Universidad de Valladolid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

Proyecto Fin de Carrera
Ingeniero en Electrónica

Reproductor de MP3 basado en disco duro IDE

Autor:
Fernando Moreno Torrero
Tutor:
Jesús M. Hernández Mangas

Valladolid, 15 de mayo de 2006

TÍTULO:	**Reproductor de MP3 basado en disco duro IDE**
AUTOR:	**Fernando Moreno Torrero**
TUTOR:	**Jesús M. Hernández Mangas**
DEPARTAMENTO:	**Electricidad y Electrónica**

Miembros del tribunal	
PRESIDENTE:	Jesús Arias Álvarez
VOCAL:	Ruth Pinacho Gómez
SECRETARIO:	Jesús M. Hernández Mangas
FECHA DE LECTURA:	
CALIFICACIÓN:	

Resumen del proyecto

Este proyecto consiste en el diseño e implementación de un reproductor MP3 Hardware mediante el uso de un microcontrolador PIC.

Dada su naturaleza de *Trabajo de fin de carrera*, se ha optado por la inclusión de elementos que si bien no son del todo necesarios para la reproducción de la música, lo hacen más atractivo de cara al diseño y a la programación.

- Inclusión de un BOOTLOADER.
- Acceso a dispositivos de almacenamiento IDE.
- Tratamiento de la información en formato FAT32.
- Pantalla LCD.
- Banco de memoria de 32Kx16

Palabras clave

Microcontrolador, PIC16F877, BOOTLOADER, HD, IDE, FAT32, SPI, MP3, LCD

Abstract

This dissertation consist of design, programming and implementation of a MP3 player. It is based on a PIC 16F877A microcontroller and uses a IDE hard disk as support of storage. The files are stored in FAT32 format.

A María

Agradecimientos

A lo largo de la ejecución de este proyecto he tenido la suerte de tener siempre a mi lado a María, no sólo porque he empezado a despertarme a su lado, sino por el apoyo y la paciencia que en todo momento ha demostrado, dulcificando con sólo su presencia los momentos de desesperación que en ocasiones me han acompañado. Por esto y mucho más, gracias María.

Además, tengo que dar las gracias a mi familia desde mi abuela hasta la más pequeñaja, mi sobrina Paula, que con sólo tres añitos escuchaba atentamente las explicaciones que yo le daba cuando ella me preguntaba: '¿qué estás haciendo tío Fer?'.

Parecía que nunca llegaría el final, pero aquí estoy, escribiendo los agradecimientos, y como de agradecimientos se trata, quiero terminar este apartado dando las gracias a todos los que con su ilusión me han empujado a llegar a la meta.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

El por qué de las Cosas

La memoria más potente
es más débil que la tinta más pálida.
~ Proverbio Chino ~

1.1. Introducción

En los último años se ha venido extendiendo en el ámbito de la informática doméstica el estándar **MPEG-3**, o **MP3**, los dos nombres que más se han empleado para denominar incorrectamente a **MPEG-1 Layer-3**, un esquema de codificación de audio general que debe su éxito a su asombrosa capacidad de compresión sin pérdida aparente de calidad, superando un ratio de 10 a 1.

Tal ha sido el boom de este formato de compresión de audio, que en pocos años ha atravesado la barrera de la informática para convertirse en un estándar a lo que aparatos portátiles de audio se refiere. Hoy en día rara es la persona que no dispone de un dispositivo que incorpore la decodificación MP3, reproductores de DVD, equipos de música, radio CD's de coche, walkmans... la lista es interminable.

En este proyecto se va a abordar el diseño tanto del hardware como del software de un reproductor de MP3. De entre las numerosas variantes en cuanto a la elección de los componentes, se ha optado por usar como fuente de datos un disco duro IDE en el cual se encuentran grabadas las canciones en formato FAT32, como decodificador de MP3, se ha optado por el uso de un circuito dedicado.

1.2. Objetivos

El objetivo último del proyecto es, por supuesto, **reproducir canciones comprimidas en formato MP3**, sin embargo, la forma en la que esto se lleva a cabo, nos obliga a

plantearnos otros objetivos intermedios:

- Cargar el código ejecutable en el microcontrolador a través de un **Bootloader**, de forma que no sea necesario extraer el chip de la placa para programarlo. Es lo que normalmente se conoce como programación in-line.

- Mostrar la información relevante en un **Display LCD** de 4 líneas y 20 caracteres por línea, que, junto con los botones, conformarán el interfaz de usuario.

- Acceder a la información contenida en un **disco duro IDE** usando para ello el estándar **AT Attachment *ATA1***.

- Interpretar de forma correcta la información extraída del disco duro, que se encuentra almacenada en el formato **FAT32** creado por Microsoft.

- Emplear un **decodificador de MPEG-1 Layer III** dedicado, para realizar el proceso de descompresión y amplificación de la música.

1.3. Fases de desarrollo del Proyecto

1. **Desarrollo del Hardware**. En esta fase se lleva a cabo el diseño del hardware que va a emplear el dispositivo. Partiendo de un bloque central formado por el microcontrolador, el BUS de datos y las líneas de control, se lleva a cabo el diseño de cada uno de los bloques periféricos, que añaden funcionalidad al dispositivo.

