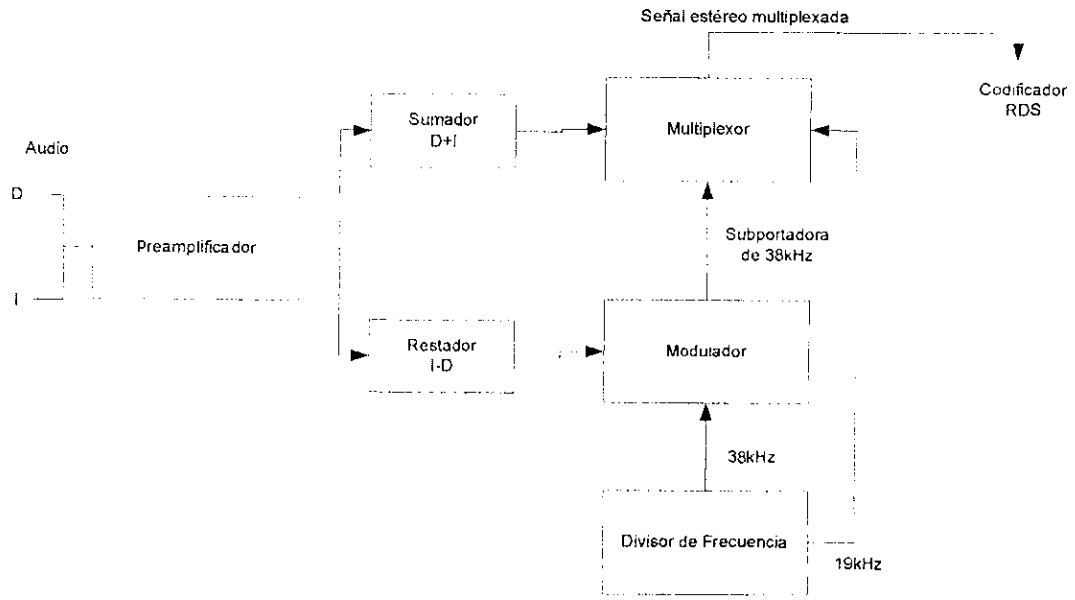


Fig. 8 Diagrama de bloques del codificador estéreo

a. Preamplificador

El preamplificador consta de elementos pasivos; su función es amplificar las frecuencias arriba de los $2kHz$ utilizando filtros pasa altos para mejorar la relación señal a ruido. Este circuito está diseñado para una entrada de audio que tenga una amplitud entre 0.5 y 1 $Vrms$. Los capacitores de $1uF$ que se encuentran a salida del circuito son capacitores de acople para adjuntar los circuitos restador y sumador.

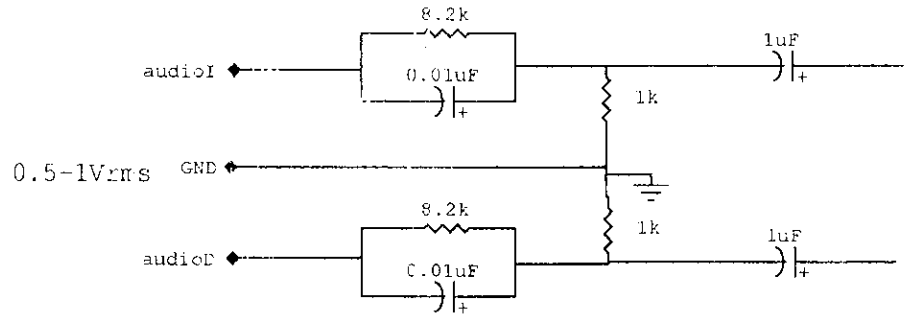


Fig. 9 Circuito preamplificador

b. Restador

Este circuito toma la señal de audio del canal izquierdo y le resta la señal de audio del canal derecho y posee un factor de amplificación de dos. Se hizo con un amplificador operacional LM1458. El capacitor que se encuentra en la salida del amplificador operacional es utilizado para acople entre este circuito y el modulador.

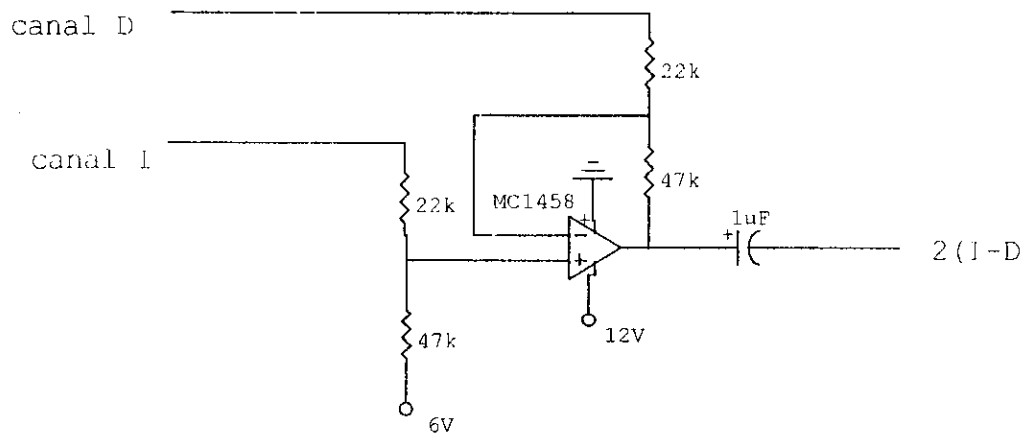


Fig. 10 Circuito restador

c. Divisor de frecuencia

Para obtener las señales de reloj de 38*kHz* (utilizada para la modulación de la resta de los canales de audio) y de 19*kHz* (la señal piloto) se utilizó un divisor de frecuencia conformado por un contador de 12 bits (CD4040). En el cual cada salida es una división por dos de la salida anterior.

En la entrada le llega la señal de reloj de 4.864*MHz* y tras dividir esta señal por un factor de 128 (2^7) se obtiene el reloj de 38*kHz* y éste dividido dos da el reloj de 19*kHz*.

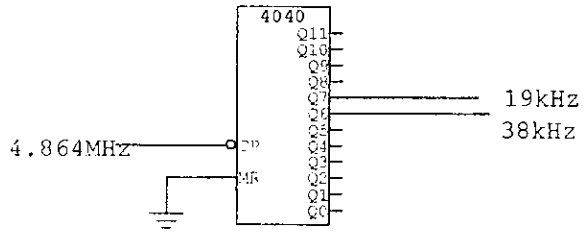


Fig. 11 Divisor de frecuencia

d. Modulador

Para la modulación de amplitud con portadora suprimida se utilizó un LM1496 en su configuración básica. Es el encargado de modular la señal de la resta de los canales de audio a una frecuencia de 38*kHz*. El potenciómetro que se dejó en el circuito sirve para balancear las corrientes de modo que se pueda suprimir la portadora lo mejor posible.

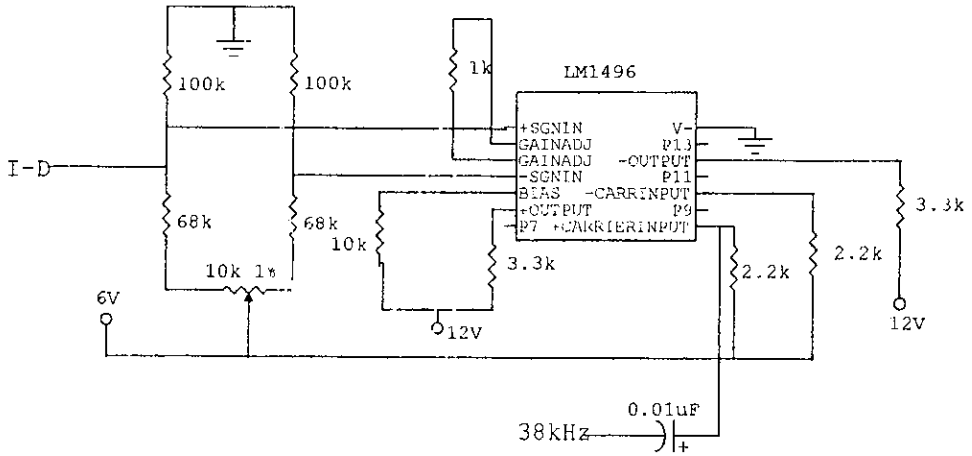


Fig. 12 Modulador de la señal restada de audio

e. Multiplexor

El multiplexor es el encargado de sumar todas las señales: la suma, la resta de los canales de audio, y la señal piloto. Se utilizó un amplificador operacional LM1458 con un circuito básico sumador, pero montado en el voltaje central que es de 6V.

Las ganancias para cada señal son distintas ya que la señal piloto debe ser mucho más débil que las del audio. Se dejó una resistencia variable en la entrada de la subportadora (resta de los canales modulada a 38*kHz*) para poder calibrar los canales de audio de forma que sus amplitudes sean iguales.

La suma de los canales derecho e izquierdo se hizo en este mismo circuito. Por eso aparece la señal del canal derecho e izquierdo separadas.

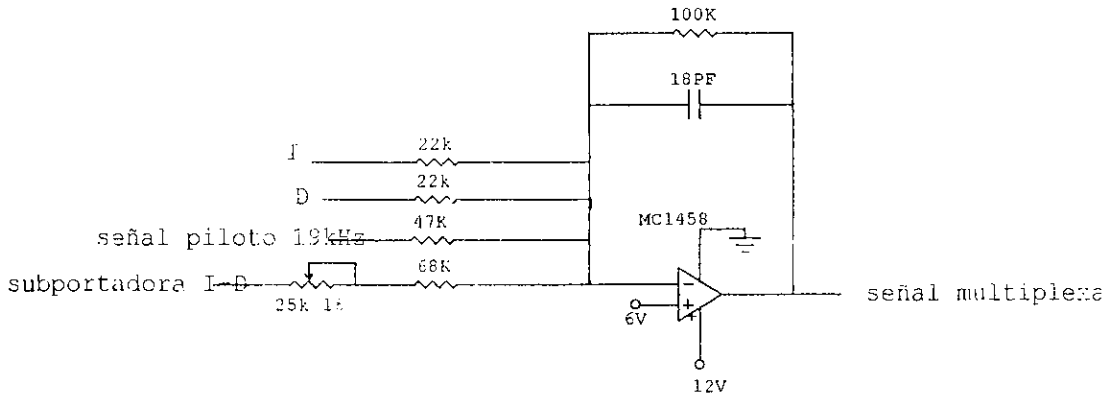


Fig. 13 Sumador de señales

2. Codificador RDS

El circuito del codificador RDS se obtuvo de Radio Pulsar, la cual pone a la disposición del público información para radio aficionados. El circuito consta de *flip-flops* o compuertas biestables , compuertas, y un amplificador operacional.

El objetivo del circuito es tomar la señal de datos (en este caso la señal proporcionada por el microcontrolador) codificarla y unirla a la señal multiplexada de audio.

Para la generación del reloj de 1,187.5 bits/seg se utilizó el TDA7330 el cual también posee un filtro pasa bandas de 57*kHz* para que la señal RDS no interfiera con la señal de audio.

La codificación se hizo según la especificación de los estándares RDS: la señal de datos debe tener una codificación diferencial y estar retrasada en 4*μseg* de la señal de reloj.

Para realizar la codificación diferencial se utilizó un *flip-flop* tipo D, y para obtener el retraso de la señal de reloj se tiene un filtro pasabajos que consiste de una resistencia y un capacitor. Después de este

filtro la señal es pasada por una compuerta para recuperar la forma de la señal de reloj. La codificación bifásica utiliza otro *flip-flop* tipo D más unas compuertas xor.

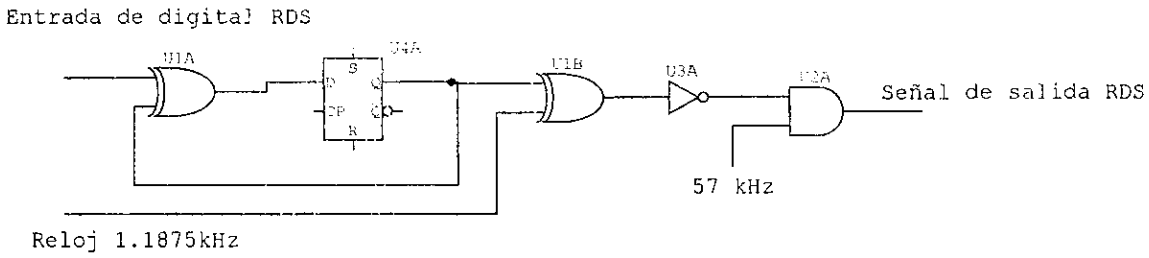


Fig. 14 (a) Esquema básico codificador RDS

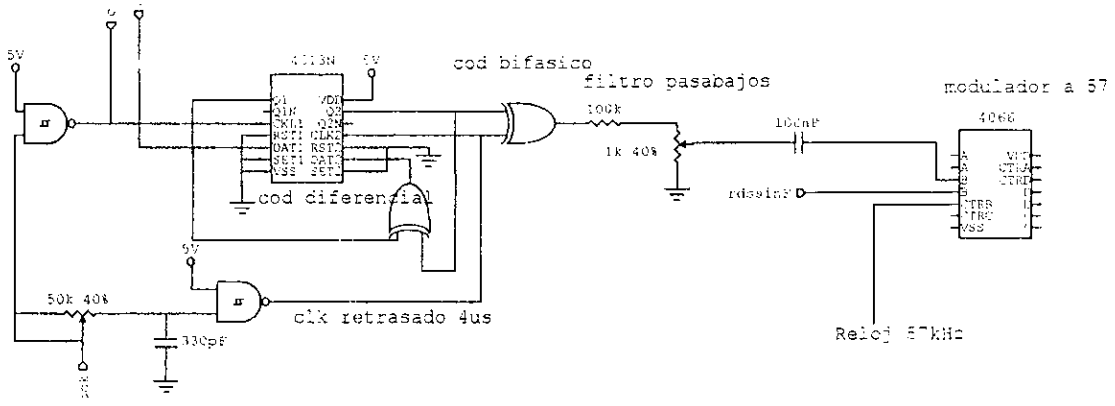


Fig. 14 (b) Codificador RDS

La modulación de la señal RDS a 57*kHz* se hizo mediante la técnica de muestreo usando como muestreador un interruptor 4066 con la señal de control igual al reloj de 57*kHz* que se obtiene de la pata T57 del integrado TDA7330. Para terminar la modulación la señal de la salida del interruptor se pasa por un filtro pasabajos.

El filtro pasabajos está compuesto por una resistencia variable y un capacitor para poder obtener el nivel correcto de filtraje.

a. Generador del reloj y filtro señal RDS

Para la generación de los relojes (*57kHz* y de 1187.5 bits/seg) y filtraje de la señal se utilizó el decodificador RDS TDA7330. Este integrado posee un circuito básico externo para la recuperación de la señal RDS muy sencillo compuesto de resistencias y capacitores.

Se escogió el TDA7330 sobre otros codificadores RDS, ya que posee un filtro pasabandas de octavo orden para la frecuencia de *57kHz* y la salida de este filtro está disponible en una pata del integrado, como se puede ver en el diagrama de bloque que se muestra en la Fig.15. Es muy importante aplicar el filtro a la señal RDS antes de multiplexarlo con la señal de audio para que la señal RDS no se salga del rango de frecuencias que tiene destinado y no cause interferencia a la señal de audio.

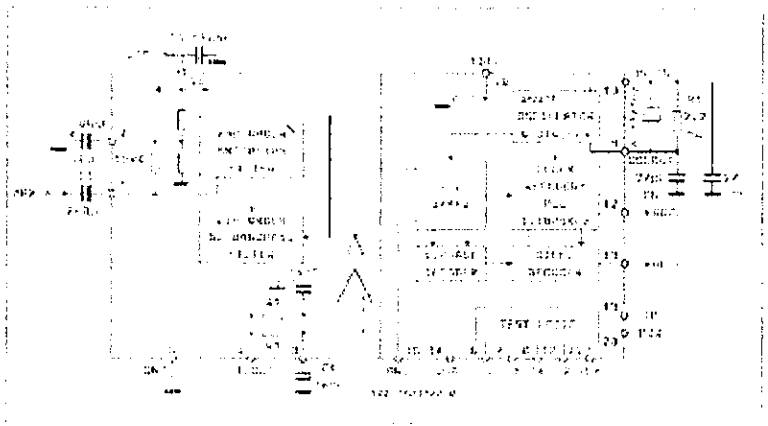


Fig. 15 Diagrama interno TDA 7330

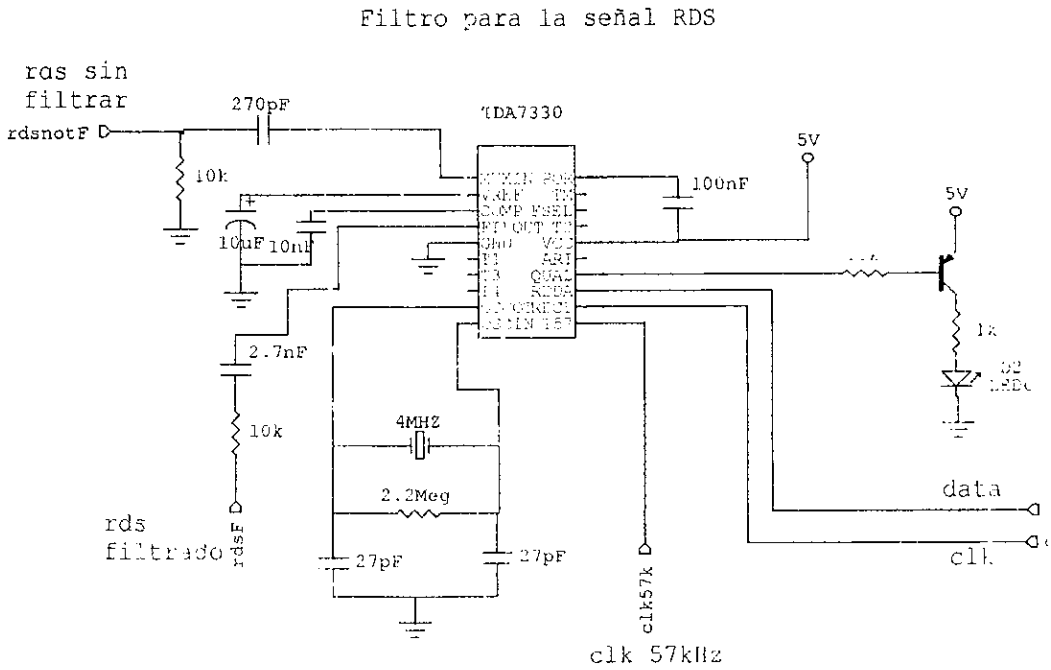


Fig. 16 Circuito externo del TDA 7330

b. Multiplexor de la señal de audio con la señal RDS

Para multiplexar la señal se usó un amplificador operacional pero antes de sumarla con la señal de audio se pasa por otro amplificador operacional para poder montar la señal y poderla sumar con la señal de audio.

Los amplificadores operacionales están alimentados con los voltajes 0V y 5V, por lo que se utiliza un divisor de voltaje para montar la señal en 2.5V. Este divisor de voltaje se encuentra a la izquierda del circuito y está compuesto por dos resistencias de $10k\Omega$.

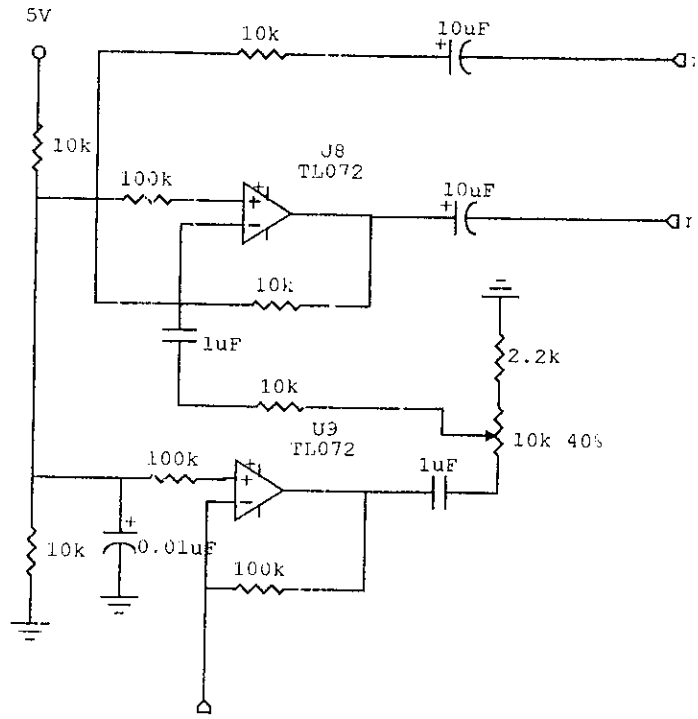


Fig. 17 Circuito sumador de señal de audio con señal de datos RDS

B. Receptor

Para el circuito receptor se tienen cuatro partes: sintonizador, demodulador FM, decodificador RDS, decodificador de audio y el circuito donde se encuentra el microcontrolador. En este circuito se tienen distintos voltajes de alimentación debido a los elementos utilizados. Pero la entrada principal de voltaje es de 12V y –12V Los demás voltajes se obtienen de reguladores de voltaje.

Como ente mediador entre la computadora y el circuito se escogió utilizar un microcontrolador PIC16F877 debido a su módulo USART y el número de puertos que tiene a disposición, ya que de aquí se obtiene el voltaje expresado en forma digital para alimentar al sintonizador, utilizando un puerto completo de 8 bits los cuales se pasan por el convertidor digital análogo.

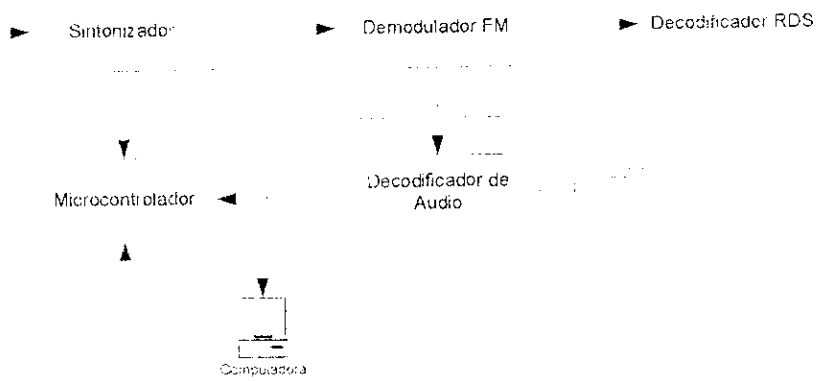


Fig. 18 Diagrama de bloques del circuito receptor

1. Sintonizador

El sintonizador que se utilizo es un circuito tanque analógico de voltaje, lo que quiere decir que la frecuencia sintonizada es proporcional al voltaje que se le aplique. El voltaje aplicado para la sintonización proviene de las salidas digitales del microcontrolador pasado por un convertidor digital análogo.

El voltaje de sintonización se encuentra entre 2V y 9V.

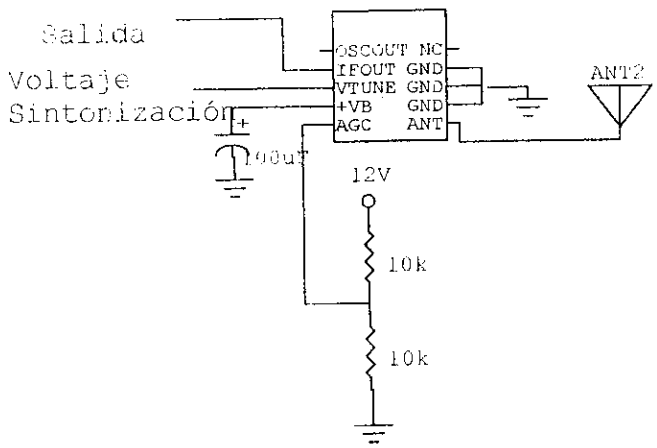


Fig. 19 Circuito del sintonizador

2. Demodulador FM

Para demodular la señal recuperada del sintonizador se utilizó un TBA120U el cual es un amplificador y demodulador de sonido.

La señal RDS se encuentra en 57*kHz* por lo que el demodular debía tener la cualidad de no filtrar las frecuencias arriba de los 38*kHz*, es por eso que se escogió este circuito integrado.

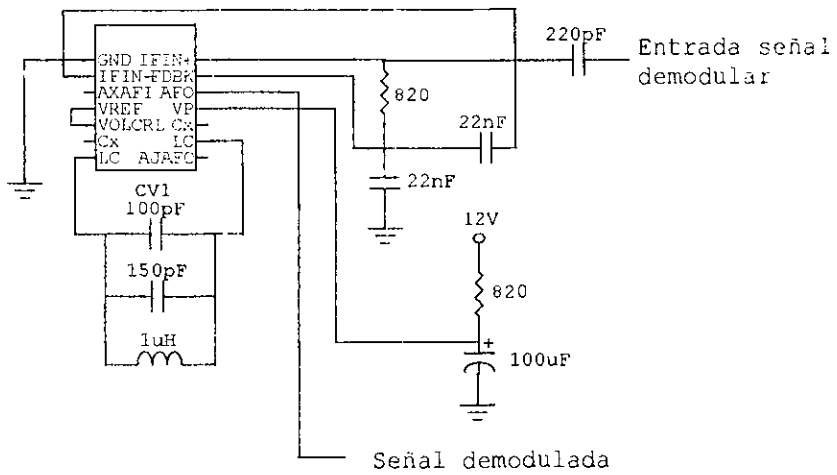


Fig. 20 Circuito demodulador de FM

3. Decodificar RDS

Para decodificar la información en RDS se utilizó el circuito integrado TDA7330. Este integrado requiere de pocos elementos externos y tiene la ventaja de que realiza todo el procedimiento de demodulación y decodificación de la señal RDS, teniendo como salidas los datos recuperados y el reloj.

El TDA7330 también tiene dos salidas muy útiles, QUAL y ARI las cuales representan la calidad de señal RDS y la existencia de señal RDS respectivamente. Estas dos salidas son usadas por la aplicación para determinar la existencia de una emisora con señal RDS.

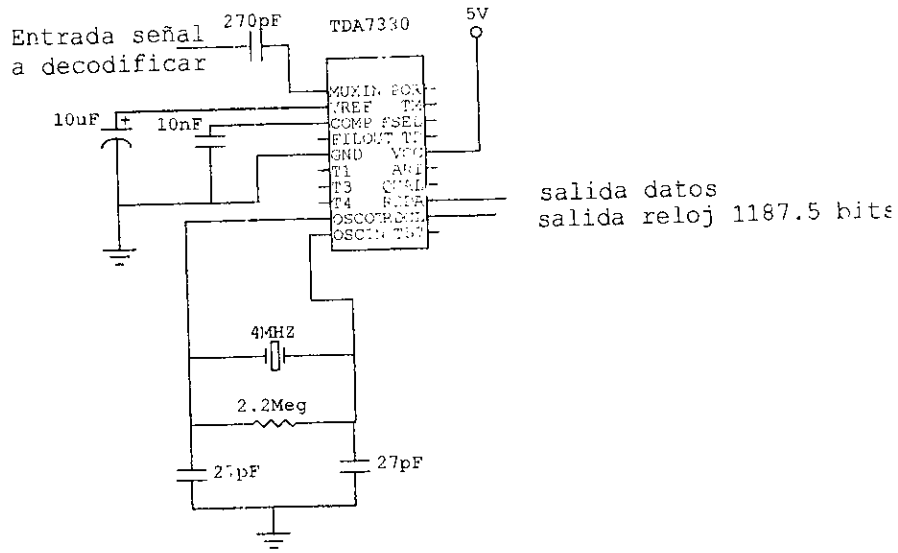


Fig. 21 Circuito TDA 7330 receptor

4. Microcontrolador

Se utilizó el microcontrolador PIC16F877. Este microcontrolador es el encargado de mandar el voltaje para el sintonizador, así como de recuperar la sincronía y traspasar la información de la existencia de la señal RDS al programa de aplicación.

Se utilizó el interrupto del pin RB0 para detectar el cambio de flanco en el reloj que se recupera con el TDA7330 y se utilizó el pin RB3 para la señal de datos .

