Excelencia que trasciende

Guatemala, 15 de abril de 2013

A: José Edwardo Barrera Santos.

De: Lda. Silvia de Buratti

Ref.: Tesis

Por este medio le comunico que luego de cumplir con los requisitos estipulados para la elaboración de su tesis, puede iniciar los trámites para obtener su título.

Atentamente,

Lda. Silvia de Buratti
Col.7754

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Elaboración de receptor portátil FM con decodificador *RDS*

Trabajo de graduación en modalidad de tesis presentado por
José Eduardo Barrera Santos
para optar al grado académico de Licenciado en Mecatrónica

Guatemala
2012

Elaboración de receptor portátil FM con decodificador *RDS*

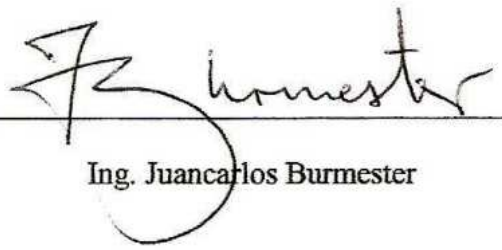
UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Elaboración de receptor portátil FM con decodificador *RDS*

Trabajo de graduación en modalidad de tesis presentado por
José Eduardo Barrera Santos
para optar al grado académico de Licenciado en Mecatrónica

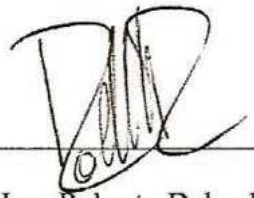
Guatemala
2012

Vo. Bo. :

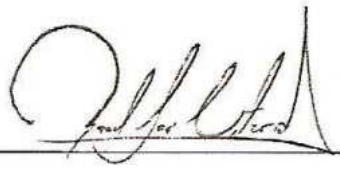
(f) 

Ing. Juancarlos Burmester

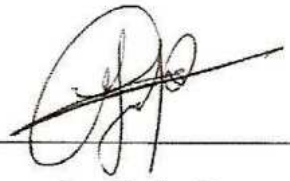
Terna examinadora:

(f) 

Ing. Roberto Delgado

(f) 

Ing. Manuel Cotero

(f) 

Ing. Pedro Romero

Fecha de aprobación: Guatemala, 4 de diciembre de 2012

PREFACIO

Como estudiante de Ingeniería Mecatrónica, tuve la oportunidad de conocer el funcionamiento de algunos elementos que nos rodean, que hasta cierto punto pueden concebirse como misterios: la reproducción de sonidos, el funcionamiento de un motor, la estructura interna de una computadora o incluso la forma en la que opera un bombillo. Se llega a desarrollar una habilidad para percibir las cosas de manera diferente, más simples y estructuradas.

Quise desarrollar un trabajo que pudiera aterrizar estos conocimientos que he adquirido en alguna aplicación directa, y que cubriera alguna de las necesidades de nuestra sociedad. Fue entonces cuando decidí enfocarme en la comunicación. Me propuse desarrollar un dispositivo de bajo costo que complemente la función que cumplen los medios de comunicación masiva: la televisión, el radio, la prensa escrita, la Internet. Aunque estos cumplen la función para la que están diseñados, cuentan con algunas debilidades que impiden que puedan ser usadas de manera más universal (como la cobertura, la portabilidad, la accesibilidad, etc.).

Después de investigar acerca de las opciones que podría utilizar, decidí enfocarme en una tecnología que nunca fue explotada por completo en nuestro país: el *Radio Data System (RDS)*. Es a través de este protocolo que, por ejemplo, algunos radios de automóviles son capaces de mostrar el nombre de la estación que está sintonizada. Si se tomara esta capacidad y se expandiera su aplicación, sería posible llegar a desarrollar un medio de comunicación masivo novedoso y funcional. Sin embargo, la pieza que haría falta en este sistema sería la que desempeñara la función de recibir y decodificar la información que se transmitiera.

Es en este punto que nace la idea de esta investigación: diseñar y elaborar un dispositivo receptor de bajo costo, que se apegara al protocolo mencionado. Al no interferir con la forma original del *RDS*, no se tiene que desarrollar tecnología nueva del lado del transmisor (estaciones radiodifusoras) sino simplemente aplicarla de una manera distinta. Finalmente, dependiendo de los resultados de la investigación, podría llegar a servir de base para una revolución en la forma que se maneja la comunicación en nuestro país.

ÍNDICE

LISTA DE CUADROS

Página

LISTA DE FIGURAS

Página

RESUMEN

En este trabajo de investigación, se exponen las consideraciones más importantes que deben seguirse para la elaboración de un dispositivo receptor FM con capacidad de manejar *Radio Data System* (*RDS*), utilizando componentes de bajo costo. Se busca apegarse fielmente a la estructura que sigue esta codificación, para que la fuente de la señal radial no tenga que ajustar su forma de manejar los mensajes para acoplarse a este receptor. Además, aumentará la versatilidad del dispositivo que le permitirá ser aplicado en otras tareas que no fueron consideradas en la realización de esta tesis.

El resultado final es un dispositivo portable y totalmente funcional, capaz de recibir señales FM, filtrar sus señales *RDS* y procesar estos datos para desplegar su información codificada en una pantalla. El dispositivo es apto para manejar tres tipos de servicios que ofrece el *RDS*: el nombre de programa (PS), la hora y fecha (CT) y *RadioText* (RT).

Los mensajes recibidos mediante este último son clasificados de acuerdo al tipo de información que contienen, en alguno de los siguientes menús: estado de tránsito, noticias y deportes. Si el mensaje no corresponde a ninguna de estas categorías, será mostrado en la pantalla principal. En cualquiera de los casos, al recibir un mensaje nuevo se producirá una notificación sonora. Cabe resaltar que es posible personalizar qué tipo de información se desea recibir (en otras palabras, escoger qué menús son de interés para el usuario).

I. INTRODUCCIÓN

El *Radio Data System* (o *RDS* por sus siglas en inglés) es un protocolo utilizado mundialmente por empresas radiodifusoras. Este es transmitido a través de la misma frecuencia radial, modulada según la Codificación Manchester en la subportadora de 57kHz. Es utilizada para comunicar información que puede ser desde el nombre de la emisora sintonizada, hasta el tipo de programa que se está transmitiendo.

En esta investigación, se desarrolla un dispositivo capaz de manejar este protocolo según los estándares oficiales, con el objetivo de servir como un receptor especializado de mensajes de texto. Para esto, se utiliza uno de los servicios que ofrece el *RDS* llamado *RadioText*, que básicamente permite la comunicación de cadenas de texto de hasta 64 caracteres de longitud. Adicionalmente, el aparato es capaz de manejar otros servicios: el de Nombre de Programa (que identifica el nombre de la estación) y el de Fecha y Hora.

Esta tecnología fue creada hace casi 30 años, sin embargo es un recurso de comunicación de bajo costo que no ha sido explotado en Guatemala. La transmisión de este protocolo no representa un costo considerable para las empresas radiodifusoras, pero ellas no se han interesado en desarrollarlo por el hecho que no existen suficientes receptores que justifiquen la inversión.

El reto que se enfrenta es el de realizar un dispositivo completamente funcional, portátil y eficiente, pero basado en componentes de bajo costo. Aunque en esta investigación se le otorga la función de identificar noticias de interés para el usuario, y categorizarlas según la naturaleza de su información, la aplicación final que se le deseara dar es completamente libre. Dado que el dispositivo respeta los estándares reales del protocolo, no se necesita el desarrollo de nueva tecnología del lado de la transmisión, haciéndolo más atractivo para las empresas radiodifusoras de este país.

II. OBJETIVOS

A. GENERALES

1. Diseñar y construir un receptor de señales FM capaz de decodificar y presentar en una pantalla LCD cadenas de texto utilizando el protocolo *RDS*.

B. ESPECÍFICOS

1. Diseñar y construir un transmisor FM controlado por la computadora capaz de comunicarse según el protocolo RDS a fin de poder hacer pruebas con el receptor.

2. Aplicar los estándares reales del protocolo RDS.

3. Implementar comunicación USB entre el receptor y la computadora para permitir la personalización de mensajes a recibir.

4. Optimizar los costos de elaboración mediante un diseño sencillo pero funcional.

III. JUSTIFICACIÓN

Lamentablemente, muchas veces se falla al no contextualizar correctamente las investigaciones que se realizan de acuerdo a la realidad del país en el que se busca aplicarlas. Aunque pueden ser avances tecnológicos muy significativos, en ocasiones, es más importante intentar satisfacer las necesidades básicas que el ambiente demanda.

En este trabajo se busca complementar el servicio que los medios de comunicación masiva realizan en Guatemala. Al mantener a la gente informada, es posible anticiparse a imprevistos más fácilmente, que varían desde desastres naturales hasta condiciones del tránsito. Actualmente se manejan mayormente cuatro medios de comunicación masiva en nuestro país: la televisión, la radio, la Internet y la prensa escrita. Cada uno de ellos cuenta con beneficios, pero también deficiencias que limitan su función. El Cuadro 1 presenta una comparativa entre algunas características de estos recursos.

Cuadro 1. Comparativa entre los medios de comunicación masiva más utilizados en Guatemala[1]

Característica	Televisión	Radio	Internet	Prensa escrita
Inmediatez: Información en tiempo real	✔	✔	✔	✘
Personalizable: Poder decidir qué me interesa	✘	✘	✔	✘
Portátil: Sin dependencia de elementos externos fijos	✘	✔	✘*	✔
Económico: De bajo costo	✘	✔	✘	✔
Amplia cobertura	✘	✔	✘	✔
Notifica cuando nueva información está disponible	✘	✘	✔	✘

✔ Beneficio
✘ Deficiencia
* Aplicable sólo en algunos casos

[1] Elaboración propia

Con el resultado final de esta investigación se podrá obtener un dispositivo capaz de recibir, decodificar y mostrar mensajes de texto provenientes de una banda FM de radiodifusión. Será un medio de comunicación masiva alternativa, con las siguientes ventajas:

- Su versatilidad permite aplicarla a una gran cantidad de funciones: mensajes de precaución a desastres naturales, información del tráfico, noticias, estado del tiempo, deportes, etc.
- La posibilidad de identificar los mensajes nuevos permite incluir alarmas u otras indicaciones sin necesidad de "refrescar" o solicitar actualizaciones.
- La naturaleza del mensaje (cadenas de texto) hace que la arquitectura del dispositivo sea más simple (tanto pantalla LCD como controlador), reduciendo sus costos.
- La inversión que deben hacer las empresas radiodifusoras es relativamente baja ya que el método de transmisión sería el mismo, únicamente tendría que adicionarse el equipo de codificación *RDS*.
- Se deja de depender de señales inalámbricas muchas veces restringidas o limitadas como el WiFi, abriendo paso a una comunicación más directa y universalmente accesible como es la de la modulación FM.
- Aunque sean los mismos mensajes los que se envíen en masa, es posible personalizar el receptor para filtrar únicamente los mensajes de interés para el usuario.
- Aplicable a cualquier estación radiodifusora, ofreciendo varios "canales" disponibles para la comunicación.

El reto es construir un receptor de bajo costo, funcional y portátil, de modo que todas estas ventajas puedan encapsularse en un solo dispositivo.

IV. MARCO TEÓRICO

Desde hace casi un siglo, la radiodifusión ha sido una parte muy importante en el proceso de comunicación masiva para el ser humano (Mishkind, 2007). Al ser un medio tan confiable y versátil, nos ha dado la posibilidad de utilizarlo en aplicaciones para las que no fue diseñado. Este trabajo se desarrollará alrededor de una de estas aplicaciones: el *Radio Data System (RDS)*.

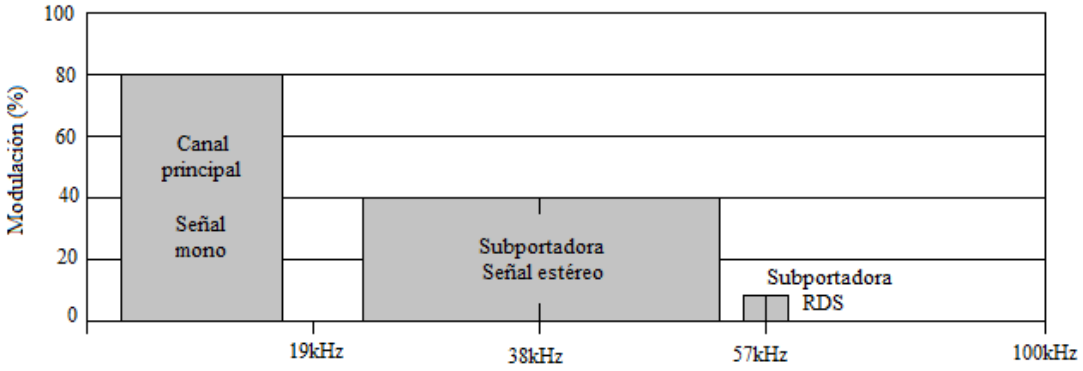
A. HISTORIA

Esta tecnología inició en 1974, cuando un grupo de investigadores del Instituto Alemán de Radiodifusión (IRT) y la empresa de autos Bosch/Blaupunkt unieron fuerzas y desarrollaron un sistema de comunicación denominado *Autofahrer Rundfunk Information*, o *ARI* por sus siglas en alemán. Básicamente este proponía utilizar una señal modulada por amplitud (AM) sobre la subportadora de 57kHz de una estación de radio. En esta, se transmitirían mensajes codificados acerca del estado del tránsito de la zona, y los radios especializados podrían recibirlos y decodificarlos según fuera necesario (Kopitz & Marks, 1999).

Sin embargo, esta propuesta no fue bien recibida por la Unión Europea de Radiodifusión (EBU), quienes tenían el poder de regir los estándares, limitaciones y requerimientos de las señales de radio utilizadas en todo el continente Europeo. Básicamente les preocupaba la fidelidad de recepción de esta señal análoga y la poca versatilidad que tendría este sistema (ya que su única función sería la de informar acerca del estado del tránsito). Así es que, a partir de 1975, se empezó a desarrollar un nuevo sistema que vendría a sustituir por completo al *ARI*: el *Radio Data System* o *RDS* por sus siglas en inglés (Kopitz & Marks, 1999).

A diferencia de su predecesor, el *RDS* se basa en un protocolo digital. Dos señales utilizadas simultáneamente, alojadas en las frecuencias 54.5kHz y 59.5kHz (nótese la propiedad de 57±2.5kHz), cifran los mensajes según la codificación Manchester y hacen a este sistema mucho más confiable que el anterior (Figura 1). Esta codificación se basa en los cambios de fase de la señal sobre el tiempo, y no en su amplitud, por lo que se adapta más a los requerimientos de la EBU. Adicionalmente, al no alojarse exactamente sobre la subportadora de 57kHz, da lugar a que los transmisores y receptores especializados para la codificación ARI sigan siendo funcionales al no interferir en su señal (Kopitz & Marks, 1999).

Figura 1. Representación de la señal de radio multiplexada[2]



Este protocolo fue concebido con la idea de cubrir una mayor gama de necesidades de los radioescuchas y las empresas de radiodifusión: estado de tránsito, nombre de la estación, tipo de programación, hora, etc. En la década de los 80 este sistema se popularizó en Europa y se extendió a Estados Unidos (donde cambió su nombre a *RBDS* por ciertos cambios que se le hicieron) (Kopitz & Marks, 1999).

En 1993 se creó el Foro *RDS*, organización que reúne a los profesionales en esta tecnología dos veces al año para conocer los avances y resultados de las más recientes investigaciones de este tema. Hoy en día es un sistema consolidado y aplicado alrededor del mundo. Fue diseñado con la versatilidad de expandir sus aplicaciones en la medida que se presenten nuevas necesidades. Aunque es una tecnología creada hace 30 años, sigue estando vigente gracias a los grandes resultados que han obtenido durante este tiempo (Kopitz & Marks, 1999).

B. CARACTERÍSTICAS GENERALES

El *RDS* cuenta con una gran cantidad de aplicaciones, que varían desde la identificación del tipo de programación, hasta el listado de frecuencias alternativas con el mismo tipo de información que pueden ser sintonizadas. Sin embargo, es indispensable que tanto la emisora como los receptores sean capaces de manejar el tipo de datos que se deseen comunicar (Seminario, 2005). Aunque la estructura de los mensajes es estándar, se ha optado en ocasiones por darle la habilidad a los receptores de decodificar sólo cierto tipo de información, dependiendo de las funciones que se desee que desempeñen. El Cuadro 2 presenta el listado de los servicios más importantes que se utilizan en este sistema.

[2] Figura adaptada de (Zuloaga, 1996)

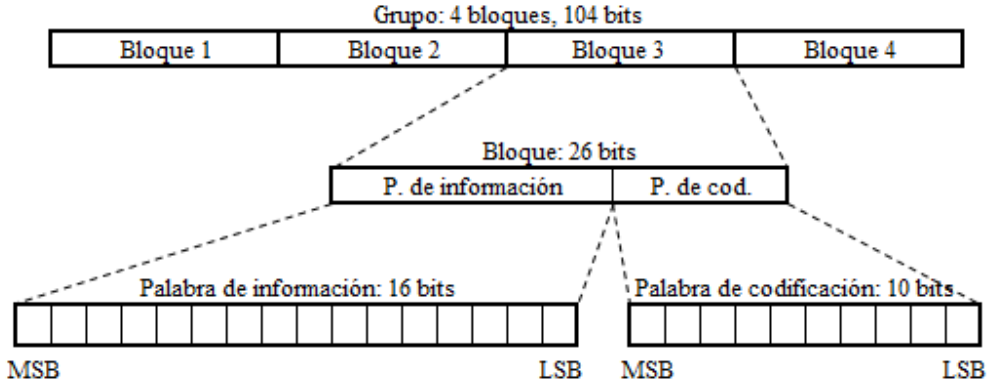
Cuadro 2. Servicios más importantes del *RDS*[3]

Código	Nombre	Descripción
PI	Identificador de programa	Código único no desplegado que indica el país o región para el que la información va dirigida.
PS	Nombre de programa	Nombre de la estación de radio. Esta es desplegada en una cadena de 8 caracteres.
AF	Frecuencias alternativas	Código que indica frecuencias alternativas que pueden ser sintonizadas para escuchar programas o información similar. Sirve para procesamiento interno de la radio, información no desplegada.
TP	Identificador de programa de tránsito	Bandera no desplegada que indica si la estación de radio puede contener información del tránsito.
TA	Identificador de anuncio de tránsito	Bandera no desplegada que indica si un anuncio de tránsito está al aire.
PTY	Identificador de tipo de programa	Identifica el tipo de información que transmite la estación de radio, a partir de un listado de 31 posibles opciones (que va desde noticias hasta deportes).
CT	Hora y fecha	Hora actual y fecha codificada según su código juliano.
RT	*RadioText*	Mensaje de texto de 64 caracteres como máximo.
DI	Identificador de decodificador	Código no desplegable que indica el tipo de transmisión que se está manejando (mono, estéreo, y otras posibilidades).
ODA	Aplicación abierta de datos	Servicio abierto que permite la implementación de nuevas aplicaciones utilizando la tecnología *RDS*.

La forma de transmitir estos tipos de información es mediante a paquetes síncronos de 104 bits transmitidos a una tasa de 1187.5bps, enviados sin separación entre uno y otro. Estos paquetes, denominados *grupos*, se componen de cuatro diferentes bloques de 26 bits cada uno (Figura 2). Cada uno de estos bloques contiene parte de la información que el grupo busca transmitir. Estos adicionalmente se divide en dos partes: la palabra de información (16 bits) y la de codificación (10 bits). Esta última sirve como un verificador de errores, que computa la palabra de información que se está transmitiendo y una palabra de desplazamiento fija del bloque en cuestión (Cuadro 3). Sólo si las cuatro palabras de codificación presentes en el grupo son correctas se analizará la información que se está transmitiendo en el grupo (Zuloaga, 1996). Este método de corrección de errores se detallará en el siguiente inciso.

[3] Cuadro adaptado de (Kopitz & Marks, 1999)

Figura 2. Estructura de los paquetes de datos para el *RDS*[4]



Cuadro 3. Palabras de desplazamiento y síndromes para cada bloque de un grupo *RDS*[5]

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

*La variación del bloque C únicamente cambia la organización de la información transmitida en el grupo.

Cada grupo válido transmitido con el protocolo *RDS* cuenta con un código en los cuatro bits más significativos del segundo bloque que identifica la información que se está transmitiendo. Es decir, cada una de las funciones que puede dársele al *RDS* (como las mostradas en el Cuadro 2) cuenta con un código de cuatro bits que ayudarán al receptor a tomar las decisiones correspondientes para manejar la información que el grupo contiene (Ogata, Sakata, Fujiwara, & Okuda, 1992). En los siguientes apartados se detallarán los servicios que fueron utilizados para el desarrollo de esta investigación.

[4] Figura adaptada de (Kopitz & Marks, 1999)
[5] Cuadro adaptado de (Kopitz & Marks, 1999)

C. DETECCIÓN DE ERRORES

La palabra de codificación enviada al final de cada bloque de todos los grupos tiene dos funciones principales: corroborar la validez de los datos recibidos en la palabra de información e identificar el bloque que se está transmitiendo. De esta manera, se podrá mantener la sincronía en la decodificación de los datos (Kopitz & Marks, 1999). Existen dos procedimientos que pueden descifrar la validez de esta palabra:

1. Este es el método usual. Básicamente es realizar el proceso inverso al que se sigue para formar esta palabra del lado del transmisor. Para esto, es importante considerar que la palabra de codificación es formada por la suma de:

   a. La palabra de desplazamiento correspondiente al grupo que está analizando, presentada en el Cuadro 3.

   b. El resto de la multiplicación de la palabra de información x con x^{10} y la división (módulo dos) de su polinomio generador $g(x) = x^{10} + x^8 + x^7 + x^5 + x^4 + x^3 + 1$.

Por lo tanto, si se realiza el procedimiento del inciso (b) de manera inversa, utilizando la palabra de información previamente recibida, se podrá obtener un vector de 10 bits al que se le debería comparar con cada una de las palabras de desplazamiento presentadas en el Cuadro 3. Si esta coincide con una de ellas, y además respeta el orden de transmisión de los bloques, se toma al bloque como válido (RDS Forum, 2008).

2. La decodificación de la palabra de codificación puede ser realizada a través de la matriz de paridad *H* de dimensión 26x10 presentada por la expresión 1. Este es el método que se aplicó en el desarrollo de esta investigación.

A una secuencia de 26 bits recibidos se le llamará el vector $\vec{x}$ de dimensión 1x26. Su multiplicación matricial con la matriz de paridad H dará como resultado un vector $\vec{s}$ de dimensión 1x10. Este, denominado "síndrome", deberá coincidir con los presentados en la tercera columna del Cuadro 3 para que sea aceptado como un bloque válido. De ser así, los primeros 16 bits del vector $\vec{x}$ será la palabra de información (RDS Forum, 2008).

$$H = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{vmatrix}$$

Expresión 1

El primer bloque de todo grupo debe ser de tipo A. Si aún no se ha obtenido sincronía, se deberán procesar todo conjunto de 26 bits hasta localizar el síndrome correspondiente. Una vez encontrado, el proceso no se repetirá hasta completar la recepción de 26 nuevos bits de información, cuyo síndrome deberá coincidir con el del tipo B. Se trabajará de la misma manera la recepción de los bloques C (o C') y D (Zuloaga, 1996). Si en algún momento se perdiera la sincronía (es decir, si el síndrome calculado no corresponde con el esperado según su orden), el proceso deberá realizarse nuevamente desde su inicio.

Este método de detección de errores permite identificar aproximadamente el 99.8% de las perturbaciones cada 11 bits (Kopitz & Marks, 1999). Según el Foro *RDS* es un sistema de sincronización de bloques confiable, que no sólo identificará los errores en la transmisión de datos sino también ayudará a encontrar el orden de transmisión de los bloques de cada grupo (RDS Forum, 2008).

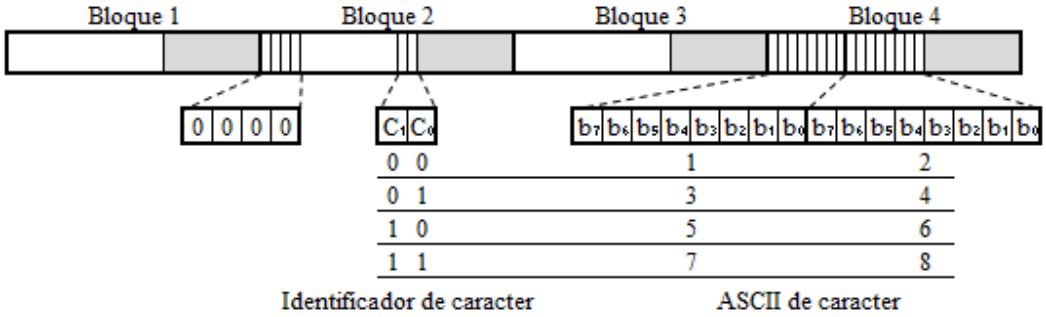
D. NOMBRE DE PROGRAMA (PS)

Es una etiqueta de ocho caracteres alfanuméricos en mayúsculas y/o minúsculas, que muestra el nombre de la estación de radio. Esta es transmitida dos caracteres a la vez, por lo que la etiqueta completa necesita de al menos cuatro grupos para completarse (European Comitee for Electrotechnical Standrization (CENELEC), 2002). Aunque la etiqueta puede incluir caracteres especiales (á, é, í, ó, ú, ñ), normalmente

los receptores no están diseñados para identificarlos correctamente, por lo que los muestran en su versión simplificada (a, e, i, o, u, n) (Kopitz & Marks, 1999).