2. **Desarrollo del Software**. Se implementa el código fuente necesario para llevar a cabo todas las operaciones necesarias para la extracción de los datos, la interacción con el usuario y la decodificación.

3. **Pruebas y ajustes finales**. Esta fase es fundamental para la detección de posibles errores. Se somete al dispositivo a una serie de pruebas de caja negra y se ajusta la interfaz de usuario.

1.4. Medios empleados

Para la elaboración del proyecto se han utilizado diversas herramientas que podemos dividirlas en:

1.4.1. Herramientas Software. *Programas de ordenador*

Este conjunto de herramientas se ha usado fundamentalmente para desarrollar la parte Software del proyecto y simular su funcionamiento.

- **UltraEdit-32 v.10.10a**. Potente editor de textos que se ha empleado para escribir todo el código fuente del proyecto.
- **MPLAB ID v.7.30**. Compilador suministrado por Microchip para sus microcontroladores PIC.
- **IC-Prog v.1.05c**. Programa para llevar a cabo la primera grabación del microcontrolador *(Carga del Bootloader)*.
- **Proteus v.6.7 SP3**. Programa para realizar el diseño hardware del dispositivo y llevar a cabo la simulación de diversas partes del circuito.
- **WinEdt v.5.4 / MIKTeX v.2.4**. Entorno de desarrollo LaTeX empleado para escribir la memoria del proyecto.
- **Dia v.0.95-pre4**. Programa para crear diversos tipos de diagramas. En este proyecto se ha usado para elaborar los diagramas de flujo.

1.4.2. Instrumentos de Laboratorio

Instrumentos empleados para la elaboración del prototipo y para el ajuste del mismo.

- **Grabador de circuitos PIC compatible JDM**. Necesario para grabar en el microcontrolador el código ejecutable del Bootloader.
- **Placas montaje de prototipos**. Soporte sobre el que se sueldan todos los componentes.
- **Osciloscopio y Polímetro digitales**. Se han usado para testar las señales digitales.
- **Diversas herramientas manuales**. Soldadores, destornilladores, pinzas...

1.4.3. Resumen de componentes utilizados

Para realizar el prototipo se han usado los siguientes componentes:

- Microprocesador Microchip PIC 16F877A.
- Circuito decodificador de MPEG-1 VLSI VS1001K.
- Display LCD retroiluminado
- Disco duro de 3.5 pulgadas
- Diversos circuito lógicos TTL como: transceptores de BUS, multiplexores, latches, contadores, puertas AND y NOT.
- Circuitos de memoria estática SRAM
- Circuitos convertidores de niveles y/o tensiones
- Circuitos reguladores de tensión
- Elementos discretos: resistencias, condensadores, osciladores, diodos.
- Botones
- Placa de montaje experimental
- Varios: tornillos, separadores, cables...

1.5. El estandar MPEG-1 Layer III

1.5.1. Introducción

Trataremos de explicar brevemente que se esconde tras un 'MP3' y en que se basan sus capacidades. Para saber como funciona no tenemos porque llegar a las matemáticas profundas del modelo psicoacústico sólo nos basta con entender algunos conceptos relativamente sencillos.

Si no te gusta leer te bastará con saber lo siguiente: *Un MP3 es un sistema de compresión de audio con el cual podemos almacenar música con calidad CD en* $\frac{1}{12}$ *del espacio original.*

MPEG-1 Layer III.

Las siglas 'MP3' responden a una abreviación de MPEG-1 layer 3. Es un algoritmo de codificación perceptual desarrollado por el consorcio MPEG (Moving Picture Expert Group) junto con el Instituto Tecnológico Fraunhofer que finalmente se ha estandarizado como norma ISO-MPEG Audio Layer 3 (IS 11172-3 y IS 13818-3) y que viene a ser un avance importante sobre los anteriores desarrollos (Layer 1 y Layer 2).

El hecho de que haya sido adoptado como una norma ISO es más importante de lo que cabría suponer. Las normas ISO definen muchos estándares del mercado y tienen peso frente a la industria. Además facilita el trabajo a las personas que quieran desarrollar aplicaciones o cualquier otra cosa dado que tiene a su alcance el funcionamiento del sistema.

Esta tecnología no es nueva, lleva desarrollándose más de 10 años, lo que ocurre es que ahora es el momento en el que el desarrollo de la tecnología la ha hecho asequible para el usuario medio. subsectionCodificación Perceptual y Oído Humano.

El sistema de codificación perceptual es un sistema de compresión con pérdida, esto quiere decir que el sonido original y el comprimido no son exactamente iguales. Estas pérdidas responden al funcionamiento del oído humano, así aunque los sonidos no son iguales si los percibimos como si lo fuesen.

Se suele comparar el sistema de compresión perceptual del sonido con los sistemas de compresión gráficos JPEG. Estos se diferencian de otros como el BMP o TIFF porque no mantiene la imagen inalterada sino que realizan aproximaciones al original en pos de una mayor compresión.