Para elevar los voltajes de nivel TTL a los voltajes utilizados por la computadora en el puerto serial se utilizó un MAX232.

También se utilizó un convertidor digital análogo para convertir la salida digital de uno de los puertos del microcontrolador a niveles de voltaje analógico para controlar la frecuencia a la cual se está sintonizando. A continuación se muestra el circuito del microcontrador con el DAC0808 y el acople entre el USART del PIC y el RS232 de la computadora mediante el RS232.

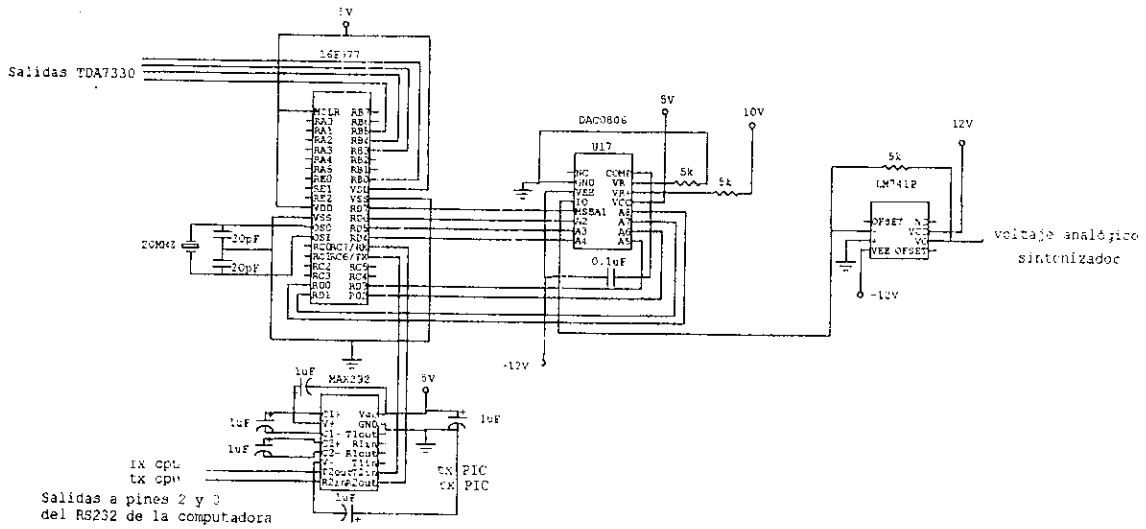


Fig. 22 Circuito microcontrolador y convertidor digital análogo

5. Decodificador de audio

Se utilizó un decodificador de audio monoestéreo, constituido por el TBA120U y un amplificador de audio LM386. Este decodificador de audio se construyó como prueba de la recuperación de la señal de audio y para que el usuario del programa pueda sintonizar las estaciones de las cuales quiere recibir los datos html.

C. Circuitos impresos

Los módulos que se describieron anteriormente fueron pasados a circuitos impresos para reducir el ruido del *protoboard*. Esto se hizo con ayuda del programa Trax Maker con el cual se realizaron las rutas de los caminos de cobre. Luego se utilizó el Isopro como interfaz con la fresadora y para realizar la prueba del aislamiento.

Se utilizaron placas de cobre de doble lado para permitir caminos en la parte inferior y superior ya que debido al gran número de rutas fue imposible realizarlo sólo en una cara. En los apendices podra encontrar la versión final de las tarjetas.

VII. PROGRAMACIÓN

En lo que respecta a la programación de la aplicación, ésta se divide en cuatro partes: el programa transmisor de la computadora, el programa transmisor del microcontrolador, el programa receptor de la computadora y el programa receptor del microcontrolador.

Para los programas de la computadora se utilizó el lenguaje de programación C++ Builder 5.0, debido a su flexibilidad y la facilidad de programar gráficamente pudiendo hacer la interfase con el usuario más amigable y presentable. Otra razón es que el lenguaje C es un lenguaje bastante conocido dejando abierta la modificación del código por algún interesado en el tema.

Los programas de los microcontroladores se desarrollaron en la aplicación MPLAB IDE, ya que este programa posee la facilidad de hacer simulaciones en un medio amigable para la programación en lenguaje ensamblador.

La interfaz escogida para la comunicación entre la computadora y el microcontrolador es serial RS232 con una velocidad de 9600 bps sin control de flujo de datos. Se escogió este tipo de comunicación ya que es una conexión común en cualquier computadora, posee sólo tres hilos y es soportada por una amplia gama de microcontroladores.

A continuación se describirán las características de cada programa, mostrando un diagrama de flujo del mismo. El código fuente de cada programa se puede encontrar en la sección de apéndices.

A. Programa transmisor de la computadora

El programa transmisor de la computadora tiene como objetivos:

- Codificar la página html
- Transmitir la información al microcontrolador mediante RS232, cuando el microcontrolador se la pida.
- Crear un ambiente amigable para el usuario de la aplicación
- Brindar al usuario información acerca de la transmisión de la página html que se va a transmitir, como por ejemplo el tiempo de transmisión, el número de paquete y el progreso de la transmisión.

El programa utiliza un proyecto con dos unidades, una donde se encuentran todas las funciones propias de la aplicación y otra donde se encuentra todo lo relacionado con la manipulación del puerto serial (abrir y cerrar el puerto serial, funciones para mandar y recibir datos, y la configuración propia del puerto como la velocidad).

Posee cuatro funciones principales que son:

- FEncoder: Es la función encargada de generar los 10 bits para corrección de errores. Consiste en multiplicar los 16 bits de data por la matriz de 16x10 para obtener los 10 bits de paridad.
- FPaquetes: Es la función que se encarga de crear el archivo codificado con su respectivos paquetes y sus encabezados
- FTransmit: Es la función que se encarga de transmitir los datos al microcontrolador cada vez que éste se lo pida, manteniendo la cuenta de lo que ya se ha transmitido y de lo que falta por transmitir.
- FEnviar: Es la encargada de iniciar las tres etapas de la transmisión que son: el inicio, que requiere abrir el puerto serial, inicializar variables, mandar comando de inicio de transmisión al PIC; la transmisión de archivos, que requiere el adquirir el archivo codificado e inicializar variables; y la finalización, que requiere el cierre del puerto serial, mandar comando al PIC de finalización de transmisión.

A continuación se muestra el diagrama de flujo principal del funcionamiento del programa de transmisión en la Fig.22. En este se puede ver los distintos eventos que el programa posee.

Como se puede ver en esta figura, la transmisión de datos al microcontrolador se realiza mediante un temporizador el cual está revisando constantemente si necesita o no mandar datos por el puerto serial. Desde la función del temporizador se hace el llamado a las funciones necesarias para transmitir esta información.

La función que codifica el archivo realiza la lectura del archivo a codificar y reúne la información que el usuario ha ingresado. Realiza un ciclo en el cual va codificando cada dos bytes (16 bits) para obtener los bits de paridad y los va guardando en el nuevo archivo que se ha creado para el archivo codificado, el cual se llama Cod.txt.

El diagrama de bloques se puede ver en la Fig.23

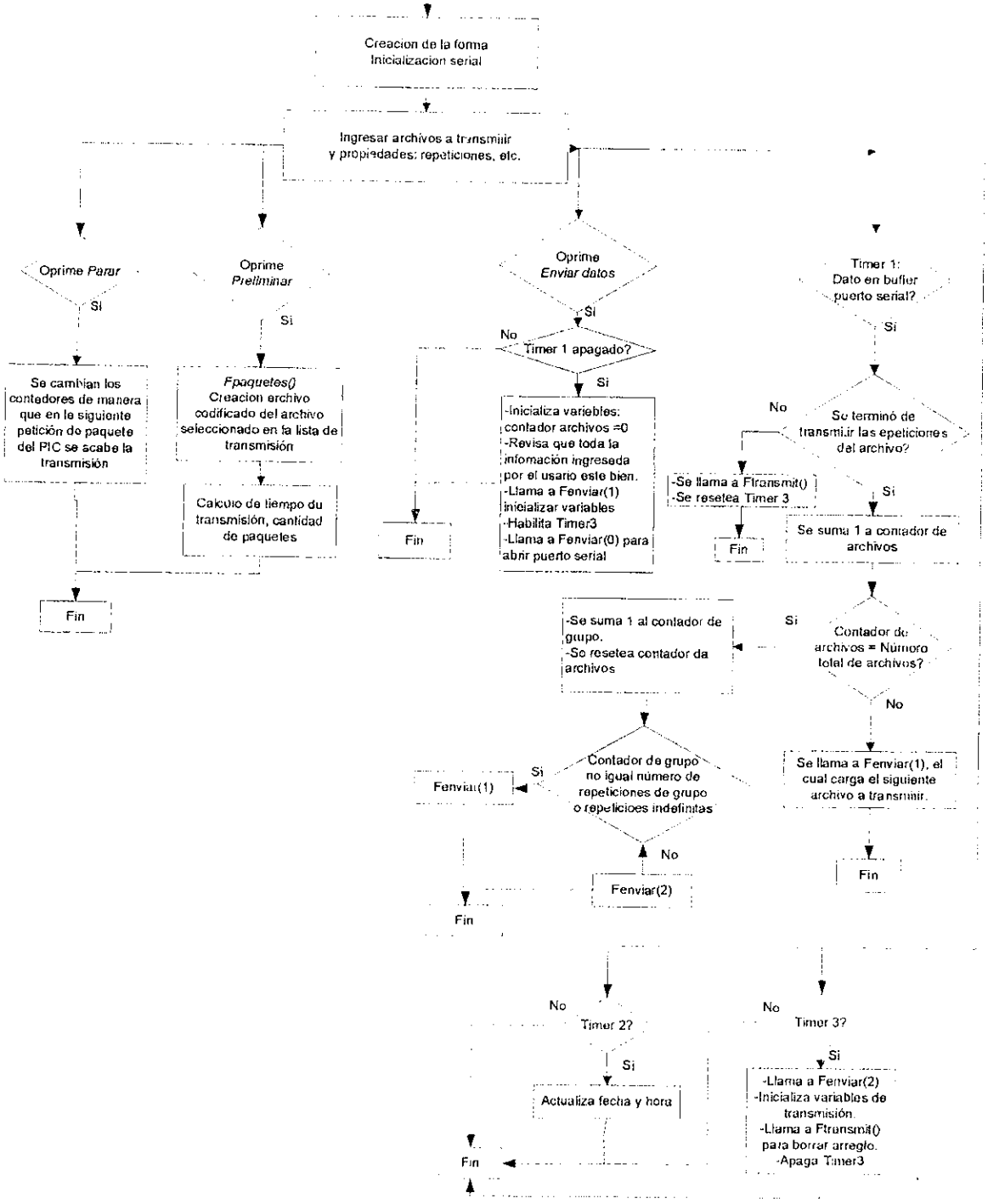


Fig. 22 Diagrama de flujo del programa de la aplicación del transmisor

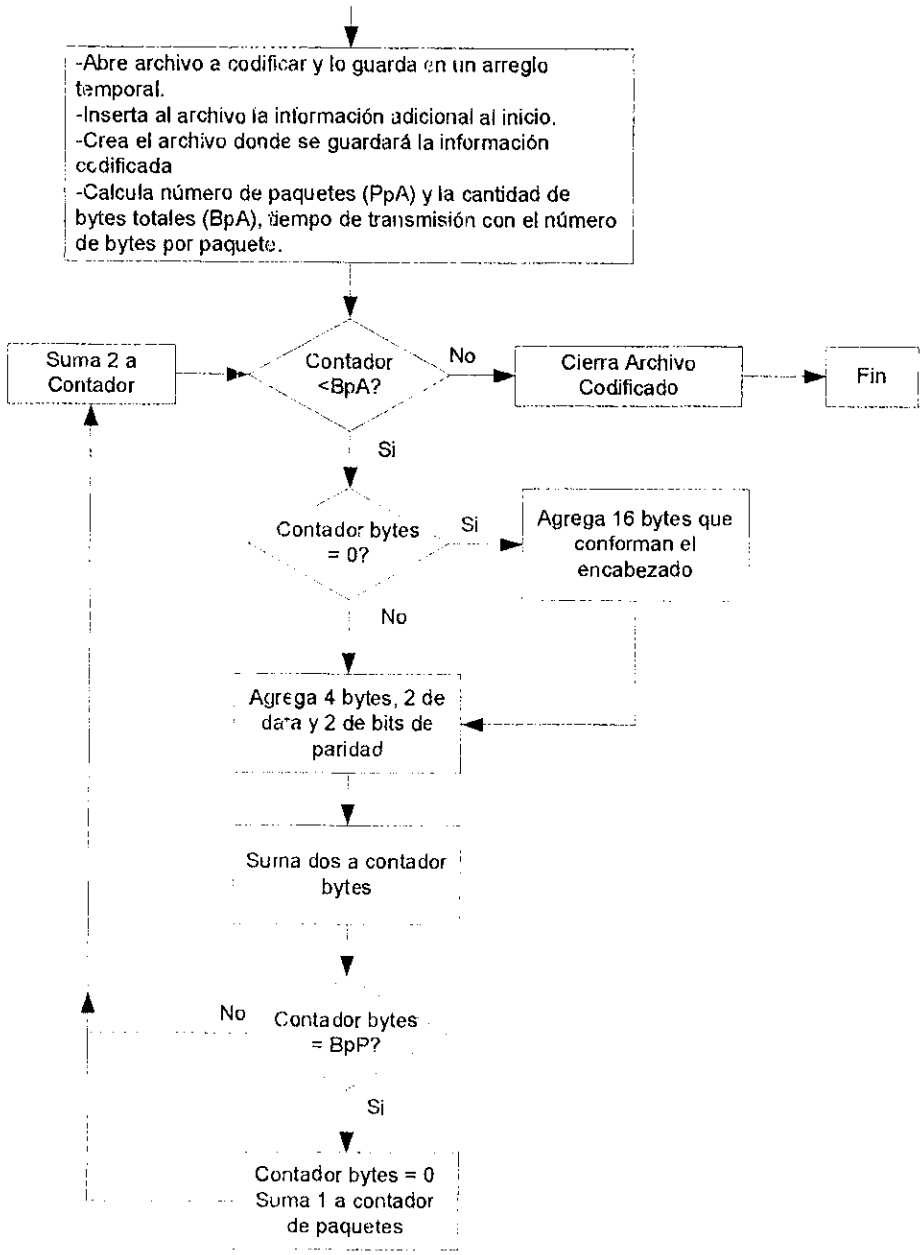


Fig. 23 Diagrama de flujo de la función FPaquetes

La función Ftransmit es la responsable de recorrer el archivo para transmitirlo y posee variables estáticas de modo que la función puede ser utilizada cada vez que se necesite y no perder qué *byte* le toca transmitir. El diagrama de flujo se muestra a continuación.

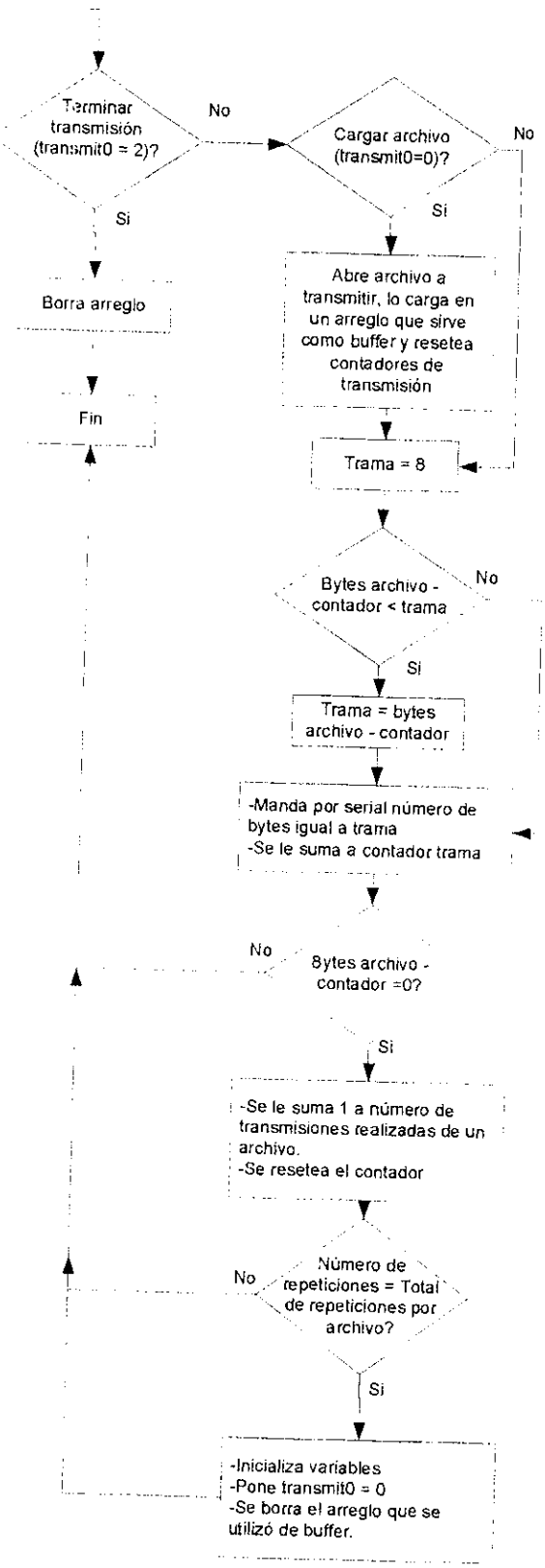


Fig. 24 Diagrama de flujo de la función FTransmit

Se optó por crear una función que fuera la encargada de iniciar la comunicación, mandar a crear el archivo codificado y terminar la comunicación. Esta función es FEnviar, de la cual se muestra el diagrama de flujo a continuación.

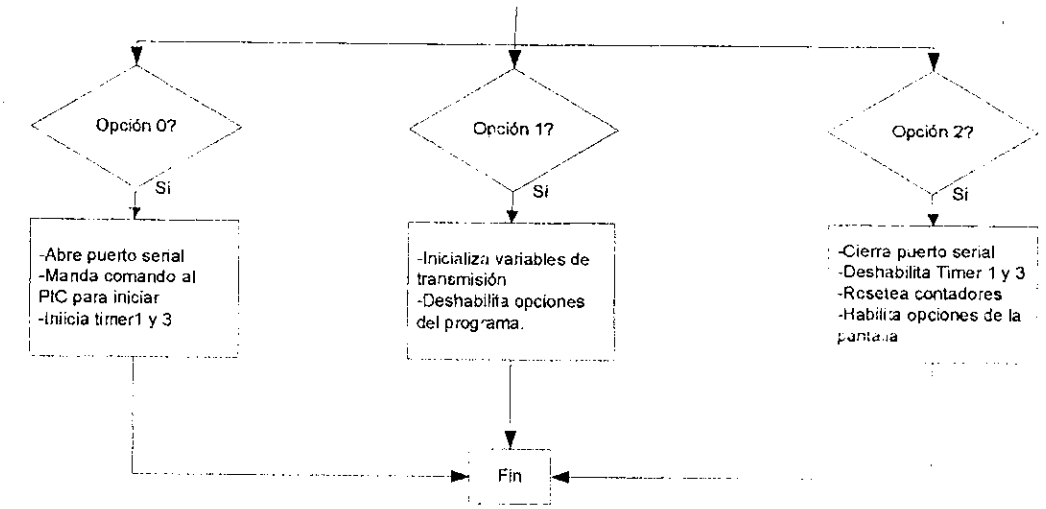


Fig. 25 Diagrama de flujo de la función FEnviar

B. Programa receptor de la computadora

El programa receptor de la computadora tiene como objetivos:

- Recuperar el archivo con el mínimo número de errores
- Decodificar el archivo
- Guardar el archivo recuperado y decodificado en el directorio C:\RadioRDS
- Extraer la información adicional transmitida
- Mandar al microcontrolador la información para la sintonización
- Brindar al usuario un ambiente amigable, fácil de manejar

Este programa al igual que el transmisor utiliza la unidad de manejo del puerto serial y la unidad con las funciones propias de la aplicación.

Existen cinco funciones principales que son:

- FDecoder: Encargada de decodificar y aplicar la corrección de errores al mensaje recibido.
- FDecodificaPaquetes: Es la función encargada de recibir el mensaje ya decodificado e identificar que tipo de mensaje es, si pertenece al encabezado, a que archivo pertenece.

- FRevision: Es la función encargada de llevar la cuenta de los *bytes* recibidos, revisar si se ha completado un paquete, o el archivo completo.
- FGuardaPaquetes: Es la función que se encarga, una vez recuperados todos los *bytes* del archivo, extraer la información que en este viene y guardar el archivo html en el directorio C:\RadioRDS\Nombre de la estación.
- FSintonizar: Es la función que se encarga de ver la calidad de señal RDS y mediante esta identificar si existe una estación con RDS de la cual se pueda recuperar datos.

El diagrama de flujo del funcionamiento del programa receptor se muestra en la Fig.26.

Al igual que en el transmisor, aquí se utiliza un temporizador para recibir los datos del microcontrolador, el cual revisa constantemente si el microcontrolador le ha mandado un bloque más.

La función FDecodPaquetes tiene como objetivo distinguir los distintos bloques que le llegan a la computadora, para ello tiene una serie de comparaciones. Si encuentra los desplazamientos correspondientes a los bloques A, B, C y D en cuatro bloques consecutivos, entonces se sabe que se tiene un nuevo paquete, y se prosigue a guardar los bloques que siguen que serán la data que forman el archivo que se está recuperando.

Fig. 26 Diagrama de flujo del programa de recepción de la computadora

Al principio de esta función se hace una evaluación respecto a la calidad de señal del bloque que se está analizando. Si la calidad de señal se cae más de dos veces en el bloque que son 26 bits, entonces el error promedio del paquete aumenta en uno. Si el paquete llega a tener un error promedio mayor que cero entonces el paquete se marca como erróneo para tener la certeza de que cada bloque que conforma el paquete sera recuperado correctamente por el algoritmo de corrección de errores. A continuación se muestra el diagrama de flujo de la función, Fig.27.

La función FRevisión trabaja en conjunto con la función FDecodPaquetes ya que es la función que se llama cuando el bloque no es parte del encabezado. En esta función se lleva la cuenta de los *bytes* que se han recuperado y por lo tanto de los paquetes que se tienen, cuáles faltan y cuales pueden contener error. Por esto la función conlleva una serie de comparaciones para saber si ya se ha recuperado todo el archivo o si necesita esperar más bloques para encontrar aquellos que le hagan falta. El diagrama de flujo se muestra a continuación en la Fig.28.

Cuando FRevision posee todos los bytes del archivo se llama a la funcion FGuardaPaquetes la cual es la encargada de obtener la información que viene dentro de los paquetes recuperados, como el nombre de la estación, etc. También es la encargada de comparar el número de actualización y de archivo para no tener que sobre escribir el mismo archivo varias veces. El diagrama de flujo se muestran en la Fig. 29

Por último se encuentra la función encargada de la sintonización la cual se basa en los datos que le entrega el PIC sobre la calidad de señal para decidir si ya se ha encontrado otra estación con RDS.

La forma en que esta función opera es mediante la suma de las pérdidas de señal en cierto número de bloques. Cada vez que el programa se encuentra en la situación que se debe hacer una sintonización automática, los bloques recibidos no se pasan a la función FDecodPaquetes; sólo se extrae la información de la señal y se llama a FSintonizar, aquí se suma el número de veces que se fue la señal en ese bloque. Se hace lo mismo con 110 bloques o hasta que la suma exceda un porcentaje de calidad de señal que se este buscando, lo cual indica que la calidad de señal no es lo suficientemente buena como para recuperar la data. El diagrama de flujo se muestra en la Fig. 30

