UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

INGENIERO EN ELECTRÓNICA

SISTEMA DECODIFICADOR FM-RDS

AUTOR: Juan Martínez Tascón

TUTOR: Jesús M. Hernández Mangas

Febrero 2004

TITULO:	**SISTEMA DECODIFICADOR FM-RDS**
AUTOR:	Juan Martínez Tascón
TUTOR:	Dr. Jesús Manuel Hernández Mangas
DEPARTAMENTO:	Electricidad y Electrónica

Miembros del Tribunal

PRESIDENTE:	D. Luis Alberto Bailón
VOCAL:	D. Jesús Arias
SECRETARIO:	D. Jesús M. Hernández
SUPLENTE:	D. José Vicente Antón
FECHA DE LECTURA:	
CALIFICACIÓN:	

RESUMEN DEL PROYECTO

El proyecto realizado se trata de un sistema decodificador FM-RDS, el cual, a partir de la información horaria contenida en la señal RDS que emiten algunas emisoras, actualiza un reloj en tiempo real RTC.

Previamente, se ha realizado un estudio de los dispositivos disponibles en el mercado capaces de recibir la señal FM (junto a la RDS), para el módulo receptor, y capaces de decodificar la señal RDS, para el módulo decodificador y procesador de dicha señal. A partir de los dispositivos seleccionados, se ha realizado el diseño del sistema y finalmente un prototipo.

ABSTRACT

This proyect is a FM-RDS decoder system, which update a real time clock (RTC) with information obtained from RDS signal, that is sended out from any commercial radio station.

A study about comercial devices has been done previously for FM receiver module and RDS decoder module. Desing and a protoype have been done with chosen devices.

PALABRAS CLAVE

RDS (Radio Data System), Hora, Fecha, Receptor, Decodificador, I^2C (Inter Integrated Circuit).

ÍNDICE

1. INTRODUCCIÓN 4

2. EL SITEMA RDS 6
- 2.1 MODULACIÓN 7
- 2.2 CODIFICACIÓN 10
- 2.3 DECODIFICADOR 11
- 2.4 FORMATO DE LOS DATOS 13
- 2.5 GRUPO 4A: FECHA Y HORA 16

3. EL BUS I2C 18
- 3.1 INTRODUCCIÓN 18
- 3.2 INTRODUCCIÓN A LA ESPECIFICACIÓN DEL BUS I2C 18
- 3.3 EL CONCEPTO DEL BUS I2C 19
- 3.4 CARACTERÍSTICAS GENERALES 21
- 3.5 TRANSFERENCIA DE BITS 22
  - 3.5.1. Validez de datos 22
  - 3.5.2. Condiciones de START y STOP 23
- 3.6 TRANSFIRIENDO DATOS 24
  - 3.6.1. Formato de byte 24
  - 3.6.2. Señal ACK (Acknowledge) 25
- 3.7 ARBITRAJE Y GENERACIÓN DE RELOJ 26
  - 3.7.1. Sincronización 26
  - 3.7.2. Arbitraje 26
- 3.8 FORMATOS DE DIRECCIÓN CON 7 BITS 28
- 3.9 DIRECCIONAMIENTO CON 7 BITS 30
  - 3.9.1. Definición de los bits del primer byte 30
- 3.10 CARACTERÍSTICAS ELÉCTRICAS PARA DISPOSITIVOS I2C 31
  - 3.10.1. Valores máximos y mínimos para Rp y Rs 34

4. ESTUDIO DE LOS DISPOSITIVOS DEL MERCADO 37
4.1 CIRCUITOS INTEGRADOS DE RADIO FM 37
4.1.1. Atmel TDA 1083 37
4.1.2. Panasonic AN 7293 38
4.1.3. Philips TDA 7088T 39
4.1.4. Philips TEA 5757 HL 39
4.1.5. Samsung S1A0427B01 40
4.1.6. Sanyo LA 1800 41
4.1.7. Sony CXA 1611 42
4.1.8. Philips TDA 7000 42
4.2 CONCLUSIONES 43
4.3 CIRCUITOS DECODIFICADORES DE RDS 44
4.3.1. Philips SAA 6579 44
4.3.2. ST-Microelectronics TDA 7330 45
4.3.3. Philips SAA 6588 45
4.4 CONCLUSIONES 46

5. EL SISTEMA DECODIFICADOR FM-RDS 47
5.1 EL CIRCUITO RECEPTOR DE RADIO FM 47
5.1.1. El circuito integrado TDA 7000 48
5.1.2. El conversor digital-analógico MAX 5382 50
5.1.3. El circuito de sintonización 50
5.1.4. El circuito de recepción 52
5.1.5. La señal compuesta MPX 54
5.2 EL SISTEMA DECODIFICADOR RDS 54
5.2.1. El preprocesador RDS 57
5.2.2. Cristal de cuarzo para el decodificador RDS 59
5.2.3. Microcontrolador PIC 16F628 59
5.2.4. Reloj/Calendario PCF8583 61
5.3 BLINDAJE Y AISLAMIENTO DE LOS CIRCUITOS 62

6. FUNCIONAMIENTO DEL SISTEMA 64
6.1 FUNCIONAMIENTO DEL CIRCUITO RECEPTOR DE RADIO FM 64
6.2 FUNCIONAMIENTO DEL CIRCUITO DECODIFICADOR DE RDS 67
6.3 ACTUALIZACIÓN DE LA FECHA Y LA HORA 72

7. PRESUPUESTO 74
7.1 COSTE DEL MÓDULO RECEPTOR DE RADIO FM 74
7.2 COSTE DEL MÓDULO DECODIFICADOR RDS 74
7.3 COSTE TOTAL DEL MATERIAL 74
7.4 COSTE DE INGENIERÍA 75
7.5 COSTE DE PRODUCCIÓN 75

APÉNDICE A: CÓDIGO DEL PROGRAMA 76

APÉNDICE B: ESQUEMAS ELÉCTRICOS 124

APÉNDICE C: LAYOUTS 126

APÉNDICE D: PROTOTIPO 127

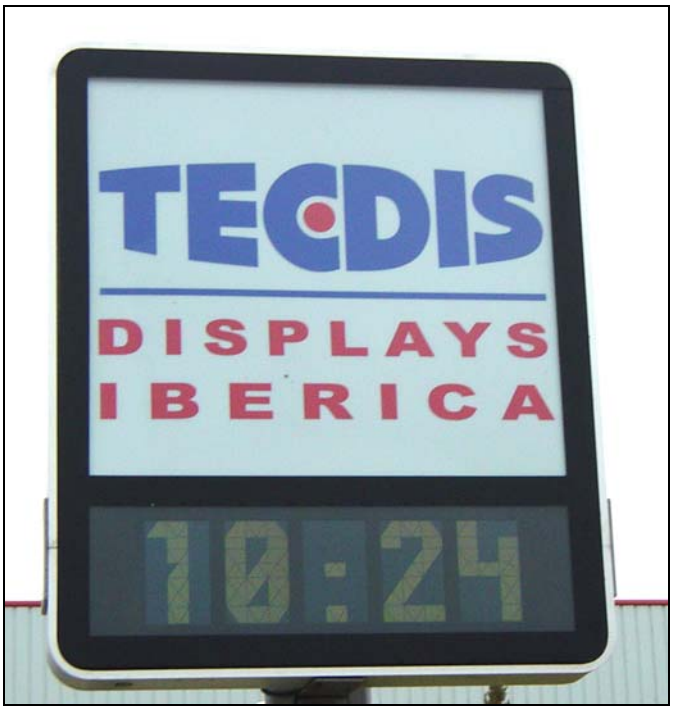
APÉNDICE E: DATASHEETS 130

BIBLIOGRAFÍA

1. INTRODUCCIÓN

El presente proyecto surge de la necesidad de obtener un reloj que siempre muestre la hora correcta, incluso después de los cambios horarios (verano e invierno), o tras un fallo en el suministro eléctrico. Esta necesidad nos fue planteada por la empresa TECDIS DISPLAYS IBÉRICA, situada en el Parque Tecnológico de Boecillo, en Valladolid, y que fabrica pantallas de cristal líquido ó LCD's.

Esta empresa necesitaba que en las pantallas de cristal líquido que muestran la fecha y la hora, estas estuviera siempre actualizas, sin necesidad de ir "poniendo en hora" todos los relojes de estos LCD's, uno a uno.



LCD mostrando la hora.

La solución propuesta se trata de un reloj actualizado mediante la información horaria emitida, vía RDS, por algunas emisoras de radio en la banda comercial (88-108 MHz). Por lo tanto, nuestro objetivo será el de diseñar y fabricar un prototipo que actualice un reloj de tiempo real (RTC), en este caso, el PCF8583, que es el RTC que utiliza la empresa TECDIS en sus paneles LCD.

La idea es obtener la información dela hora y la fecha que emiten algunas emisoras de FM en la banda comercial. Dicha información se transmite en la señal RDS (Radio Data System), entre otras cosas.

Por lo tanto, el sistema a diseñar ha de ser capaz de sintonizar las emisoras, identificarlas, y si son emisoras que emiten la información relativa a la fecha y la hora, fijarlas y comenzar a decodificar la señal RDS. Esta información se enviará al RTC actualizándolo.

En la siguiente figura podemos ver un diagrama de bloques de lo que pretendemos conseguir en este proyecto. Podemos ver como todo el sistema está gestionado por un microcontrolador para hacerlo totalmente autónomo.

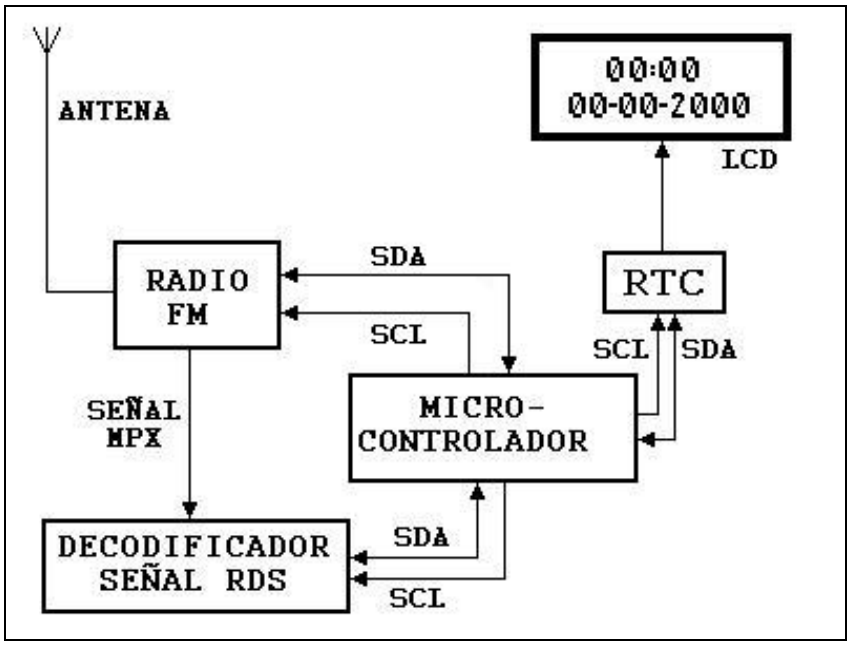


Diagrama de bloques del sistema a diseñar.

Tambien debemos hacer que las comunicaciones entre el RTC, nuestro sistema y el exterior se realicen por medio de un bus serie de tipo I2C (Inter-Integrated Circuit), que consta de dos líneas, una para la señal de reloj (SCL) y otra para los datos (SDA).

Una vez visto cual es el objeto de este proyecto, vamos a empezar con uas nociones teóricas sobre la señal RDS y después dobre el bus I2C. Posteriormente, pasaremos a tratar lo que es el diseño del sistema en si.

2. EL SISTEMA RDS

El sistema RDS (*Radio Data System*) es un sistema de transmisión de datos por emisoras de radio FM comerciales en sus canales de emisión regular (banda de 88 MHz 108 MHz), sin afectar la calidad del audio normalmente transmitido. Los datos transmitidos proveen de una serie de servicios al público con receptores de radio RDS, y permite contar con novedosos servicios a través de equipos de aplicación específica.

En esencia, la idea del sistema RDS es enviar datos en forma digital junto con una señal de radio en frecuencia modulada FM. Los datos transmitidos pueden llegar a un gran número de usuarios gracias a la amplia cobertura de la red de emisoras FM y a un costo mínimo por parte de éstas. La información enviada con el sistema de RDS puede ser muy diversa:

- Identificación de la emisora,
- Frecuencias alternativas de la misma emisora,
- Fecha y hora local,
- Información sobre los programas emitidos,
- Radiotexto,
- Servicio de buscapersonas,
- Telecontrol,
- Etc.

Dependiendo del equipo receptor utilizado y de la aplicación, el sistema RDS puede prestar múltiples servicios a los usuarios, entre los cuales pueden citarse:

- Presentación del nombre de la emisora en la pantalla del radiorreceptor,
- Traducción de música o comentarios en la pantalla del radiorreceptor,
- Sintonización automática de emisiones alternativas en el caso de atenuación de señales.
- Sincronización horaria del radiorreceptor.
- Buscapersonas de gran cobertura sin necesidad de utilizar un canal especial.
- Telecontrol de indicadores de tráfico en carreteras.
- Etc.

Ya con anterioridad se han utilizado los canales de FM comercial para transmitir información adicional. Se destacan el sistema SCA (*Subsidiary Communication Authorisation*) de música ambiental y el sistema ARI (*Autofahrer Rundfunk Information*) de información de tráfico.

El ARI fue el primer sistema de transmisión de información digital mediante subportadora en la banda de FM. Desarrollado en la República Federal Alemana en los años 70, este sistema de información de tráfico es utilizado en Alemania, Austria, Suiza y partes de Estados Unidos.

A mediados de los años 70 se comenzó a pensar en un sistema que pudiera ofrecer más servicios al cliente que el sistema ARI. Así se creó un grupo de trabajo, bajo los auspicios de la Unión Europea de Radiodifusión (UER), con el objetivo de desarrollar un sistema con mayores prestaciones, y cumpliendo con los siguientes objetivos:

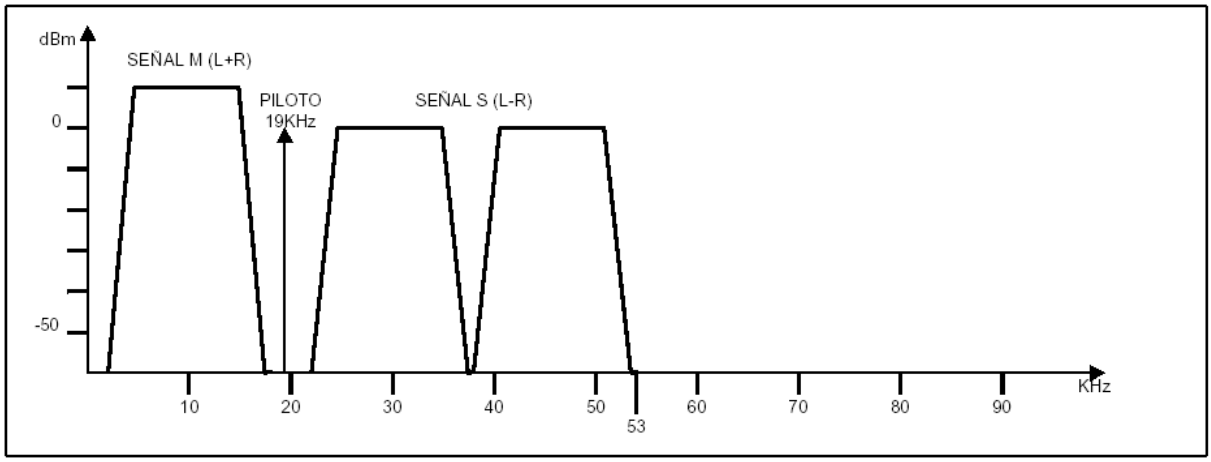
- Las señales del sistema deben ser compatibles con los receptores existentes, sin causar interferencia en la recepción del sonido.
- La recepción debe ser fiable en un área tan grande, por lo menos, como el área de cobertura del programa principal.
- La velocidad de transmisión debe cubrir las necesidades de identificación del programa como la de futuros desarrollos.
- El formato de los mensajes debe ser flexible, permitiendo ajustes para los requerimientos de cada emisora.
- El sistema debe ser capaz de permitir una recepción por medio de receptores de bajo costo.

En base a estas exigencias, en 1984 fueron publicadas las especificaciones del sistema RDS en el documento técnico de la UER 3244.

2.1 MODULACIÓN

El ancho de banda disponible en una señal de compuesta a la entrada del modulador de FM es de unos 90 KHz, de los cuales 53 KHz son ocupados por las señales de audio estéreo.

Los restantes 33 KHz disponibles en la banda pueden ser aprovechados, siempre y cuando se cumplan ciertas condiciones de interferencia entre canales adyacentes.



Espectro de la señal compuesta en la radio FM convencional.

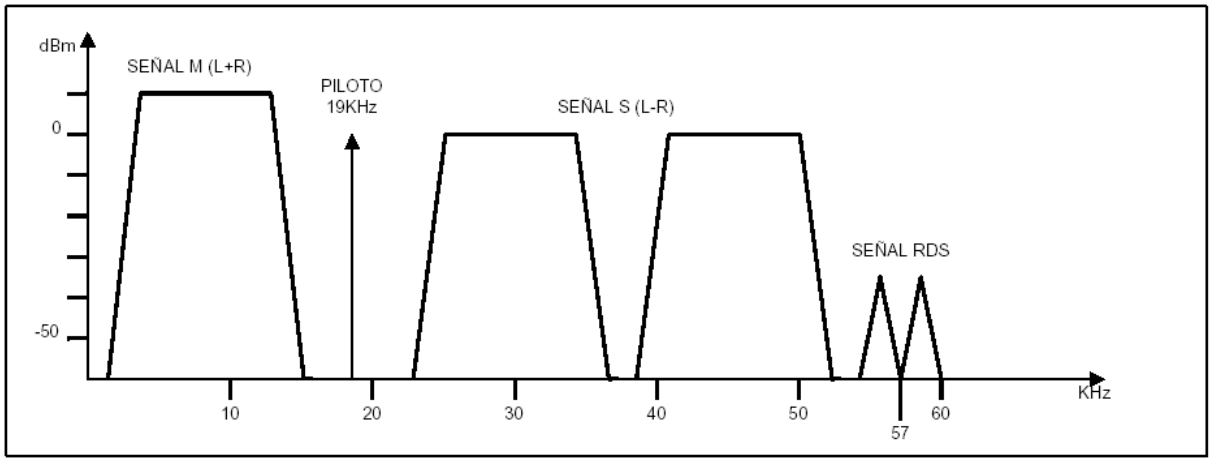
En los años 60 y 70 algunas emisoras comerciales de FM aprovecharon la banda sobrante para la transmisión de programas de audio auxiliares para música ambiental.
Estos sistemas, conocidos por las siglas SCA, no alcanzaron suficiente popularidad, dado que por los niveles de transmisión requeridos, se producía en ocasiones un gran nivel de intermodulación en el programa de audio principal.

Después, en los años 70 nació el sistema ARI de información de tráfico. Es un sistema digital relativamente simple que requiere un sencillo decodificador y emplea una subportadora de 57 KHz.

En el sistema RDS, al igual que en el sistema ARI, se utiliza una modulación en subportadora de 57 KHz. Esta frecuencia está sincronizada en cuadratura con el tercer armónico de la frecuencia piloto para transmisiones estéreo (19 KHz ± 2 Hz x 3 = 57 KHz ± 6 Hz). Durante las emisiones mono, en las cuales no existe frecuencia piloto, la subportadora de RDS, no está sincronizada pero mantiene su valor de 57 KHz ± 6 Hz.

La modulación utilizada es PSK (*Phase Shift Keying*) con desviación de fase de ±90 grados con una tolerancia de ±10 grados. Este sistema de modulación produce un nulo en la frecuencia de subportadora, con toda la energía alrededor de un par de bandas laterales

separadas de la frecuencia central. De esta manera se obtiene una modulación con subportadora suprimida.

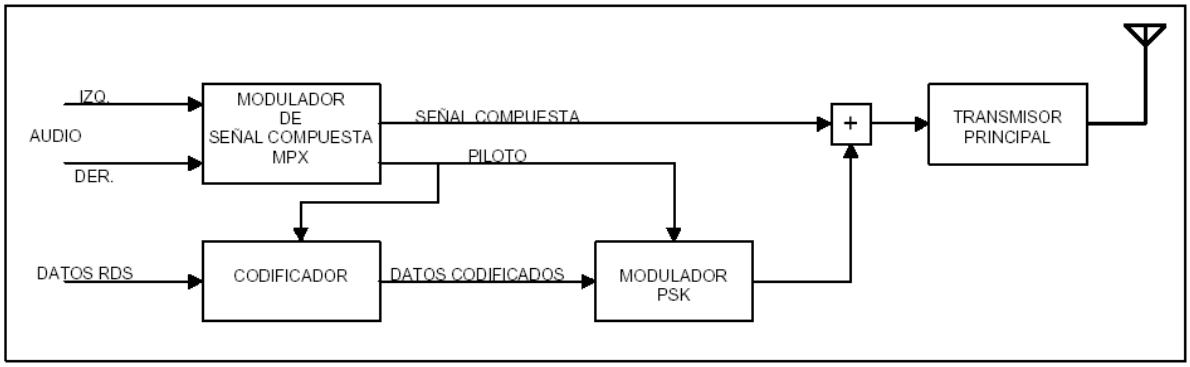


Espectro de la señal compuesta en emisiones FM con sistema rds.

Las bandas laterales de la señal RDS tienen una amplitud pico de 3% de la señal de audio transmitida, y debe ser estrictamente controlada para evitar interferencias que degraden la calidad de audio del programa principal.

En la siguiente imagen podemos ver el espectro de una señal compuesta de una emisora con sistema RDS visualizada en el analizador de espectros.

Señal compuesta con RDS en un analizador de espectros.



Obtención de la señal compuesta de FM con RDS.

La señal RDS es modulada por un codificador binario a un régimen de 1187.5 bps, lo cual corresponde a la portadora piloto dividida por 48 (19 KHz / 48 =1187.5).

2.2 CODIFICACIÓN

Los datos del sistema RDS se codifican en dos etapas, primero por un codificador diferencial y luego por un codificador bifase. Tal y como se mencionó con anterioridad, los datos se codifican a una velocidad de 1187.5 bps , lo cual simplifica el diseño del decodificador.

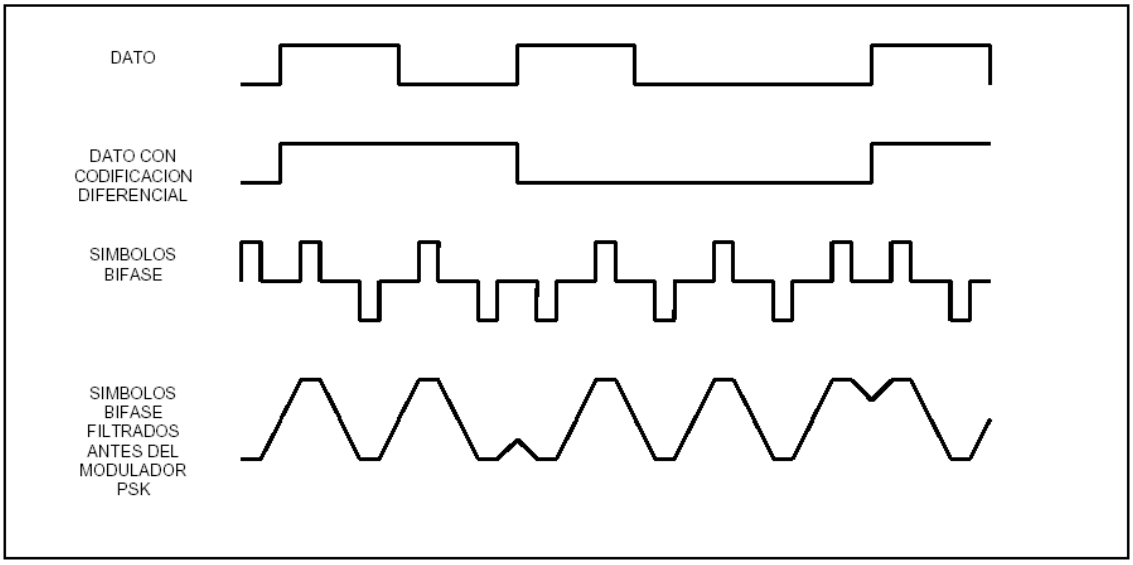
El codificador diferencial tiene como objetivo la recuperación de los ceros y unos en el decodificador aún cuando la señal llegue invertida al receptor. Los datos se codifican según la expresión:

$$\text{Salida} = \text{Entrada actual} \oplus \text{Entrada anterior}$$

El codificador bifase tiene como objetivo incorporar la información del reloj de sincronismo de datos. En esta codificación se envía un positivo seguido de negativo para indicar un ‘uno’ y un negativo seguido de un positivo para indicar un ‘cero’. Para esta operación el codificador requiere un reloj de sincronismo de 2 veces el reloj de transmisión, es decir, 2375 Hz.

Antes de alimentar el modulador FSK, la salida del codificador bifase se hace pasar por un filtro que conforma la señal.

En la siguiente figura se pueden observar las transformaciones que sufre la señal de datos antes de entrar al modulador PSK.



Pasos de codificación de datos para el sistema RDS.

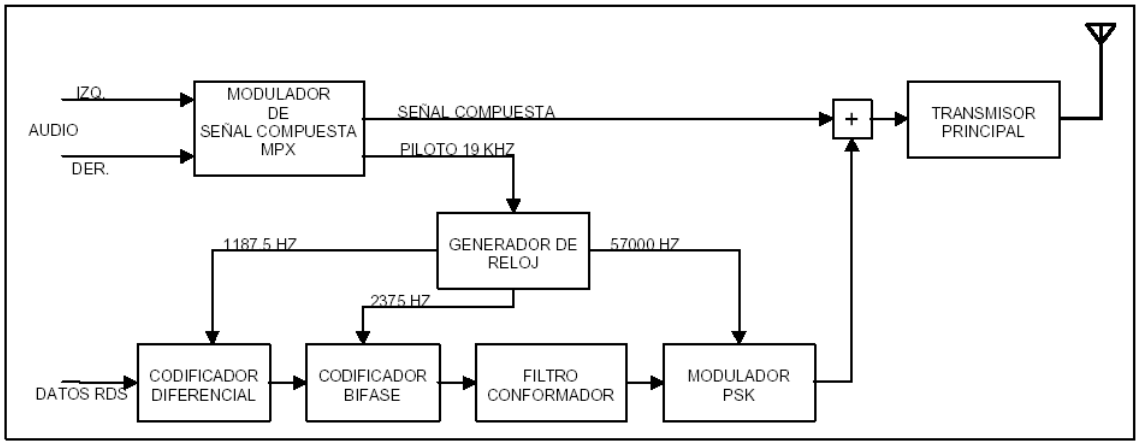


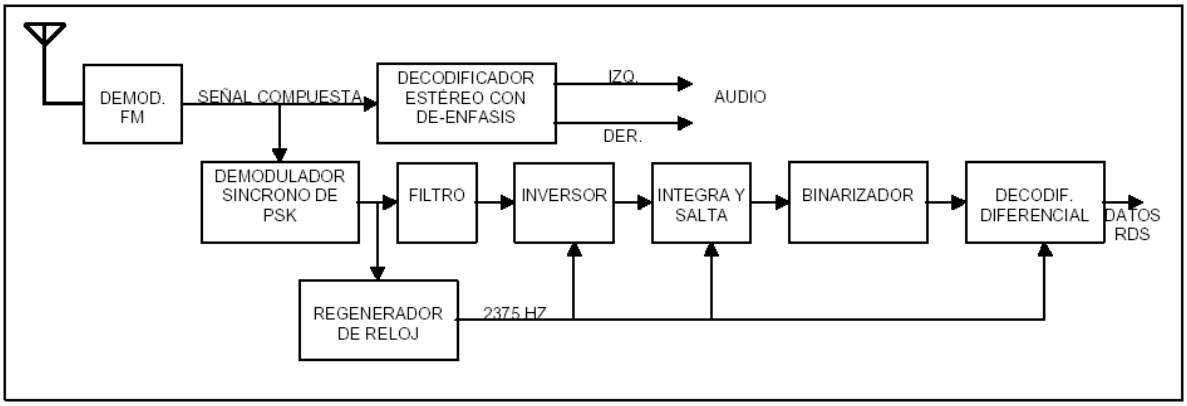
Diagrama de bloques de un transmisor FM con sistema RDS.

2.3 DECODIFICADOR

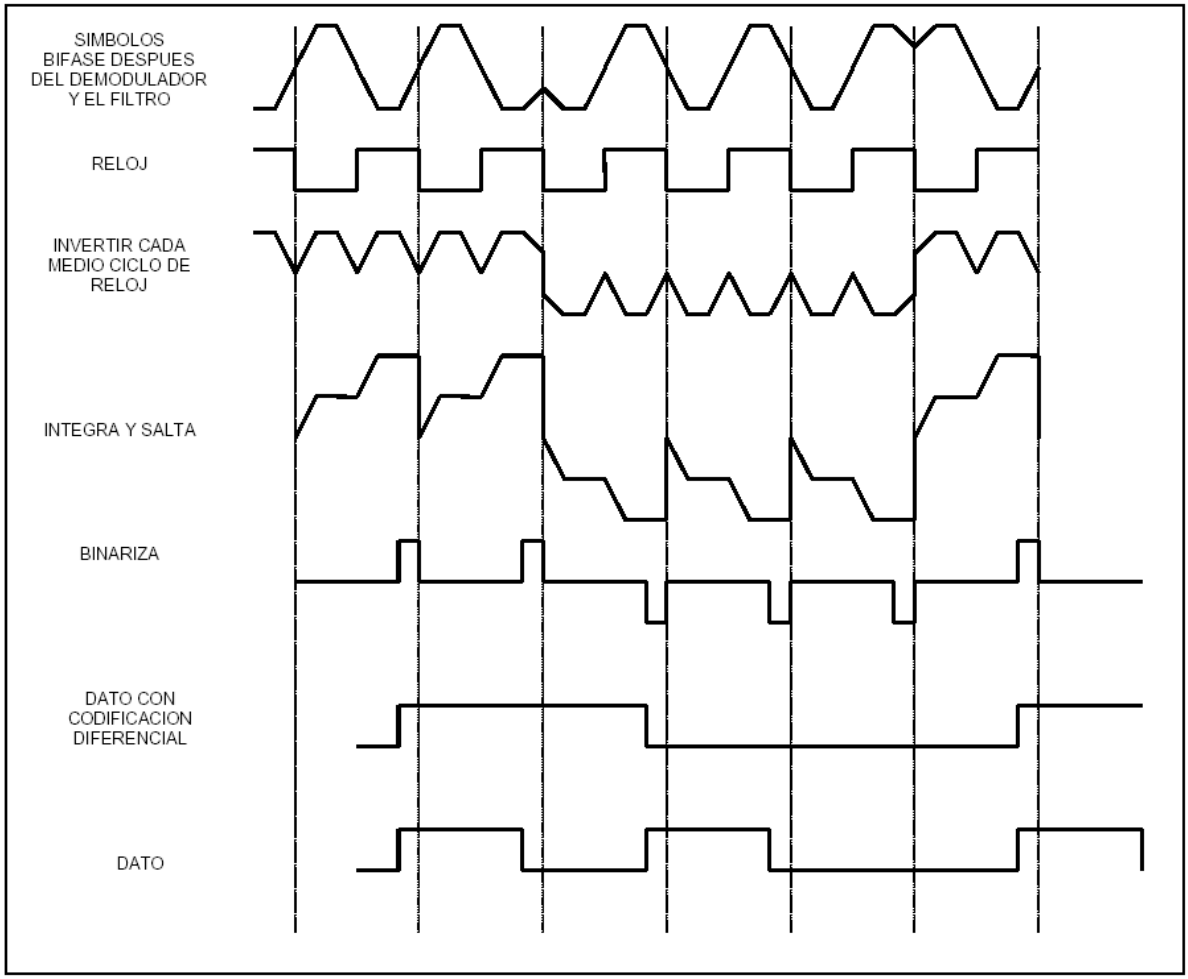
En el receptor RDS, la señal de entrada al decodificador se toma del demodulador de FM, antes de hacerla pasar por el filtro de de-énfasis. Esta señal es filtrada en banda para

separar la señal de 57 KHz RDS y demodularla en forma síncrona. La salida de este demodulador es la señal bifase filtrada que se hace pasar por un inversor un circuito de "integra y salta", por un binarizador y por último pasa al decodificador diferencial que reconstruye los datos del sistema RDS.

La fase del reloj de referencia es muy importante para la decodificación, es por ello que se hace tanto hincapié en la precisión de la sincronización de las diferentes frecuencias en el transmisor.



Demodulador-decodificador RDS.



Proceso de decodificación RDS.

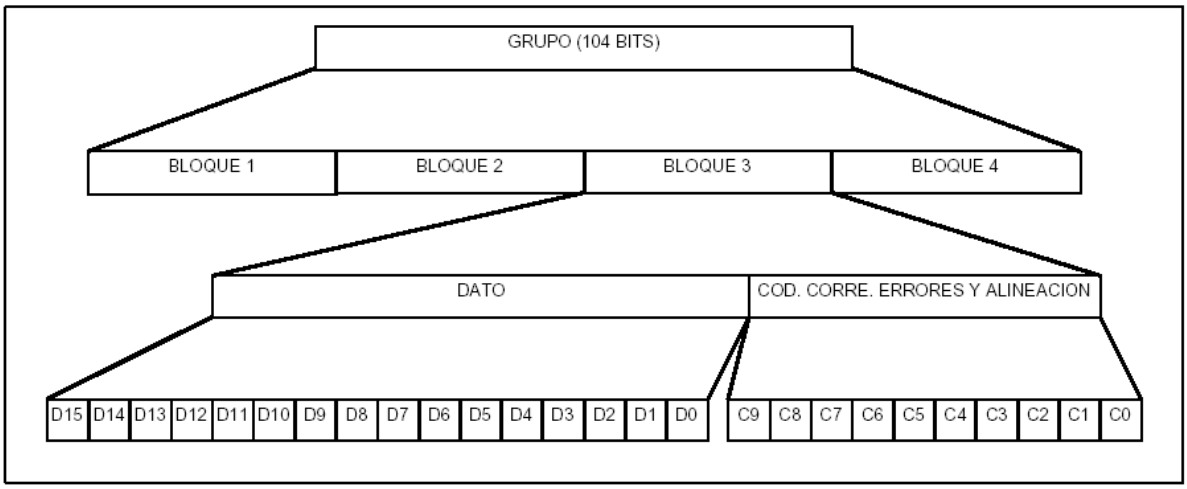
2.4 FORMATO DE LOS DATOS

La especificación EBU 3244-E[2] define el formato de los datos RDS transmitidos, además de la descripción detallada de algunas de las aplicaciones más utilizadas.

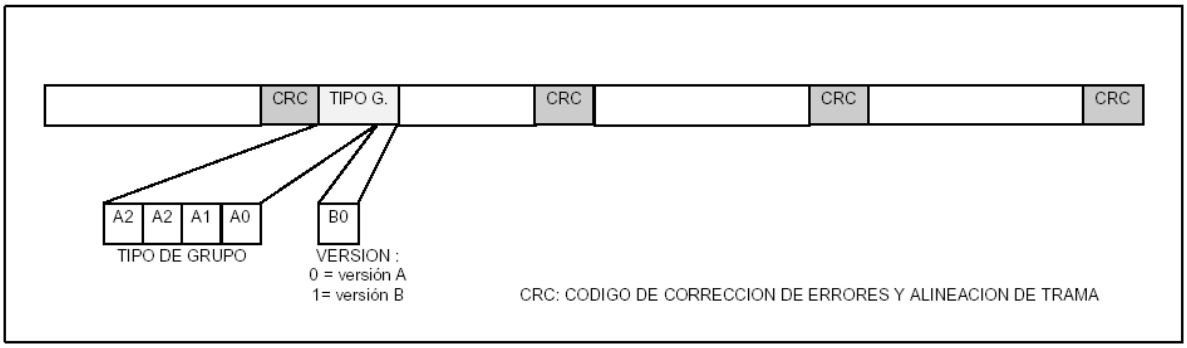
En el sistema RDS los datos son transmitidos en paquetes de 104 bits, denominados *grupos*, divididos en 4 *bloques* de 26 bits. Cada bloque contiene 16 bits de datos y 10 bits de código para corrección de errores y alineación de trama.

Para permitir la flexibilidad en la utilización del sistema, los grupos se dividen en dos versiones de 16 tipos cada uno. Cada tipo de grupo está destinado a un tipo de aplicación particular. El tipo y versión del grupo de datos siempre se hallan indicados en el segundo

bloque del mismo. Cuatro bits identifican los 16 tipos de grupos y un quinto bit identifica si el grupo es versión A ó B.



Paquete de información en el sistema RDS.