El rango de frecuencias que percibe el oído humano esta aproximadamente entre los 20Hz y los 20kHz siendo más sensible entre los 2Hz y 4KHz.

Además cuando tenemos una señal de un volumen alto en una frecuencia y otra de un volumen más bajo en una frecuencia cercana esta queda tapada por la anterior. Esto es lo que se llama efecto de enmascaramiento.

Así pues de lo que se trata es de aprovechar los defectos del oído humano para desechar todo aquello que realmente no vamos a oír. Por supuesto cada uno tiene su oído y por eso para asegurar el éxito de estos sistemas se utilizan métodos estadísticos.

Codificación de Sub Bandas.

Para aprovechar estas características se utiliza un sistema denominado Codificación de Sub Bandas.

En este proceso la señal original se descompone en subbandas mediante un banco de filtros o algún método parecido. Estas subbandas son comparadas con el original mediante el modelo psicoacústico que determina que bandas son importantes cuales no y cuales pueden ser eliminadas.

Dependiendo del bit rate al que vayamos a producir la codificación este proceso eliminara más o menos datos siguiendo el modelo psicoacústico hasta lograr la compresión necesaria.

Luego se cuantifican y codifican las subbandas restantes y el resultado es finalmente comprimido mediante un algoritmo standard Huffman o LZW.

Cifras.

Dentro del formato 'MP3' podemos comprimir con distinto ancho de banda, modo y bit rate obteniendo distintas calidades según para que vayamos a utilizar ese sonido.

Canal del sonido	Ancho de Banda	Modo	Bit rate	Ratio de Compresión
Sonido telefónico	2.5 kHz	mono	8 kbps	96:1
Mejor que onda corta	4.5 kHz	mono	16 kbps	48:1
Mejor que radio AM	7.5 kHz	mono	32 kbps	24:1
Similar a radio FM	11 kHz	estéreo	56...64 kbps	26...24:1
Cercano al CD	15 kHz	estéreo	96 kbps	16:1
CD	Más de 15 kHz	estéreo	112..128kbps	14..12:1

Cuadro 1.1: Ratios de compresión.

En un disco compacto tenemos una onda de 44.1kHz 16bit estéreo eso significa aproximadamente 1400Kbps (44100 x 16 x 2 bits por segundo). Codificándolo por ejemplo en un 'MP3' de 128kbps obtenemos una reducción en torno al $\frac{1}{12}$ del espacio inicial.

También se puede optar por compresiones a mayor bit rate llegando a 192kbps o incluso 320kbps. Pero el más popular es el de 128kbps con el que se consigue una calidad excelente con una compresión sobresaliente.

1.5.2. Compresión de Audio.

Digitalización.

El sonido es una onda continua que se propaga a través del aire u otros medios formada por diferencias de presión, de forma que puede detectarse por la medida del nivel de presión en un punto. Las ondas sonoras poseen las características propias y medibles de las ondas en general, tales como reflexión, refracción y difracción. Al tratarse de una onda continua, se requiere un proceso de digitalización para representarla como una serie de números. Actualmente, la mayoría de las operaciones realizadas sobre señales de sonido son digitales, pues tanto el almacenamiento como el procesado y transmisión de la señal en forma digital ofrece ventajas muy significativas sobre los métodos analógicos. La tecnología digital es más avanzada y ofrece mayores posibilidades, menor sensibilidad al ruido en la transmisión y capacidad para incluir códigos de protección frente a errores, así como encriptación. Con los mecanismos de decodificación adecuados, además, se pueden tratar simultáneamente señales de diferentes tipos transmitidas por un mismo canal. La desventaja principal de la señal digital es que requiere un ancho de banda mucho mayor que el de la señal analógica, de ahí que se realice un exhaustivo estudio en lo referente a la compresión de datos, algunas de cuyas técnicas serán el centro de nuestro estudio.

El proceso de digitalización se compone de dos fases: muestreo y cuantización. En el muestreo se divide el eje del tiempo en segmentos discretos: la frecuencia de muestreo será la inversa del tiempo que medie entre una medida y la siguiente. En estos momentos se realiza la cuantización, que, en su forma más sencilla, consiste simplemente en medir el valor de la señal en amplitud y guardarlo. El teorema de Nyquist garantiza que la frecuencia necesaria para muestrear una señal que tiene sus componentes más altas a una frecuencia dada f es como mínimo $2f$. Por tanto, si el rango superior de la audición humana está en torno a los 20 Khz, la frecuencia que garantiza un muestreo adecuado para cualquier sonido audible será de unos 40 Khz. Concretamente, para obtener sonido de alta calidad se utilizan frecuencias de 44'1 Khz, en el caso del CD, por ejemplo, y hasta 48 Khz, en el caso del DAT. Otros valores típicos son los submúltiplos de la primera, 22 y 11 Khz. Según la naturaleza de la aplicación, por supuesto, las frecuencias adecuadas pueden ser muy inferiores, de tal manera que el proceso de la voz acostumbra a realizarse a una frecuencia de entre 6 y 20 Khz, o incluso menos. En lo referente a la cuantización, es evidente que cuantos más bits se utilicen para la división del eje de la amplitud, más 'fina' será la partición y por tanto menor el error al atribuir una amplitud concreta al sonido en cada instante. Por ejemplo, 8 bits ofrecen 256 niveles de cuantización y 16, 65536. El margen dinámico de la audición humana es de unos 100 dB. La división del eje se puede realizar a intervalos iguales o

según una determinada función de densidad, buscando más resolución en ciertos tramos si la señal que se trata tiene más componentes en cierta zona de intensidad, como veremos en las técnicas de codificación.