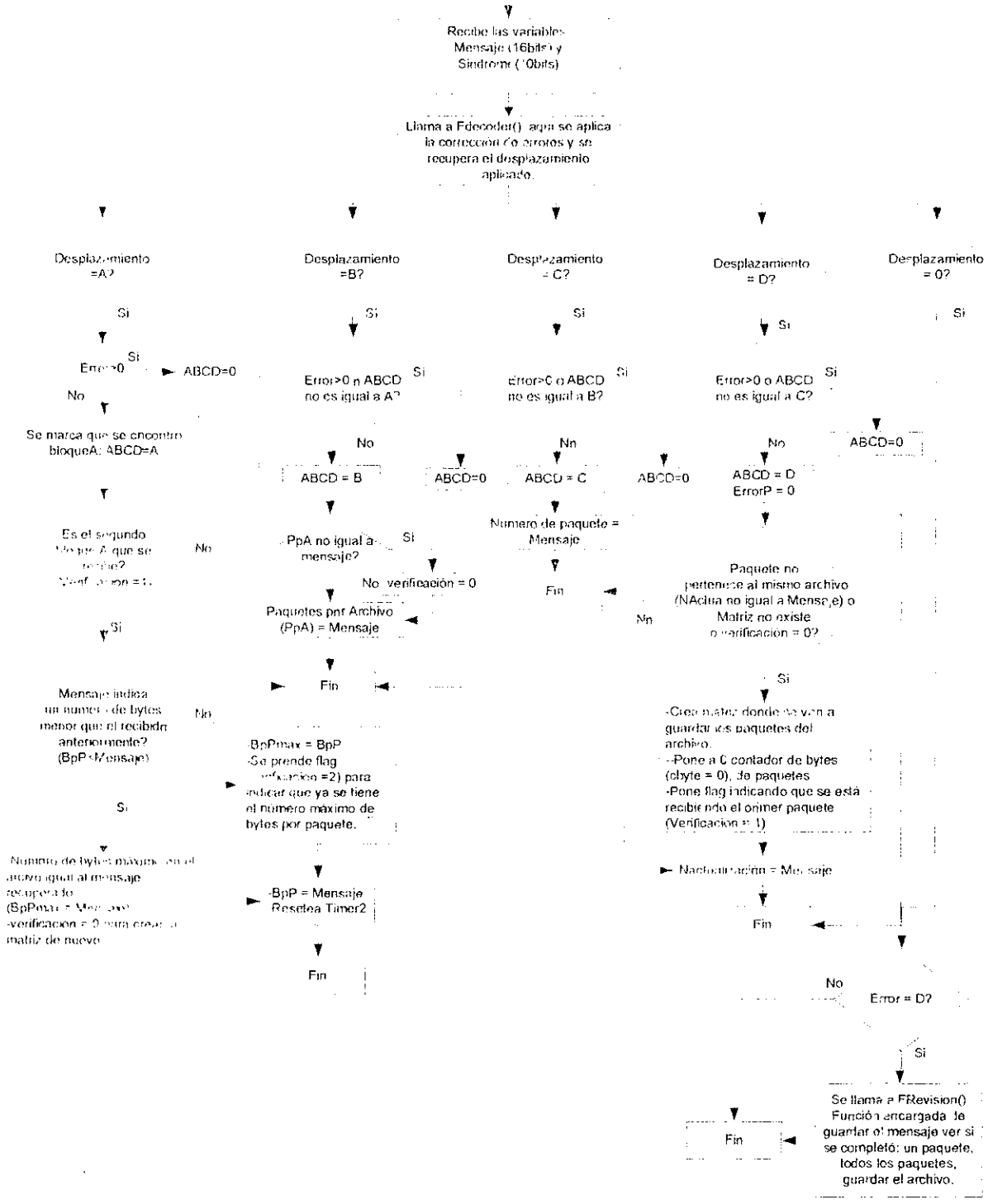


Fig. 27 Diagrama del flujo función FDecodPaquetes

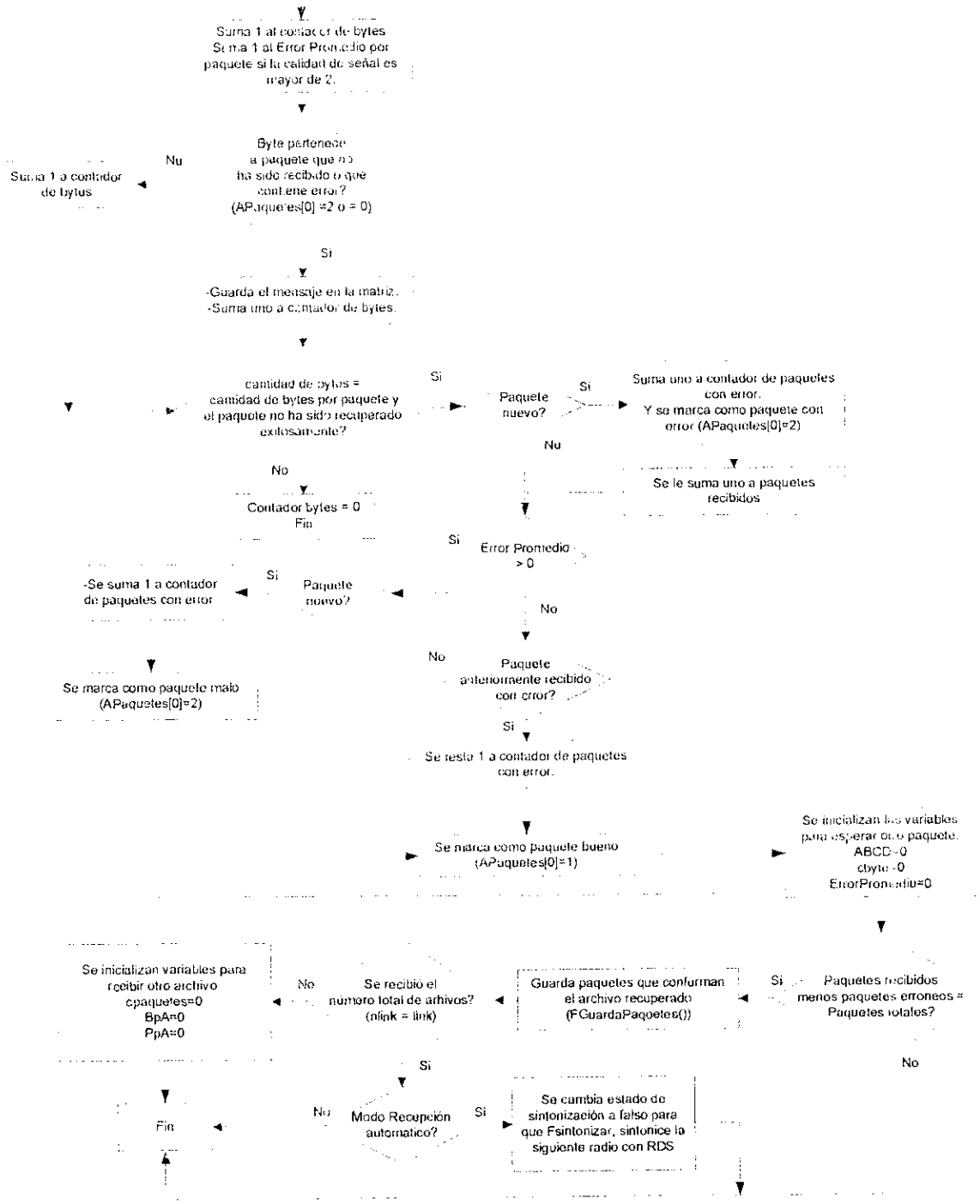


Fig. 28 Diagrama de flujo FRevision

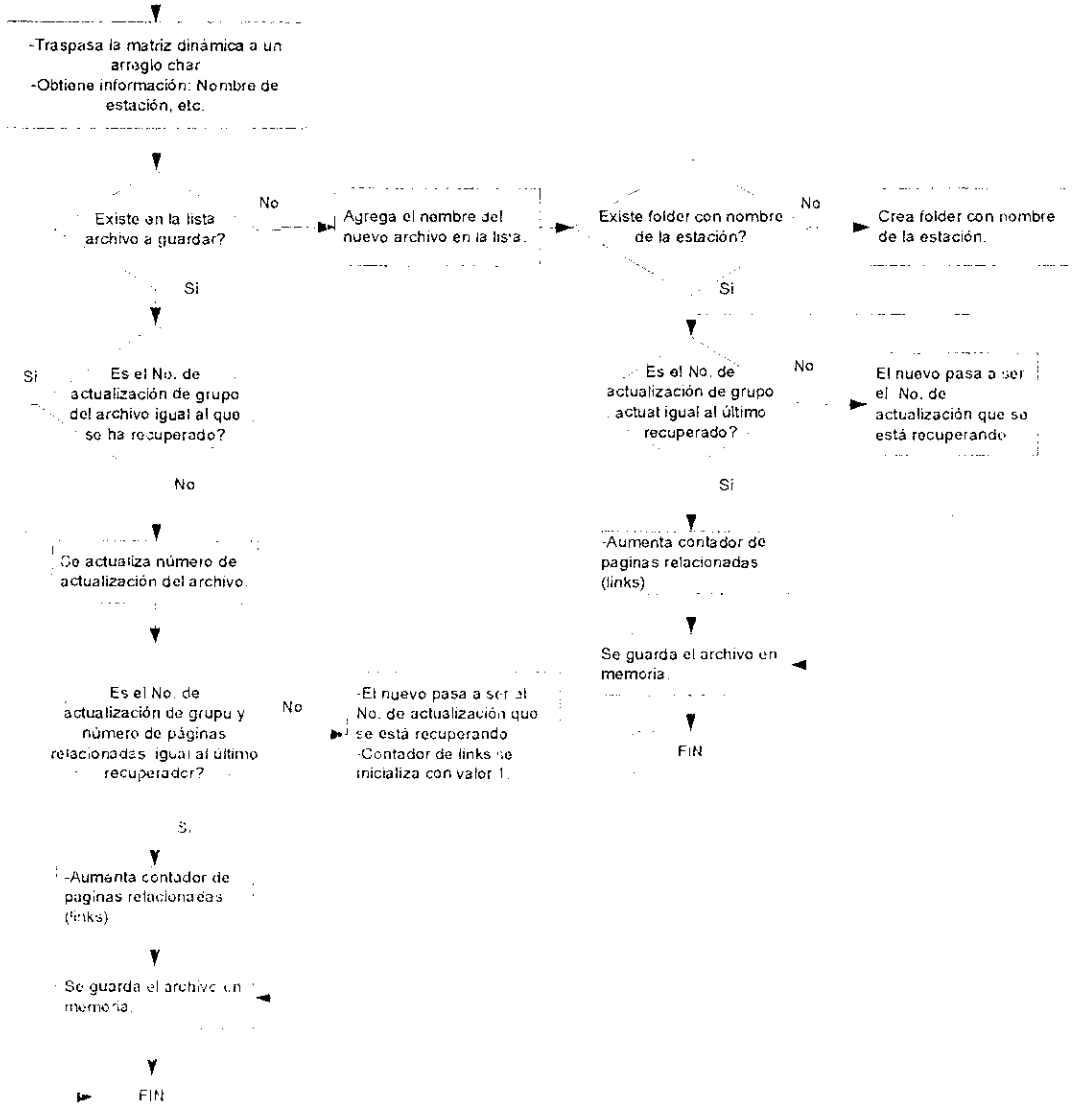


Fig. 29 Diagrama de flujo FguardaPaquetes

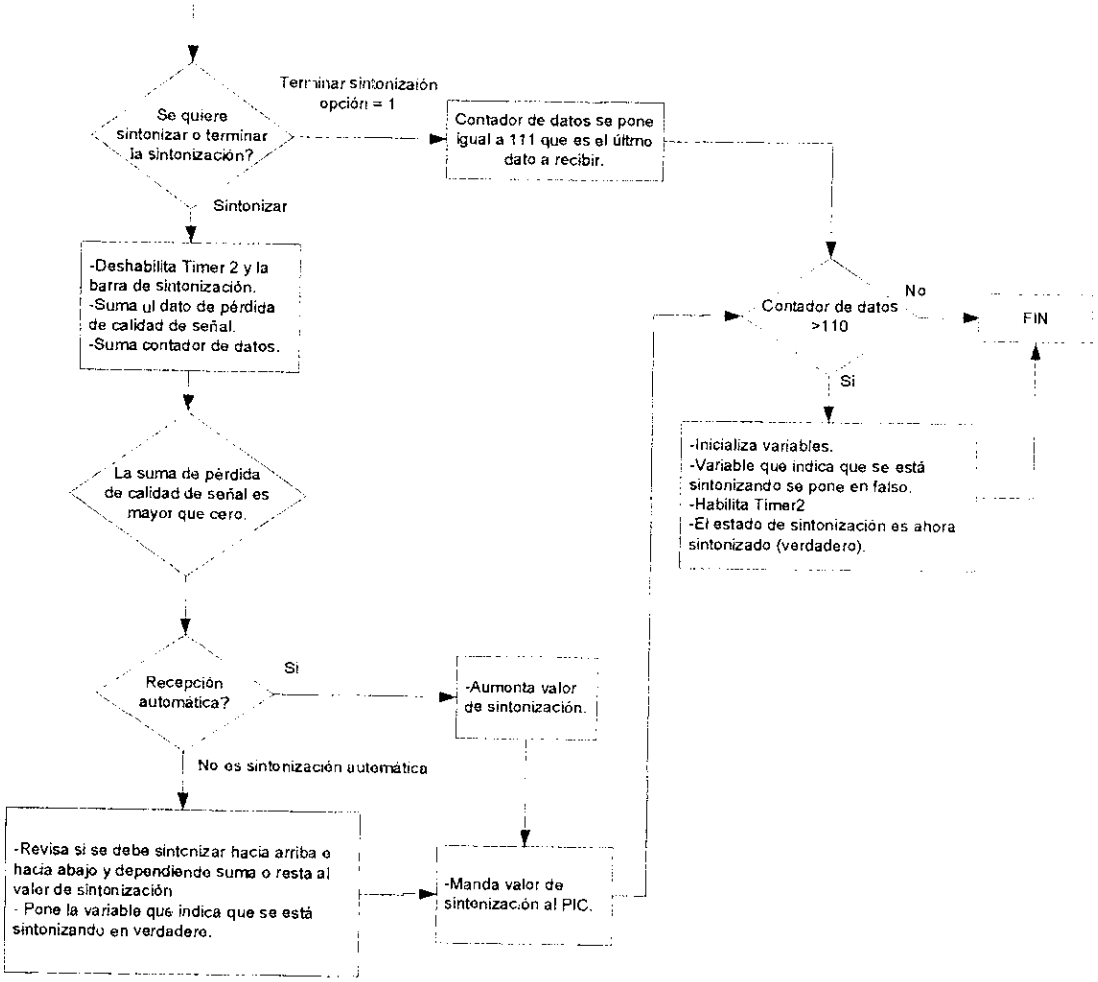


Fig. 30 Diagrama de flujo FSintonizar

C. Programa transmisor del microcontrolador

El programa transmisor del microcontrolador realiza las siguientes funciones:

- Encargarse de la transmisión sincrona de los datos
- Pedirle los datos a la computadora para transmitir cuando ésta le indique y mantener un *buffer* para que los datos siempre estén listos.

Los pines que se utilizaron del microcontrolador son:

- RB0 : como entrada para la señal de reloj
- RB3: como salida para la señal de datos
- RX: como receptor serial de la información transmitida por la computadora
- TX: como transmisor serial de información para la computadora.

La señal de reloj se colocó en el pin RB0 con la intención de utilizar el interrupto para este pin, de manera que cada vez que hay un cambio de flanco (de alto a bajo) en la señal del reloj se carga el bit de datos que se encuentre en el puerto RB3.

La comunicación serial se configuró para una velocidad de 9600, habilitando el interrupto de la recepción serial, de esta manera cada vez que se recibe un *byte* de información de la computadora este se guarda en la posición del *buffer* que le toque.

Al interrupto de RB0 se le dio una mayor prioridad pues éste no se puede hacer esperar, porque estropearía el flujo de información.

Como medida de seguridad para la transmisión serial síncrona se decidió asignar 16 *bytes* de la memoria RAM como *buffer* de transmisión. En estos 16 *bytes* se alojan un total de cuatro bloques por lo que mientras se está transmitiendo el contenido de dos bloques se le pide a la computadora mandar los siguientes dos bloques, de esta manera no importa si la computadora se tarda un poco en responder al pedido del microcontrolador la comunicación síncrona no para ni se queda sin información válida que transmitir.

A continuación se muestran los diagramas de flujo del programa transmisor, Fig..31.

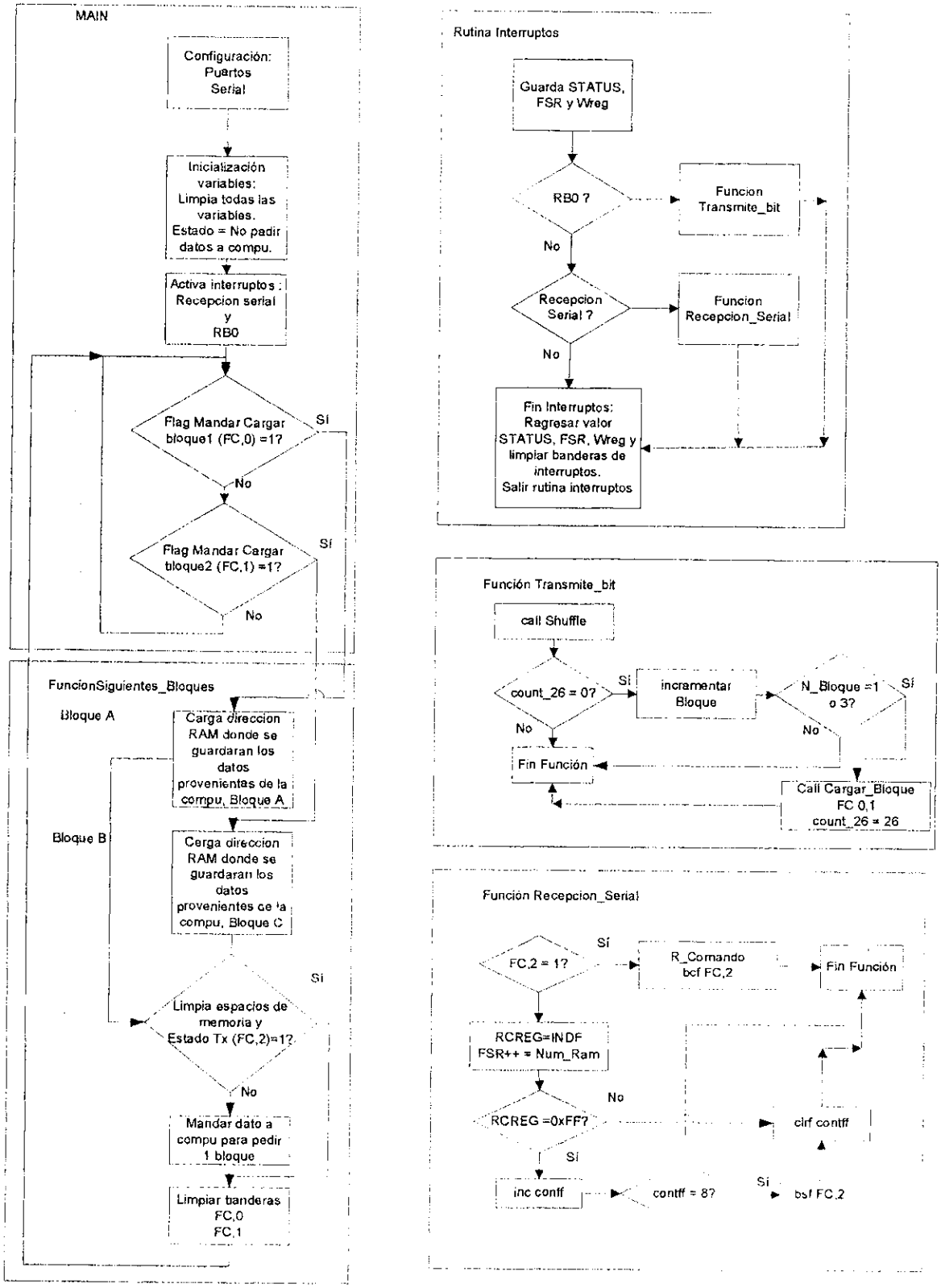


Fig. 31 Diagrama de flujo programa transmisor microcontrolador

D. Programa receptor del microcontrolador

El programa receptor del microcontrolador realiza las siguientes funciones:

- Encargarse de la recepción síncrona de los datos
- Encontrar la sincronía
- Poner la información en *bytes* para mandárselos a la computadora
- Mandar los datos recuperados cuando la computadora le indique
- Cambiar la salida de voltaje para el sintonizador cuando la computadora le indique

Los pines que se utilizaron del microcontrolador son

- RB0 : como entrada para la señal de reloj
- RB3: como salida para la señal de datos
- RB: como entrada para recibir la señal de calidad del TDA7330
- RB : como entrada para recibir la señal de ARI/RDS del TDA7330
- RX: como receptor serial de la información transmitida por la computadora
- TX: como transmisor serial de información para la computadora.
- PORTD: para la salida digital del sintonizador, estas salidas se pasan al convertidor digital análogo para hacer el voltaje apto para el sintonizador.

Al igual que en el transmisor se utilizan dos interruptores, el producido por RB0 y el producido por la recepción serial, y siempre tiene mayor prioridad el interrupto del RB0.

El interrupto de la recepción serial se usa para leer los comandos que la computadora manda al microcontrolador, como: el cambio del voltaje para el sintonizador, parar o empezar a mandar datos a la computadora.

El microcontrolador manda cada vez un bloque a la computadora, para que la computadora conozca el orden de los *bytes*.

Para guardar los bits que se van recuperando en cada señal de reloj, se apartó espacio en la memoria RAM, llamados StoRegX los cuales van corriendo los bits de manera que el que entra de último es el bit menos significativo de los *bytes*. Estos registros ocupan 7 *bytes* para alojar dos bloques.

1. Encontrar la sincronía

La forma de saber si se tiene sincronía es cuando en los registros StoRegX se tiene el bloque A seguido del bloque B. Cada vez que entra un bit se realiza la multiplicación de los registros por la matriz de 26x10 para saber si se tiene un bloque del encabezado. La segunda condición (tener el bloque B después del bloque A) es para estar seguros de que se tiene el encabezado de un paquete, con esta verificación la probabilidad es muy baja de que se cumpla esta condición por perturbaciones en el medio.

Los datos se transmiten a la computadora cuando ésta le indique.

Para trasladar la información sobre la calidad de la señal cuando se recuperó el paquete se utilizaron los bits sobrantes en los *bytes* donde se encuentran los 10 bits de redundancia, ya que el segundo *byte* sólo usa los dos bits más significativos dejando libres 6 bits.

***Bytes* de paridad**	
Información de archivo	**Información de señal**
10 bits	6bits (1 bit Señal RDS y 5 bits para representar las perdidas de señal en 26 bits)

Tabla 6 Bits de transmisión de microcontrolador a computadora

De estos seis bits el más significativo se usa para transmitir la existencia de señal RDS y los 5 bits restantes para sumar la cantidad de veces que la calidad de señal cayó y como se toma una muestra por bit el número máximo que este valor puede tener es 26 y 2 elevado a la 5 es 32 por lo que el valor cabe perfectamente en estos 5 bits.

A continuación se muestra el diagrama de bloques del programa del receptor, en la Fig.32

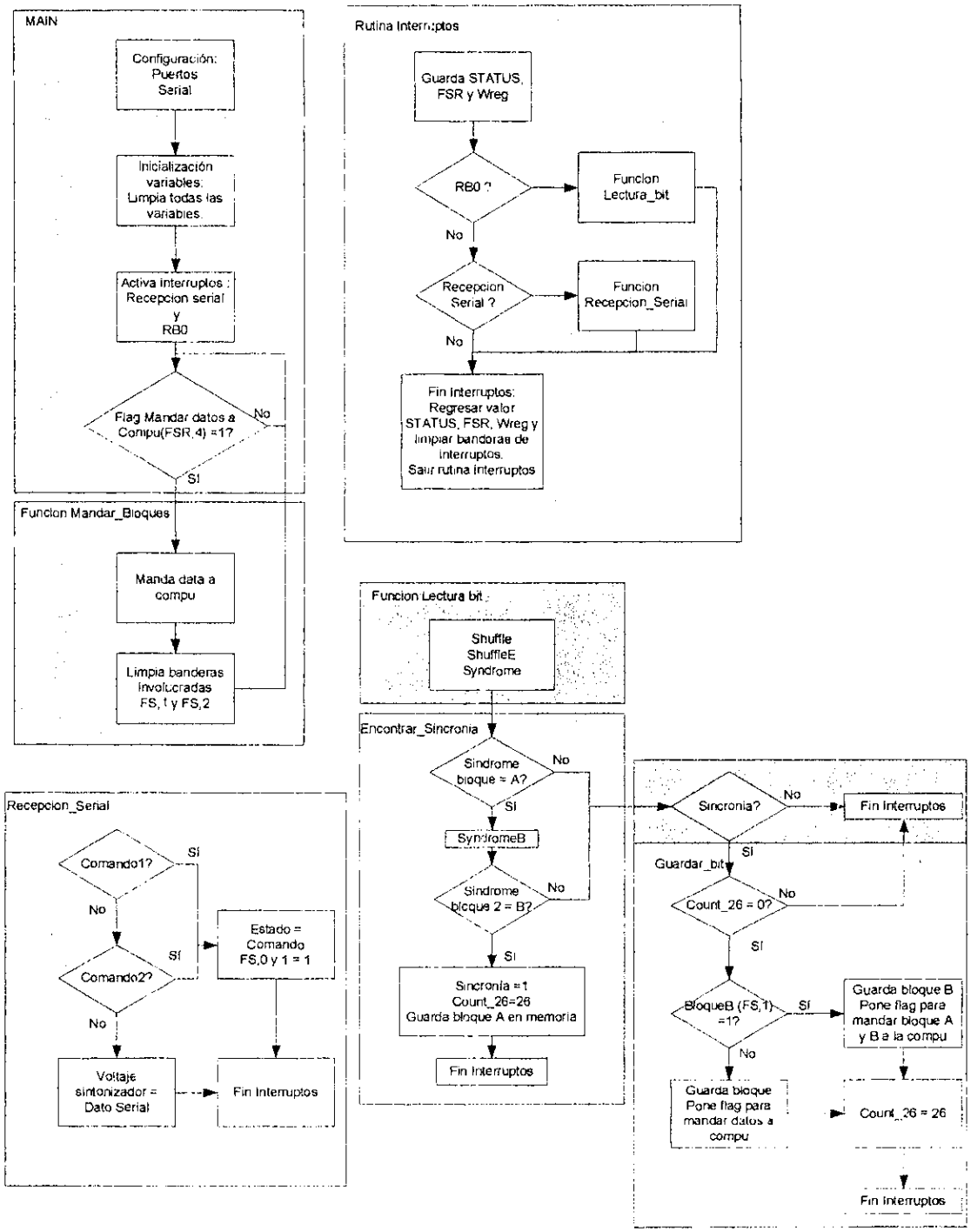


Fig. 32 Diagrama de bloques del programa receptor microcontrolador

VIII. RESULTADOS

Para probar el funcionamiento del protocolo y de la aplicación desarrollada se transmitieron archivos de varios tipos (html, jpeg, txt) con distintos tamaños para ver el comportamiento de la recepción según el tamaño de los *bytes* por paquete y el tamaño del archivo en sí.

Se notó que utilizando de 32-70 *bytes* por paquete los archivos menores a 20*kB* son recuperados a la segunda transmisión. Y aquellos archivos mayores de 20*kB* (menores que 50*kB*) necesitan, en su mayoría, una tercera repetición para ser recuperados por el receptor. Esto se debe a que mientras más grandes sean los archivos, los datos están más propensos a sufrir algún error y por lo tanto se puedan perder ciertos *bytes* de un paquete o llegar con un número de errores que hace que la información sea irrecuperable.

También se hicieron pruebas con paquetes de 100-200 *bytes*, pero el número de repeticiones necesarias, en ciertos casos, debía ser mayor a tres y el tiempo de transmisión no se reduce considerablemente como para equilibrar el número de repeticiones que se deben transmitir.

Por lo tanto entre las pruebas realizadas se cree que el mejor número de *bytes* por paquete es de 70, ya que los archivos se recuperan en la segunda transmisión al igual que con 32 *bytes* pero el tiempo se reduce por ser un número menor de paquetes.

En cuanto al tamaño del archivo que se desea transmitir se puede ver que los archivos de 15 *kB* o menos, son los más adecuados ya que el tiempo y número de repeticiones necesarias para su recuperación son bajos. De esta manera se puede decir que para la transmisión de un archivo grande (mayor de 15 *kB*) es más eficiente fraccionarlo y transmitir archivos pequeños.

A continuación en la Tabla 7 se muestran los datos obtenidos en la transmisión de diez archivos de tamaño distintos con tres tamaños de paquetes.

Tamaño archivo en *bytes*	Tiempo de transmisión de una repetición	Número de paquetes	Número de repeticiones para recuperar archivo	Tiempo aproximado de recuperación
1971	0min 27seg	63	2	0min 54seg
2783	0min 38seg	88	2	1min 16seg
3296	0min 45seg	104	2	1min 30seg
4989	1min 8seg	157	2	2min 16seg
7665	1min 45seg	241	2	3min 30seg
8948	2min 2seg	281	2	4min 4seg
10083	2min 18seg	316	2	4min 36seg
16760	3min 49seg	525	2-3	9min 32seg
20166	4min 36seg	631	3	13min 48seg
48048	10 min 58seg	1503	3	32min 54seg

Tabla 7 Pruebas con 32 *bytes* por paquete

Tamaño archivo en *bytes*	Tiempo de transmisión de una repetición	Número de paquetes	Número de repeticiones para recuperar archivo	Tiempo aproximado de recuperación
1971	0min 24seg	29	2	0min 48seg
2783	0min 34seg	41	2	1min 8seg
3296	0min 40seg	48	2	1min 20seg
4989	1min 1seg	72	2	2min 2seg
7665	1min 33seg	110	2	3min 6seg
8948	1min 49seg	129	2	3min 38seg
10083	2min 3seg	145	2	4min 6seg
16760	3min 24seg	240	2-3	8min 30seg
20166	4min 6seg	289	3	12min 18seg
48048	9 min 46seg	687	3	29min 18seg

Tabla 8 Pruebas con 70 *bytes* por paquete

Tamaño archivo en *bytes*	Tiempo de transmisión de una repetición	Número de paquetes	Número de repeticiones para recuperar archivo	Tiempo estimado de recuperación
1971	0min 22seg	10	3	1min 6seg
2783	0min 32seg	15	3	1min 36seg
3296	0min 37seg	17	3	1min 51seg
4989	0min 57seg	26	3	2min 51seg
7665	1min 27seg	39	3-4	5min 4seg
8948	1min 42seg	45	4	6min 48seg
10083	1min 55seg	51	4	7min 40seg
16760	3min 11seg	84	4	12min 44seg
20166	3min 49seg	101	4	15min 16seg
48048	9 min 7seg	241	4-5	41min 1seg

Tabla 9 Pruebas con 200 *bytes* por paquete

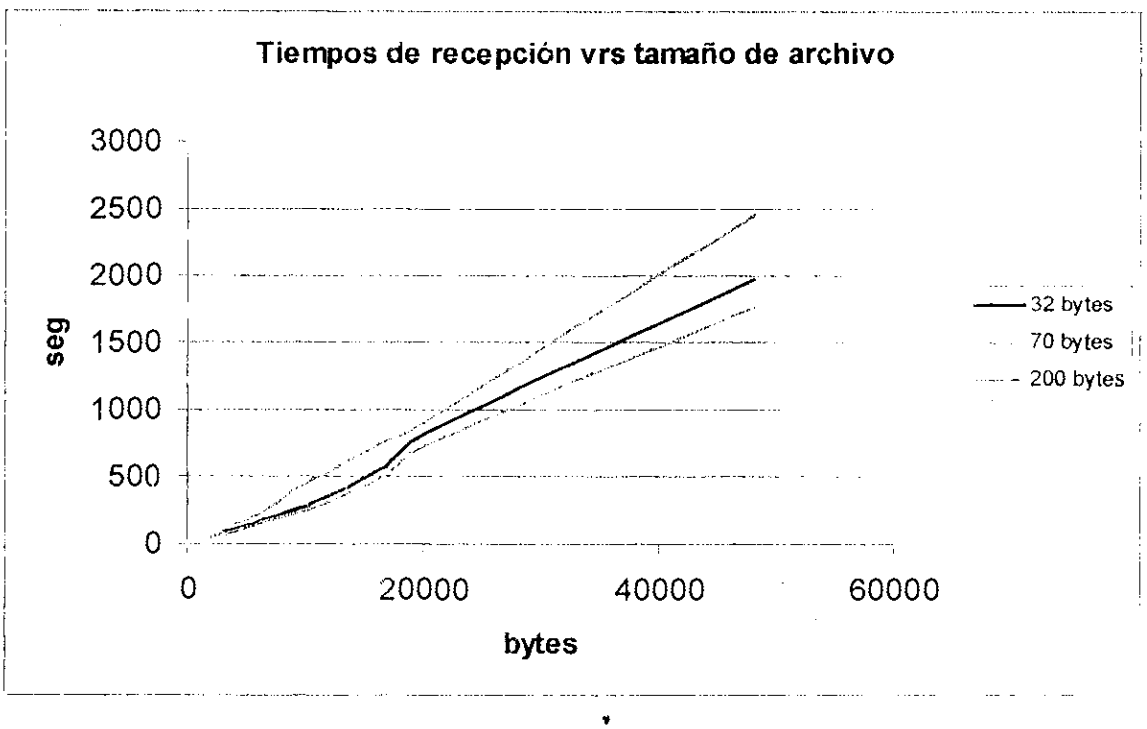


Fig. 33 Tiempo de recepción vrs. tamaño archivo

IX. CONCLUSIONES

A partir de los resultados obtenidos se llegó a las siguientes conclusiones:

- La transmisión de datos html es posible mediante las bases del sistema RDS.
- La flexibilidad del protocolo utilizado hace posible el envío de archivos de cualquier tipo.
- El protocolo de transmisión de datos logra recuperar satisfactoriamente los datos transmitidos en un medio con buena señal RDS.
- El tamaño de los archivos transmitidos debe ser moderado debido a que las distorsiones de señal del medio hacen necesarias repeticiones numerosas en el transmisor y aumentan el tiempo de recuperación del archivo.
- El número de *bytes* por paquete no puede ser muy grande ya que la probabilidad de que haya un error en la señal es mayor, demorando la recuperación total de archivo.
- El paquete compuesto por 32 *bytes* probó ser un número adecuado para la transmisión de datos en sistemas con una cantidad media de ruido, mientras que el paquete de 70 *bytes*, demostró ser superior en ambientes con muy buena recepción, ya que permite reducir el tiempo de transmisión de datos en comparación al de 32 *bytes*.