Los cuatro bits que identifican este servicio son 0000. El código ASCII de cada uno de los dos caracteres que se envían por grupo, se encuentra en la palabra de información del cuarto bloque, separado en dos bytes. Además, el identificador de la posición que estos caracteres ocupan dentro del nombre de programa se encuentra en los dos bits menos significativos de la palabra de información del segundo bloque (Figura 3) (Elektor Electronics, 1991).

Figura 3. Estructura de los grupos de datos que contienen caracteres del nombre de programa (PS)[6]



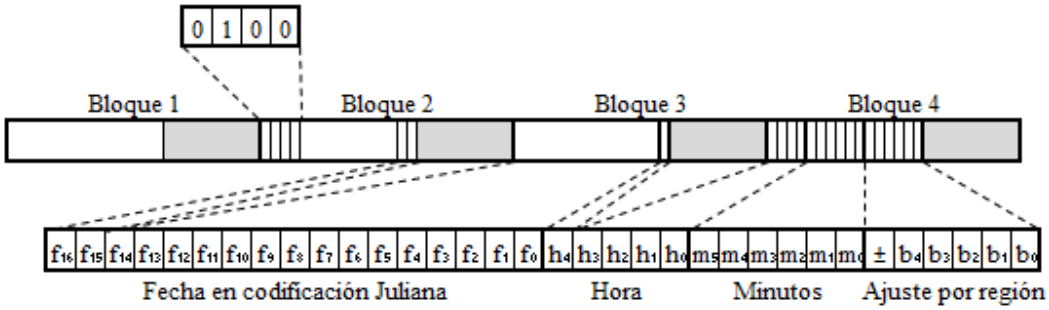
E. HORA Y FECHA (CT)

Cuando los cuatro bits más significativos del segundo bloque de un grupo son 0100, se está transmitiendo información acerca de la hora (en formato de 24 horas) y fecha actual. La codificación se hace según la estructura que se muestra en la Figura 4, y este grupo se caracteriza por enviarse una vez cada minuto. Es importante mencionar que la fecha se transmite según su representación Juliana (Elektor Electronics, 1991).

Para el desarrollo de esta investigación, se trabajó únicamente con la información correspondiente a la hora y minutos que este grupo contiene. Aunque el ajuste por región es un recurso que puede ser muy útil en territorios extensos, Guatemala no es un país donde este recurso sea necesario.

[6] Figura adaptada de (European Comitee for Electrotechnical Standrization (CENELEC), 2002)

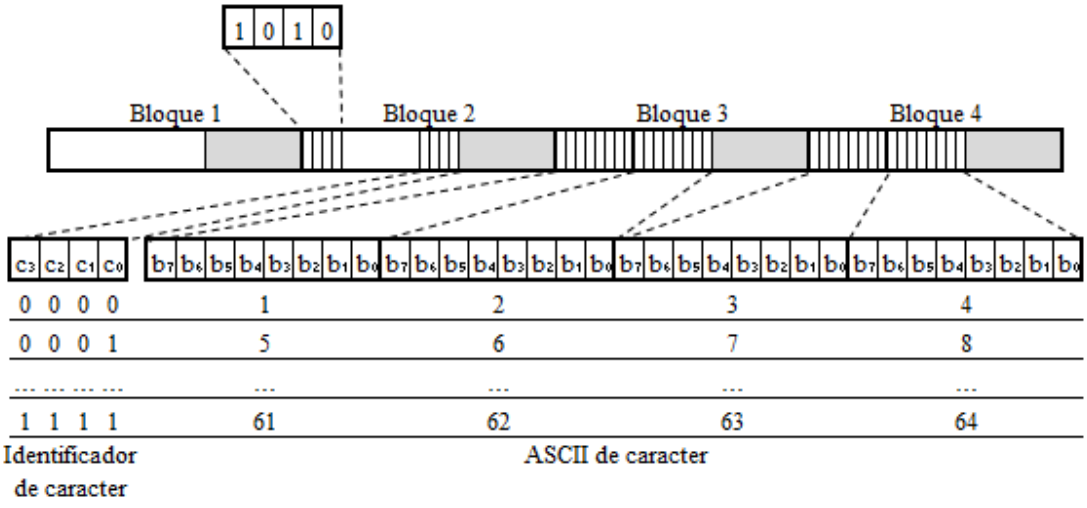
Figura 4. Estructura de los grupos de datos que contienen información de la hora y fecha (CT)[7]



F. RADIOTEXT (RT)

El protocolo *RDS* tiene la opción de transmitir mensajes de texto de 64 caracteres de largo, mediante el servicio de *RadioText*. Al igual en el caso de nombre de programa, son necesarios más de un grupo de datos para completar el mensaje. En este caso se necesitan 16, y los ASCII de los caracteres se encuentran en la parte alta y baja de los bloques 3 y 4. La identificación de su posición dentro de la cadena la denotan los cuatro bits menos significativos del bloque 2, tal como lo muestra la Figura 5. Los bits de identificación correspondientes a este servicio son 1010 (Elektor Electronics, 1991).

Figura 5. Estructura de los grupos que contienen caracteres que forman el *RadioText* (RT)[8]



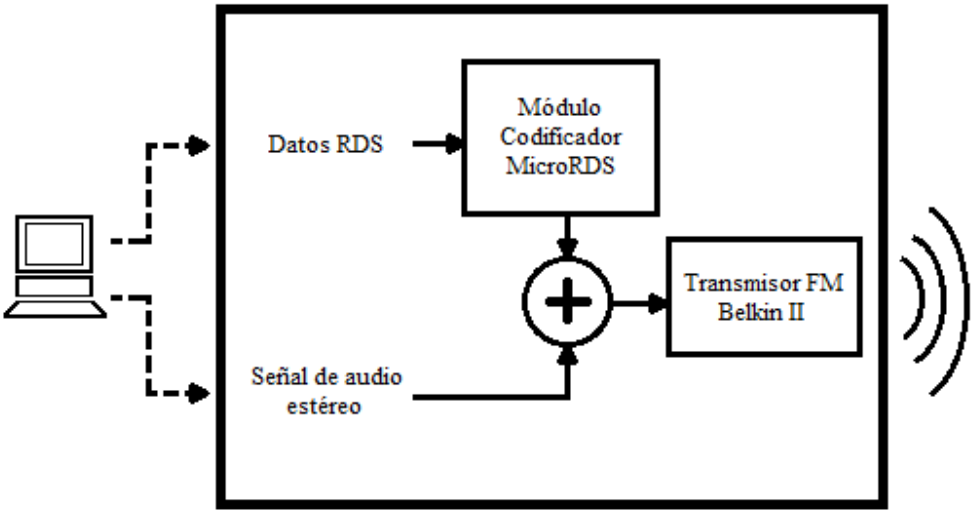
[7] Figura adaptada de (European Comitee for Electrotechnical Standrization (CENELEC), 2002)
[8] Figura adaptada de (European Comitee for Electrotechnical Standrization (CENELEC), 2002)

V. METODOLOGÍA

A. TRANSMISOR

Se armó un transmisor de audio FM de bajo alcance capaz de comunicarse mediante el protocolo *RDS*. Su función será únicamente el de proporcionar las señales correspondientes para realizar las pruebas del receptor, utilizando los estándares oficiales del protocolo que las empresas radiodifusoras usan actualmente. Un esquema general del dispositivo se muestra en la Figura 6.

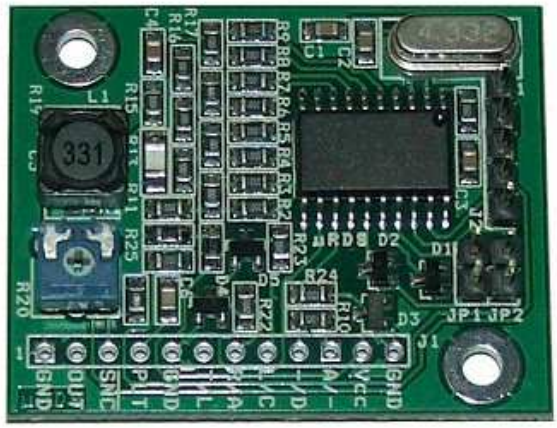
Figura 6. Diagrama de bloques del dispositivo transmisor FM+*RDS*[9]



La transmisión de datos *RDS* se realiza mediante el protocolo *RS-232*, y la señal de audio a través de la salida de audífonos de la computadora. La primera es acondicionada a niveles TTL mediante el circuito correspondiente del *MAX232*, para luego ser llevada hacia el módulo codificador *MicroRDS* (Figura 7) basado en el chip *MRDS1322*. Esta tarjeta, producida por la empresa checa *Pira*, está diseñada para codificar los mensajes según los requerimientos del *RDS*, produciendo una señal de aproximadamente 1.2Vpp a una frecuencia modulada justo en la subportadora de 57kHz. Esta señal es multiplexada con la proveniente del audio estéreo de la computadora, y esta sería enviada a la siguiente fase.

[9] Elaboración propia

Figura 7. Módulo *MicroRDS*, producido por la empresa *Pira*[10]



Para transmitir estas señales hacia las frecuencias usuales de las radiodifusoras (entre 87 y 108MHz) se utilizó el transmisor FM Belkin II (Figura 8). Popularmente conocidos como "Transmisores MP3", aparatos como este tienen como propósito original el modular señales de audio hacia alguna de las frecuencias mencionadas anteriormente, para así, escucharlas en equipos de radio de automóviles. Este dispositivo en particular está basado en el chip *BH1415F*, y aunque no está diseñado para transmitir señales *RDS*, su arquitectura interna no interfiere con el protocolo. Es una solución práctica para las necesidades que se tenían para el desarrollo de esta investigación.

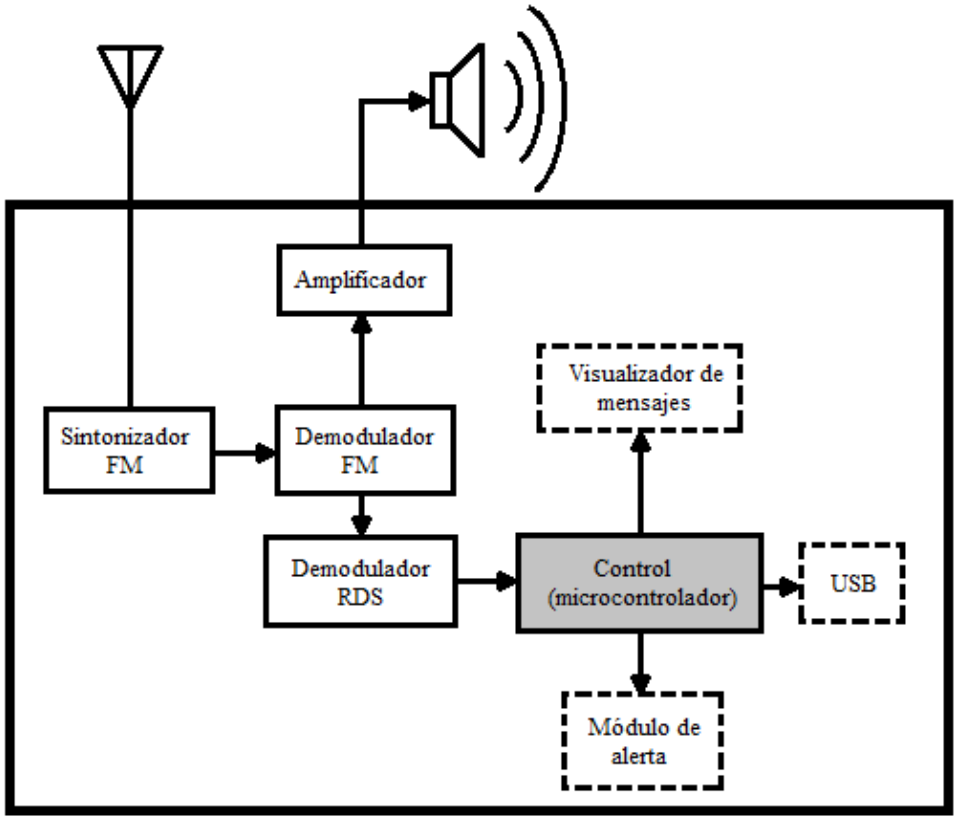
Figura 8. Transmisor FM Belkin II[11]

[10] Figura tomada de http://www.pira.cz/rds/show.asp?art=micrords_encoder
[11] Figura adaptada de http://www.belkin.com/IWCatProductPage.process?Product_Id=606060#

B. CONSIDERACIONES GENERALES DEL RECEPTOR

Para facilitar la realización del dispositivo receptor FM se decidió dividir el problema en partes, que se representan en la Figura 9. Las flechas que conectan cada uno de los bloques representan la dirección del flujo de información, y los recuadros punteados representan los módulos periféricos con los que el usuario podrá tener contacto. Se detallarán cada una de estas fases más adelante.

Figura 9. Diagrama de bloques del dispositivo receptor FM+*RDS*[12]



Dado que entre los objetivos del trabajo se propuso optimizar costos, en cuanto a componentes y construcción, se tomaron las medidas correspondientes para la elección de los elementos a utilizar. Sin embargo, se procuró que el dispositivo final fuera perfectamente funcional, al mismo tiempo que fuera lo más compacto posible. Por estas razones, se tomó especial atención al diseño y realización del circuito.

Aunque el dispositivo receptor tendría la posibilidad de ser sintonizado a la frecuencia que uno deseara, por motivos de prueba se decidió fijarla a 88.3MHz durante el desarrollo del trabajo. Sin embargo, el

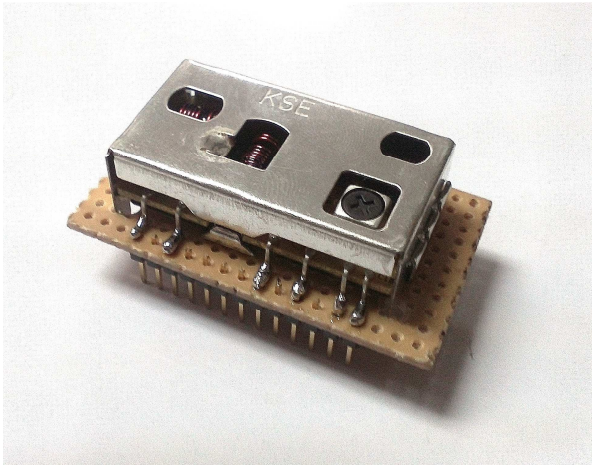
[12] Elaboración propia

dispositivo final sería capaz de decodificar los mensajes de cualquier estación que contara con señal *RDS*, ya que respetaría los estándares de este protocolo.

Cabe resaltar que se escogieron niveles de voltaje para el circuito receptor de 9 y 5V, según fuera necesario para cada módulo. Se escogieron estos valores ya que dan la facilidad de conseguirse con una batería de 9V, y un regulador de 5V de baja potencia como el *7805*. Esto contribuiría a mantener las dimensiones lo más compacto que se pudiera.

1. Sintonizador, demodulador y amplificador FM. Para facilitar la sintonización de la señal de radio en el dispositivo se utilizó el componente *Front End TRFD1H* (Figura 10). Este es un circuito tanque analógico, formado por componentes discretos como bobinas y capacitores en su interior, que puede ser controlado utilizando las señales adecuadas en sus terminales. Este componente es catalogado como un oscilador controlado por voltaje (o VCO por sus siglas en inglés), ya que permite alcanzar la frecuencia de resonancia que se busca para la sintonía de las estaciones, variando únicamente, un voltaje DC entre 1.2 y 6.8V aproximadamente. Se ajustaron los las magnitudes de los componentes externos que la hoja de datos de este componente solicitaba para poder hacerlo funcionar a los niveles de voltaje que se utilizarían para el circuito receptor (en este caso, 9V).

Figura 10. Componente *Front End TRFD1H*[13]

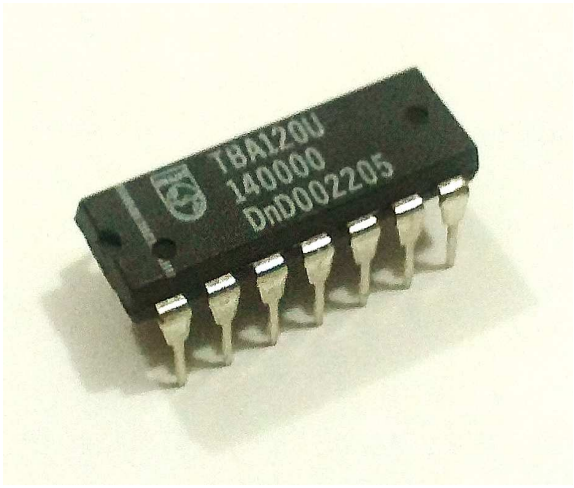


En cuanto al demodulador FM se utilizó el componente *TBA120U* (Figura 11). Este componente tiene la característica de no filtrar las frecuencias de 57kHz de la señal proveniente del sintonizador, por lo que puede ser utilizada para la demodulación del *RDS*. Nuevamente se ajustaron los valores de los componentes presentados en la hoja de datos para ajustarlos a los niveles de voltaje que se utilizarían en el circuito.

[13] Componente *TRFD1H* adaptado para usarse en *protoboard*

Originalmente el componente está diseñado para trabajar a 12V DC, y como consecuencia de reducir su voltaje, disminuyó también la calidad de la señal de audio del dispositivo (en cuanto a nitidez y volumen). Sin embargo, la señal a 57kHz no se vio afectada considerablemente, y pudo ser trabajada por el demodulardor *RDS* posteriormente.

Figura 11. Componente *TBA120U*[14]

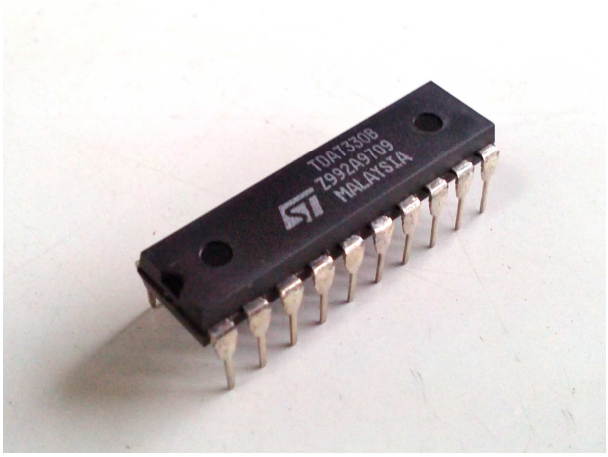


Por último, se utilizó el componente *LM386N* para amplificar la señal de audio proveniente de la fase anterior, para luego ser reproducida por una bocina de 8Ω. La función primordial de este módulo fue la de aprovechar el sonido como una guía para la correcta sintonización de la frecuencia a la que se deseaba trabajar. Es por ello se le otorgó la posibilidad de enviar audio al aparato transmisor que se describió anteriormente.

2. Demodulador *RDS*. Para esta fase del procedimiento se utilizó el componente *TDA7330B* (Figura 12). Este componente procesa una señal demodulada FM para obtener dos resultados: una señal de reloj constante a 1187.5bits/s y otra de datos síncronos. Ambas señales, que varían digitalmente entre 0 y 5V DC, son enviadas al microcontrolador para su proceso correspondiente. Este integrado filtra y decodifica las señales de 57±2.5kHz (Figura 13) según la codificación Manchester, a una tasa de 1187.5bits/s. El resultado de este proceso se complementa con la señal de reloj producida también por el componente, ya que cada flanco de caída de esta segunda señal, indicará que un nuevo bit está listo para ser procesado.

[14] Componente utilizado para el proceso de demodulación FM

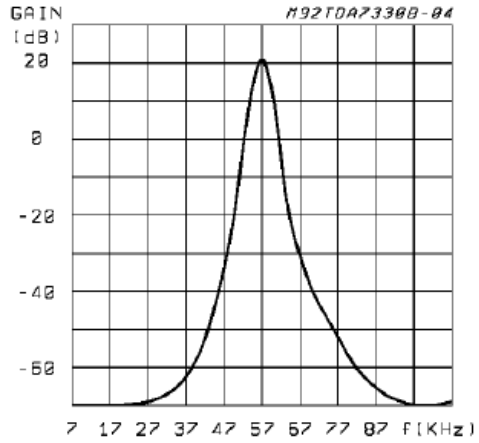
Figura 12. Componente *TDA7330B*[15]



Cabe mencionar que aunque la frecuencia sintonizada no cuente con señal *RDS* propia, este componente producirá datos aleatorios. Esto se debe a que el circuito integrado demodulador FM no filtrará las señales a frecuencias de 57kHz, y cualquier señal que este detecte lo procesará como si fuera realmente *RDS*. Por ello, es imprescindible el manejo adecuado de los datos por parte del microcontrolador, ya que sólo respetando las normativas del protocolo podrán identificarse los datos válidos que deben tomarse en cuenta.

En la Figura 14 se pueden identificar las señales de reloj y de datos provenientes de este circuito integrado, tal como las recibiría el módulo de control más adelante.

Figura 13. Ganancia vs. Frecuencia del *TDA7330B*[16]



[15] Circuito integrado utilizado para la demodulación *RDS*
[16] Figura tomada de la hoja de datos del *TDA7330B*

Figura 14. Representación de la señal de datos y de reloj producida por el *TDA7330B*[17]

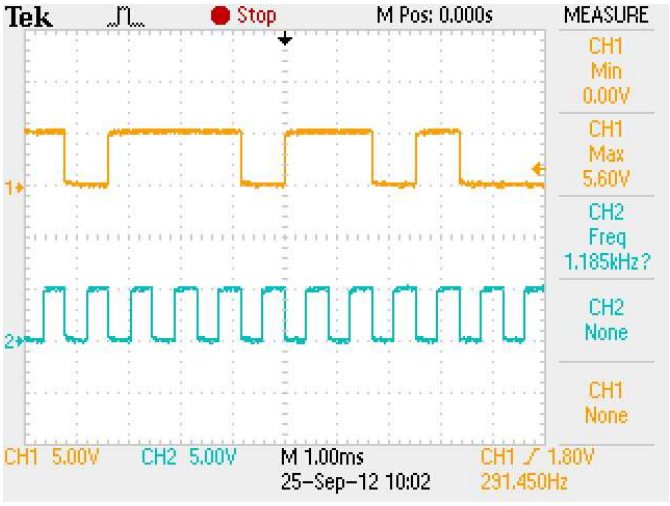
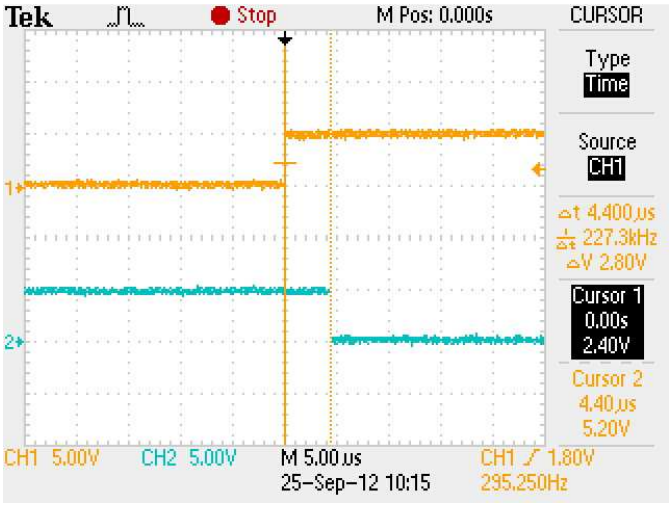


Figura 15. Tiempo de acondicionamiento de señal disponible entre nuevo bit y señal de reloj[18]



Como puede observarse en la Figura 15, el *TDA7330B* proporciona un espaciamiento de aproximadamente 4.4µs entre el cambio de bit en la secuencia de datos, y la señal de reloj que lo evidencia. Esto es de gran ayuda para el procesamiento de información, ya que si estos dos cambios sucedieran en el mismo instante, se podrían identificar datos falsos a causa de no estar completamente estabilizados al momento de tomar la muestra.

[17] Señales obtenidas del osciloscopio en pruebas de laboratorio
[18] Señales obtenidas del osciloscopio en pruebas de laboratorio

3. Visualizador de mensajes. Se utilizó para esta fase una pantalla LCD de tipo COG (chip on glass) de la marca Winstar Display, basado en el circuito integrado *NT7605* (Figura 16). Este tiene la capacidad de ser operado mediante dos bits de control y cuatro u ocho bits de comunicación paralela. Se optó por manejarlo con cuatro bits, ya que de esta manera se reduciría el número de pines del microcontrolador dedicados a esta tarea y así disminuir los costos del mismo. Esta pantalla permite visualizar un conjunto de 2x16 caracteres alfanuméricos, formados cada uno por una matriz de 7x5 puntos.