El proceso completo se denomina habitualmente PCM (Pulse Code Modulation) y así nos referiremos a el en lo sucesivo. Se ha descrito de forma sumamente simplista, principalmente porque está ampliamente tratado y es sobradamente conocido, siendo otro el campo de estudio de este trabajo. Sin embargo, entraremos en detalle en todo momento que sea necesario para el desarrollo de la exposición.

Codificación y Compresión.

Antes de describir los sistemas de codificación y compresión, debemos detenernos en un breve análisis de la percepción auditiva del ser humano, para comprender por qué una cantidad significativa de la información que proporciona el PCM puede desecharse. El centro de la cuestión, en lo que a nosotros respecta, se basa en un fenómeno conocido como enmascaramiento.

El oído humano percibe un rango de frecuencias entre 20 Hz y 20 Khz. En primer lugar, la sensibilidad es mayor en la zona alrededor de los 2-4 Khz, de forma que el sonido resulta más difícilmente audible cuanto más cercano a los extremos de la escala se encuentra. En segundo lugar está el enmascaramiento, cuyas propiedades utilizan exhaustivamente los algoritmos más interesantes: cuando la componente a cierta frecuencia de una señal tiene una energía elevada, el oído no puede percibir componentes de menor energía en frecuencias cercanas, tanto inferiores como superiores. A una cierta distancia de la frecuencia enmascaradora, el efecto se reduce tanto que resulta despreciable; el rango de frecuencias en las que se produce el fenómeno se denomina banda crítica (critical band). Las componentes que pertenecen a la misma banda crítica se influyen mutuamente y no afectan ni se ven afectadas por las que aparecen fuera de ella. La amplitud de la banda crítica es diferente según la frecuencia en la que nos situemos y viene dada por unos determinados datos que demuestran que es mayor con la frecuencia. Hay que señalar que estos datos se obtienen por experimentos psicoacústicos, que se realizan con expertos entrenados en percepción sonora, dando origen con sus impresiones a los modelos psicoacústicos.

Este que hemos descrito es el llamado enmascaramiento simultáneo o en frecuencia. Existe, asimismo, el denominado enmascaramiento asimultáneo o en el tiempo, así como otros fenómenos de la audición que no resultan relevantes en este punto. Por ahora, centrémonos en la idea de que ciertas componentes en frecuencia de la señal admiten un

mayor ruido del que generalmente consideraríamos tolerable y, por tanto, requieren menos bits para ser codificadas si se dota al codificador de los algoritmos adecuados para resolver máscaras.

La digitalización de la señal mediante PCM es la forma más simple de codificación de la señal, y es la que utilizan tanto los CD como los sistemas DAT. Como toda digitalización, añade ruido a la señal, generalmente indeseable. Como hemos visto, cuantos menos bits se utilicen en el muestreo y la cuantización, mayor será el error al aceptar valores discretos para la señal continua, esto es, mayor será el ruido. Para evitar que el ruido alcance un nivel excesivo hay que emplear un gran número de bits, de forma que a 44'1 Khz. y utilizando 16 bits para cuantizar la señal, uno de los dos canales de un CD produce más de 700 kilobits por segundo (kbps). Como veremos, gran parte de esta información es innecesaria y ocupa un ancho de banda que podría liberarse, a costa de aumentar la complejidad del sistema decodificador e incurrir en cierta pérdida de calidad. El compromiso entre ancho de banda, complejidad y calidad es el que produce los diferentes estándares del mercado y formará la parte esencial de nuestro estudio.

Calidad	**Muestreo**	**Bits/Muestra**	**Modo**	**Tasa de Bits**	**Frecuencia**
Teléfono	8 kHz	8	mono	64 kbps	200-3400 Hz
Radio AM	11.025 kHz	8	mono	88 kbps	
Radio FM	22.050 kHz	16	estéreo	705.6 kbps	
CD	44.1 kHz	16	estéreo	1411.2 kbps	20-20000 Hz
DAT	48 kHz	16	estéreo	1536 kbps	20-20000 Hz

Cuadro 1.2: Comparación de formatos de calidad de audio

Un modo mejor de codificar la señal es mediante PCM no-lineal o cuantización logarítmica, que como ya comentamos consiste en dividir el eje de la amplitud de tal forma que los escalones sean mayores cuanta más energía tiene la señal, con lo que se consigue una relación $\frac{señal}{ruido}$ igual o mejor con menos bits. Con este método se puede reducir el canal de CD de audio a 350 kbps, lo cual evidentemente es una mejora sustancial, aunque puede reducirse mucho más. Otros sistemas similares nos llevan a la cuantización adaptativa (APCM), diferencial (DPCM) y la mezcla de ambas, ADPCM. Así prosigue la reducción del ancho de banda, pero sin llegar a los niveles que proporciona el tener en cuenta los efectos del enmascaramiento.