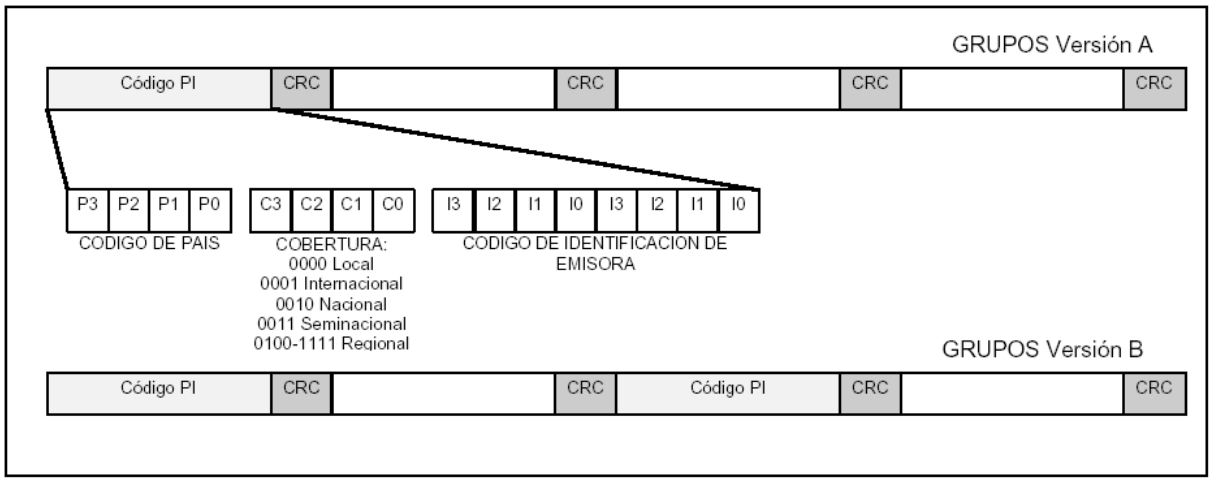


Tipo de grupo y versión.

VERSION A					VERSION B				
A3	A2	A1	A0	APLICACION	A3	A2	A1	A0	APLICACION
0	0	0	0	Información básica de sintonía	0	0	0	0	Información básica de sintonía
0	0	0	1	Información del programa	0	0	0	1	Información del programa
0	0	1	0	Radiotexto	0	0	1	0	Radiotexto
0	0	1	1	Información de otras redes	0	0	1	1	Información de otras redes
0	1	0	0	Hora y fecha	0	1	0	0	-
0	1	0	1	Canales transparentes de datos	0	1	0	1	Canales transparentes de datos
0	1	1	0	Aplicaciones de la emisora	0	1	1	0	Aplicaciones de la emisora
0	1	1	1	Buscapersonas	0	1	1	1	-
1	0	0	0	-	1	0	0	0	-
1	0	0	1	-	1	0	0	1	-
1	0	1	0	-	1	0	1	0	-
1	0	1	1	-	1	0	1	1	-
1	1	0	0	-	1	1	0	0	-
1	1	0	1	-	1	1	0	1	-
1	1	1	0	Soporte ampliado de otras redes	1	1	1	0	Soporte ampliado de otras redes
1	1	1	1	-	1	1	1	1	Información de sintonía rápida

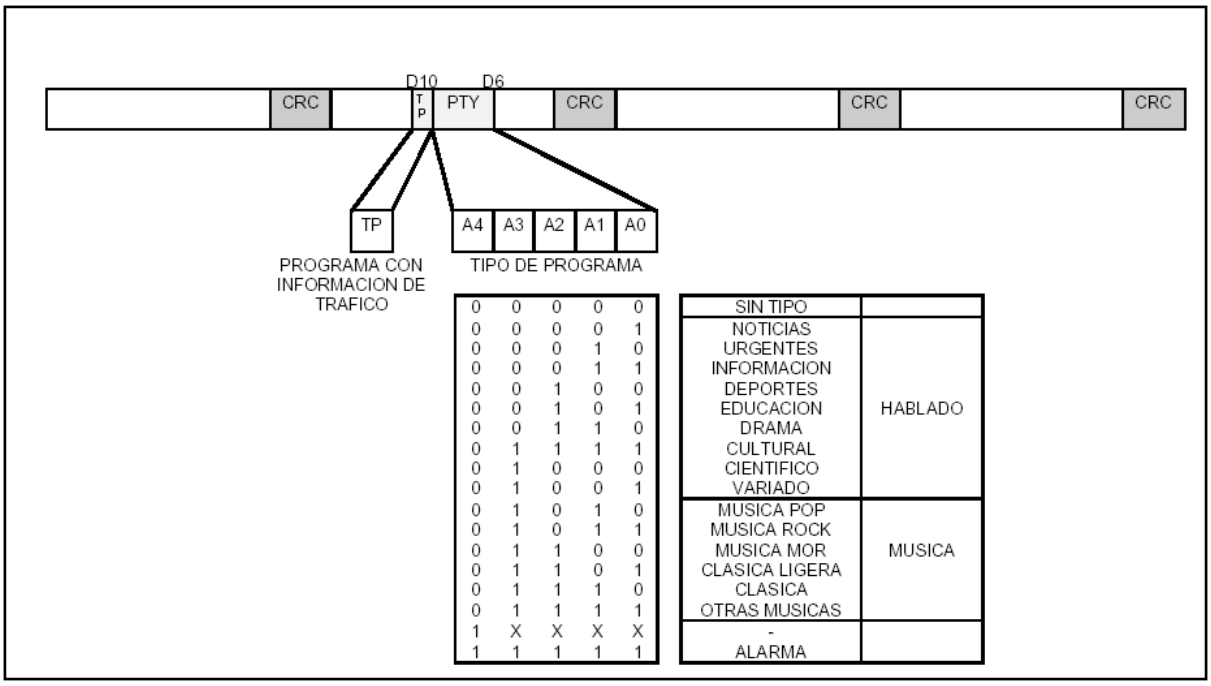
Tipos de grupos.

Además de los que definen el tipo y versión del grupo, hay otros bits fijos que deben transmitirse siempre. Entre de ellos están los 16 bits que conforman un bloque con el código PI (*Programme identification*). Este código, ocupa el primer bloque en los grupos versión A y en los grupos versión B se repite en también en el tercer bloque. El código PI contiene un código que identifica a la emisora. Los cuatro primeros bits indican el país de la emisión, los siguientes cuatro la cobertura de emisión (local, internacional, nacional, seminacional o regional), y los últimos 8 bits son asignados por un comité en cada país para identificar la emisora.



Código de identificación de la emisora PI.

También son bits fijos en los paquetes RDS un bit para señalar que se emite información de tráfico (TP: *Traffic Programme*) y 5 bits para especificar el tipo de programa que se emite (PTY: *Programme Type*). Estos bits se ubican en el segundo bloque de cada grupo.



Tipo de programa e información de tráfico.

En definitiva, en el sistema RDS cada paquete de información o grupo contiene siempre el código PI, el tipo y versión de grupo, la identificación de programa de tráfico y el tipo de programa. Ellos ocupan 27 bits en los grupos versión A y 43 bits en los grupos versión B, dejando libres 37 y 21 bits, respectivamente, para información que depende del tipo de grupo.

A continuación, veremos cómo es el grupo 4ª, que contiene la información de la fecha y la hora, que es lo fundamental en este proyecto.

2.5 GRUPO 4A: FECHA Y HORA

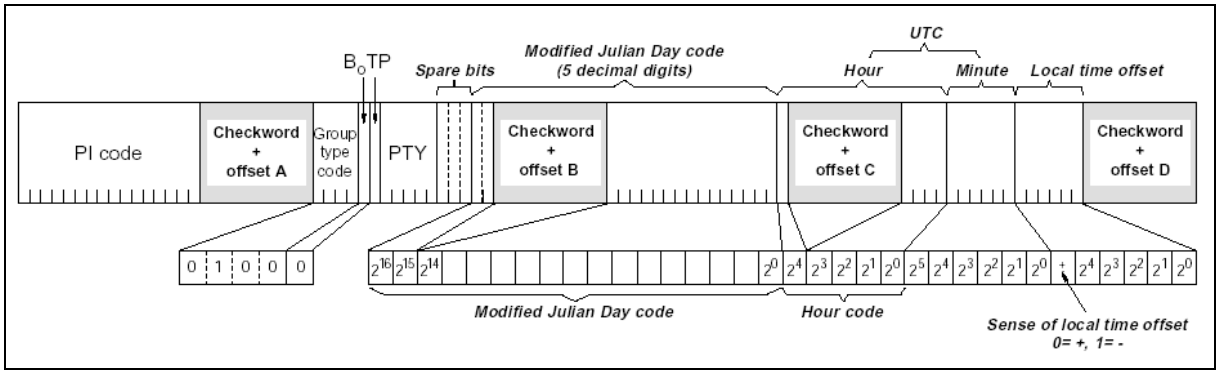
Los grupos tipo 4 solo están definidos para la versión A. En ellos se incluye la hora y la fecha. El radiodifusor toma la hora de algún patrón nacional o regional y la envía en el sistema RDS.

La hora es transmitida en el formato UTC (*Coordinated Universal Time*), en esta forma es independiente del país de emisión. Esta hora es la del meridiano de Greenwich y la hora en cualquier zona del planeta se considera desplazada un número entero de horas, que

puede variar según sea verano o invierno. Por ello, para el sistema RDS, debe enviarse también el coeficiente de corrección horaria correspondiente a la región de cobertura.

Este grupo (4A), se introduce de tal manera que el principio del siguiente minuto se produce en un margen de ± 0,1 segundos del final de la transmisión del grupo que lleva la información de la fecha y la hora. Es decir, esta información se envía cada minuto.

La fecha se transmite como día del calendario Juliano modificado, conocido como *Modified Julian Day*. Se representa por medio de 17 bits, con lo cual el sistema es capaz de operar hasta el año 2100. El equipo receptor se encarga de traducir el día Juliano al formato de día-mes-año según el país para el que se diseñe el receptor.



Transmisión de la fecha y la hora (Grupo 4A).

3. EL BUS I2C

3.1 INTRODUCCIÓN

El bus I2C fue desarrollado por Philips hace unos veinte años y ha sido ampliamente usado en todo el mundo electrónico. El bus I2C (Inter Circuito Integrado) consta de dos cables (SDA para los datos y SCL para el reloj) y es bi-direccional.

Originalmente en la especificación de bus la dirección estaba definida con siete bits, permitiendo el direccionamiento de 128 dispositivos (2^7 = 128) y una velocidad máxima de 100 Kbits/s. Pero desde la revisión de 1992, la dirección pasó a ser de 10 bits y la velocidad máxima de transferencia de 400 Kbits/s (en modo rápido), siendo compatible con la primera especificación.

En las siguientes especificaciones del bus I2C sólo hablaremos del direccionamiento de 7 bits con una velocidad de 100 Kbits/s, ya que es lo que tenemos en el presente proyecto.

3.2 INTRODUCCIÓN A LA ESPECIFICACIÓN DEL BUS I^2C.

Para aplicaciones de control digital de 8- bit, semejantes a aquellas que requieren microcontroladores, se pueden establecer ciertos criterios de diseño:

- Un sistema completo normalmente consta al menos de un microcontrolador y otros dispositivos periféricos semejantes las memorias.
- El coste de conexión de los diversos dispositivos dentro del sistema debe ser reducido al mínimo.
- Un sistema que cumple una función de control, no requiere transferencia de datos a alta velocidad
- La eficacia global depende de los dispositivos escogidos y de la naturaleza de las estructuras bus interconectadas.

Para producir un sistema que satisfaga estos criterios, son necesarias una serie de estructuras de bus. Aunque los buses serie no tengan la capacidad de "producción" de los buses paralelos, necesitan menos cables y menos pines de conexión del CI. No obstante, un bus no es simplemente un cable interconectado, este plasma todos los formatos y procedimientos para la comunicación dentro del sistema.

Los dispositivos intercomunicados mediante bus serie deben de tener algún tipo de protocolo el cual evite cualquier posibilidad de confusión, pérdida de datos y bloqueo de información. Los dispositivos rápidos deben de ser capaces de comunicarse con dispositivos lentos. Los sistemas no deben depender de los dispositivos conectados en ellos, de lo contrario las modificaciones o mejoras serían imposibles. Un procedimiento tiene que ser también ideado para decidir qué dispositivo tendrá el control del bus y cuando. Y, si dispositivos diferentes con diferentes velocidades de reloj son conectados al bus, la fuente del reloj del bus debe ser definida. Todos estos criterios están involucrados en la especificación del bus I^2C.

3.3 EL CONCEPTO DEL BUS I^2C

El bus I^2C soporta cualquier proceso de fabricación del CI (NMOS, CMOS, bipolar). Dos cables, *serial data* (SDA) y *serial clock* (SCL), llevan la información entre los dispositivos conectados al bus. Cada dispositivo es reconocido por una única dirección (si es un microcontrolador, controlador LCD, memoria o interfaz de teclado) y se pueden aplicar tanto transmisores como receptores, dependiendo de la función del dispositivo. Obviamente un controlador LCD es únicamente receptor, mientras que una memoria puede comportarse tanto como emisor o receptor. Además de transmisores y receptores, los dispositivos pueden también ser considerados como maestros o esclavos cuando desempeñan transferencia de datos (ver en tabla 1). Un maestro es un dispositivo el cuál inicia una transferencia de datos en el bus y genera las señales del reloj para permitir la transferencia. En ese momento, cualquier dispositivo direccionado es considerado un esclavo.

TÉRMINO	DESCRIPCIÓN
Emisor	Dispositivo que envía los datos por el bus.
Receptor	Dispositivo que recibe los datos por el bus.
Maestro	Dispositivo que inicia una transferencia, genera la señal de reloj y termina la transferencia
Esclavo	Dispositivo direccionado por el maestro.
Multi-maestro	Más de un maestro puede intentar controlar el bus al mismo tiempo sin corromper la información.
Arbitraje	Procedimiento que asegura que, si más de un maestro intenta controlar el bus simultáneamente, sólo uno lo consigue y el mensaje no se corrompe.
Sincronización	Procedimiento para sincronizar las señales de reloj de dos ó más dispositivos.

Tabla 1. Definición de la terminología del I^2C-bus.

El Bus I^2C es multi-maestro. Esto significa que más de un dispositivo capaz de controlar el bus puede ser conectado a este. Como los maestros son normalmente microcontroladores, vamos a considerar el caso de una transferencia de datos entre dos microcontroladores conectados a el bus I^2C (Figura 3).

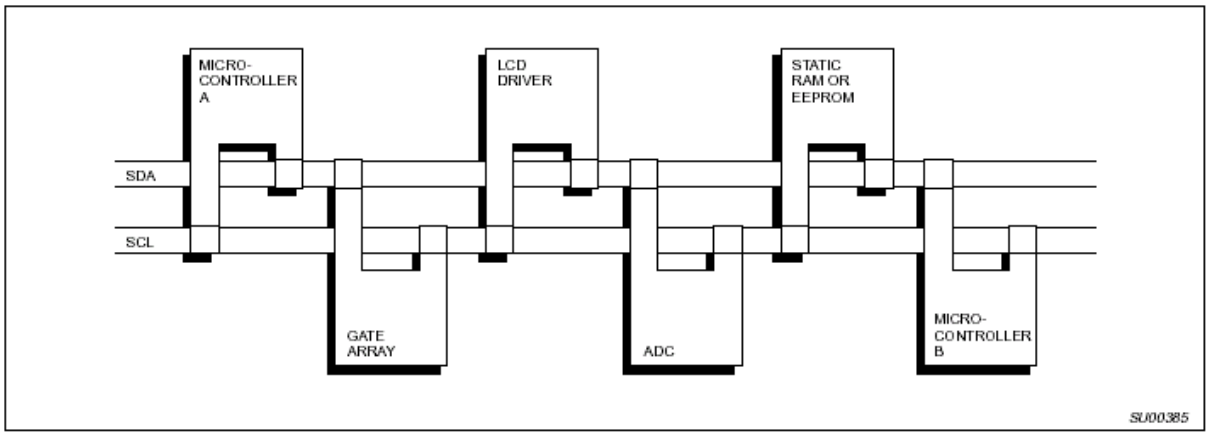


Figura 3. Ejemplo de configuración de un bus I2C con dos microcontroladores.

La transferencia de datos procedería de la siguiente manera:

1. Supongamos que el microcontrolador A quiere enviar información al µC B:
   - microcontrolador A (maestro), direcciona al microcontrolador B (esclavo)
   - microcontrolador A (maestro-transmisor), envía los datos al microcontrolador B (esclavo-receptor)
   - microcontrolador A termina la transferencia.

2. Si el microcontrolador A quiere recibir información del microcontrolador B:
   - microcontrolador A (maestro) direcciona al microcontrolador B (esclavo)
   - microcontrolador B (maestro-receptor) recibe los datos del microcontrolador B (esclavo-transmisor).
   - microcontrolador A termina la transferencia.

Aún en este caso, el maestro (microcontrolador) genera el cronometraje y termina la transferencia.

La posibilidad de conectar más de un microcontrolador al bus I^2C significa que más de un maestro podría tratar de iniciar la transferencia de datos al mismo tiempo. Para evitar el caos que todo esto podría causar, se ha desarrollado un procedimiento de arbitraje. Este procedimiento consiste en una conexión AND de todos los interfaces I^2C con el bus.

Si dos o más maestros tratan de introducir información en el bus, el primero que produzca un "uno" cuando el otro produce un "cero" perderá el arbitraje. Las señales del reloj durante el arbitraje son una combinación sincronizada de los relojes generada por los maestros usando una conexión AND a la línea SCL.

La generación de las señales del reloj del bus I^2C es siempre responsabilidad de los dispositivos del maestro; cada maestro genera sus propias señales de reloj cuando está transfiriendo datos al bus. Las señales de reloj del bus generadas por un maestro pueden ser únicamente alteradas cuando son estiradas por un dispositivo esclavo lento, o por otro maestro cuando suceda un arbitraje.

3.4 CARACTERISTICAS GENERALES.

Tanto SDA como SCL son líneas direccionales conectadas a un suministro de tensión positiva mediante una resistencia de pull-up (ver figura 4). Cuando el bus está libre, ambas líneas están a nivel alto (HIGH). Las etapas de salida de los dispositivos conectados al bus deben ser de tipo colector abierto o drenador abierto para poder usar la conexión AND. Los

datos en el bus I2C pueden ser transferidos a una velocidad máxima de 100 kbit/s en el modo standard, o de 400 kbit/s en el modo rápido. El número de dispositivos conectados al bus depende únicamente del límite de capacidad del bus de 400 pF.

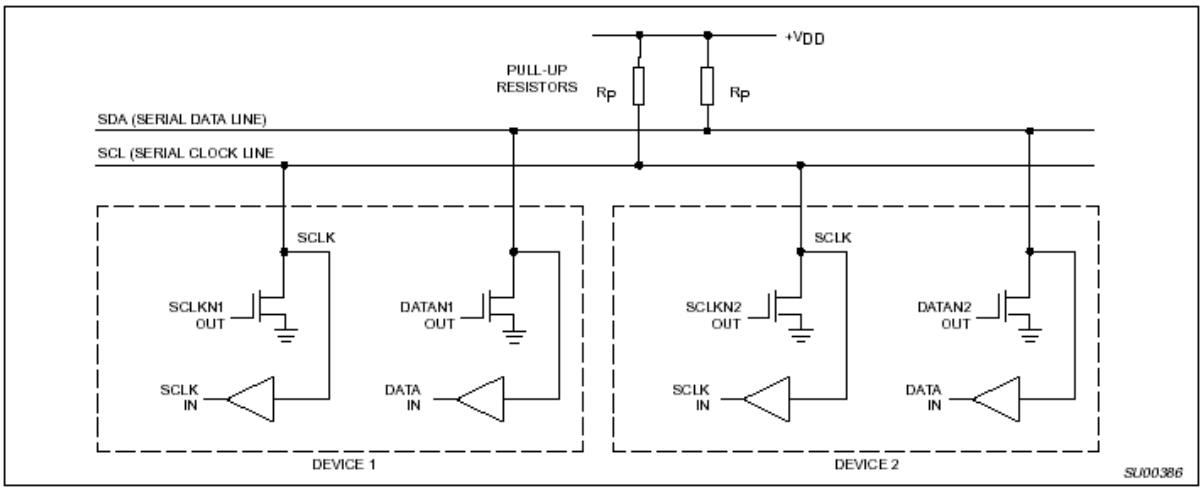


Figura 4. Conexión de los dispositivos al bus I2C.

3.5 TRANSFERENCIA DE BITS

Debido a la variedad de diferentes tecnologías (CMOS, NMOS, bipolar) que pueden ser conectadas al bus I2C, los niveles del '0' lógico (LOW) y el '1' lógico (HIGH) no están fijadas y dependen del nivel asociado de Vdd. Por cada bit de datos transferido se genera un pulso de reloj.

3.5.1 Validez de datos.

Los datos de la línea SDA deben ser estables durante el periodo alto (HIGH) del reloj. El estado alto o bajo de la línea de datos solo puede ser cambiado cuando la señal de reloj de la línea SCL es bajo (ver figura 5).

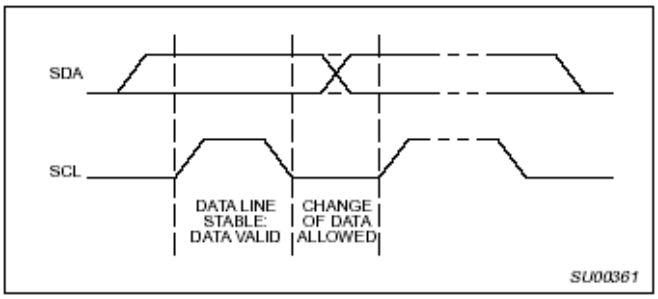


Figura 5. Transferencia de bits en el bus I2C.

3.5.2 Condiciones de START y STOP.

En el bus I2C las únicas situaciones que deben ser definidas son las condiciones de START y STOP.

Una transición de nivel alto a bajo en la línea SDA mientras la línea SCL está en nivel alto indica la condición de START.

Una transición de nivel bajo a alto en la línea SDA mientras la línea SCL está en nivel alto indica la condición de STOP.

Las condiciones de START y STOP son generadas siempre por el maestro. El bus es considerado como "ocupado" después de la condición de START. El bus es considerado nuevamente libre después de que pase un cierto tiempo de la condición de STOP.

La detección de las condiciones de START y STOP por los dispositivos conectados al bus es fácil si estos incorporan el hardware necesario de interfaz. No obstante, los microcontroladores que no tengan esta interfaz tienen que muestrear la línea SDA al menos dos veces por periodo de reloj para detectar la transición.

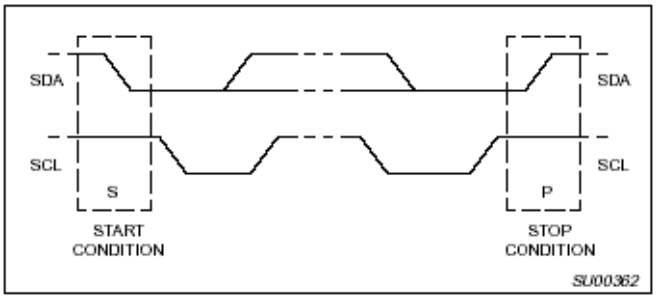


Figura 6. Condiciones de START y STOP.

3.6 TRANSFIRIENDO DATOS.

3.6.1 Formato de Byte.

Cada uno de los bytes puestos en la línea SDA deben tener una longitud de 8 bits. El número de bytes que pueden ser transmitidos en cada transferencia no tiene restricciones. Cada byte debe estar seguido por un bit de "acknowledge". Los datos son transmitidos empezando por el bit más significativo (MSB). Si un receptor no puede recibir otro byte de datos completo hasta que haya realizado alguna otra función (por ejemplo la atención a una interrupción interna) puede mantener la línea SCL en nivel bajo para forzar al transmisor a un estado de espera. La transferencia de datos continuará cuando el receptor esté listo para otro byte de datos y libere la línea SCL.

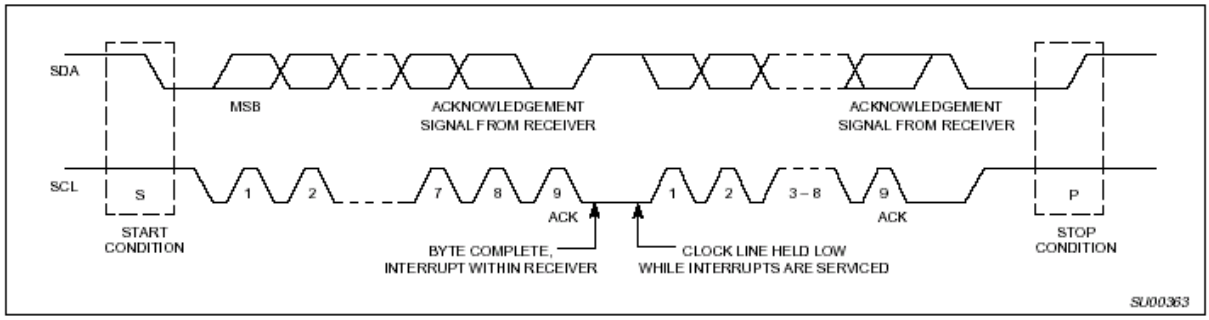


Figura 7. Transferencia de datos en el bus I2C.

3.6.2 Señal ACK (Acknowledge)

Transferir datos con ACK es obligatorio. El pulso de reloj asociado a ACK es generado por el maestro. El emisor libera la línea SDA (HIGH) durante el pulso de reloj asociado a ACK.

El receptor debe poner en bajo la línea SDA durante el pulso de reloj de ACK de manera que tengamos un nivel bajo estable en SDA durante el nivel alto del pulso de reloj (figura 8). Por supuesto los tiempos de "hold" y "set-up" deben ser tenidos en cuenta.

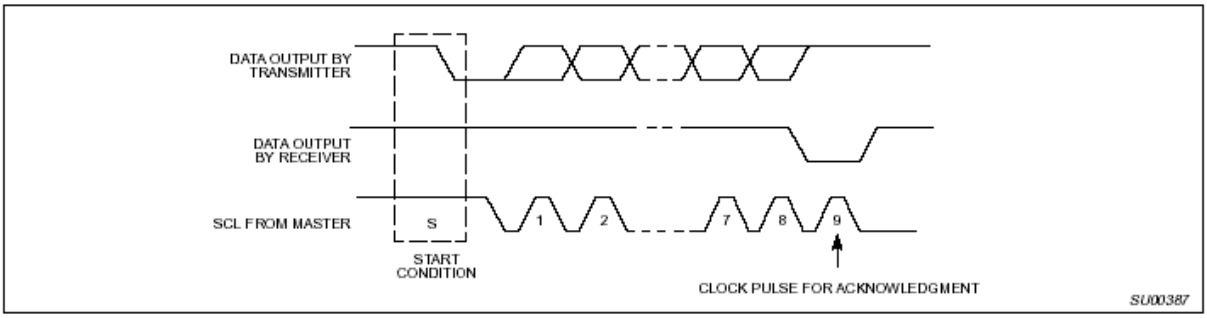


Figura 8. Acknowledge en el bus I2C.

Generalmente un receptor que ha sido direccionado está obligado a generar una señal ACK después de que se ha recibido cada byte, excepto cuando el mensaje comienza con una dirección CBUS (dispositivos compatibles).

Si un esclavo-receptor genera la señal ACK pero un tiempo después no se recibe ningún byte de datos más, el maestro debe abortar la transferencia. Esto lo indica el esclavo generando la señal no ACK en el primer byte siguiente. El esclavo deja la línea de datos en nivel alto y el maestro genera la condición de STOP.

Si un maestro-receptor está implicado en una transferencia debe indicar el final de los datos al esclavo-transmisor mediante la no generación de la señal ACK en el último byte que fue enviado por el esclavo. El esclavo-transmisor debe liberar la línea de datos para permitir al maestro generar una condición de STOP o nuevamente de START.

3.7 ARBITRAJE Y GENERACIÓN DE RELOJ.

3.7.1 Sincronización

Todos los maestros generan su propio reloj en la línea SCL para transmitir mensajes por el bus I2C. Los datos solo son válidos durante el periodo de nivel alto del reloj. Un reloj definido es por lo tanto necesario para el procedimiento de arbitraje bit a bit.

La sincronización de reloj es realizada usando la conexión AND de los interfaces I2C con la línea SCL. Esto quiere decir que la línea SCL tendrá un nivel bajo cuando alguno de

los relojes de los maestros presentes en la línea estén a nivel bajo y tendrá un nivel alto cuando todos los relojes de los maestros presentes en la línea estén a nivel alto.

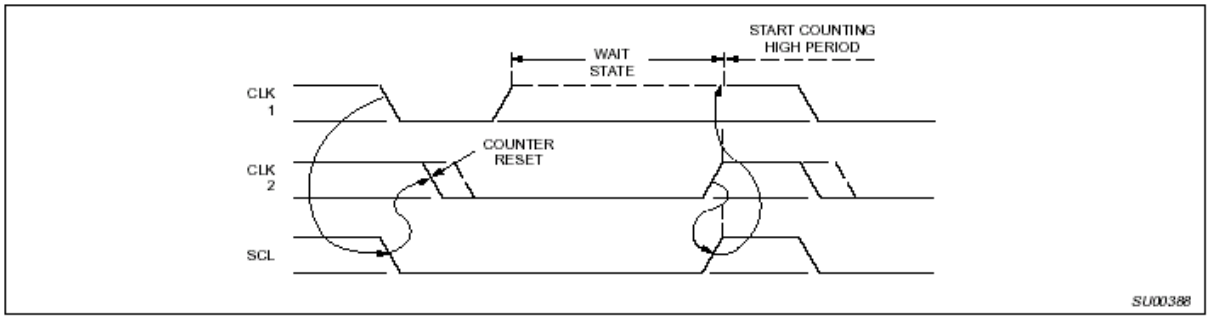


Figura 9. Sincronización del reloj durante el procedimiento de arbitraje.

De esta manera un reloj SCL sincronizado se genera con su periodo de nivel bajo determinado por el dispositivo con el periodo en nivel bajo más largo, y su periodo a nivel alto está determinado por aquel que tenga el periodo en nivel alto más corto.

3.7.2 Arbitraje

Un maestro puede comenzar una transferencia solamente si el bus está libre. Dos o mas maestros pueden generar una condición de START con el mínimo tiempo de "hold" de la condición de START lo que define una condición de START en el bus.

El arbitraje se efectúa en la línea SDA mientras la línea SCL está en nivel alto, de tal manera que el maestro que transmita un nivel alto, mientras que otro maestro transmite un nivel bajo, desconectará su etapa de salida de datos porque el nivel en el bus no se corresponde con su propio nivel.

El arbitraje puede continuar durante muchos bits. Su primera etapa es la comparación de los bits de dirección. Si ambos maestros están intentando direccionar el mismo dispositivo el arbitraje continua con la comparación de los datos. Debido a que la dirección y la información de datos del bus I2C son usadas para el arbitraje, no se pierde información durante este proceso.

Un maestro que pierde el arbitraje puede generar pulsos de reloj hasta el final del byte en el cual pierde el arbitraje.

Si un maestro también incorpora la función esclavo y pierde el arbitraje durante la etapa de direccionamiento es posible que el maestro que gana el arbitraje esté intentando direccionar a dicho dispositivo. El maestro "perdedor" debe entonces cambiar inmediatamente al modo esclavo-receptor.

La figura 10 muestra el procedimiento de arbitraje para dos maestros. Por supuesto pueden estar involucrados mas maestros, dependiendo del número de dispositivos maestros que estén conectados al bus.

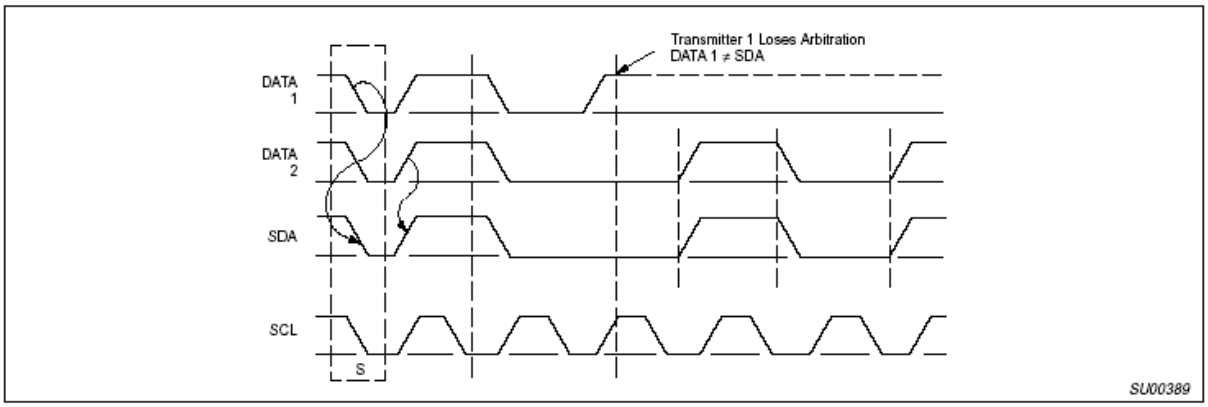


Figura 10. Procedimiento de arbitraje de dos maestros.

Debido a que el control del bus I2C se decide únicamente en el envío de la dirección y los datos desde los maestros, no hay un maestro "principal" ni ninguna otro orden de prioridad para acceder al bus.

Debemos poner especial atención si, durante una transferencia serie, el procedimiento de arbitraje está aún en progreso en el momento en el que se transmite una condición de START repetida ó una condición de STOP por el bus I2C. Es decir, el arbitraje no se permite entre:

- Una condición de START repetida y un bit de datos.
- Una condición de STOP y un bit de datos.
- Una condición de START repetida y una condición de STOP.

3.8 FORMATOS DE DIRECCIÓN CON 7 BITS.

La transferencia de datos sigue el formato mostrado en la figura 11. Después de la condición de START (S), se envía la dirección del esclavo al que nos dirigimos. Esta dirección es de 7 bits, seguida de un octavo bit que indica el sentido de los datos por el bus (lectura ó escritura). Un '0' indica una transmisión (escritura); y un '1' indica la solicitud de datos (lectura). Una transferencia de datos termina siempre con una condición de STOP (P) generada por el maestro. No obstante, si un maestro aún quiere seguir comunicándose por el bus, puede generar una condición de START repetida (Sr) y direccionar otro esclavo sin generar primero la condición de STOP. Son posibles distintas combinaciones de lectura/escritura.

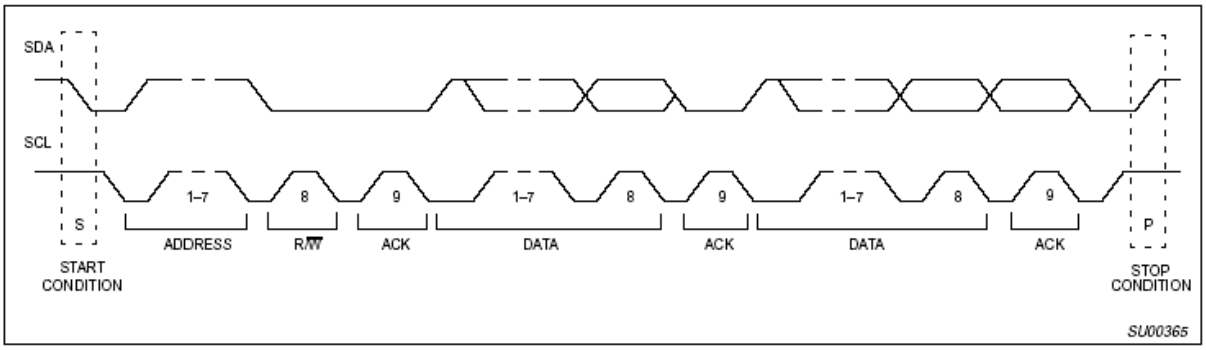


Figura 11. Una transferencia de datos completa.

Los posibles formatos de transferencia son los siguientes:

- El maestro-emisor transmite datos al esclavo-receptor. El sentido de la transferencia no cambia. (Figura 12).
- El maestro lee lo que envía el esclavo inmediatamente después del primer byte (Figura 13). En el momento del primer ACK, el maestro-emisor se convierte en maestro-receptor y el esclavo-receptor se convierte en esclavo-emisor. Este primer ACK es generado aún por el esclavo. La condición de STOP es generad por el maestro.
- Forma combinada (Figura 14). Durante un cambio de sentido de la transferencia, la condición de START y la dirección del esclavo son repetidas, pero con el bit de

lectura/escritura contrario. Si un maestro-receptor envía una condición de START repetida, previamente ha enviado un NO ACK.

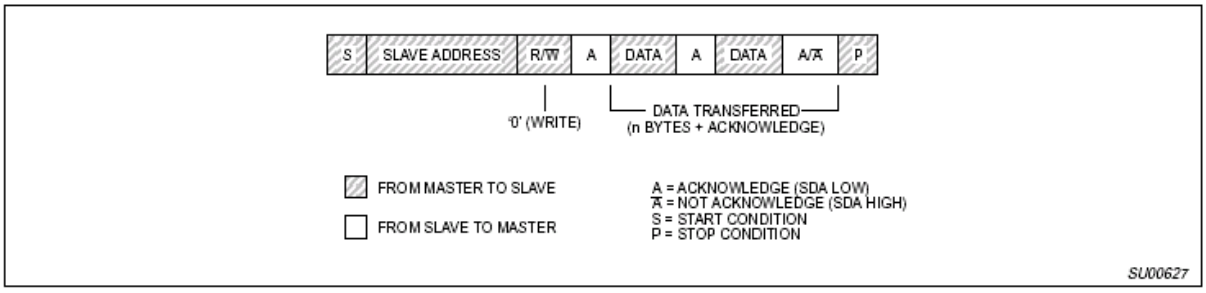


Figura 12. Un maestro-emisor direcciona a un esclavo con direccionamiento de 7 bits.