X. RECOMENDACIONES

Para mejorar los resultados obtenidos en este trabajo se proponen las siguientes recomendaciones:

- Utilizar un sintonizador digital para mejorar la forma en que se sintonizan las estaciones de radio con sus respectivas señales de RDS.
- Adecuar los programas de los microcontroladores para sustituir el PIC16F877 por microcontroladores que posean, además de todas las cualidades necesarias, una presentación más compacta y un precio más económico, evitando de esta forma el desperdicio de espacio en el PCB y el desperdicio de recursos del microcontrolador. Por ejemplo el PIC16F627 en el transmisor, y el PIC16F73 en el receptor.
- Modificar los campos de paquetes para que el protocolo sea totalmente compatible con el sistema RDS utilizado en Europa y Estados Unidos.
- Incluir en los programas de aplicación una opción de inicio automático del programa, para que funcione como un servicio más de la computadora y que se inicie al momento de que el sistema operativo empiece a funcionar.
- Agregar controladores para que el programa sea compatible con varios sistemas operativos como Windows o Linux (Unix) en sus distintas versiones.
- Utilizar la aplicación desarrollada en este trabajo para otras aplicaciones de transmisión de datos que sólo necesiten transmisiones de una vía.

XI. BIBLIOGRAFÍA

Morelos-Zaragoza, Robert. 2002. *The Art of Error Correcting Coding*. John Wiley and Sons. Nueva York, Estados Unidos. 220 págs.

PIC16F87X Datasheet. 2003. Microchip Technology Inc. Chicago, Estados Unidos. 284 págs.

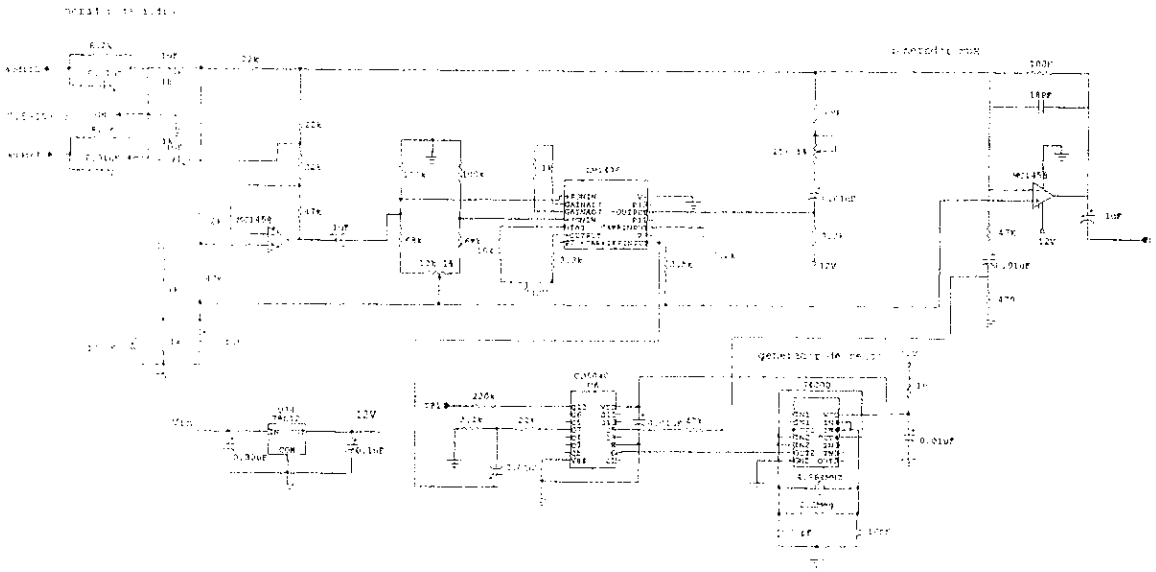
RDS Milestone. 2000. EBU and RDS Forum. Inglaterra. 12 págs.

United States RBDS Standard: Specification of the radio broadcast data system (RBDS). 1998. National Radio System Committee. 204 págs.

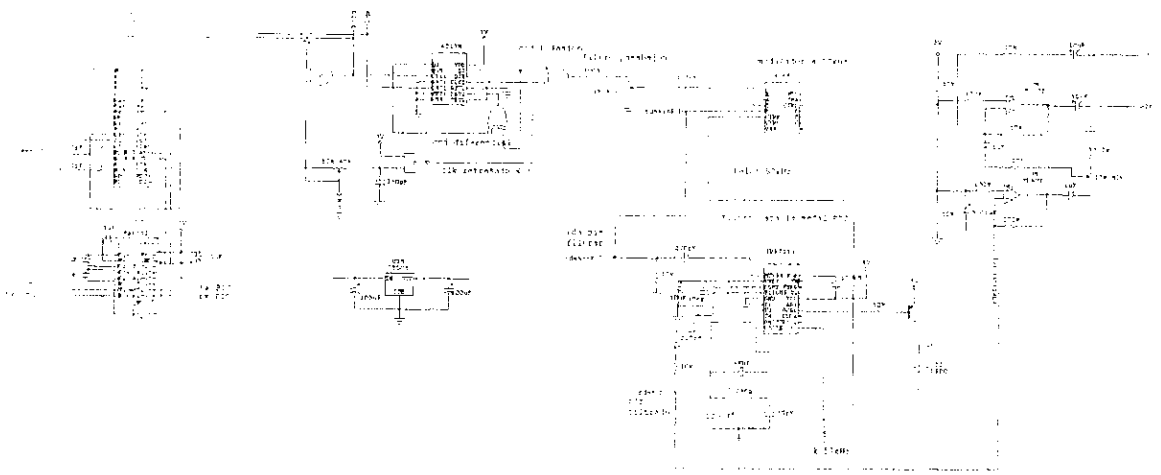
XII. APÉNDICE

A. Diagrama completo del módulo transmisor

1. Generador de Audio



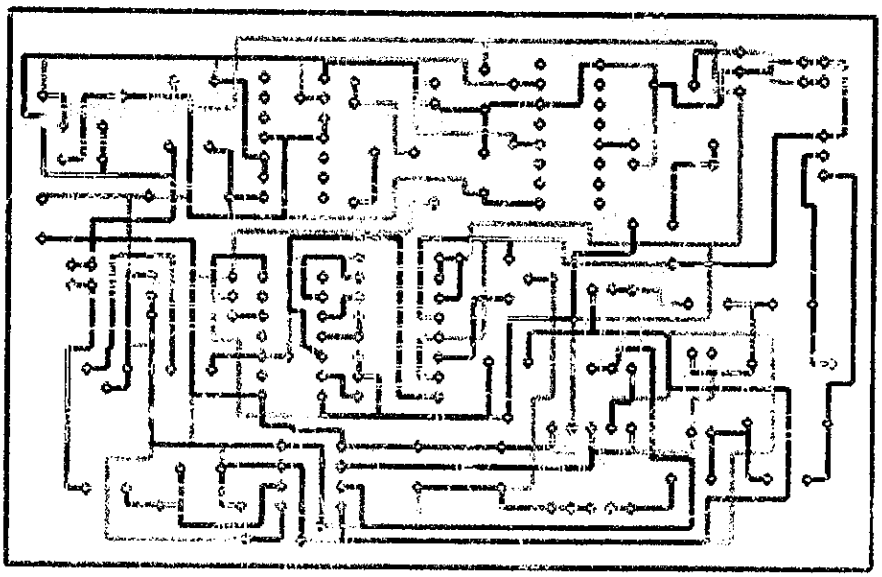
2. Codificador RDS



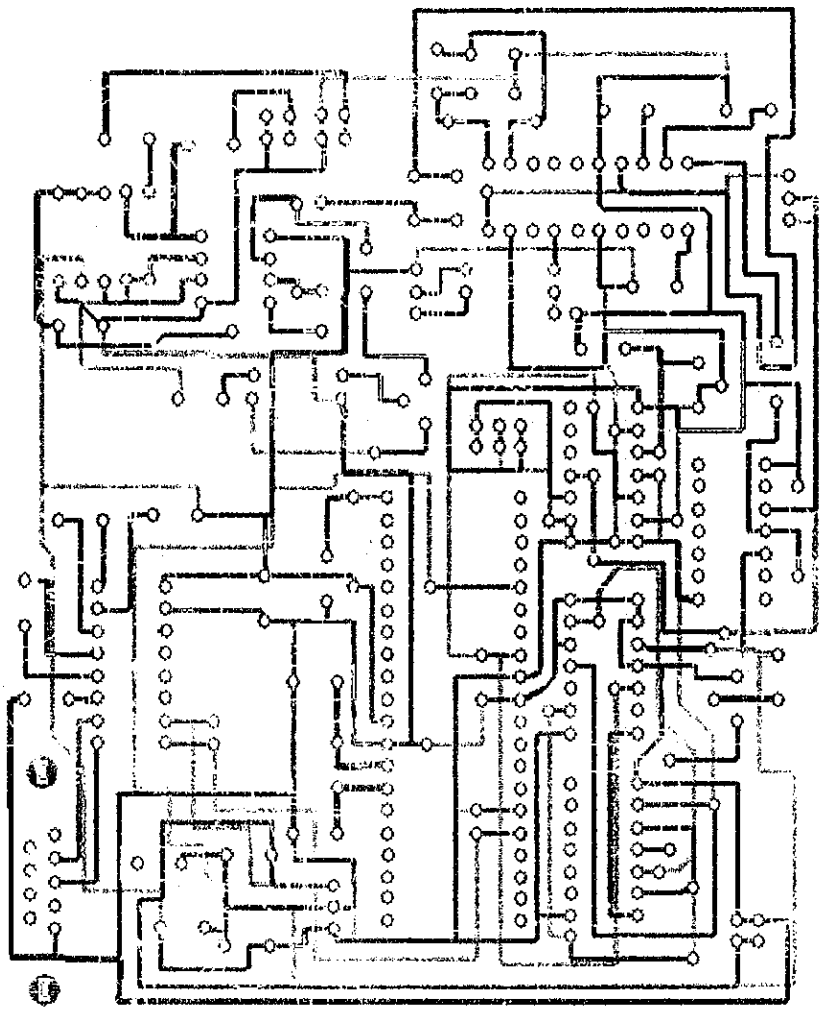
B. Diagrama completo del módulo receptor

C. Esquemas de los circuitos impresos

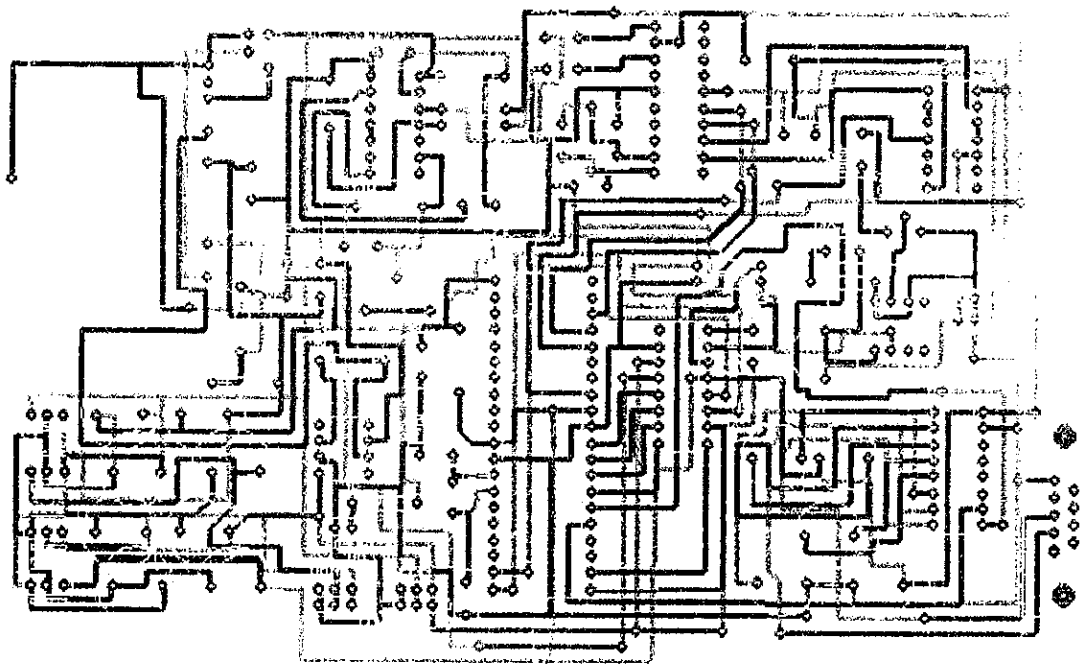
1. Codificador estéreo



2. Codificador RDS



3. Receptor



D. Código en lenguaje ensamblador para el PIC16F877 transmisor

```
;  Nombre:        Trdsp.asm                    *

;  Notas: Manda bloques de data RDS recibida de la computadora
;         Datos salen en flanco de bajada.
;      RB1 = alto cuando se transmite bloque s bajo en todos los demas
;                Pide 2 bloques a la computadora mientras transmite los otros dos
;                Limpia la memoria de los bloques donde va a entrar la nueva data
;                Una vez la computadora deja de mandar el archivo el PIC no le pide mas
;                bloques hasta que le manden un comando.
;*****************************************************************************
      list    p=16f877
      #include <p16f877.inc>

      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _HS_OSC & _LVP_OFF

;***** CONSTANTES
clk         EQU     .0          ;pines para comunicacion tda7330b
data_out    EQU     .3

;***** VARIABLES DE ALMACENAMIENTO DE BLOQUES
Reg1A       EQU  0x20  ;Bloque A1
Reg1B       EQU     0x21  ;Bloque A2
Synd1A      EQU  0x22  ;8bits mas significativos de Sindrome
Synd1B      EQU     0x23  ;2bits menos significativos de Sindrome

Reg2A       EQU     0x24
Reg2B       EQU     0x25
Synd2A      EQU     0x26
Synd2B      EQU     0x27

Reg3A       EQU     0x28
Reg3B       EQU     0x29
Synd3A      EQU     0x2A
Synd3B      EQU     0x2B

Reg4A       EQU     0x2C
Reg4B       EQU     0x2D
Synd4A      EQU     0x2E
Synd4B      EQU     0x2F

;***** VARIABLES DE USO DINAMICO
StoReg1     EQU     0x30  ;registros para transmision
StoReg2     EQU     0x31
StoReg3     EQU     0x32
StoReg4     EQU     0x33

N_Bloque    EQU     0x3A  ;contador #bloque que se transmite
Count_26    EQU     0x3C  ;contador de 26 bits


W_Temp      EQU     0x3E  ;temporal
Status_Temp EQU     0x3F
FSR_temp    EQU     0x40

Flag_Cargar EQU     0x41  ;bit0 indica cargar Bloque A y B, Bit1 C y D, Bit2 Indica no transmitir a la compu
Cont_FF     EQU     0x42  ;lleva la cuenta de los bytes que llegan que son FF
Num_Ram     EQU     0x43  ;direccion RAM a la cual toca grabar el byte recibido por serial
```

```
;*****************************************************************
        ORG   0x000
        goto  main

;*****************************************************************
;*****************INTERRUPTOS*************************************
;*****************************************************************
        ORG   0x004
Interruptos:
        movwf   W_Temp              ;guarda el Status
    swapf   STATUS, w
    movwf   Status_Temp
        movf  FSR,w
        movwf   FSR_temp                ;guarda posicion FSR

        banksel   PIR1
        btfsc  INTCON, INTF         ;interrupto de RB0
        goto   Transmite_Bit

        btfsc  PIR1, RCIF           ;revisa si es interrupto de recepcion serial
        goto   Recepcion_Serial

Fin_Interruptos
        movf  FSR_temp,w
        movwf   FSR
        swapf   Status_Temp, w    ; regresa el status
        movwf   STATUS
        swapf Status_Temp, F
        swapf   Status_Temp, w
        retfie

;*****************************************************************
;Transmite_Bit: manda un bit a data_out y si ya envio el ultimo bit
;de un bloque carga el siguiente bloque.
;*****************************************************************
Transmite_Bit
        bcf     INTCON, INTF    ;borra flag
        call    Shuffle             manda bit a data_out

        banksel   PORTB
        movlw   1
        xorwf N_Bloque, w
        btfsc  STATUS, Z
        goto   Transmite_BloqueA

Transmite_BloqueB
        bcf     PORTB, 1        ;para control de transmision bloque A
        goto   Transmite_Normal

Transmite_BloqueA
        bsf     PORTB, 1

Transmite_Normal
        decfsz Count_26,f           ;decrementa contador
        goto   Fin_Interruptos      ;si no es 0 entonces se sale

        incf   N_Bloque,f           si es igual a 0 entonces ya se tx bloque entonces N_Bloque+1
        call   Cargar_Bloques       carga siguiente bloque

        movlw   1
        xorwf N_Bloque,w
        btfsc  STATUS,Z
        bsf     Flag_Cargar,1           ;indica que hay que cargar bloques C y D

        movlw   3
        xorwf N_Bloque,w
        btfsc  STATUS,Z
        bsf     Flag_Cargar,0           ;indica que hay que cargar bloques A y B

        movlw   26
```

```
        movwf   Contt_26                ;pone cuenta a 26 de nuevo
        goto    Fin_Interruptos         ;se sale del interrupto

;*****************************************************
;Recepcion_Serial: recibe datos de la computadora, si el flag
;de no transmitir esta prendido entonces el dato que le llegue
;se toma como comando de inicio y debe ser igual a 1.
;de lo contrario se toma como dato para transmision
;*****************************************************
Recepcion_Serial
        btfss   Flag_Cargar,2           ;si esta prendido entonces es comando
        goto    Recepcion_Bloques       ;si no son datos

Recepcion_Comando
        movf    RCREG,w
        xorlw   .73                     ;si es igual a 'I' entonces es el comando de inicio
        btfsc   STATUS,Z
        bcf     Flag_Cargar,2           ;se limpia la bandera
        goto    Fin_Interruptos

Recepcion_Bloques
        movf    Num_Ram,w               ;pone el puntero en la posicion de memoria Num_Ram
        movwf   FSR
        movf    RCREG,w
        movwf   INDF                    ;mueve el byte recibido a la memoria
        incf    Num_Ram,f

        movlw   0xFF                    ;si se recibe un byte 0xFF
        xorwf   INDF,w
        btfss   STATUS,Z
        goto    Recepcion_NFF

Recepcion_FF
        incf    Cont_FF,f       ;se aumenta el contador
        movlw   .8              ;si es igual a 8 representa la terminacion de la tx
        xorwf   Cont_FF,w
        btfss   STATUS,Z
        goto    Fin_Interruptos
        clrf    Cont_FF
        bsf     Flag_Cargar,2   ;se deshabilita el mandar dato a la computadora
        goto    Fin_Interruptos

Recepcion_NFF
        clrf    Cont_FF                 ;se limpia el contador
        goto    Fin_Interruptos


;*****************************************************
;Cargar_Bloques: Carga el numero de bloque correcto a los bytes
;que seran enviados por data_out  StoReg1-4
;*****************************************************
Cargar_Bloques

        movlw   .5
        xorwf   N_Bloque,w
        btfss   STATUS,Z
        goto    Cargar_0
                        ;si contador de bloques igual a 5
        movlw   .1              ;quiere decir que ya se regreso al
        movwf   N_Bloque        ;bloque 1

Cargar_0
        movf    N_Bloque,w
        call    Cargar_Tabla    ;segun el contador se carga la direccion
        movwf   FSR             ;correcta de memoria en FSR

        movf    INDF,w
        movwf   StoReg1                 ;mueve el byte bloque mas significativo
        incf    FSR,f
```

```
        movf  INDF,w
        movwf   StoReg2             ;mueve 2do byte bloque   menos significativo
        incf   FSR,f

        movf  INDF,w
        movwf   StoReg3             ;mueve parte mas significativa de Sindrome
        incf   FSR,f

        movf  INDF,w
        movwf   StoReg4             ;mueve parte menos sigificativa de Sindrome

        return

;*****************************************************
;Cargar_Tabla: Tabla direcciones de los distintos bloques
;*****************************************************
Cargar_Tabla
        addwf   PCL, f
        nop
        retlw  0x20  ;direccion Bloque 1
        retlw  0x24  ;direccion Bloque 2
        retlw  0x28  ;direccion Bloque 3
        retlw  0x2C  ;direccion Bloque 4

;*****************************************************
;Shuffle: rota los bytes donde se guardan los 26 bits de manera
;que el primer bit que sale es el menos significativo
;Usa los registros StoReg4-1
;*****************************************************
Shuffle
        bcf     STATUS, C
        rlf     StoReg4, F          ; rotar bit
        rlf     StoReg3, F
        rlf     StoReg2, F
        rlf     StoReg1, F
        btfsc  STATUS, C        ; bit nuevo
        goto   Shuffle_1

Shuffle_0
        bcf     PORTB, data_out         ;Manda bit
        goto   Fin_Shuffle

Shuffle_1
        bsf     PORTB, data_out

Fin_Shuffle
        return

;*****************************************************
;Siguientes_Bytes: Manda dato por serial para avisar a la computadora
;que mande los siguientes bloques
;*****************************************************
Siguientes_Bytes

Siguiente_A

        movlw   0x20            ;Carga direccion datos bloque A y B
        movwf   Num_Ram
        movlw   0x27
        movwf   FSR
        call   clean_RAM        ;limpia los bloques con 1s y 0s
        btfsc  Flag_Cargar,2
        goto   Fin_siguiente
        movlw   '0'                     ;Manda a pedir bloques
        call   Serial_Tx
        goto   Fin_siguiente

Siguiente_B
        movlw   0x28            ;Carga direccion datos bloque C y D
        movwf   Num_Ram             ;Carga la direccion de RAM donde se guardaran los datos bloque A y B
```

```
        movlw   0x2F
        movwf   FSR
        call    clean_RAM
        btfsc   Flag_Cargar,2
        goto    Fin_siguiente
        movlw   'I'                     Manda a pedir bloques
        call    Serial_Tx

Fin_siguiente
        bcf     Flag_Cargar,0           ;limpia banderas
        bcf     Flag_Cargar,1
        return

;*****************************************************
;Clean_RAM: Limpia datos de ultima posicion hasta Num_Ram
;con byte b'10101010'
;*****************************************************

clean_RAM
        movlw   b'10101010'
        movwf   INDF
;       clrf    INDF                    ;
        decf    FSR,f
        movf    Num_Ram,w
        subwf   FSR,w
        btfsc   STATUS,C
        goto    clean_RAM
        return
;*****************************************************
;Clean_RAM1: Limpia datos de ultima posicion hasta Num_Ram
;con valor 0x00
;*****************************************************

clean_RAM1
        clrf    INDF                    ;
        decf    FSR,f                   ;
        movf    Num_Ram,w
        subwf   FSR,w
        btfsc   STATUS,C
        goto    clean_RAM1
        return

;*****************************************************
;Serial_Tx: Manda dato en Wreg por serial
;*****************************************************
Serial_Tx
        movwf   TXREG

SerialTx_Wait_Flag
        btfss   PIR1,TXIF               ;bandera
        goto    SerialTx_Wait_Flag

        return


;*****************************************************
;I_Serial: INICIACION PUERTO SERIAL
;*****************************************************
I_Serial:
        banksel SPBRG
    movlw   .129                        ;para 20MHz y  9600 baud rate
    movwf   SPBRG
    bcf     TXSTA,SYNC                  ;activado  modo asincrono
    bsf     TXSTA, BRGH                 ;BRGH high
    bsf     PIE1, RCIE                  ;interrupto recepcion
    bcf     PIE1, TXIE                  ;interrupto  transmision
    bsf     TXSTA,TXEN                  ;transmision activada
        asf             TRISC,6                     ;como entrada los pines RX/TX
        bsf             TRISC,7
```

```
        banksel     RCSTA
    bsf     RCSTA, SPEN         ;puerto serial activado
    bsf     RCSTA, CREN         ;recepcion activada

        return


;*****************************************************************
;*****************MAIN********************************************
;*****************************************************************
main

        banksel TRISA
        movlw       b'11111111'
        movwf       TRISA                   ;puertos como entradas
        movwf       TRISB                   ;1 clock y 0 datos
        bcf         TRISB, data_out         ;data_out como salida
        bcf         TRISB, 1            ;Marca tx bloque A
        bcf         OPTION_REG, INTEDG          ;se activa con flanco de bajada

        call    I_Serial            ;inicializa puerto serial

        movlw       0x42            ; Se pone al principio memoria RAM
        movwf       FSR                 ; banco 0

        movlw       0x20
        movwf       Num_Ram
        call    clean_RAM           ; limpia todas las posiciones entre 0x20-0x80

I_Int

        movlw       .3
        movwf       N_Bloque            ;inicializacion de variables para pedir primer bloque a la computadora
        movlw       .26
        movwf       Count_26

        call    Cargar_Bloques      ;pone listo el primer bloque para transmision 0's

        bsf         Flag_Cargar,0           ;indica cargar info de serial a bloque AB
        bsf         Flag_Cargar,2           ;indica que espera comando de la compu para pedir bloques

        bsf     INTCON, GIE             ;global activado
    bsf     INTCON, PEIE        ;interruptos perifericos activados
        bsf         INTCON, INTE            ;interrupto RB0 activado


main_loop
        nop
        btfsc   Flag_Cargar, 0          ;indica que se debe pedir bloque A y B
        call    Siguiente_A
        btfsc   Flag_Cargar, 1          ;indica que se debe pedir Bloque C y D
        call    Siguiente_B
        nop
        goto    main_loop

        END
```