Codificación sub-banda (SBC).

La codificación sub-banda o SBC (sub-band coding) es un método potente y flexible para codificar señales de audio eficientemente. A diferencia de los métodos específicos para

ciertas fuentes, el SBC puede codificar cualquier señal de audio sin importar su origen, ya sea voz, música o sonido de tipos variados. El estándar MPEG Audio es el ejemplo más popular de SBC, y lo analizaremos posteriormente en detalle.

El principio básico del SBC es la limitación del ancho de banda por descarte de información en frecuencias enmascaradas. El resultado simplemente no es el mismo que el original, pero si el proceso se realiza correctamente, el oído humano no percibe la diferencia. Veamos tanto el codificador como el decodificador que participan en el tratamiento de la señal.

La mayoría de los codificadores SBC utilizan el mismo esquema. Primero, un filtro o un banco de ellos, o algún otro mecanismo descompone la señal de entrada en varias subbandas. A continuación se aplica un modelo psicoacústico que analiza tanto las bandas como la señal y determina los niveles de enmascaramiento utilizando los datos psicoacústicos de que dispone. Considerando estos niveles de enmascaramiento se cuantizan y codifican las muestras de cada banda: si en una frecuencia dentro de la banda hay una componente por debajo de dicho nivel, se desecha. Si lo supera, se calculan los bits necesarios para cuantizarla y se codifica. Por último se agrupan los datos según el estándar correspondiente que estén utilizando codificador y decodificador, de manera que éste pueda descifrar los bits que le llegan de aquél y recomponer la señal.

La decodificación es mucho más sencilla, ya que no hay que aplicar ningún modelo psicoacústico. Simplemente se analizan los datos y se recomponen las bandas y sus muestras correspondientes.

En los últimos diez años la mayoría de las principales compañías de la industria de audio han desarrollado sistemas SBC. A finales de los años ochenta, un grupo de estandarización del ISO llamado Motion Picture Experts Group (MPEG) comenzó a desarrollar los estándares para la codificación tanto de audio como de vídeo. Veremos el MPEG Audio como un ejemplo de un sistema práctico SBC.

1.5.3. MPEG Audio.

El estándar MPEG audio.

El estándar MPEG Audio contempla tres niveles diferentes de codificación-decodificación de la señal de audio, de los cuales sólo el primero está totalmente terminado. Los otros dos son aplicables, y de hecho se utilizan habitualmente, pero siguen abiertos a ampliaciones. Estos tres niveles son:

- **MPEG-1**: Codificación de imágenes en movimiento y audio asociado para medios de almacenamiento digital hasta 1'5 Mbit/s
- **MPEG-2**: Codificación genérica de imágenes en movimiento e información de audio asociada
- **MPEG-3**: La planificación original contemplaba su aplicación a sistemas HDTV; finalmente fue incluido dentro de MPEG-2.
- **MPEG-4**: Codificación de objetos audiovisuales

A su vez, MPEG describe tres esquemas de codificación de audio denominados esquema-1, esquema-2 y esquema-3. Del primero al tercero aumentan tanto la complejidad del codificador como la calidad del sonido. Los tres son compatibles jerárquicamente, esto es, el decodificador esquema-i es capaz de interpretar información producida por un codificador esquema-i y todos los niveles por debajo del i. Así, un decodificador esquema-3 acepta los tres niveles de codificación, mientras el esquema-2 sólo acepta el 1 y el 2.

MPEG define, para cada esquema, el formato del bitstream y el decodificador (que puede ser implementado de diferentes maneras). Con vistas a admitir futuras mejoras no se define el codificador, pero en un apartado informativo se da un ejemplo de un codificador para cada uno de los esquemas. Hay que decir que tanto MPEG-1 como MPEG-2 emplean estos tres esquemas, pero este último añade nuevas características.

Introducción al sistema MPEG-1.

Este es el sistema que describe la norma ISO en lo referente al sistema MPEG-1:
Codificación: el codificador procesa la señal de audio digital y produce el bitstream empaquetado para su almacenamiento y/o transmisión. El algoritmo de codificación no está determinado, y puede utilizar enmascaramiento, cuantización variable y escalado. Sin embargo, debe ajustarse a las especificaciones del decodificador.