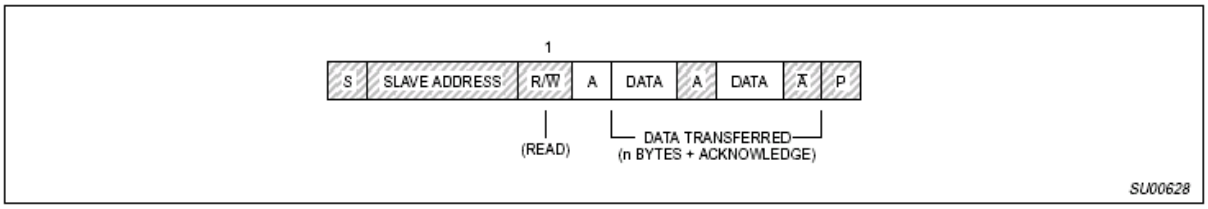


Figura 13. Un maestro lee lo enviado por el esclavo inmediatamente después del primer byte.

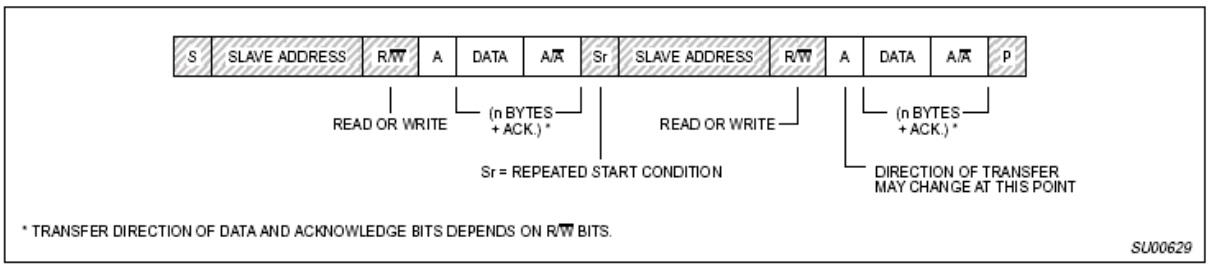


Figura 14. Forma combinada.

3.9 DIRECCIONAMIENTO CON 7 BITS

El direccionamiento en el bus I2C es tal que, después de la condición de START, el primer byte determina qué esclavo es seleccionado por el maestro. La única excepción es la "llamada general", que puede direccionar cualquiera de los dispositivos esclavos. Cuando se usa esta dirección (la llamada general) todos los dispositivos esclavos deberían, en teoría, responder con un ACK. No obstante, los dispositivos pueden estar diseñados para ignorar esa llamada. El segundo byte de la llamada general determina la acción que deben realizar los dispositivos esclavos.

3.9.1 Definición de los bits del primer byte.

Los primeros siete bits del primer byte componen la dirección del esclavo (figura 15). El octavo bit es el LSB (bit menos significativo), el cual determina el sentido de la transmisión. Un '0' en el octavo bit del primer byte, significa que el maestro va a escribir información en el dispositivo esclavo. Un '1' en esta misma posición, significa que el maestro va a leer información del dispositivo esclavo.

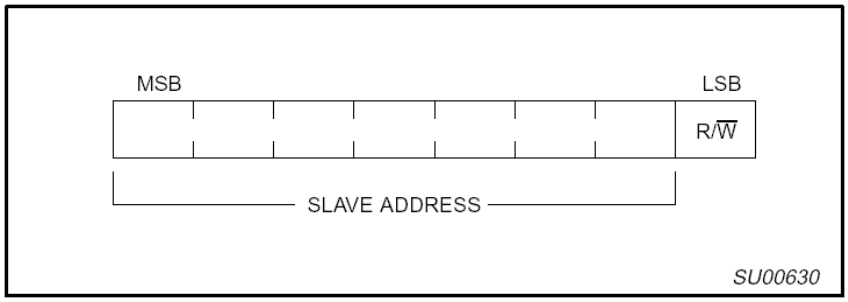


Figura 15. Primer byte tras la condición de START.

Cuando se envía una dirección, cada dispositivo del sistema compara los 7 primeros bits recibidos después de la condición de START, con su propia dirección. Si coinciden, el esclavo se considera direccionado por el maestro como un esclavo-emisor ó un esclavo-receptor, dependiendo del bit de lectura/escritura.

La dirección de un esclavo puede estar formada por una parte fija y otra programable. Debido a que puede haber varios dispositivos idénticos en un sistema, la parte programable de la dirección del esclavo determina el máximo número de dispositivos idénticos conectados a un bus I2C. El número de bits programables en la dirección de un esclavo depende principalmente del número de pines disponibles en él. Por ejemplo, un dispositivo con 4 bits fijos y 3 programables en su dirección, implica que se pueden conectar 8 dispositivos idénticos en un mismo bus I2C.

El comité del bus I2C coordina la asignación de direcciones I2C, y determina que hay dos grupos de ocho direcciones (0000XXX y 1111XXX) reservadas para los propósitos mostrados en la tabla 2.

DIRECCIÓN	BIT R/W	DESCRIPCIÓN
0000 000	0	Llamada general.
0000 000	1	Byte de START
0000 001	X	Dirección CBUS (Dispositivos compatibles).
0000 010	X	Dirección reservada para diferentes formatos de bus.
0000 011	X	Reservadas para futuros propósitos.
0000 1XX	X	
1111 1XX	X	
1111 0XX	X	Direccionamiento de 10 bits.

Tabla 2. Definición de las direcciones reservadas en I2C.

3.10 CARACTERÍSTICAS ELÉCTRICAS PARA DISPOSITIVOS I2C

Las especificaciones eléctricas para las estradas y salidas de los dispositivos I2C y las características de las líneas que se conectan a ellos, están reflejadas en las tablas 3 y 4 que se muestran a continuación.

PARAMETER	SYMBOL	STANDARD-MODE DEVICES		FAST-MODE DEVICES		UNIT
		Min.	Max.	Min.	Max.	
LOW level input voltage:	V_{IL}					V
fixed input levels		–0.5	1.5	–0.5	1.5	
V_{DD}-related input levels		–0.5	$0.3V_{DD}$	–0.5	$0.3V_{DD}$	
HIGH level input voltage:	V_{IH}					V
fixed input levels		3.0	*1)	3.0	*1)	
V_{DD}-related input levels		$0.7V_{DD}$	*1)	$0.7V_{DD}$	*1)	
Hysteresis of Schmitt trigger inputs:	V_{hys}					V
fixed input levels		n/a	n/a	0.2	–	
V_{DD}-related input levels		n/a	n/a	$0.05V_{DD}$	–	
Pulse width of spikes which must be suppressed by the input filter	t_{SP}	n/a	n/a	0	50	ns
LOW level output voltage (open drain or open collector):						V
at 3 mA sink current	V_{OL1}	0	0.4	0	0.4	
at 6 mA sink current	V_{OL2}	n/a	n/a	0	0.6	
Output fall time from V_{IHmin} to V_{ILmax} with a bus capacitance from 10 pF to 400 pF:	t_{of}					ns
with up to 3 mA sink current at V_{OL1}		–	250[3)]	$20 + 0.1C_b$[2)]	250	
with up to 6 mA sink current at V_{OL2}		n/a	n/a	$20 + 0.1C_b$[2)]	250[3)]	
Input current each I/O pin with an input voltage between 0.4 V and $0.9V_{DDmax}$	I_i	–10	10	–10[4)]	10[4)]	μA
Capacitance for each I/O pin	C_i	–	10	–	10	pF

n/a = not applicable
1. Maximum $V_{IH} = V_{DDmax} + 0.5$ V
2. C_b = capacitance of one bus line in pF.
3. The maximum t_f for the SDA and SCL bus lines quoted in Table 4 (300 ns) is longer than the specified maximum t_{of} for the output stages (250 ns). This allows series protection resistors (R_s)to be connected between the SDA/SCL pins and the SDA/SCL bus lines as shown in Figure 37 without exceeding the maximum specified t_f.
4. I/O pins of fast-mode devices must not obstruct the SDA and SCL lines if V_{DD} is switched off.

Tabla3. Características de las etapas de entrada/salida SDA y SCL para dispositivos I2C.

PARAMETER	SYMBOL	STANDARD-MODE I²C-BUS		FAST-MODE I²C-BUS		UNIT
		Min.	Max.	Min.	Max.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	μs
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	μs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	μs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	μs
Set–up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	μs
Data hold time:	$t_{HD;DAT}$					
for CBUS compatible masters (see NOTE, Section 9.1.3)		5.0	–	–	–	μs
for I²C–bus devices		0[1)]	–	0[1)]	0.9[2)]	μs
Data set–up time	$t_{SU;DAT}$	250	–	100[3)]	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	$20 + 0.1C_b$[4)]	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	$20 + 0.1C_b$[4)]	300	ns
Set–up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	μs
Capacitive load for each bus line	C_b	–	400	–	400	pF

All values referred to V_{IHmin} and V_{ILmax} levels (see Table 3).
1. A device must internally provide a hold time of at least 300 ns for the SDA signal (referred to the V_{IHmin} of the SCL signal) in order to bridge the undefined region of the falling edge of SCL.
2. The maximum $t_{HD;DAT}$ has only to be met if the device does not stretch the LOW period (t_{LOW}) of the SCL signal.
3. A fast-mode I²C-bus device can be used in a standard-mode I²C-bus system, but the requirement $t_{SU;DAT} \geq 250$ ns must then be met. This will automatically be the case if the device does not stretch the LOW period of the SCL signal. If such a device does stretch the LOW period of the SCL signal, it must output the next data bit to the SDA line $t_{r\,max} + t_{SU;DAT} = 1000 + 250 = 1250$ ns (according to the standard-mode I²C-bus specification) before the SCL line is released.
4. C_b = total capacitance of one bus line in pF.

Tabla 4. Características de las líneas SDA y SCL para los dispositivos I2C.

Los dispositivos I2C con niveles fijos de entrada de 1,5 V y 3 V, pueden cada uno tener su propia tensión de alimentación. Las resistencias de pull-up deben estar conectadas a una tensión de alimentación de 5V ± 10% (figura 21).

Cuando mezclamos dispositivos con tensiones de entrada fijas y dispositivos con niveles de entrada relativos a Vdd, los últimos deben estar conectados a una tensión común de alimentación de 5V ± 10% y deben tener resistencias de pull-up conectadas a sus pines SDA y SCL, como se ve en la figura 23.

Los niveles de entrada están definidos de manera que:

- El margen de ruido del nivel bajo (LOW) es de 0.1 Vdd.
- El margen de ruido del nivel alto (HIGH) es de 0.2 Vdd.
- Como vemos en la figura 24, las resistencias en serie Rs pueden ser usadas como protecciones frente a picos de alta tensión en las líneas SDA y SCL.

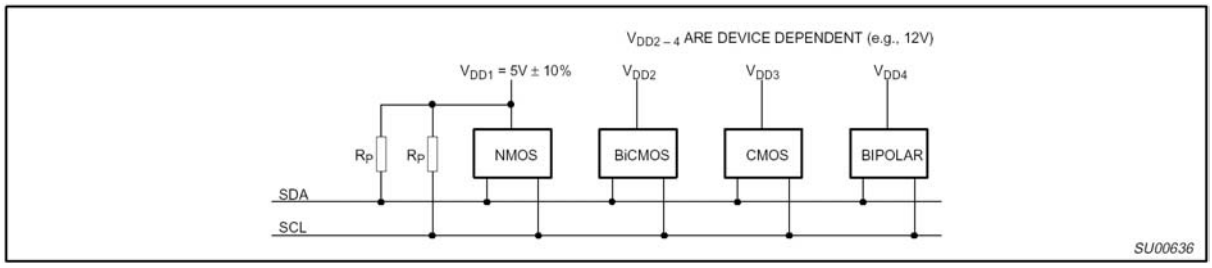


Figura 21. Dispositivos con nivel fijo de entrada conectados al bus I2C.

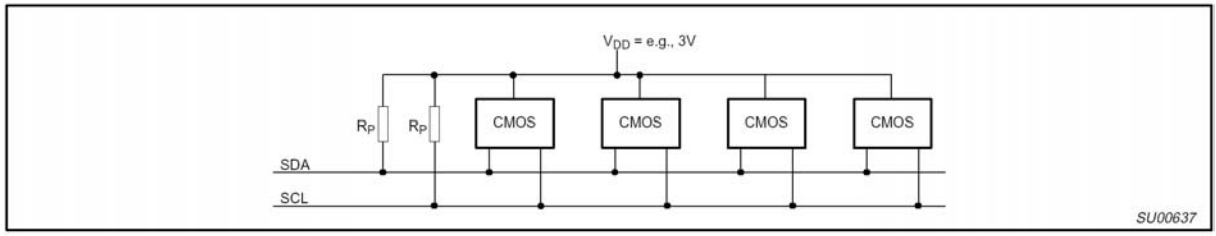


Figura 22. Dispositivos con un amplio rango de tensión de alimentación conectados al bus I2C.

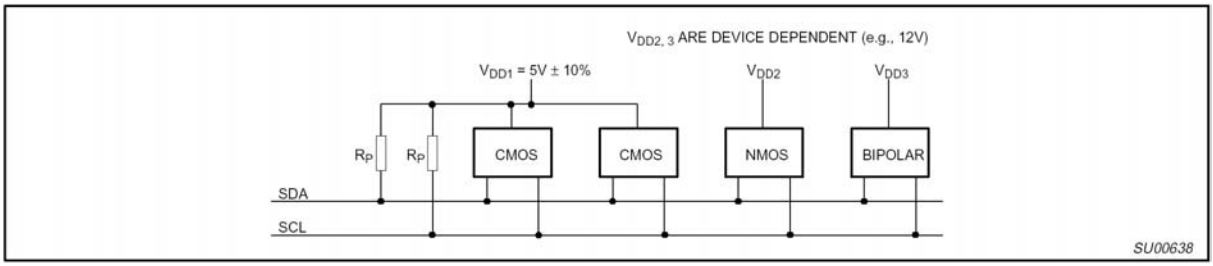


Figura 23. Dispositivos con niveles de entrada relativos a Vdd (Vdd1), mezclados con dispositivos de nivel fijo de entrada (Vdd2,3) en el bus I2C.

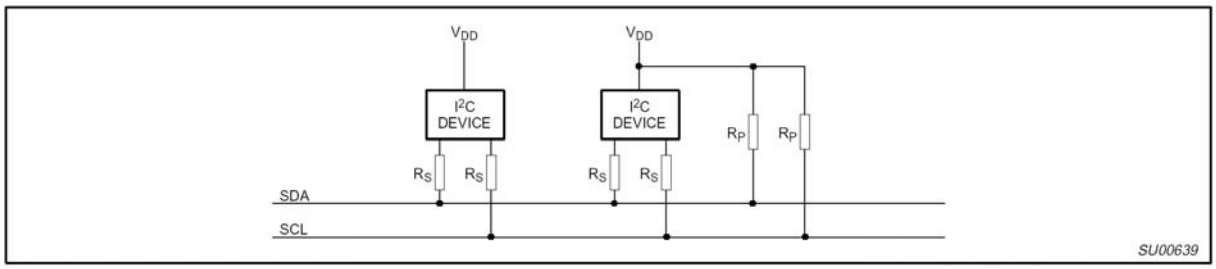


Figura 24. Resistencias en serie (Rs) para protección contra picos de alta tensión.

3.10.1 Valores máximos y mínimos de las resistencias Rp y Rs.

Para el modo estándar de los dispositivos I2C, los valores de las resistencias Rp y Rs de la figura 24, dependen de los siguientes parámetros:

- Tensión de alimentación.
- Capacidad en el bus.
- Número de dispositivos conectados (corriente de entrada + corriente de fugas).

La tensión de alimentación limita el mínimo valor de la resistencia Rp, debido a la corriente de "sumidero" (sink) de 3 mA a V_{OLmax} = 0.4 V, especificada para las etapas de salida. La figura 25 muestra el valor de Vdd en función del valor de la resistencia Rp_{min} . El margen de ruido deseable de 0.1 Vdd para el nivel bajo (LOW), limita el máximo valor de Rs. La figura 26 muestra el valor de Rs_{max} en función del valor de la resistencia Rp.

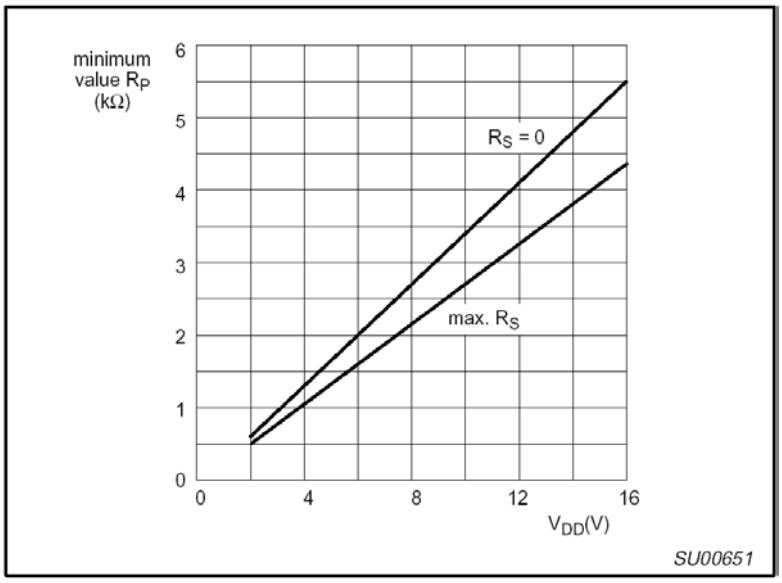


Figura 25. Valor mínimo de Rp en función de la tensión de alimentación, con Rs como parámetro.

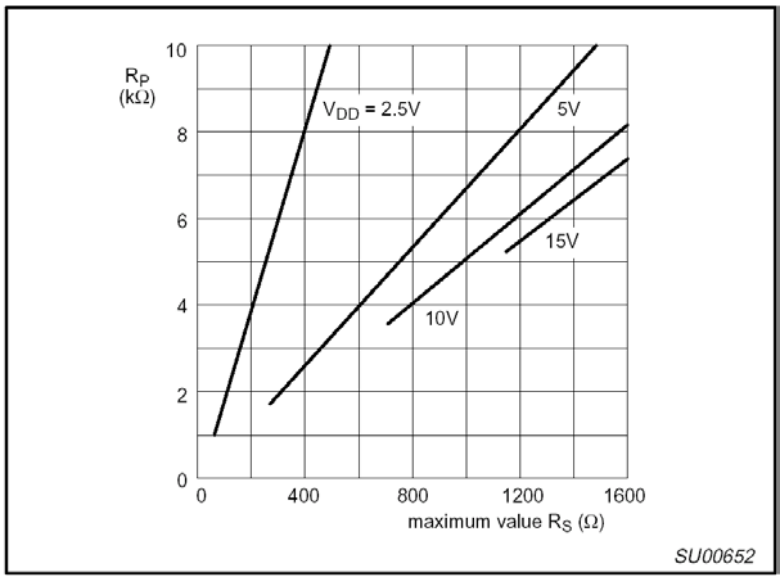


Figura 26. Valor máximo de Rs en función de Rp, con la tensión de alimentación como parámetro.

La capacidad del bus es la capacidad total del cable, conexiones y pines. Esta capacidad limita el valor máximo de Rp, debido al tiempo de subida especificado. La figura 27 muestra Rp_{max} en función de la capacidad del bus.

La máxima corriente de entrada del nivel alto (HIGH) de cada conexión de entrada/salida, tiene un valor máximo especificado de 10 μA, Debido a margen de ruido deseado de 0.2 Vdd para el nivel alto, esta corriente de entrada limita el valor máximo de Rp. Este límite depende de Vdd. La figura 28 muestra el valor de la corriente total de entrada del nivel alto (HIGH) en función de Rp_{max}.

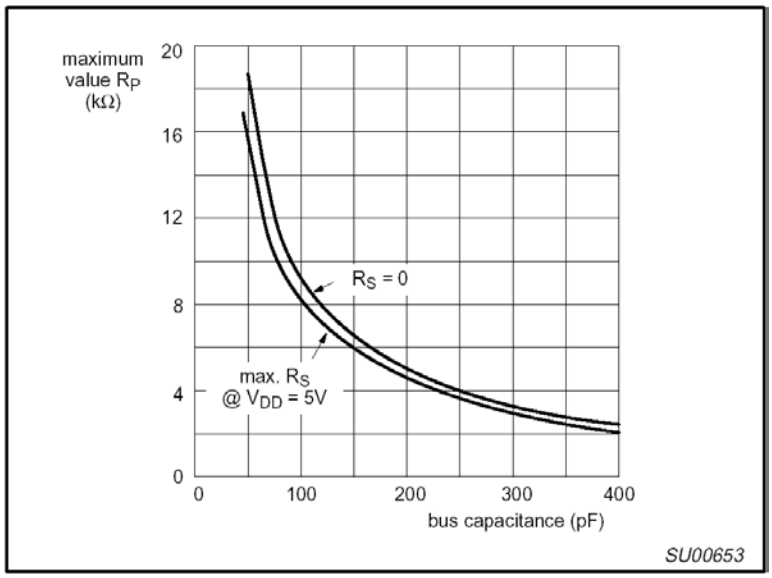


Figura 27. Valor máximo de Rp en función de la capacidad del bus para un bus I2C estándar.

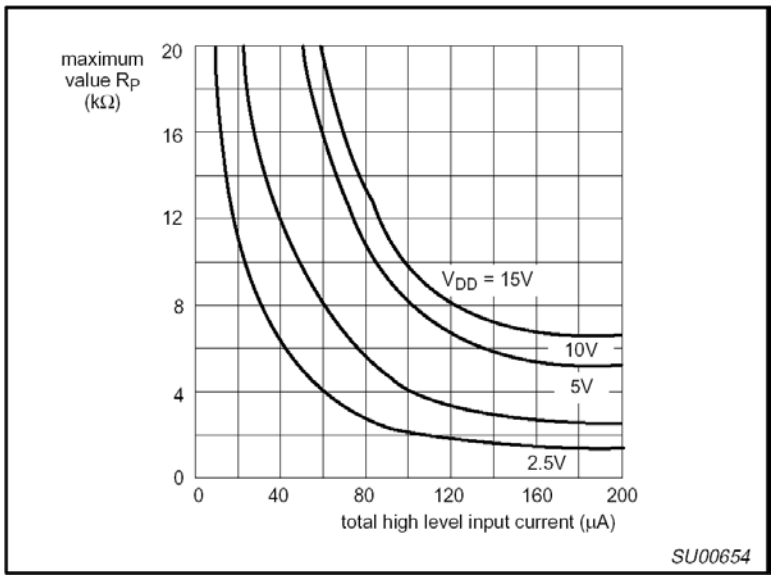


Figura 28. Corriente de entrada total del nivel alto (HIGH) en función del valor máximo de Rp con la tensión de alimentación como parámetro.

4. ESTUDIO DE LOS DISPOSITIVOS DEL MERCADO

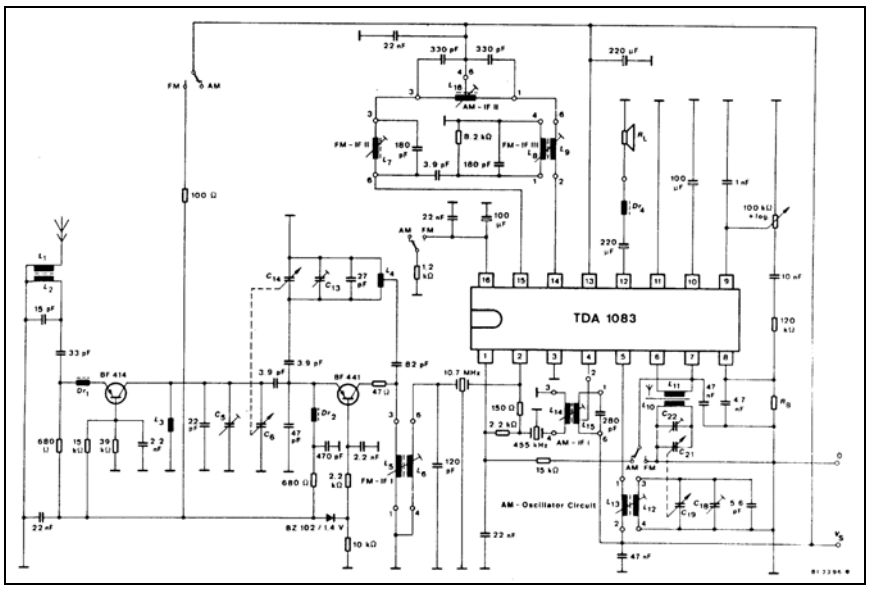
Tanto para lo que será el circuito de recepción de radio FM como para el circuito de decodificación y procesado de los datos RDS, existen en el mercado diferentes circuitos integrados a partir de los cuales podemos conseguir la realización de este proyecto. Por este motivo, se ha llevado a cabo un estudio de las distintas opciones ofrecidas por los diferentes fabricantes y se ha realizado una selección en función, principalmente, de su calidad, precio, complejidad del montaje y disponibilidad.

A continuación veremos algunas de las opciones barajadas como posibles, a lo largo del proceso de búsqueda y selección de componentes que integran el proyecto. Las distintas opciones se encuentran ordenadas por fabricante en orden alfabético, incluyendo el motivo de su descarte.

4.1 CIRCUITOS INTEGRADOS DE RADIO FM

4.1.1 Atmel TDA 1083.

Este circuito integrado es un circuito de radio de los denominados *Radio on a chip,* que dispone de todas las etapas de un receptor de radio AM/FM, excepto de la etapa de sintonización.



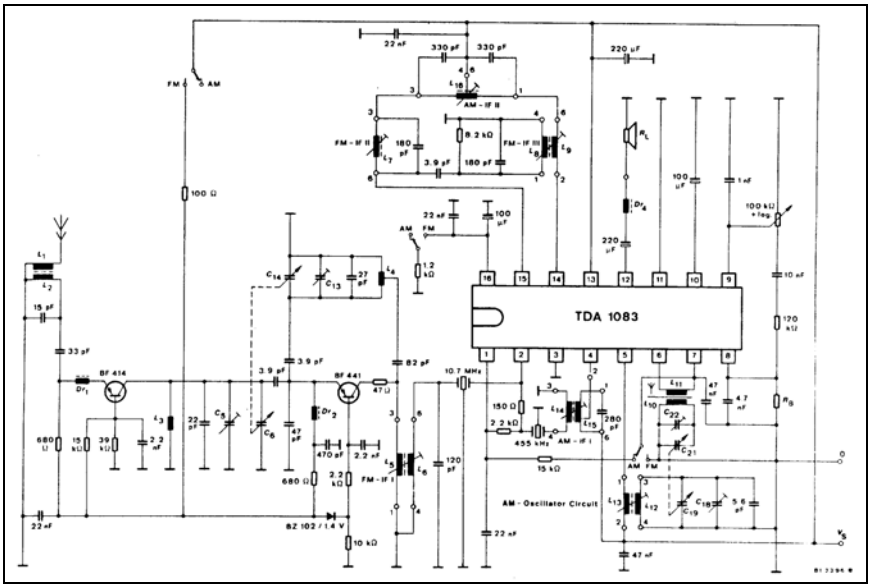
Montaje del circuito ATMEL TDA 1083.

El principal problema de este circuito radica, como se puede ver en la figura anterior, en la gran complejidad del montaje requerido para su funcionamiento, esto es, en el gran número de componentes pasivos externos que se necesitan.

Esto, sumado a la dificultad de conseguir a través de los principales distribuidores (RS Amidata, Farnell, etc), hace que no sea un buen candidato a tener en cuenta, al menos, en este proyecto.

4.1.2 Panasonic AN7293.

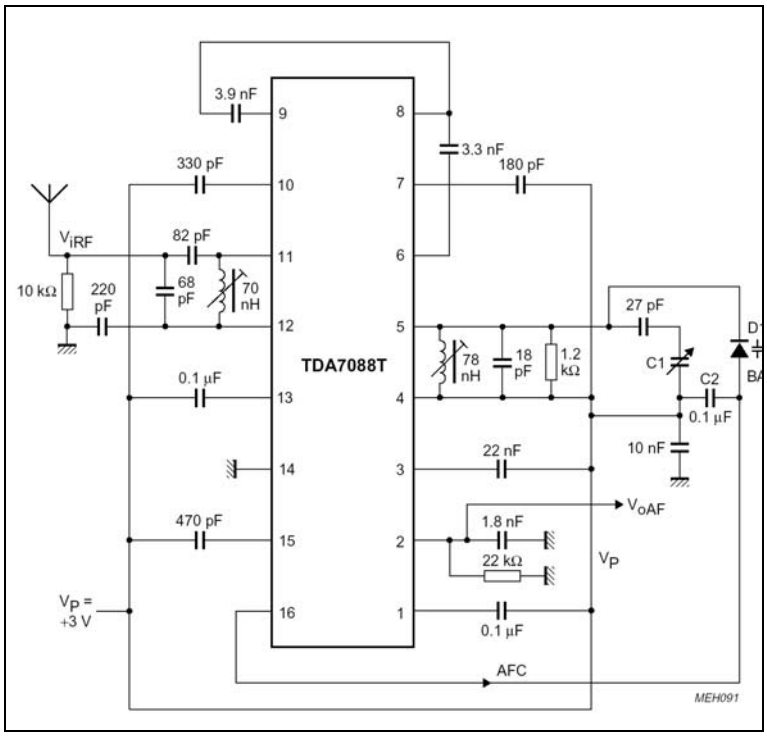
Este es otro de los circuitos barajados como una de las opciones. Este circuito nos ofrece una mayor calidad de recepción que el anterior y no requiere gran número de componentes externos. Sin embrago, al igual que en el caso anterior, carece de circuito de sintonización y necesita que se le proporcione directamente la señal a la frecuencia intermedia; y lo que es peor, necesita un cristal de cuarzo de una frecuencia poco común (912 kHz). Esto, sumado a la dificultad para conseguirlo, hace que sea descartado.



Montaje del PANASONIC AN7293.

4.1.3 Philips TDA7088T.

Este era sin duda el candidato ideal para la realización de este proyecto. Requiere pocos componentes externos, dispone de un circuito automático de búsqueda de emisoras, es relativamente fácil de conseguir (en distribuidores internacionales tipo ARROW y AVNET) y tiene un precio bastante aceptable (en torno a 1,5 €). Sin embargo, existen tres razones de peso para descartar a este candidato. En primer lugar, no es fácil conseguir muestras para la realización del prototipo y si queremos comprarlo, debemos hacer un pedido superior a 1000 unidades (AVNET). En segundo lugar, requiere un diodo varicap (BB910) para su sintonización, que no es fácil de obtener. Y finalmente, es un circuito de montaje superficial y 16 pines, lo que hace imposible que podamos incluirlo en nuestro montaje debido a que no disponemos de la tecnología necesaria para realizar prototipos en montaje superficial (SMD).

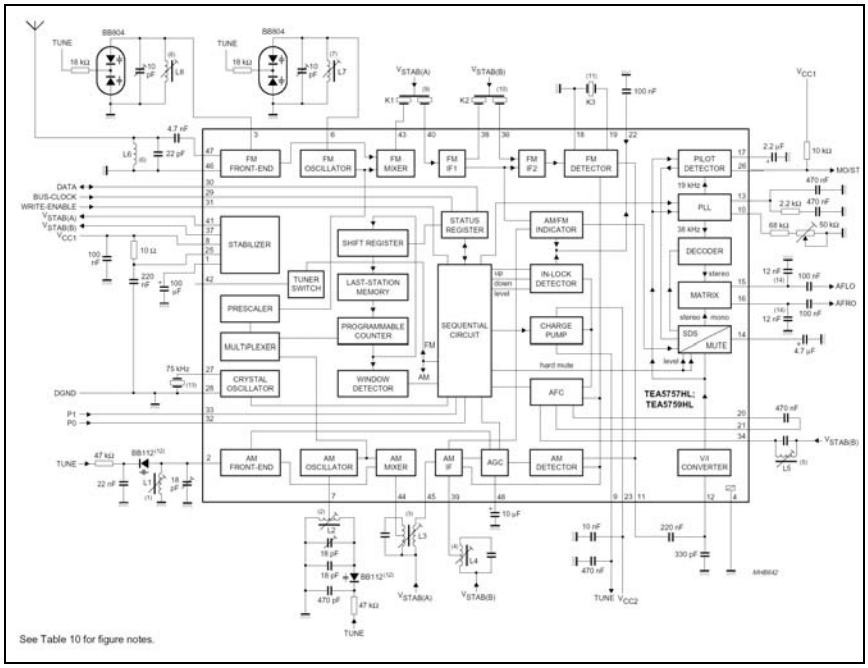


Montaje del TDA 7088T.

4.1.4 Philips TEA 5757HL.

Este circuito era otro de los grandes candidatos a ser incluidos en el presente proyecto. Se trata de un circuito de radio FM/AM de sintonización automática y elevada calidad, controlado vía bus serie de 2 líneas (muy parecido al I2C) desde un microcontrolador,

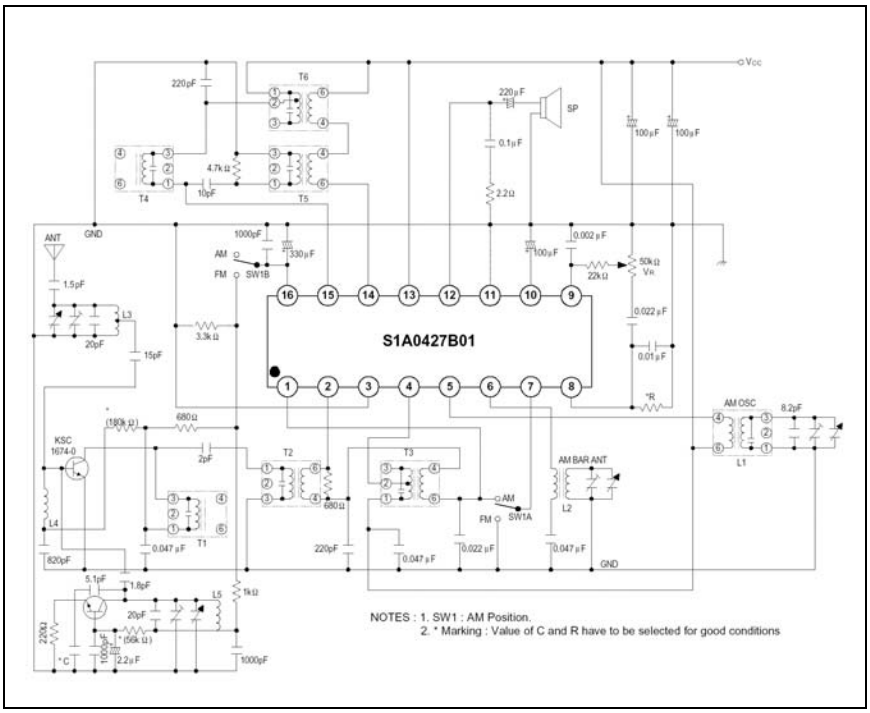
y con un número aceptable de componentes externos. Incluso se puede conseguir en RS Amidata por unos 6.5 €, y llegamos a obtener muestras de la casa Philips. Sin embargo, este circuito tenía dos principales inconvenientes. El primero de ellos, es el uso de componentes como el diodo varicap doble BB804, que no es fácil de conseguir, al igual que un cristal de cuarzo con una frecuencia de 75 kHz y distintos filtros cerámicos. Además, se presenta en un encapsulado tipo LQFP de 48 pines, lo que sumado a la imposibilidad de realizar este tipo de soldaduras (solamente podríamos con un adaptador que resulta demasiado voluminoso y caro), nos ha llevado a descartar a este candidato.



Montaje del TEA 5757HL.

4.1.5 Samsung S1A0427B01.

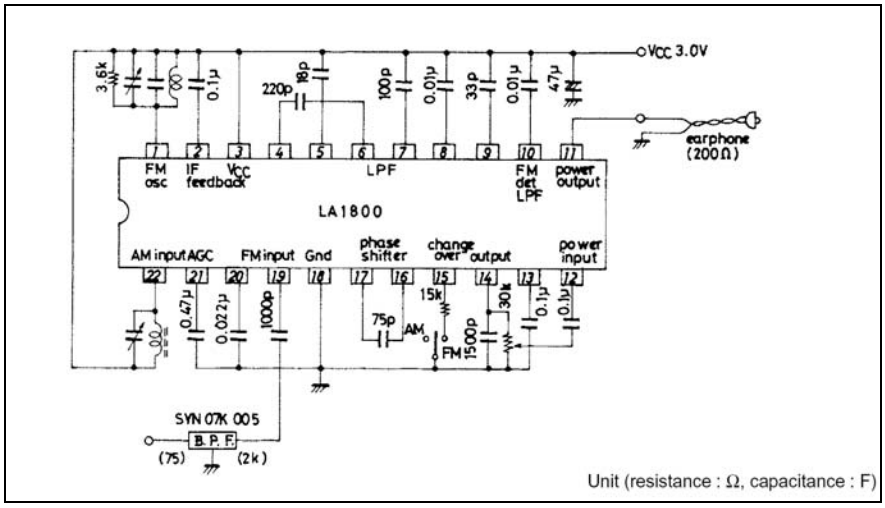
Este circuito, al igual que los anteriores, es un circuito de radio integrado, que se presenta en un encapsulado tipo DIP16, lo que, incluso no siendo de montaje superficial, le hace tener un tamaño reducido. Además, no requiere de demasiados componentes externos, y los que necesita no son muy raros, solamente algunos “botes” de radiofrecuencia. El principal problema de este circuito es la imposibilidad de conseguirlo, incluso a través de los grandes distribuidores internacionales. De no ser por esto, hubiera resultado ser un candidato más que aceptable para nuestro proyecto.