E. Código en lenguaje ensamblador para el PIC16F877 receptor

```
;  Nombre:          Drdsp.asm                    *
;
;  Notas:   Recoge bits: RB0 clk, RB3 datos y  muestra en:

;                   RB1 muestra los bits que se van guardando
;                   RB4 entrada de RDS signal TDA7330
;                       RB5 entrada de Quality signal TDA7330
;                   PORTD salida digital para sintonizar

;                   Mandada de datos por serial activado manda bloque A,B y bloque1
;                       Realizado para version con paquetes de #bytes indefinido
;                   Recibe datos por serial                  *
;                   Revisa block A y B para recuperar la sincronia
;                       Manda datos por serial solo cuando la compu se lo indica
;                       0x01 para empezar y 0x02 para terminar
;                       Manda cantidad de errores en un bloque en los bits menos
;                       significativos del sindrome.
;*****************************************************************************
      list    p=16f877
      #include <p16f877.inc>

      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _HS_OSC & _LVP_OFF


;******* CONSTANTES
clk           EQU  .0           ;pines para comunicacion tda7330b
data_in       EQU  .3
rds_sgn       EQU .4
qlty_sgn      EQU .5

;******* VARIABLES DE ALMACENAMIENTO DE BLOQUES
Reg1A         EQU  0x20  ;Byte mas significativo data
Reg1B         EQU  0x21  ;Byte menos significativo data
Synd1A        EQU  0x22  ;8 bits mas significativos del sindrome
Synd1B        EQU  0x23  ;2 bits menos significativos del sindrome

Reg2A         EQU  0x24
Reg2B         EQU  0x25
Synd2A        EQU  0x26
Synd2B        EQU  0x27

Reg3A         EQU  0x28
Reg3B         EQU  0x29
Synd3A        EQU  0x2A
Synd3B        EQU  0x2B

;***** VARIABLES DE USO DINAMICO
StoReg1       EQU  0x2C         ;variables donde se van guardando y corriendo
StoReg2       EQU  0x2D         ;los bits entrantes
StoReg3       EQU  0x2E
StoReg4       EQU  0x2F
StoReg5       EQU  0x30
StoReg6       EQU  0x31
StoReg7       EQU  0x32

StoReg1X  EQU  0x33         ;se utilizan para copiar los registros originales
StoReg2X  EQU  0x34         ;y obtener el sindrome
StoReg3X  EQU  0x35
StoReg4X  EQU  0x36
StoReg5X  EQU  0x37
StoReg6X  EQU  0x38
StoReg7X  EQU  0x39

StoReg1E  EQU  0x3A         ;se utilizan para guardar el estatus de la señal
StoReg2E  EQU  0x3B         ;quality y existencia de RDS.
StoReg3E  EQU  0x3C
```

```
StoReg4E    EQU  0x5D
StoReg5E    EQU  0x3E
StoReg6E    EQU  0x3F
StoReg7E    EQU  0x40


HiSynd              EQU  0x41  ;sindrome parte alta y baja
LowSynd             EQU  0x42

CountReg    EQU  0x44  ;contador de Syndrome
Count_26    EQU  0x45  ;contador de 26 bits
DelayReg    EQU  0x46  ;contador para delay

W_Temp              EQU  0x47  ;temporal
Status_Temp         EQU  0x48
FSR_temp    EQU  0x49

FlagSincronia       EQU  0x4A ;00000xBA  A y B = indican sincronia
                                        ;                   1 = Bloque de datos
Dato_Serial EQU  0x4C
FSR_Fin             EQU 0x4D
FSR_E               EQU 0x4E
Estado              EQU  0x4F           ;estado de transmision; que le manda la computadora
Suma_Error EQU 0x50             ;suma de los banderas defalta de calidad de señal
;*********************;*************************************************
        ORG   0x000
          goto  main

;*****************************************************************
;*****************INTERRUPTOS*************************************
;*****************************************;*************************
        ORG   0x004

Interruptos:
        movwf       W_Temp                  ;guarda el Status
        swapf STATUS, w
        movwf       Status_Temp
        movf  FSR,w
        movwf       FSR_temp                ;guarda posicion FSR

        banksel     PIR1
        btfsc  INTCON, INTF             ;interrupto de RB0
        goto   Lectura_Bit

        btfsc  PIR1, RCIF               ;revisa si es interupto de recepcion serial
        goto   Recepcion_Serial


Fin_Interruptos
        bcf         INTCON,INTF
        movf  FSR_temp,w
        movwf       FSR
        swapf       Status_Temp, w      ;regresa el status
        movwf       STATUS
        swapf Status_Temp, F
        swapf       Status_Temp, w

        retfie

;*****************************************************
;Lectura_Bit: Lectura de un bit de la linea de datos
;*****************************************************
Lectura_Bit
        call   Shuffle                      ;inserta el bit en el arreglo de bytes
        call   ShuffleE                 ;inserta el bit que representa la calidad de señal
        call   Syndrome                 ; obtiene el Sindrome
        goto   Encontrar_Sincronia          ;compara sindromes

Lectura_Bit_Cont
        bcf         INTCON, INTF            ; bortar flag de RB0
```

```
        btfss   FlagSincronia, 0        ; si no hay flag de sincronia se sale del interrupto
        goto    Fin_Interruptos
        goto    Guardar_Bit

;**********. **********************************************
;Encontr a_Sincronia, sirve para obtener el sindrome y
;verificar si es el bloque A
;Usa: HiSynd y LowSynd
;*************************************************************
Encontrar_Sincronia:
        movlw   0xF6                    ; sindrome = A?
        xorwf HiSynd, w                 ; compara los 8 bits mas significativos
        btfss   STATUS, Z
        goto    Lectura_Bit_Cont        ; sino es se va a ver si no es bloque B
        clrw
        xorwf LowSynd, w                ; compara los dos bits menos significativos
        btfss   STATUS, Z
        goto    Lectura_Bit_Cont        ; sino es el sindrome regresa

Encontrar_BloqueB:

        movf    StoReg4,w       ;mueve los registros a los temporales
        movwf   StoReg1X
        movf    StoReg5,w
        movwf   StoReg2X
        movf    StoReg6,w
        movwf   StoReg3X
        movf    StoReg7,w
        movwf   StoReg4X

        bcf     STATUS,C        ;ro a los registros para que queden los
        rlf     StoReg4X, f     ;siguientes bytes en posicion para
        rlf     StoReg3X, f     ;obtener el sindrome
        rlf     StoReg2X, f
        rlf     StoReg1X, f

        bcf     STATUS,C
        rlf     StoReg4X, f
        rlf     StoReg3X, f
        rlf     StoReg2X, f
        rlf     StoReg1X, f

        call    SyndromeB               ;obtiene el sindrome

        movlw   0xF5                    ; sindrome = B?
        xorwf HiSynd, w                 ; compara los 8 bits mas significativos
        btfss   STATUS, Z
        goto    Lectura_Bit_Cont        ;sino es se va a ver si no es bloque B
        clrw
        xorwf LowSynd, w                ; compara los dos bits menos significativos
        btfss   STATUS, Z
        goto    Lectura_Bit_Cont        ; sino es el sindrome regresa

        clrf    FlagSincronia
        bsf     FlagSincronia,0                 ;pone que encontro bloque A

        movlw   0x20                    ;se indica la direccion de los bytes para guardar la info
        movwf   FSR
        call    Store_data
        movlw   .26
        movwf   Count_26                ;Pone el numero de bits a leer para los siguientes bloques

        goto    Fin_Interruptos         ;Para salir de la funcion de interruptos

;*************************************************************
; Guardar_Bit: lleva la cuenta de los bits guardados
; y si ya se recogio un bloque lo manda a guardar
;usa: Store_data y utiliza espacios de memoria 0x24-0x31
;*************************************************************
Guardar_Bit
```

```
        decfsz Count_26, f      ;decrementa contador de bits para completar bloque
                                ;se revisa si ya se juntaron 26 bits
        goto   Fin_Interruptos  ;sino se esperan los demas bits
        btfss  FlagSincronia,1  ;revisa si ya se tiene el bloque B
        goto   BlockB
        goto   Block1

Block1
        movlw   0x28            ;pone en FSR direccion de block1
        movwf   FSR
        call    Store_data
        bsf     FlagSincronia, 2    ;setea la bandera de recuperacion block1
        goto    R26aCount

BlockB:

        movlw   0x24            ;se indica la direccion de los bytes para guardar la info.
        movwf   FSR
        call    Store_data
        bsf     FlagSincronia, 1    ;indica que ya se recibio bloque B


R26aCount:
        bsf     FlagSincronia, 4    ;indica que ya hay bloque AyB o 1 para mandar por serial
        movlw   .26
        movwf   Count_26            ;Pone el numero de bits a leer para los siguientes bloques
        goto    Fin_Interruptos

;*************************************************************
;Store_data: Guarda los byte desde la posicion indicada
;*************************************************************
Store_data
        movf  StoReg1, w  ; guarda los datos en Reg1A-B SyndA-B
        movwf   INDF

        incf    FSR,f
        movf    StoReg2, w
        movwf   INDF

        incf    FSR,f
        movf  StoReg3, w
        movwf   INDF

        incf    FSR,f
        movlw   b'11000000'     ;hace un and para quitar cualquier basura en el synd
        andwf StoReg4,w

        movwf   INDF

        call    FSuma_Error
        movf  Suma_Error,w      ;agrega suma de error (no.<26) con un OR
        iorwf  INDF,f

        return

;*************************************************************
;FSuma_Error: Suma las faltas de señal en el byte que se va a mandar 3A-3D
;variables: FSR_E, 0x3A-0x3C, Suma_Error
;*************************************************************
FSuma_Error
        movf  FSR,w
        movwf   FSR_E               ;temporal para guardar FSR

        clrf    Suma_Error ;limpia variable

        movlw   0x3A
        movwf   FSR

FSuma_C
        btfsc  INDF,0
```

```
        incf    Suma_Error, f       ;suma uno si existe una falta de señal RDS
        btfsc   INDF,1
        incf    Suma_Error, f
        btfsc   INDF,2
        incf    Suma_Error, f
        btfsc   INDF,3
        incf    Suma_Error, f
        btfsc   INDF,4
        incf    Suma_Error, f
        btfsc   INDF,5
        incf    Suma_Error, f
        btfsc   INDF,6
        incf    Suma_Error, f
        btfsc   INDF,7
        incf    Suma_Error, f

        incf    FSR,f
        movlw   0x3D
        xorwf FSR,w
        btfss   STATUS, Z
        goto    FSuma_C

        btfsc   INDF,6
        incf    Suma_Error, f
        btfsc   INDF,7
        incf    Suma_Error, f

        movf  FSR_E,w
        movwf   FSR

        btfsc  PORTB, rds_sgn           ; agrega bit de señal RDS
        bsf             Suma_Error, 5
        return

;*****************************************************************
;Recepcion serial: Compara el valor que viene del serial
;para ver a que comando pertenece
;*****************************************************************
Recepcion_Serial:
        banksel     PORTA
        movf  RCREG, w
        movwf       Dato_Serial

        movlw       0x01                ;Revisa si es comando1(transmitir por serial) o comando2(dejar de transmitir por serial)
        xorwf Dato_Serial,w
        btfsc  STATUS,Z
        goto   Recepcion_Activar

        movlw       0x02
        xorwf Dato_Serial,w
        btfsc  STATUS,Z
        goto   Recepcion_Activar
        goto   Recepcion_Dato

Recepcion_Activar
        movf  Dato_Serial,w
        movwf       Estado              ;Estado indica si hay que mandarle datos a la compu o no.
            bsf             FlagSincronia,0
        bsf             FlagSincronia,1
        goto   Fin_Interruptos


Recepcion_Dato
        movf  Dato_Serial,w
        movwf       PORTD
        goto   Fin_Interruptos

;*****************************************************************
;Syndrome:multiplica los 26 bits con la matriz de paridad y
;recupera el síndrome, esta función fue diseñada por Lieven Hollevoet
```

```
;*****************************************************************
Syndrome:

        movf  StoReg4, w
        movwf       StoReg4X
        movf  StoReg3, w
        movwf       StoReg3X
        movf  StoReg2, w
        movwf       StoReg2X
        movf  StoReg1, w
        movwf       StoReg1X

SyndromeB:
        movlw       0x1A
        movwf       CountReg
        clrf   HiSynd                   ;El síndrome se guarda en Hi y LowSind
        clrf   LowSynd


loops
        rlf    StoReg4X, F              ;se rotan los registros
        rlf    StoReg3X, F
        rlf    StoReg2X, F
        rlf    StoReg1X, F
        btfss  STATUS, C
        goto   loopx
        movf   CountReg, w
        call   MatrixHi          ;se obtiene el valor de la matriz
        xorwf       HiSynd, F
        movf   CountReg, w
        call   MatrixLo          ;se obtiene el valor de la matriz
        xorwf       LowSynd, F
        decfsz      CountReg, F ;se decrementa contador
        goto   loops             ; repetir 26 veces
        return
loopx
        movlw       0x03
        call   Delay
        decfsz      CountReg, F
        goto   loops
        return


;*****************************************************************
;Delay: deja pasa cierto tiempo
;*****************************************************************
Delay
        movwf       DelayReg
del
        decfsz      DelayReg, F
        goto   del
        return


;*****************************************************************
;MatrixHi:
ontiene los 8 bits de la matriz de paridad
;*****************************************************************
MatrixHi
        addwf       PCL, F
        nop
        retlw  0xC6
        retlw  0xE3
        retlw  0xA9
        retlw  0x3D
        retlw  0x7B
        retlw  0xF7
        retlw  0x80
        retlw  0x6E
        retlw  0xDD
```

```
        retlw  0xD5
        retlw  0xC4
        retlw  0xE7
        retlw  0xA1
        retlw  0x2D
        retlw  0x5B
        retlw  0xB7
        retlw  0x00
        retlw  0x00
        retlw  0x01
        retlw  0x02
        retlw  0x04
        retlw  0x08
        retlw  0x10
        retlw  0x20
        retlw  0x40
        retlw  0x80

;*****************************************************
;MatrixLo: contiene los 2 bits mas bajos de la matriz de paridad
;*****************************************************
;
MatrixLo
        addwf   PCL, F
        nop
        retlw  0xC0
        retlw  0xC0
        retlw  0xC0
        retlw  0xC0
        retlw  0x80
        retlw  0x00
        retlw  0x40
        retlw  0xC0
        retlw  0x80
        retlw  0x40
        retlw  0xC0
        retlw  0xC0
        retlw  0xC0
        retlw  0xC0
        retlw  0x80
        retlw  0x00
        retlw  0x40
        retlw  0x80
        retlw  0x00
        retlw  0x00
        retlw  0x00
        retlw  0x00
        retlw  0x00
        retlw  0x00
        retlw  0x00

;*****************************************************
;GetBit: espera un dato de entrada y lo guarda en el arreglo
;de bytes
;Usa: Shuffle, Syndrome
;*****************************************************
;
GetBit
        btfss   PORTB, clk
        goto    GetBit                  ; espera el flanco de subida del reloj
        call    Shuffle                 ; rota todos los bits en los bytes
        call    ShuffleE                ; guarda calidad de señal del bit
        call    Syndrome                ; Calcula el Sindrome
        return

;*****************************************************
;Shuffle: rota los bytes donde se guardan los 20 bits de manera
;que el ultimo bit que entra es el menos significativo
;*****************************************************
;
Shuffle
```

```
        bcf     STATUS, C
        btfsc   PORTB, data_in  ; bit nuevo
        goto    Shuffle_1
        goto    Shuffle_0

Shuffle_0
        bcf     PORTB,1         ;muestra los bits que se van guardando
        goto    Shuffle_R
Shuffle_1
        bsf     PORTB,1
        bsf     STATUS,  C      ;si el bit de entrada es uno setea el carry
Shuffle_R
        rlf     StoReg7, f      ;se rotan los registros
        rlf     StoReg6, f
        rlf     StoReg5, f
        rlf     StoReg4, f
        rlf     StoReg3, f
        rlf     StoReg2, f
        rlf     StoReg1, f
        return

;*****************************************************
;ShuffleE: rota los bytes donde se guardan los 26 bits de señal de manera
;que el ultimo bit que entra es el menos significativo
;*****************************************************
ShuffleE
        banksel PORTB
        bcf     STATUS, C
        btfsc   PORTB, rds_sgn  ;revisa si hay señal RDS
        goto    ShuffleE_1
        btfss   PORTB, qly_sgn  ;revisa calidad de señal
        goto    ShuffleE_1
        goto    ShuffleE_R

ShuffleE_1
        bsf     STATUS,  C

ShuffleE_R
        rlf     StoReg7E, f     ; rotar bit
        rlf     StoReg6E, f
        rlf     StoReg5E, f
        rlf     StoReg4E, f
        rlf     StoReg3E, f
        rlf     StoReg2E, f
        rlf     StoReg1E, f
        return


;*****************************************************
;Mandar_Bloques: Manda 1 o 2 bloques a la computadora
;*****************************************************
Mandar_Bloques:
        banksel TXREG       ;banco0

        btfss   Estado,0        ;si estado = 0x01 manda datos si no, no.
        goto    Mandar_Fin

        movlw   0x20                    ;Posicion bloque A y B
        movwf   FSR
        movlw   0x28
        movwf   FSR_Fin

        btfss   FlagSincronia,2
        goto    Mandar_Bloques_Loop
        movlw   0x28                    ;Posicion bloque1
        movwf   FSR
        movlw   0x2C
        movwf   FSR_Fin

Mandar_Bloques_Loop
```

```
            movf  INDF,w
            call  Serial_Tx

            incf  FSR,f
            movf  FSR_Fin,w                 ;Compara con la direccion Final
            xorwf FSR,w
            btfss STATUS,Z
            goto  Mandar_Bloques_Loop

Mandar_Fin
            banksel     PORTB
            bcf         FlagSincronia,4     ;Limpia bandera de mandar a serial
            bcf         FlagSincronia,2     ;Limpia bandera de que se recibio bloque 1
            bcf         PORTB,2                 ;apaga bit de sincronia bloque A
            return


;*******************************************************
;Serial_Tx: Manda dato en Wreg por serial
;*******************************************************
Serial_Tx
            movwf       TXREG

SerialTx_Wait_Flag
            nop
            btfss  PIR1, TXIF
            goto   SerialTx_Wait_Flag
            return


;*******************************************************
;I_Serial  INICIACION PUERTO SERIAL
;*******************************************************
I_Serial:
      banksel     SPBRG
     movlw   .129                ;para 20MHz y  9600 baud rate
     movwf   SPBRG
     bcf    TXSTA,SYNC                  ;activado  modo asincrono
     bsf    TXSTA, BRGH                 ;BRGH high
     bsf    PIE1, RCIE              ;interrupto recepcion
     bcf    PIE1, TXIE              ;interrupto transmision
     bsf    TXSTA,TXEN                  ;transmision activada
            bsf         TRISC,6                 ;como entrada los pines RX/TX
            bsf         TRISC,7

      banksel     RCSTA
     bsf    RCSTA, SPEN                 ;puerto serial activado
     bsf    RCSTA, CREN                 ;recepcion activada

      return


;*******************************************************
;*****************MAIN**********************************
;*******************************************************
main

      banksel TRISA
      movlw       B'11111111'
      movwf       TRISA                   ;puertos como entradas
      movwf       TRISB                   ;clock y datos
      bcf         TRISB, 1            ;datos recibidos
      bcf         TRISB, 2            ;sincronia
;     bcf         TRISB, rds_sgn          ;datos recibidos
;     bcf         TRISB, qdy_sgn          ;sincronia
      clrf   TRISD                    ;salida de digitales para sincronizacion
      bsf         OPTION_REG, INTEDG      ;se activa con flanco de subida     call    I_Serial
```

```
        movlw   0x20            ; Se pone al principio memoria RAM
        movwf   FSR             ; banco 0

clean_RAM
        clrf    INDF            ; limpia todas las posiciones
        incf    FSR, f          ; entre 0x20-0x80
        btfss   FSR, 7
        goto    clean_RAM

        call    I_Serial        ;inicializa puerto serial

        movlw   0x02
        movwf   Estado          ;indica que la compu no quiere datos todavia

        banksel PORTA
        clrf    PORTD
        clrf    PORTB
        movlw   .100            ;pone un valor inicial al puerto D
;       movwf   N_Voltaje
        movwf   PORTD

        bsf     INTCON, GIE     ;global activado
    bsf     INTCON, PEIE        ;interruptos perifericos activados
        bsf     INTCON, INTE    ;activa interrupto RB0

main_loop
        btfsc   FlagSincronia, 4                ;se queda esperando datos
        call    Mandar_Bloques
        nop
        goto    main_loop

            END
```