Las muestras se introducen en el codificador y a continuación el mapeador crea una representación filtrada y submuestreada de la señal de entrada. Las muestras mapeadas se denominan tanto muestras de subbanda (esquemas 1 y 2) como muestras de subbanda transformadas (esquema 3). El modelo psicoacústico crea una serie de datos (dependiendo de la implementación del codificador) que sirven para controlar la cuantización y codificación. Este último bloque crea a su vez su propia serie de datos, de nuevo dependiendo de la implementación. Por último, el bloque de empaquetamiento de trama se encarga de

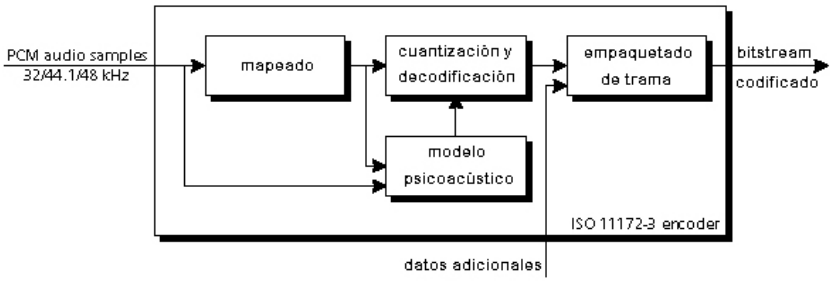


Figura 1.1: Codificador según la norma ISO 11172-3

agrupar como corresponde todos los datos, pudiendo añadir algunos más, llamados datos adicionales, como por ejemplo CRC o información del usuario.

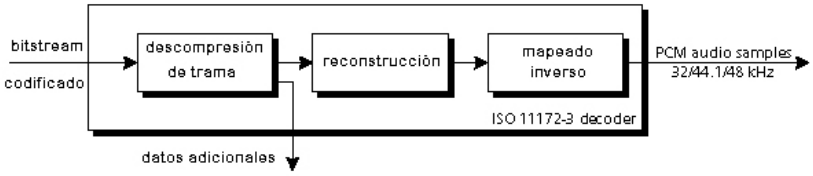


Figura 1.2: Decodificador según la norma ISO 11172-3

Decodificación: el decodificador debe procesar el bitstream para reconstruir la señal de audio digital. La especificación de este elemento sí esta totalmente definida y debe seguirse en todos sus puntos. La figura ilustra el esquema del decodificador.

Los datos del bitstream son desempaquetados para recuperar las diversas partes de la información. El bloque de reconstrucción recompone la versión cuantizada de la serie de muestras mapeadas. El mapeador inverso transforma estas muestras de nuevo a PCM.

Esquemas:

1. Incluye la división del mapeado básico de la señal de audio digital en 32 subbandas, segmentación para el formateo de los datos, modelo psicoacústico y cuantización fija. El retraso mínimo teórico es de 19 ms.

2. Incluye codificación adicional, factores de escala y diferente composición de trama. El retraso mínimo teórico es de 35 ms.

3. Incluye incremento de la resolución en frecuencia, basado en el uso de un banco de filtros híbrido. Cuantización no uniforme, segmentación adaptativa y codificación entrópica de los valores cuantizados. El retraso mínimo teórico es de 59 ms.

Esquema	Objetivo	Compresión	Calidad 64 kbps	Calidad 128 kbps	Retardo
Esquema-1	192 kbps	4 a 1			19 ms
Esquema-2	128 kbps	6 a 1	2'1 a 2'6	Más de 4	35 ms
Esquema-3	64 kbps	12 a 1	3'6 a 3'8	Más de 4	59 ms

Cuadro 1.3: Resumen de datos de los tres esquemas

La calidad viene dada del 1 al 5, siendo el 5 la superior. Hay que señalar que pese a los números de la norma ISO, el retraso típico acostumbra a ser tres veces mayor en la práctica.

Modos: hay cuatro modos de funcionamiento para cualquiera de estos tres esquemas.

- **Single channel o canal único:** Una señal en un bitstream.
- **Dual channel o canal doble:** Dos señales independientes en un mismo bitstream.
- **Stereo:** Como el anterior, perteneciendo las señales al canal izquierdo y derecho de una señal estéreo original.
- **Joint stereo:** Como el anterior, aprovechando ciertas características del estéreo como irrelevancia y redundancia de datos para reducir la tasa de bits.

MPEG-1 en detalle.

Tras haber visto la introducción que figura en los documentos ISO, podemos pasar a analizar en detalle el funcionamiento del sistema. A continuación haremos hincapié en las características y diferencias entre los tres esquemas de MPEG-1.

La codificación:

1. **El banco de filtros:** realiza el mapeado del dominio del tiempo al de la frecuencia. Existen dos tipos: el **polifase** y el **híbrido** polifase/MDCT. Estos bancos proporcionan tanto la separación en frecuencia para el codificador como los filtros de reconstrucción del decodificador. Las muestras de salida del banco están cuantizadas.

2. **El modelo psicoacústico:** calcula el nivel a partir del cual el ruido comienza a ser perceptible, para cada banda. Este nivel se utiliza en el bloque de asignación de $\frac{bit}{ruido}$ para determinar la cuantización y sus niveles. De nuevo, se utilizan dos diferentes. En ambos, los datos de salida forman el SMR (signal-to-mask ratio) para cada banda o grupo de bandas.