Montaje del SAMSUNG S1A0427B01

4.1.6 Sanyo LA1800.

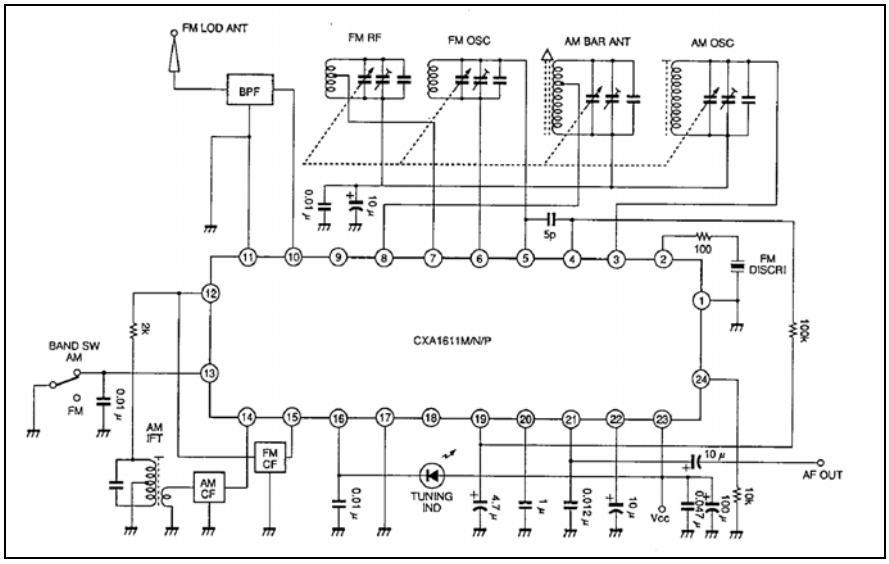
Este circuito receptor de radio FM/AM, tiene la particularidad de necesitar un reducido número de componentes externos, además de un circuito sintonizador. Se presenta con un encapsulado tipo DIP de 22 pines. El motivo de su inmediato descarte es la imposibilidad de conseguirlo, incluso a través de distribuidores internacionales. Queda por tanto, como otra muestra de las diferentes opciones que nos encontramos en la búsqueda de circuito de radio integrados para las frecuencias de FM comercial.



Montaje del SANYO LA1800.

4.1.7 Sony CXA 1611.

Este circuito de SONY es otro de los denominados *radio on a chip*. Este circuito tiene la ventaja de necesitar pocos componentes externos. Sin embargo, algunos de los componentes pasivos que requiere son bobinas, tanto para el oscilador del circuito de sintonización, como para el oscilador de frecuencia intermedia, lo que quiere decir que se necesitará un ajuste muy preciso de dichos componentes, lo cual aumenta la dificultad de su montaje en la práctica. Debido a que otros circuitos de los estudiados no requieren más que la bobina para el circuito de sintonización y a que este circuito de SONY no es fácil de conseguir, se ha descartado de los posibles candidatos.



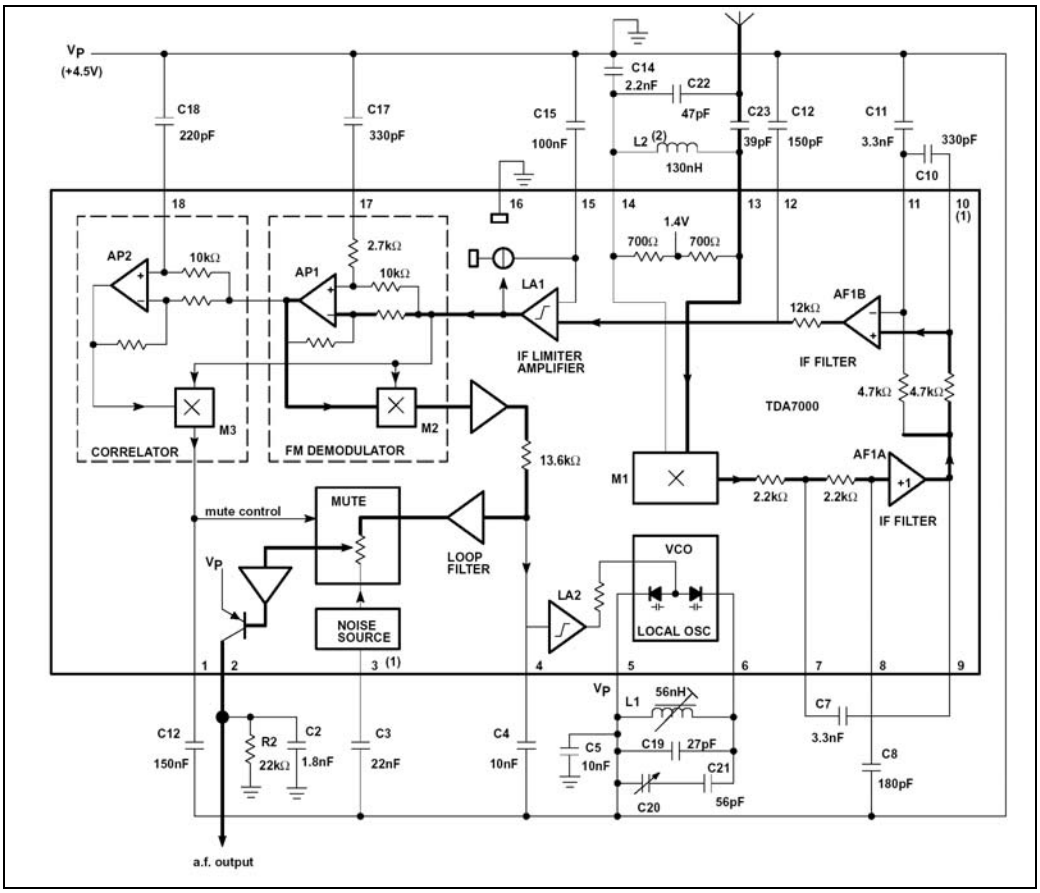
Montaje del SONY CXA 1611.

4.1.8 Philips TDA 7000.

Este circuito de radio integrado de la casa Philips ha sido el candidato elegido para este proyecto. Es un circuito integrado con todas las etapas de un receptor heterodino, que requiere de un reducido número de componentes externos (algunas resistencias, condensadores y un par de bobinas). Se presenta en un encapsulado tipo DIP de 18 pines y dispone de un "hermano gemelo" en montaje superficial (TDA7010).

Lo más atractivo de este circuito, además de su más que aceptable calidad con una baja complejidad, es que se puede conseguir en cualquier distribuidor de componentes electrónicos (Electroson, RS Amidata, Farnell, ...) y a un precio bastante reducido (2 € en Electrosón y 2,51 € en RS).

Por todo esto, este circuito (TDA7000) ha sido elegido para ser el *núcleo* de nuestro circuito receptor de radio FM.



Montaje del PHILIPS TDA 7000.

4.2 CONCLUSIONES

Después de estudiar varios circuitos de radio y haber elegido el TDA 7000 de la casa Philips, podemos llegar a la conclusión de que aparentemente existen en el mercado gran cantidad de circuitos de radio integrados. Sin embargo, el mayor problema que hemos encontrado, que además es causa de la mayor parte de los descartes, es la imposibilidad de conseguirlos incluso a través de los distribuidores internacionales de mayor prestigio.

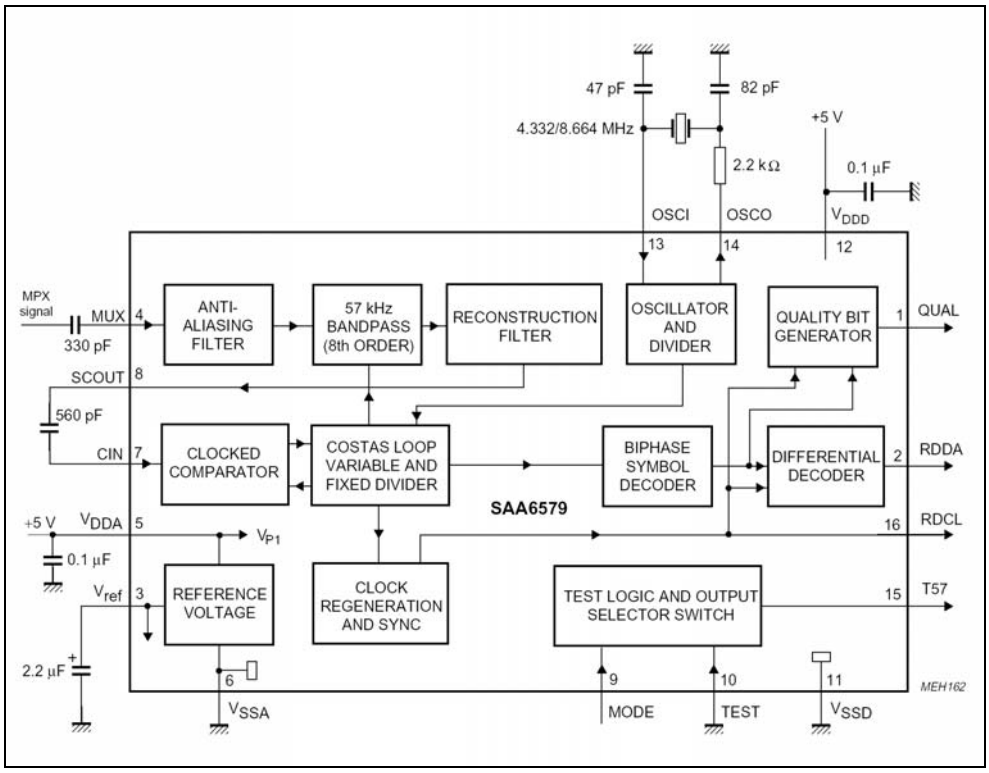
No obstante, considero que hemos tenido bastante suerte al poder disponer de un circuito tan barato y de buena calidad que además no necesita gran número de componentes externos, como es el Philips TDA 7000. De todas formas, si tuviéramos algo que decir en contra del TDA 7000, es que hubiera sido deseable haber podido disponer de circuitos con salida en

ESTEREO, que disponen de un mayor ancho de banda a la salida del demodulador, lo que nos hubiera beneficiado a la hora de obtener la señal RDS con mayor calidad.

4.3 CIRCUITOS DECODIFICADORES DE RDS

4.3.1 Philips SAA6579.

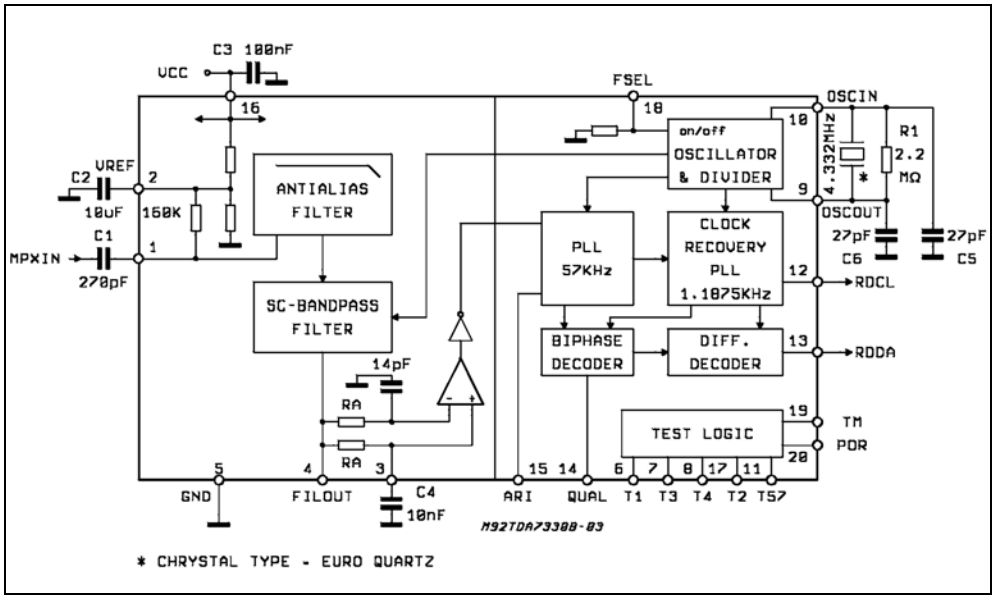
Este circuito integrado es un decodificador RDS de la casa Philips. Se trata de un circuito que requiere unos pocos componentes externos, además del necesario cristal de cuarzo de 4.332 MHz (indispensable en cualquier decodificador RDS y muy difícil de conseguir). Podemos conseguirlo en distribuidores tipo AVNET ó ARROW por un precio de unos 3.5 € con encapsulado DIP16. Dispone de conexión por bus serie semejante al bus I2C. No va a ser el circuito elegido para este proyecto, debido a que hay opciones mejores en el mercado.



Montaje del SAA 6579 de Philips.

4.3.2 ST-Microelectronics TDA 7330.

Es otro de los circuitos decodificadores de RDS que podemos encontrar en el mercado. Se vende en distribuidores internacionales por un precio aproximado de unos 4.5 €. Se presenta con un encapsulado tipo DIP de 20 pines y dispone de una conexión por bus s serie de dos líneas semejante al del circuito anterior. Necesita de pocos componentes externos (además del cristal de 4.332 MHz) y es muy popular en los distintos montajes que podemos encontrar en internet. Tampoco va a resultar ser el candidato elegido por haber mejores opciones en el mercado.



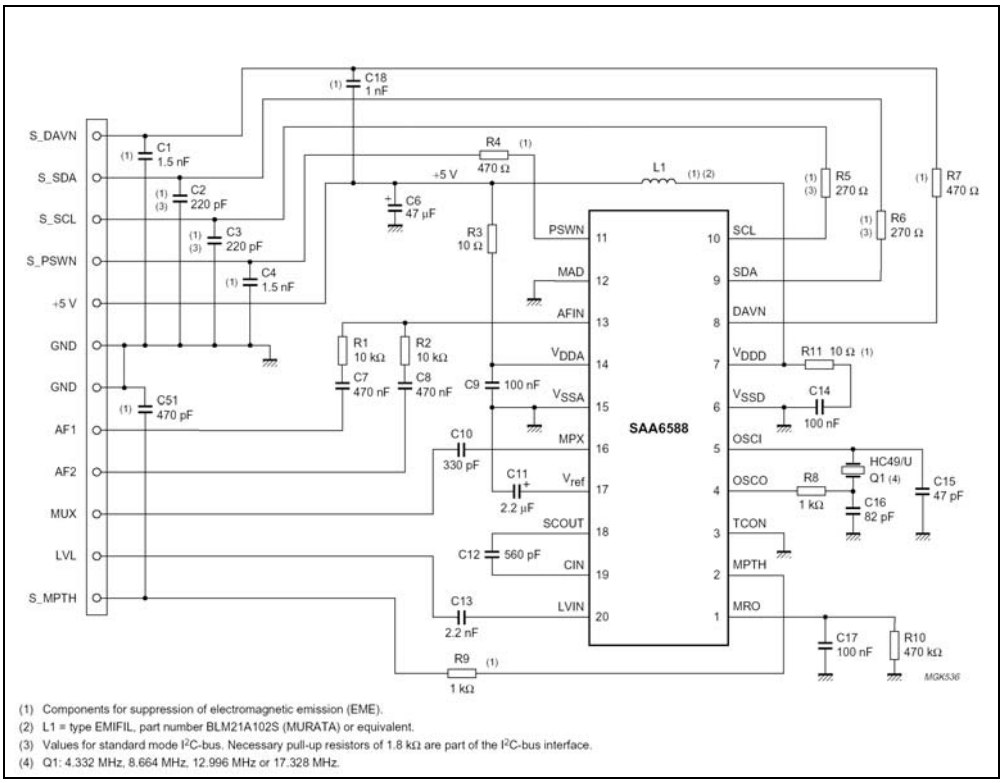
Montaje del TDA 7330 de ST Microelectronics.

4.3.3 Philips SAA 6588.

Este circuito decodificador de RDS ha sido el candidato elegido para el presente proyecto. El principal motivo es que además de decodificar la señal RDS, actúa como pre-procesador de la misma. Debido a esto, almacena la información en unos registros de salida (7 registros de salida) y dispone de 3 registros de entrada donde se especifica la configuración del funcionamiento del circuito. Además, nos ofrece información sobre la calidad de la señal RDS, el número de errores recibidos en cada bloque, dispone de corrección de errores, auto-sincronización con la señal RDS y tiene una

conexión tipo bus I2C. Se puede conseguir en distribuidores como EURODIS ELECTRONICS por un precio de aproximadamente 5.5 €.

Podremos ver una explicación más detallada de este circuito a lo largo de este documento.



Montaje del SAA 6588 de Philips.

4.4 CONCLUSIONES.

Después del estudio de las diferentes opciones que encontramos en el mercado, podemos observar que no hay demasiadas opciones donde elegir si lo que buscamos es un decodificador RDS. Y más en concreto, si buscamos circuitos pre-procesadores de la señal RDS como el elegido, veremos que no hay más que el SAA 6588 de Philips (al menos así lo he comprobado yo). Pero el principal problema con el que nos encontramos, es con la gran dificultad para conseguir el cristal de cuarzo de 4.332 MHz que necesitan los decodificadores de RDS. En mi humilde opinión, creo que los fabricantes podían darse cuenta de este problema y fabricar dispositivos con estos cristales integrados, si es que es posible. Pero esto no es el objeto del presente proyecto por lo que no nos extenderemos más hablando de este tema.

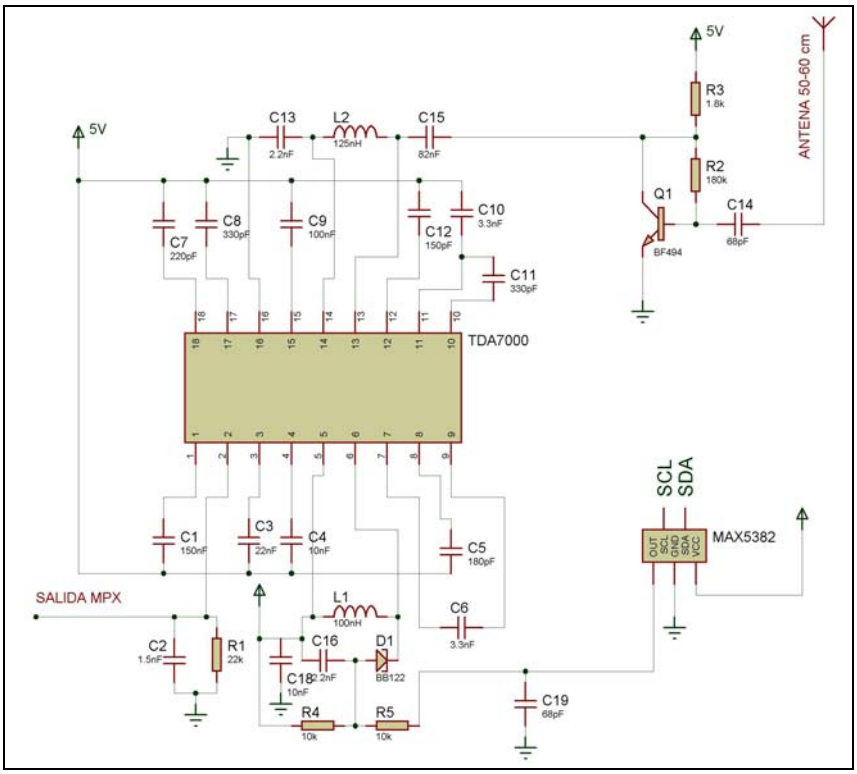
5. EL SISTEMA DECODIFICADOR FM-RDS

A continuación vamos a describir en detalle los circuitos que componen el sistema decodificador FM-RDS. Son dos circuitos, que se encuentran en placas de circuito impreso separadas y conectadas entre si, uno de los cuales se encarga de la recepción de la señal FM-RDS, y el otro se encarga de controlar el funcionamiento del sistema así como de la decodificación e interpretación de los datos obtenidos de la señal RDS.

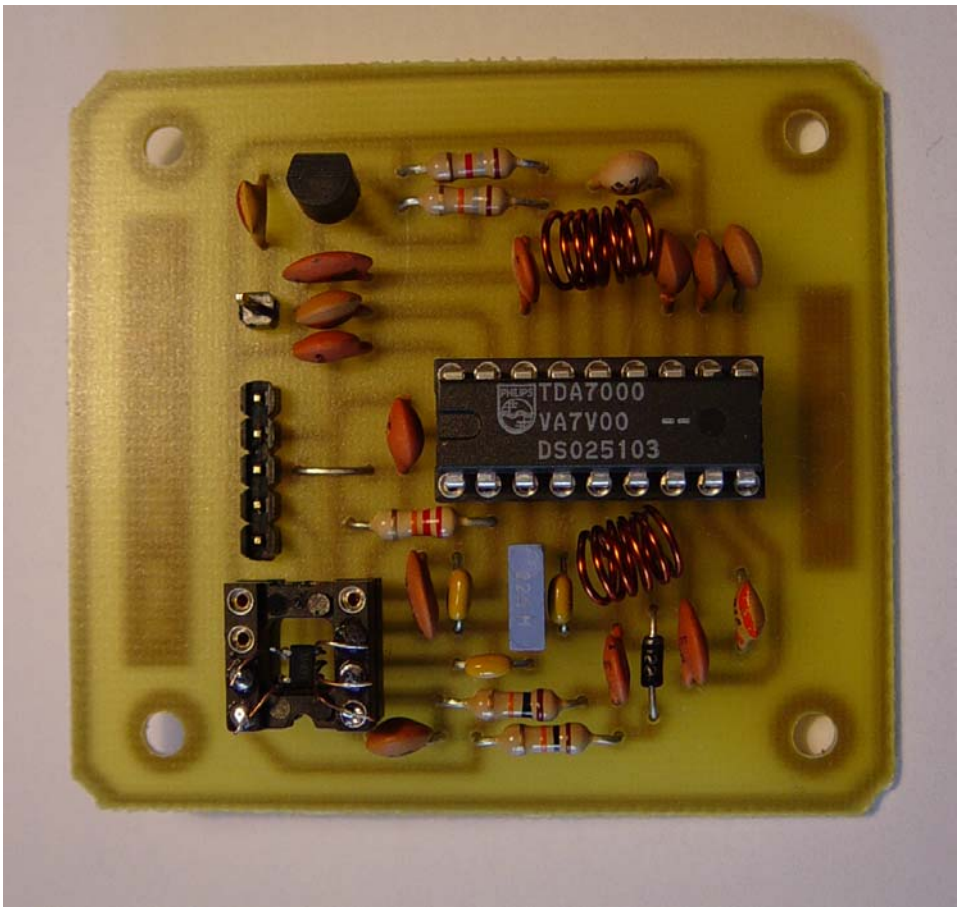
5.1 EL CIRCUITO RECEPTOR DE RADIO FM.

El circuito receptor de radio FM es el encargado de obtener la información emitida por las diferentes emisoras que encontramos en el intervalo de frecuencias de 88 a 108 MHz.

Se trata de un circuito receptor de sintonización automática, gestionada por el microcontrolador vía I2C, compuesto por un circuito integrado de radio (Philips-TDA7000), un conversor digital-analógico (Maxim-MAX 5382), un diodo varicap (BB122), un transistor de radiofrecuencia (BF494) y algunos componentes pasivos, como podemos ver en la figura.



Esquema del circuito receptor de radio FM.

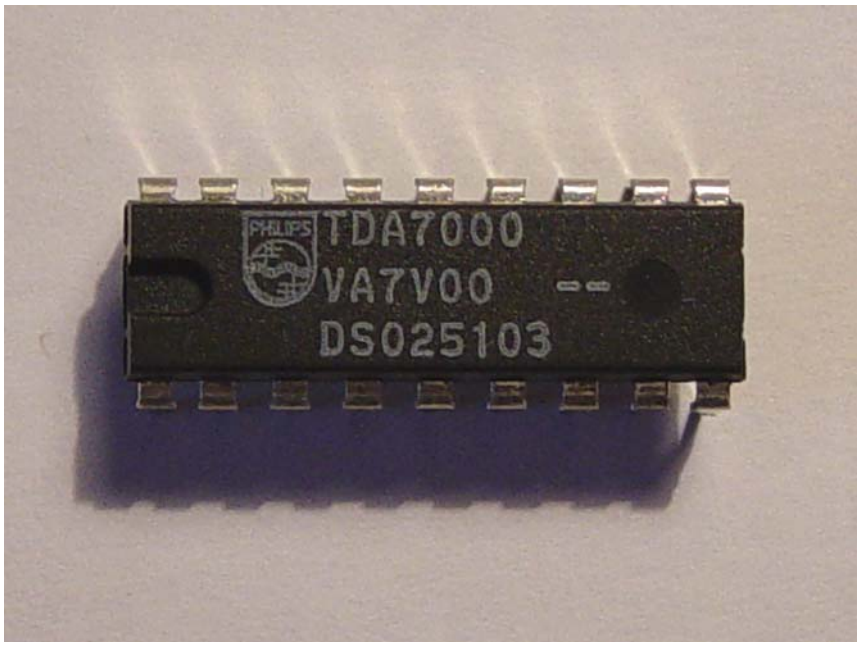


Detalle del circuito receptor de radio FM.

5.1.1 El circuito integrado TDA7000

El TDA7000 de la casa Philips es un circuito integrado monolítico diseñado para radios FM monoaurales portátiles que necesita un reducido número de componentes pasivos, y que trabaja con una frecuencia intermedia (IF) de 70 kHz, por lo que las radiaciones del circuito durante su funcionamiento son despreciables.

El TDA7000 consta de un oscilador local y un mezclador, un filtro activo de frecuencia intermedia de dos etapas seguido de un limitador / amplificador de IF, un demodulador FM de cuadratura, y un circuito de "*mute*" de audio.



Detalle del circuito integrado TDA 7000 de Philips.

Este circuito integrado tiene un sistema FLL (Frequency-Locked-Loop) con una frecuencia intermedia de 70 kHz.. La selectividad de la frecuencia intermedia es obtenida con filtros RC activos. La única función que necesita realizar ajustes es la del circuito resonante del oscilador, la cual selecciona la frecuencia de recepción. La recepción de espúreos es evitada por el circuito de mute, el cual también elimina señales demasiado ruidosas.

En el datasheet del TDA7000, incluido en los apéndices de este documento, podemos ver una descripción más en profundidad de dicho integrado.

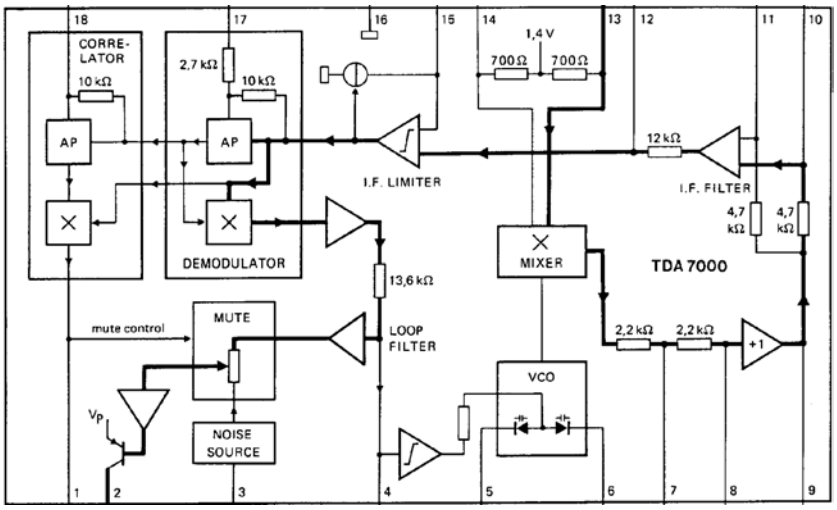


Diagrama de bloques del circuito integrado TDA7000.

5.1.2 El Conversor Digital Analógico MAX5382

El CDA MAX5382 es un conversor digital-analógico de 8 bits de la casa MAXIM. Tiene un encapsulado "miniatura" tipo SOT23, pensado para montaje superficial, con solamente 5 pines, y un sencillo interfaz de 2 hilos con protocolo I2C a través del cual puede comunicarse con múltiples dispositivos.

Este integrado trabaja en un rango de alimentación de +2,7 a +5,5 V, y tiene una tensión de referencia interna de 0,9xVdd, lo cual, en nuestro caso de alimentación a +5V, se traduce en un rango de tensión de salida de 0 a 4,5V.

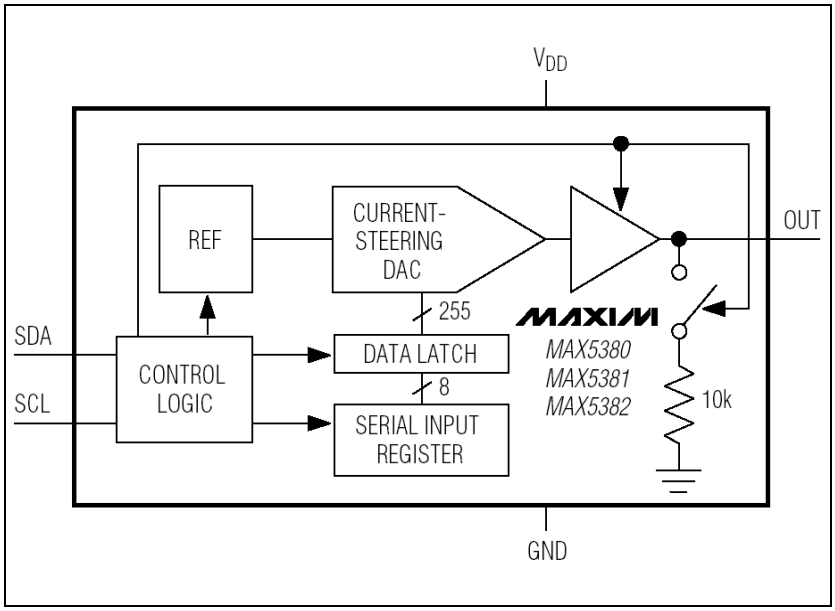


Diagrama de bloques del CDA MAX5382.

En el datasheet del MAX5382, incluido en los apéndices de este documento, podemos ver una descripción más en profundidad de dicho integrado.

5.1.3 El circuito de sintonización.

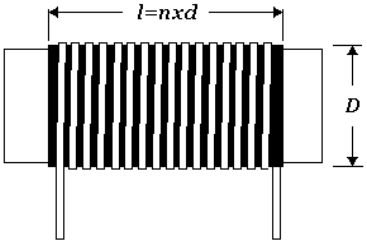
La sintonización de la radio se realiza variando la tensión aplicada en inversa al diodo varicap BB122, de manera que este varíe su capacidad y así estamos cambiando la frecuencia

de resonancia del circuito tanque formado por la bobina L1, el condensador C16 y el diodo varicap BB122, de acuerdo con la fórmula:

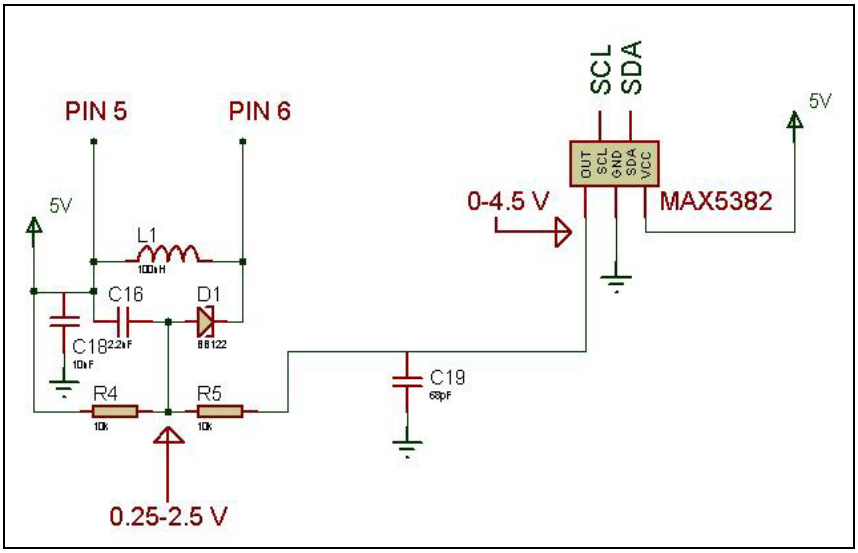
$$f_r = \frac{1}{2\pi\sqrt{LC}}$$

El valor de la bobina utilizada se calculó a través de la fórmula siguiente usada en cálculo de bobinas:

$$L(\mu H) = \frac{D^2 \cdot n^2}{46 \cdot D + 102 \cdot l}$$



Esta frecuencia de resonancia es aplicada directamente entre las pines 5 y 6 del circuito integrado TDA7000, que conducen a un mezclador, obteniendo a la salida de este una frecuencia intermedia de 70 kHz.

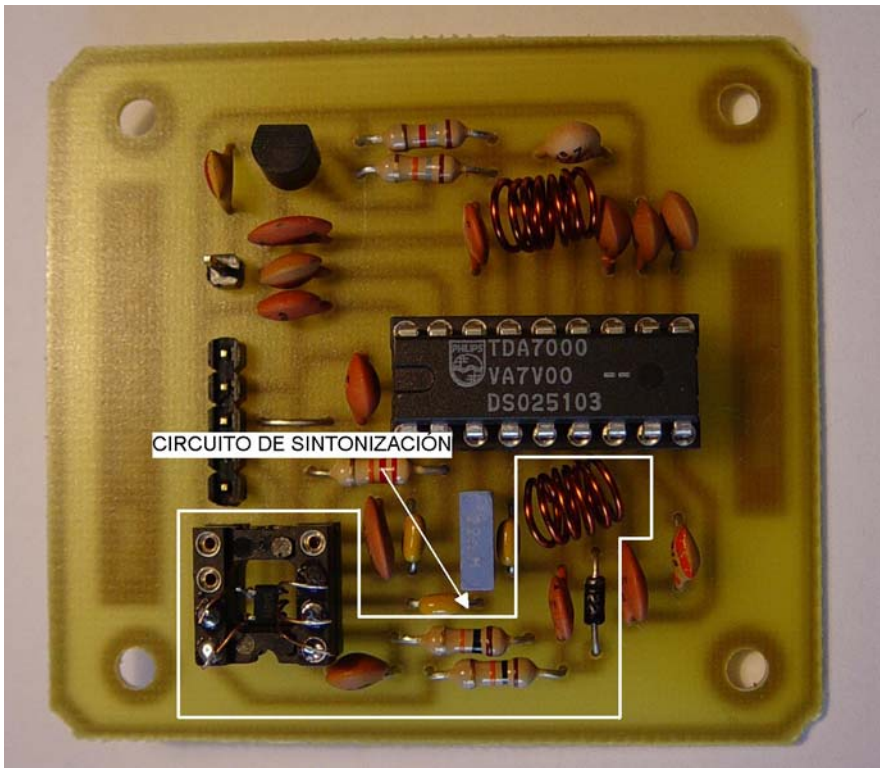


Esquema del circuito de sintonización de la radio FM.

La tensión de sintonización es proporcionada por el conversor digital-analógico MAX5382, que a partir de los datos digitales recibidos desde el microcontrolador, genera una tensión continua y estabilizada a su salida. Esta tensión es aplicada al varicap a través de un divisor de tensión formado por las resistencias R4 y R5.

El objeto de este divisor de tensión es el de aumentar la precisión en la sintonización, ya que con el rango de salida del conversor D/A (0-4,5V), la frecuencia de resonancia del circuito sintonizador tiene una variación de aproximadamente 40 MHz, mientras que el intervalo de FM comercial es de unos 20 MHz. Por lo tanto, este divisor de tensión, convierte la variación de tensión aplicada al ánodo del varicap de 4,5 V a 2.25 V.

Con este rango de tensión aplicada al ánodo del varicap y teniendo en cuenta que en el cátodo tenemos una tensión fija de 5V, el rango de tensión inversa aplicada al varicap es de 2.5-4.75V, lo que posteriormente traducimos en un rango de frecuencias de aproximadamente 88-108 MHz.

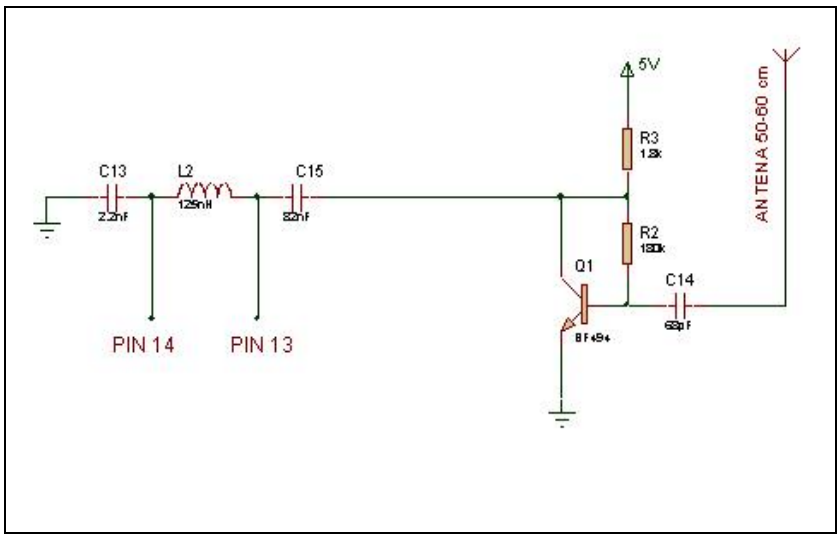


Detalle del circuito de sintonización.