Figura 16. Pantalla tipo COG utilizada en el trabajo[19]

C. MÓDULO DE CONTROL

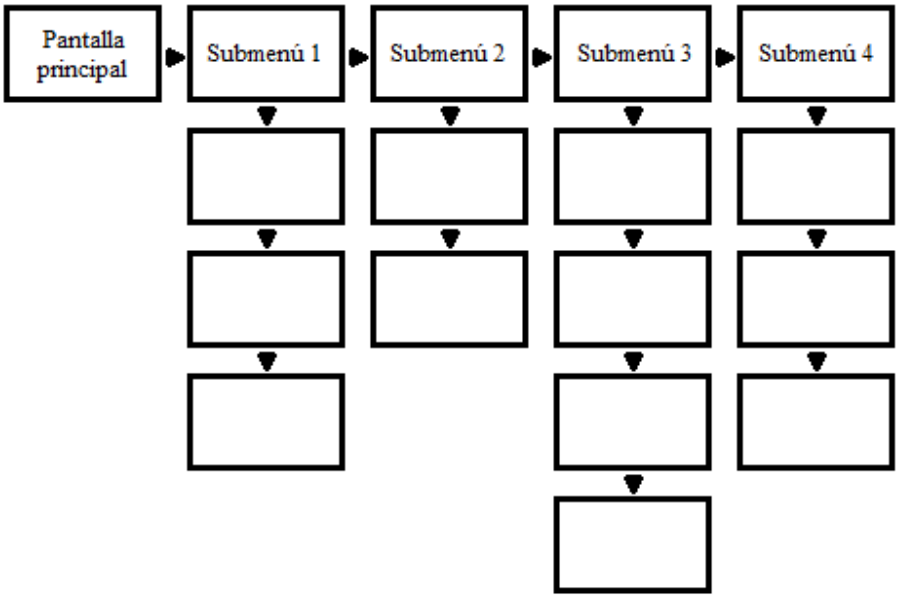
Este módulo permitió decodificar la señal *RDS* de la estación sintonizada según los estándares que rige el protocolo. Se optó por utilizar el microcontrolador PIC16F88 de Microchip por sus características en cuanto a memoria *FLASH*, *EEPROM*, *RAM*, empaquetado y funciones periféricas como el *UART*. Algunas de las tareas que cumple son:

- Recibir el flujo de información proveniente del bloque demodulador descrito anteriormente.
- Verificar la validez de la trama, de acuerdo al método de corrección de errores que el protocolo *RDS* indica.
- Almacenar correctamente la información recibida.
- Controlar la información que se muestra en la pantalla de visualización.
- Manejar las funciones periféricas como la comunicación con la computadora, el estado de los botones y las señales de alertas.

Se decidió utilizar una estructura de menús como la presentada en la Figura 17. Cada submenú (que a partir de este momento se le llamará “página”) contiene información sobre algún determinado tópico: noticias, deportes, estado de tránsito, etc. Al escoger uno de ellos, se podrá navegar por los últimos mensajes correspondientes a ese tema, partiendo del más reciente hacia el más antiguo (considerando una longitud máxima de la fila *PEPS*). Para facilitar la navegación se utilizaron tres botones: “izquierda”, “derecha” y “más” (este último se utilizaría para el movimiento vertical en el esquema de la Figura 17).

[19] Figura tomada de http://www.mikroe.com/products/view/277/various-components/

Figura 17. Estructura de acceso a menús[20]



El dispositivo final debe informar al usuario cuando un nuevo mensaje es recibido a través del módulo de alarma. Además, se decidió mostrar el mensaje nuevo en la pantalla sin importar la página en la que se encuentre el usuario previo a la recepción. Sólo así el usuario estará consciente de qué tipo de información fue recibida al momento en el que la alarma fue activada. Este nuevo mensaje es colocado inmediatamente en el tope de la fila de su página, para poder navegar naturalmente por los menús luego de leer el nuevo mensaje.

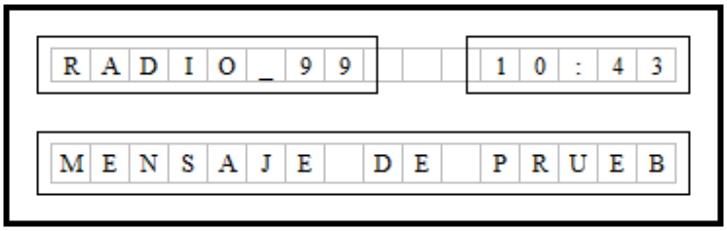
Si la fila de la página correspondiente ya se encuentra llena al momento en el que se recibe un nuevo mensaje, el último será desechado.

Como se mencionó en el marco teórico, se utilizarán tres servicios importantes que brinda el *RDS*: el nombre de programa, la hora y fecha, y el *RadioText*. Este último será el encargado de manejar los mensajes de las páginas, por lo que estos serán de un máximo de 64 caracteres.

[20] Elaboración propia

1. Estructura de pantallas. Existen tres posibles tipos de mensajes que pueden ser mostrados en la pantalla, dependiendo de la posición en el que se encuentre el usuario según el esquema de la Figura 17. El primero es el correspondiente a la pantalla principal, y muestra la información más general de la estación de radio. Un ejemplo de la información que podría encontrarse en esta pantalla se muestra en la Figura 18.

Figura 18. Estructura de la pantalla principal[21]



El recuadro izquierdo de la primera fila incluye el nombre de la estación de radio que está sintonizada, y es actualizada mediante el servicio de "Nombre de programa" del *RDS*. Usualmente esta cadena de 8 caracteres es la que se envía con mayor frecuencia por parte de las estaciones de radio, por lo que idealmente no tardará mucho tiempo en ser refrescada al momento de iniciar el proceso de decodificación. En el extremo derecho se muestra la hora actual en formato de 24 horas, y es actualizada por el servicio de "Hora y Fecha" del *RDS* una vez por minuto.

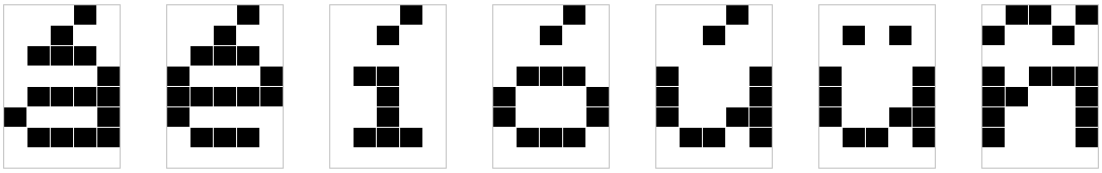
Por último, la fila inferior contiene los mensajes *RadioText* que no pueden ser catalogados en ningún submenú disponible. Este puede ser utilizado para cualquier aplicación que la estación de radio desee: publicidad, identificación de nombres y artistas de canciones, nombre del programa actual de la radio, o información general. El mensaje puede ser de un máximo de 64 caracteres, por lo que si su longitud es mayor a los 16 (espacio disponible en la pantalla) se deberá realizar un corrimiento paulatino hasta que todo el mensaje haya sido mostrado. Se decidió dejar los primeros 16 caracteres del mensaje fijos durante un segundo antes de iniciar el corrimiento, el cual, se hará carácter por carácter dejando una diferencia de 250ms entre cada uno. Esto permite que la lectura del mensaje sea natural y amigable con el usuario.

Se dispuso que este dispositivo tuviera la habilidad de mostrar los caracteres especiales importantes en el idioma español: las tildes, la letra "ñ" y la "u" con diéresis. Aunque el protocolo *RDS* está diseñado para poder transmitir los caracteres ASCII de estas letras, el circuito integrado que controla la pantalla no cuenta con estos caracteres en su base de caracteres pre-guardados. Por esta razón se codificó cada uno de ellos manualmente, y fueron agregados a la memoria interna del chip de la COG. Debido a sus características

[21] Elaboración propia

propias, se tiene la posibilidad de agregar únicamente 7 caracteres nuevos a su base de datos. En base a ello se escogieron los presentados en la Figura 19.

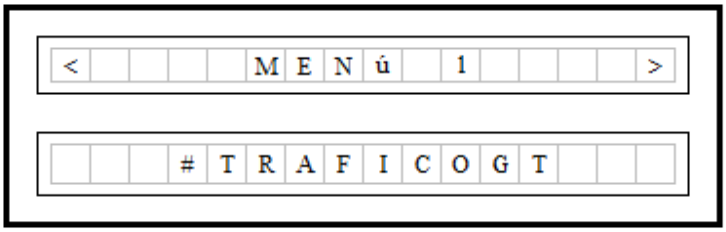
Figura 19. Caracteres especiales agregados al *NT7605*[22]



Cuando un carácter de este grupo es identificado en un *RadioText* (o en el Nombre de programa), inmediatamente se manda la instrucción correspondiente para que uno de estos sean mostrados. Cabe resaltar que no se implementaron estos caracteres especiales en mayúsculas, por lo que si uno de ellos es identificado, se mostrará su carácter especial correspondiente.

El siguiente tipo de información que puede ser mostrada es el identificador de página, como se muestra en la Figura 20. En la fila superior se muestra el número de menú que corresponde a la página, y en la inferior el tipo de información que se manejará en el grupo.

Figura 20. Estructura de los títulos de página[23]



Como puede deducirse en la figura anterior, se consideró como una buena opción, el utilizar la misma cuenta de *Twitter* de las empresas radiodifusoras como la fuente de los mensajes *RadioText*. Ello representa dos ventajas directas:

- La estación de radio no debe invertir el tiempo de un colaborador en mantener actualizada la información en este canal, porque ya se estaría haciendo desde su propia cuenta de *Twitter*.

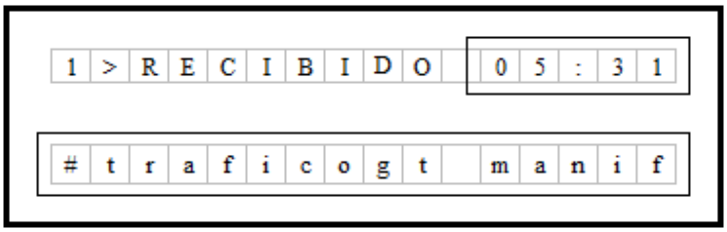
[22] Figura adaptada de la hoja de datos del *NT7605*
[23] Elaboración propia

- La identificación por parte del receptor del tipo de información que contienen los mensajes *RadioText* se simplifica de gran manera, ya que se reduce a identificar ciertas palabras clave en su contenido (en forma de *hashtags*, como "*#traficogt*").

El identificador del grupo se muestra en el título de página, y todos los mensajes en este grupo tendrán la característica de contar con esa palabra en su contenido.

Finalmente, la otra manifestación que puede tener la pantalla del dispositivo es la que muestra un mensaje perteneciente a un submenú, como se muestra en la Figura 21.

Figura 21. Estructura de la representación de un mensaje del submenú "*#traficogt*"[24]

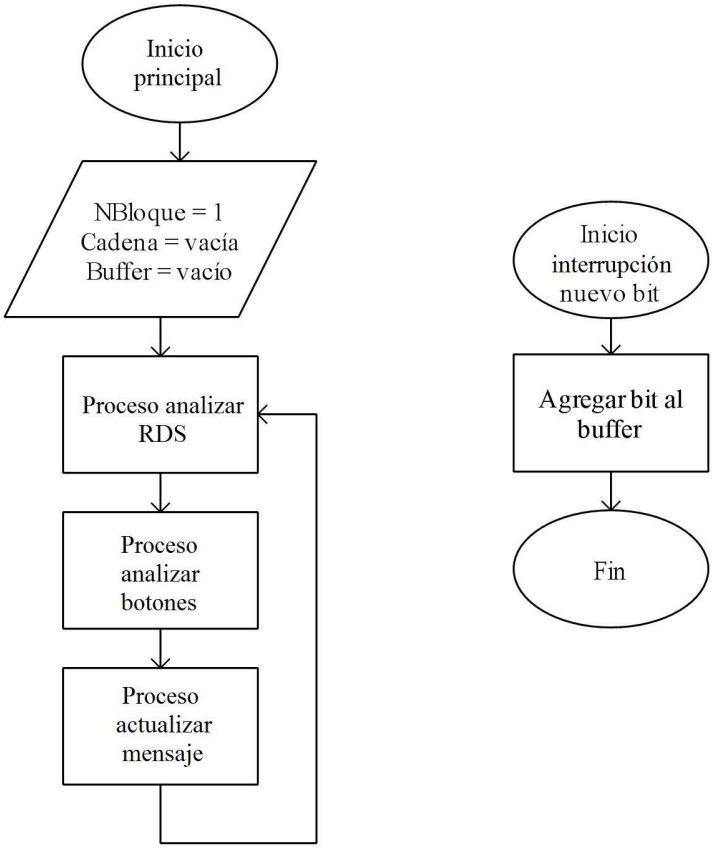


La primera posición de la línea superior identifica el número de mensaje del menú, los cuales son ordenados de forma cronológica. Cuando un nuevo mensaje es agregado a la fila, estos identificadores son actualizados para el resto de mensajes en la página. En la esquina superior derecha se encuentra la hora a la que el mensaje que está siendo desplegado fue recibido, según el servicio de "Hora y fecha" del *RDS*. Por último, en la fila inferior se muestra el mensaje correspondiente, con el mismo tipo de corrimiento que se mencionó para la pantalla principal.

2. Algoritmo. De manera muy general, las funciones a realizar por el módulo de control pueden ser resumidas en tres procesos: el de decodificar los mensajes *RDS*, el de verificar el estado de los botones y el de mantener actualizados los textos en la pantalla de visualización. Estas son las tres tareas más importantes que el ciclo principal del algoritmo del microcontrolador realiza, como puede observarse en el diagrama de flujo de la Figura 22.

[24] Elaboración propia

Figura 22. Diagramas de flujo del ciclo principal y el proceso de interrupción[25]



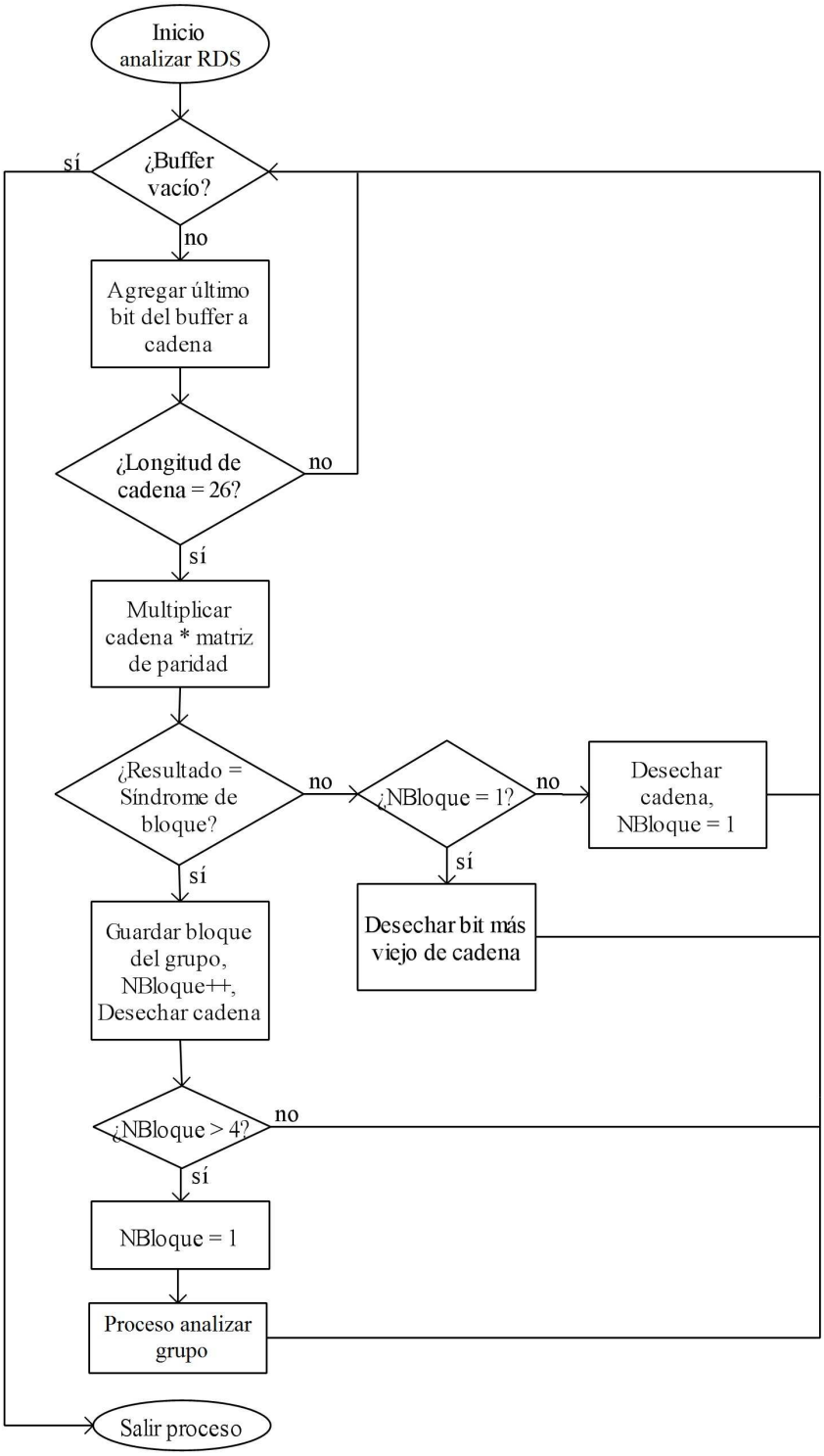
En la figura se muestra también el proceso de interrupción que ocurrirá al recibir un nuevo bit del demodulador *RDS*. Este puede ocurrir en cualquier punto del flujo de programa, y su impacto se hará más notable al examinar el proceso de análisis del *RDS*. Pueden observarse tres variables importantes que serán usadas a lo largo del algoritmo:

- NBloque: Número de bloque que se está analizando actualmente. Para completar un grupo, deben recibirse cuatro bloques válidos según las características descritas en el marco teórico (por lo tanto, esta variable podrá oscilar únicamente entre 1 y 4).
- Cadena: Contendrá el listado de un máximo de 26 bits, y será la variable que tendrá que completarse para iniciar el proceso de verificación de bloque.
- Buffer: Dado que el proceso de interrupción puede ocurrir en cualquier punto del programa, es necesaria una variable de este tipo que almacene los bits recibidos para su análisis posterior.

En la Figura 23 se presenta el diagrama de flujo del proceso de analizar *RDS*, que utiliza estas tres variables descritas.

[25] Elaboración propia

Figura 23. Diagrama de flujo del proceso de analizar *RDS*[26]



[26] Elaboración propia

Para evitar el desbordamiento del buffer (*i.e.* pérdida de información), se decidió mantener en el proceso de análisis de *RDS* hasta que el buffer se haya vaciado. Es posible asegurar que el proceso es lo suficientemente eficiente como para que no se produzca un ciclo infinito dentro de este mismo proceso. Si el buffer no está vacío, se agregará su bit más antiguo a la cadena de datos.

Una vez se consigan 26 bits de la cadena, será posible verificar si corresponde a un grupo válido. Para esto, se multiplica este vector $\vec{x}$ (de 1×26 de dimensión) por la matriz de paridad descrita en el marco teórico *H* (de 26×10). Para calcular este vector de 1×10, denominado $\vec{s}$, se siguieron los siguientes pasos:

- Como se puede observar en la matriz de paridad (Expresión 1), el grupo conformado por sus primeras diez filas y diez columnas corresponden a la matriz identidad I_{10}. Si se analiza el proceso de multiplicación, se puede deducir que este conjunto posee la tarea de identificar la presencia o ausencia de los diez bits más significativos del vector $\vec{x}$. En caso que alguno de ellos tenga el valor de 1, su posición correspondiente en el vector $\vec{s}$ deberá inicializarse en uno

  Por lo tanto, como primer paso en la multiplicación matricial, se inicializó el vector $\vec{s}$ con la estructura que posee el vector $\vec{x}[26:17]$ (siendo estos sus diez bits más significativos). Con esto, se evita la tarea de multiplicar $\vec{x}$ con los primeros diez valores de cada columna de *H*.

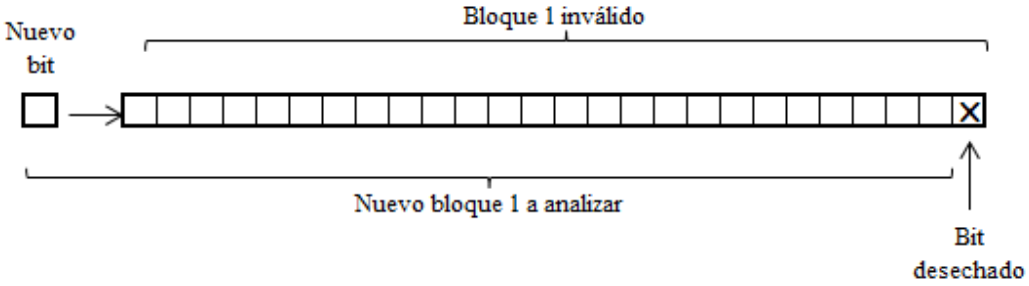
- Para completar el proceso, se deben multiplicar los 16 bits restantes de $\vec{x}$ (denominados b_n, con $16 \geq n \geq 1$) con la submatriz $H[11:26;\ 1:10]$, y adicionar estos resultados al estado actual de $\vec{s}$ prefijado en el paso anterior. Para simplificar esta tarea, se realizó lo siguiente:
  - En caso que el bit *n* de $\vec{x}$ sea 0, su multiplicación matricial con la columna correspondiente en *H* será 0. Por lo tanto, si se presenta este caso, el valor de la posición *n* de $\vec{s}$ no se cambiará al inicializado en el paso anterior.
  - Por el contrario, si el bit *n* de $\vec{x}$ es 1, su multiplicación matricial con la columna correspondiente en *H* producirá la suma de cada uno de los bits que la conforman. Este resultado, será adicionado al valor con el que $\vec{s}$ fue inicializado en la posición *n* correspondiente previamente.

    La suma binaria que se realiza en este caso se efectuó con la función lógica “o exculsivo” (*XOR*).

Al completar este proceso, se obtendrá el vector $\vec{s}$ correspondiente al grupo de 26 bits que se está analizando. Dependiendo del número de bloque que se esté esperando en el momento que fue iniciado este proceso, se compara el vector $\vec{s}$ con su síndrome correspondiente (mostrados en el Cuadro 3). Si estos dos no coinciden, puede suceder una de estas dos opciones:

- Si el número de bloque que se está esperando es el primero, no es recomendable desechar la cadena completa ya que simplemente no se ha encontrado la sincronía de los datos. Si este fuera este el caso, se debe desechar únicamente el bit más viejo de la cadena, para que al recibir uno nuevo sea posible verificar la validez de este grupo.

Figura 24. Proceso que se debe seguir al identificar un bloque 1 inválido[27]



- Para cualquier otro caso (es decir, si se está esperando el bloque 2, 3 o 4 del grupo) debe desecharse la cadena completa y el resto de bloques de ese grupo que ya hayan sido almacenados. Esto obedece el método de corrección de errores que el protocolo indica y evita que se reciba información errónea o incompleta.

Por otro lado, si el bloque que se recibe es válido, debe almacenarse e incrementar NBloque, para esperar el siguiente hasta completar el grupo. Finalmente, cuando todos los bloques del grupo se hayan conseguido, se volverá a esperar un nuevo bloque 1 válido (NBloque = 1) y será posible procesar el grupo completo según el tipo de información que el mismo contenga.

Para identificar el tipo de información que contiene el grupo se revisarán sus cuatro bits de control, representados en las Figuras 3, 4 y 5. Si estos coinciden con los que el dispositivo está capacitado para procesar (PS, CT y RT) se obtendrá la información que contienen según la estructura identificada y se almacenará según sea el caso.

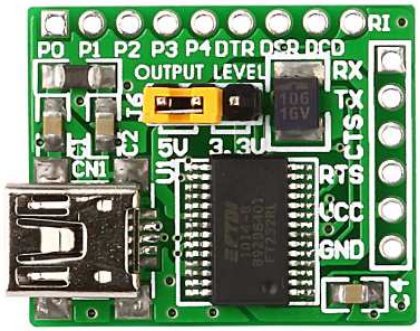
D. COMUNICACIÓN CON LA COMPUTADORA

Para poder configurar el tipo de mensajes que el usuario desee recibir, se implementó la comunicación vía con la computadora vía *USB*. Se añadió al dispositivo el módulo *USB UART* producido por

[27] Elaboración propia

Mikroelectronika (Figura 25), para que mediante un cable *USB–microUSB* pudiera ser conectado con la computadora.