F. Código en lenguaje Borland C++ Builder de la aplicación de transmisión

```
//Programa UEnvio.cpp requiere la utilizacion de Serialunit.h por las funciones
//de manejo puerto serial
//Entrada es un archivo el cual codifica y empaqueta
//Envia bytes FF para avisarle al PIC que la transmision termino.

//Requerimientos para el codificador
//---------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include <dos.h>
#include <vcl.h>
#pragma hdrstop

#include "UEnvioFinal.h"
#include "Serialunit2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CSPIN"
#pragma resource "*.dfm"

//Variables de codificador
//---------------------------------------------
#define MAXRAND LONG_MAX   // for random number generation
#define G    0x5b9       // x^{10} + x^8 + x^7 + x^5 + x^4 + x^3 + 1
#define P    0x31b       // x^9 + x^8 + x^4 + x^3 + x + 1
#define K21  32767       // 2^{k-1} - 1
#define K2   32768       // 2^{k-1}
#define FLAG  1024       // 2^{n-k}
#define NK2    512       // 2^{n-k-1}
#define NK   0x7ff       // n-k+1 ones
#define TRAP  0x01f      // trapping pattern = 0000011111
#define RED2    9        // n-k-2
#define DBA  0x0000000C   //Desplazamiento bloque A
#define DBB  0x00000198
#define DBC  0x00000168
#define DBCP 0x00000350
#define DBD  0x000001B4

//Variables globales
//---------------------------------------------
int finrepeticiones;    //Variable que indica el # de veces a repetir la
                        //el archivo
int BpP;                //Bytes por paquete
int BpA;                //Bytes por archivo
int PpA;                //Paquetes que se generan por archivo
float TpA;              //Tiempo de transmision por Archivo
int NActualizacion;     //Numero de actualizacion, mas numero de archivo
int transmit0;          //si es igual a cero carga el buffer con archivo indicado
int NArchivo;           //Numero de archivo
int CArchivo;           //Contador de archivos que se transmite
int ContNRepG;          //Contador de repeticiones por grupo de archivos

//-------Funciones Codificador---------------------------------------------
int FEncoder(int M);        //Funcion que obtiene los bits de correccion de errores
//Esta function fue diseñada por Morelos-Zaragoza
//-------Funciones Programa------------------------------------------------
int FFreqtoInt(void);               //Convierte frecuencia de string a entero
int FMensaje(int byte1, int byte2); //Une los bytes en una palabra
```

```
void FPaquetes(int CArchivo);       //Crea archivo codificado del archivo CArchivo
void FTransmit(void);               //Transmite el archivo al PIC
void FEnviar(int status);           //Manda comando al PIC
void FCalculoP(int bpa);            //regresa paquetes por Archivo y tiempo de transmision
void FBorrar(int opcion);           //borra valores pantalla

//--------Funciones puerto serial-----------------------------------------
extern int FSerial (int NCom);          //Abre puerto serial
extern void FManda(char *texto, int numbytes);  //Manda arreglo por serial
extern int  FRecibe (char* dato, int numbytes); //Recibe arreglo del serial
extern void FCierraSerial(void);                //Cierra puerto serial
//------------------------------------------------------------------------
//------------------------------------------------------------------------

TFEnvioFM *FEnvioFM;
__fastcall TFEnvioFM::TFEnvioFM(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------------------------
// FCalculoP: Funcion que calcula el numero de paquetes y tiempo de transmision por archivo
//---------------------------------------------------------------------------

void FCalculoP(int bpa){
    div_t x,

    x = div(bpa,2);
    if(x.rem>0){        //si es archivo impar le suma uno al numero de bytes
      bpa = bpa + 1;
      BpA = BpA + 1;
    }
    x = div(bpa,BpP);   //division para encontrar numero de PpA
    PpA = x.quot;
    if(x.rem > 0){      //si hay residuo suma un paquete
      PpA = PpA+1;
    }
    FEnvioFM->LPpA->Caption = PpA;              //despliega valores en pantalla
    TpA = BpA + 8*PpA;
    TpA =  ((TpA*8+TpA*5)/1187)/60;             //tiempo en minutos
    FEnvioFM->LTpA->Caption = FloatToStrF(TpA,ffFixed,7,3);
}


//---------------------------------------------------------------------------
// FMensaje: Funcion que une a dos bytes en un entero de 16 bits para poder mandarlo
// a la funcion que obtiene el Sindrome
//---------------------------------------------------------------------------

int FMensaje(int byte1, int byte2){
    int byte12;
    int temp;

     byte12 = byte1;
     byte12 = byte12 & 0x000000FF;
     byte12 = byte12 << 8;
     temp = byte2 &  0x000000FF;
     byte12 = byte12 | temp;                //Union de dos bytes para tener palabra de 16bits
     return byte12;
}

//---------------------------------------------------------------------------
// FTransmit: funcion encargada de transmitir los datos al PIC por
//tramas de numero especifico
//Variables:transmit0, el cual indica si se debe transmitir un archivo nuevo
//cuando es igual a 0
//Funciones: FManda() y FBorrar()
//---------------------------------------------------------------------------
void FTransmit(void){
 int iFileHandleT;          //puntero para el archivo
```

```
int iFileLengthT;            //Largo del archivo
static int iBytesReadT;      //# de bytes leidos
static char *pszBufferTt;    //puntero que guarda el principio del arreglo
static char *pszBufferT;     //puntero de buffer
static int ntransmision;     //numero de transmisiones archivo completo
static int pactual;          //Numero de bytes transmitidos
int trama;                   //Numeros de bytes que se pasaran al PIC
int i;

if(transmit0==2){
   delete [] pszBufferT;
}
if(transmit0 == 0){
  try
   {
//Lee txt a transmitir
   //iFileHandle = FileOpen(FEnvioFM->Pagina->Text, fmOpenRead);   //prueba manda archivo original
   iFileHandleT = FileOpen(FEnvioFM->Pagina2->Text, fmOpenRead);   //encontrar pointer
   iFileLengthT = FileSeek(iFileHandleT,0,2);            //pone el puntero al final
   FileSeek(iFileHandleT,0,0);
   pszBufferT = new char[iFileLengthT+4];              //Lee cuantos caracteres tiene HTML
   iBytesReadT = FileRead(iFileHandleT, pszBufferT, iFileLengthT); //HTML en Buffer
   FileClose(iFileHandleT);                            //Cierra archivo
   //Termina leer archivo

   pszBufferTt = pszBufferT;  //backup de puntero
   pactual = 0;               //posicion que se debe transmitir
   transmit0 = 1;             //pone transmit0 =1 para indicar que ya se cargo buffer
   ntransmision=0;            //pone a cero el numero de transmision

  //  FEnvioFM->Label->Visible=True;
   FEnvioFM->Progreso->Max = iBytesReadT;
   FBorrar(3);              //Inicializacion objetos forma
   FBorrar(1);
   FEnvioFM->StatusBar1->Panels->Items[7]->Text = iBytesReadT;
   FEnvioFM->StatusBar1->Panels->Items[1]->Text = FEnvioFM->ListaHTML->Items->Strings[CArchivo];
   }
   catch(...)
   {
   Application->MessageBox("File Error", "Error", MB_OK);
   }
}

    trama = 8;               //pasa 8 bytes que son 2 bloques ej. bloque A+2bytes y B+2bytes

    if (ntransmision==0){   //Si numero de transmision es igual a 0, muestra nombre de archivo
         FEnvioFM->StatusBar1->Panels->Items[3]->Text = ntransmision;  //en la barra de estado
         FEnvioFM->StatusBar1->Panels->Items[4]->Text = ContNRepG;
    }

    //revisa si los bytes que quedan son menor que la trama que se manda
    if (iBytesReadT-pactual<trama) trama = iBytesReadT - pactual;  //si es cambia el valor de trama

    FManda(pszBufferT, trama);    //Manda bytes por serial
    pactual = pactual + trama;    //aumenta contador
    if (iBytesReadT-pactual ==0){  //revisa si ya se transmitio todo el archivo
          ntransmision = ntransmision+1;
          FEnvioFM->StatusBar1->Panels->Items[3]->Text= ntransmision;
          FEnvioFM->StatusBar1->Panels->Items[4]->Text=ContNRepG;
          pszBufferT = pszBufferTt;
          pactual = 0;
          if(ntransmision==FEnvioFM->CNRepeticiones->Value){
               transmit0 = 0;       //Inicializa las variables para empezar de nuevo
               finrepeticiones=1;   //flag para avisar que se termino la tarea
               delete [] pszBufferT;
          }
               ;
    else
         pszBufferT =pszBufferT+trama;   //mueve el puntero a la siguiente trama
```

```
        FEnvioFM->StatusBar1->Panels->Items[6]->Text = pactual;
        FEnvioFM->Progreso->Position = pactual;
    }

    //---------------------------------------------------------------------------
    // FPaquetes: Funcion que crea el archivo ya codificado y con sindrome,
    //el archivo a codificar se encuentra en ListaHTML y se le debe pasar la posicion
    //que el archivo ocupa en esta lista (CArchivo)
    //---------------------------------------------------------------------------
    void FPaquetes(int CArchivo){
     int iFileHandle;     //puntero para el archivo
     int iFileLength;     //Largo del archivo
     int iBytesRead;      //# de bytes leidos

     char *pszBuffer;     //puntero de buffer
     char *pSindrome;     //puntero de buffer para el sindrome
     char *pAux;          //puntero auxiliar para colocar encabezado
     char *pEncabezado;   //puntero para los bytes que hacen el encabezado
     String Archivo;

     int Mensaje, Sindrome, SindromecharL, SindromecharH;
     int frecuencia, link;
     int npaquete, nbyte, epaquete, ebyte;   //contadores
     int temp, k, posicion                   //variables temporales
     int EncA, EncB, EncC, EncD;             //infirmacion que perteneee al encabezado

      try
      {
    //Lee txt a codificar
        Archivo = FEnvioFM->ListaHTML->Items->Strings[CArchivo];
        iFileHandle = FileOpen(Archivo, fmOpenRead);        //encontrar pointer
        iFileLength = FileSeek(iFileHandle,0,2);            //pone el puntero al final
        FileSeek(iFileHandle,0,0);
        pszBuffer = new char[iFileLength+2+26];             //Crea un buffer del tamaño del archivo
        pszBuffer[iFileLength+26] =0x20;                    //+ bytes informacion adicional
        pszBuffer[iFileLength+26+1] =0x20;                  //Limpia últimas dos posiciones

        Archivo = ExtractFileName(Archivo);                 //se obtiene el nombre del archivo
        frecuencia = FFreqtoInt();                          //agrega info. encabezado al buffer
        NArchivo = FEnvioFM->ListaHTML->Items->Count;       //Guarda la cantidad de archivos a transmitir
        FEnvioFM->TNLinks->Text = NArchivo;
        BpP = FEnvioFM->CBpP->Value;                        //Obtiene los bytes por paquete
        NActualizacion = FEnvioFM->CActualizacion->Value;   //Obtiene numero de actualizacion

        pAux = &pszBuffer[0];                               //puntero al principio del buffer

        strcpy(pAux, Archivo.c_str());                      //nombre de archivo Nombre*.*
        strcpy(pAux+10, FEnvioFM->TEstacion->Text.c_str()); //Estacion, Pais, Frecuencia
        strcpy(pAux+21, FEnvioFM->TPais->Text.c_str());
        pAux[24] = frecuencia;
        pAux[25] = NArchivo;


        iBytesRead = FileRead(iFileHandle, pszBuffer+26, iFileLength); //Archivo se copia en Buffer
        FileClose(iFileHandle);                             //Cierra archivo
    //Termina

    //Crea archivo donde va a estar la codificacion, el nombre debe encontrarse en textbox Pagina2
        iFileHandle = FileCreate(FEnvioFM->Pagina2->Text);

        BpA = iBytesRead+26;
        FCalculoP(BpA);                 //Calcula PpA = Paquetes por Archivo y tiempo de transmision

        ebyte = 0;
        epaquete = 0;

    //Ciclo para agregar el sindrome a cada 2 bytes (16 bits)
    //Y el encabezado a cada paquete

        for (k=0;k<BpA;k=k+2){
```

```
pSindrome = new char[2];

if(cbyte == 0){   //Si cbyte = 0 se agrega un encabezado

    pEncabezado = new char[16];

    if(BpA-(BpP * cpaquete)< BpP) BpP = BpA-(BpP * cpaquete); //si el nbytes restantes < BpP entonces se modifica BpP
    EncA = BpP;                          //Numero de bytes por paquete
    pEncabezado[0] = (EncA>>8) & 0x000000FF;      //Parte alta
    pEncabezado[1] = EncA & 0x000000FF;           //Parte baja
    Sindrome = FEncoder(EncA)^DBA;
    Sindrome = (Sindrome << 6) & 0x0000FFC0;      //Se corre para que queden los 8 MSB en SH y 2LSB en SL
    pEncabezado[2] = (Sindrome >> 8)& 0x000000FF;;
    pEncabezado[3] = Sindrome & 0x000000FF;       //Se regresan a forma de byte

    EncB = PpA;                          //Numero de paquetes por archivo
    pEncabezado[4] = (EncB>>8) & 0x000000FF;      //Parte alta
    pEncabezado[5] = EncB & 0x000000FF;           //Parte baja
    Sindrome = FEncoder(EncB)^DBB;
    Sindrome = (Sindrome << 6) & 0x0000FFC0;      //Se corre para que queden los 8 MSB en SH y 2LSB en SL
    pEncabezado[6] = (Sindrome >> 8)& 0x000000FF;;
    pEncabezado[7] = Sindrome & 0x000000FF;       //Se regresan a forma de byte

    EncC = cpaquete;                     //Numero de Paquete
    pEncabezado[8] = (EncC>>8) & 0x000000FF;      //Parte alta
    pEncabezado[9] = EncC & 0x000000FF;           //Parte baja
    Sindrome = FEncoder(EncC)^DBC;
    Sindrome = (Sindrome << 6) & 0x0000FFC0;      //Se corre para que queden los 8 MSB en SH y 2LSB en SL
    pEncabezado[10] = (Sindrome >> 8)& 0x000000FF;;
    pEncabezado[11] = Sindrome & 0x000000FF;      //Se regresan a forma de byte

    EncD = ((NActualizacion << 8)& 0x0000FF00) | CArchivo;   //Numero de actualizacion-agrupo y Numero de
archivo=carchivo
    pEncabezado[12] = (EncD>>8) & 0x000000FF;     //Parte alta
    pEncabezado[13] = EncD & 0x000000FF;          //Parte baja
    Sindrome = FEncoder(EncD)^DBD;
    Sindrome = (Sindrome << 6) & 0x0000FFC0;      //Se corre para que queden los 8 MSB en SH y 2LSB en SL
    pEncabezado[14] = (Sindrome >> 8)& 0x000000FF;
    pEncabezado[15] = Sindrome & 0x000000FF;      //Se regresan a forma de byte

    FileWrite(iFileHandle, pEncabezado, 16);      //Se graban al archivo el encabezado de cada paquete
    delete [] pEncabezado;
}

Mensaje = 0;
Mensaje = FMensaje(pszBuffer[k], pszBuffer[k+1]);
Sindrome = FEncoder[Mensaje];          //Se encuentra el sindrome con FEncoder

Sindrome = (Sindrome << 6) & 0x0000FFC0; //Se corre para que queden los 8 MSB en SH y 2LSB en SL

SindromecharL= Sindrome & 0x000000FF;          //Se regresan a forma de byte
SindromecharH= (Sindrome >> 8)& 0x000000FF;;

pSindrome[0]=SindromecharH;                    //Se guardan en el arreglo
pSindrome[1]=SindromecharL;

FileWrite(iFileHandle, pszBuffer+k, 2);   //Se graban al archivo info
FileWrite(iFileHandle, pSindrome, 2);     //Sindrome

delete [] pSindrome;                   //Borra arreglo

cbyte = cbyte + 2;

if(cbyte == BpP){        //si cbyte llega a los respectivos bytes por paquete
    cbyte = 0;       //se pone a cero cbyte y se suma una a la cuenta de
    cpaquete = cpaquete + 1;   //paquetes
}

}
FEnvioFM->Progreso->Max=BpA;
```

```
    FileClose(iFileHandle);                    //Cerrar archivo
    delete [] pszBuffer;
   }
   catch(...)        //si existe algun error se despliega un mensaje
   {
    Application->MessageBox("Error en codificación de archivo. ","Error", MB_OK);
   }
}



//---------------------------------------------------------------------------
//FEncoder: Funcion que obtiene los 10 bits de redundancia (residuo de la division).
//Divide el mensaje dentro de G (polinomio caracteristico), el residuo
//se guarda en S.  La division se hace bit a bit mediante rotacion de registros.
//---------------------------------------------------------------------------
 int FEncoder(int M) {
 int S, j, i, bit;

 S = 0;
 j = K2;
 for (i=15; i>=0; i--) {
  bit = ( (j & M) >> i ) & 0x01;
  j = j>>1;
  if (bit)
   S = ((S<<1)^FLAG) & NK;
  else
   S = (S<<1) & NK;
  if (S & FLAG)
   S = (S^G) & NK;
  }

 return S;
}

//---------------------------------------------------------------------------
//FEnviar: Funcion que maneja los distintos comandos que se le mandan al PIC
//asi como inhibitar y deshabilitar las opciones del programa segun el estado
//en el cual este el programa.
//status 0= Inicio de timers y manda comando de inicio al PIC
//status 1= Obtiene archivo codificado, deshabilita opciones de la forma
//status 3= Manda comando de finalizacion de transmision y habilita las opciones
//de la forma
//---------------------------------------------------------------------------
void FEnviar(int status){

    char comando[1];                                //comando de inicio
    char comandoF[8]={0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}; //comando de finalizacion

 switch(status){

    case 0:
        if(FSerial(FEnvioFM->CCOM->Value)==-1){          //Abre el serial
            FEnviar(2);
            break;
        }
        comando[0]='I';
        FManda (comando,1);
        FEnvioFM->Color = clSilver;          //Cambia color de pantalla
        FEnvioFM->Timer1->Enabled = True;        //Activa timer para esperar dato del PIC
        FEnvioFM->Timer3->Enabled = True;
        break;

    case 1:
        finrepeticiones = 0;
        FPaquetes(CArchivo);
        FEnvioFM->Progreso->Visible = True;
        transmit0 = 0;                  //Inicializa variable de FTransmit para que
                                        //cargue nuevo archivo
        FEnvioFM->BCargar->Enable = False;    //Deshabilita la opcion de cargar datos
```

```
        FEnvioFM->CCOM->Enabled=False;
        FEnvioFM->CBpP->Enabled=False;
        FEnvioFM->CActualizacion->Enabled=False
        FEnvioFM->CNRepG->Enabled=False;
        FEnvioFM->CNRepeticiones->Enabled=False;
        FEnvioFM->TEstacion->Enabled=False;
        FEnvioFM->TFrecuencia->Enabled=False;
        FEnvioFM->TPais->Enabled=False;
        FEnvioFM->Pagina2->Enabled=False;
        FEnvioFM->TNLinks->Enabled=False;
        FEnvioFM->BBrowse->Enabled=False;
        FEnvioFM->Button3->Enabled=False;
        FEnvioFM->BCargar->Enabled=False;
        FEnvioFM->BBrowse->Enabled=False;
        FEnvioFM->TxIndefinido->Enabled=False;
        FEnvioFM->ToolButton1->Enabled=False;
        FEnvioFM->ToolButton2->Enabled=False;
        break;

    case 2:
        FManda (comandoF, 8);        //Avisa al PIC que se acabo la transmision
        FCierraSerial();
        finrepeticiones = 0;
        FEnvioFM->Timer1->Enabled = False;       //Apaga timer
        FEnvioFM->Timer3->Enabled = False;
        transmit0 = 0;
        ContNRepG = 0;
        CArchivo = 0;
        FEnvioFM->Progreso->Position = 0;
        FEnvioFM->BCargar->Enabled = True;       //Habilita boton de cargar
        FEnvioFM->BEnviar->Enabled = True;
        FEnvioFM->Color = clAppWorkSpace;        //Regresa el color de la pantalla

        FEnvioFM->CCOM->Enabled=True;
        FEnvioFM->CBpP->Enabled=True;
        FEnvioFM->CActualizacion->Enabled=True;
        FEnvioFM->CNRepG->Enabled=True;
        FEnvioFM->CNRepeticiones->Enabled=True;
        FEnvioFM->TEstacion->Enabled=True;
        FEnvioFM->TFrecuencia->Enabled=True;
        FEnvioFM->TPais->Enabled=True;
        FEnvioFM->Pagina2->Enabled=True;
        FEnvioFM->TNLinks->Enabled=True;
        FEnvioFM->BBrowse->Enabled=True;
        FEnvioFM->Button3->Enabled=True;
        FEnvioFM->BCargar->Enabled=True;
        FEnvioFM->BBrowse->Enabled=True;
        FEnvioFM->TxIndefinido->Enabled=True;
        FEnvioFM->ToolButton1->Enabled=True;
        FEnvioFM->ToolButton2->Enabled=True;

        FBorrar(1);
        break;

    default:
        break;
  }
}

//---------------------------------------------------------------------------
//FFregtoInt:Cambia la frecuencia a integer: 0-210 corresponde 88-109 de 0.1-0.1
//---------------------------------------------------------------------------

 int FFregtoInt(void){
    float fstring;

    fstring = StrToFloat(FEnvioFM->TFrecuencia->Text);
    fstring = (fstring-88)/0.1;
    fstring =ceil(fstring)
    return fstring;
```

```
}

//---------------------------------------------------------------------------
// FBorrar: Borra ciertos valores de la pantalla segun la opcion
//---------------------------------------------------------------------------

void FBorrar(int opcion){
    switch(opcion){
      case 0:          //Borrar Lista
        FEnvioFM->ListaHTML->Clear();
        break;
      case 1:          //Borrar datos Toolbar
        FEnvioFM->StatusBar1->Panels->Items[1]->Text = " ";
        FEnvioFM->StatusBar1->Panels->Items[3]->Text = " ";
        FEnvioFM->StatusBar1->Panels->Items[4]->Text = " ";
        FEnvioFM->StatusBar1->Panels->Items[6]->Text = " ";
        FEnvioFM->StatusBar1->Panels->Items[7]->Text = " ";
        break;
      case 2:          //Borra datos paquete
        FEnvioFM->CActualizacion->Value  = 0;
        FEnvioFM->LPpA->Caption  = 0;
        FEnvioFM->LTpA->Caption  = 0;
        FEnvioFM->LHora->Caption  = 0;
        break;
      case 3:          //Borra barra de progreso
        FEnvioFM->Progreso->Position = 0;
        break;
      default:
        break;
    }

}


//---------------------------------------------------------------------------
//          Eventos de la aplicacion
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//FormCreate:Cuando se crea la aplicacion, se actualiza el reloj
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::FormCreate(TObject *Sender)
{
    StatusBar1->Panels->Items[8]->Text =FormatDateTime("d/m/yyyy ' - 'hh:mm AM/PM", Now());
}

//---------------------------------------------------------------------------
//Timer1:  Es el encargado de recibir el dato que el PIC manda para avisar
//a la computadora que debe mandar mas datos y lleva la cuenta de las
//repeticiones que se llevan de los archivos.
//Funciones que usa: FRecibe(), FTransmit(), FEnviar()
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::Timer1Timer(TObject *Sender)
{
    int cuenta;
    char comando1[1];

    cuenta = FRecibe(comando1,1);    //Si recibe quiere decir que el PIC esta listo para recibir otra trama

    if(cuenta==1){
        if(finrepeticiones == 0){  //finrepeticiones==0 indica que todavia no se ha terminado de transmitir el archivo
            FEnvioFM->BEnviar->Enabled = False;
            FTransmit();
            Timer3->Enabled=false;
            Timer3->Enabled=true;
        }
        else{
            CArchivo = CArchivo+1;
            if(NArchivo!=CArchivo)
                FEnviar(1);     //Carga siguiente archivo y pone transmit0=0 para que FTransmit cargue el nuevo archivo
```

```
            else{
                ContNRepG = ContNRepG +1;
                CArchivo = 0;

                if(TxIndefinido->Checked || ContNRepG!=CNRepG->Value)
                    FEnviar(1);
                else
                    FEnviar(2);

            }

        }

    }   //se cierra if mayor
}

//---------------------------------------------------------------------------
//Timer2: Encargado de refrescar la hora en la barra de estado
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::Timer2Timer(TObject *Sender)
{
    StatusBar1->Panels->Items[8]->Text =FormatDateTime("d/ m/ yyyy ' - ' hh:mm AM/PM", Now());
}

//---------------------------------------------------------------------------
//Timer3: En caso de que por alguna razon se trate de usar el comando mandar sin
//estar encendido el PIC o de que este no responda Timer 3 lo saca del modo
//de transmision y reinicializa variables para que el programa no se quede esperando
//datos del PIC
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::Timer3Timer(TObject *Sender)
{
    ShowMessage("Módulo no responde.");

    CArchivo=0;          //Inicializa contadores
    finrepeticiones=0;
    BEnviar->Enabled=True;
    ContNRepG = 0;
    Timer1->Enabled=false;  //Deshabilita Timers
    Timer3->Enabled=false;
    FEnviar(2);          //Vuelve a activar todos las opciones
    if(transmit0==1){
        transmit0 = 2;
        FTransmit();
    }
}

//---------------------------------------------------------------------------
//ACargar: Funcion que sirve para tener un preliminar del archivo
//y crear el archivo codificado
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::ACargarExecute(TObject *Sender)
{
    int seleccion;
    seleccion = ListaHTML->ItemIndex;
    if(seleccion == -1)
        ShowMessage("No existe archivo que cargar.");
    else
        FPaquetes(seleccion);
}

//---------------------------------------------------------------------------
//AEnviar: Evento que sirve para iniciar la transmision, inicializa los contadores
//y llama a la funcion FEnviar
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::AEnviarExecute(TObject *Sender)
{
    if(Timer1->Enabled == False){
        NArchivo = ListaHTML->Items->Count;   //Guarda la cantidad de archivos a transmitir
        CArchivo = 0;                         //Inicializa contadores
```

```
        ContNRepG = 0;
        if(NArchivo==0 || TEstacion->Text=="" || TFrecuencia->Text=="" || TPais->Text==""){
            if(NArchivo==0)
                ShowMessage("No hay archivos que transmitir."),
            else
                ShowMessage("No ha llenado todas las casillas.");
        }
        else{
            Timer3->Enabled=true;     //Activa timer de seguridad
            FEnviar(1);               //Manda comando al PIC
            FEnviar(0);               //Manda a codificar primer archivo
        }
    }
}

//---------------------------------------------------------------------------
//APara: Esta encargado de terminar la transmision, modifica los valores de los
//contadores para parar la transmision
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::AParaTxExecute(TObject *Sender)
{
    TxIndefinido->Checked=False;
    CArchivo=NArchivo-1;     //Para no transmitir mas archivos
    finrepeticiones=1;       //Para indicar que no mas repeticiones
    BEnviar->Enabled=True;
    ContNRepG = CNRepG->Value-1;

}

//---------------------------------------------------------------------------
//AAgregar: Boton que sirve para abrir un nuevo archivo y agregarlo a la lista
//de archivos a transmitir
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::AAgregarExecute(TObject *Sender)
{
   if(OPagina->Execute()){   //Abre dialogo de abrir archivo
    Pagina->Text = OPagina->FileName;   //se copia el nombre del archivo a Pagina
    StatusBar1->Panels->Items[1]->Text = Pagina->Text;   //y a la barra de estado

    if(ListaHTML->Items->IndexOf(Pagina->Text)!= -1)     //si ya esta en la lista se da aviso
        ShowMessage(Pagina->Text + " ya está en la lista ");
    else
        ListaHTML->Items->Add(Pagina->Text);   //sino se agrega

    TNLinks->Text = ListaHTML->Items->Count;       //y se aumenta el contador
   }
}

//---------------------------------------------------------------------------
//ARemover: Quita el archivo seleccionado de la lista
//---------------------------------------------------------------------------

void __fastcall TFEnvioFM::ARemoverExecute(TObject *Sender)
{
    int seleccion;

     seleccion = ListaHTML->ItemIndex;   //obtiene cual esta seleccionado
     ListaHTML->Items->Delete(seleccion); //lo borra
     TNLinks->Text = ListaHTML->Items->Count;   //y disminuye el contador
}

//---------------------------------------------------------------------------
//TxIndefinido: Seleccionado sirve para transmitir los archivos idefinidamente
//---------------------------------------------------------------------------

void __fastcall TFEnvioFM::TxIndefinidoClick(TObject *Sender)
{
    ContNRepG = 0;
    if (TxIndefinido->Checked)
        CNRepG->Enabled = False;      //se deshabilita la casilla de
```

```
        else                    //repeticiones por grupo
            CNRepG->Enabled = True;
}

//---------------------------------------------------------------------------
//Resetear:Boton que sirve para borrar todo lo que se ha ingresado
//Utiliza FBorrar().
//---------------------------------------------------------------------------

void __fastcall TFEnvioFM::Resetear1Click(TObject *Sender)
{
      FBorrar(1);
      FBorrar(2);
      FBorrar(3);
      BEnviar->Enabled=True;
      CArchivo = 0;
}

//---------------------------------------------------------------------------
//Salir: Opcion del menu para salir del programa
//---------------------------------------------------------------------------
void __fastcall TFEnvioFM::Salir2Click(TObject *Sender)
{
   if (MessageDlg("Está seguro que quiere cerrar el programa?", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo, 0) ==
mrYes)
      Close();  //Cierra programa
}
//---------------------------------------------------------------------------
```