5.1.4 El circuito de recepción.

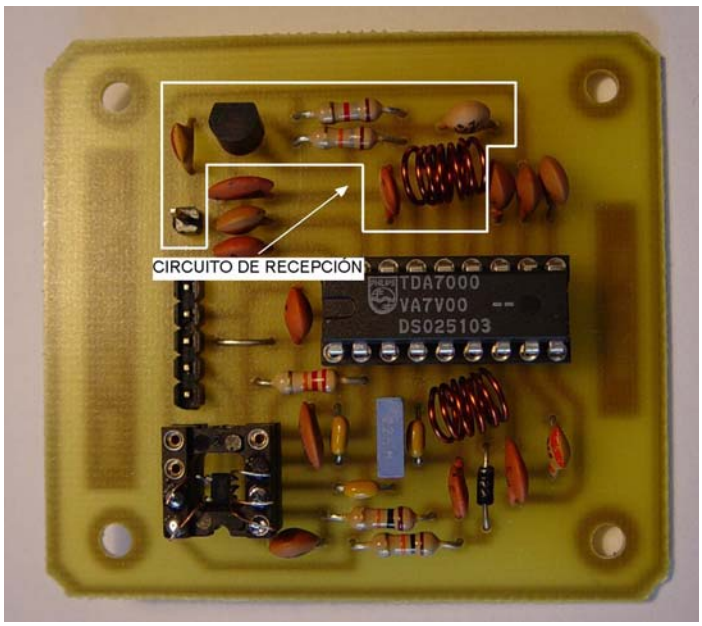
La recepción de la señal FM se realiza a través de un amplificador de radiofrecuencia formado por las resistencias R2 y R3, y el transistor de radiofrecuencia Q1.

Una vez amplificada la señal, esta pasa por un filtro pasabanda LC formado por la bobina L2 y los condensadores C13 y C15, que nos elimina posibles interferencias en la recepción de frecuencias fuera de la banda de FM comercial.



Esquema del circuito de recepción de la señal FM.

Una vez amplificada y filtrada la señal recibida, se introduce directamente en el circuito integrado TDA7000 a través de los pines 13 y 14, que conducen al mezclador, que junto con la frecuencia obtenida del circuito de sintonización, nos proporciona a la salida una reducción de la frecuencia de la señal modulada FM a una frecuencia intermedia de 70 kHz.

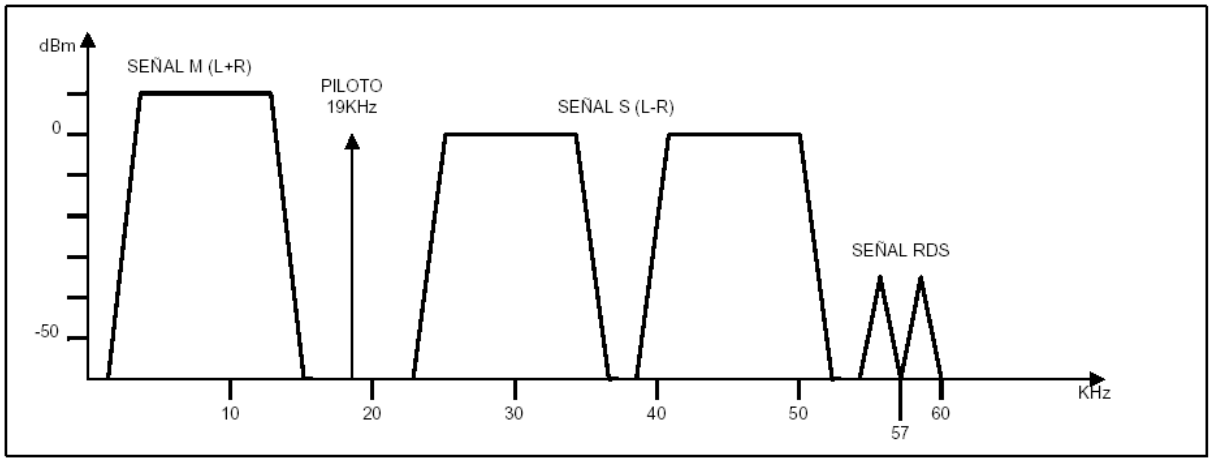


Detalle del circuito de recepción.

La antena es un hilo de cobre aislado con una longitud de entre 50 y 60 cm, formando de esta manera una antena de cuarto de onda.

5.1.5 La señal compuesta MPX.

La señal compuesta MPX ó *múltiplex,* es donde se encuentra modulada la información tanto de audio como de RDS, y la podemos obtener del pin 2 del circuito integrado TDA7000. De este pin sacaremos la señal RDS para introducirla directamente en el preprocesador de RDS.



Espectro de la señal compuesta MPX.

5.2 EL SISTEMA DECODIFICADOR RDS

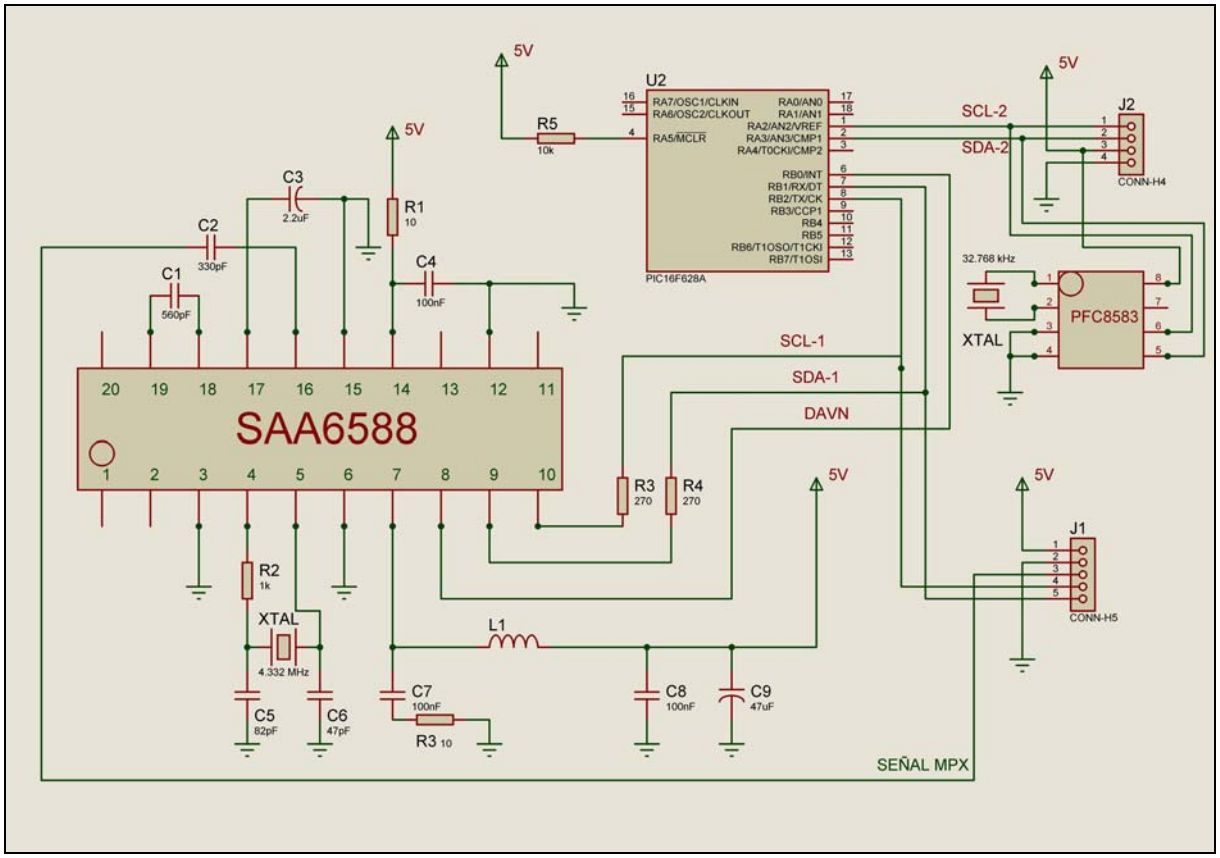
Se ha denominado así a la parte del circuito encargada de demodular, decodificar y procesar la señal RDS, para la obtención de la información relativa a la fecha y la hora así como la relativa a la identificación de la emisora.

Los componentes principales de esta parte son, además de un cristal de cuarzo y algunos pasivos, un circuito reloj/calendario, un microcontrolador y un preprocesador de la señal RDS.

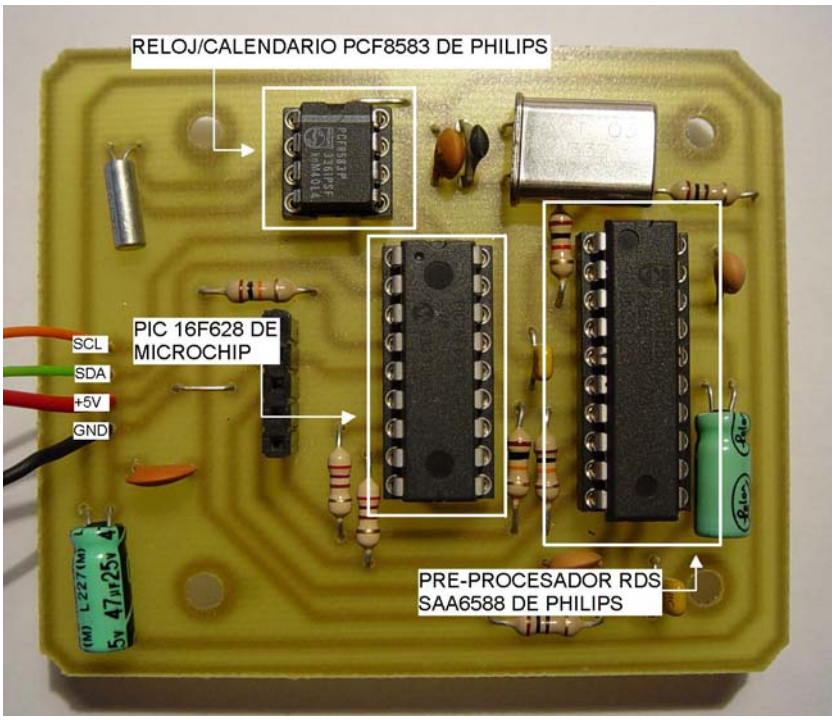
- El microcontrolador se encarga de gestionar el funcionamiento de todo el sistema, tanto del preprocesador RDS, como del circuito de radio, así como de la decodificación de la información enviada por el preprocesador RDS.

- El preprocesador se encargará de demodular la señal RDS, corregir los errores y ordenar la información para enviarla al microcontrolador vía bus I2C.

- El reloj/calendario será actualizado por el sistema vía I2C, y podrá ser consultado de la misma manera en cualquier momento.



Esquema del sistema decodificador RDS



Detalle del sistema decodificador RDS.

Hay que decir que en este circuito podemos encontrar dos buses I2C. Uno de ellos es el que comunica el microcontrolador (por los pines RB1 –SDA- y RB2 –SCL-) con el conversor digital-analógico del circuito de radio y con el pre-procesador RDS, haciendo posible la sintonización y la obtención de la información RDS.
El otro bus I2C es el que comunica el microcontrolador (pines RA2 –SCL- y RA3 –SDA-) con el exterior del sistema y con el reloj/calendario PCF8583 de Philips. Por este bus obtendremos la información de la fecha y la hora actualizadas.

De estos dos buses I2C, en el sistema sólo se incluyen las resistencias de pull-up del primero de ellos (el que une microcontrolador con el conversor digital-analógico y el pre-procesador), ya que las correspondientes al otro bus deben estar incluidas en el sistema en el cual se vaya a integrar este circuito, que funcionará como si fuera un dispositivo esclavo.

En la parte inferior de la placa del sistema decodificador RDS, se ha soldado la inductancia BLM21A102S de la casa MURATA, y que está reflejada en el esquema eléctrico del circuito como L1. Esta inductancia sirve como filtro para interferencias electromagnéticas.

A continuación, vamos a ver brevemente el funcionamiento de esta parte del sistema, ya que estará explicado con más detalle en el apartado "FUNCIONAMIENTO DEL SISTEMA".

5.2.1 El preprocesador RDS.

El preprocesador RDS es un integrado de la casa PHILIPS denominado SAA6588, que se encarga, principalmente, de la demodulación de la señal RDS obtenida a partir de la señal compuesta ó multiplex, proveniente de la salida del demodulador del receptor de radio.

El preprocesador, pasa la señal multiplex por un filtro pasabanda de capacidad conmutada de octavo orden centrado en 57 KHz, eliminando así los componentes de audio de la señal. Esta señal filtrada, es demodulada y digitalizada a continuación, para pasar a decodificarla, estando así disponible, vía bus I2C, la información contenida en la señal RDS.

Una vez demodulada la señal RDS, tenemos una "tira" continua de bits formando bloques y estos a su vez, grupos. El preprocesador se sincroniza cuando ha recibido dos bloques de forma correcta, procediendo entonces a su procesado y almacenamiento en los registros de salida.

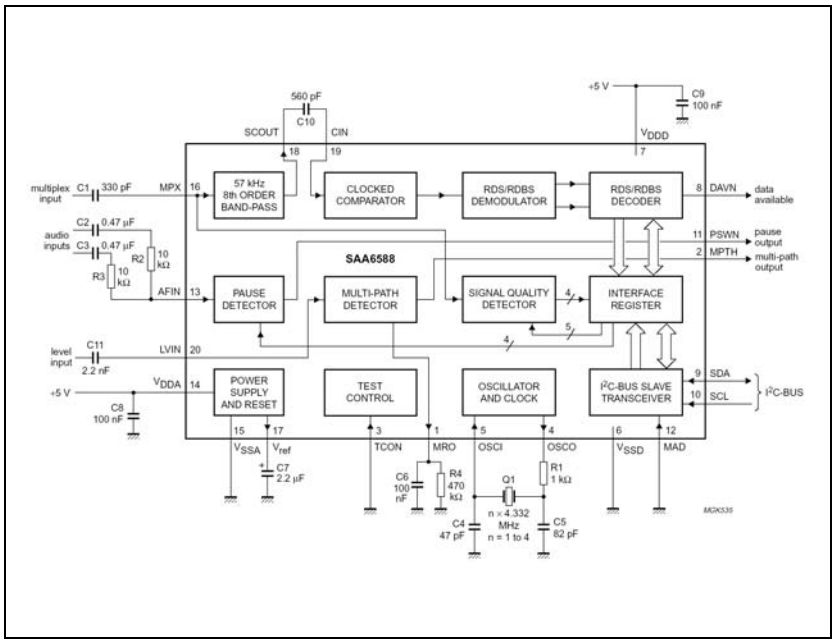
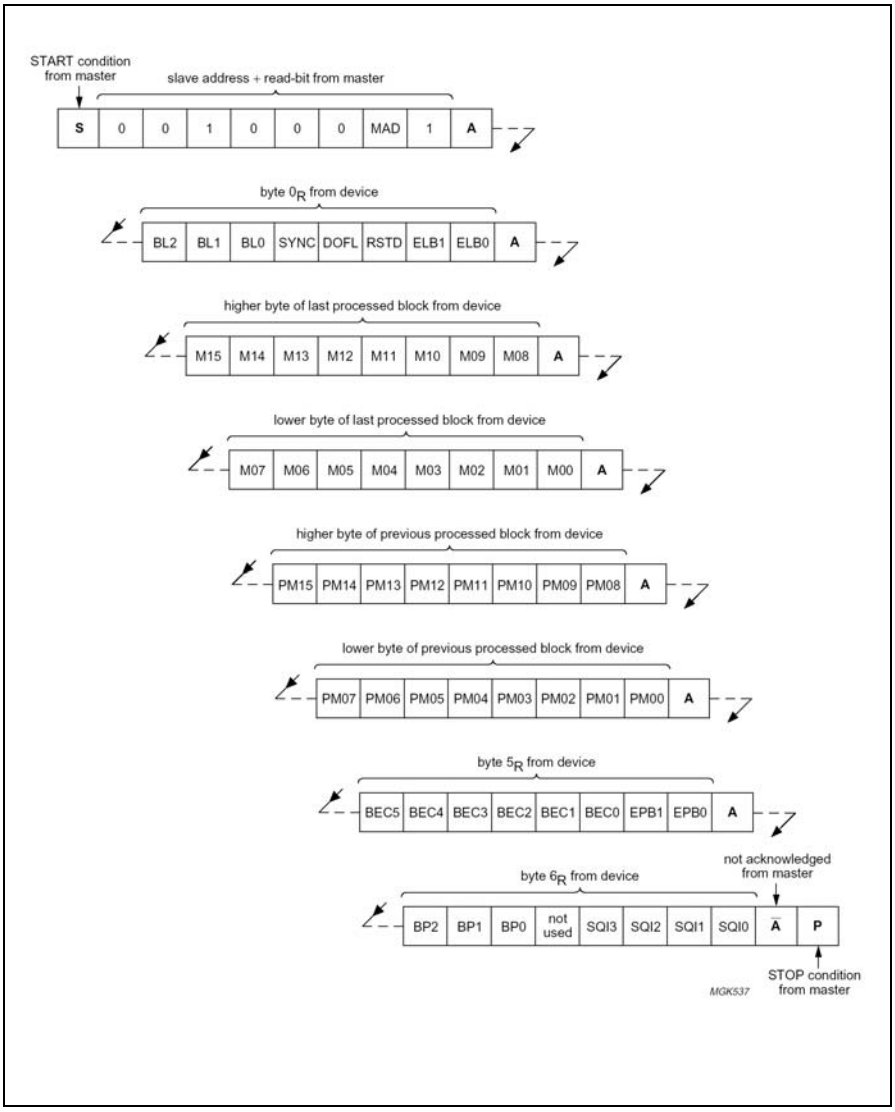


Diagrama de bloques del preprocesador SAA6588 de PHILIPS.

El preprocesador dispone de tres registros de entrada de control donde podremos especificar el modo de funcionamiento deseado del mismo vía bus I2C. Si no se especifica una configuración determinada, el sistema tiene una por defecto, que será la deseada en nuestro proyecto, por lo que no tendremos que escribir nada en los registros de control.

Además, disponemos de siete registros de salida, donde se irá almacenando la información RDS, así como la información relativa al estado del preprocesador, y que pueden ser leídos vía bus I2C.



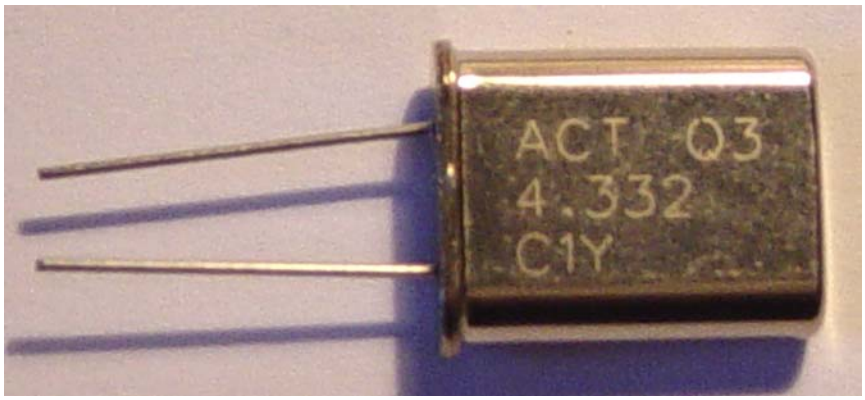
Protocolo de salida de datos del SAA6588.

5.2.2 Cristal de cuarzo para el decodificador RDS.

Este cristal de cuarzo es uno de los componente más difíciles de conseguir de todo el circuito. Se trata de un cristal de cuarzo con una frecuencia de oscilación de 4.332 MHz, sobre la cual se permite una variación máxima de 30 ppm. La exactitud de esta frecuencia es un parámetro crítico del sistema, ya que a partir de la frecuencia generada por el cristal, el pre-procesador de la señal RDS, puede obtener la subportadora de 57 kHz que lleva la información RDS. Esto lo hace a partir de un divisor de frecuencias que realiza la siguiente operación:

$$\boxed{\frac{4.332MHz}{76} = 57kHz}$$

En la siguiente imagen podemos ver un detalle del cristal de cuarzo de 4.332 MHz que se presenta, en este caso, en un encapsulado tipo HC49/U.



Detalle del cristal de cuarzo de 4.332 MHz.

5.2.3 Microcontrolador PIC16F628.

Este microcontrolador de la casa Microchip es el encargado del funcionamiento global del sistema. Es un microcontrolador de 8 bits con 18 pines, 16 de los cuales son de entrada/salida, organizados en dos puertos (A y B) de 8 pines cada uno de ellos.

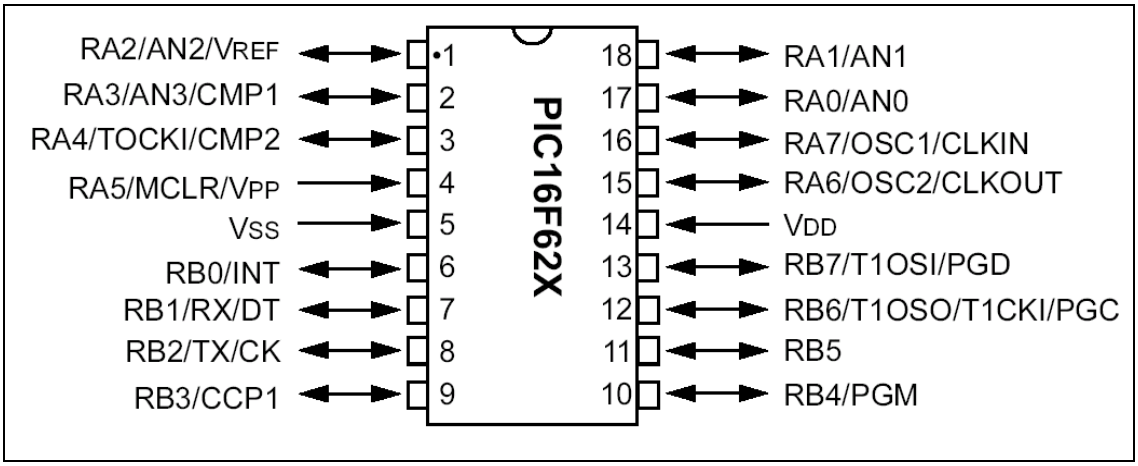


Diagrama de pines del PIC 16F628.

A continuación resumimos las características principales de este microcontrolador:

- Set de 35 instrucciones.
- Velocidad de trabajo de hasta 20 MHz.
- 2048 palabras de memoria FLASH de programa.
- 224 bytes de memoria RAM
- 128 bytes de memoria EEPROM.
- Palabras de instrucción de 14 bits.
- Anchura del bus de datos de 8 bits.
- 16 registros hardware con funciones especiales.
- Pila hardware de 8 niveles.
- Modos de direccionamiento directo, indirecto y relativo.
- 10 fuentes de interrupción.
- 16 pines de entrada/salida con control individual de dirección.
- Módulo comparador analógico.
- Módulo USART (Universal Synchronous Asynchronous Receiver Transmiter).
- 3 temporizadores.
- Módulo CCP (Compare, Capture, PWM).

En este sistema funciona con el oscilador interno (IntRC) que proporciona una velocidad de 4 MHz.

Detalle del PIC 16F628 de Microchip.

5.2.4 Reloj/Calendario PCF8583.

El PCF8583 es un circuito de reloj/calendario basado en una RAM estática CMOS de 2048 bits, organizados en 256 palabras de 8 bits cada una. Las direcciones y los datos son transferidos en serie a través del bus bidireccional I2C.

Este circuito, como ya hemos dicho, tiene una RAM de 256 palabras de 8 bits, un registro de dirección auto-incrementado de 8 bits, un circuito oscilador interno de 32.768 kHz, un divisor de frecuencias, un interfaz para bus serie bidireccional I2C y un circuito de "power-on reset".

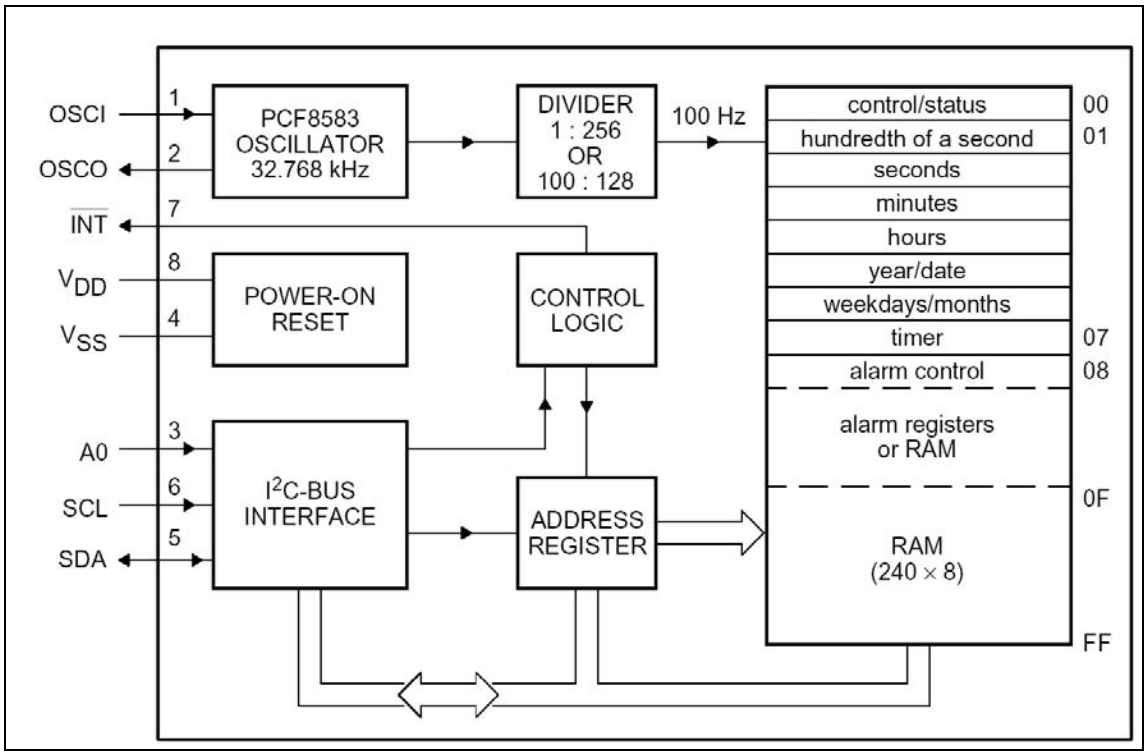


Diagrama de bloques del PCF8583.

Los primeros 16 bytes de la RAM están pensados como registros de función especial, de los cuales, el primero, es usado como un registro de estado/función. El resto de los 8 bytes, se usan como contadores de la función reloj.

En el datasheet del PCF8583, incluido en los apéndices de este documento, podemos ver una descripción más en profundidad de dicho integrado.

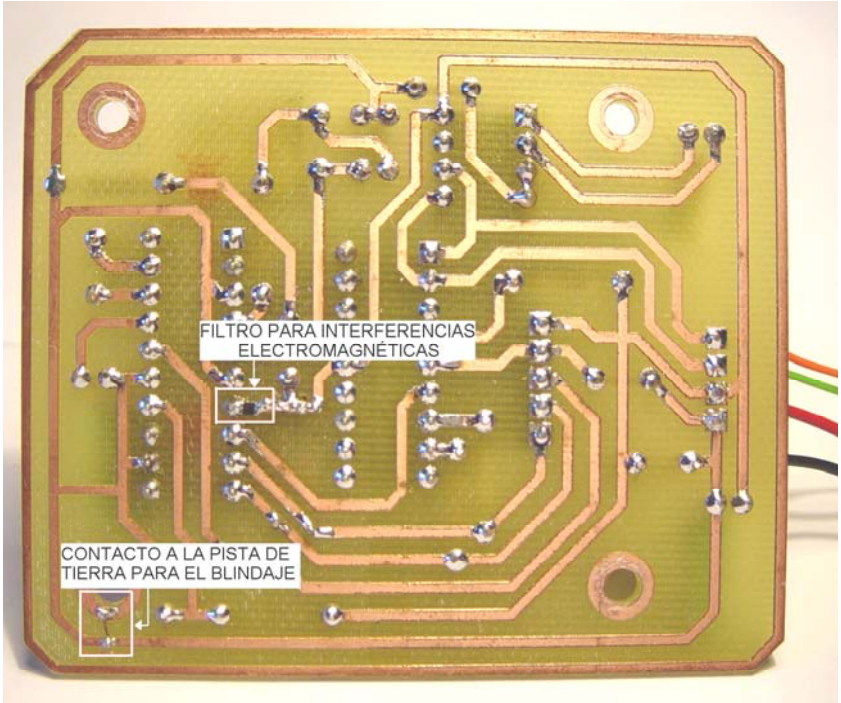
5.3 BLINDAJE Y AISLAMIENTO DE LOS CIRCUITOS.

Debido a problemas de compatibilidad electromagnética, ha sido necesario separar mediante un blindaje las dos placas de circuito que componen nuestro sistema. Esto se ha realizado introduciendo entre los dos circuitos una placa de cobre aislado con conexión a tierra, proporcionando de esta manera un plano de tierra que aísla electromagnéticamente a los dos circuitos. A continuación podemos ver una imagen de este blindaje.

Detalle de la placa de blindaje que sirve de plano de tierra.

La conexión a tierra se realiza a través de uno de los tornillos, cuyas tuercas hacen contacto tanto en el plano de tierra (como podemos ver en la foto), como en la placa inferior

del circuito (la correspondiente al decodificador RDS), la cual se ha puesto a tierra mediante un contacto .



Detalle de la parte inferior de la placa del decodificador RDS.

6. FUNCIONAMIENTO DEL SISTEMA

En este apartado vamos a explicar en detalle el sistema decodificador FM-RDS desde el punto de vista de su funcionamiento. Vamos a distinguir dos partes principales, la de la radio y la del decodificador de la señal RDS.

6.1 FUNCIONAMIENTO DEL CIRCUITO RECEPTOR DE RADIO FM.

Como ya hemos mencionado anteriormente, este circuito es el encargado de sintonizar las emisoras y recibir la información RDS de la que obtendremos la fecha y la hora actualizadas.

El funcionamiento de este circuito está gestionado por el microcontrolador del sistema. Este, envía la información por medio del I2C al conversor digital-analógico para que vaya aumentando progresivamente su tensión de salida, que se aplica directamente al diodo varicap, que inicialmente tiene una tensión inversa de 2.5V. Al ser el conversor D/A de 8 bits, tendremos la posibilidad de recorrer 256 niveles distintos en la banda de frecuencias de 88 MHz a 108 MHz aproximadamente. Esto implica que el salto de frecuencia entre dos niveles seguidos es de aproximadamente 74 kHz.
Al ir aumentando la tensión de salida del conversor, la tensión en inversa aplicada al diodo varicap es cada vez más pequeña, y por lo tanto, su capacidad más grande. Cuanto más grande sea la capacidad del diodo, más pequeña es la frecuencia de resonancia del circuito oscilador, de acuerdo con la fórmula:

$$\boxed{f_r = \frac{1}{2\pi\sqrt{LC}}}$$

Por lo tanto, vemos que en el circuito receptor de radio FM hace un recorrido desde 108 MHz a 88 MHz. Cuando encuentra alguna emisora que emita una señal RDS, se detiene momentáneamente para recoger la información relativa a la identificación de la emisora. Esta

información se encuentra contenida en los 16 bits que conforman un bloque con el código PI (*Programme identification*). Este código, ocupa el primer bloque en los grupos versión A y en los grupos versión B se repite en también en el tercer bloque. El código PI contiene un código que identifica a la emisora. Los cuatro primeros bits indican el país de la emisión, los siguientes cuatro la cobertura de emisión (local, internacional, nacional, seminacional o regional), y los últimos 8 bits son asignados por un comité en cada país para identificar la emisora.

Si la emisora sintonizada no es de Radio Nacional de España (1,2,3,4 ó 5), se ignora y se continúa la búsqueda. Si pertenece a RNE, bien sea RNE1, RNE2, RNE3, RNE4 ó RNE5, se comprueba si la recepción tiene errores. Si es así, se vuelve a aumentar la posición del sintonizador y se vuelve a comprobar si hay errores. Esto se hace hasta que se reciba sin errores la señal RDS ó nos salgamos de la frecuencia de esa emisora, descartándola por mala recepción, y busquemos la siguiente emisora de RNE.

Se han elegido las emisoras de Radio Nacional de España ya que se ha observado que son las únicas emisoras que emiten la hora y la fecha actualizadas cada minuto en la provincia de Valladolid, lugar en el que se ha desarrollado el presente proyecto.

El algoritmo de búsqueda que sigue el microcontrolador está reflejado en la siguiente figura que muestra el diagrama de flujo de dicho algoritmo.

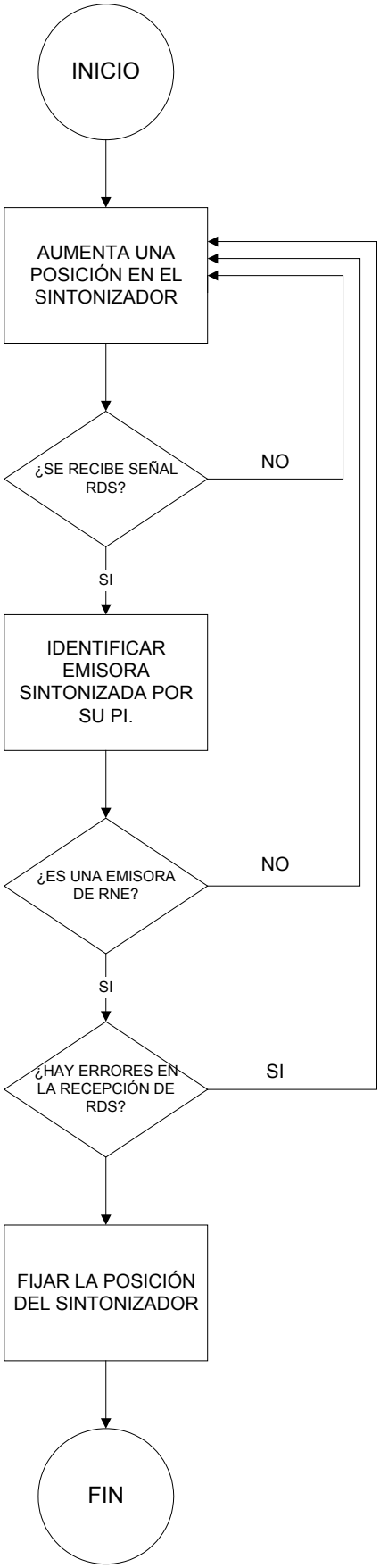


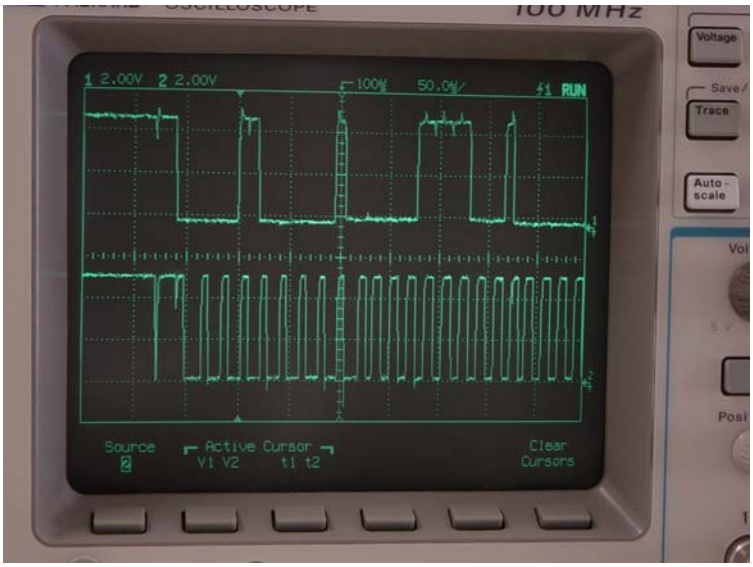
Diagrama de flujo del algoritmo de búsqueda de emisora.

6.2 FUNCIONAMIENTO DEL CIRCUITO DECODIFICADOR DE RDS.

Este circuito es el encargado de demodular, decodificar y procesar la señal RDS, además de gestionar el funcionamiento de todo el sistema (incluyendo la radio) desde el microcontrolador PIC 16F628 que incorpora. Como ya se ha explicado el funcionamiento del circuito receptor de radio FM, vamos a explicar que es lo que ocurre una vez que ya tenemos la señal MPX en el circuito decodificador RDS.