3. **Asignación de $\frac{bit}{ruido}$:** examina tanto las muestras de salida del banco de filtros como el SMR proporcionado por el modelo psicoacústico, y ajusta la asignación de bit o ruido, según el esquema utilizado, para satisfacer simultáneamente los requisitos de tasa de bits y de enmascaramiento.

4. **El formateador de bitstream:** toma las muestras cuantizadas del banco de filtros, junto a los datos de asignación de $\frac{bit}{ruido}$ y otra información lateral para formar la trama.

Los tres esquemas utilizan diferentes algoritmos para cumplir estas especificaciones:

Esquema I:

- El mapeado tiempo-frecuencia se realiza con un banco de filtros polifase con 32 subbandas. Los filtros polifase consisten en un conjunto de filtros con el mismo ancho de banda con interrelaciones de fase especiales que ofrecen una implementación eficiente del filtro subbanda. Se denomina filtro subbanda al que cubre todo el rango de frecuencias deseado. En general, los filtros polifase combinan una baja complejidad de computación con un diseño flexible y múltiples opciones de implementación.

- El modelo psicoacústico utiliza una FFT (Fast Fourier Transform) de 512 puntos para obtener información espectral detallada de la señal. El resultado de la aplicación de la FFT se utiliza para determinar los enmascaramientos en la señal, cada uno de los cuales produce un nivel de enmascaramiento, según la frecuencia, intensidad y tono. Para cada subbanda, los niveles individuales se combinan y forman uno global, que se compara con el máximo nivel de señal en la banda, produciendo el SMR que se introduce en el cuantizador.

- El bloque de cuantización y codificación examina las muestras de cada subbanda, encuentra el máximo valor absoluto y lo cuantiza con 6 bits. Este valor es el factor de escala de la subbanda. A continuación se determina la asignación de bits para cada subbanda minimizando el NMR (noise-to-mask ratio) total. Es posible que algunas subbandas con un gran enmascaramiento terminen con cero bits, es decir, no se codificará ninguna muestra. Por último las muestras de subbanda se cuantizan linealmente según el número de bits asignados a dicha subbanda concreta.

- El trabajo del empaquetador de trama es sencillo. La trama, según la definición ISO, es la menor parte del bitstream decodificable por sí misma. Cada trama empieza con una cabecera para sincronización y diferenciación, así como 16 bits opcionales de CRC para detección y corrección de errores. Se emplean, para cada subbanda, 4 bits para describir la asignación de bits y otros 6 para el factor de escala. El resto de bits en la trama se utilizan para la información de samples, 384 en total, y con la opción de añadir cierta información adicional. A 48 Khz, cada trama lleva 8 ms de sonido.

Esquema II:

- El mapeado de tiempo-frecuencia es idéntico al del esquema I.
- El modelo psicoacústico es similar, salvo que utiliza una FFT de 1024 puntos para obtener mayor resolución espectral. En los demás aspectos, es idéntico.
- El bloque de cuantización y codificación también es similar, generando factores de escala de 6 bits para cada subbanda. Sin embargo, las tramas del esquema II son tres veces más largas que las del esquema I, de forma que se concede a cada subbanda tres factores de escala, y el codificador utiliza uno, dos o los tres, según la diferencia que haya entre ellos. La asignación de bits es similar a la del esquema I.
- El formateador de trama: la definición ISO de trama es la misma que en el punto anterior. Utiliza la misma cabecera y estructura de CRC que el esquema I. El número de bits que utilizan para describir la asignación de bits varía con las subbandas: 4 bits para las inferiores, 3 para las medias y dos para las superiores, adecuándose a las bandas críticas. Los factores de escala se codifican junto a un número de dos bits que indica si se utilizan uno, dos o los tres. Las muestras de subbanda se cuantizan y a continuación se asocian en grupos de tres, llamados gránulos. Cada uno se codifica con una palabra clave, lo que permite interceptar mucha más información redundante que en el esquema I. Cada trama contiene, pues, 1152 muestras PCM. A 48 Khz. cada trama lleva 24 ms de sonido.

Esquema III:

- El esquema III es sustancialmente más complicado que los dos anteriores e incluye una serie de mejoras cuyo análisis resultaría desbordante, de manera que no entraremos en tantos detalles. Su diagrama de flujos es conceptuálmente semejante al visto para los otros dos esquemas, salvo que se realizan múltiples iteraciones para procesar los datos con el mayor nivel de calidad en un cierto tiempo, lo cual complica su diseño hasta el punto de que los diagramas ISO ocupan decenas de páginas.

- El mapeado de tiempo-frecuencia añade un nuevo banco de filtros, el DCT (Discrete Cosine Transform), que con el polifase forman el denominado filtro híbrido. Proporciona una resolución en frecuencia variable, 6x32 o 18x32 subbandas, ajustándose mucho mejor a las bandas críticas de las diferentes frecuencias.

- El modelo psicoacústico es una modificación del empleado en el esquema II, y utiliza un método denominado predicción polinómica. Incluye los efectos del enmascaramiento temporal.