Figura 25. Módulo *USB UART*[28]



Esta tarjeta brinda la facilidad de poder utilizar el puerto *USB* mediante la comunicación UART, facilitando el intercambio de información en dos vías entre la computadora y el dispositivo. Se consideró desarrollar la tarea de comunicar el PIC con la computadora directamente por *USB*, utilizando algún microcontrolador que pudiera ser configurado como *device USB*; sin embargo las siguientes razones confirmaron que este protocolo no sería el más adecuado para esta aplicación (Axelson, 2009):

- La cantidad de información que se desea transmitir es muy baja (unos cuantos bytes de estado).
- No se necesita una velocidad de transmisión tan alta como las que ofrece el protocolo *USB* (puede ser hasta 5Gbps para el tipo 3.0).
- La función que se desea implementar con esta comunicación no justifica la implementación de este complejo protocolo.
- Es necesario contar con Identificador de Producto e Identificador de Vendedor para poder distribuir dispositivos que cuenten con habilidad.
- Por último, para contar con el derecho de utilizar esta tecnología, es necesario pagar una tarifa anual tan alta que incrementaría de gran manera los costos de mantener y distribuir un dispositivo de este tipo. Ello no concuerda con los objetivos de la investigación, por lo que se decidió considerar otras opciones.

El Módulo *USB UART* de Mikroelectronika brinda la comodidad de brindarle al dispositivo una conexión *microUSB*, por la cual se pudiera conectar a la computadora y manejarlo como si fuera un dispositivo que realmente se comunica mediante este protocolo.

[28] Figura tomada de http://www.mikroe.com/add-on-boards/communication/usb-uart/

Básicamente la información que se intercambian el microcontrolador y la computadora son los índices de los menús que se desean visualizar en el dispositivo. La aplicación que utiliza desde la computadora para esta función fue desarrollada en Microsoft Visual C# (Figura 26) y permite obtener la configuración actual del dispositivo, así como actualizarla según los intereses del usuario.

Figura 26. Interfaz de la aplicación para configurar los menús del dispositivo desde la computadora[29]

[29] Aplicación utilizada para la configuración de menús, realizada en Microsoft Visual C#

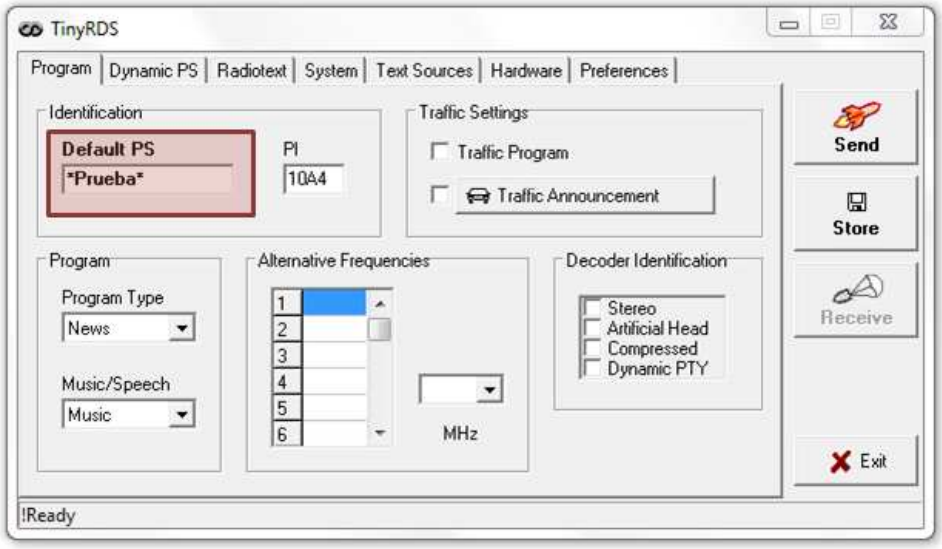
VI. RESULTADOS

A. TRANSMISOR

Para verificar si el transmisor funcionaba correctamente se utilizó la aplicación *TinyRDS* 1.0b, de la empresa *Pira*. Esta versión gratuita está diseñada especialmente para aficionados, a través de la cual se podrán controlar sus productos (como la tarjeta *MicroRDS* utilizada en este dispositivo). Cuenta con los servicios más populares del *RDS*, incluyendo los de interés para esta investigación.

La aplicación se comunica con la tarjeta a través del protocolo *RS-232*, como se mencionó anteriormente. La señal producida por esta última será "inyectada" en la señal de audio que se transmite por la frecuencia que indique el transmisor FM Belkin II. A continuación se muestran los resultados de este procedimiento, al fijar como Nombre de Programa la cadena **Prueba** en la aplicación (Figura 27), sintonizando la frecuencia 88.3MHz FM (Figura 28). Se utilizó como receptor un dispositivo independiente de esta investigación, el celular *LG Optimus Black* (Figura 29), para poder comprobar que el protocolo se respetara según los requerimientos reales.

Figura 27. Interfaz de la aplicación *TinyRDS*. Énfasis en el Nombre de Programa (PS)[30]



[30] Figura obtenida de la aplicación *TinyRDS*

Figura 28. Transmisor sintonizado a la frecuencia de 88.3MHz[31]

Figura 29. Mensaje de prueba enviado desde el transmisor hacia el celular *LG Optimus Black*[32]

Usando como receptor este celular, se determinó que el alcance del transmisor es de 3m sin obstáculos. Se obtuvieron mejores resultados al sintonizar frecuencias que no son usadas actualmente dentro del

[31] Fotografía del dispositivo transmisor, basado en el aparato *FM Belkin II*
[32] Pantalla del celular *LG Optimus Black*, utilizando la aplicación interna de radio FM

espectro FM, ya que se evita la interferencia de señales. Aunque el rango no es muy grande, fue suficiente para realizar las pruebas controladas con el dispositivo receptor elaborado.

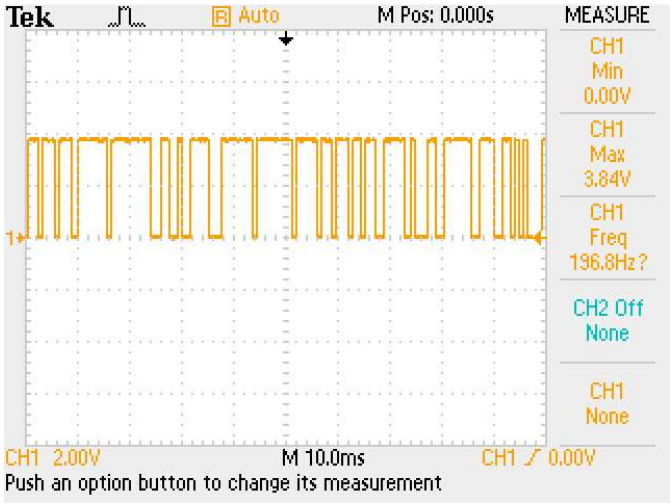
B. RECEPTOR

Al tener el control de ajustar el audio y las señales *RDS* de una frecuencia FM específica, se pudo comprobar fácilmente el correcto funcionamiento del dispositivo receptor. Se sabía qué se deseaba escuchar y qué mensajes se debían visualizar, por lo que se recurrió a verificar si cada una de las fases del dispositivo cumplía su función destinada. Sólo de esta manera se podría alcanzar el resultado final que se propuso. A continuación se presentan los resultados más importantes de cada una de ellas.

1. Sintonización. Se utilizaron dos recursos para ajustar la frecuencia de sintonización del dispositivo:

a. El circuito integrado *TDA7330B*, utilizado para el proceso de demodulación *RDS*, cuenta con una terminal que indica la calidad de la señal que está trabajando. Este pin producirá un voltaje de 5V constante sólo si la señal porta otra con frecuencia de 57kHz (aún si esta señal no corresponde con datos válidos *RDS*). En la Figura 30 se muestra el estado de esta terminal si la frecuencia sintonizada no cuenta con esta señal.

Figura 30. Ejemplo de indicador de calidad de señal del *TDA7330B*, a una frecuencia sin *RDS*[33]



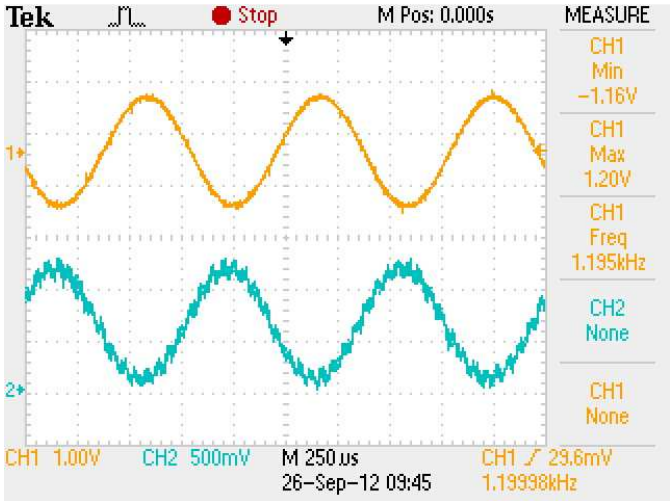
[33] Señal obtenida del osciloscopio en pruebas de laboratorio

b. Aunque el audio que se transmite de la fuente no es necesario para el funcionamiento del RDS, se aplicó para facilitar el proceso de sintonización. Este fue el primer parámetro que se utilizó, ya que mediante el sonido que el receptor reprodujera fue posible encontrar una primera aproximación de la frecuencia sintonizada.

2. Demodulación. Aunque ya se hubiera comprobado que el transmisor lograba incluir la señal *RDS* en la frecuencia a la que estuviera sintonizada, se debía verificar si esta misma era detectada por el receptor. El hecho que se reprodujera en el receptor el audio que se enviaba, no implicaba que en realidad este contara con la señal de datos de interés. Para corroborarlo, se envió desde la computadora un tono constante a 1200Hz, y se tomó una muestra de la forma que tendría esta señal en el transmisor (antes de inyectarle el *RDS*) y el receptor (luego del proceso de demodulación) para el caso en el que se utilizara el *RDS* o no.

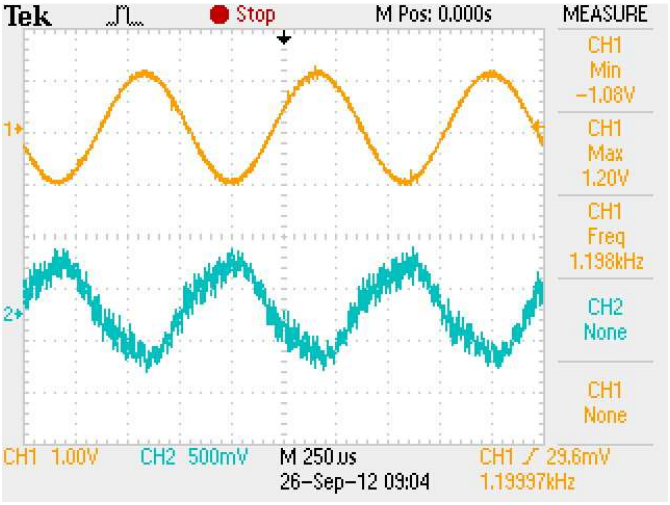
Las Figuras 31 y 32 muestran los resultados de estas mediciones. La señal superior para cada una de ellas representa la señal de audio sin *RDS* obtenida desde el transmisor, y la inferior cómo es percibida por el receptor. En la Figura 31 no se le inyecta la señal *RDS* en ningún momento, mientras que en la siguiente sí se hace.

Figura 31. Señal en el transmisor (arriba) y receptor (abajo) de un tono a 1.2kHz sin *RDS*, con *TBA120U*[34]



[34] Señales obtenidas del osciloscopio en pruebas de laboratorio

Figura 32. Señal en el transmisor (arriba) y receptor (abajo) de un tono a 1.2kHz con *RDS*, con *TBA120U*[35]



Cabe resaltar que aunque la señal percibida desde el receptor corresponde a la transmitida en forma invertida, esta característica no afecta al sonido que se reproduce por la bocina.

En ambos casos la señal del receptor cuenta con ruido, producto de la calidad de los componentes utilizados y del *protoboard*. Sin embargo, se puede observar que la segunda figura porta otra frecuencia más pronunciada que la primera. Dado que fue el único parámetro que se modificó, esta debería corresponder a la señal *RDS*. Esta suposición se fundamenta también con el indicador de calidad de señal del *TDA7330B*, el cual se mantuvo activo en el segundo caso. Con esto se pudo asegurar que la frecuencia sintonizada contaba en efecto con una frecuencia portada de 57kHz.

Sin embargo, hasta este punto no era posible asegurar que ella contendría datos *RDS* válidos, que pudieran ser decodificados por el módulo de control. El integrado *TDA7330B* produce dos señales, sin importar si se le ingresa una frecuencia sin *RDS*: una de reloj y otra de datos. Para comprobar si estas representaban datos válidos de este protocolo, se utilizó la aplicación gratuita *RDS Spy* (obtenida en https://rdsspy.com). Se ingresaron las señales producidas por el *TDA7330B* (Figura 34) a la computadora mediante la conexión a micrófono de la computadora. Desde la aplicación, se fijó la configuración de entrada de señal como lo muestra la Figura 33, para que fueran analizadas por el *RDS Spy*.

[35] Señal obtenida del osciloscopio en pruebas de laboratorio

Figura 33. Configuración de las conexiones a las señales del TDA7330B[36]

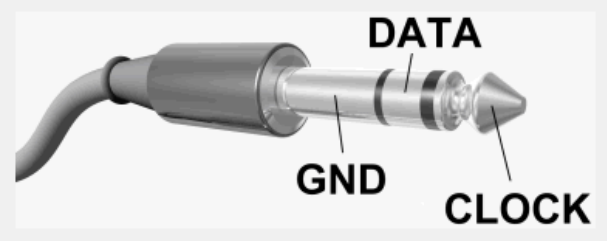
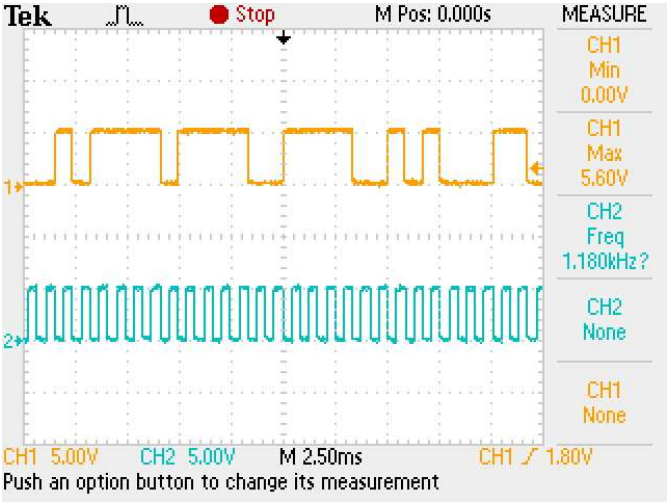


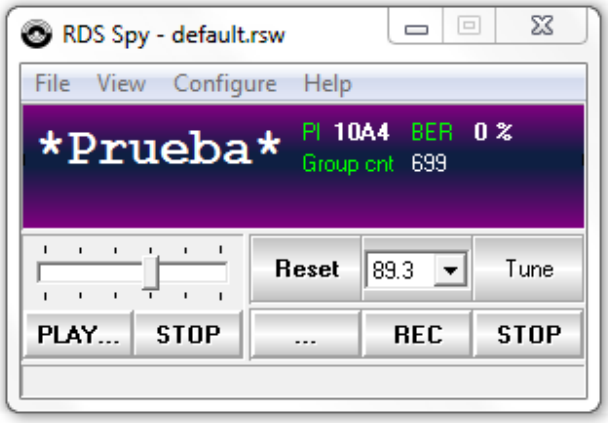
Figura 34. Señales producidas por el *TDA7330B*, que se decodificarían por el *RDS Spy*[37]



Esta aplicación decodifica los datos que se le ingresan según el protocolo *RDS*, y si se trata de información válida, muestra los resultados en pantalla. Tiene la capacidad de procesar todos los servicios que ofrece el *RDS* (incluyendo los desarrollados en esta investigación: el Nombre de Programa, Fecha y Hora, *RadioText*). Como se ve en la Figura 35, la decodificación de los datos producidos por el integrado sí corresponden a datos válidos *RDS* (dado que muestra la cadena **Prueba** como Nombre de Programa). Esto da la pauta que el módulo de control sí recibirá datos válidos cuando la frecuencia esté correctamente sintonizada.

[36] Figura tomada de la aplicación *RDS Spy*
[37] Señales tomadas del osciloscopio en pruebas de laboratorio

Figura 35. Interfaz del *RDS Spy*, mostrando el Nombre de Programa **Prueba**[38]



3. Módulo de control. Una vez cerciorado que la señal *RDS* proveniente del *TDA7330B* sí correspondiera a grupos válidos, se procedió a perfeccionar el código del módulo de control. Se mantuvo como base el diagrama de flujo de la Figura 23, pero se procuró realizar un código lo suficientemente eficiente para poder realizar la tarea de decodificación sin problemas. Algunas de las medidas que se tomaron fueron:

- Utilizar el oscilador interno más veloz que el microcontrolador ofrecía (8MHz).
- Utilizar un buffer de 32 bits de profundidad, que permite recuperar hasta 26.99ms de datos sin producir un desbordamiento.
- No salir del ciclo de validación de *RDS* hasta vaciar el buffer por completo.
- Evitar lecturas innecesarias de las memorias *EEPROM* y *FLASH* durante el desarrollo del programa.
- Utilizar funciones internas de MikroC Pro (compilador de lenguaje C de Mikroelectronika) para realizar tareas que de utilizar ciclos manuales hubieran sido menos eficientes.

Sin embargo, para el caso de las páginas que muestran *RadioText*, los procedimientos repetitivos de escritura a la pantalla COG produjeron un retraso acumulativo en el flujo del programa. Como se mencionó anteriormente, la forma de visualizar el mensaje completo se hizo mediante su corrimiento cada 250ms. Esta llamada constante al proceso de escritura hizo que el buffer no se vaciara con la rapidez necesaria para evitar su desbordamiento.

[38] Interfaz de la aplicación *RDS Spy*

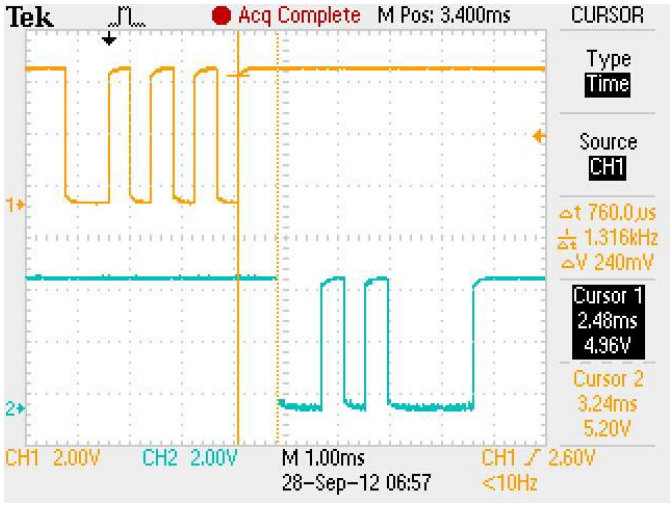
Como resultado, si el usuario se encuentra en una pantalla de refrescamiento constante de textos, el receptor perderá un bit aproximadamente cada 13.5s. Esto puede ocasionar la pérdida de sincronía, razón por la cual, se desecha el grupo que se está completando en ese momento al ocurrir este problema, y se reinicia el proceso de sincronización.

4. Comunicación con la computadora. La personalización del tipo de noticias que se deseen recibir se hace mediante este recurso. Desde la computadora, es posible realizar las siguientes dos opciones:

a. Obtener la configuración actual que posee el dispositivo: Mediante un byte de solicitud, 0xAA, la computadora pide el estado que en ese momento se maneja en el dispositivo. Este se envía en un solo byte, que indica los números de menús que se encuentran activos en ese momento.

En la Figura 36 se puede apreciar este procedimiento. La gráfica de arriba representa el byte de solicitud enviado por la computadora, según el protocolo *UART*. Si este lo identifica el microcontrolador como el valor 0xAA, retorna el byte de estado por la terminal correspondiente (gráfica inferior).

Figura 36. Byte de solicitud de estado (arriba) y de retroalimentación (abajo)[39]



El tiempo que transcurre antes de retornar el byte de estado depende completamente del microcontrolador. Para no alterar su proceso, este no será analizado hasta terminar un ciclo completo de su procedimiento principal. Sin embargo, este intervalo de tiempo no superará unos pocos milisegundos (para el caso de la Figura 36 fueron sólo 760us). De igual forma, la aplicación de la computadora está

[39] Señales obtenidas del osciloscopio en pruebas de laboratorio

configurada para no producir un mensaje de error de comunicación hasta que no hayan pasado 500ms sin recibir respuesta.

b. Luego de que el usuario haya elegido la configuración de menús que desee (Figura 26), podrá indicarle al dispositivo los cambios realizados. En este caso no se utiliza un byte de activación, simplemente se envía la información correspondiente desde la computadora. Si el dispositivo no puede identificar el byte como 0xAA, lo manejará como información de estado. Luego de actualizar sus configuraciones, automáticamente se posicionará en la pantalla principal, para que el usuario pueda navegar por los menús escogidos sin problema.

En ambos casos, la comunicación es iniciada desde la computadora. Esto desliga al microcontrolador de la función de mantener informada a la computadora de su estado, si no es que esta lo solicite. Con esto, se optimiza el proceso de decodificación del dispositivo y se evita la pérdida de información.

VII. ANÁLISIS DE RESULTADOS

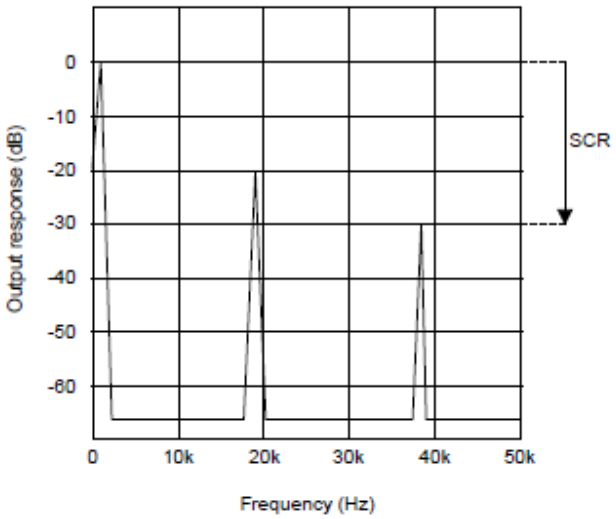
A. TRANSMISOR

Uno de los componentes más importantes de este dispositivo es el transmisor Belkin Tunecast II. Este cumple la función de transmitir tanto la señal de audio como los datos *RDS* que se le inyectan desde el *MicroRDS*. Sin embargo, el transmisor originalmente sólo presenta entradas para audio, y no está diseñado para transmitir otro tipo de datos, como los utilizados en el protocolo que se está desarrollando en esta investigación. La solución a esta contradicción se presenta en la Figura 37, tomada de la hoja de datos del circuito integrado que controla al dispositivo, el *BH1415F*.

.

Como se puede observar, de forma aproximada, su diseño le permite filtrar aquellas frecuencias que no sean múltiplos de 19kHz. Estas coinciden con las frecuencias piloto de la modulación FM, si se compara con su representación en la Figura 1. A medida que estas incrementan, su atenuación será cada vez más marcada (en la hoja de datos se especifica que para la frecuencia piloto de 38kHz se obtiene una atenuación típica del 30dB).

Figura 37. Diagrama del comportamiento de la señal de salida a diferentes frecuencias de modulación[40]



[40] Figura tomada de la hoja de datos del integrado *BH1415F*

De seguir con este comportamiento, aunque la figura no lo muestra explícitamente, se puede intuir que la frecuencia piloto de 57kHz (múltiplo de 19kHz) tampoco será filtrada del todo. Esta coincide con la frecuencia piloto de la transmisión *RDS*, que oportunamente no requiere de niveles muy altos de amplitud para ser correctamente decodificada.

Los resultados de este dispositivo fueron lo suficientemente buenos como para ser utilizado en el desarrollo del dispositivo receptor. Aunque la calidad de la transmisión no fue la mejor (cosa que se corroboró con un radio receptor independiente, contenido en el *LG Optimus Black*) y su alcance fue muy limitado (3m sin obstáculos), fue posible realizar todas estas pruebas sin mayores complicaciones.

B. RECEPTOR

En los siguientes incisos se detallan las consideraciones más importantes que se encontraron en cuanto a la realización del dispositivo receptor. En algunos casos se ratifica la validez del módulo realizado, y en otras se muestran sus debilidades. Por último, se detallan las mejoras que se le hicieron a estos módulos para obtener resultados más satisfactorios.

1. Módulo demodulador. Una de las razones por las que la señal de audio del dispositivo fue tan pobre (Figuras 31 y 32) se debe a que no se consideró un módulo demultiplexor. Inmediatamente después de demodular la frecuencia sintonizada por el circuito tanque, la señal fue amplificada y reproducida por la bocina. Sin embargo, al no aplicarle un filtro de ningún tipo, esta aún contaba con frecuencias parásitas que perjudicaban la nitidez del sonido (como, por ejemplo, la correspondiente al *RDS*). Como consecuencia, el audio que se produjo por el dispositivo fue de muy pobre calidad.

El hecho que esta señal no fuera demultiplexada, le permitió al demodulador *RDS* trabajarla sin problema, produciendo resultados acertados. Dado que la función del sonido producido por el receptor sirvió únicamente como un parámetro de sintonización, no se hizo mucho énfasis en mejorar este problema al momento del diseño preliminar del circuito. Sin embargo, como se detallará más adelante, sí se consideró este detalle al diseñar el circuito final del dispositivo receptor.