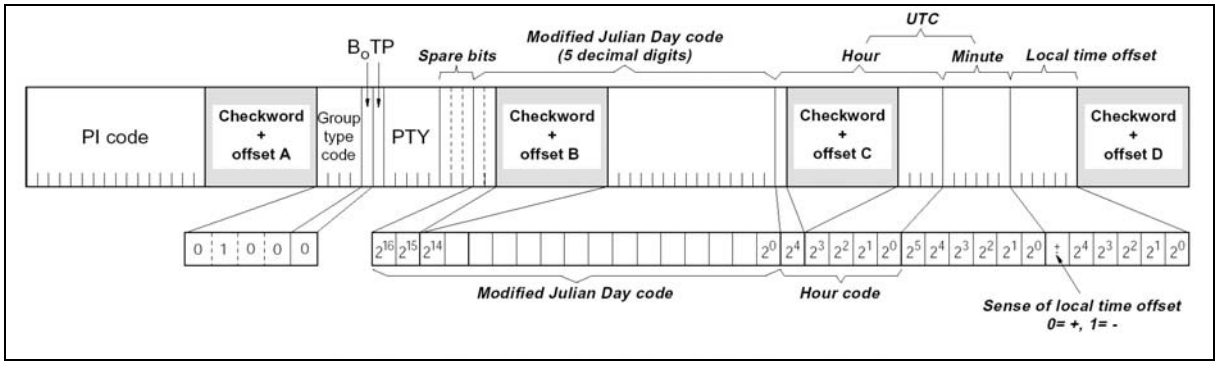
En primer lugar, la señal MPX entra al pre-procesador RDS SAA6588 de Philips, por el pin 16 de este. El pre-procesador filtra y amplifica la señal para, posteriormente, demodularla, decodificarla y almacenarla en bloques en los registros de salida de los que dispone.

Una vez que tenemos la información RDS disponible en los registros de salida del pre-procesador, los vamos "recogiendo" vía bus I2C desde el microcontrolador para procesar la información. Para esto, el microcontrolador "solicita" la información por el bus I2C al pre-procesador, cada vez que este le avisa de que dispone de información actualizada (cada 21 ms) poniendo a nivel bajo la línea DAVN, que entra al PIC a través del pin RB0. En el PIC se encuentran habilitadas las interrupciones por flanco de bajada del pin RB0, por lo que al recibir el aviso, el microcontrolador paraliza lo que esté haciendo para solicitar y recibir la información RDS.



Detalle de la comunicación I2C visualizada en el osciloscopio. La gráfica superior se corresponde con la línea SDA del bus I2C, mientras que la inferior corresponde a la línea SCL.

En el momento en el que el microcontrolador dispone de la información actualizada RDS, comienza a procesar dicha información. Primeramente, la almacena en unos registros para poder disponer de la información durante todo el procesamiento de la misma. Una vez almacenada, comprobaremos si los datos obtenidos tienen errores (información facilitada por el SAA6588). En caso afirmativo, con un sólo error que tengamos, se descartan esos datos y esperamos a que lleguen los siguientes. Si los datos no contienen errores, lo que hacemos es comprobar que tipo de información estamos recibiendo, es decir, identificamos el grupo al que pertenecen esos bloques de datos RDS. Si los datos que estamos recibiendo no pertenecen al grupo 4A (información de la fecha y la hora), los descartamos, ya que el objeto de este proyecto es la de obtener únicamente la fecha y hora actualizadas de la señal RDS.



Información contenida en el grupo 4A.

Si los datos recibidos pertenecen al grupo 4A, en primer lugar son almacenados en otros registros colocados según sean información relativa a la hora, minutos, día, mes ó año, para poder manejarlos en el procesado sin alterar la información recibida. Una vez almacenados, los datos de la fecha deben ser traducidos al calendario normal, ya que la fecha viene expresada en términos de calendario Juliano modificado (MJC). Esto se consigue aplicando las siguientes fórmulas de conversión:

$$Y' = \text{int}\left[\frac{MJD - 15078.2}{365.25}\right]$$

$$M' = \text{int}\left[\frac{MJD - 14956.1 - \text{int}(Y' \cdot 365.25)}{30.6001}\right]$$

$$Día = MJD - 14956 - \text{int}(Y' \cdot 365.25) - \text{int}(M' \cdot 30.6001)$$

$$Año = Y' + K$$; dónde K = 1 si M' es 14 ó 15. Si no, K = 0.

$$Mes = M' - 1 - K \cdot 12$$

En cuanto tenemos los datos organizados en archivos, y la fecha expresada en términos del calendario UTC, debemos actualizar la fecha y la hora del PCF8583, pero antes debemos comprobar si la información es válida o no. Esto se hace porque en condiciones de recepción pobre de la emisora FM, la información obtenida puede estar corrompida, aunque el pre-procesador no la identifique como errónea. A continuación podemos ver el diagrama de flujo del algoritmo de actualización de fecha y hora.

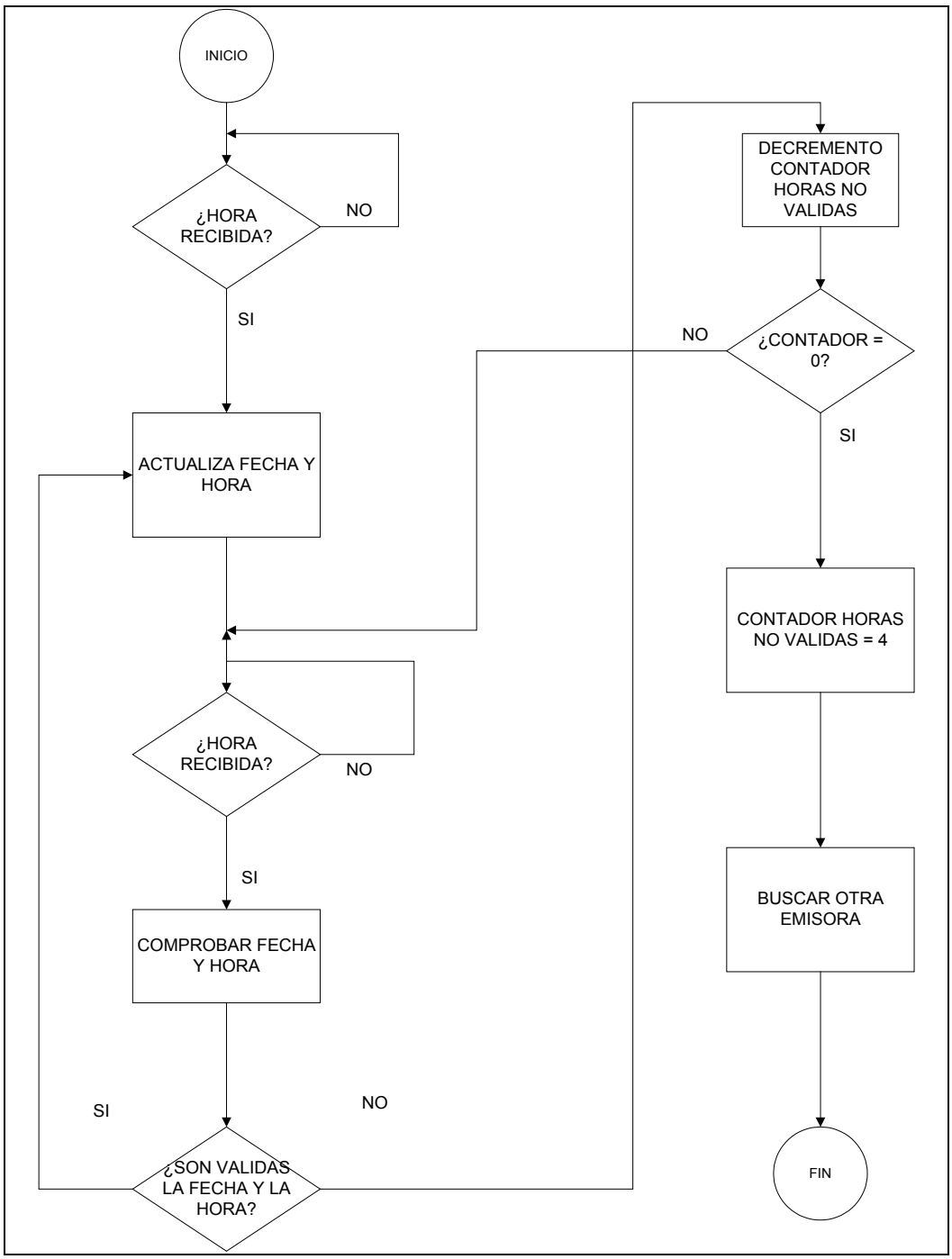


Diagrama de flujo del algoritmo de actualización de la fecha y la hora.

En el algoritmo anterior vemos que la lógica seguida es la siguiente: una vez preparada la fecha y la hora, se comprueba si es la primera vez que se recibe la hora, y si es así se actualiza sin más. Esto es debido a que al recibir por primera vez la fecha y la hora, no tenemos forma de comprobar si es válida ó errónea, por lo que asumiremos que la información es correcta. La siguiente vez que recibamos la información de fecha y hora, deberemos comprobar si es una información válida ó no. Esto lo hacemos mediante una comparación con la información recibida anteriormente de la forma que vemos en el siguiente diagrama de flujo correspondiente al algoritmo de comprobación de fecha y hora, y que explicaremos un poco más adelante.

Si la información recibida es considerada como válida, actualizamos directamente la fecha y la hora, poniendo a cero los segundos. Si no es así, decrementaremos un contador que nos indica el número de veces que la información es errónea, y volvemos a esperar que llegue nueva información. Cuando el contador llegue a cero (se permite un máximo de 4 paquetes erróneos), podremos considerar dos cosas:

- Que la recepción de esa emisora es muy pobre.
- O bien, que la primera hora recibida era errónea y el resto de paquetes eran correctos ó incorrectos.

En cualquiera de los casos, lo que debemos hacer es volver a cargar el contador de paquetes erróneos, considerar que la hora que actualmente tenemos no es correcta, y proceder a la búsqueda de otra emisora dónde la calidad de recepción sea mejor y nos garantice un correcto funcionamiento del sistema.

Una vez encontrada una nueva emisora, el proceso para la actualización de la fecha y la hora vuelve a ser el mismo, desde el principio.

Debido a lo que se acaba de exponer, debemos decir que en condiciones de recepción pobre, es posible que la primera actualización sea incorrecta, mostrando una hora y una fecha errónea en el reloj/calendario. No obstante, en un tiempo mínimo de 4 minutos y máximo de 16 minutos desde su conexión (en condiciones normales), el sistema tendría la información correcta de la fecha y la hora.

A continuación podemos ver el diagrama de flujo correspondiente al algoritmo de comprobación de la fecha y la hora.

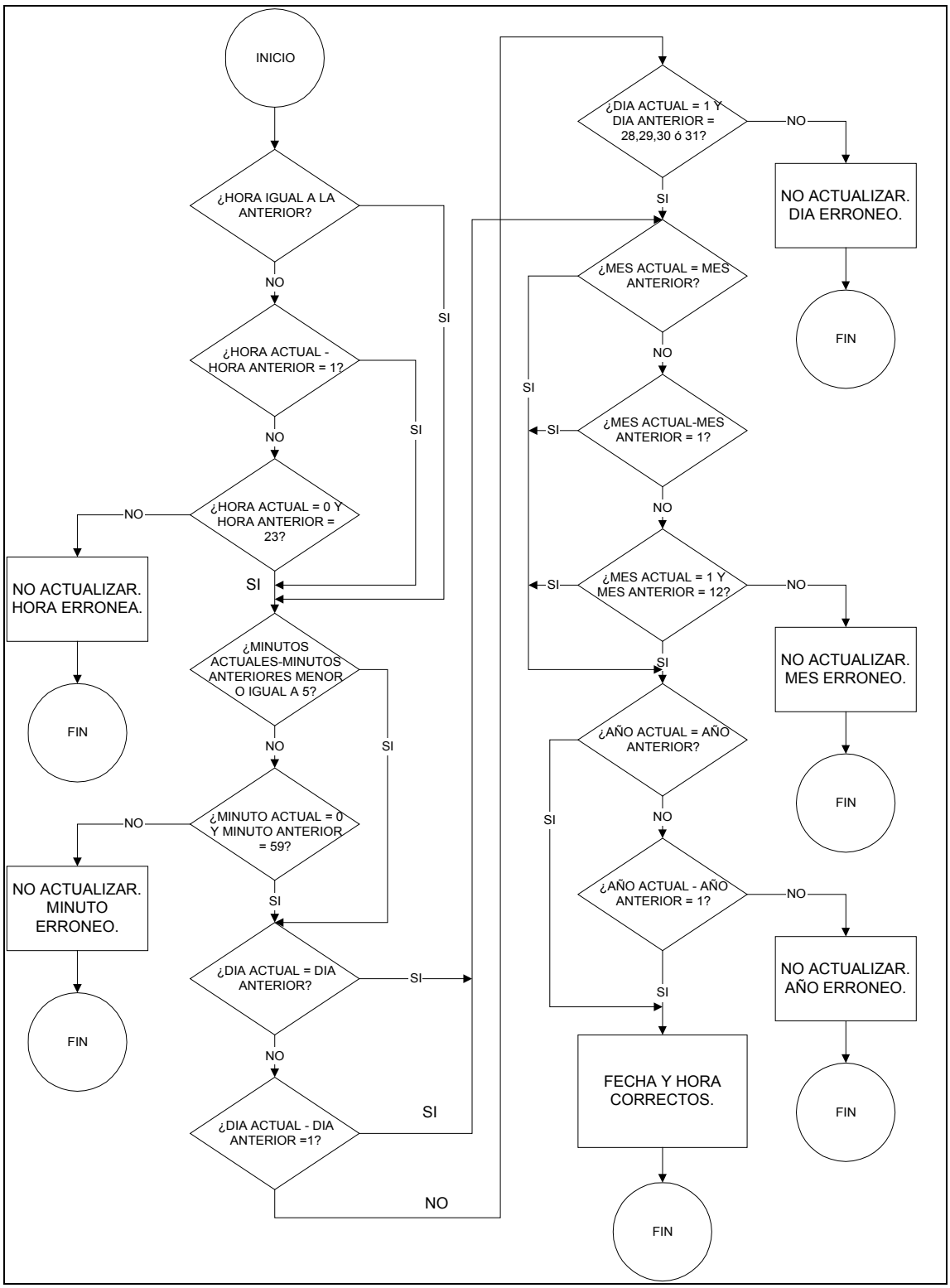


Diagrama de flujo del algoritmo de comprobación de fecha y hora.

Podemos ver que la comprobación de la fecha y la hora se realiza mediante la comparación de dichos datos, con los recibidos anteriormente. Vemos como en el caso de las horas, días, meses y año, se permite una variación de una unidad con respecto de la anteriormente obtenida, para respetar los cambios de hora, día, mes y año. Además se permite un cambio especial para cada uno, siendo en el caso de las horas el cambio de 23 a 0; en el caso de los días, el cambio de 28, 29, 30 ó 31 a 1; y en el caso de los meses, el cambio de 12 a 1. En el caso de los minutos, se permite una variación máxima entre el dato obtenido y el anterior de cinco unidades, por si se ha perdido algún paquete de información entre las dos actualizaciones. Así mismo, se permite el cambio de los minutos de 59 a 0.

En el caso de que alguno de los datos resulte erróneo, consideraremos que el resto de la información puede estar también corrompida, por lo que no se actualizará ningún dato obtenido en ese paquete, así que se esperará a que llegue de nuevo la información de la fecha y la hora.

Además de esta comprobación de la información recibida, el sistema dispone de una medida de seguridad que hace que, en el caso de no recibir información de la fecha y la hora (ni válida ni errónea) en un máximo de 4 minutos, se considere que la recepción es pobre y procederemos a buscar otra emisora.

Con esta lógica de funcionamiento explicada, podemos considerar que **la correcta actualización de la fecha y la hora del sistema** (objeto del proyecto)**, está garantizada, siempre que se disponga de una suficiente calidad de recepción y que las emisoras** (en este caso las de RNE) **emitan correctamente la información.**

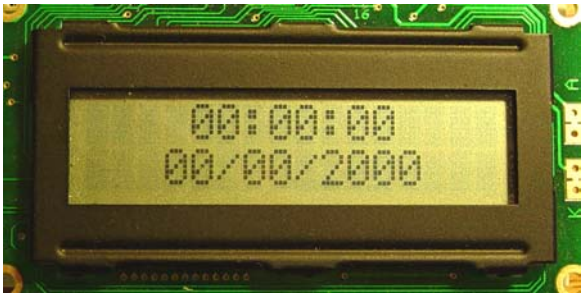
6.3 ACTUALIZACIÓN DE LA FECHA Y LA HORA.

Una vez que disponemos de la información de la fecha y la hora correcta, se procede a la actualización en el RTC (Reloj de Tiempo Real), por el bus I2C. Para ello, primeramente ordenamos los datos y los pasamos a BCD (formato en el que almacena los datos el PCF8583 de Philips). La información relativa al año, se almacena en la posición 0x70 hexadecimal del

PCF8583, que es una de las posiciones de RAM libre de las que dispone. Esto se hace así ya que el RTC no tiene un registro asignado para el año, sino que sólo distingue entre años bisiestos y no bisiestos.

El bus que comunica el microcontrolador con el RTC es el mismo que comunica nuestro circuito con el exterior, por lo que el sistema en el que se encuentre instalado este circuito, solamente debe consultar la información contenida en el RTC, que estará actuando como un dispositivo esclavo.

En la siguientes imágenes podemos ver el LCD del sistema de test desarrollado para comprobar el correcto funcionamiento del sistema. En la primera imagen podemos ver el sistema nada más conectarse. En la imagen siguiente, la información relativa a la fecha y la hora ya ha llegado y se actualiza el sistema.



Detalle del LCD del sistema de test, antes y después de la actualización vía RDS.

7. PRESUPUESTO

7.1 COSTE DEL MÓDULO RECEPTOR DE RADIO FM.

DESCRIPCIÓN	UNIDADES	COSTE UNITARIO EN EUROS (sin iva)	COSTE TOTAL EN EUROS (sin iva)
C.I. TDA 7000 de PHILIPS	1	2,00	2,00
Diodo Varicap BB122	1	0,28	0,28
Conector macho de 5 pines	1	0,16	0,16
Transistor BF 494	1	0,11	0,11
Conversor Digital Analógico MAX5382 de MAXIM	1	1,00	1,00
Resistencias de 0,25 W	5	0,08	0,40
Condensadores cerámicos	17	0,06	1,02
Placa de circuito impreso	1	1,00	1,00
TOTAL			**5,97 €**

7.2 COSTE DEL MÓDULO DECODIFICADOR RDS.

DESCRIPCIÓN	UNIDADES	COSTE UNITARIO EN EUROS (sin iva)	COSTE TOTAL EN EUROS (sin iva)
C.I. SAA6588 de PHILIPS	1	6,00	6,00
PIC 16F628-04/P de MICROCHIP	1	4,00	4,00
C.I. PCF 8583 de PHILIPS	1	2,98	2,98
Cristal de Cuarzo de 4.332 MHz.	1	1,50	1,50
Cristal de Cuarzo de 32.768 KHz	1	0,92	0,92
Inductancia BLM21A102S de MURATA	1	0,12	0,12
Conector Hembra 5 pines	1	0,25	0,25
Resistencias 0.25 W	8	0,08	0,64
Condensadores cerámicos	7	0,06	0,42
Condensadores electrolíticos	2	0,12	0,24
Placa de circuito impreso	1	1,00	1,00
TOTAL			**18,07 €**

7.3 COSTE TOTAL DEL MATERIAL.

DESCRIPCIÓN	COSTE TOTAL EN EUROS
Módulo receptor de radio FM	5,97
Módulo decodificador RDS	18,07
TOTAL	**24,04 €**

7.4 COSTE DE INGENIERÍA.

DESCRIPCIÓN	COSTE UNITARIO EN EUROS (con iva)	Nº UNIDADES	COSTE TOTAL EN EUROS (con iva)
Diseño Hardware	18 € / hora	1,5 meses=240 hr.	4.320
Diseño Software	18 € / hora	1,5 meses=240 hr.	4.320
TOTAL			**8.640 €**

7.5 COSTE DE PRODUCCIÓN.

DESCRIPCIÓN	COSTE UNITARIO	COSTE TOTAL		
		Nº DE UNIDADES PRODUCIDAS		
		100	**1.000**	**10.000**
Material	24,04 € / unidad	2.404	24.040	240.400
Mano de Obra	3 € / unidad	300	3.000	30.000
Ingeniería	8.640 €	8.640	8.640	8.640
COSTE FABRICACIÓN		11.344	35.680	279.040
Beneficio Industrial (20%)		2.268,8	7.136	55.808
COSTE TOTAL (sin iva)		13.612,8	42.816	334.848
I.V.A. (16%)		2.178,05	6.850,56	53.575,68
COSTE TOTAL (iva incluido)		**15.790,85 €**	**49.666,56 €**	**388.423,68 €**
COSTE POR UNIDAD (sin iva)		136,13	42,82	33,48
I.V.A. (16%)		21,78	6,85	5,36
COSTE POR UNIDAD (iva incluido)		**157,91 €**	**49,67 €**	**38,84 €**