G Código en lenguaje Borland C++ Builder de la aplicación de recepción

```
//URecibir.cpp
//Datos se reciben por serial a traves de timer1
//Recibe los distinto paquetes y los junta en un solo arreglo para abrir un archivo.
//Manda el dato de tunning al PIC asi como cuando empezar a transmitir y cuando ya no.
//Comandos utilizados:
//     -Para empezar a recibir datos: 1 binario
//     -Para terminar 2 binario.
//Guarda archivo como Nombre de Estacion/Nombre de Archivo

//Librerias requeridas para decodificador
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "URecibir.h"
#include "Serialunit.h"


#include <stdio.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <io.h>
#include <dos.h>
#include <stdio.h>
#include "sysdyn.h"
#include <dstring.h>

#pragma package(smart_init)
#pragma link "CSPIN"
#pragma resource "*.dfm"

//Variables de codificador y decodificador
//-----------------------------------------
#define G    0x5b9      // x^{10} + x^8 + x^7 + x^5 + x^4 + x^3 + 1
#define P    0x31B      // x^9 + x^8 + x^4 + x^3 + x + 1
#define K21  32767      // 2^{k-1} - 1
#define K2   32768      // 2^{k-1}
#define FLAG  1024      // 2^{n-k}
#define NK2   512       // 2^{n-k-1}
#define NK   0x7FF      // n-k+1 unos
#define TRAP 0x01F      // patron de errores = 0000011111
#define RED2   9        // n-k-2
#define DBA  0x000000FC   //Desplazamiento bloque A
#define DBB  0x00000198
#define DBC  0x00000168
#define DBD  0x000001B4

//-------Funciones Decodificador
int FDecoder(int Mr, int S);          //int Sr, i, j, k, bit, Mhat, error;
//Esta function fue diseñada por Morelos-Zaragoza
void FDecodifica(void);               // lee archivo texto y guarda la info recuperada.
int FMensaje (int byte1, int byte2);   //junta byte1 y byte2 en byte1byte2
int FSindrome(int byte1, int byte2);   //junta sindrome y lo desplaza 6 espacios hacia la derecha
void FGuardaMr (int Mr, int status);   //Guarda Mensaje recuperado en la parte del arreglo que le toca
void FDimensionAP(void);              //inicializa el arreglo de paquetes para recuperar el archivo
void FDecodPaquetes(int Mensaje, int Sindrome); //Decodifica informacion y la Guarda en arreglos de paquetes.
void FGuardaPaquetes(void);           //Guarda los paquetes recuperados en un archivo
void FTraspasoArreglo(char *pArreglo);  //Traspasa de matriz dinamica a arreglo char
void FRevision(int Mr);               //Revisa si los paquetes ya se recibieron si tienen error o no.
void FSintonizar(int opcion);         //Encuentra la siguiente estacion con RDS
void FMandaTunning(void);             //Manda el entero que representa el voltaje de sintonización
void FEorrar(int opcion);             //Borra el contenido de objetos como de la barra de estado
```

```
//Funciones serial
extern void FSerial (int NCom);                    //Abre puerto serial
extern void FManda (char* texto, int numbytes);    //Manda cadena por puerto serial
extern int FRecibe (char* dato, int numbytes);     //Recibe cadena del puerto serial
extern void FCierraSerial(void);                   //Cierra puerto serial

//----Variables Globales
int Sr;              //Sindrome Recuperado
int BpA;             //Bytes por Archivo
int BpP;             //Bytes por Paquete
int BpPmax;          //Numero máximo de bytes por paquete
int PpA;             //Paquetes por Archivo
int NActualizacion;  //Actualizacion
int epaquete;        //contador de paquetes recuperados
int ebyte;           //contador de bytes recuperados en cada paquete
int NPaquete;        //Numero de paquete recibido
int Tunning;         //Valor entero para sintonizar el radio
int ErrorP;          //Errores posibles acumulados en un paquete
int epaqueteE;       //Numero de paquetes con errores
char ABCD;           //Indicador de recepcion del encabezado del paquete
int Error;           //Error posible de cada paquete
int verificacion;    //Bandera de la funcion FDecodPaquetes
int link;            //Guarda el numero de archivos que se transmitieron
int n_link;          //Lleva la cuenta de los archivos que se reciben del mismo grupo
bool sintonizado;    //Bandera de sintonizado, 1 si esta sironizado; 0 si no
int  auto_tunning;   //Bandera que indica que se esta utilizando sintonización automática
bool V_sintonizador; //Bandera que indica si se esta sintonizando alguna estación
bool VTimer2;        //Bandera utilizada
char *str1 = "c:\\RadioRDS\\";   //Cadenas para nombre de archivo
char *strN = "                ";

DynamicArray<DynamicArray<char> > APaquetes;    //arreglo dinamico para guardar paquetes de archivo recibido

TFRecibeFM *FRecibeFM;
//---------------------------------------------------------------------------
__fastcall TFRecibeFM::TFRecibeFM(TComponent* Owner)
    : TForm(Owner)
{
}

//---------------------------------------------------------------------------
//FGuardaPaquetes: Guarda paquetes recuperados en un archivo con el nombre que viene indicado
//en el archivo recuperado
//Utiliza variables:BpPmax, PpA
//Funciones: FTraspasoArreglo
//---------------------------------------------------------------------------
void FGuardaPaquetes(void){
 int iFileHandle;
 int iFileLength;
 int iBytesRead;
 char *pMensaje;      //Arreglo para mensaje recuperado
 char *pInfo;         //Arreglo temporal para traspasar la info.
 char *pNombreArchivo; //Arreglo para guardar el nombre del archivo.
 int frecuencia, Act_grupo, posicion;
 static linkv, NActualizacionV;

    pMensaje = new char[BpPmax*PpA];    //Crea arreglo char con el tamaño del archivo recuperado
    FTraspasoArreglo(pMensaje);

    StrLCopy(strN,pMensaje,10);          //En strN queda el nombre del archivo con terminacion
    pInfo = new char[11];
    StrLCopy(pInfo, pMensaje+16, 11);    //recuperar estacion
     FRecibeFM->LEstacionD->Caption = pInfo;

     FRecibeFM->Pagina3->Text=str1+FRecibeFM->LEstacionD->Caption+"\\"+strN;

    delete [] pInfo;
    pInfo = new char[3];
    StrLCopy(pInfo, pMensaje+21, 3);    //recuperar pais
```

```
    FRecibeFM->LPaisD->Caption = pInfo;

    frecuencia = pMensaje[24];        //recupera la frecuencia
    FRecibeFM->LFrecuencialD->Caption =  FloatToStr((frecuencia *0.1)+88);

    link = pMensaje[25];
    FRecibeFM->LNLinksD->Caption = IntToStr(link);

    Act_grupo = (NActualizacion & 0x0000FF00)>>8;  //Guarda la actualización del grupo recuperado

    posicion= FRecibeFM->ListaHTML->Items->IndexOf(FRecibeFM->Pagina3->Text); //Encuentra posición del archivo
recuperado si ya existe en lista HTML

    if(posicion== -1){                //Si no existe lo agrega a la lista y lo guarda

        FRecibeFM->ListaHTML->Items->Add(FRecibeFM->Pagina3->Text);
        FRecibeFM->ListaHTML_Act->Items->Add(Act_grupo);

        if(!DirectoryExists("c:\\RadioRDS\\"+FRecibeFM->LEstacionD->Caption))     //Revisa si ya existe el directorio de la
estacion
            if(!CreateDir("c:\\RadioRDS\\"+FRecibeFM->LEstacionD->Caption))     //Si no existe lo crea
                throw Exception("No se puede crear archivo");

        if((Act_grupo == NActualizacionV) && (link == linkv))       //Revisa el numero de actualización, si pertenece al mismo
grupo
            n_link = n_link + 1;                    //suma 1 a los links recuperados
        else{
            NActualizacionV = Act_grupo;                //Si es otro grupo entonces limpia contador y empieza
            linkv = link;                           //denuevo
            n_link = 1;
        }

        iFileHandle = FileCreate(FRecibeFM->Pagina3->Text);     //Crea nuevo archivo para guardar la info.
        FileWrite(iFileHandle, pMensaje+26,BpA-26);             //Guarda la información
        FileClose(iFileHandle);                         //Cierra el archivo
    }
    else{
        if(Act_grupo != StrToInt(FRecibeFM->ListaHTML_Act->Items->Strings[posicion])){  //Si el archivo ya existe
            FRecibeFM->ListaHTML_Act->Items->Delete(posicion);              //Cambia la actualización por la más reciente
            FRecibeFM->ListaHTML_Act->Items->Insert(posicion, Act_grupo);

            if((Act_grupo == NActualizacionV) && (link == linkv))       //Si es del mismo grupo suma 1 a los links
                n_link = n_link + 1;
            else{
                NActualizacionV = Act_grupo;                //si es otro grupo limpia el contador y empieza de nuevo
                linkv = link;
                n_link = 1;
            }

            iFileHandle = FileCreate(FRecibeFM->Pagina3->Text);     //Crea nuevo archivo para guardar la info.
            FileWrite(iFileHandle, pMensaje+26,BpA-26);
            FileClose(iFileHandle);

        }
    }

    FRecibeFM->LNActualizacion->Caption = Act_grupo;                //Actualiza la información que se despliega en la
pantalla.
    FRecibeFM->Carpetas->Directory = "c:\\RadioRDS\\"+FRecibeFM->LEstacionD->Caption;
    FRecibeFM->LNLinksDn->Caption = IntToStr(n_link);

    delete [] pMensaje; //borra arreglos
    delete [] pInfo;
}


//---------------------------------------------------------------------------
//FTraspasoArreglo: Traspasa lo que hay en APaquetes al puntero que se le pase.
//Utiliza variables: APaquetes, BpPmax
//---------------------------------------------------------------------------
```

```
void FTraspasoArreglo(char *pArreglo){
  int i,j,s;

     s=0;
     for(i=0; i<APaquetes.Length; i++){
        for(j=1; j<BpPmax+1; j++){
           pArreglo[s] = APaquetes[i][j];
           s=s+1;
        }
     }
}

//---------------------------------------------------------------------------
//FDecodPaquetes:  Identifica los paquetes, obtiene la información necesaria de los encabezados
// y los manda a guardar en APaquetes, en la posición correcta.
//Utiliza variables: Mensaje, Sindrume, Error, BpP, BpPmax, PpA, NPaquete, NActualizacion, verificacion ,ABCD
//Funciones:  Revision(), FDecoder()
//---------------------------------------------------------------------------

void FDecodPaquetes(int Mensaje, int Sindrome){
     int Mr;

     if(Error>2)          //Si error por bloque > 2 entonces se marca bloque como malo, para asegurar una correcta recuperación de los
paquetes.
          Error = 1;
     else
          Error = 0;      //si error < 2 entonces la correccion de errores lo puede recuperar

     Mr=FDecoder(Mensaje, Sindrome); //Se manda el bloque a la funcion FDecoder para corregir y recuperar el
desplazantiento

     switch (Sr) {
          case DBA :          //BloqueA

            if(Error>0)
               ABCD=0;       //Por la importancia del encabezado no se acepta ningun error.

            else{
               ABCD='A';
               if (verificacion ==1){
                                 //Revisa que el primer paquete recuperado no sea el ultimo paquete
                    if (BpP<Mensaje){          //ya que podria tener un tamaño de bytes mas pequeño
                              verificacion = 0;
                         BpPmax = Mensaje;      //Guarda el numero maximo de bytes por paquete
                    }
                    else{
                         BpPmax = BpP;          //Guarda el numero maximo de bytes por paquete
                              verificacion = 2;
                    }
               }

               BpP=Mensaje;
               if (verificacion == 0)
                    BpPmax=BpP;          //si es primera vez guarda el numero de bytes en BpPmax
               cbyte=0;
               FRecibeFM->Timer2->Enabled = false;          //Resetea el contador del timer2
               FRecibeFM->Timer2->Enabled = true;
            }

            break;

          case DBB :                    //Bloque B
            if(Error>0 || ABCD !='A')        //Revisa que ya se haya recuperado el bloque A sino lo descarta
               ABCD=0;                  //Poniendo ABCD igual a 0 forza al programa a esperar el siguiente
                                   //encabezado
            else{
               ABCD='B';
               if(PpA!=Mensaje)
                         verificacion = 0;
               PpA = Mensaje;          //Guarda los paquetes por archivo.
```

```
            }

            FRecibeFM->Progreso->Max = PpA;    //Coloca el valor maximo en la barra de progreso que indica los paquetes
recuperados
            FRecibeFM->StatusBar1->Panels->Items[6]->Text = PpA;       //Lo despliega en la barra de estado

            break;

          case DBC :
            if(Error>0 || ABCD !='B')       //igual que en el anterior espera que se haya recuperado ya el bloque B
               ABCD=0;
            else{
               ABCD='C';
               NPaquete = Mensaje;        //Guarda el numero de paquete que se va a recuperar
            }
            break;

          case DBD :
            if(Error>0 || ABCD !='C')        //Igual que con los bloques anteriores
               ABCD=0;

            else{
               ABCD='D';
               ErrorP=0;

               if((NActualizacion != Mensaje || verificacion==0)|| APaquetes.Length == 0){   //Si no son iguales quiere decir que es
otra pagina
                    APaquetes.Length = 0;              //Borra arreglo y
                    FDimensionAP();                    //Se crea nuevo arreglo.
                    verificacion = 1;
               }
               NActualizacion = Mensaje;      //se guarda la actualizacion del archivo
            }
            break;

            default:{
               if(ABCD == 'D')              //Si ya se recupero todo el encabezado, se prosigue a guardar...
                        FRevision(Mr);         //los bytes que contiene

               break;
            }       //default
          }         //fin case
}                   //fin funcion


//---------------------------------------------------------------------------
//FRevision: Revisa los bytes entrantes para ver si ya se completo el archivo
//Usa variables: cbyte, cpaquetes, cpaquetesE, APaquetes, BpA, PpA, BpP
//Funciones: FGuardaPaquetes(), FGuardaMr()
//---------------------------------------------------------------------------

void FRevision(int Mr){
 char comando[1];
 int status;            //guarda estado del paquete: recuperado con error, sin error, vacio.

     status=APaquetes[NPaquete][0];

      FGuardaMr(Mr, status);  //Guarda los bytes recibidos en APaquetes

//Se revisa el paquete

       if(cbyte == BpP && status!=1){     //si contador de bytes = numero de bytes por paquete y Paquete no ha sido recibido sin
errores
            if(status ==0){                    //Aumenta cantidad de bytes si paquete es nuevo
                 BpA = BpA + cbyte;
              cpaquete = cpaquete + 1;  //aumenta numero de paquete recibido
          }

            if(ErrorP == 0){         //se revisa la cantidad de posibles errores en el paquete
                 if(status ==2)    //Si era un archivo con error se corrige al estatus de recuperado correctamente
```

```
                epaqueteE = epaqueteE - 1;   //Resta uno a paquetes con error
                APaquetes[NPaquete][0]=1;
            }
            else{
                if(status ==0)              //si es primera vez que se recupera el paquete malo entonces
                    epaqueteE = epaqueteE+1;   //cuenta paquete con error y aumenta el contador
                APaquetes[NPaquete][0]=2;       //y le asigna 2 para marcarlo con error
            }

            ErrorP = 0;                 //se inicializan variables pra recibir siguiente paquete
            ABCD = 0;                   //indicador de recepcion de encabezado
            cbyte = 0;                  //contador de bytes = 0

            FRecibeFM->Progreso->Position = epaquete-epaqueteE;     //Despliega cantidad de paquetes en barra de progreso
            FRecibeFM->StatusBar1->Panels->Items[2]->Text = epaquete-epaqueteE; //En barra de estado
            FRecibeFM->StatusBar1->Panels->Items[4]->Text = epaqueteE;

            if ((epaquete-epaqueteE) == PpA){  //Si el numero de paquetes es igual al numero de paquetes por archivo
                    FGuardaPaquetes();      //Se llama funcion para guardar paquetes en archivo

                if (n_link ==link){      //Se revisa las paginas que han sido recuperados, si es igual al total
                    if(FRecibeFM->CAutomatico->Checked ==True){   //y si esta en modo de recepcion automatica
                        sintonizado=false;                  //se cambia sintonizado para encontrar la siguiente estacion
                        Tunning = Tunning+1;                //se aumenta la variable de sintonizacion
                    }
                }
                    APaquetes.Length = 0;       //Se borra el arreglo dinamico
                    verificacion = 0;           // verificacion se pone a cero para siguiente archivo
                epaquete = 0;               //Se inicializan las variables para esperar otro archivo
                    BpA=0;
                BpP=0;
                PpA=0;
            }
        }   //if mayor

        if(cbyte==BpP && status==1){    //si el paquete ya habia sido recuperado
            cbyte = 0;              //borra el contador de paquetes y el indicador
            ABCD=0;                 //de encabezado
        }
}       //fin funcion


//---------------------------------------------------------------------------
//FGuardaMr: Guarda Mr en Arreglo APaquetes mientras se recupera el archivo.
//Variables utilizadas: cbyte, ErrorP, Error, status, APaquetes
//---------------------------------------------------------------------------

void FGuardaMr(int Mr, int status){

        cbyte = cbyte + 1;              //se aumenta el contador de bytes
        ErrorP= Error + ErrorP;         //se suman el error del bloque entrante

        if((status ==0 ||status ==2)&& NPaquete<APaquetes.Length && cbyte<BpP){     //Si el paquete no ha sido recibido sin
errores entonces se guarda el byte
            APaquetes[NPaquete][cbyte] = (Mr>>8) & 0x000000FF;      //Parte alta
            cbyte = cbyte + 1;
            APaquetes[NPaquete][cbyte] = Mr & 0x000000FF;           //Parte baja
        }
        else
            cbyte = cbyte +1;                                       //Si ya, solo se aumenta el contador para llevar la cuenta
}

//---------------------------------------------------------------------------
//FDimensionAP: Asigna las dimesiones de Arreglo dinamico Paquetes
//y inicializa variables relacionadas: cbyte, epaqueteE, epaquete, BpA
//Variables utiliza:  BpP, PpA
//---------------------------------------------------------------------------

void FDimensionAP(void){
 int s;
```

```
    APaquetes.Length= PpA;
    for(s=0; s<PpA; s=s+1){
        APaquetes[s].Length = BpP+1;
        APaquetes[s][0] = 0;
    }
    BpA=0;
//    BpPmax = 0;
    cbyte = 0;
    cpaqueteE = 0;
    cpaquete = 0;
}

//---------------------------------------------------------------------------
//Encuentra la siguiente estacion con RDS, para lo cual hace una suma de la
//calidad de señal en una muestra de 110 datos y si la calidad de señal es mayor a
//a cierto porcentaje entonces se toma que esta sintonizado.
//La entrada opcion sirve para indicar si se quiere que sinonice (opcion =1) o
//si se quiere que termine de sintonizar no importando donde este.
//Variables: sintonizado, que es donde queda el estado de la sintonizacion
//True=si esta sintonizado; False=si no esta sintonizado
//---------------------------------------------------------------------------

void FSintonizar(int opcion){
    static int suma_qlty, cont_qlty;

    if(opcion == 1)
        cont_qlty=111;              //se pone el contador en el ultimo dato a recibir

    else{
    FRecibeFM->Timer2->Enabled = false;       //Se deshabilitan los botones relacionados
    FRecibeFM->CAutomatico->Enabled = false;  //para no tener conflictos con que se presionen
    FRecibeFM->CAutoTunning->Enabled = false; //mientras se esta sintonizando
    FRecibeFM->ScrollBar1->Enabled = false;

    suma_qlty = suma_qlty + Error;
    cont_qlty = cont_qlty + 1;

    FRecibeFM->SB_Tunning->Panels->Items[0]->Text = "Sintonizando.....";

    if(suma_qlty > 0){              //0% de 2860 = 110 datos * 26 errores
        suma_qlty = 0;              //quiere decir que señal tiene muchos errores
        cont_qlty =0;               //entonces se pasa a la siguiente estacion

        if(FRecibeFM->CAutomatico->Checked == true){  //si esta en recepcion automatica
            if(Tunning >= 230)
                Tunning = 45;               //revisa los limites
            else
                Tunning = Tunning+1;        //suma uno a Tunning
            FMandaTunning();                //Lo manda al PIC

        }

        if(auto_tunning!=0){                //revisa si esta en auto_tunning
            if(auto_tunning ==1){           //si auto_tunning=1 entonces debe reducir Tunning
                if(Tunning <= 45){  //se revisan los limites
                    V_sintonizador = true;
                    Tunning = 230;
                }
                else{
                    V_sintonizador = true;
                    Tunning = Tunning-1;
                }

            }
            else{                   //si auto_tunning=0 entonces debe aumentar Tunning
                if(Tunning >= 230){ //se revisan lo limites
                    V_sintonizador = true;
                    Tunning = 45;
                }
                else{
```

```
                    V_sintonizada = true;
                    Tunning = Tunning+1;
                }
            }
            FMandaTunning();   //Manda al PIC el nuevo valor de sintonizacion
        }

      }
    }
    if(cont_qlty>110){          //si llega hasta aqui, la señal está bien
        auto_tunning = 0;   //o sea ya esta sintonizado
        suma_qlty = 0;      //inicializa variables para siguiente vez
        cont_qlty =0;

        FRecibeFM->ScrollBar1->Position = Tunning;

        V_sintonizador = false;
        sintonizado = true;

        FRecibeFM->ScrollBar1->Enabled = true;  //Habilita botones y barra de
        FRecibeFM->CAutomatico->Enabled = true; //sintonizacion
        FRecibeFM->CAutoTunning->Enabled = true;
        FRecibeFM->Timer2->Enabled = true;
        if(opcion==1)
            FRecibeFM->SB_Tunning->Panels->Items[0]->Text = "Parado....";
        else
            FRecibeFM->SB_Tunning->Panels->Items[0]->Text = "Sintonizado";
    }

}

//---------------------------------------------------------------------------
//FBorrar: Funcion que borra los valores que se despliegan en la pantalla
//Segun la opcion borra distintos elementos
//opcion = 0: Borra valores en barra de estado
//opcion = 1: Borra el contenido de los lebels
//---------------------------------------------------------------------------
void FBorrar(int opcion){

    switch(opcion){

    case 0:
       FRecibeFM->StatusBar1->Panels->Items[2]->Text = "";
       FRecibeFM->StatusBar1->Panels->Items[4]->Text = "";
       FRecibeFM->StatusBar1->Panels->Items[6]->Text = "";
       FRecibeFM->Progreso->Position = 0;
       break;

    case 1:
        FRecibeFM->LEstacionD->Caption = "_____ _____";
        FRecibeFM->LFrecuenciaD->Caption = "__";
        FRecibeFM->LPaisD->Caption = "__";
        FRecibeFM->LNLinksD->Caption = "__";
        FRecibeFM->LNLinksDn->Caption = "__";
        FRecibeFM->LNActualizacion->Caption = "__";
        FRecibeFM->Pagina3->Text = "_____";
        break;

    }

}

//---------------------------------------------------------------------------
// FSindrome: Junta los dos bytes del Syndrome en 1 palabra de 16 bits
//---------------------------------------------------------------------------
int FSindrome(int byte1, int byte2){
    int byte12;
    byte12 = byte1;
    byte12 = byte12 & 0x000000FF;
    byte12 = byte12<<8;
```

```
    byte12 = byte12 | (byte2 &  0x000000FF);          //Recupera sindrome de 16 bits
    byte12 = (byte12 >>6) & 0x000003FF;
    return byte12;
}

//---------------------------------------------------------------------------
//FMensaje:pone el mensaje en una variable de 16 bits
//---------------------------------------------------------------------------
int FMensaje (int byte1, int byte2){
     int byte12;
    byte12 = byte1;
    byte12 = byte12 & 0x000000FF;
    byte12 = byte12 << 8;
    byte12 = byte12 | (byte2 &  0x000000FF);          //Recupera palabra de 16 bits
    return byte12;
}

//---------------------------------------------------------------------------
//Manda la variable Tunning al PIC
//---------------------------------------------------------------------------
void FMandaTunning(void){
  char *comando;

    FRecibeFM->LTunning->Caption = Tunning;
    FRecibeFM->LTunningF->Caption = Tunning*0.1135 + 83.424;

    comando = new char [1];
    comando[0]=Tunning;
    FManda(comando,1);
    delete [] comando;
}

//---------------------------------------------------------------------------
//FDecoder: Funcion que se encarga de aplicar la correccion de errores a
//a la data recuperada. Es la funcion inversa de FEncoder en el transmisor
//---------------------------------------------------------------------------

int FDecoder(int Mr, int S){
    int i, j, k, bit, Mhat, error;

//Paso1:  Multiplica por P y divide dentro de G.

 Sr = 0;       //Se inicializa la variable donde va a quedar el sindrome
 j = K2;

 for (i=15; i>=0; i--) {     //ciclo para pasar por los 16 bits de la data
    bit = ( (j & Mr) >> i ) & 0x01;
    j = j>>1;
    if (bit)
        Sr = ((Sr<<1)^P) & NK;
    else
        Sr = (Sr<<1) & NK;
    if (Sr & FLAG)
        Sr = (Sr^G) & NK;
 }

// Paso2: Se cargan los bits de paridad en el registro

 j = NK2;

 for (i=9; i>=0; i--) {     //ciclo para cargar los 10 bits de paridad
    bit = ( (j & S) >> i ) & 0x01;
    j = j>>1;
    if (bit)
        Sr = ((Sr<<1)^P) & NK;
    else
        Sr = (Sr<<1) & NK;
    if (Sr & FLAG)
        Sr = (Sr^G) & NK;
 }
```

```
// PAso3: Verificacion de sindrome y correccion de errores

 j = K2;
 Mhat = 0;    //inicializa la variable donde quedara el mensaje corregido
 for (i=15; i>=0; i--) {
     bit = ( (j & Mr) >> i ) & 0x01;
// Si hay errores se corrigen
     if (!(Sr & TRAP)) {
          error = ((Sr & NK2) >> RED2) & 0x01;
          bit ^= error;
          Sr = (Sr<<1) & NK;
     }
     else {
          Sr = (Sr<<1) & NK;
          if (Sr & FLAG)
               Sr = (Sr^G) & NK;
     }
     Mhat += (bit*j);
     j = j>>1;
 }

 return Mhat;
}  // end for M

//---------------------------------------------------------------------------
//FDecodifica:Abre un archivo y lo decodifica con FDecoder guardandolo en otro archivo
//Funcion de prueba, utiliza las mismas funciones que la funcion real.
//---------------------------------------------------------------------------

void FDecodifica (){

 int iFileHandle;
 int iFileLength;
 int iBytesRead;
 char *pszBuffer;     //Arreglo para mensaje leido
 int t;
 int Mensaje, Sindrome, Sgn_RDS;

    //Lee txt codificado
try{
    iFileHandle=0;
    iFileHandle = FileOpen(FRecibeFM->Pagina2->Text, fmOpenRead);   //encontrar pointer
    iFileLength = FileSeek(iFileHandle,0,2);        //pone el puntero al final
    FileSeek(iFileHandle,0,0);
    pszBuffer = new char[iFileLength+1];
    iBytesRead = FileRead(iFileHandle, pszBuffer, iFileLength);
    FileClose(iFileHandle);
   }
catch(...)
   {
    Application->MessageBox("Can't perform one of the following file operations: Open, Seek, Read, Close.", "File Error", IDOK);
   }

//Decodifica txt codificado con funcion FDecoder
    for (t=0;t<iBytesRead;t=t+4)
    {
    Mensaje = FMensaje(pszBuffer[t],pszBuffer[t+1]);        //Recupera palabra de 16 bits
    Error = pszBuffer[t+3] &  0x1F;              //deja solo el dato del error
    Sgn_RDS = (pszBuffer[t+3] &  0x10)>>1;
    pszBuffer[t+3] = pszBuffer[t+3] &  0xC0;
    Sindrome =  FSindrome(pszBuffer[t+2],pszBuffer[t+3]);     //Recupera sindrome de 16 bits

    if (Sgn_RDS == 1)
         FRecibeFM->StatusBar1->Panels->Items[8]= 0;
    else
         FRecibeFM->StatusBar1->Panels->Items[8]->Text= 1;


    FDecodPaquetes(Mensaje, Sindrome);       //Prueba de la funcion
```

```
    }

    delete [] pszBuffer;
}
//---------------------------------------------------------------------------
            Llamadas de inicializacion Forma
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::FormCreate(TObject *Sender)
{
    Tunning = 100;                      //Inicializa valor de tunning
    LTunning->Caption = Tunning;
    LTunningF->Caption = Tunning*0.1155 + 83.424;
    FSerial(FRecibeFM->CCOM->Value);            //Abre puerto serial

    FMandaTunning();

    if(!DirectoryExists("c:\\RadioRDS")){       //Si no existe el directorio principal, lo crea.
            if(!CreateDir("c:\\RadioRDS"))
                throw Exception("No se puede crear archivo");
    }

    L1->Caption="c:\\RadioRDS";
    Carpetas->DirLabel = L1;
}

//---------------------------------------------------------------------------

void __fastcall TFRecibeFM::FormClose(TObject *Sender, TCloseAction &Action)
{
    FCierraSerial();        //Cierra puerto serial
}


//---------------------------------------------------------------------------
            Evertos de la Forma
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//Timer encargado de recibir los bytes por serial
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::Timer1Tuner(TObject *Sender)
{
    int cuenta, Mensaje, Sindrome, Sgn_RDS;
    int dato3;
    char datos[4];

    cuenta = FRecibe(datos,4);          //Revisa si hay por lo menos cuatro datos en el buffer
                                        //del puerto serial
    if(cuenta==4){                      //si hay cuatro bytes en el buffer

        Timer1->Enabled=false;      //se extrae la informacion contenida en los bytes recibidos
        dato3 = datos[3]&0x000000FF;
        Mensaje = FMensaje(datos[0], datos[1]);
        Error = dato3 & 0x1F;           //deja solo el dato del error
        Sgn_RDS = (dato3 & 0x20);       //deja solo el dato de la señal
        datos[3] = dato3 & 0xC0;
        Sindrome = FSindrome(datos[2], datos[3]);

        SQlty->Height = Error * 241/26;   //despliega la calidad de señal

        if(VTimer2==true){              //si el timer2 se ha activado
            VTimer2=false;

              APaquetes.Length = 0;     //Se borra el arreglo dinamico
              verificacion = 0;         // verificacion se pone a cero para siguiente archivo
            epaquete = 0;
            epaqueteF= 0;
          BpA=0;                    //Se inicializan variables para empezar desde cero la
            BpP=0;                  //recuperacion del archivo
            PpA=0
```

```
            ABCD=0;
            ErrorP=0;
        }

        if(sintonizado == true || (CAutomatico->Checked==false && CAutoTunning->Checked== false))
            FDecodPaquetes(Mensaje, Sindrome);    //si no esta sintonizando entra a la funcion
        else{
            if(CAutomatico->Checked==True || auto_tunning != 0)
                FSintonizar(0);              //quiere decir que se esta sintonizando
        }

        if (Sgn_RDS == 0x20)                 //se despliega la señal que existe
            StatusBar1->Panels->Items[8]->Text=0;
        else
            StatusBar1->Panels->Items[8]->Text=1;

        Timer1->Enabled=true;
    }
}

//---------------------------------------------------------------------------
//Timer2:Da 10 minutos para recuperar paquetes que llegaron erroneos o cambiar
//de estacion si es necesario
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::Timer2Timer(TObject *Sender)
{
    if(CAutomatico->Checked == true && sintonizada == true){
        sintonizado = false;        //si esta en recepcion automatica
        Tunning = Tunning + 1;      //cambia el estado de sintonizacion y aumenta uno
    }                               //a Tunning
    VTimer2 = true;                 //pone la bandera de Timer2 en verdadero
    LTimer2->Caption = 1;
    FBorrar(0);                     //y borra los datos desplegados en pantalla
    FBorrar(1);
}

//---------------------------------------------------------------------------
//Timer3: Despliega la hora en la barra de estado
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::Timer3Timer(TObject *Sender)
{
    StatusBar1->Panels->Items[9]->Text = FormatDateTime("d/ m/ yyyy ' - ' hh:mm AM/PM", Now());
}

//---------------------------------------------------------------------------
//Decodifica archivo txt de prueba
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::BDecodificarClick(TObject *Sender)
{
    NActualizacion = 255;           //Numero para empezar a comparar
    FRecibeFM ->Color = clSilver;
    verificacion = 0;
    FDecodifica();
    FRecibeFM ->Color = clAppWorkSpace;
}

//---------------------------------------------------------------------------
//AParar: es la accion que realiza cuando se quiere que el PIC deje de mandar datos
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::AParar1Execute(TObject *Sender)
{
    char *comando;

    comando = new char [1];
    comando[0]=0x02;            //Comando para el PIC parar transmitir
    FManda(comando,1);
    delete [] comando;

    FRecibeFM->Color = clAppWorkSpace;  //Se cambia color de pantalla
```

```
    BCCOM->Enabled=True;              //se habilita el boton
    Timer2->Enabled = false;          //y se deshabilita Timer2
    FSintonizar(1);                   //se llama a sintonizar con opcion 1
}

//---------------------------------------------------------------------------
//ARecibir: Accion para indicarle al PIC que mande datos a la computadora
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::ARecibirExecute(TObject *Sender)
{
    char *comando;

    if(CAutomatico->Checked == True){     //si recepcion automatica
        sintonizado = false;          //se cambia estado de sintonizado para que
        if(ScrollBar1->Position==45)  //sintonice una estacion
            ScrollBar1->Position = 46;
        else
            ScrollBar1->Position = 45;
    }

    NActualizacion = 0xFFFF;      //Numero para empezar a comparar
    FRecibeFM->Color = clSilver;  //se cambia color de la pantalla
    Timer1->Enabled = true;       //habilita timer de recepcion
    Timer2->Enabled = true;       //habilita timer de revision de señal.
    comando = new char [1];
    comando[0]=0x01;              //Comando para el PIC para empezar a transmitir
    FManda(comando,1);
    delete [] comando;

    BCCOM->Enabled = False;       //se deshabilita boton

}

//---------------------------------------------------------------------------
//Cuadro de seleccion que indica que esta activada la sintonizacion automatica
//despliega si esta en sintonizacion automatica o manual
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::CAutoTunningClick(TObject *Sender)
{
    if(CAutoTunning->Checked == false && CAutomatico->Checked == false )
        SB_Tunning->Panels->Items[0]->Text = "Sintonizacion Manual";
    else{
        SB_Tunning->Panels->Items[0]->Text = "Sintonizacion Automática";
        CAutomatico->Checked = false;
    }
}

//---------------------------------------------------------------------------
//Cuadro de seleccion que indica que esta activada la recepcion automatica
//despliega si esta en recepcion automatica o manual
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::CAutomaticoClick(TObject *Sender)
{
    sintonizado = false;

    if(CAutoTunning->Checked == false && CAutomatico->Checked == false)
        SB_Tunning->Panels->Items[0]->Text = "Sintonización Manual";
    else{
        SB_Tunning->Panels->Items[0]->Text = "Recepcion Automática";
        CAutoTunning->Checked = false;
    }
}
//---------------------------------------------------------------------------
//Cambia puerto serial que se esta utilizando
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::BCCOMClick(TObject *Sender)
{
    FCierraSerial();              //Cierra puerto que estaba abierto
    FSerial(FRecibeFM->CCOM->Value);  //Se abre nuevo puerto
}
```

```
//---------------------------------------------------------------------------
//Cambio de estado de la barra de sintonizacion
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::ScrollBar1Change(TObject *Sender)
{
// si el boton de autotunning esta seleccionado y no esta sintonizando entonces
     if(CAutoTunning->Checked == True && V_sintonizador==false){
          sintonizado = false;          //cambia estado de sintonizado
          if(Tunning > ScrollBar1->Position)
               auto_tunning = 1;        //Resta si barra se movio hacia arriba
          else
               auto_tunning = 2;        //Suma si barra se movio hacia abajo
     }
     Tunning = FRecibeFM->ScrollBar1->Position;  //Actualiza Tunning
     FMandaTunning();                            //Manda comando a PIC

}
//---------------------------------------------------------------------------
//Maneja los filtros de los archivos desplegados
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::FiltrosChange(TObject *Sender)
{
     LArchivos->Mask=Filtros->Mask;
}

//---------------------------------------------------------------------------
//Maneja las carpetas desplegadas
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::CarpetasChange(TObject *Sender)
{
     LArchivos->Directory = Carpetas->Directory;
}

//---------------------------------------------------------------------------
//Acciones de menu
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//Opcion de salir
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::Salir3Click(TObject *Sender)
{
    if (MessageDlg("Está seguro que quiere cerrar el programa?", mtConfirmation, TMsgDlgButtons() << mbYes << mbNo, 0) ==
mrYes)
          Close()
}

//---------------------------------------------------------------------------
//Boton de simonizacion manual, solo cambia estado de cuadros de seleccion
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::Manual1Click(TObject *Sender)
{
     CAutomatico->Checked = false;
     CAutoTunning->Checked = false;
}
//---------------------------------------------------------------------------
//Boton de sintonizacion automantica, solo cambia estado de cuadros de seleccion
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::Automatico1Click(TObject *Sender)
{
     CAutoTunning->Checked= true;
}
//---------------------------------------------------------------------------
//Boton de recepcion automantica, solo cambia estado de cuadros de seleccion
//---------------------------------------------------------------------------
void __fastcall TFRecibeFM::RecepcionAutomtica1Click(TObject *Sender)
{
     CAutomatico->Checked = true;
}
//---------------------------------------------------------------------------
```