Por otro lado, el proceso de sintonización del dispositivo, mediante una señal analógica producida por un divisor de voltaje, produjo resultados poco precisos. Considerando que este proceso es muy subjetivo, ya que depende de la percepción del audio por parte del usuario, los resultados de la demodulación variaban según la precisión con la que se sintonizaba al dispositivo.

2. Comunicación con la computadora. Puede considerarse que la necesidad de conectar el dispositivo con una computadora para poder realizar cambios en su configuración reduce su nivel de practicidad. Es una tarea que fácilmente podría ser realizada desde el mismo dispositivo, considerando las siguientes observaciones:

- La cantidad de información actualmente se comunica a la computadora es muy baja.
- El dispositivo cuenta con todas las facilidades para poder realizar esta tarea: una pantalla de visualización, botones, y suficiente memoria.
- La computadora no le provee al dispositivo información nueva. Todos los datos que se comunican pueden ser conocidos desde el mismo aparato receptor.

Si se deseara mantener la comunicación con la computadora, podría considerarse utilizarla para incluir nuevos menús a la configuración del dispositivo por parte del usuario. Sin embargo, aun así, quien tiene el control de manejar el tipo de información que se transmite será la emisora radial. Si, por el contrario, fuera esta la interesada en proveerle un nuevo menú al receptor, podría considerarse el utilizar el servicio *RDS* de "Aplicación abierta de datos" (*ODA*). Es importante mencionar que para aplicarla, el receptor tendría que estar capacitado para manejar este tipo de servicio.

De cualquier forma, para esta aplicación en específico, es posible prescindir de la comunicación con la computadora por parte del receptor.

3. Pérdida de datos en el módulo de control. Como resultado de las llamadas constantes a escritura en la pantalla COG, se la pérdida de datos en el sistema de control cada 13 segundos aproximadamente. Sin embargo, esta pérdida no representa problemas significativos en el proceso de recolección de información, debido a las siguientes razones:

- El proceso de detección de errores es lo suficientemente robusto como para identificar errores de sincronía, por lo que se evita el procesamiento de datos inválidos.
- La velocidad de transmisión, 1187.5bits/s, permite recibir hasta 150 grupos completos antes de percibir una pérdida. Esto representa un 99.34% de eficiencia en el procesamiento de datos si el dispositivo está correctamente sintonizado, suficientemente alto para esta aplicación.
- El protocolo *RDS* se basa en la repetición constante de información. Por ejemplo, para el caso del Nombre de Programa (PS), es recomendable repetir los 4 grupos que conforman la cadena hasta más de una vez cada segundo. Por lo tanto, si se llegara a perder uno de los grupos que le conforman, se necesitaría menos de un segundo para recuperar este valor.

- Usualmente la velocidad con la que se cambian los mensajes que se transmiten es baja (aunque depende de la emisora).

El problema más grande que se podría presentar por esta pérdida de información, sucedería en el servicio de Hora y Fecha. Este grupo es enviado una única vez por minuto, por lo que si llegara a perder esta información, la hora desplegada en el dispositivo no cambiaría con el transcurso del minuto. Sin embargo, este problema tampoco es crítico, ya la probabilidad de que se pierda este grupo en específico es de 1:685 (0.15%) durante el minuto. Además, si llegara a suceder, únicamente, se tendría que esperar un minuto para volver a presentar en la pantalla la hora exacta que envía la emisora.

C. MEJORAS AL CIRCUITO

Se aplicaron algunos cambios al circuito receptor en base a los resultados, con la intención apegarse al máximo a los objetivos de la investigación relacionados con la minimización, optimización de costos, y calidad. Estos se hicieron presentes en el diseño de la placa final del dispositivo, y se mencionan a continuación:

1. Circuito receptor FM. Se utilizó el circuito integrado *TEA5767* (Figura 38), que cumplió la función de sintonizador y demodulador FM, representado en la Figura 9. Este chip posee un controlador interno capaz de manejar estas dos funciones utilizando pocos elementos externos. Su alimentación puede variar entre 2.5 y 5V DC, perfecto para esta aplicación. Es controlado a través de comunicación I^2C, que se manejará desde el módulo de control del receptor.

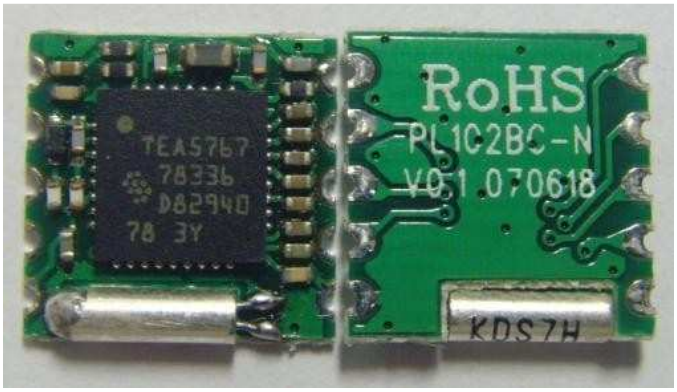
A continuación se mencionan algunas de las ventajas más grandes que se pueden resaltar de este módulo comparado con el anterior:

a. Dimensiones. El integrado tiene unas dimensiones de 6×6mm, y requiere de muy pocos elementos externos para su funcionamiento. Esto reduce de gran manera el tamaño que ocuparían estos módulos en la placa final, ya que ellos necesitarían de un espacio de aproximadamente 30cm2.

Es posible conseguir una placa pre ensamblada de este integrado junto con sus elementos externos, ilustrada en la Figura 38. El módulo permite el acceso a las terminales de control del circuito integrado mediante diez contactos, que incluyen: dos para la comunicación I^2C, una de antena, una de escritura/lectura, entre otras.

Las dimensiones de este módulo incrementan a 11.2×11mm, lo cual no implica cambios muy significativos en el espacio que ocupará en la placa, y facilitará el proceso de su soldadura a la placa final.

Figura 38. Módulo *TEA5767* utilizado para el proceso de demodulación FM en la placa final[41]



b. Precio. Mientras este módulo se puede conseguir desde los US$2.50, el conjunto de los componentes utilizados para el anterior pueden llegar a valer hasta US$20.00. Uno de los objetivos de este trabajo de investigación es optimizar los costos de los componentes del dispositivo final, por lo que esta opción se hace más llamativa para el diseño final del dispositivo.

Adicionalmente, como puede observarse en la Figura 38, este módulo cuenta ya con todos los componentes externos que necesita, por lo que no es necesario invertir en otros componentes para este módulo.

c. Consumo de corriente. Este dispositivo necesita de un máximo de 10mA para su funcionamiento, mientras que el utilizado previamente podía llegar a demandar hasta 50mA (incluyendo los módulos de sintonización y demodulación FM). Esta característica se traduce en una disminución de potencia del dispositivo receptor y, por tanto, un mayor tiempo de vida útil por parte de la batería utilizada.

Con este ahorro de potencia, la vida útil de una batería alcalina de 9V común puede incrementar hasta en 50 horas.[42]

d. Comunicación I^2C. A través de este protocolo, es posible controlar este integrado de manera digital. Una de las características que ofrece es la sintonización de la frecuencia FM deseada mediante el envío de una secuencia de códigos específica (Figura 39). Con esto, se evitó la necesidad de realizar este

[41] Imagen tomada de http://www.siliconray.com/tea5767-fm-radio-receiver-module-bpr5767.html
[42] Información tomada de la hoja de especificaciones de la alcalina MN1604 Duracell (9V)

proceso mediante una señal analógica, el voltaje controlador del *VCO*. Como resultado, la sintonización se hizo más robusta, exacta, y simple: fue controlada directamente desde el módulo de control, y mediante los botones de izquierda y derecha del dispositivo se le permitió al usuario controlar la emisora que se deseara escuchar. La Figura 40 muestra la interfaz que se muestra en pantalla al realizar este proceso.

Figura 39. Señal de comunicación I^2C del *TEA5767* y el módulo de control[43]

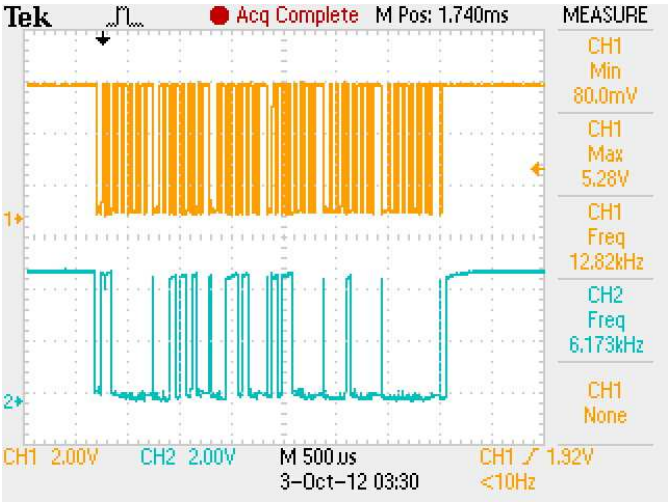
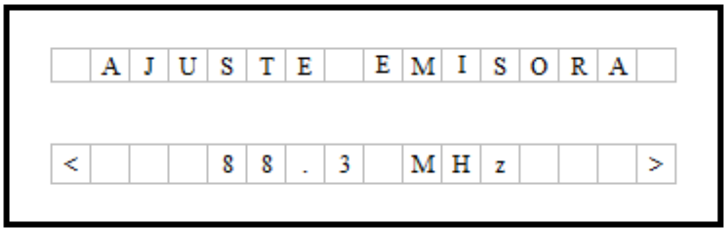


Figura 40. Estructura de la pantalla de sintonización de frecuencia[44]

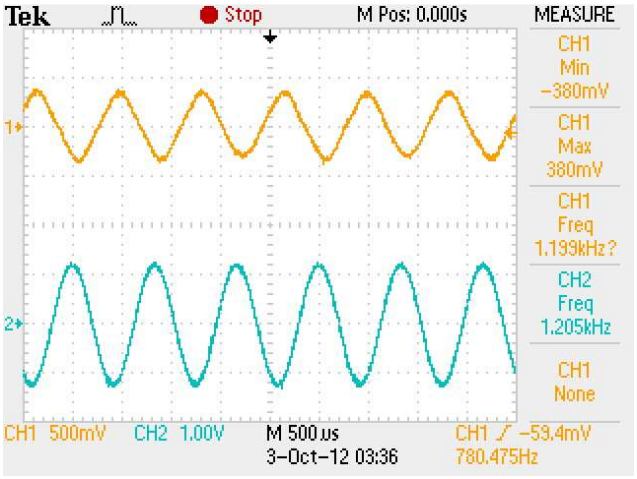


e. Calidad. En el diseño anterior, luego de demodular la frecuencia FM sintonizada, se procedió inmediatamente a amplificar la señal. No se consideró la inclusión de un módulo demultiplexor que separara las señales de audio con *RDS*, razón por la cual, la calidad del audio no era la mejor (prácticamente con una forma parecida a la mostrada en las Figuras 31 y 32). Este dispositivo, por el contrario, realiza esta función de manera interna, y provee tanto la señal de audio en forma estéreo, como la señal previa a la demultiplexión (útil para el análisis *RDS*). Como resultado, se obtuvo una señal de audio

[43] Señales tomadas del osciloscopio en pruebas de laboratorio
[44] Elaboración propia

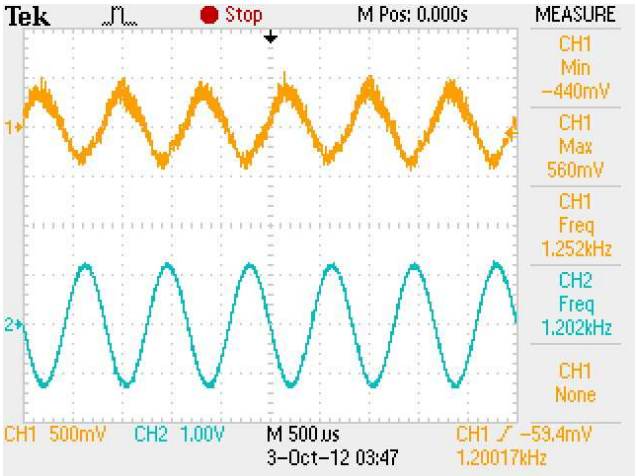
más limpia, como se puede observar en la Figura 41, donde se compara la señal de audio presente en el transmisor y la forma que la misma tendría en la salida del *TEA5767*.

Figura 41. Señales de audio en el transmisor (abajo) y receptor (arriba)[45]



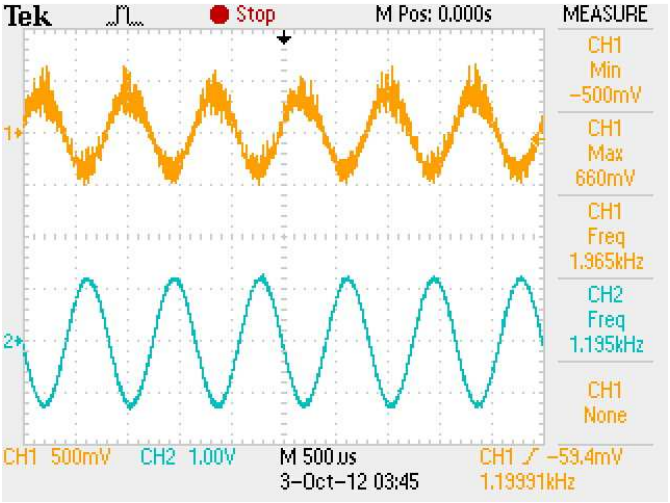
La señal previa a la demultiplexión, utilizada por el demodulador *RDS* para su posterior análisis, también cumplió con los requisitos que exige el *TDA7330B*. Cuando la señal que el receptor procesaba contaba con información *RDS*, este integrado mostró en efecto una señal alta en su indicador de calidad. Las siguientes figuras muestran la forma de esta señal en el receptor, cuando la misma contó con información *RDS* (Figura 43) y cuando no (Figura 42).

Figura 42. Señal en el transmisor (arriba) y receptor (abajo) de un tono a 1.2kHz sin *RDS*, con *TEA5767*[46]



[45] Señales tomadas del osciloscopio en pruebas de laboratorio
[46] Señales tomadas del osciloscopio en pruebas de laboratorio

Figura 43. Señal en el transmisor (arriba) y receptor (abajo) de un tono a 1.2kHz con *RDS*, con TEA5767[47]

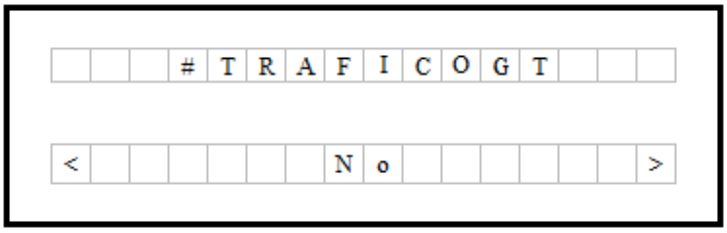


2. Medio de reproducción de audio. Dado que uno de los objetivos de esta investigación fue el de que el dispositivo final fuera completamente portable, se decidió utilizar unos audífonos como medio de reproducción, dejando de lado la reproducción por bocina. Se cambiaron algunos componentes del módulo de amplificación de audio, para que este se ajustara a la intensidad que resultara más cómoda para el usuario. Además, se mantuvo la posibilidad de graduar el volumen de forma manual mediante un potenciómetro.

3. Personalización de la información. En el diseño final de la placa no se incluyó la posibilidad de comunicarse con la computadora, por las razones expuestas en los incisos anteriores. Se agregó una nueva página al dispositivo, específica para la su configuración por parte del usuario. Esta incluye la interfaz mostrada en la Figura 40, utilizada para fijar la frecuencia sintonizada por parte del circuito, y la mostrada a continuación en la Figura 44, especial para la configuración de los menús activos en el receptor.

[47] Señales tomadas del osciloscopio en pruebas de laboratorio

Figura 44. Diseño de la página de configuración de menú[48]



Mediante los botones de navegación, fue posible personalizar el tipo de información que se deseaba recibir. Adicionalmente, estos cambios eran almacenados de manera no volátil en el dispositivo, de modo que cuando este se encendiera, se hiciera presente la última configuración guardada por el usuario.

4. Almacenamiento de mensajes *RadioText*. En el diseño preliminar, las cadenas *RadioText* que detectaba el dispositivo receptor se almacenaban en la memoria *FLASH* interna del PIC16F88. Esto se hizo con la intención de aprovechar al máximo los recursos que ofrecía el microcontrolador, evitando la necesidad de recurrir a elementos externos para su almacenado. Sin embargo, esta decisión presentaba las siguientes desventajas:

- Dado que se reservó un sector de memoria para el almacenamiento de cadenas *RadioText*, el código de programa del microcontrolador se debía limitar al espacio restante. Esto influyó en la cantidad de habilidades que el dispositivo podía ser capaz de realizar.
- Este tipo de memoria se caracteriza por tener tiempo de vida de hasta 100,000 ciclos de escritura, lo cual no es mucho para los estándares que se manejan en memorias (por ejemplo, las *EEPROM* pueden durar hasta más de 1,000,000 ciclos). Por esta razón, las memorias *FLASH* se utilizan comúnmente para almacenar datos que no serán rescritos constantemente, cosa que no concuerda con la aplicación que se le está dando.
- Aunque se tuviera un espacio de memoria reservado para su almacenamiento, este no podía ser tan grande que imposibilitara la programación de las habilidades básicas que el dispositivo debía manejar. En otras palabras, al otorgarle un sector pequeño de memoria a cada una de las partes, se limitaron ambas.

Por estas razones, y considerando todas las mejoras al programa descritas en los últimos incisos, se decidió utilizar una memoria *EEPROM* externa al microcontrolador. Esta estaría destinada específicamente al almacenamiento de las cadenas *RadioText* que el dispositivo fuera recibiendo a lo largo del tiempo.

[48] Elaboración propia

Se utilizó el componente *24LC128*, que contiene un espacio disponible de 16k×8 (128kBits). Los procedimientos de lectura y escritura son controlados mediante la comunicación I^2C con el módulo de control, y así como en el caso del demodulador FM, fue necesario enviar una secuencia específica de códigos para que realizara las funciones que se desearan.

Únicamente se utilizaron 1,024kBytes de memoria, suficientes para almacenar cuatro cadenas *RadioText* de cuatro menús diferentes. Sin embargo, esta cantidad puede ser incrementada según la aplicación que se le desee dar a esta investigación.

VIII. CONCLUSIONES

Al realizar la presente investigación, se llegaron a las siguientes conclusiones:

- Se completó el diseño y construcción de un dispositivo receptor de señales FM portable capaz de demodular el protocolo *RDS*, y mostrar los resultados en una pantalla LCD.
- Tanto el dispositivo transmisor como el receptor desarrollados en esta investigación respetan los estándares que requiere el protocolo *RDS* en los servicios de Nombre de programa, Hora y fecha, y *RadioText* (comprobado mediante el uso del celular *LG Optimus Black* y aplicaciones independientes como *RDS Spy*).
- La tasa de pérdida de información que maneja el dispositivo realizado no representa un problema para la aplicación que se le dio, ya que se obtiene un 99.34% de eficiencia en el procesamiento de datos, si está correctamente sintonizado.
- Cuando el usuario se ubica en una página de refrescamiento de texto constante, como en las que muestran los mensajes *RadioText* mediante el corrimiento de caracteres, se registró una pérdida de un bit de información cada 13.5s.
- Un dispositivo puede prescindir de la comunicación con computadora (por ejemplo *USB*), si esta no le provee información que el mismo aparato pueda conocer. Sin embargo, debe considerarse que el mismo tenga la capacidad de interactuar con el usuario de alguna manera (por ejemplo, a través de una pantalla).
- El dispositivo final, para funcionar correctamente, no requiere un cambio en la tecnología actual por parte de las emisoras radiales.

IX. RECOMENDACIONES

Luego de realizar el presente trabajo de investigación se sugieren las siguientes recomendaciones para conseguir mejores resultados:

- Programarle al módulo de control del receptor la capacidad de manejar otros servicios que ofrece el *RDS* para poder obtener un dispositivo más completo.
- Modificar la forma de manejar el servicio de *RadioText* para que puedan visualizarse hasta 140 caracteres por mensaje (longitud máxima de un *tweet*).
- Cambiar el diseño del dispositivo para que todos sus componentes puedan ser operados a 3V, de manera que se simplifique su alimentación mediante dos baterías de 1.5V.
- Utilizar componentes electrónicos que requieran un consumo menor de corriente, para prolongar el tiempo de vida útil del dispositivo.
- Mediante el servicio de "Aplicación Abierta de datos" (*ODA*), permitir la inclusión de nuevos menús para el dispositivo en formas de actualizaciones. Para esto, se tendría que desarrollar tanto en el receptor como en la fuente emisora de la señal radial.
- Reorganizar la forma en la que se manejan los datos en la memoria *EEPROM* externa, para permitir el almacenamiento de más cadenas *RadioText*.
- Integrar la bocina y la antena en el receptor, que actualmente necesita de estos componentes externos para funcionar.

X. BIBLIOGRAFÍA

Axelson, J. (2009). *USB Complete.* Lakeview Research.

Elektor Electronics. (Febrero de 1991). *Radio Data System (RDS) decoder.* Recuperado el 22 de Junio de 2012, de http://techdoc.kvindesland.no/radio/transvertere/20051108160511837.pdf

European Comitee for Electrotechnical Standrization (CENELEC). (15 de Febrero de 2002). *Specification of the radio data system (RDS) for VHF/FM sound broadcasting in the frequency range from 87.5 to 108.0 MHz.* Recuperado el 13 de Mayo de 2012, de National Standards Authority of Ireland (NSAI): http://www.saiglobal.com/pdftemp/previews/osh/is/en/2002/i.s.en62106-2002.pdf

González, L. (2003). Transmisión y recepción de datos html en bandas FM de radiodifusión. Guatemala: Tesis (Licenciatura en Ingeniería Electrónica)- Universidad del Valle de Guatemala, Facultad de Ciencias y Humanidades.

Kopitz, D., & Marks, B. (1999). *RDS: The Radio Data System.* Recuperado el 3 de Mayo de 2012, de Departamento de Ingeniería Eléctrica, IITB: http://www.ee.iitb.ac.in/student/~bhagwan/communication%20theory/Radio%20Frequency%20and%20Radar%20Systems/RDS..The%20Radio%20Data%20System.pdf

Mishkind, B. (Marzo de 2007). *Radio History.* Obtenido de Old Radio: http://www.oldradio.com/archives/stations/atlanta/wsb.pdf

Ogata, T., Sakata, N., Fujiwara, A., & Okuda, M. (1992). *Fujitsu-Ten.* Recuperado el 25 de Septiembre de 2012, de Radio Data System (RDS): http://www.fujitsu-ten.com/business/technicaljournal/pdf/5-1E.pdf

RDS Forum. (2008). *Newly uptdated RDS Specification.* Genova, Suiza.