APÉNDICE A: CÓDIGO DEL PROGRAMA

1. PROGRAMA PRINCIPAL

```
TITLE  "SISTEMA DECODIFICADOR FM-RDS"
       LIST P=16F628, R=DEC
       ERRORLEVEL -305
       INCLUDE "P16F628.inc"

       __CONFIG _CP_OFF & _WDT_OFF & _INTRC_OSC_CLKOUT & _PWRTE_OFF
       __IDLOCS 2c1ch

;-----------------------------------------------------------
;---------------- Variables en el banco 1. ----------------------
;-----------------------------------------------------------

      VARIABLE FileStart2 = 0xA0
      ORG     FileStart2

HoraAnt        res     1
MinutoAnt      res     1
DiaAnt         res     1
MesAnt         res     1
AñoAnt         res     1
ContNoAct      res     1
WTemporal      res     1
STATUSTemp     res     1
Pasadas1       res     1
Pasadas2       res     1
IntentosI2C    res     1
IntentosI2C2   res     1

;-----------------------------------------------------
;---------------- Variables en el banco 0. -----------------
;-----------------------------------------------------


       VARIABLE FileStart = 20h
       ORG     FileStart

; Variables locales

Minutos        res     1
Horas          res     1
```

```
Dias            res     1
Aux             res     1
Meses           res     1
Año             res     1
Dato1           res     1
Dato2           res     1
Dir             res     1
R0              res     1
R1              res     1
R2              res     1
R3              res     1
R4              res     1
R5              res     1
R6              res     1
AuxPro          res     1       ; Para ayudarnos en el procesado de datos
MJD1             res    1       ; Estos tres bytes están reservados para tener la fecha del
MJD2             res    1       ; calendario Juliano que se representa por 17 bits
MJD3            res     1       ; El LSB del MJD3 pertenece a la HORA
HORA1           res     1       ; En estos dos bytes guardamos la informacion actualizada
HORA2           res     1       ; de la HORA de los MINUTOS y del OFFSET de la hora local.
OFFSET          res     1       ; Aqui almacenaremos el offset para realizar los calculos.
Grupo           res     1       ; Nos servirá para identificar los bloques de los grupos.
Product         res     6
BitCount        res     1
nratorH         res     1
nratorM          res    1
nratorL         res     1
denomH          res     1
denomM          res     1
denomL          res     1
shiftH          res     1
shiftM          res     1
shiftL          res     1
remainH         res     1
remainM         res     1
remainL         res     1
Y1              res     1
Y2              res     1
Y3              res     1
ContadorNORDS1      res     1
ContadorNORDS2      res     1
ContadorNORDS3      res     1
ContadorNOERROR     res     1
ContEspera1     res     1
```

```
ContEspera2   res    1
ContEspera3   res    1
Sintonia      res    1
Detecta       res    1
Errores       res    1
PI1           res    1
PI2           res    1
PI3           res    1
PI4           res    1
RNE           res    1
AuxSint       res    1
MinutosBCD    res    1
HorasBCD      res    1
AñoDiasBCD    res    1
NumBinario    res    1
ContUnidad    res    1
UnidadBCD     res    1
DecenaBCD     res    1


Multiplier EQU Product+3

RestaUno          equ    3
AumentaUno        equ    4
Caracter          equ    32
RDS               equ    0
PRIMERAHORA       equ    1
ACTUALIZA         equ    2
RNEX              equ    3
ENHORA            equ    4
FINBUSQ           equ    5
CAMBIOEMIS        equ    6

        VARIABLE FileStart = $


        ORG 0
        goto    Inicio

;-----------------------------------------------------------------------
;----------------------------- INTERRUPCIONES -------------------------------
;-----------------------------------------------------------------------

        ORG     4
```

```
        bcf     INTCON,GIE
        bcf     INTCON,INTE
        bcf      INTCON,T0IE          ; Inhabilitamos las ;interrupciones globales,las de
                                      ; TMR0 y las de RB0
        bsf     STATUS,RP0            ; Selecciono banco 1.
        movwf WTemporal
        bcf     STATUS,RP0            ; Selecciono banco 0.
        movfw STATUS
        bsf     STATUS,RP0            ; Selecciono banco 1.
        movwf STATUSTemp
        bcf     STATUS,RP0            ; Selecciono banco 0.

        btfss   INTCON,INTF
        goto    Temporizador

        bsf     Detecta,RDS
        call    SecuenciaI2C
        call    Procesa_Datos         ; Mandamos procesar los datos.
        movlw 255
        movwf ContadorNORDS1
        movwf ContadorNORDS2
        movlw 15
        movwf ContadorNORDS3
        bcf     INTCON,INTF
FinalizaInt
        bsf     STATUS,RP0            ; Selecciono banco 1.
        movfw STATUSTemp
        bcf     STATUS,RP0            ; Selecciono banco 0.
        movwf STATUS
        movlw b'10110000'
        movwf INTCON                   ; Habilitamos las interrupciones globales, las de
                                      ;TMR0 y las de RB0.
        bsf     STATUS,RP0            ; Selecciono banco 1.
        movfw WTemporal
        bcf     STATUS,RP0            ; Selecciono banco 0.
        retfie

Temporizador
        bsf     STATUS,RP0            ; Selecciono banco 1.
        decfsz Pasadas1
        goto    Seguir1
        movlw 255
        movwf Pasadas1
        decfsz Pasadas2
```

```
        goto    Seguir1
        movlw 16
        movwf Pasadas2
        bcf     STATUS,RP0          ; Selecciono banco 0.
        bsf     Detecta,CAMBIOEMIS
Seguir1
        bcf     STATUS,RP0          ; Selecciono banco 0.
        bcf     STATUS,2
        movlw 1
        movwf TMR0
        bcf     INTCON,T0IF
        goto    FinalizaInt

;-------------------------- FINAL DE INTERRUPCIONES -------------------------


;------------------------------------------------------------------------------
;----------- INICIALIZACION DEL PROGRAMA Y LAS VARIABLES -----------------------
;------------------------------------------------------------------------------

Inicio
        movlw b'00000111'
        movwf CMCON                 ; Deshabilitamos el módulo comparador de PORTA.

        movlw b'10010000'
        movwf INTCON                ; Habilitamos las interrupciones globales y las de RB0.
        bsf     STATUS,RP0          ; Selecciono banco 1.
        bsf     TRISB,0             ; La patilla RB0 es una entrada.

        bcf     OPTION_REG,INTEDG   ; Las interrupciones en RB0 se producirán por flanco de
                                    ; bajada.
        bcf     OPTION_REG,T0CS
        bcf     OPTION_REG,PSA      ; Asignamos la pre-escala al módulo TMR0.
        bsf     OPTION_REG,PS0
        bsf     OPTION_REG,PS1
        bsf     OPTION_REG,PS2      ; Hemos seleccionado la pre-escala de 1:256.
        clrf    HoraAnt
        clrf    MinutoAnt
        clrf    MesAnt
        clrf    DiaAnt
        clrf    AñoAnt
        movlw 4
        movwf ContNoAct
        bcf     STATUS,RP0          ; Vuelvo al banco 0.
```

```
        clrf    Grupo               ; Ponemos a cero las variables
        clrf    Dias
        clrf    Horas
        clrf    Minutos
        clrf    Meses
        clrf    Año
        clrf    Detecta
        movlw   255
        movwf   ContadorNORDS1
        movwf   ContadorNORDS2
        movlw   15
        movwf   ContadorNORDS3
        clrf    Errores
        clrf    Sintonia
        clrf    AuxSint

        goto    EmisoraArriba

;-----------------------------------------------------------------------------

FijaEmisora
        bsf     Detecta,FINBUSQ     ; Nos dice que ya hemos terminado la búsqueda y se
                                    ; pueden actualizar la fecha y la hora.
        clrf    Errores
        call    HabilitaInt
        bsf     INTCON,5            ; Habilitamos las interrupciones de TMR0
        bsf     STATUS,RP0          ; Selecciono banco 1.
        movlw   255
        movwf   Pasadas1
        movlw   16
        movwf   Pasadas2
        bcf     STATUS,RP0          ; Selecciono banco 0.
        movlw   1
        movwf   TMR0

Bucle_principal
        btfsc   Errores,5
        goto    EmisoraArriba
        bcf     STATUS,RP0          ; Selecciono banco 0.
        btfsc   Detecta,CAMBIOEMIS
        goto    EmisoraArriba
        decfsz  ContadorNORDS1      ; Estos contadores nos darán unos segundos antes de
                                    ; comprobar que no hay señal RDS.
```

```
        goto    Bucle_principal
        decfsz  ContadorNORDS2
        goto    SigueContador
        decfsz  ContadorNORDS3
        goto    SigueContador2
        goto    EmisoraArriba
SigueContador
        movlw   255
        movwf   ContadorNORDS1
        goto    Bucle_principal
SigueContador2
        movlw   255
        movwf   ContadorNORDS2
        goto    Bucle_principal


;-------------------------------------------------------------------------------
;-------------------- Aqui se realiza la búsqueda de las emisoras -------------------
;-------------------------------------------------------------------------------

EmisoraArriba
        bcf     Detecta,RDS
        bcf     Detecta,FINBUSQ
        bcf     Detecta,PRIMERAHORA
        bcf     Detecta,CAMBIOEMIS
        bcf     Detecta,ENHORA
        clrf    INTCON          ; Inhabilitamos las interrupciones globales y las de RB0
        call    Subir
        bcf     Detecta,RDS
        call    HabilitaInt
        call    Espera
        call    Espera
        clrf    INTCON          ; Inhabilitamos las interrupciones globales y las de RB0
        btfss   Detecta,RDS
        goto    EmisoraArriba
        bcf     Detecta,RDS
        bcf     Detecta,RNEX
        call    IdentificaEmisora
        btfss   Detecta,RNEX
        goto    EmisoraArriba
        goto    FijaEmisora
;-------------------------------------------------------------------------------
HabilitaInt:
        bsf     INTCON,GIE
```

```
        bsf     INTCON,INTE         ; Habilitamos las interrupciones globales y las de RB0.
        return

;-------------------------------------------------------------------------------
;-- Aqui identificamos la emisora que hemos sintonizado y seleccionamos la que queremos.
;-------------------------------------------------------------------------------

IdentificaEmisora:

        movfw PI4
        andlw b'00001111'
        bcf   STATUS,Z
        xorlw 14                    ; Miramos si es una emisora española.
        btfss STATUS,Z
        goto  Continua              ; Si la emisora no es española, continuamos la
búsqueda.

        decfsz PI2
        goto  Continua              ; Si la emisora no pertenece a RNE, continuamos la
                                    ; búsqueda.
        movfw PI1
        bcf   STATUS,Z
        xorlw 1
        btfsc STATUS,Z
        goto  RNE1

        movfw PI1
        bcf   STATUS,Z
        xorlw 3
        btfsc STATUS,Z
        goto  RNE1

        movfw PI1
        bcf   STATUS,Z
        xorlw 2
        btfsc STATUS,Z
        goto  RNE1

        movfw PI1
        bcf   STATUS,Z
        xorlw 5
        btfsc STATUS,Z
        goto  RNE1
```

```
        goto    Continua

RNE1
        clrf    Errores
        call    HabilitaInt     ; Habilitamos las interrupciones para ver el numero de errores
                                ; en la recepción.
        call    Espera
        call    Espera
        call    Espera
        call    Espera
        call    Espera
        call    Espera
        call    Espera
        bcf     STATUS,Z
        movfw   Errores
        iorlw   0
        btfss   STATUS,Z
        goto    Continua        ; Si la recepción tiene errores, no almacenamos su posición y
                                ; continuamos la búsqueda.

        bsf     Detecta,RNEX

Continua
        clrf    INTCON          ; Inhabilitamos las interrupciones globales, las de TMR0 y las de
                                ; RB0
        return

;-----------------------------------------------------------------------------
;------- Aqui realizamos la operación de aumentar en una posición el sintonizador.-------
;-----------------------------------------------------------------------------

Subir:
        incf    Sintonia,F
        bsf     STATUS,RP0
        movlw   10
        movwf   IntentosI2C
        bcf     STATUS,RP0
Start4
        clrf    Indica_Error
        movlw   0
        movwf   Dir
        call    I2C_Start
```

```
        btfsc   Indica_Error,0          ; Comprobamos si se ha producido un error al poner la
                                        ; condicion de  Start.
        goto    Error_Start3

        movlw b'01100000'   ; En I2C_Data ponemos la direccion del dispositivo esclavo al
                            ; que nos dirigimos, en este caso al DAC MAX5382LEUK.
        movwf I2CData
        bcf     I2CData,0   ; Especificamos que el modo SHUTDOWN este desactivado.
        call    I2C_Write
        btfsc   Indica_Error,1 ; Comprobamos si se ha producido un error al escribir la
                            ; direccion del dispositivo esclavo.
        goto    Error_Escritura4

        movfw Sintonia
        movwf I2CData
        call    I2C_Write
        btfsc   Indica_Error,1 ; Comprobamos si se ha producido un error al escribir en
                            ; el dispositivo esclavo.
        goto    Error_Escritura4
        call    I2C_Stop

        bcf     INTCON,INTF ; Ponemos a cero el flag de interrupción de RB0.
        return

Error_Start3
        call    Espera      ; Esperamos unos instantes por si el bus estuviera ocupado
        bsf     STATUS,RP0  ; Seleccionamos el banco 1.
        decfsz IntentosI2C  ; Decrementamos el contador de intentos.
        goto    Start4
        bcf     STATUS,RP0
        return

Error_Escritura4
        call    I2C_Stop
        bsf     STATUS,RP0  ; Seleccionamos el banco 1.
        decfsz IntentosI2C  ; Decrementamos el contador de intentos.
        goto    Start4
        bcf     STATUS,RP0

        return
```

```
;-------------------------------------------------------------------------------
;-------------------- Esta es una subrutina de espera de unos 50 ms. ----------------
;-------------------------------------------------------------------------------
Espera:

CargaContador2
        movlw 255
        movwf ContEspera2
CargaContador1
        movlw 255
        movwf ContEspera1
SigueEsperando
        decfsz ContEspera1
        goto   SigueEsperando
        decfsz ContEspera2
        goto   CargaContador1
        return

;-------------------------------------------------------------------------------
;----- AQUI PREPARAMOS Y ENVIAMOS LA FECHA Y LA HORA ACTUALIZADAS ------------
;---------------------------- AL PCF 8583 VIA BUS I2C. ----------------------------
;-------------------------------------------------------------------------------

EnviaRTC:
        clrf   INTCON        ; Inhabilitamos las interrupciones.
        bsf    STATUS,RP0
        movlw 250
        movwf IntentosI2C2
        movlw 40
        movwf IntentosI2C
        bcf    STATUS,RP0
        movlw 0
        movwf Dir
        goto   Start3

        INCLUDE "i2cb.asm"

Start3
        bcf    STATUS,RP0
        clrf   Indica_Error2
        call   I2C_Start2
        btfsc  Indica_Error2,0       ; Comprobamos si se ha producido un error al poner la
                                     ; condicion de Start.
        goto   Error_Start2
```

```
movlw b'10100000'   ; En I2C_Data ponemos la direccion del dispositivo PCF 8583
movwf I2CData2
bcf   I2CData2,0    ; Especificamos que se va a realizar una escritura.
call  I2C_Write2
btfsc Indica_Error2,1       ; Comprobamos si se ha producido un error al escribir la
                            ; direccion del dispositivo esclavo.
goto  Error_Escritura2

movlw b'00000000'   ; Dirección del registro STATUS, donde empezamos la escritura.
movwf I2CData2
call  I2C_Write2
btfsc Indica_Error2,1       ; Comprobamos si se ha producido un error al escribir
                            ; en el  dispositivo esclavo.
goto  Error_Escritura2

movlw b'10000000'           ; En el registro STATUS especificamos que se pare la
                            ; cuenta para poder actualizar los demás registros.
movwf I2CData2
call  I2C_Write2
btfsc Indica_Error2,1       ; Comprobamos si se ha producido un error al escribir
                            ; en el  dispositivo esclavo.
goto  Error_Escritura2

clrf  I2CData2      ; Ponemos a cero el contador de centésimas de segundo.
call  I2C_Write2
btfsc Indica_Error2,1       ; Comprobamos si se ha producido un error al escribir
                            ; en el  dispositivo esclavo.
goto  Error_Escritura2

clrf  I2CData2              ; Ponemos a cero el contador de los segundos.
call  I2C_Write2
btfsc Indica_Error2,1       ; Comprobamos si se ha producido un error al escribir
                            ; en el  dispositivo esclavo.
goto  Error_Escritura2

movfw Minutos
movwf NumBinario
call  Bin_a_BCD
btfsc DecenaBCD,0           ; Colocamos las decenas de minuto y las  unidades de
                            ; minuto en su sitio.
bsf   UnidadBCD,4
btfsc DecenaBCD,1
bsf   UnidadBCD,5
btfsc DecenaBCD,2
```

```
        bsf     UnidadBCD,6
        btfsc   DecenaBCD,3
        bsf     UnidadBCD,7
        movfw UnidadBCD
        movwf I2CData2
        call    I2C_Write2      ; Aqui ya hemos enviado los minutos en formato BCD al RTC.
        btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                        ; en el  dispositivo esclavo.
        goto    Error_Escritura2

        movfw Horas             ; Mandamos las horas actualizadas con los bits de
                                ; configuración (AM/PM y 24h/12h).

        movwf NumBinario
        call    Bin_a_BCD
        btfsc   DecenaBCD,0             ; Colocamos las decenas de hora y las  unidades de hora
                                        ; en su sitio.
        bsf     UnidadBCD,4
        btfsc   DecenaBCD,1
        bsf     UnidadBCD,5
        bcf     UnidadBCD,7             ; Seleccionamos el modo de 24 h.
        movfw UnidadBCD
        movwf I2CData2
        call    I2C_Write2              ; Aqui ya hemos enviado las horas en formato BCD al
                                        ; RTC.
        btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                        ; en el  dispositivo esclavo.
        goto    Error_Escritura2

        movfw Dias                      ; Mandamos los días y el año actualizado.
        movwf NumBinario
        call    Bin_a_BCD
        btfsc   DecenaBCD,0             ; Colocamos las decenas de días y las  unidades de días
                                        ; en su sitio.
        bsf     UnidadBCD,4
        btfsc   DecenaBCD,1
        bsf     UnidadBCD,5
        btfss   Año,0
        btfsc   Año,1
        goto    NoBisiesto
        goto    Bisiesto
NoBisiesto
        bsf     UnidadBCD,6
        goto    SigueAño
```

```
Bisiesto
        bcf     UnidadBCD,6
        bcf     UnidadBCD,7
SigueAño
        movfw UnidadBCD
        movwf I2CData2
        call    I2C_Write2
        btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                        ; en el  dispositivo esclavo.
        goto    Error_Escritura2

        movfw Meses                     ; Mandamos el mes y el día de la semana actualizado.
        movwf NumBinario
        call    Bin_a_BCD
        btfsc   DecenaBCD,0             ; Colocamos las decenas de meses y las  unidades de
                                        ; meses en su sitio.
        bsf     UnidadBCD,4
        btfsc   DecenaBCD,1
        bsf     UnidadBCD,4
        movfw UnidadBCD
        movwf I2CData2
        call    I2C_Write2
        btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                        ; en el  dispositivo esclavo.
        goto    Error_Escritura2

        call    I2C_Stop2

        bcf     STATUS,RP0
        call    I2C_Start2
        btfsc   Indica_Error2,0         ; Comprobamos si se ha producido un error al poner la
                                        ; condición de Start.
        goto    Error_Start2
        movlw b'10100000'   ; En I2C_Data ponemos la dirección del dispositivo PCF 8583
        movwf I2CData2
        bcf     I2CData2,0      ; Especificamos que se va a realizar una escritura.
        call    I2C_Write2
        btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir la
                                        ; dirección del dispositivo esclavo.
        goto    Error_Escritura2

        movlw 0x70                      ; Dirección del registro de la RAM, donde vamos a
                                        ; almacenar el año.
        movwf I2CData2
```

```
call    I2C_Write2
btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                ; en el  dispositivo esclavo.
goto    Error_Escritura2

movfw Año                        ; Mandamos el año actualizado.
movwf NumBinario
call    Bin_a_BCD
btfsc   DecenaBCD,0             ; Colocamos las decenas de año y las  unidades de año
                                ; en su sitio.
bsf     UnidadBCD,4
btfsc   DecenaBCD,1
bsf     UnidadBCD,4
btfsc   DecenaBCD,2
bsf     UnidadBCD,6
btfsc   DecenaBCD,3
bsf     UnidadBCD,7
movfw UnidadBCD
movwf I2CData2
call    I2C_Write2              ; Aqui ya hemos enviado las dos ultimas cifras del año
                                ; en formato BCD al RTC.
btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                ; en el  dispositivo esclavo.
goto    Error_Escritura2

call    I2C_Stop2

bcf     STATUS,RP0
call    I2C_Start2              ; Vamos a indicar que vuelva a realizarse la cuenta en el
                                ; RTC.
btfsc   Indica_Error2,0         ; Comprobamos si se ha producido un error al poner la
                                ; condición de Start.
goto    Error_Start2
movlw b'10100000'   ; En I2C_Data ponemos la dirección del dispositivo PCF 8583
movwf I2CData2
bcf     I2CData2,0              ; Especificamos que se va a realizar una escritura.
call    I2C_Write2
btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                ; en el  dispositivo esclavo.
goto    Error_Escritura2

movlw b'00000000'               ; Dirección del registro STATUS, donde empezamos la
                                ; escritura.
movwf I2CData2
```

```
        call    I2C_Write2
        btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                        ; en el  dispositivo esclavo.
        goto    Error_Escritura2

        movlw b'00000000'   ; En el registro STATUS especificamos que continúe la cuenta.
        movwf I2CData2
        call    I2C_Write2
        btfsc   Indica_Error2,1         ; Comprobamos si se ha producido un error al escribir
                                        ; en el  dispositivo esclavo.
        goto    Error_Escritura2

        call    I2C_Stop2

        SDA_High2
        SCL_High2

        call    HabilitaInt
        bsf     INTCON,5        ; Habilitamos las interrupciones de TMR0

        return

Error_Start2
        nop                     ; Esperamos unos instantes por si el bus estuviera ocupado
        nop
        nop
        nop
        nop
        bsf     STATUS,RP0      ; Seleccionamos el banco 1.
        decfsz IntentosI2C2     ; Decrementamos el contador de intentos.
        goto    Start3
        decfsz IntentosI2C
        goto    CargaContador
        bcf     STATUS,RP0
        clrf    Indica_Error2
        call    HabilitaInt
        bsf     INTCON,5        ; Habilitamos las interrupciones de TMR0
        SDA_High2
        SCL_High2
        return

CargaContador
        movlw 250
        movwf IntentosI2C2
```

```
        goto    Start3

Error_Escritura2
        call    I2C_Stop2
        bsf     STATUS,RP0      ; Seleccionamos el banco 1.
        decfsz  IntentosI2C     ; Decrementamos el contador de intentos.
        goto    Start3
        bcf     STATUS,RP0
        clrf    Indica_Error2
        call    HabilitaInt
        bsf     INTCON,5        ; Habilitamos las interrupciones de TMR0
        SDA_High2
        SCL_High2
        return


;-------------------------------------------------------------------------------
;------------------------ Aqui recibimos los datos vía bus I2C ------------------------
;-------------------------------------------------------------------------------

SecuenciaI2C:

        movlw 0
        movwf Dir
        goto Start

        INCLUDE "i2c.asm"

Start

        SDA_High
        SCL_High

        call    I2C_Start       ; Ponemos la condición de inicio en el bus.
        movlw   b'00100001'
        movwf   I2CData
        bsf     I2CData,0       ; Especificamos que se va a realizar una lectura.
        call    I2C_Write       ; Escribimos en el bus la dirección del dispositivo esclavo al que
                                ; llamamos.
        call    I2C_Read        ; Comenzamos a hacer la lectura de los datos que nos están
                                ; enviando.
        movfw   I2CData
        movwf   R0              ; Ya tenemos en R0 el primer byte.
```

```
        call    I2C_Read
        movfw I2CData
        movwf R1            ; Ya tenemos en R1 el segundo byte.

        call    I2C_Read
        movfw I2CData
        movwf R2            ; Ya tenemos en R2 el tercer byte.

        call    I2C_Read
        movfw I2CData
        movwf R3            ; Ya tenemos en R3 el cuarto byte.

        call    I2C_Read
        movfw I2CData
        movwf R4            ; Ya tenemos en R4 el quinto byte.

        call    I2C_Read
        movfw I2CData
        movwf R5            ; Ya tenemos en R5 el sexto byte.

        call    I2C_Read_NAK        ; Leemos el ultimo byte y mandamos el No_ACK al
                                    ; esclavo.
        movfw I2CData
        movwf R6            ; Ya tenemos en R6 el ultimo byte.

        call I2C_Stop       ; Finalizamos la transmisión de datos por el bus.

        return


;-------------------------------------------------------------------------------
;-------------- PROCESAMOS LOS DATOS DEL PRE-PROCESADOR SAA6588 -------------
;-------------------------------------------------------------------------------
;       Aqui tenemos la subrutina para trabajar en el MODO ESTANDAR DE PROCESADO.
;       Esto implica que la información del pre-procesador se actualiza con cada
;       nuevo bloque procesado. En este programa solo recogeremos y procesaremos
;       la información RDS correspondiente a la hora, la fecha y el PI de la emisora.
;-------------------------------------------------------------------------------

Procesa_Datos:

        bcf     STATUS,RP0  ; Selecciono banco 0.
        btfsc   R0,0        ; Primero comprobamos que los bloques no tienen errores.
        goto    Salida
```

```
        btfsc   R0,1
        goto    Salida

        movfw R5
        movwf Errores           ; En "Errores" tengo el número de errores de bloque contados.
        rrf     Errores,F
        rrf     Errores,F
        bcf     Errores,7
        bcf     Errores,6

        clrf    AuxPro
        movfw R0
        movwf AuxPro
        rrf     AuxPro,F
        rrf     AuxPro,F
        rrf     AuxPro,F
        rrf     AuxPro,F
        rrf     AuxPro,F
        movlw b'00000111'   ; Mascara para quedarme solo con los 3 bits mas bajos.
        andwf  AuxPro,F
        movfw AuxPro
        bcf     STATUS,Z
        andlw  b'11111111'  ; Comprobamos si es un bloque A.
        btfsc   STATUS,Z
        goto    BloqueA
        decfsz AuxPro
        goto    NoBloqueB
        goto    BloqueB

BloqueA
        movfw R1
        andlw  b'11110000'
        movwf PI4
        rrf     PI4
        rrf     PI4
        rrf     PI4
        rrf     PI4
        bcf     PI4,4           ; Por si acaso había algo en el bit de acarreo al hacer la
                                ; rotación.
        movfw R1
        andlw  b'00001111'
        movwf PI3

        movfw R2
```

```
        andlw  b'11110000'
        movwf  PI2
        rrf    PI2
        rrf    PI2
        rrf    PI2
        rrf    PI2
        bcf    PI2,4          ; Por si acaso había algo en el bit de acarreo al hacer la
                              ; rotación.
        movfw  R2
        andlw  b'00001111'
        movwf  PI1

        goto   Salida


BloqueB
        movfw  R1
        movwf  AuxPro
        movlw  b'11111000'    ; Mascara para quedarme con los 5 bits superiores.
        andwf  AuxPro,F
        bcf    STATUS,Z       ; Lo ponemos a cero para comprobarlo mas adelante.
        movlw  b'01000000'
        subwf  AuxPro,F
        btfsc  STATUS,Z       ; Comprobamos si el resultado de la resta ha sido 0.
        goto   Grupo4A
        goto   Salida

Grupo4A
        movfw  R2
        movwf  AuxPro
        movlw  b'00000011'    ; Nos quedamos con los dos bits mas bajos que seran los
        andwf  AuxPro,F       ; dos bits mas altos de la FECHA compuesta de 17 bis.
        movfw  AuxPro
        movwf  MJD1           ; Aqui guardamos los dos bits antes citados. El resto del Byte
                              ; no se usa de momento.
        bsf    MJD1,4         ; Aqui ya usaremos el bit 4 de esta palabra (MJD1), para indicar
                              ; que el ultimo bloque procesado fue un bloque B del grupo 4A
                              ; y que por lo tanto, los dos bloques siguientes a procesar
                              ; tienen la informacion que queremos (CLOCK TIME).
        goto   Salida

NoBloqueB
        btfss  MJD1,4         ; Miramos si el ultimo bloque era un bloque B del grupo 4A.
        goto   Salida
```

```
        btfss   MJD1,5
        goto    BloqueC
        goto    BloqueD

BloqueC
        bsf     MJD1,5          ; Indicamos que estamos en un bloque C y que el siguiente sera
                                ; un bloque D.
        movfw R1
        movwf MJD2              ; Guardamos los datos del calendario Juliano.
        movfw R2
        movwf MJD3              ; El LSB pertenece a la HORA.
        goto    Salida

BloqueD
        bcf     MJD1,4
        bcf     MJD1,5          ; Ponemos a cero los indicadores de bloque.
        movfw R1
        movwf HORA1             ; Guardamos los datos de la hora, minutos y offset de la hora
                                ; local.
        movfw R2
        movwf HORA2
        goto    OrdenaDatos


OrdenaDatos
        btfsc   R0,0            ; Esto lo hacemos para que en el caso de que el bloque
                                ; procesado tenga algún error, no se actualice
        goto    Salida          ; la información de la fecha y la hora.
        btfsc   R0,1
        goto    Salida

        movfw HORA1             ; Primero preparamos los minutos.
        movwf Minutos
        rlf     Minutos,F
        rlf     Minutos,F
        movlw b'00111100'
        andwf  Minutos,F
        movfw HORA2
        movwf AuxPro
        rrf     AuxPro,F
        rrf     AuxPro,F
        rrf     AuxPro,F
        rrf     AuxPro,F
        rrf     AuxPro,F
```

```
        rrf     AuxPro,F
        movlw   b'00000011'
        andwf   AuxPro,F
        movfw   AuxPro
        addwf   Minutos,F       ; Aqui ya tenemos los minutos preparados.

        movlw   59
        movwf   ContadorNOERROR     ; Nos servirá para evitar que la hora sea mayor de 23 y
                                    ; los minutos mayor de 59.
        movfw   Minutos
        movwf   AuxPro          ; Podemos usar esta variable ya que ahora está sin uso en esta
                                ; parte del programa.
        bcf     STATUS,Z        ; Ponemos a 0 el flag de cero.
        andlw   255             ; Comprobamos si el numero recibido es cero (cambio de hora).
        btfsc   STATUS,Z
        goto    CorrectoM

BucleComprobacionM
        decfsz  AuxPro
        goto    SigueComprobandoM
        goto    CorrectoM
SigueComprobandoM
        decfsz  ContadorNOERROR
        goto    BucleComprobacionM
        goto    NoCorrectoM             ; Si los mintos son mas de 59, no se actualizan.

NoCorrectoM
        bsf     STATUS,RP0              ; Selecciono banco 1.
        decfsz  ContNoAct               ; Decrementamos el contador de no actualización.
        goto    Salida
        bcf     STATUS,RP0              ; Selecciono banco 0.
        bsf     Detecta,CAMBIOEMIS      ; Indicamos que se debe cambiar la emisora por
                                        ; recepción  pobre.
        bsf     STATUS,RP0              ; Selecciono banco 1.
        movlw   4
        movwf   ContNoAct
        goto    Salida

CorrectoM
        movfw   HORA1                   ; Ahora vamos con la hora.
        movwf   Horas
        rrf     Horas,F
        rrf     Horas,F
        rrf     Horas,F
```

```
        rrf     Horas,F
        movlw   b'00001111'
        andwf   Horas,F
        btfss   MJD3,0
        goto    Primer_Bit_0
        bsf     Horas,4
        goto    Sigue_Hora

Primer_Bit_0
        bcf     Horas,4

Sigue_Hora
        movfw   HORA2                   ; Vamos a sumar el offset a la hora.
        movwf   OFFSET
        movlw   b'00111111'
        andwf   OFFSET,F
        btfss   OFFSET,5                ; Comprobamos si hay que sumar o restar.
        goto    Suma
        goto    Resta

Suma
        bcf     OFFSET,5        ; Lo ponemos a cero porque solo nos indicaba si había que
                                ; sumar o restar.
        rrf     OFFSET,F        ; Porque el offset viene dado en múltiplos de medias horas.
        bcf     OFFSET,7        ; Por si al rotar se había puesto a 1.
        movfw   OFFSET
        addwf   Horas
        goto    Continuamos

Resta
        bcf     OFFSET,5        ; Lo ponemos a cero porque solo nos indicaba si había que
                                ; sumar o restar.
        rrf     OFFSET,F        ; Porque el offset viene dado en múltiplos de medias horas.
        bcf     OFFSET,7        ; Por si al rotar se había puesto a 1.
        movfw   OFFSET
        subwf   Horas

Continuamos                     ; Aqui ya tenemos la hora preparada.
        movfw   Horas
        movwf   AuxPro          ; Podemos usar esta variable ya que ahora está sin uso en esta
                                ; parte del programa.
        movlw   23
        movwf   ContadorNOERROR         ; Nos servirá para evitar que la hora sea mayor de 23 y
                                        ; los minutos mayor de 59.
```

```
BucleComprobacionH
        decfsz AuxPro
        goto   SigueComprobandoH
        goto   CorrectoH

SigueComprobandoH
        decfsz ContadorNOERROR
        goto   BucleComprobacionH
        goto   NoCorrectoH          ; Si la hora es superior a 23, no se actualiza.

CorrectoH
        call   MJDaCALENDARIO
        clrf   MJD1
        clrf   MJD2
        clrf   MJD3
        btfsc  Detecta,RNEX
        call   Comprobacion
        bcf    STATUS,RP0           ; Selecciono banco 0.
        btfss  Detecta,FINBUSQ      ; Comprobamos que la busqueda de la emisora ha
                                    ; terminado.
        goto   Salida
        btfsc  Detecta,ACTUALIZA
        call   EnviaRTC
        goto   Salida

NoCorrectoH
        bsf    STATUS,RP0           ; Selecciono banco 1.
        decfsz ContNoAct            ; Decrementamos el contador de no actualización.
        goto   Salida
        bcf    STATUS,RP0           ; Selecciono banco 0.
        bsf    Detecta,CAMBIOEMIS   ; Indicamos que se debe cambiar la emisora por
                                    ; recepción pobre.
        bsf    STATUS,RP0           ; Selecciono banco 1.
        movlw  4
        movwf  ContNoAct
        goto   Salida

Salida
        bcf    STATUS,RP0           ; Selecciono banco 0.
        return
```

```
;-------------------------------------------------------------------------------
;------- Aqui se realiza la comprobación de que la fecha y la hora son válidas. ----------
;-------------------------------------------------------------------------------

Comprobacion:
        btfss   Detecta,PRIMERAHORA     ; Miramos a ver si es la primera hora que se
                                        ; recibe.
        goto    Actualizar
        goto    Comprueba

Comprueba
        movfw Horas
        movwf AuxPro
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movfw HoraAnt
        bcf     STATUS,RP0      ; Selecciono banco 0.
        bcf     STATUS,Z
        subwf   AuxPro,f        ; Comprobamos si la hora anterior y la actual son iguales.
        btfsc   STATUS,Z
        goto    HoraCorrecta
        decfsz AuxPro           ; De esta manera aceptamos una diferencia de una unidad entre
                                ; la hora anterior y la actual.
        goto    SigueComprobacionH
        goto    HoraCorrecta

SigueComprobacionH              ; Comprobamos si el cambio de hora es de las 23 a las 0.
        movfw Horas
        bcf     STATUS,Z
        iorlw   0               ; Miramos si la hora actual son las 0.
        btfss   STATUS,Z
        goto    NoActualizar
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movfw HoraAnt           ; Miramos si la hora anterior era las 23.
        bcf     STATUS,RP0      ; Selecciono banco 0.
        bcf     STATUS,Z
        xorlw   23
        btfss   STATUS,Z
        goto    NoActualizar

HoraCorrecta                    ; Aqui ya sabemos que la hora es correcta y comprobamos los
                                ; minutos.
        movfw Minutos
        movwf AuxPro
        bsf     STATUS,RP0      ; Selecciono banco 1.
```

```
        movfw MinutoAnt
        bcf   STATUS,RP0   ; Selecciono banco 0.
        bcf   STATUS,Z
        subwf AuxPro,f     ; Comprobamos si los minutos anteriores y los actuales son
                           ; iguales.
        btfsc STATUS,Z
        goto  MinutoCorrecto
        movlw 6
        movwf Aux

BucleCompruebaMin
        decfsz Aux
        goto   CompruebaMin
        goto   SigueComprobacionMin

CompruebaMin
        decfsz AuxPro      ; De esta manera aceptamos una diferencia de cinco unidades
                           ; entre los  minutos anteriores y los actuales.
        goto   BucleCompruebaMin
        goto   MinutoCorrecto

SigueComprobacionMin       ; Comprobamos si el cambio de minutos es de 59 a 0.
        movfw Minutos
        bcf   STATUS,Z
        iorlw 0            ; Miramos si los minutos actuales son 0.
        btfss STATUS,Z
        goto  NoActualizar
        bsf   STATUS,RP0   ; Selecciono banco 1.
        movfw MinutoAnt    ; Miramos si el minuto anterior era 59.
        bcf   STATUS,RP0   ; Selecciono banco 0.
        bcf   STATUS,Z
        xorlw 59
        btfss STATUS,Z
        goto  NoActualizar

MinutoCorrecto             ; Aqui ya sabemos que los minutos son correctos y
                           ; comprobamos los días.
        movfw Dias
        movwf AuxPro
        bsf   STATUS,RP0   ; Selecciono banco 1.
        movfw DiaAnt
        bcf   STATUS,RP0   ; Selecciono banco 0.
        bcf   STATUS,Z
        subwf AuxPro,f     ; Comprobamos si los dias anteriores y los actuales son iguales.
```

```
        btfsc   STATUS,Z
        goto    DiaCorrecto
        decfsz  AuxPro          ; De esta manera aceptamos una diferencia de una unidad entre
                                ; el día anterior y el actual.
        goto    SigueComprobacion
        goto    DiaCorrecto

SigueComprobacion               ; Comprobamos si el cambio de hora es de las 23 a las 0.
        movfw   Dias
        bcf     STATUS,Z
        xorlw   1               ; Miramos si el día actual es el 1.
        btfss   STATUS,Z
        goto    NoActualizar
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movfw   DiaAnt          ; Miramos si el día anterior era 31.
        bcf     STATUS,RP0      ; Selecciono banco 0.
        bcf     STATUS,Z
        xorlw   31
        btfsc   STATUS,Z
        goto    DiaCorrecto
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movfw   DiaAnt          ; Miramos si el día anterior era 30.
        bcf     STATUS,RP0      ; Selecciono banco 0.
        bcf     STATUS,Z
        xorlw   30
        btfsc   STATUS,Z
        goto    DiaCorrecto
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movfw   DiaAnt          ; Miramos si el día anterior era 29.
        bcf     STATUS,RP0      ; Selecciono banco 0.
        bcf     STATUS,Z
        xorlw   29
        btfsc   STATUS,Z
        goto    DiaCorrecto
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movfw   DiaAnt          ; Miramos si el día anterior era 28.
        bcf     STATUS,RP0      ; Selecciono banco 0.
        bcf     STATUS,Z
        xorlw   28
        btfsc   STATUS,Z
        goto    DiaCorrecto
        goto    NoActualizar
```

```
DiaCorrecto         ; Aqui ya sabemos que el dia es correcto y comprobamos los meses.
        movfw Meses
        movwf AuxPro
        bsf   STATUS,RP0   ; Selecciono banco 1.
        movfw MesAnt
        bcf   STATUS,RP0   ; Selecciono banco 0.
        bcf   STATUS,Z
        subwf AuxPro,f     ; Comprobamos si los meses anteriores y los actuales son
                           ; iguales.
        btfsc STATUS,Z
        goto  MesCorrecto
        decfsz AuxPro      ; De esta manera aceptamos una diferencia de una unidad entre
                           ; los meses anteriores y los actuales.
        goto  SigueComprobacionM
        goto  MesCorrecto

SigueComprobacionM         ; Comprobamos si el cambio de meses es de 12 a 1.
        movfw Minutos
        bcf   STATUS,Z
        xorlw 1            ; Miramos si los meses actuales son 1.
        btfss STATUS,Z
        goto  NoActualizar
        bsf   STATUS,RP0   ; Selecciono banco 1.
        movfw MesAnt       ; Miramos si el mes anterior era 12.
        bcf   STATUS,RP0   ; Selecciono banco 0.
        bcf   STATUS,Z
        xorlw 12
        btfss STATUS,Z
        goto  NoActualizar

MesCorrecto         ; Aqui ya sabemos que los meses son correctos y comprobamos el año.
        movfw Año
        movwf AuxPro
        bsf   STATUS,RP0   ; Selecciono banco 1.
        movfw AñoAnt
        bcf   STATUS,RP0   ; Selecciono banco 0.
        bcf   STATUS,Z
        subwf AuxPro,f     ; Comprobamos si el año anterior y el actual son iguales.
        btfsc STATUS,Z
        goto  Actualizar
        decfsz AuxPro      ; De esta manera aceptamos una diferencia de una unidad entre
                           ; el año anterior y el actual.
        goto  NoActualizar
        goto  Actualizar
```

```
Actualizar
        bsf     Detecta,PRIMERAHORA
        bsf     Detecta,ENHORA          ; Indicamos que el sistema tiene ya una hora valida.
        bsf     Detecta,ACTUALIZA
        bsf     STATUS,RP0              ; Selecciono banco 1.
        movlw   4
        movwf   ContNoAct
        bcf     STATUS,RP0      ; Selecciono banco 0.
        movfw   Horas
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movwf   HoraAnt
        bcf     STATUS,RP0      ; Selecciono banco 0.
        movfw   Minutos
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movwf   MinutoAnt
        bcf     STATUS,RP0      ; Selecciono banco 0.
        movfw   Dias
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movwf   DiaAnt
        bcf     STATUS,RP0      ; Selecciono banco 0.
        movfw   Meses
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movwf   MesAnt
        bcf     STATUS,RP0      ; Selecciono banco 0.
        movfw   Año
        bsf     STATUS,RP0      ; Selecciono banco 1.
        movwf   AñoAnt
        movlw   255
        movwf   Pasadas1
        movlw   16
        movwf   Pasadas2
        bcf     STATUS,RP0      ; Selecciono banco 0.
        return

NoActualizar
        bcf     Detecta,ACTUALIZA
        bsf     Detecta,PRIMERAHORA
        bsf     STATUS,RP0              ; Selecciono banco 1.
        decfsz  ContNoAct               ; Decrementamos el contador de no actualización.
        return
        movlw   4
        movwf   ContNoAct
        bcf     STATUS,RP0              ; Selecciono banco 0.
```

```
        bsf     Detecta,CAMBIOEMIS  ; Indicamos que se debe cambiar la emisora por
                                    ; recepción pobre.
        return


;-----------------------------------------------------------------------------
;----------- Aqui haremos los cálculos para pasar de MJD al calendario normal.---------
;-----------------------------------------------------------------------------

MJDaCALENDARIO:             ; Aplicaremos las formulas de conversión establecidas.

        rrf     MJD3,F      ; Primero hemos de reordenar correctamente los registros, ya
                            ; que al ser el ultimo
        btfss   MJD2,0      ; bit de MJD3 correspondiente a la información de la hora, para
                            ; realizar los calculos de la fecha tenemos que desplazar
        bcf     MJD3,7      ; a la derecha una posición a todos los registros MJD.
        btfsc   MJD2,0
        bsf     MJD3,7
        rrf     MJD2,F
        btfss   MJD1,0
        bcf     MJD2,7
        btfsc   MJD1,0
        bsf     MJD2,7
        rrf     MJD1,F
        movlw   b'00000001'
        andwf   MJD1,F      ; Aqui ya están ordenados.
        call    MJDx100
        bcf     STATUS,C
        movlw   b'11101100' ; Aqui realizamos la operación ((MJDx100)-1507820)
        subwf   Product+5,F
        movlw   b'00000001'
        btfss   STATUS,C
        movlw   b'00000010'
        bcf     STATUS,C
        subwf   Product+4,F
        movlw   b'00010111'
        btfss   STATUS,C
        movlw   b'00011000'
        subwf   Product+3,F

        movfw   Product+5
        movwf   nratorL
        movfw   Product+4
        movwf   nratorM
```

```
movfw Product+3
movwf nratorH
movlw b'10101101'
movwf denomL
movlw b'10001110'
movwf denomM
clrf  denomH
call  Div24          ; Vamos a dividir por 36525 para obtener Y'.

movfw nratorL
movwf Y1             ; Guardamos estos valores para usarlos después.
movfw nratorM
movwf Y2
bcf   STATUS,C
movlw b'01100100'    ; Aqui realizamos la operacion (Y'-100). De esta manera
                     ; podemos almacenar
subwf  nratorL,F     ; el valor del año en un solo byte (ya hemos pasado el 2000).
movlw b'00000000'
btfss STATUS,C
movlw b'00000001'
subwf  nratorM,F
bcf   STATUS,C

movfw nratorL
movwf Año

movfw Y1             ; Ahora calcularemos (Y'x365,25)x100 y lo almacenaremos para
                     ; usarlo posteriormente.
movwf nratorL
movfw Y2
movwf nratorM
clrf  nratorH

movlw b'10101101'
movwf Multiplier+2
movlw b'10001110'
movwf Multiplier+1
clrf  Multiplier
call  MULTIPLY_24x24

movfw Product+5      ; Ahora dividimos lo anterior por 100 y lo volveremos a
                     ; multiplicar por 100 para quedarnos solo con
movwf nratorL        ; la parte entera.
movfw Product+4
```

```
movwf nratorM
movfw Product+3
movwf nratorH
call     DIVIDE100

clrf     Multiplier
clrf     Multiplier+1
movlw b'01100100'
movwf Multiplier+2   ; En el multiplicador cargamos 100.

call     MULTIPLY_24x24

movfw Product+5
movwf Y1             ; Aqui tenemos el LSB de (Y'x36525).
movfw Product+4
movwf Y2
movfw Product+3
movwf Y3             ; Aqui tenemos el MSB de (Y'x36525).

call     MJDx100
call     MJDx100
movfw Product+5
movwf PI1
movfw Product+4
movwf PI2
movfw Product+3
movwf PI3

bcf      STATUS,C
movlw b'00111010'  ; Aqui realizamos la operacion ((MJDx100)-1495610)
subwf   Product+5,F
movlw b'11010010'
btfss    STATUS,C
movlw b'11010011'
bcf      STATUS,C
subwf   Product+4,F
movlw b'00010110'
btfss    STATUS,C
movlw b'00010111'
subwf   Product+3,F

bcf      STATUS,C
movlw b'00110000'  ; Aqui realizamos la operacion ((MJDx100)-1495600) para el
                   ; calculo de los días.
```

```
subwf   PI1,F
movlw b'11010010'
btfss   STATUS,C
movlw b'11010011'
bcf     STATUS,C
subwf   PI2,F
movlw b'00010110'
btfss   STATUS,C
movlw b'00010111'
subwf   PI3,F


bcf     STATUS,C
movfw Y1            ; Aqui realizamos la operacion ((MJDx100)-1495600)-
                    ; -(Y'x36525).
subwf   PI1,F
movfw Y2
btfss   STATUS,C
addlw  b'00000001'
bcf     STATUS,C
subwf   PI2,F
movfw Y3
btfss   STATUS,C
addlw  b'00000001'
subwf   PI3,F


bcf     STATUS,C
movfw Y1            ; Aqui realizamos la operacion ((MJDx100)-1495610)-
                    ; -(Y'x36525).
subwf   Product+5,F
movfw Y2
btfss   STATUS,C
addlw  b'00000001'
bcf     STATUS,C
subwf  Product+4,F
movfw Y3
btfss   STATUS,C
addlw  b'00000001'
subwf   Product+3,F

movfw Product+5
movwf Y1            ; Aqui tenemos el LSB de ((MJDx100)-1495610)-(Y'x36525)).
movfw Product+4
```

```
movwf Y2
movfw Product+3
movwf Y3            ; Aqui tenemos el MSB de ((MJDx100)-1495610)-(Y'x36525).

movfw Product+5     ; Ahora dividimos lo anterior por (30,6x100)=3060.
movwf nratorL
movfw Product+4
movwf nratorM
movfw Product+3
movwf nratorH
movlw b'11110100'
movwf denomL
movlw b'00001011'
movwf denomM
clrf  denomH
call  Div24

movfw nratorL
movwf Meses
movfw Meses         ; hora calcularemos (Y'x365,25)x100 y lo almacenaremos para
                    ; usarlo posteriormente.
movwf nratorL
clrf  nratorM
clrf  nratorH
movlw b'11110100'   ; Calculamos (M'x3060).
movwf Multiplier+2
movlw b'00001011'
movwf Multiplier+1
clrf  Multiplier

call  MULTIPLY_24x24

movfw Product+5     ; Ahora dividimos lo anterior por 100 y lo volveremos a
                    ; multiplicar
movwf nratorL       ; por 100 para quedarnos solo con la parte entera.
movfw Product+4
movwf nratorM
movfw Product+3
movwf nratorH
call  DIVIDE100

clrf  Multiplier
clrf  Multiplier+1
movlw b'01100100'
```

```
movwf Multiplier+2   ; En el multiplicador cargamos 100.

call     MULTIPLY_24x24

bcf      STATUS,C
movfw Product+5      ; Aqui realizamos la operacion ((MJDx100)-1495600)-
                     ; -(Y'x36525)-(M'x3060).
subwf   PI1,F
movfw Product+4

btfss    STATUS,C
addlw  b'00000001'
bcf      STATUS,C
subwf   PI2,F
movfw Product+3
btfss    STATUS,C
addlw  b'00000001'
subwf   PI3,F

movfw PI1
movwf Y1
movfw PI2
movwf Y2
movfw PI3
movwf Y3
movfw Y1             ; Ahora dividimos lo anterior por 100.
movwf nratorL
movfw Y2
movwf nratorM
movfw Y3
movwf nratorH

call     DIVIDE100

movfw nratorL
movwf Dias
btfss    Meses,1
goto     Acabo
btfss    Meses,2
goto     Acabo
btfss    Meses,3
goto     Acabo
incf     Año
movlw b'00001100'
```

```
        subwf  Meses,F

Acabo
        decf    Meses
        return
;-------------------------------------------------------------------------------
MJDx100:
        movfw MJD1
        movwf nratorH
        movfw MJD2
        movwf nratorM
        movfw MJD3
        movwf nratorL
        clrf     Multiplier
        clrf     Multiplier+1
        movlw b'01100100'
        movwf Multiplier+2          ; En el multiplicador cargamos 100.
        call     MULTIPLY_24x24     ; Ahora lo multiplicamos por 100 para evitar los
                                    ; decimales de las formulas de conversion. El máximo
                                    ; resultado posible ocupara 24 bits (en 3 bytes entra).
        return
;-------------------------------------------------------------------------------
DIVIDE100:
        movlw b'01100100'
        movwf denomL
        clrf     denomM
        clrf     denomH
        call     Div24
        return

;-------------------------------------------------------------------------------
;-------------------- SUBRUTINA DE MULTIPLICACION DE 24X24 BITS ----------------
;-------------------------------------------------------------------------------

MULTIPLY_24x24:

      CLRF    Product
      CLRF    Product+1
      CLRF    Product+2
      MOVLW   D'24'
      MOVWF   BitCount
      RRF     Product+3,F
      RRF     Product+4,F
      RRF     Product+5,F
```

```
ADD_LOOP_24x24

        BTFSS   STATUS,C
        GOTO  SKIP_LOOP_24x24
        MOVF    nratorL,W
        ADDWF   Product+2,F
        MOVF    nratorM,W
        BTFSC   STATUS,C
        INCFSZ  nratorM,W
        ADDWF   Product+1,F
        MOVF    nratorH,W
        BTFSC   STATUS,C
        INCFSZ  nratorH,W
        ADDWF   Product,F

SKIP_LOOP_24x24
        RRF     Product,F
        RRF     Product+1,F
        RRF     Product+2,F
        RRF     Product+3,F
        RRF     Product+4,F
        RRF     Product+5,F
        DECFSZ  BitCount,F
        GOTO    ADD_LOOP_24x24

        RETURN
;-------------------- FIN DE  LA SUBRUTINA DE MULTIPLICACION --------------------

;------------------------------------------------------------------------------
;----------------------- SUBRUTINA DE DIVISION DE 24/24 BITS ---------------------
;------------------------------------------------------------------------------

Div24:

divizn
        movlw .24
        movwf BitCount
        movf nratorH,w
        movwf shiftH
        movf nratorM,w
        movwf shiftM
        movf nratorL,w
        movwf shiftL
```

```
        clrf nratorH
        clrf nratorM
        clrf nratorL
        clrf remainH
        clrf remainM
        clrf remainL

dloop
        bcf STATUS,C
        rlf shiftL,f
        rlf shiftM,f
        rlf shiftH,f
        rlf remainL,f
        rlf remainM,f
        rlf remainH,f
        movf denomH,w
        subwf remainH,w
        btfss STATUS,Z
        goto nochk
        movf denomM,w
        subwf remainM,w
        btfss STATUS,Z
        goto nochk
        movf denomL,w
        subwf remainL,w

nochk
        btfss STATUS,C
        goto nogo
        movf denomL,w
        subwf remainL,f
        btfss STATUS,C
        decf remainM,f
        movf remainM,w
        xorlw 0xff
        btfsc STATUS,Z
        decf remainH,f
        movf denomM,w
        subwf remainM,f
        btfss STATUS,C
        decf remainH,f
        movf denomH,w
        subwf remainH,f
        bsf STATUS,C
```

```
nogo:
        rlf nratorL,f
        rlf nratorM,f
        rlf nratorH,f
        decfsz BitCount,f
        goto dloop

          Return
;------------------------- FIN DE  LA SUBRUTINA DE DIVISION ---------------------

;------------------------------------------------------------------------------
;------------- SUBRUTINA PARA PASAR DE BINARIO A BCD NATURAL ----------------
;------------------------------------------------------------------------------

Bin_a_BCD:
          clrf     ContUnidad    ; Inicializamos las variables.
          clrf     UnidadBCD
          clrf     DecenaBCD
          bcf      STATUS,Z
          movfw    NumBinario
          addlw    0
          btfsc    STATUS,Z      ; Comprobamos si el número que tenemos es cero.
          return
          movlw    10            ; Cargamos un 9 en el contador de unidades.
          movwf    ContUnidad

Bucle_conversion
          bcf      STATUS,Z
          incf     UnidadBCD
          decfsz   ContUnidad
          goto     Sigue_Conversion
          movlw    10
          movwf    ContUnidad
          clrf     UnidadBCD
          incf     DecenaBCD

Sigue_Conversion
          decfsz   NumBinario
          goto     Bucle_conversion
          return
;------------------------------------------------------------------

        END
```