H. Código en lenguaje Borland C++ Builder del puerto serial

```
#include <vcl.h>
#pragma hdrstop
#include <windows.hpp>
#include <vcl\vcl.h>
#include "SerialunitFinal.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------


//Variables globlales

HANDLE HSerial;                     //Handle del puerto serial
COMMTIMEOUTS etmoNew = {0}, etmoOld;  //variable para guardar la configuracion anterior

//Funciones serial
int FSerial (int NCom);               //Funcion que abre puerto serial
void FManda (char* texto, int numbytes);  //Funcion que manda un arreglo char por puerto serial
int FRecibe (char* dato, int numbytes);   //Funcion que recibe un arreglo char del puerto serial
void FCierraSerial(void);             //Funcion que cierra el puerto serial
//---------------------------------------------------------------------------
//FSerial:  Abre y configura el puerto serial devuelve handle
//---------------------------------------------------------------------------
int FSerial (int NCom){
     DCB Configuracion;

//Abrir el puerto  Com 1
switch (NCom){
 case 1:
     HSerial = CreateFile("COM1",GENERIC_READ | GENERIC_WRITE,0,0,OPEN_EXISTING,0,0);
     break;

 case 2:
     HSerial = CreateFile("COM2",GENERIC_READ | GENERIC_WRITE,0,0,OPEN_EXISTING,0,0);
     break;
 default:
     HSerial = CreateFile("COM1",GENERIC_READ | GENERIC_WRITE,0,0,OPEN_EXISTING,0,0);
     break;
}
// Si no se puede abrir el puerto entonces desplegar mensaje
 if(HSerial == INVALID_HANDLE_VALUE){
     ShowMessage("Puerto ya esta abierto o no existe");
     return -1;      //regresa -1
 }

//Configurar el puerto
 else{
 GetCommTimeouts(HSerial,&etmoOld);
 etmoNew.ReadTotalTimeoutConstant = 1;
 etmoNew.ReadTotalTimeoutMultiplier = 0;
 etmoNew.WriteTotalTimeoutMultiplier = 0;
 etmoNew.WriteTotalTimeoutConstant = 0;
 SetCommTimeouts(HSerial, &etmoNew);

 Configuracion.DCBlength = sizeof(DCB);
 GetCommState(HSerial, &Configuracion);      //Obtener la configuracion
 BuildCommDCB("9600,N,8,1", &Configuracion);  //baud rate, paridad, bits, stop bits
                                 //por default no handshaking
 SetCommState(HSerial, &Configuracion);      //Aplicar la configuracion
 return 0;
 }
}

//---------------------------------------------------------------------------
//FManda: manda "numbytes" de un arreglo "texto" char y guarda en cuenta el numero
//de bytes mandados
```

```
//---------------------------------------------------------------------------
void FManda (char* texto, int numbytes){
  DWORD cuenta;

  WriteFile(HSerial,   //handle del puerto a escribir
        texto,    //direccion de la variable donde estan los datos a mandar
        numbytes,  //numero de bytes a escribir
        &cuenta,   //direccion variable DWORD donde se guarda el #de bytes mandados
        NULL);
}

//---------------------------------------------------------------------------
//FRecibe: recibe "numbytes" datos del puerto serial y
//los guarda en "cuentaL" el numero de bytes recibidos
//---------------------------------------------------------------------------
int FRecibe (char* dato, int numbytes){
  DWORD cuentaL;

  ReadFile( HSerial,   //Handle del puerto serial
        dato,     //Direccion de la variable para almacenar los datos
        numbytes,  //numero de bytes a leer
        &cuentaL,  //direccion variable DWORD donde se almacena numero de bytes leidos
        NULL);

  return cuentaL;
}

//---------------------------------------------------------------------------
//FCierraSerial: cierra el puerto serial y regresa los valores originales
//de tiempos
//---------------------------------------------------------------------------
void FCierraSerial(void){

    SetCommTimeouts(HSerial, &etmoOld);  //Regresa los tiempos originales
    CloseHandle(HSerial);              //Cierra el puerto
}
```

I. Manual del usuario para la aplicación de transmisión

MANUAL DEL USUARIO

RDSdata
Aplicación: Envío de datos html por FM

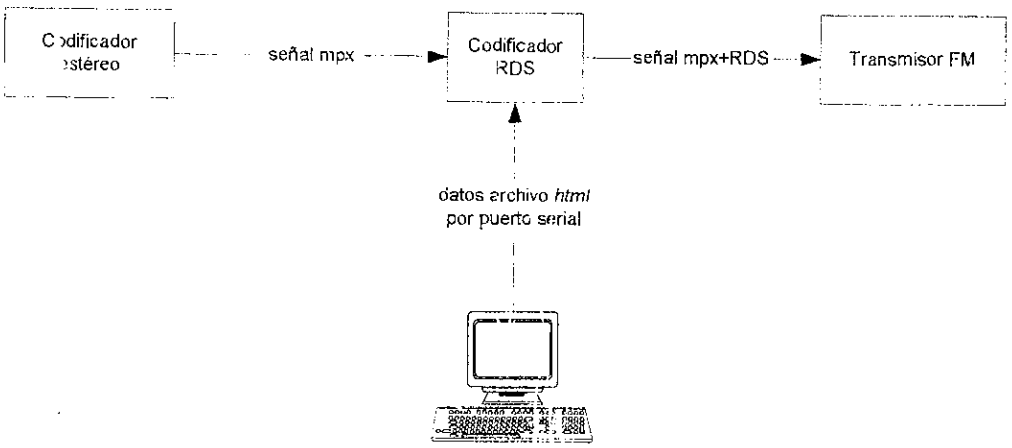
Introducción

El programa RDSdata es un programa de aplicación que funciona como una interfase entre el codificador RDS y la computadora. Esta aplicación está diseñada para tener un control de los archivos que se desean transmitir, con la opción de indicar la cantidad de veces que se quiere repetir la transmisión.

Entre la información que transmite el programa se encuentra el nombre de la estación, el país y la frecuencia, que serán datos útiles en el receptor para ordenar la información que obtenga bajo carpetas con el nombre de la estación.

Este programa interactúa con un módulo externo que se encarga de hacer la codificación RDS. La forma en la que se comunica con el módulo externo es a través de un puerto serial con conector DB9.

El diagrama le muestra el orden en que deben estar sus módulos conectados para lograr una transmisión eficiente:



Instalación

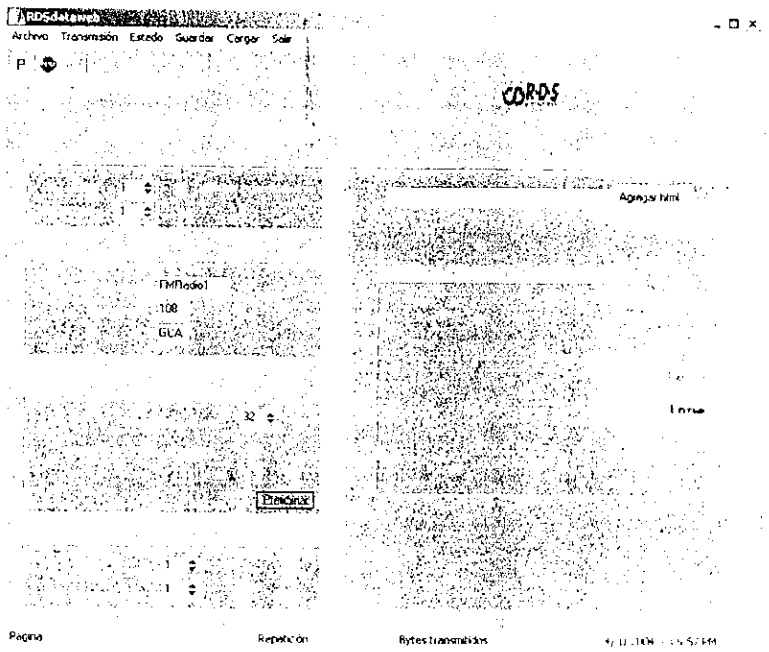
Antes que nada, revise que su computadora tenga los siguientes requerimientos mínimos para su correcto funcionamiento:

- Win2000 o superior
- Puerto serial con conector DB9 disponible
- Run-time libraries de Delphi.
- 32 Mb de memoria RAM (se recomienda tener 64 Mb o más).
- Por lo menos 200 kb de memoria disponible en disco duro.
- Procesador Pentium II 300 Mhz o superior (Puede presentar problemas con procesadores inferiores).

El programa consiste en un archivo único ejecutable por lo que es suficiente que se copie en alguna parte de su disco duro, preferiblemente, en una carpeta especial para este ubicada en el disco principal.

Iniciando el Programa

Ejecute el programa. Verá una pantalla como la que se muestra a continuación:

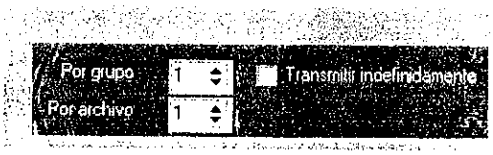


Los campos que se pueden apreciar son:

- Número de repeticiones
- Datos estación
- Datos archivo codificado
- Transmisión
- Otros

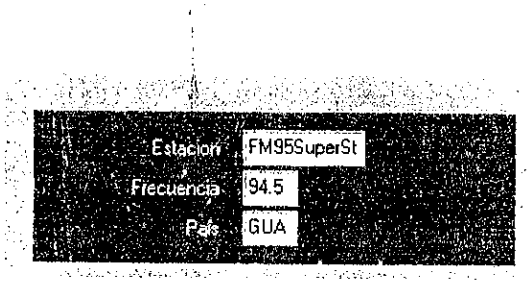
Funciones de los Campos

1. Número de repeticiones



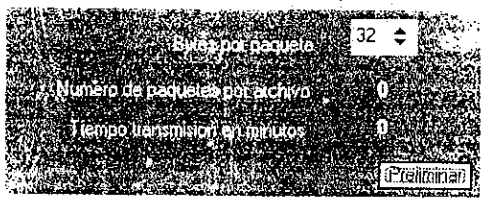
Su función es la de especificar el número de repeticiones que se desea transmitir. En sus opciones se puede especificar el número de veces que se van a transmitir todos los archivos del grupo, así como el número de veces que se transmitirá cada archivo en cada una de las repeticiones del grupo. La selección de la opción "Transmitir indefinidamente" implicará que se ignore el número de transmisiones por grupo, y transmitirá el grupo una y otra vez hasta que se le ordene parar manualmente.

2. Datos de Estación



Su función es la de ingresar los datos de la estación que está transmitiendo. Estos datos serán interpretados por el receptor, y serán útiles para la creación de la carpeta propia de la estación que contendrá los archivos recibidos de ésta. Es importante notar que, debido a lo anterior, el nombre de la estación debe estar definido únicamente por caracteres alfanuméricos, de lo contrario, el programa receptor no podrá crear la carpeta correspondiente y perderá los archivos de la estación.

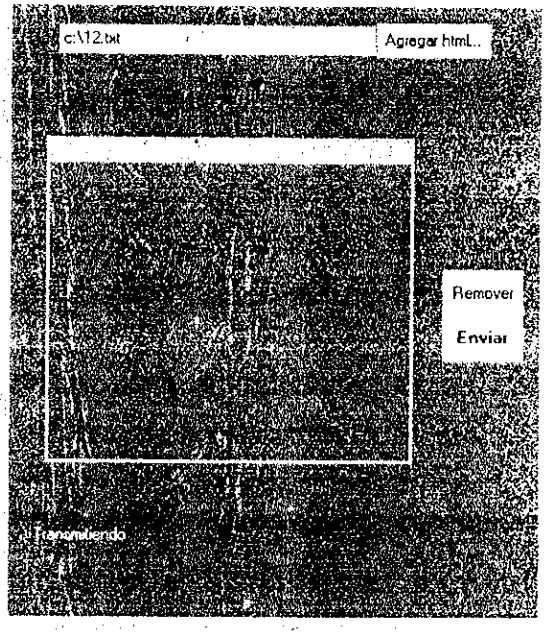
3. Datos de Archivo Codificado



En este campo se puede seleccionar el número de bytes que tendrá cada uno de los paquetes a transmitir. Un aumento en el número de bytes por paquete implica una reducción en el tiempo de transmisión, pero necesitará una mejor calidad de señal para una transferencia de datos exitosa. Por ejemplo: un valor aceptable es de 32 bytes por paquete, aunque, en mejores condiciones, se pueden lograr buenas transmisiones con 70 bytes por paquete con una significativa reducción en el tiempo de transmisión.

El botón "Preliminar" se utiliza para mostrar el número de paquetes que tendrá el archivo que se ha seleccionado previamente, así como un tiempo estimado de transmisión.

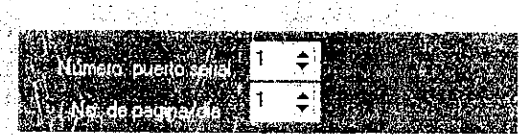
4. Transmisión



En este campo se agregan los archivos que se desean transmitir. Se pulsa el botón "Agregar *html*" y luego se selecciona el archivo que se desea transmitir. Al elegirlo aparecerá en la lista de archivos html. Los archivos que están en esta lista son los que se transmitirán. Se pueden eliminar archivos seleccionándolos de la lista y pulsando el botón "Remover". Para lograr recepciones eficientes y un tiempo de comunicación no demasiado largo, se recomienda que los archivos que se transmitan no excedan los 10 Kb.

Al pulsar el botón "Enviar" comienza la transmisión de los datos. Mientras se esté transmitiendo, el fondo del programa cambia de color, de gris oscuro a gris claro, y regresa al color original al concluir la transmisión. Durante la transmisión, el estado de la transmisión se muestra en la barra de estado del programa

5. Otros



En este campo se puede seleccionar el número de puerto serial que se utilizará como salida hacia el módulo codificador RDS. Puede ser seleccionar del COM 1 o COM2.

La opción de "No. de página/día" se utiliza por si en el transcurso del día el archivo que estamos transmitiendo ha tenido modificaciones y se desea transmitir la nueva versión. El número que aparece en esta casilla representa el número de actualización que se le ha hecho al archivo en el día.

6. Botones de Acceso Rápido

Se utilizan para acceder de manera rápida a funciones clave. El primer botón tiene la función equivalente al botón "Preliminar" del campo "Datos de archivo codificado". El segundo botón, tiene la función de detener la transmisión, y el último botón tiene la función de iniciar la transmisión, equivalente con el botón "Enviar" del campo "Transmisión".

7. Barra de Menú

Se pueden acceder a las funciones principales del programa.

- Archivo: Agregar o quitar archivos html.
- Transmisión: Iniciar o parar una transmisión.
- Estado: Resetear el transmisor.
- Guardar: Guardar la configuración actual.
- Cargar: Carga una configuración previamente guardada.
- Salir: Sale del programa.

Links

Entre el grupo de archivos *html* que se desea transmitir, pueden ir links de una página, pero deben ir referenciadas a: C:\RadioRDS\Nombe estación \...., ya que es dónde quedarán guardadas las páginas en el receptor. Tome en cuenta que también puede enviar archivos que no sean html como por ejemplo alguna imagen pequeña que lleva su página, etc.

Problemas

- Si al presionar transmitir el programa aparenta estar trabado, revise la conexión con el módulo físico y la alimentación de éste. Oprima el botón de reset del módulo físico, espere a que el programa le indique que no pudo comunicarse con el módulo y trate de nuevo.

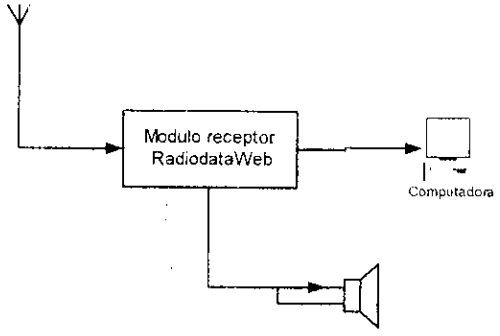
J. Manual del usuario para la aplicación de recepción

MANUAL DEL USUARIO

RDSdata
Aplicación: Recepción de datos html por FM

Introducción

La aplicación RDSdata es un programa que decodifica los datos *html* que vienen en las estaciones de radio que prestan este servicio de RDS, guardando la información recuperada en el disco duro para que se pueda ver con cualquier explorador.
Este programa funciona como la interfaz entre el módulo receptor de datos y la computadora.



Instalación

Antes que nada, revise que su computadora tenga los siguientes requerimientos mínimos para su correcto funcionamiento:

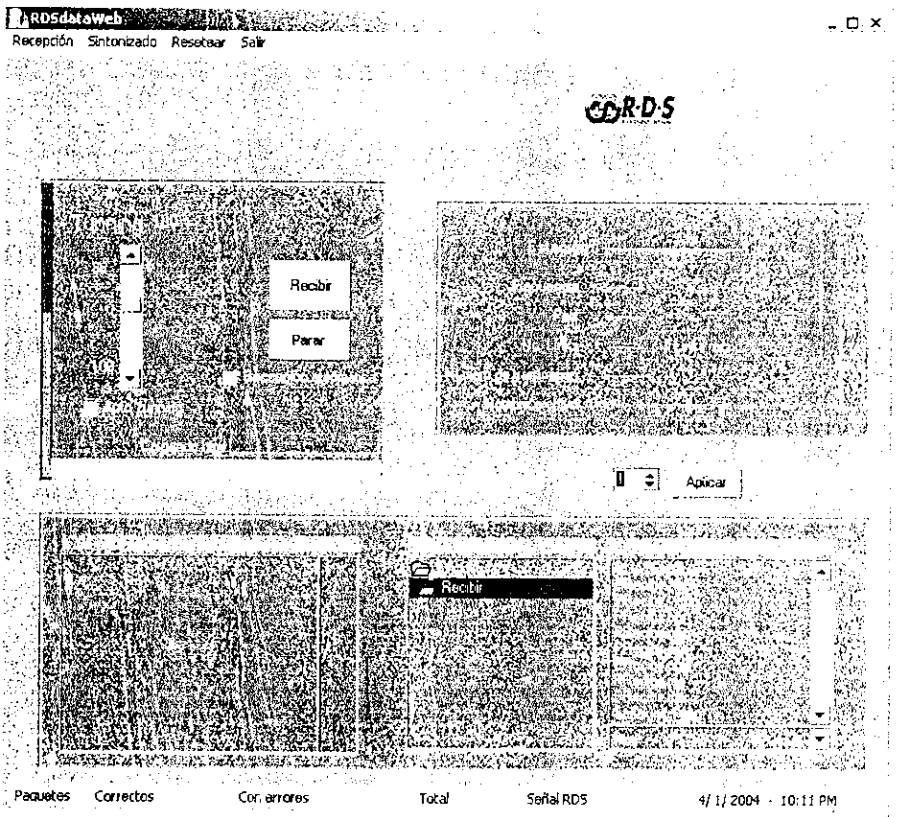
- Win2000 o superior
- Puerto serial con conector DB9 disponible
- Run-time libraries de Delphi.
- 32 Mb de memoria RAM (se recomienda tener 64 Mb o más).
- Por lo menos 200 kb de memoria disponible en disco duro.
- Procesador Pentium II, 300 Mhz o superior (Puede presentar problemas con procesadores inferiores).

El programa consiste en un archivo único ejecutable por lo que es suficiente que se copie en alguna parte de su disco duro, preferiblemente, en una carpeta especial para que esté ubicada en el disco principal.

El módulo físico tiene dos salidas, una salida serial con conector DB9 que va al puerto serial de la computadora, y una salida de audio que puede conectar a sus bocinas si lo desea.

Iniciando el Programa

Ejecute el Programa. Verá una pantalla como la que se muestra a continuación.

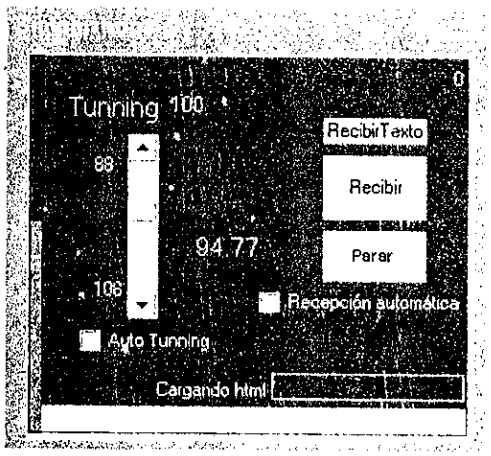


Los campos que se pueden apreciar son:

- Sintonizador
- Datos estación
- Ubicación archivos

Funciones de los Campos

1.- Sintonizador



Este campo tiene la funeión de sintonizar el receptor de radio en la estación indicada. Presenta una barra de "Dial" y se muestra la frecuencia numéricamente a un lado de esta barra. Modificando la posición de esta barra se pueden sintonizar otras estaciones de radio. El "Dial" trabaja como un dial analógico.

A un lado del campo aparece una barra de color verde con rojo. Esta barra representa la calidad de la señal RDS que se está recibiendo. Si está totalmente verde, quiere decir que se tiene una excelente señal.

2. Recepción con sintonización manual

En esta forma usted debe mover el dial para sintonizar la estación que desee, la barra que tiene del lado derecho del dial le indicará la calidad de la señal RDS.

3. Recepción con sintonizador automático

La opción de Auto Tunning hará cambio de estación automáticamente a la siguiente estación que tenga servicio RDS. Para hacer uso de esta opción, se debe seleccionar la casilla, y luego presionar una de las dos flechas del "Dial", el sistema hará el cambio automático en la dirección de la flecha que se haya presionado.

4. Recepción de datos automático

La opción de recepción automática se encarga de recorrer completamente el "Dial" en busca de estaciones que transmitan RDS. Cuando encuentra una estación que tiene este servicio, espera a recibir los archivos que se están transmitiendo. Al tener todos los archivos que transmite esta estación, busca la siguiente estación con servicio RDS. Al terminar el "Dial", inicia nuevamente el mismo proceso.

El botón "Recibir" inicia la recepción de datos. Al momento de iniciar una recepción, el fondo del programa cambiará de enlor, de gris oscuro a gris claro. Es necesario que el programa esté recibiendo datos en el momento de querer utilizar cualquiera de las dos opciones de sintonía automática y para reeuperar la información que se esté mandado de la estación.

El botón "Parar" detiene cualquier recepción que se esté dando.

Datos de la estación

En esta seceión se despliegan los datos de la estación del último archivo recuperado.

Archivos recuperados

En esta sección se muestran los archivos recuperados, así como su ubicación en el disco duro. Los archivos son guardados en una carpeta que se crea la primera vez que se corre el programa llamada RadioRDS bajo el directorio principal C:\. Bajo este directorio cada vez que se recupera un archivo de una estación se crea una carpeta con el nombre de la estación, de manera que todos los archivos de esa estación quedarán guardados ahí.

Al momento de tomar la opción "Resetear" se borran las listas de documentos recibidos, así como los datos de la estación que se despliegan.

Problemas

Los problemas que se pueden presentar en el programa, tiene que ver más con la calidad de la señal RDS.

- Si está en modo de sintonización automática o auto tunning y el programa no logra sintonizar una estación es debido a que no existe una buena señal RDS.

K. Glosario

amperio: Unidad de intensidad de corriente eléctrica del Sistema Internacional equivalente a la intensidad de la corriente que, al circular por dos conductores paralelos, rectilíneos, de longitud infinita, de sección circular despreciable y colocados a la distancia de un metro uno de otro en el vacío, origina entre dichos conductores una fuerza de dos diezmillonésimas de newton por cada metro de conductor. (Símb. *A*).

bit: Unidad de medida de información equivalente a la elección entre dos posibilidades igualmente probables.

***buffer*:** Unidad de memoria intermedia.

***byte*:** Octeto (II unidad de información).

CENELEC: Comité Europeo de Normalización Electrónica.

CMOS: Complementary Metal Oxide Semiconductor, es una tecnología semiconductora usada en los transistores que son manufacturados en la mayoría de circuitos utilizados para las computadoras. Están hechos de silicon y germanio.

EBU: Unión Europea de Emisoras.

***flip-flop*:** Circuito que alterna entre dos estados posibles cuando se recibe un pulso en la entrada.

FM: Frecuencia modulada

hercio: Unidad de frecuencia del Sistema Internacional, que equivale a la frecuencia de un fenómeno cuyo período es un segundo. (Símb. *Hz*).

hipertexto: Texto que contiene elementos a partir de los cuales se puede acceder a otra información.

html: Hyper Text Markup Language, es un lenguaje internacional para publicar hipertexto por la WWW (World Wide Web).

microcontrolador: Microprocesador que comprende elementos fijos, como la unidad central y sus memorias, y elementos personalizados en función de la aplicación.

mili: Significa una milésima (10^{-3}) parte. Se aplica a nombres de unidades de medida para designar el submúltiplo correspondiente. (Símb. *m*).

ohmio: Unidad de resistencia eléctrica del Sistema Internacional, equivalente a la resistencia eléctrica que da paso a una corriente de un amperio cuando entre sus extremos existe una diferencia de potencial de un voltio. (Símb. Ω).

***protoboard*:** Placa para realizar conexiones entre componentes electrónicos sin necesidad de soldadura facilitando el armado de circuitos.

RBDS: Radio Broadcast Data System, sistema de radiodifusión de datos.

RDS: Radio Data System, sistema de datos radiofónicos.

sincronía: Coincidencia de hechos o fenómenos en el tiempo.

TTL: Transistor-Transistor Logic, una tecnología común de semiconductores para la construcción de circuitos integrados lógicos digitales. Originado en Texas Instruments en 1965.

VHF: Very High Frecuency. Frecuencias del espectro radioeléctrico comprendidas entre los 30*MHz* y los 300*MHz*.

voltio: Unidad de potencial eléctrico y fuerza electromotriz del Sistema Internacional, equivalente a la diferencia de potencial que hay entre dos puntos de un conductor cuando al transportar entre ellos un coulomb se realiza el trabajo de un julio. (Símb. *V*).

vatio: Unidad de potencia eléctrica del Sistema Internacional, que equivale a un julio por segundo. (Símb. *W*).