Seminario, G. (2005). *RDS Radio data system: Fundamentos y Aplicaciones.* Recuperado el 8 de Mayo de 2012, de GS Broadcast Services: http://www.gsbroadcast.com.ar/RDS%20%20fulll.pdf

Zuloaga, A. (Julio de 1996). *Radio Data System.* Recuperado el 8 de Mayo de 2012, de Universidad del País Vasco: http://www.reocities.com/CapeCanaveral/8482/docu004.pdf

XI. ANEXOS

A. CÓDIGO DE PROGRAMA DEL MICROCONTROLADOR

```
//*********************************************************************
//
//     José Eduardo Barrera
//     Universidad del Valle de Guatemala
//     Módulo de control, decodificador RDS
//
//*********************************************************************



//---------------------------- MEMORIA FLASH --------------------------------
//                              0x0000 - 0x0FFF
//
//Código
//   0x000 - 0xFFF (4,096 palabras)
//



//---------------------------- MEMORIA EEPROM -------------------------------
//                                0x00 - 0xFF
//
//Radiotext principal:
//   0x00 - 0x3F (64 palabras)
//
//Horas de los radiotext:
//   0x40 - 0x4F (16 palabras)
//   0x50 - 0x5F (16 palabras)
//   0x60 - 0x6F (16 palabras)
//   0x70 - 0x7F (16 palabras)
//
//Nombres de menús
//   0x80 - 0x8F (16 palabras)
//   0x90 - 0x9F (16 palabras)
//   0xA0 - 0xAF (16 palabras)
//   0xB0 - 0xBF (16 palabras)
//
//ID de menús
//   0xC0 - 0xC3 (4 palabras)
//   0xC4 - 0xC7 (4 palabras)
//   0xC8 - 0xCB (4 palabras)
```

```
//   0xCC - 0xCF (4 palabras)
//
//Título de menú actual
//   0xD0 - 0xDF (16 palabras)
//
//Configuración de menús
//        0xE0 (1 palabra)
//
//Título de recibido
//   0xE0 - 0xEF (16 palabras)
//
//Frecuencia sintonizada
//        0xEE (Decenas y unidades)
//        0xEF (Decimales)



//------------------------------ PORTA ----------------------------------------
//RA0 - RS (COG)
//RA1 - EN (COG)
//RA2 - D4 (COG)
//RA3 - D5 (COG)
//RA4 - DATA RDS
//RA5 - VPP (ICP)



//------------------------------ PORTB ----------------------------------------
//RB0 - CLOCK RDS
//RB1 - I2C DATA
//RB2 - IZQUIERDA
//RB3 - ABAJO
//RB4 - I2C CLOCK
//RB5 - DERECHA
//RB6 - D6 & CLK (COG & ICP)
//RB7 - D7 & I/O (COG & ICP)



//Inicialización de la COG
sbit LCD_RS at RA0_bit;
sbit LCD_EN at RA1_bit;
sbit LCD_D4 at RA2_bit;
sbit LCD_D5 at RA3_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISA0_bit;
sbit LCD_EN_Direction at TRISA1_bit;
sbit LCD_D4_Direction at TRISA2_bit;
sbit LCD_D5_Direction at TRISA3_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
```

```
//Inicialización del I2C versión software
sbit Soft_I2C_Scl at RB4_bit;
sbit Soft_I2C_Sda at RB1_bit;
sbit Soft_I2C_Scl_Direction at TRISB4_bit;
sbit Soft_I2C_Sda_Direction at TRISB1_bit;



//Caracteres especiales para la pantalla LCD
//Caracteres: á, é, í, ó, ú, ü, ñ
const unsigned char ESP_CHAR[56] = {2,4,14,1,15,17,15,0, 2,4,14,17,31,16,14,0,
     2,4,0,12,4,4,14,0, 2,4,0,14,17,17,14,0, 2,4,0,17,17,19,13,0,
     0,10,0,17,17,19,13,0, 13,18,0,23,25,17,17,0};


//Constantes para hacer la revisión del síndrome
const unsigned short CRC_CONST[32] = {0b00000010, 0b11011100, 0b00000001,
     0b01101110, 0b00000000, 0b10110111, 0b00000010, 0b10000111, 0b00000011,
     0b10011111, 0b00000011, 0b00010011, 0b00000011, 0b01010101, 0b00000011,
     0b01110110, 0b00000001, 0b10111011, 0b00000010, 0b00000001, 0b00000011,
     0b11011100, 0b00000001, 0b11101110, 0b00000000, 0b11110111, 0b00000010,
     0b10100111, 0b00000011, 0b10001111, 0b00000011, 0b00011011};


//Variables para almacenar la cadena de bits
unsigned short BLOCK_1L = 0;
unsigned short BLOCK_1H = 0;
unsigned short BLOCK_2L = 0;
unsigned short BLOCK_2H = 0;
unsigned short BLOCK_3L = 0;
unsigned short BLOCK_3H = 0;
unsigned short BLOCK_4L = 0;
unsigned short BLOCK_4H = 0;

//Variables para la revisión de los checkwords y offsets
unsigned short CHECK_L = 0;
unsigned short CHECK_H = 0;
unsigned short INFO_A = 0;
unsigned short INFO_B = 0;
unsigned short INFO_C = 0;
unsigned short INFO_D = 0;

//Buffer para la recepción de nuevos datos
unsigned long BUFF = 0;
unsigned long BUFF_I = 0;

//Contadores y auxiliares
unsigned short ACT_BLK = 1;                     //1 - 4
unsigned short ACT_BIT = 0;                     //0 - 26
unsigned short ACT_PAG = 0;                     //0 - 4, 5 para la especial
unsigned short ACT_LIN = 0;                     //0 - 4
unsigned short ACT_CHAR = 0;                    //0 - 64

//Temporizadores para caracteres y botonazos
```

```
unsigned short TMR_CHR = 0;
unsigned short TMR_BTN = 0;

unsigned short LIN_M[5] absolute 0xEA;            //0 - 4

//Apuntadores para los radio text más recientes de cada categoría
//Orden: Promociones, Noticias, Deportes, Tráfico (c/u 0-3 para 4 noticias)
unsigned short ADD_TXT[4] absolute 0x42;

//Variable donde se almacenará el RadioText que se está recibiendo
char RADIO_TEXT[64];

//Temporal útil para la línea de mensaje mostrado actualmente
char TEMP_MEN[17] absolute 0x20;

//Mensaje activo a mostrar en COG
char TIT_ACT[17] absolute 0x31;
char MEN_ACT[66] absolute 0xA0;

//Mensaje superior del menú principal
char MEN_PRINC[17] absolute 0x46;

//Banderas
unsigned short MY_FLAGS = 0b00110101;
//B0: Activo cuando los botones están habilitados
//B1: Se activa cuando es el nuevo grupo es versión B
//B2: Se debe actualizar el texto a mostrar
//B3: Se completaron 250ms
//B4: Se debe cambiar la información de la línea superior
//B5: Se debe cambiar la información de la línea inferior
//B6: Activo cuando el último grupo analizado de RadioText fue B

unsigned short CONF_FLAG = 0b00000000;
//B0: Se inicia el identificador de cambio de menú
//B1: Activo cuando se está utilizando el menú especial
//B2: Activo: cambio en menú especial hacia la izquierda. Viceversa

unsigned short CONF_CNT = 0;

//Variables para almacena los registros útiles para la frecuencia de
//   sintonización
unsigned int FREQ = 0;
unsigned long FREQ_C = 0;
unsigned short FREQ_CL = 0;
unsigned short FREQ_CH = 0;

unsigned short SHOW_MENU;
unsigned short END_RT = 0;



//*************************** INTERRUPCIONES ********************************
//Procedimiento que controla las medidas a tomar según el tipo de interrupción
//   que se reciba
void interrupt() {
```

```
//Interrupción de tiempo
if (TMR1IF_bit == 1){

     //Significa que pasaron 250ms
     //Se debe activa bandera de 250ms (MY_FLAGS.B3)
     TMR1IF_bit = 0;
     MY_FLAGS.B3 = 1;

     //Reinicio de temporizador
     TMR1H = 0x0B;
     TMR1L = 0xDC;
     TMR_CHR++;
     TMR_BTN++;

     //Se vuelven a habilitar los botones sólo si han pasado al menos
     //   500ms
     if (TMR_BTN.B1 == 1){
          TMR_BTN = 0;
          MY_FLAGS.B0 = 1;
     }

           //Se debe esperar un segundo para realizar un cambio de
           //  tipo de menú
           if ((CONF_FLAG.B0 == 1) && (PORTB.B3 == 0)) CONF_CNT++;
           else{

                   //Significa que no se desea cambiar el tipo de menú
                   CONF_CNT = 0;
                   CONF_FLAG.B0 = 0;
           }
}

//Interrupción para almacenar el nuevo bit
if (INTCON.INTF == 1){
     //Se almacenará sólo si queda espacio en el buffer
     if (BUFF_I < 0x80000000){

          BUFF_I <<= 1;
          if (BUFF_I == 0) BUFF_I = 1;
          if (PORTA.B4 == 1) BUFF |= BUFF_I;

     }
     else{

          //Si no queda espacio, se vacía y se resetean los contadores
          //   (para no detectar información falsa)
          BUFF = 0;
          BUFF_I = 0;
          ACT_BIT = 0;
          ACT_BLK = 1;

     }
     INTCON.INTF = 0;
}
```

```
}



//************************ INICIALIZACIÓN DE LCD ****************************
//Procedimiento que prepara la pantalla LCD con los caracteres especiales
//   definidos (á, é, í, ó, ú, ü, ñ)
void Lcd_Load(){
     unsigned short CONT, CONT2, AUX;

     for (CONT2 = 0; CONT2 < 7; CONT2++){

          AUX = CONT2 << 3;
          LCD_Cmd(64 + AUX);

          for (CONT = 0; CONT < 8; CONT++){
               //Comando para almacenar el caracter especial en su dirección
               //   correspondiente
               LCD_Chr_Cp(ESP_CHAR[AUX + CONT]);
          }
     }
}



//********************* ACONDICIONADOR DE CARACTERES ************************
//Procedimiento que prepara la pantalla LCD con los caracteres especiales
//   definidos (á, é, í, ó, ú, ü, ñ)
char code2Char(char CHAR_FROM){
     char CHAR_TO;
     CHAR_TO = CHAR_FROM;

     //Se cambiará el caracter sólo sí es especial
     if (CHAR_TO > 0x7F) switch (CHAR_TO){
          case 0x80:
          case 0xC1:
               CHAR_TO = 0xE1;                    //á
               break;
          case 0x82:
          case 0xC9:
               CHAR_TO = 0xE9;                    //é
               break;
          case 0x84:
          case 0xCD:
               CHAR_TO = 0xED;                    //í
               break;
          case 0x86:
          case 0xD3:
               CHAR_TO = 0xF3;                    //ó
               break;
          case 0x88:
          case 0xDA:
               CHAR_TO = 0xFA;                    //ú
```

```
                break;
           case 0x99:
           case 0xDC:
                CHAR_TO = 0xFC;                    //ü
                break;
           case 0x9A:
           case 0xD1:
                CHAR_TO = 0xF1;                    //ñ
                break;
      }
      return CHAR_TO;
}



//********************* LECTOR DE LA MEMORIA EEPROM *************************
//Procedimiento que lee los N caracteres de la memoria EERPOM, partiendo de
//   la dirección formada por ADD_UP:ADD_DN. El resultado es almacenado en
//   el espacio de memoria apuntado por *MEN
void readNBytes(unsigned short ADD_UP, unsigned short ADD_DN,
      unsigned short *MEN, unsigned short N){

      unsigned short CONT;

      //Como se utiliza la librería de I2C versión software, es aconsejable
      //   inhabilitar las interrupciones mientras se están enviando comandos
      GIE_bit = 0;

      //Envío de procedimiento de escritura vacía, para ubicar al contador
      //   interno de la memoria en el punto que se desea obtener
      Soft_I2C_Start();
      Soft_I2C_Write(0b10100000);
      Soft_I2C_Write(ADD_UP);
      Soft_I2C_Write(ADD_DN);
      Soft_I2C_Start();

      //Se envía el código de lectura
      Soft_I2C_Write(0b10100001);

      //Para los N-1 bytes que se leerán, debe dar una señal de
      //   NOT_AKN (según las especificaciones de la memoria)
      for (CONT = 0; CONT < (N - 1); CONT++){
           MEN[CONT] = Soft_I2C_Read(1);
      }

      //El último byte a leer debe retornar una señal de AKN (según las
      //   especificaciones de la memoria)
      MEN[CONT] = Soft_I2C_Read(0);
      Soft_I2C_Stop();
      GIE_bit = 1;
}
```

```
//******************** ESCRITOR DE LA MEMORIA EEPROM ************************
//Procedimiento que escribe los 64 caracteres de un RadioText en la memoria
//   externa EEPROM
void writeRT_EEPROM(unsigned short ADD_UP, unsigned short ADD_DN){

     unsigned short CONT;

     //Como se utiliza la librería de I2C versión software, es aconsejable
     //   inhabilitar las interrupciones mientras se están enviando comandos
     GIE_bit = 0;

     //Se envía el código de escritura
     Soft_I2C_Start();
     Soft_I2C_Write(0b10100000);
     Soft_I2C_Write(ADD_UP);
     Soft_I2C_Write(ADD_DN);

     //Se envían los 64 caracteres del RadioText
     IRP_bit = 1;
     for (CONT = 0; CONT<64; CONT++){
          Soft_I2C_Write(RADIO_TEXT[CONT]);
     }
     IRP_bit = 0;
     Soft_I2C_Stop();
     GIE_bit = 1;
}



//******************** POSICIONAR RADIOTEXT EN MEMORIA ***********************
//Procedimiento que posiciona el mensaje de la variable RADIOTEXT en su
//   ubicación correspondiente, dependiendo de la información que contenga.
//   Revisa primero si es uno nuevo
//   Direcciones de los bloques: 0xC00, 0xD00, 0xE00, 0xF00
void placeRadioText(){

     unsigned short AUX, TEMP;
     unsigned short CONT, CONT2, CONT3;
     unsigned short ADD_H, ADD_L;

     CONT3 = SHOW_MENU;
     for (CONT = 0; CONT < 4; CONT++){

          //Se prepara la dirección base
          AUX = 0xC0 + (CONT << 2);
          //Se compara la identificación del radiotext
          for (CONT2 = 0; CONT2 < 4; CONT2++){

               IRP_bit = 1;
               TEMP = RADIO_TEXT[CONT2];
               IRP_bit = 0;

               //Si el mensaje no contuviera el identificador, se revisa la
```

```
     //   siguiente opción
     if (TEMP != EEPROM_Read(AUX + CONT2)) CONT2 = 99;
}

//CONT2 será 4 sólo si no se encontró diferencia entre la
//   identificación y los primeros 4 caracteres de la cadena
if (CONT2 == 4){

     //Se revisa si se están recibiendo cadenas del menú encontrado
     if (CONT3.B0 == 1){

          //Se ajusta la dirección al bloque a comparar
          ADD_H = CONT;
          ADD_L = ADD_TXT[CONT] << 6;

          //Se debe revisar si es un texto nuevo o ya se había
          //   recibido
          for (CONT2 = 0; CONT2 < 64; CONT2 = CONT2 + 4){

               IRP_bit = 1;
               AUX = RADIO_TEXT[CONT2];
               IRP_bit = 0;

               readNBytes(ADD_H, ADD_L + CONT2, TEMP_MEN, 1);

               //Si al menos un caracter es diferente, se debe
               //   almacenar esta cadena
               if (TEMP_MEN[0] != AUX) CONT2 = 96;
          }

          //En caso que las cadenas fueran diferentes, se guarda
          if (CONT2 == 100){

               if (ACT_PAG < 5){
                    //Se resetean los contadores
                    ACT_PAG = CONT + 1;
                    ACT_LIN = 1;

                    //Se debe visualizar desde el primer caracter
                    ACT_CHAR = 0;
                    TMR_CHR = 0;

                    //Se activa la bandera para cambio de texto
                    MY_FLAGS |= 0b00110100;
                    MY_FLAGS.B3 = 0;

               }

               //Se debe caer sobre la cadena más antigua, por
               //   lo que se ajusta el apuntador
               ADD_TXT[CONT]--;
               if (ADD_TXT[CONT].B7 == 1) ADD_TXT[CONT] = 3;
               ADD_H = CONT;
               ADD_L = ADD_TXT[CONT] << 6;
```

```
                    //Se escribe la cadena de RadioText en la dirección
                    //   especificada
                    writeRT_EEPROM(ADD_H, ADD_L);

                    //Se ajusta también la hora del mensaje
                    AUX = 0x40 + (CONT << 4) + (ADD_TXT[CONT] << 2);
                    EEPROM_Write(AUX + 0, MEN_PRINC[11]);
                    EEPROM_Write(AUX + 1, MEN_PRINC[12]);
                    EEPROM_Write(AUX + 2, MEN_PRINC[14]);
                    EEPROM_Write(AUX + 3, MEN_PRINC[15]);

                    //Si aún no se ha llenado la capacidad máxima de
                    //   cadenas, se aumenta en 1 la cantidad actual
                    if (LIN_M[CONT] < 4) LIN_M[CONT]++;

                    Delay_ms(10);
               }
          }

          //Se debe salir de la rutina, ya se hizo la comparación
          CONT = 10;
     }

     //Para ajustar la variable que revisa los menús activos
     CONT3 >>= 1;
}
//Si CONT queda como 4 es porque se salió el ciclo sin encontrar un
//   ID correspondiente. Por lo tanto, es un mensaje del menú principal

if (CONT == 4){

     //Se debe revisar si es un texto nuevo o ya se había recibido
     for (CONT2 = 0; CONT2 < 64; CONT2++){
          IRP_bit = 1;
          AUX = RADIO_TEXT[CONT2];
          IRP_bit = 0;
          if (EEPROM_Read(CONT2) != AUX) CONT2 = 99;
     }

     //Se guardará la cadena en caso que sea diferente
     if (CONT2 == 100){

          if (ACT_PAG < 5){
               //Se resetean los contadores
               ACT_PAG = 0;
               ACT_LIN = 0;

               //Se debe visualizar desde el primer caracter
               ACT_CHAR = 0;
               TMR_CHR = 0;

               //Se activa la bandera para cambio de texto
               MY_FLAGS |= 0b00110100;
               MY_FLAGS.B3 = 0;
```

```
                }

                for (CONT2 = 0; CONT2 < 64; CONT2++){

                     //Se almacena la nueva información
                     IRP_bit = 1;
                     AUX = RADIO_TEXT[CONT2];
                     IRP_bit = 0;
                     EEPROM_Write(CONT2, AUX);
                }
           }
     }
}



//********************** RESETEO DE RADIOTEXT ********************************
//Procedimiento que limpia la información contenida en la variable de
//   RadioText, para prepararlo para una nueva recepción
void resetRadioText(){
     unsigned short CONT;

     IRP_bit = 1;
     for (CONT = 0; CONT < 64; CONT++){

          //La forma de limpiarlo es mediante caracteres 0xFF en todas sus
          //   localidades
          RADIO_TEXT[CONT] = 0xFF;

     }
     IRP_bit = 0;

}



//************************ REVISIÓN DE GRUPO ********************************
//Procedimiento que revisa el grupo completo luego de haberse completado
void checkGroup(){
     char CONT, AUX, AUX2;

     switch (BLOCK_2H & 0xF8){
          case 0x00:
          case 0x08:

               //Significa que tenemos un grupo de tipo 0A o 0B
               //Se actualiza el nombre de la estación

               AUX = BLOCK_2L & 0x03;
               AUX <<= 1;
               AUX2 = code2Char(BLOCK_4H);

               if (MEN_PRINC[AUX] != AUX2){
                    MEN_PRINC[AUX] = AUX2;
```

```
            //Se activa la bandera para cambio de texto superior
            if (ACT_PAG == 0) MY_FLAGS |= 0b00010100;
        }

        AUX2 = code2Char(BLOCK_4L);
        if (MEN_PRINC[AUX + 1] != AUX2){
            MEN_PRINC[AUX + 1] = AUX2;

            //Se activa la bandera para cambio de texto superior
            if (ACT_PAG == 0) MY_FLAGS |= 0b00010100;
        }
        break;
    case 0x40:

        //Significa que tenemos un grupo 4A
        //Se obtuvo información de la hora

        AUX = BLOCK_4H & 0xF0;
        AUX >>= 4;
        if (BLOCK_3L.B0 == 1) AUX.B4 = 1;

        //Se transforma la información en decenas y unidades
        //   Horas
        MEN_PRINC[11] = (AUX/10)+48;
        MEN_PRINC[12] = (AUX%10)+48;

        //   Minutos
        AUX = BLOCK_4H & 0x0F;
        AUX <<= 2;
        if (BLOCK_4L.B6 == 1) AUX.B0 = 1;
        if (BLOCK_4L.B7 == 1) AUX.B1 = 1;
        MEN_PRINC[14] = (AUX/10)+48;
        MEN_PRINC[15] = (AUX%10)+48;

        //   Dos puntos
        MEN_PRINC[13] = 0x3A;

        //Se activa la bandera para cambio de texto superior
        if (ACT_PAG == 0) MY_FLAGS |= 0b00010100;
        break;

    case 0x20:

        //Significa que tenemos un grupo 2A
        //Se actualiza el RadioText

        if (MY_FLAGS.B6 != BLOCK_2L.B4){
            MY_FLAGS.B6 = BLOCK_2L.B4;
            //Se debe resetear el RadioText
            resetRadioText();
        }

        CONT = BLOCK_2L & 0x0F;
        CONT <<= 2;
```

```
                    //Se verifica si este grupo de 4 bits no fue escrito
                    //   anteriormente
                    IRP_bit = 1;
                    AUX = RADIO_TEXT[CONT];
                    IRP_bit = 0;

                    //Únicamente se revisa el primer caracter, ya que si está vacío
                    //   siginfica que el resto de caracteres de ese grupo también
                    //   lo estarán
                    if (AUX == 0xFF){

                         //Se almacenan cada uno de los cuatro caracteres
                         AUX = code2Char(BLOCK_3H);
                         IRP_bit = 1;
                         RADIO_TEXT[CONT + 0] = AUX;
                         IRP_bit = 0;

                         AUX = code2Char(BLOCK_3L);
                         IRP_bit = 1;
                         RADIO_TEXT[CONT + 1] = AUX;
                         IRP_bit = 0;

                         AUX = code2Char(BLOCK_4H);
                         IRP_bit = 1;
                         RADIO_TEXT[CONT + 2] = AUX;
                         IRP_bit = 0;

                         AUX = code2Char(BLOCK_4L);
                         IRP_bit = 1;
                         RADIO_TEXT[CONT + 3] = AUX;
                         IRP_bit = 0;

                         //Se revisa si el RadioText ya fue completamente
                         //   recibido
                         IRP_bit = 1;
                         for (CONT = 0; CONT < 64; CONT++){
                              if (RADIO_TEXT[CONT] == 0xFF) CONT = 99;
                         }
                         IRP_bit = 0;

                         //El ciclo anterior se realizará por completo únicamente
                         //   si no se encuentran espacios en el RadioText
                         if (CONT == 64){

                              placeRadioText();
                              //Se resetea el RadioText para esperar uno nuevo
                              resetRadioText();

                         }
                    }
                    break;
               }
          }
```

```
//************************ AJUSTADOR DE INFORMACIÓN ***********************
//Procedimiento que ajusta los bits recibidos para colocarlos donde
//   corresponda. Los bits útiles quedarán en los registros INFO_B e INFO_C,
//   siendo INFO_B.B0  el primero en entrar (MSB)
void adjustData(){

     //Las llamadas "if's" son sólo para ubicar el banco en el que se
     //   encuentra la variable, para poder realizar el corrimiento
     //   en assembler.
     if (INFO_A);
     asm RRF _INFO_A, F
     if (INFO_B);
     asm RRF _INFO_B, F
     if (INFO_C);
     asm RRF _INFO_C, F
     if (INFO_A);
     asm RRF _INFO_A, F
     if (INFO_B);
     asm RRF _INFO_B, F
     if (INFO_C);
     asm RRF _INFO_C, F

}



//************************ REVISIÓN DE BLOQUE ******************************
//Se revisa si el checkword concuerda con el del bloque en el que se encuentra
void checkBlock(){

     switch (ACT_BLK){

          case 1:
               //En caso que se está esperando el primer bloque
               //Se revisa si el síndrome corresponde al primer bloque
               if ((CHECK_H == 0x03) && (CHECK_L == 0xD8)){

                    adjustData();
                    //Se almacena la información
                    BLOCK_1H = INFO_B;
                    BLOCK_1L = INFO_C;
                    ACT_BLK++;
               }

               //Si no es un bloque 1 válido, se desecha únicamente el bit más
               //   viejo, para buscar la sincronía
               else ACT_BIT = 25;
               break;

          case 2:
               //En caso que se está esperando el segundo bloque
```

```
          //Se revisa si el síndrome corresponde al segundo bloque
          if ((CHECK_H == 0x03) && (CHECK_L == 0xD4)){

               adjustData();
               //Se almacena la información
               BLOCK_2H = INFO_B;
               BLOCK_2L = INFO_C;
               ACT_BLK++;
          }

          //Si no es un bloque 2 válido, se desecha todo lo que se haya
          //   almacenado para volver a buscar la sincronía
          else ACT_BLK = 1;
          break;

     case 3:
          //En caso que se está esperando el tercer bloque
          //Se revisa si el síndrome corresponde al tercer bloque

          //El algoritmo está capacitado para recibir grupos 3A y 3B.
          if (((CHECK_H == 0x02) && (CHECK_L == 0x5C)) ||
               ((CHECK_H == 0x03) && (CHECK_L == 0xCC))){

                    adjustData();
                    //Se almacena la información
                    BLOCK_3H = INFO_B;
                    BLOCK_3L = INFO_C;
                    ACT_BLK++;
          }

          //Si no es un bloque 3B o 3A válido, se desecha todo lo que se
          //   haya almacenado para volver a buscar la sincronía
          else ACT_BLK = 1;
          break;

     case 4:
          //En caso que se está esperando el último bloque
          //Se revisa si el síndrome corresponde al cuarto bloque
          if ((CHECK_H == 0x02) && (CHECK_L == 0x58)){

               adjustData();
               //Se almacena la información
               BLOCK_4H = INFO_B;
               BLOCK_4L = INFO_C;

               //Si se recibió este bloque correcto, significa que ya se
               //   tiene el grupo completo. En este caso, se llama el
               //   siguiente procedimiento para analizar su información
               checkGroup();
          }

          //Si no es un bloque 3B válido, se desecha todo lo que se haya
          //   almacenado para volver a buscar la sincronía
          ACT_BLK = 1;
          break;
```

```
     }
}



//************************ AGREGAR BIT DEL BUFFER ****************************
//Procedimiento para agregar un bit a la cadena de datos
void pushBit(){

     unsigned short CONT, AUX, INFO_C_T, INFO_D_T;

     //El siguiente ciclo se realiza hasta vaciar el buffer, para aprovechar
     //   este procedimiento lo más posible y evitar el desbordamiento del
     //   mismo.
     while (BUFF_I > 0)
     {
          //Se decrementa en 1 el buffer, dato agregado
          BUFF_I >>= 1;
          //Fue agregado un bit
          ACT_BIT++;

          //La llamada al if únicamente ubica el bloque del buffer
          if (BUFF);
          asm BCF STATUS, C
          asm RRF _BUFF+3, F
          asm RRF _BUFF+2, F
          asm RRF _BUFF+1, F
          asm RRF _BUFF+0, F

          //El bit más viejo del buffer es colocado al final de la cola
          if (INFO_D);
          asm RLF _INFO_D, F
          if (INFO_C);
          asm RLF _INFO_C, F
          if (INFO_B);
          asm RLF _INFO_B, F
          if (INFO_A);
          asm RLF _INFO_A, F

          //Se revisa si se terminó de llenar la cola
          if (ACT_BIT == 26){

               //Se llenó un bloque
               ACT_BIT = 0;
               CHECK_H = INFO_A & 0x03;
               CHECK_L = INFO_B;
               INFO_C_T = INFO_C;
               INFO_D_T = INFO_D;

               for (CONT = 0; CONT < 16; CONT++){

                    // Se inicia por el primer bit que entró
                    if (INFO_D_T);
                    asm RLF pushBit_INFO_D_T_L0, F
```

```
                    if (INFO_C_T);
                    asm RLF pushBit_INFO_C_T_L0, F
                    if (AUX);
                    asm RLF pushBit_AUX_L0, F

                    //Se hace la multiplicación matricial por la matriz de
                    //   control
                    if (AUX.B0 == 1){
                         AUX = CONT << 1;
                         CHECK_H = CHECK_H ^ CRC_CONST[AUX];
                         CHECK_L = CHECK_L ^ CRC_CONST[AUX + 1];
                    }
               }

               //Se tiene el bloque listo para su análisis
               checkBlock();
          }
     }
}



//********************** DESPLEGAR LÍNEA EN LCD ******************************
//En base a los parámetros LINE (1/2) y MEN, se escribe el mensaje solicitado
//   en la línea correspondiente de la LCD
void printLine(unsigned short LINE, unsigned short *MEN){
     unsigned short CONT;

     //Se muestra la línea de forma preliminar, se supone que no hay
     //   caracteres especiales
     Lcd_Out(LINE, 1, MEN);

     //Se revisa cada caracter para comprobar que no sea especial
     for (CONT = 0; CONT < 16; CONT++){

          //Si ubica un caracter especial, se coloca su identificador
          //   correspondiente en el espacio que lo necesite
          switch (MEN[CONT]){
               case 0xE1:
                    Lcd_Chr(LINE, CONT + 1, 0);        //á
                    break;
               case 0xE9:
                    Lcd_Chr(LINE, CONT + 1, 1);        //é
                     break;
               case 0xED:
                     Lcd_Chr(LINE, CONT + 1, 2);       //í
                    break;
               case 0xF3:
                    Lcd_Chr(LINE, CONT + 1, 3);        //ó
                    break;
               case 0xFA:
                    Lcd_Chr(LINE, CONT + 1, 4);        //ú
                    break;
               case 0xFC:
```

```
                    Lcd_Chr(LINE, CONT + 1, 5);        //ü
                    break;
               case 0xF1:
                    Lcd_Chr(LINE, CONT + 1, 6);        //ñ
                    break;
          }
     }
}



//********************** MOSTRAR MENSAJE EN LCD ****************************
//Procedimiento que muestra el mensaje actual de COG
void showMessage(){

     unsigned short CONT;

     //Se revisa inicialmente si se debe mostrar la primera línea
     if (MY_FLAGS.B4 == 1){
          printLine(1, TIT_ACT);
          MY_FLAGS.B4 = 0;
     }

     //Se revisa si se debe mostrar la segunda línea
     if (MY_FLAGS.B5 == 1){
          for (CONT = 0; CONT < 16; CONT++){
               TEMP_MEN[CONT] = MEN_ACT[ACT_CHAR + CONT];
          }
          printLine(2, TEMP_MEN);
          MY_FLAGS.B5 = 0;
     }
}



//************************** CAMBIAR MENSAJE ******************************
//Procedimiento que prepara el nuevo texto a mostrar
void changeMessage(){

     unsigned short CONT, AUX, AUX2;
     MY_FLAGS.B2 = 0;

     //Primero revisar si se debe mostrar el texto del menú principal
     if (ACT_PAG == 0){

          //Se coloca el mensaje principal como el título actual
          for (CONT = 0; CONT < 64; CONT++){
               AUX = CONT>>2;
               TIT_ACT[AUX] = MEN_PRINC[AUX];
               MEN_ACT[CONT] = EEPROM_Read(CONT);
          }

          //Se busca el final de la cadena
```

```
     END_RT = 64;
     while (MEN_ACT[END_RT - 1] == 0x20){
          END_RT--;
     }
}
else{

     if (ACT_LIN == 0){

          //En caso que se esté mostrando el título del menú
          for (CONT = 0; CONT < 16; CONT++){
               TIT_ACT[CONT] = EEPROM_Read(0xD0 + CONT);
          }

          AUX = 0;
          AUX2 = SHOW_MENU;

          //Se busca el número de menú que se está visualizando
          for (CONT = 0; CONT < ACT_PAG; CONT++) {
               if (AUX2.B0 == 1) AUX++;
               AUX2 >>= 1;
          }
          TIT_ACT[10] = AUX + 48;

          //Se prepara la dirección de EEPROM donde se encuentra el
          //   nombre del menú
          AUX = 0x70 + (ACT_PAG << 4);
          for (CONT = 0; CONT < 16; CONT++){
               MEN_ACT[CONT] = EEPROM_Read(AUX + CONT);
          }

     }
     else{

          //En este caso se está mostrando un mensaje RadioText
          //El siguiente ciclo me lleva al texto que se desee visualizar,
          //   según el momento en el que fuera recibido
          AUX = ADD_TXT[ACT_PAG - 1];
          for (CONT = 1; CONT < ACT_LIN; CONT++){
               AUX++;
               if (AUX == 4) AUX = 0;
          }

          //Se obtiene el mensaje RadioText
          readNBytes(ACT_PAG - 1, AUX << 6, MEN_ACT, 64);

          //Se busca el final de cadena
          END_RT = 64;
          while (MEN_ACT[END_RT - 1] == 0x20){
               END_RT--;
          }

          //Se obtiene el título del mensaje
          for (CONT = 0; CONT < 16; CONT++){
               TIT_ACT[CONT] = EEPROM_Read(0xE0 + CONT);
```

```
                }
                TIT_ACT[0] = ACT_LIN + 48;

                //Se prepara la dirección para obtener la hora del mensaje
                AUX = 0x40 + ((ACT_PAG - 1) << 4) + (AUX << 2);
                TIT_ACT[11] = EEPROM_Read(AUX + 0);
                TIT_ACT[12] = EEPROM_Read(AUX + 1);
                TIT_ACT[14] = EEPROM_Read(AUX + 2);
                TIT_ACT[15] = EEPROM_Read(AUX + 3);
                TIT_ACT[13] = MEN_PRINC[13];
           }
     }
}



//********************** CAMBIAR MENSAJE ESPECIAL ****************************
//Procedimiento que cambia el mensaje a mostrar en caso que el usuario se
//   encuentre en la página especial
void changeMessageSpecial(){
     unsigned short CONT, AUX;

     MY_FLAGS.B2 = 0;

     //Se revisa si se está en la página de sintonización
     if (ACT_LIN == 0){

          //Se muestra el mensaje de instrucción
          for (CONT = 0; CONT < 16; CONT++){
               TIT_ACT[CONT] = EEPROM_Read(0xF0 + CONT);
               MEN_ACT[CONT] = 0x20;
          }
          MEN_ACT[8] = 0x2E;

          //De acuerdo a la frecuencia activa, se ajusta el mensaje a
          //   desplegar
          MEN_ACT[9] = (FREQ%10) + 48;
          AUX = FREQ/10;
          MEN_ACT[7] = (AUX%10) + 48;
          AUX /= 10;
          MEN_ACT[6] = (AUX%10) + 48;
          AUX /= 10;
          if (AUX != 0) MEN_ACT[5] = AUX + 48;

     }
     else{

          //Se muestra el nombre del menú que se está configurando
          AUX = 0x70 + (ACT_LIN << 4);
          for (CONT = 0; CONT < 16; CONT++){
               TIT_ACT[CONT] = EEPROM_Read(AUX + CONT);
               MEN_ACT[CONT] = 0x20;
          }
```

```
          MEN_ACT[0] = 0x3C;
          MEN_ACT[15] = 0x3E;
          //Si en caso el menú está activo, se mostrará el mensaje "Sí".
          //   De lo contrario, se mostrará "No"
          AUX = SHOW_MENU >> (ACT_LIN - 1);
          if (AUX.B0 == 1){
               MEN_ACT[7] = 0x53;
               MEN_ACT[8] = 0xED;
          }
          else{
               MEN_ACT[7] = 0x4E;
               MEN_ACT[8] = 0x6F;
          }
     }
}



//************************* AJUSTE DE ÍNDICE ******************************
//Procedimiento que ajusta los índices según el tiempo que ha pasado
void adjustIndex(){
     MY_FLAGS.B3 = 0;

     //Se cambiará el índice del mensaje sólo si se encuentra en el menú
     //   principal o se está mostrando un mensaje RadioText
     if ((ACT_PAG == 0) || (ACT_LIN != 0)){

          //Se revisa qué número de caracter se está mostrando en ese momento
          switch (ACT_CHAR)
          {
               case 0:

                    //En caso que se esté mostrando el primer caracter, no se
                    //   correrá la información hasta que haya pasado al menos
                    //   1 segundo.
                    if ((TMR_CHR == 5) && (END_RT > 16)){

                         //Significa que ha pasado 1s
                         TMR_CHR = 0;
                         ACT_CHAR++;

                         //Se tiene un nuevo mensaje listo a mostrar
                         MY_FLAGS.B5 = 1;
                    }
                    break;
               default:

                    //Si no se está mostrando el primer caracter, cuando haya
                    //   transcurrido 250ms, se corre el mensaje
                    ACT_CHAR++;
                    TMR_CHR = 0;

                    //Se tiene un nuevo mensaje listo a mostrar
                    MY_FLAGS.B5 = 1;
```

```
                    //Si se llego al final de la cadena, se deberá mostrar
                    //   nuevamente desde su inicio
                    if (ACT_CHAR == (END_RT + 1)) ACT_CHAR = 0;
                    break;
               }
          }
}



//************************* SINTONIZADOR ***********************************
//Procedimiento que manda el comando de sintonización al TEA5767
void sintonizar(){

     //Ajuste de la frecuencia a sintonizar
     FREQ_C = ((FREQ*100000+225000)/32768) << 2;
     FREQ_CH = FREQ_C >> 8;
     FREQ_CL = FREQ_C & 0xFF;

     //Se deshabilitan las interrupciones temporalmente
     GIE_bit = 0;
     Soft_I2C_Start();

     //Dirección del chip TEA5767
     Soft_I2C_Write(0xC0);

     //Frecuencia inicial
     Soft_I2C_Write(FREQ_CH);
     Soft_I2C_Write(FREQ_CL);

     //Configuraciones generales
     Soft_I2C_Write(0xB0);
     Soft_I2C_Write(0x10);
     Soft_I2C_Write(0x00);
     Soft_I2C_Stop();
     GIE_bit = 1;

}



//********************** ACCIONADOR DE BOTONES ****************************
//Este procedimiento realiza los cambios necesarios a las variables
//   dependiendo de la página en la que se encuentre el usuario, dentro del
//   menú de configuraciones
void moveSpecial(){
     unsigned short AUX;

     //Se revisa si está en la línea de sintonización
     if (ACT_LIN == 0){

          //Si el usuario solicitó el cambio de frecuencia hacia abajo,
```

```
          //   se decrementa la variable de sintonización
          if (CONF_FLAG.B2 == 1){
               FREQ -= 2;
               if (FREQ < 875) FREQ = 1079;
          }
          //Si el usuario solicitó el cambio de frecuencia hacia arriba,
          //   se incrementa la variable de sintonización
          else{
               FREQ += 2;
               if (FREQ > 1079) FREQ = 875;
          }

          //Se almacenan los cambios realizados a la variable
          AUX = FREQ%10;
          EEPROM_Write(0xEF, AUX);
          AUX = FREQ/10;
          EEPROM_Write(0xEE, AUX);

          //Se sintoniza la nueva frecuencia deseada
          sintonizar();
     }
     else{

          //Se cambia el estado del menú correspondiente a la línea en la
          //   que el usuario se encuentre
          AUX = 0b00000001;
          AUX <<= (ACT_LIN - 1);
          SHOW_MENU ^= AUX;

          //Se guardan los cambios
          EEPROM_Write(0xE0, SHOW_MENU);

          Delay_ms(10);
     }
}



//********************** IDENTIFICADOR DE BOTONES **************************
//Procedimiento que revisa el botón presionado, y toma las medidas según sea
//   el caso
void newButton(){
     unsigned short AUX;

     //Se revisan los estados de los botones
     switch (PORTB & 0b00101100){

          case 0b00100100:
               //Fue presionado RB3 (MÁS)
               if (ACT_PAG != 0){

                    //Si no se está mostrando el mensaje principal, se
                    //   visualiza la siguiente línea
                    ACT_LIN++;
```

```
                    //Si se alcanzó la última línea, se visualiza nuevamente
                    //   el primer mensaje
                    if (ACT_LIN == (LIN_M[ACT_PAG - 1] + 1)) ACT_LIN = 0;
                    ACT_CHAR = 0;
                    TMR_CHR = 0;

                    //Se indica que se deben mostrar los cambios en los
                    //   mensajes
                    MY_FLAGS |= 0b00110100;

                    if (ACT_PAG == 5) CONF_FLAG.B0 = 1;
               }
               else CONF_FLAG.B0 = 1;
               break;

          case 0b00101000:
               //Fue presionado RB2 (IZQUIERDA)

               if (ACT_PAG != 5){

                    //Se busca el siguiente menú a la izquierda que se
                    //   encuentre activo
                    AUX = SHOW_MENU << 1;
                    AUX.B0 = 1;
                    if (ACT_PAG == 0){
                         AUX <<= 3;
                         ACT_PAG = 5;
                    }
                    else AUX <<= (8 - ACT_PAG);

                    //Se reduce la página actual
                    ACT_PAG--;
                    while (AUX.B7 != 1){
                         AUX <<= 1;
                         ACT_PAG--;
                    }

                    //Se reinician las variables de visualización
                    ACT_LIN = 0;
                    ACT_CHAR = 0;
                    TMR_CHR = 0;
                    MY_FLAGS |= 0b00110100;

               }
               else{

                    //Se está haciendo una configuración en el menú especial
                    MY_FLAGS |= 0b00100100;
                    CONF_FLAG.B2 = 1;
                    moveSpecial();
               }
               break;

          case 0b00001100:
               //Fue presionado RB5 (DERECHA)
```

```
                if (ACT_PAG != 5){

                     //Se busca el siguiente menú a la derecha que se encuentre
                     //   activo
                     AUX = (SHOW_MENU | 0x10) >> ACT_PAG;
                     ACT_PAG++;
                     while (AUX.B0 != 1){
                          AUX >>= 1;
                          ACT_PAG++;
                     }
                     if (ACT_PAG == 5) ACT_PAG = 0;

                     //Se reinician las variables de visualización
                     ACT_LIN = 0;
                     ACT_CHAR = 0;
                     TMR_CHR = 0;
                     MY_FLAGS |= 0b00110100;
                }
                else{

                     //Se está haciendo una configuración en el menú especial
                     MY_FLAGS |= 0b00110100;
                     CONF_FLAG.B2 = 0;
                     moveSpecial();
                }
                break;

     }

     //Se bloquean los botonazos por 500ms
     MY_FLAGS.B0 = 0;
     TMR_BTN = 0;

}



//********************** INICIALIZACIÓN DE VARIABLES ************************
//Procedimiento que inicializa las variables principales
void dataInit(){
     unsigned short CONT;

     //Reinicio del RadioText
     resetRadioText();

     //Se llena el mensaje actual sólo con espacios en blanco
     for (CONT = 0; CONT < 65; CONT++){
          MEN_ACT[CONT] = 0x20;
          MEN_PRINC[CONT>>2] = 0x20;
     }

     //Inicialización de otras variables
     //Finales de cadena
```

```
     TEMP_MEN[16] = 0x00;
     TIT_ACT[16] = 0x00;
     MEN_ACT[65] = 0x00;
     MEN_PRINC[16] = 0x00;
     FREQ = EEPROM_Read(0xEE) * 10 + EEPROM_Read(0xEF);

     for (CONT = 0; CONT < 4; CONT++){

          //Ningún mensaje al iniciar el programa
          LIN_M[CONT] = 0;

          //Los apuntadores a las cadenas más recientes se inicializan en 3
          ADD_TXT[CONT] = 3;
     }
     LIN_M[4] = 4;

     //Se inicializan los menús activos, de acuerdo a la información
     //   almacenada en EEPROM
     SHOW_MENU = EEPROM_Read(0xE0);
}



//************************** PROCEDIMIENTO PRINCIPAL ************************
//Procedimiento principal
void main() {

     //Configuración de oscilador interno
     OSCCON = 0b01111100;

     //Configuración de puertos
     //Puertos digitales
     //Salidas RA0:RA3,RA5 (COG)
     //Entrada RA4 (DATA RDS)
     //Entrada RB0 (CLOCK RDS), RB2,B3,B5 (Botones)
     //Salidas RB6:RB7 (COG), RB1,RB4 I2C
     ANSEL = 0;
     ADCON0 = 0;
     ADCON1 = 0;
     PORTA = 0;
     PORTB = 0;
     TRISA = 0b00010000;
     TRISB = 0b00101101;

     //Inicialización de variables
     dataInit();

     //Inicialización de TIMER1
     T1CON = 0b00110001;
     PIR1 = 0b00000000;
     TMR1H = 0x0B;
     TMR1L = 0xDC;

     //Inicialización de pantalla COG
```

```
Lcd_Init();
Lcd_Load();
//Limpia la pantalla
Lcd_Cmd(_LCD_CLEAR);
//Se esconde el cursor
Lcd_Cmd(_LCD_CURSOR_OFF);
MY_FLAGS |= 0b00110100;

//Inicialización de I2C
GIE_bit = 0;
Soft_I2C_Init();
sintonizar();

//Configuración de interrupciones
//PORTB Pull-Up activo
//RB0/INT en flanco de caída
OPTION_REG = 0b00111111;
//Timer1 habilitado
PIE1 = 0b00000001;
//Global interrupt
//Interrupciones periféricas
//RB0/INT habilitado
INTCON = 0b11010000;

//Ciclo principal
while (1){

     if (CONF_CNT == 8){

          //Han pasado 2 segundos de solicitud de cambio, se realiza
          //Primero se cambia la página a visualizar
          if (CONF_FLAG.B1 == 0) ACT_PAG = 5;
          else ACT_PAG = 0;
          ACT_LIN = 0;
          ACT_CHAR = 0;
          TMR_CHR = 0;

          CONF_FLAG ^= 0b00000010;
          //Cambio de textos
          MY_FLAGS |= 0b00110100;
          MY_FLAGS.B0 = 0;

          CONF_CNT = 0;
     }

     //Se revisa si se presionó un botón
     if (MY_FLAGS.B0 == 1)
          if ((PORTB & 0b00101100) != 0b00101100) newButton();

     //Se revisa si se debe agregar un bit a la cadena de revisión
     if (BUFF_I > 0) pushBit();

     //Se revisa si el tiempo que ha pasado solicita un cambio de
     //   índice
     if (ACT_PAG < 5){
```

```
                if (MY_FLAGS.B3 == 1) adjustIndex();

                //Si se está solicitando, se debe cambiar el mensaje (incluye
                //   mensajes en la línea superior e inferior)
                if (MY_FLAGS.B2 == 1) changeMessage();
           }
           else{
                MY_FLAGS.B3 = 0;
                if (MY_FLAGS.B2 == 1) changeMessageSpecial();

           }

           //Si se tiene información lista para buscar, se muestra el
           //   mensaje nuevo
           if (MY_FLAGS.B3 == 0) showMessage();

     }
}
```