PROGRAMA PARA EL MANEJO DEL BUS I2C DEL PUERTO B DEL PIC

```
I2CPORT equ PORTB
SDA equ 1
SCL equ 2

        ERRORLEVEL -224


SDA_High MACRO
        bsf     _TRISB,SDA    ; SDA será entrada
                              ; y valdrá 1 por el Pull-UP
        movfw _TRISB
        tris    I2CPORT
        ENDM

SDA_Low MACRO
        bcf     I2CPORT,SDA  ; SDA = 0
        bcf     _TRISB,SDA    ; SDA será salida
        movfw _TRISB
        tris    I2CPORT
        ENDM

SCL_High MACRO
        bsf     _TRISB,SCL    ; SCL será entrada
                              ; y valdrá 1 por el Pull-UP
        movfw  _TRISB
        tris    I2CPORT
        ENDM

SCL_Low MACRO
        bcf     I2CPORT,SCL  ; SCL = 0
        bcf     _TRISB,SCL    ; SCL será salida
        movfw _TRISB
        tris    I2CPORT
        ENDM


ProgramStartI2C equ $
        org FileStart
_TRISB              res  1 ; Guarda la dirección de PORTB
I2CBit              res 1
I2CData             res 1
Indica_Error        res 1
```

```
        VARIABLE FileStart = $
        org ProgramStartI2C


I2C_Start:
        clrf    Indica_Error    ; Aqui ponemos a cero el indicador de error
        SDA_High
        SCL_High                ; Pongo a uno SDA y SCL
        call    I2C_Delay       ; Le pongo un retardo antes de hacer la comprobación.
        btfss   I2CPORT,SCL     ; SCL=1 ?. Comprobamos posibles errores.
        goto    I2C_Start_Error
        btfss   I2CPORT,SDA     ; SDA=1 ?. Comprobamos posibles errores.
        goto    I2C_Start_Error
        call    I2C_Delay
        SDA_Low
        call    I2C_Delay
        SCL_Low
        call    I2C_Delay
        return

I2C_Start_Error
        bsf     Indica_Error,0  ; Indicamos que se ha producido un error al poner la condición
        return                  ; de start.

I2C_Stop:
        SCL_Low
        SDA_Low
        call    I2C_Delay
        SCL_High
        call    I2C_Delay
        SDA_High
        call    I2C_Delay
        return

I2C_Read:
        movlw d'8'              ; Vamos a recibir 8 bits en cada "paquete".
        movwf I2CBit
_RdBuc
        SCL_High
        call    I2C_Delay
        btfss   I2CPORT,SDA
        goto    _RdBit0
```

```
        bsf     STATUS,C
        goto    _RdCont

_RdBit0
        bcf     STATUS,C

_RdCont
        rlf     I2CData,F
        SCL_Low
        call    I2C_Delay
        decfsz  I2CBit,F
        goto    _RdBuc
        SDA_Low
        SCL_High                ; Para enviar el ACK
        call    I2C_Delay
        SCL_Low
        SDA_High                ; Volvemos a dejarlo como entrada para que siga recibiendo
                                ; datos
        return

I2C_Read_NAK:
        movlw d'8'              ; Vamos a recibir 8 bits en cada "paquete".
        movwf I2CBit

_RdBucN
        SCL_High
        call    I2C_Delay
        btfss   I2CPORT,SDA
        goto    _RdBiN0
        bsf     STATUS,C
        goto    _RdConN

_RdBiN0
        bcf     STATUS,C

_RdConN
        rlf     I2CData,F
        SCL_Low
        call    I2C_Delay
        decfsz  I2CBit,F
        goto    _RdBucN

        SDA_High
        SCL_High                ; Para enviar el NACK
```

```
        call    I2C_Delay
        SCL_Low
        call    I2C_Delay
        call    I2C_Delay
        return

I2C_Write:
        movlw d'8'          ; Vamos a transmitir 8 bits en cada "paquete".
        movwf I2CBit

_WrBuc
        rlf     I2CData,F
        btfss   STATUS,C
        goto    _WrBit0
        SDA_High
        goto    _WrCont

_WrBit0
        SDA_Low

_WrCont
        SCL_High
        call    I2C_Delay
        SCL_Low
        call    I2C_Delay
        decfsz I2CBit,F
        goto    _WrBuc
        SDA_High            ; Para recibir el ACK liberamos el bus dejando la patilla de
                            ; SDA como entrada.
        call    I2C_Delay
        SCL_High
        call    I2C_Delay
        btfsc   I2CPORT,SDA ; Comprobamos si se ha recibido o no el ACK del dispositivo
                            ; esclavo.
        goto    I2C_WrError
        bcf     STATUS,C
        SCL_Low
        return

I2C_WrError
        bsf     Indica_Error,1 ; Indicamos que se no se ha recibido el ACK del dispositivo
                            ; esclavo.
        SCL_Low
        return
```

```
I2C_Delay:
        nop
        return
```

PROGRAMA PARA EL MANEJO DEL BUS I2C DEL PUERTO A DEL PIC

```
I2CPORT2 equ PORTA
SDA2 equ 3
SCL2 equ 2

        ERRORLEVEL -224

SDA_High2 MACRO
        bsf     _TRISB2,SDA2        ; SDA será entrada
                                    ; y valdrá 1 por el Pull-UP
        movfw _TRISB2
        tris    I2CPORT2
        ENDM

SDA_Low2 MACRO
        bcf     I2CPORT2,SDA2       ; SDA = 0
        bcf     _TRISB2,SDA2        ; SDA será salida
        movfw _TRISB2
        tris    I2CPORT2
        ENDM

SCL_High2 MACRO
        bsf     _TRISB2,SCL2        ; SCL será entrada
                                    ; y valdrá 1 por el Pull-UP
        movfw  _TRISB2
        tris    I2CPORT2
        ENDM

SCL_Low2 MACRO
        bcf     I2CPORT2,SCL2       ; SCL = 0
        bcf     _TRISB2,SCL2        ; SCL será salida
        movfw _TRISB2
        tris    I2CPORT2
        ENDM
```

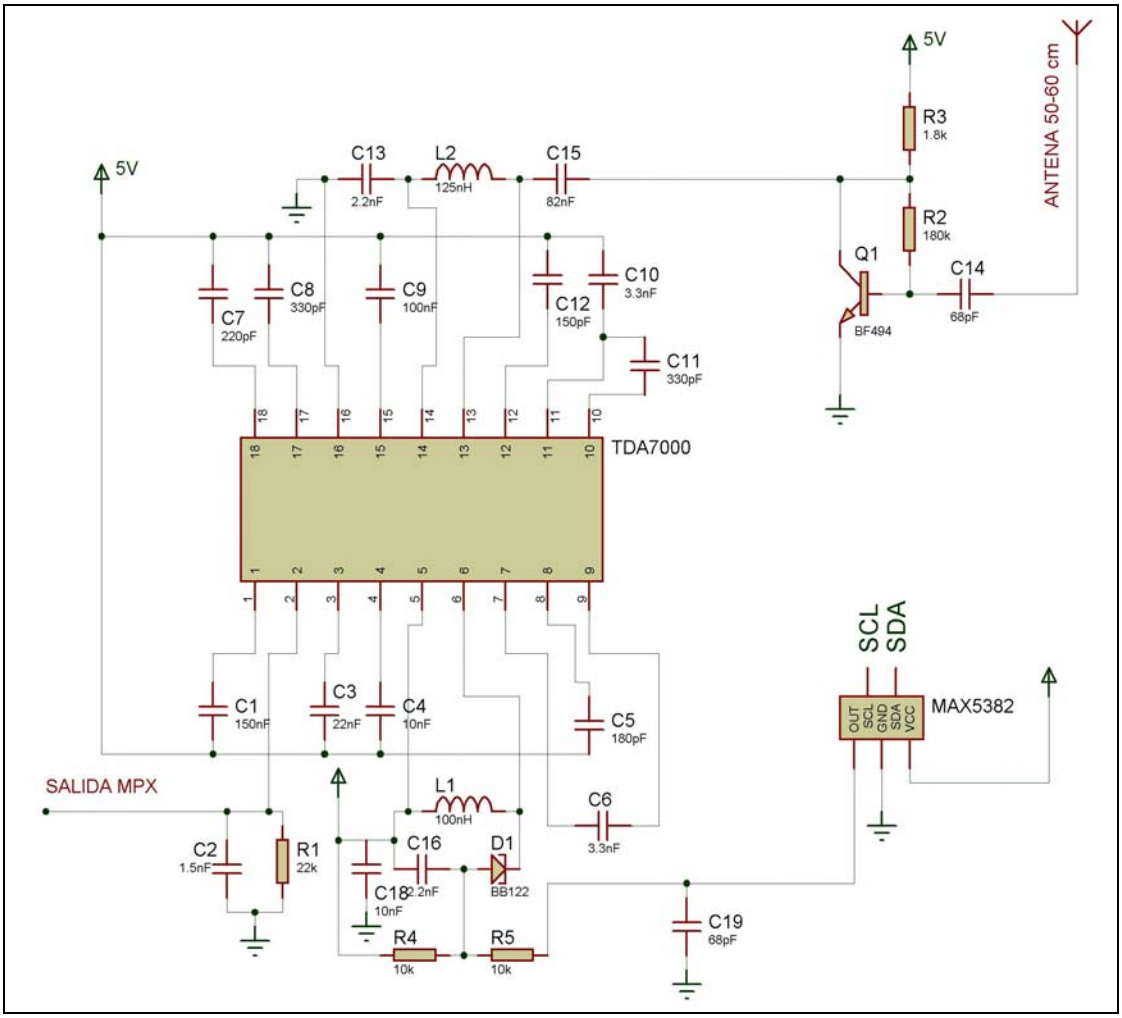
```
ProgramStartI2C2 equ $
        org FileStart
_TRISB2         res 1                   ; Guarda la dirección de PORTA
I2CBit2         res 1
I2CData2        res 1
Indica_Error2   res 1


        VARIABLE FileStart = $
        org ProgramStartI2C2

I2C_Start2:

        clrf    Indica_Error2           ; Aqui ponemos a cero el indicador de error
        SDA_High2
        SCL_High2                       ; Pongo a uno SDA y SCL
        call    I2C_Delay2              ; Le pongo un retardo antes de hacer la comprobación.
        btfss   I2CPORT2,SCL2           ; SCL=1 ?. Comprobamos posibles errores.
        goto    I2C_Start_Error2
        btfss   I2CPORT2,SDA2           ; SDA=1 ?. Comprobamos posibles errores.
        goto    I2C_Start_Error2
        call    I2C_Delay2
        SDA_Low2
        call    I2C_Delay2
        SCL_Low2
        call    I2C_Delay2
        return

I2C_Start_Error2
        bsf     Indica_Error2,0         ; Indicamos que se ha producido un error al poner la
        return                          ; condición de start.

I2C_Stop2:
        SCL_Low2
        SDA_Low2
        call    I2C_Delay2
        SCL_High2
        call    I2C_Delay2
        SDA_High2
        call    I2C_Delay2
        return

I2C_Read2:
        movlw d'8'              ; Vamos a recibir 8 bits en cada "paquete".
```

```
        movwf I2CBit2

_RdBuc2
        SCL_High2
        call    I2C_Delay2
        btfss   I2CPORT2,SDA2
        goto    _RdBit02
        bsf     STATUS,C
        goto    _RdCont2
_RdBit02
        bcf     STATUS,C

_RdCont2
        rlf     I2CData2,F
        SCL_Low2
        call    I2C_Delay2
        decfsz I2CBit2,F
        goto    _RdBuc2
        SDA_Low2
        SCL_High2               ; Para enviar el ACK
        call    I2C_Delay2
        SCL_Low2

        SDA_High2               ; Volvemos a dejarlo como entrada para que siga recibiendo
        return                  ; datos

I2C_Read_NAK2:
        movlw d'8'              ; Vamos a recibir 8 bits en cada "paquete".
        movwf I2CBit2

_RdBucN2
        SCL_High2
        call    I2C_Delay2
        btfss   I2CPORT2,SDA2
        goto    _RdBiN02
        bsf     STATUS,C
        goto    _RdConN2

_RdBiN02
        bcf     STATUS,C

_RdConN2
        rlf     I2CData2,F
        SCL_Low2
```

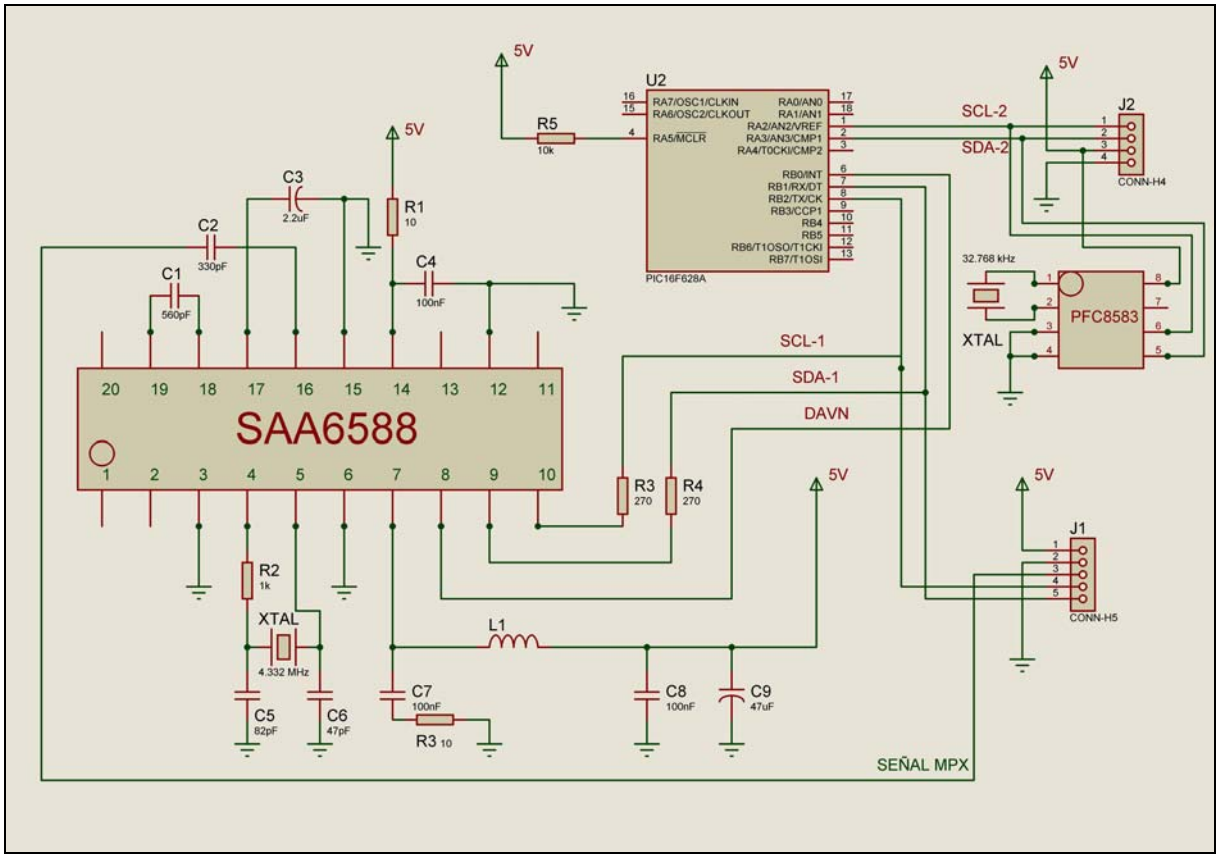
```
        call    I2C_Delay2
        decfsz  I2CBit2,F
        goto    _RdBucN2
        SDA_High2
        SCL_High2               ; Para enviar el NACK
        call    I2C_Delay2
        SCL_Low2
        call    I2C_Delay2
        call    I2C_Delay2
        return

I2C_Write2:
        movlw   d'8'            ; Vamos a transmitir 8 bits en cada "paquete".
        movwf   I2CBit2

_WrBuc2
        rlf     I2CData2,F
        btfss   STATUS,C
        goto    _WrBit02
        SDA_High2
        goto    _WrCont2

_WrBit02
        SDA_Low2

_WrCont2
        SCL_High2
        call    I2C_Delay2
        SCL_Low2
        call    I2C_Delay2
        decfsz  I2CBit2,F
        goto    _WrBuc2
        SDA_High2               ; Para recibir el ACK liberamos el bus dejando la patilla de SDA
                                ; como entrada.
        call    I2C_Delay2
        SCL_High2
        call    I2C_Delay2
        btfsc   I2CPORT2,SDA2           ; Comprobamos si se ha recibido o no el ACK del
                                        ; dispositivo esclavo.
        goto    I2C_WrError2
        bcf     STATUS,C
        SCL_Low2
        return
```

```
I2C_WrError2
        bsf     Indica_Error2,1         ; Indicamos que se no se ha recibido el ACK del
                                        ; dispositivo esclavo.
        SCL_Low2
        return

I2C_Delay2:
        nop

        return
```

APÉNDICE B: ESQUEMAS ELÉCTRICOS

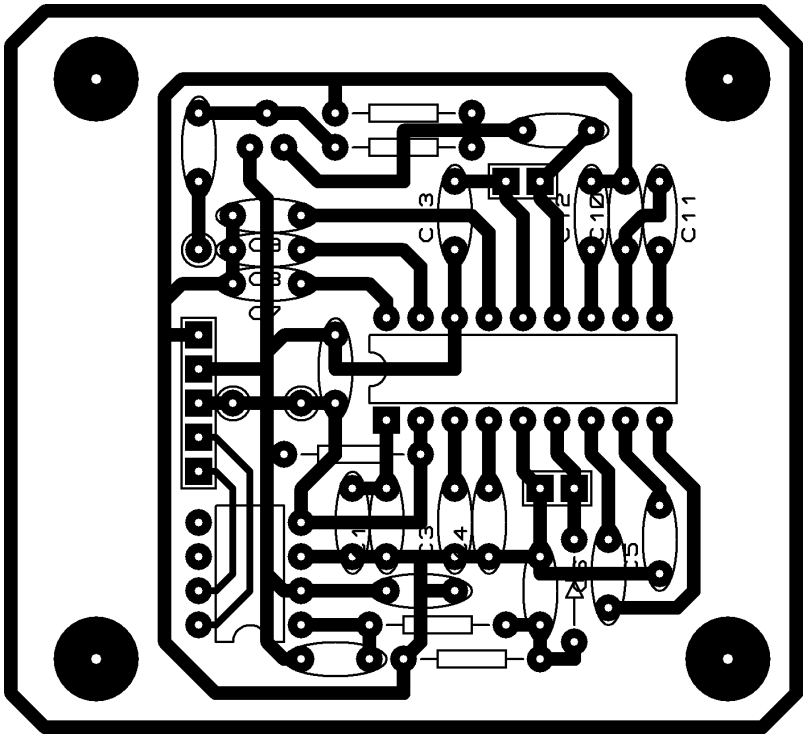


Esquema del circuito receptor de radio FM.

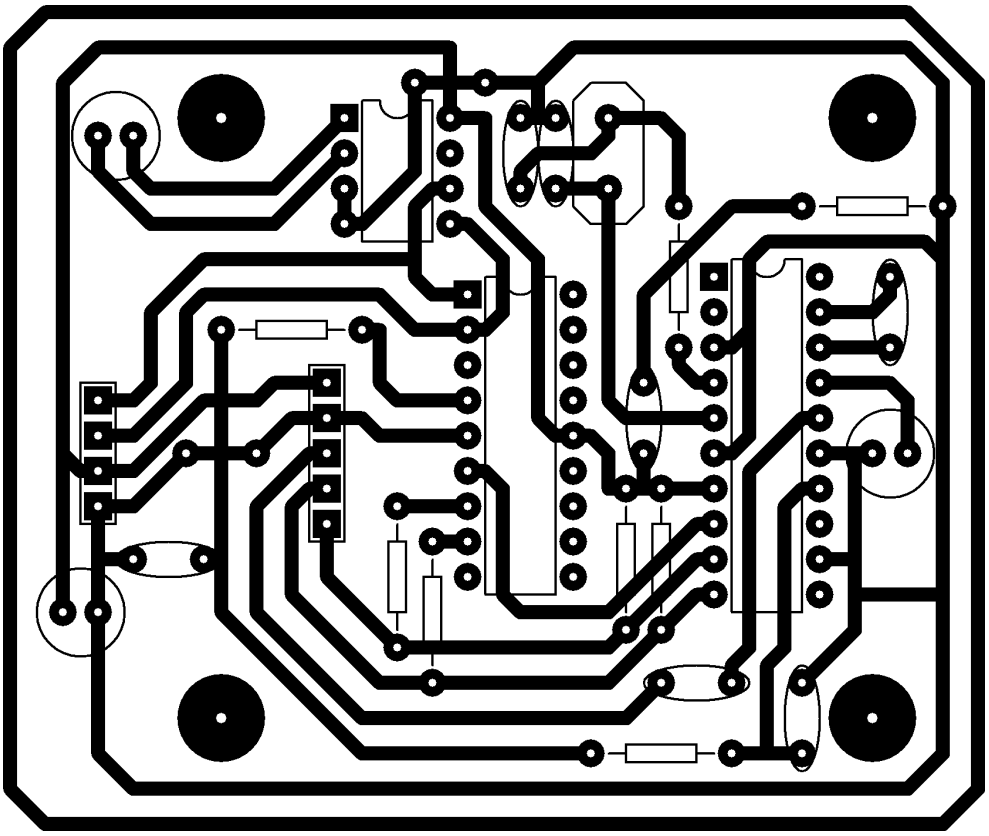


Esquema del sistema decodificador RDS

APÉNDICE C: LAYOUTS



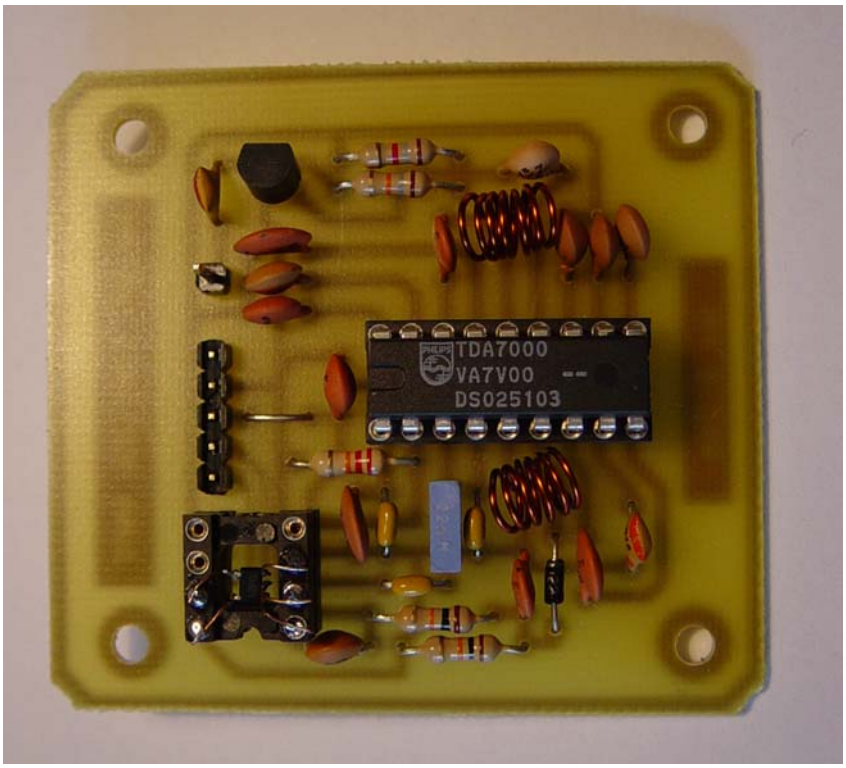
Layout del circuito receptor de radio FM.



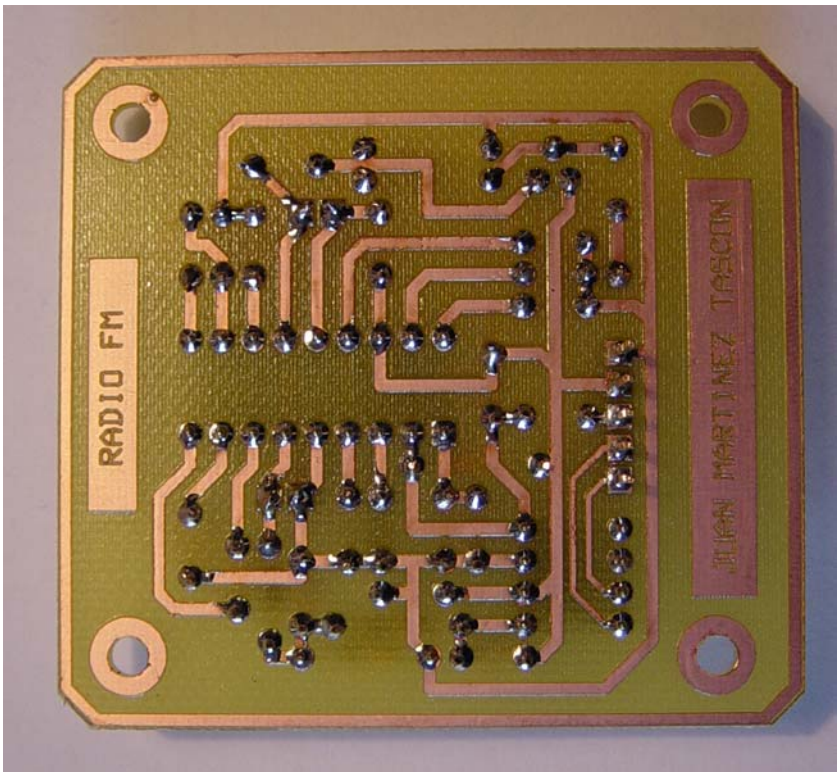
Layout del circuito decodificador RDS.

APÉNDICE D: PROTOTIPO

CIRCUITO DE RADIO FM

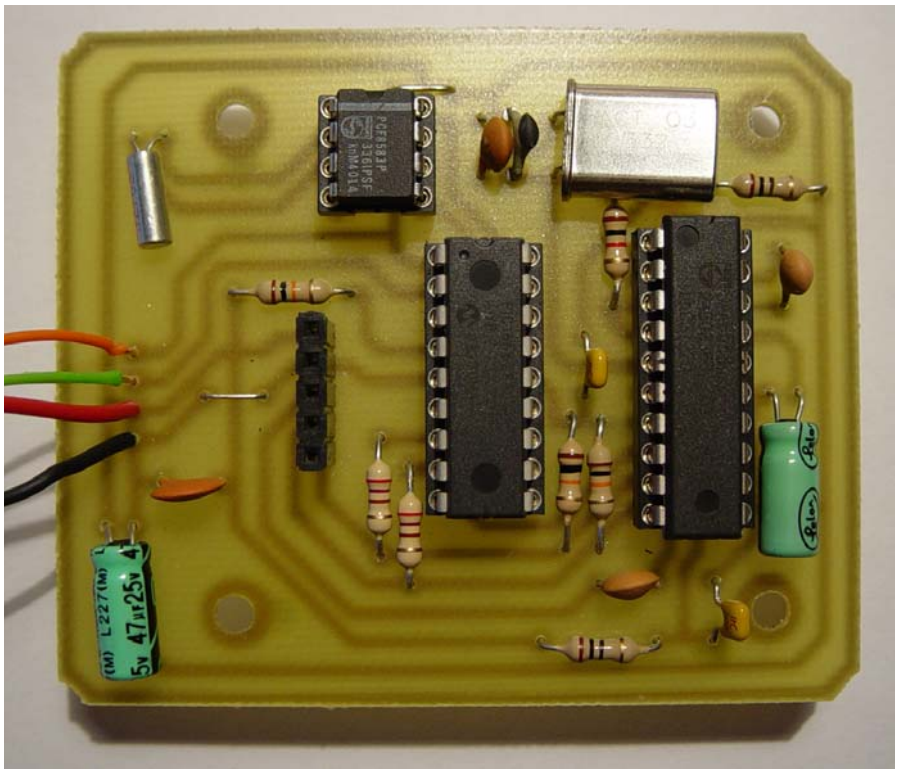


Vista superior del circuito receptor de radio FM.

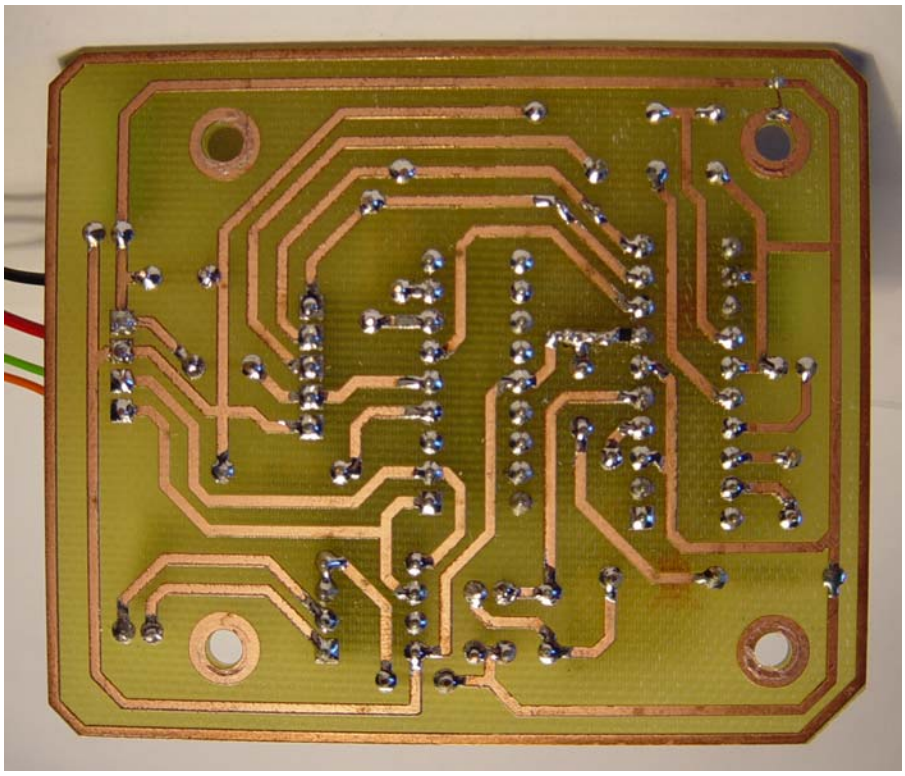


Vista inferior del circuito receptor de radio FM.

CIRCUITO DECODIFICADOR RDS

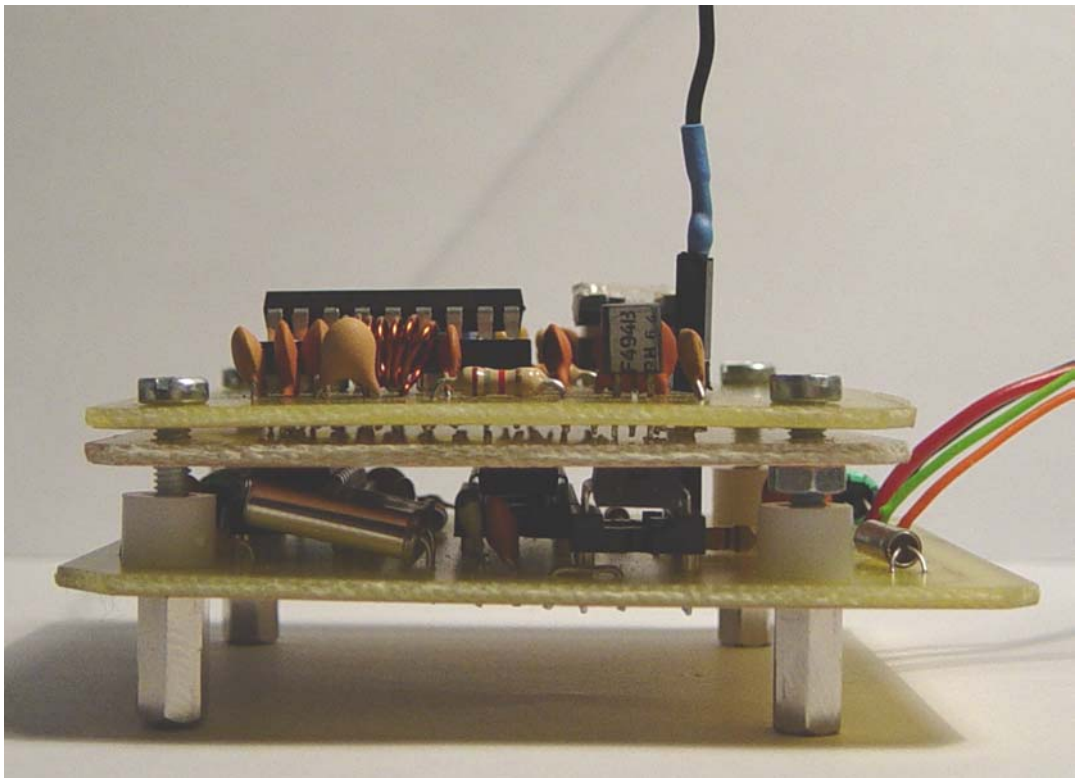


Vista superior del circuito decodificador RDS.

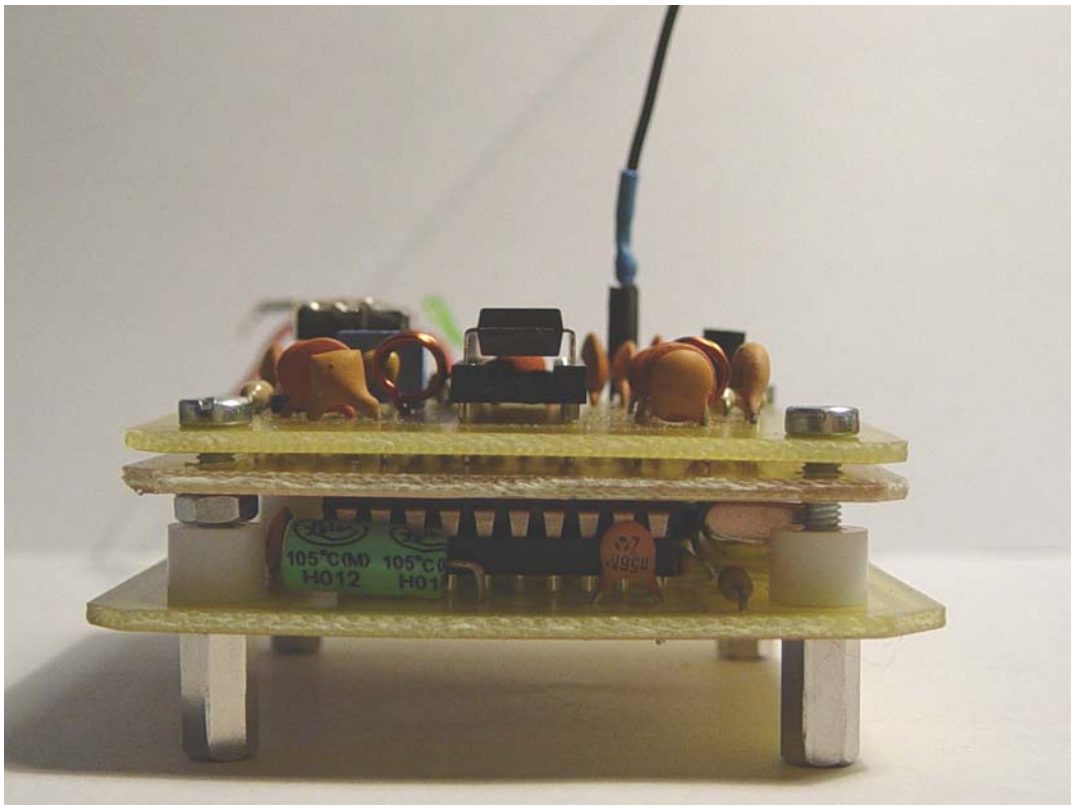


Vista inferior del circuito decodificador RDS.

SISTEMA DECODIFICADOR FM-RDS



Vista lateral del sistema decodificador FM-RDS.



Vista frontal del sistema decodificador FM-RDS

BIBLIOGRAFÍA

- [1] *RDS Universal Encoder Communication Protocol.UECP Version 5.1*
  European Broadcasting Union. RDS Forum. Agosto 1997.

- [2] *United States RBDS Standar. Specification of the radio broadcast data system (RBDS).*
  National Radio Systems Comitee. Abril 1998.

- [3] *Radio Data System (RDS).*
  Aitor Zuloaga Izaguirre. Universidad del País Vasco. Julio 1996.

- [4] *RDS Forum.* http://www.rds.org.uk/

- [5] *The I2C-Bus Specification*. Philips Semiconductor. Version 2.1. Enero 2000

- [6] *Buses de datos en sistemas electrónicos*. Artículo de la revista internacional de electrónica ELEKTOR. Nº 284

- [7] *Radio Receivers.* Miomir Filipovic.
  http://www.mikroelektronika.co.yu/english/product/books/rrbook/rrbook.htm

- [8] http://www.rtve.es/rne/emisoras/rds.htm