B. CÓDIGO DEL PROGRAMA DEL CONFIGURADOR DE MENÚS DE LA COMPUTADORA

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Collections;

namespace RDS_Selector
{
    public partial class Form1 : Form
    {


        //Constructor de la forma
        public Form1()
        {
            InitializeComponent();
        }



        //Procedimiento que solicita al microcontrolador la configuración
        //      del los mensajes activos
        public void RequestConfig()
        {
            try
            {

                //Variable donde se almacenará el mensaje de
                //    activación 0x55
                byte[] Data = new byte[1];

                //Se inicializa el puerto serial
                serialPort1.Open();

                //Petición de estado actual de configuración
                Data[0] = (byte)170;
                serialPort1.Write(Data, 0, 1);

                //Inicio del Timer, útil para rescatar el procedimiento en
                //      caso de encontrarse un error en la transmisión
                timer1.Start();

            }
            catch (Exception)
            {
```

```
        //Ocurrió algún error de conexión
        MessageBox.Show("Verifique conexión del dispositivo");

    }
}



//Procedimiento que lee la configuración reportada por el
//      microcontrolador
public void ReadConfig()
{
    //Inicialización de procedimiento. Se detiene el Timer, no
    //      hubo problemas de comunicación
    byte[] configFlag = new byte[1];
    timer1.Stop();

    try
    {

        //Lectura de la información enn el buffer
        serialPort1.Read(configFlag, 0, 1);
        BitArray bitsFlag = new BitArray(configFlag);

        //Se actualizan los estados de los CheckBox, dependiendo
        //      de la información que se obtiene de la comunicación
        Invoke(new Action(() =>
        {
            cb_noti.Checked = bitsFlag[1];
            cb_depo.Checked = bitsFlag[2];
            cb_traf.Checked = bitsFlag[3];
        }));

    }
    catch (Exception)
    {
        //Ocurrió algún problema en la comunicación
        MessageBox.Show("Recepción errónea");
    }

    serialPort1.Close();

}



//Procedimiento que envía la configuración escogida hacia el
//      microcontrolador
public void SendConfig()
{
    try
    {

        //Variables de envío
        BitArray bitsFlag = new BitArray(8);
```

```
            byte[] Data = new byte[1];

            //Inicialización de variables, en función del estado de
            //      los CheckBox
            bitsFlag[1] = cb_noti.Checked;
            bitsFlag[2] = cb_depo.Checked;
            bitsFlag[3] = cb_traf.Checked;
            bitsFlag.CopyTo(Data, 0);

            //Envío de la información
            serialPort1.Open();
            serialPort1.Write(Data, 0, 1);
            serialPort1.Close();

        }
        catch (Exception)
        {
            //Problema en la conexión del dispositivo
            MessageBox.Show("Verifique conexión del dispositivo");
        }
    }



    //Procedimientos de los botones
    #region Botones
    private void getConfig_Click(object sender, EventArgs e)
    {
        //Solicitud de configuración
        RequestConfig();
    }
    private void setConfig_Click(object sender, EventArgs e)
    {
        //Envío de la configuración
        SendConfig();
    }
    #endregion



    //Procedimientos para evetos especiales
    #region Controladores
    private void timer1_Tick(object sender, EventArgs e)
    {
        //Se completaron 500ms y no se recibió respuesta del
        //      microcontrolador
        timer1.Stop();
        serialPort1.Close();
        MessageBox.Show(@"No se pudo establecer comunicación con
            el dispositivo");
    }
    private void serialPort1_DataReceived(object sender,
        SerialDataReceivedEventArgs e)
    {
        //Nuevo byte recibido, almacenado en el buffer
```

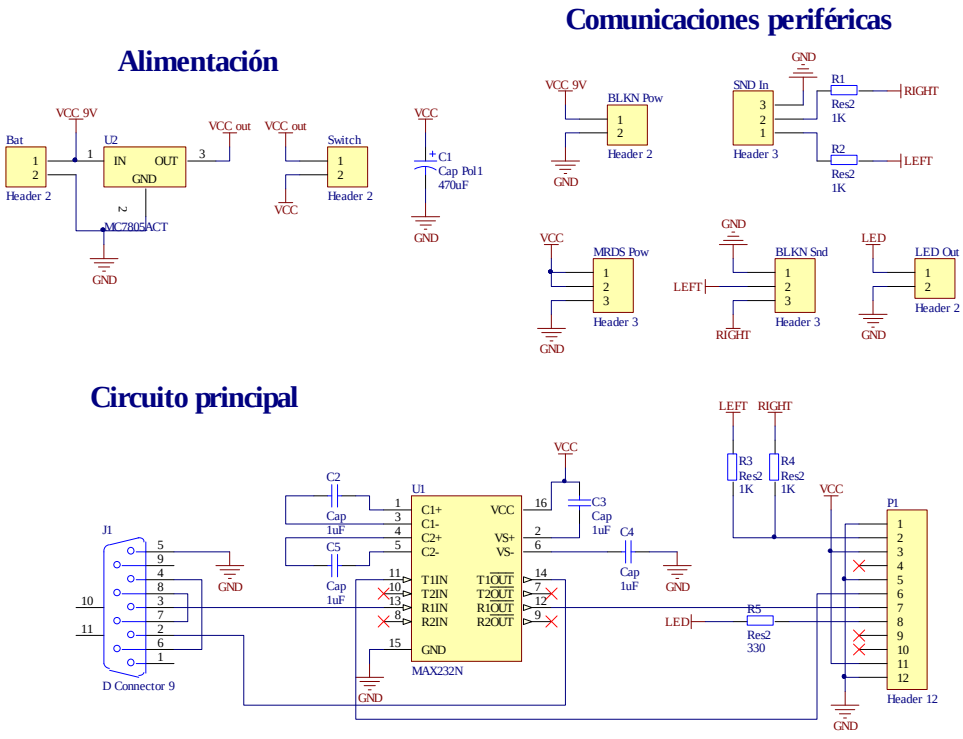
```
            ReadConfig();
        }
        #endregion

    }
}
```

C. DISEÑO DE CIRCUITOS

1. Circuito transmisor

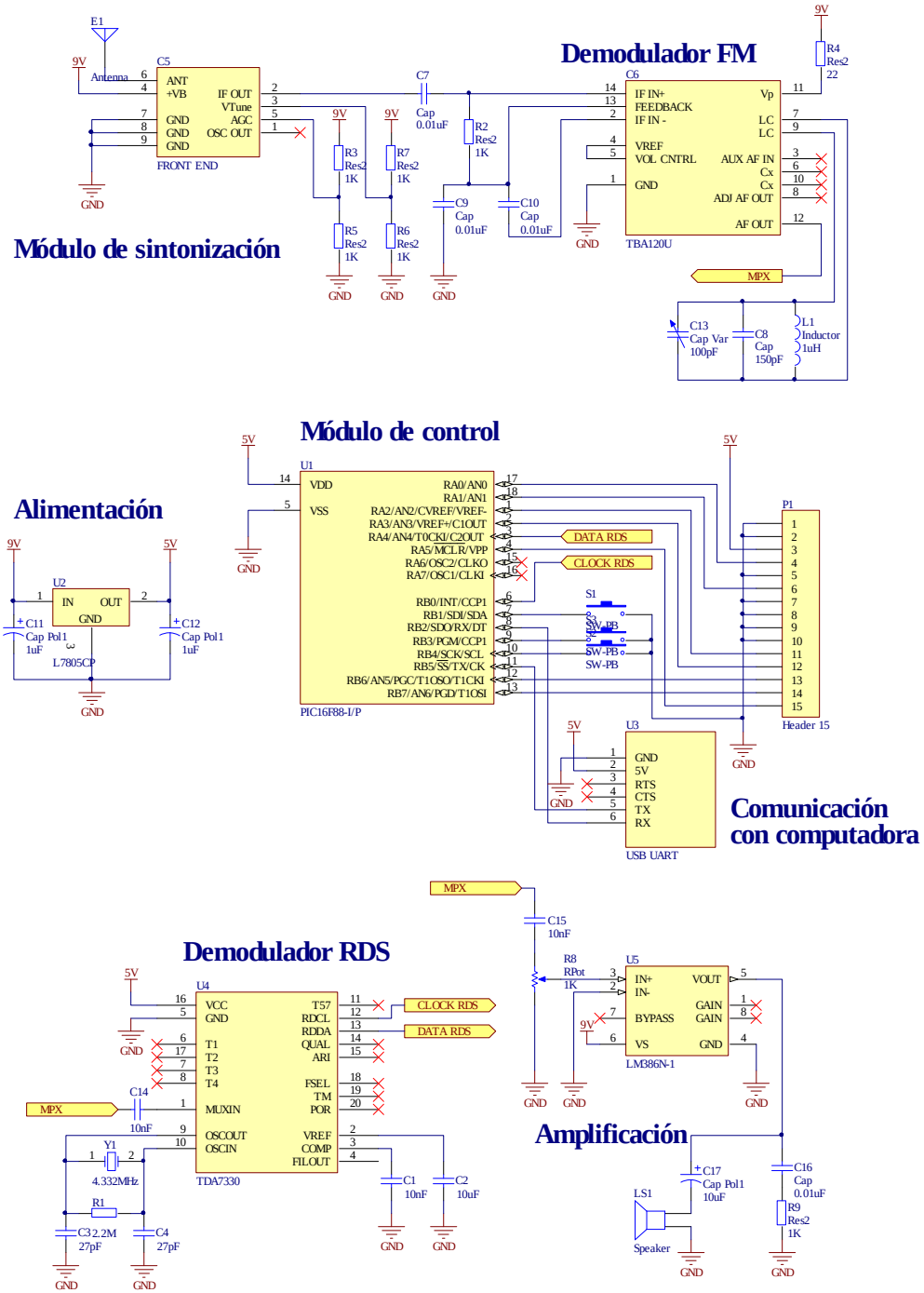
Figura 45. Diseño del circuito transmisor FM[49]



[49] Elaboración propia, realizado en la herramienta *Altium Designer*

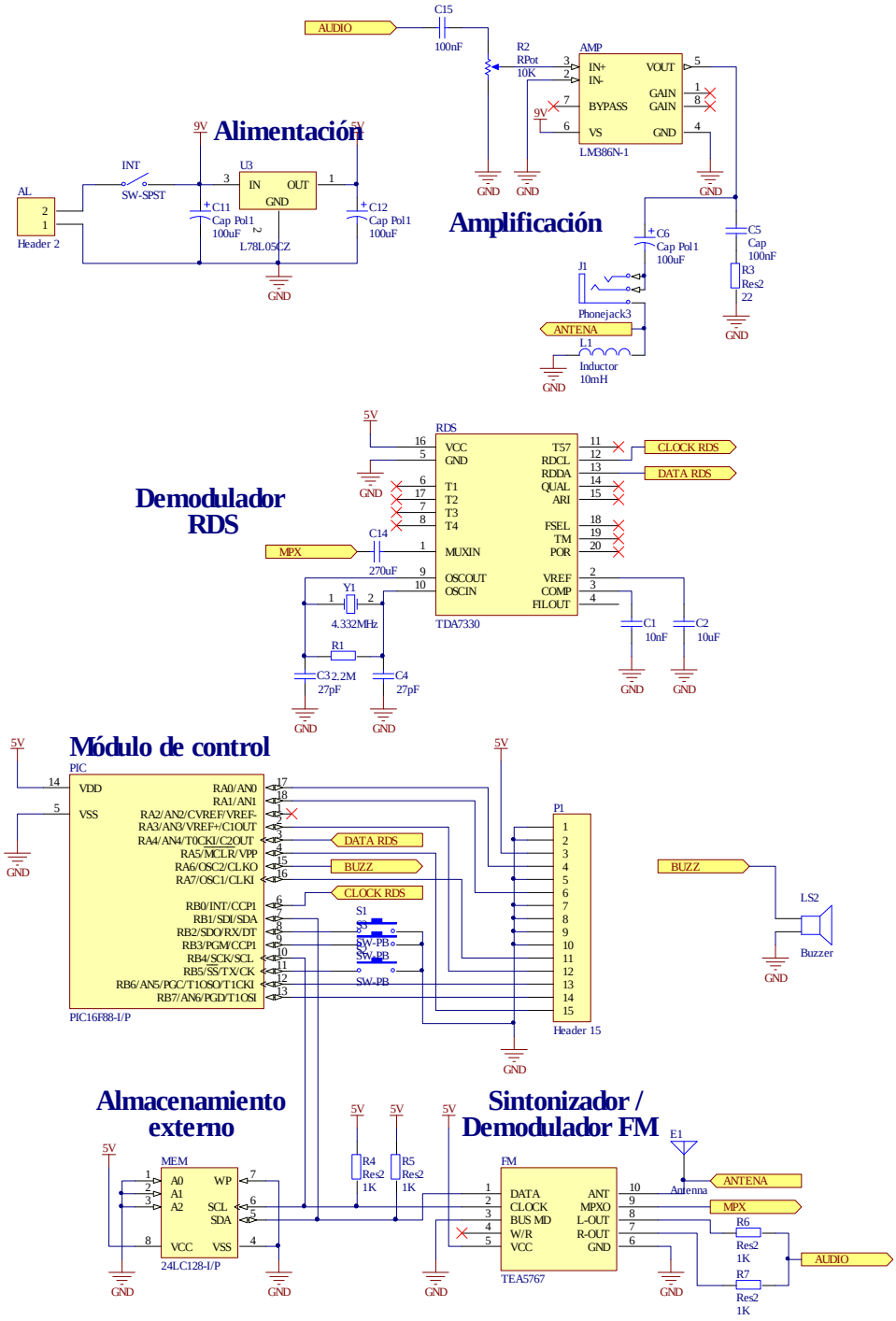
2. Circuito receptor

Figura 46. Diseño del circuito receptor FM, versión preliminar[50]



[50] Elaboración propia, realizado en la herramienta *Altium Designer*

Figura 47. Diseño del circuito receptor FM, versión final[51]



[51] Elaboración propia, realizado en la herramienta *Altium Designer*

D. PRINTED CIRCUIT BOARDS (PCB'S)

Figura 48. Diseño del PCB del circuito transmisor[52]

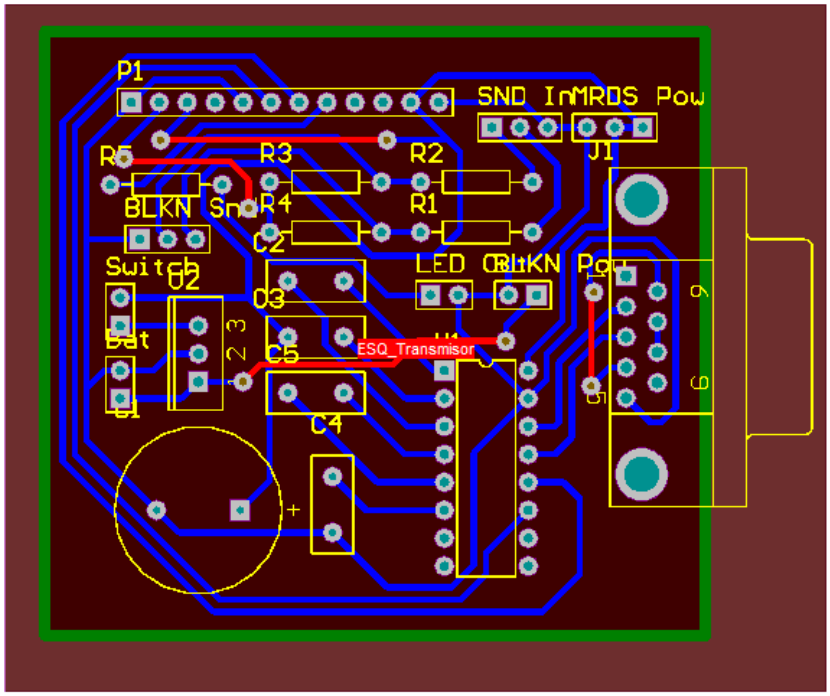
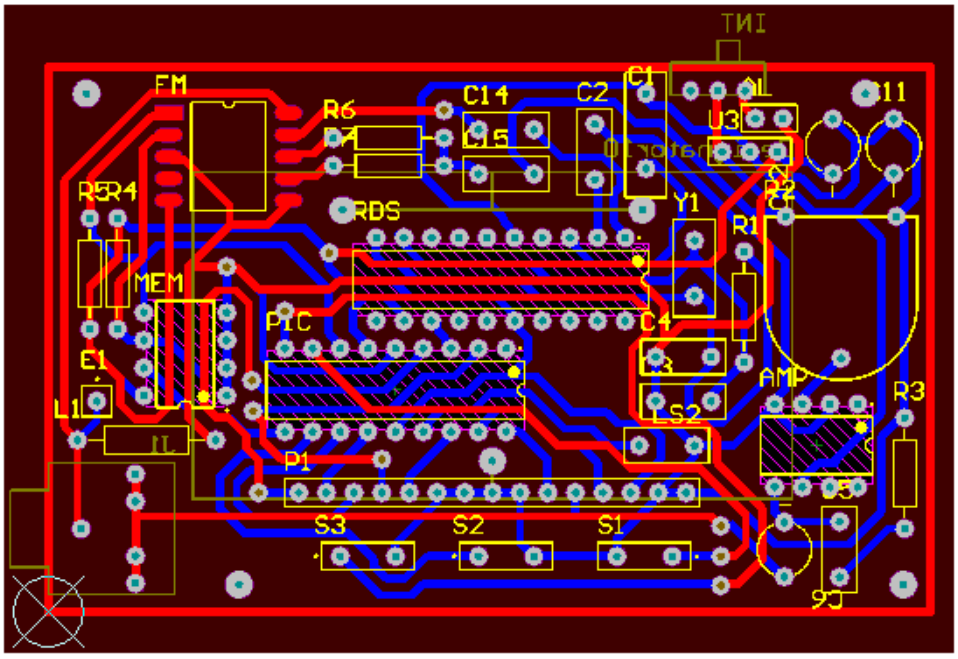


Figura 49. Diseño del PCB del circuito receptor[53]



[52] Elaboración propia, realizado en la herramienta *Altium Designer*
[53] Elaboración propia, realizado en la herramienta *Altium Designer*

E. FOTOGRAFÍAS

Figura 50. Placa utilizada para el circuito transmisor[54]

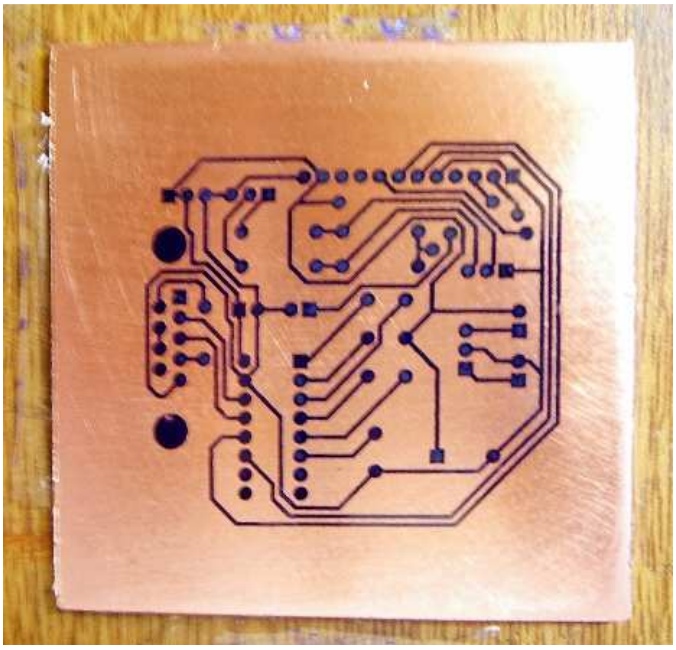
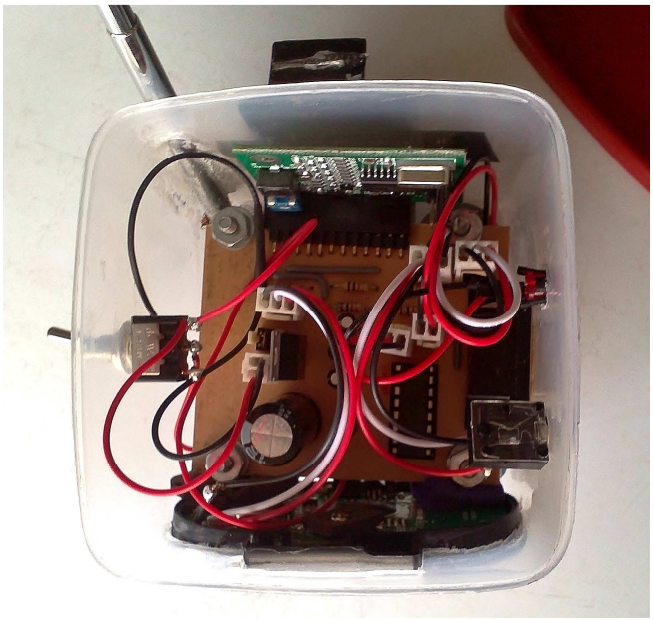


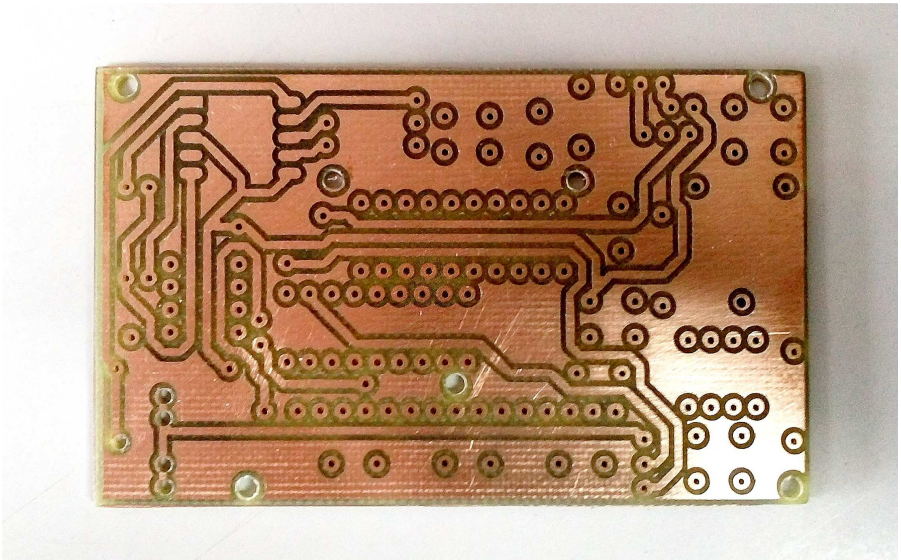
Figura 51. Estructura interna del dispositivo transmisor[55]



[54] Elaboración propia
[55] Elaboración propia

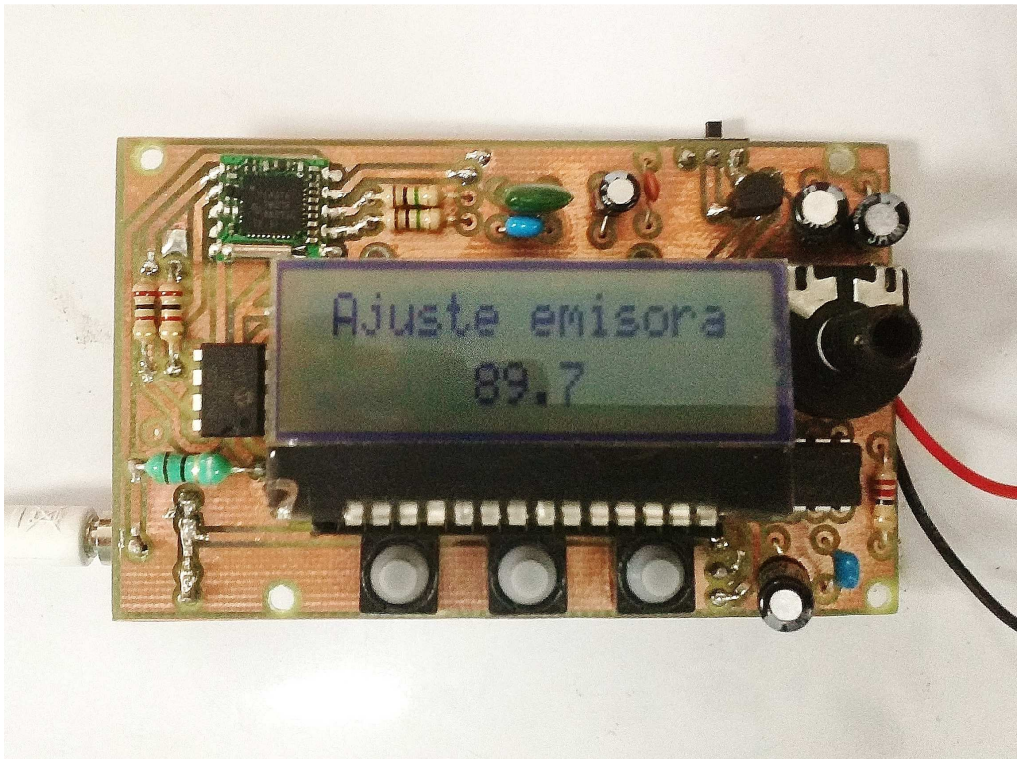
Figura 52. Presentación final del dispositivo transmisor[56]

Figura 53. Placa utilizada para el circuito receptor[57]



[56] Elaboración propia
[57] Elaboración propia

Figura 54. Dispositivo receptor final[58]



[58] Elaboración propia

XII. GLOSARIO

1. Amplitud Modulada (AM): Forma de modulación lineal en donde se varía la amplitud de la señal portadora en función del nivel que maneja la señal que se desea transmitir.

2. Buzzer: Componente electrónico que produce un sonido a una frecuencia específica al aplicársele un voltaje DC.

3. Chip on Glass (COG): Modalidad de control de pantallas LCD, en donde el circuito integrado controlador se ubica en la misma pantalla de visualización. Entre las ventajas más grandes de esta tecnología se encuentra la eficiencia en el aprovechamiento de espacio.

4. Codificación Manchester: Método de codificación de señales digitales binarias en donde cada bit de información es representado como un flanco de subida o bajada. La señal codificada tiene el doble de la frecuencia de la señal de datos, y respeta las siguientes características:
    a. Una transición de nivel alto a bajo corresponde a un 0 lógico.
    b. Una transición de nivel bajo a alto corresponde a un 1 lógico.

5. Frecuencia Modulada (FM): Forma de modulación lineal en donde se varía la frecuencia de la señal portadora en función del nivel que maneja la señal que se desea transmitir.

6. Hashtag: Palabra o frase clave de un mensaje, que ayuda a categorizarlo según el tipo de información que maneja. Es caracterizado por poseer el símbolo # en su inicio. Es un medio que utilizan algunas plataformas como *Twitter* para el manejo de su información.

7. Inter-Integrated Circuit (I^2C): Protocolo de comunicación creado por Phillips especialmente diseñado para la comunicación con dispositivos periféricos mediante únicamente dos conexiones: una de datos y otra de reloj. Usualmente lo componen un solo dispositivo "maestro", encargado de controlar el flujo de información; y uno o más "esclavos", que son componentes periféricos como memorias EEPROM, sensores, etc.

8. Liquid Crystal Display (LCD): Medio de visualización de imágenes (caracteres, figuras, etc.) que se basa en las propiedades de modulación de luz de cristales líquidos.

9. Memoria FLASH: Tecnología no volátil de almacenamiento de información. Se caracteriza por la incapacidad de borrar espacios unitarios de información y, en algunos casos, de escribirlos. Este procesamiento de bloques enteros de información reduce su tiempo de vida, ya que las unidades de almacenamiento serán rescritas aún si no se necesitan.

10. Memoria RAM: Tecnología volátil de almacenamiento de información. Esta es retenida en la medida en que la memoria se mantenga energizada. El tiempo necesario para su acceso es muy rápido, y su vida útil es prácticamente ilimitada.

11. Memoria ROM Programable y Borrable Eléctricamente (EEPROM): Tecnología no volátil de almacenamiento de información. Es más lenta que la tecnología FLASH, pero permite la el borrado, la lectura y escritura de unidades de información. Esta característica le permite también tener un tiempo de vida útil más largo que el de la memoria FLASH.

12. Oscilador Controlado por Voltaje (VCO): Oscilador electrónico que varía su frecuencia en función del voltaje DC aplicado. Es utilizado en sistemas de sintonización, ya que a través de él es posible igualar la frecuencia de resonancia buscada.

13. Primero en entrar, primero en salir (PEPS) / First In, First Out (FIFO): Estructura de datos que organiza la información según su orden de llegada. Al recibir un nuevo paquete de información, es colocado al final de una cola virtual; y al extraer información de ella se obtendrá el primer dato en haber sido almacenado.

14. Protocolo: Conjunto de reglas que rigen la forma de comunicación entre dos o más partes de un sistema.

15. Radio Data Syastem (RDS): Protocolo de comunicación radial, en donde se codifican bloques de información mediante la Codificación Manchester, en la frecuencia de 57kHz de una frecuencia específica del espectro FM.

16. Recommended Standart 232 (RS-232): Es un protocolo de comunicación serial o paralela entre dos o más elementos, que cumplen estándares específicas. Los elementos que le componen son: el Equipo terminal de datos (DTE) y el Equipo de comunicación de datos (DCE).

17. Subportadora: Señal análoga o digital que se transmite dentro de otra señal (portadora), modulada a una frecuencia y ancho de banda específica. Esto permite la comunicación de información adicional en la misma línea de datos.

18. Transistor-Transistor Logic (TTL): Conjunto de circuitos digitales cuya arquitectura se basa en resistores y transistores BJT. Manejan los niveles de 0V y 5V DC, representando los ceros y unos lógicos respectivamente.

19. Universal Asynchronous Receiver/Transmitter (UART): Tipo de comunicación serial o paralela, comúnmente aplicado en protocolos como EIA, RS-232, RS-422 o RS-485. Este brinda la versatilidad de poder ser configurado según diferentes velocidades de transmisión y formatos de datos.

20. Universal Serial Bus (USB): Conjunto de estándares que rigen el protocolo de comunicación, los cables y conectores que deben usarse para la utilización de esta tecnología. Fue creado para estandarizar la forma de comunicar elementos periféricos como impresoras, ratones, cámaras digitales, teclados, etc.