- El bloque de cuantización y codificación también emplea algoritmos muy sofisticados que permiten tramas de longitud variable. La gran diferencia con los otros dos esquemas es que la variable controlada es el ruido, a través de bucles iterativos que lo reducen al mínimo posible en cada paso.

- El formateador de trama: la definición de trama para este esquema según ISO varía respecto de la de los niveles anteriores: 'mínima parte del bitstream decodificable mediante el uso de información principal adquirida previamente'. Las tramas contienen información de 1152 muestras y empiezan con la misma cabecera de sincronización y diferenciación, pero la información perteneciente a una misma trama no se encuentra generalmente entre dos cabeceras. La longitud de la trama puede variarse en caso de necesidad. Además de tratar con esta información, el esquema III incluye codificación Huffman de longitud variable, un método de codificación entrópica que sin pérdida de información elimina redundancia. Los métodos de longitud variable se caracterizan, en general, por asignar palabras cortas a los eventos más frecuentes, dejando las largas para los más infrecuentes.

La decodificación:

Es mucho más sencilla que la codificación, de manera que con lo ya comentado en partes anteriores basta para seguir los siguientes diagramas ISO que incluyen algunas notas aclaratorias al margen que no forman parte de las figuras originales de la norma.

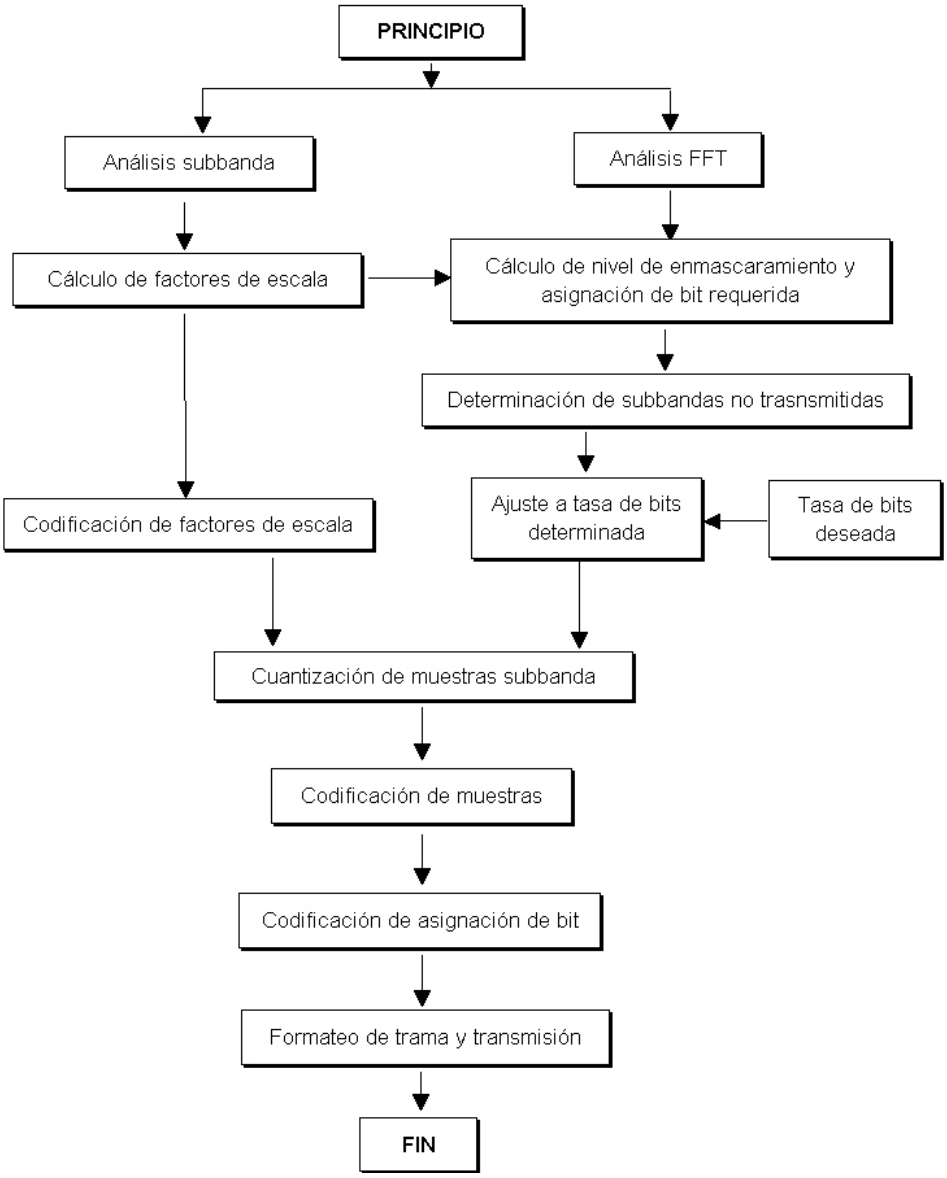


Figura 1.3: Diagrama de flujo del codificador para esquema-1 y esquema-2 según ISO 11172-3

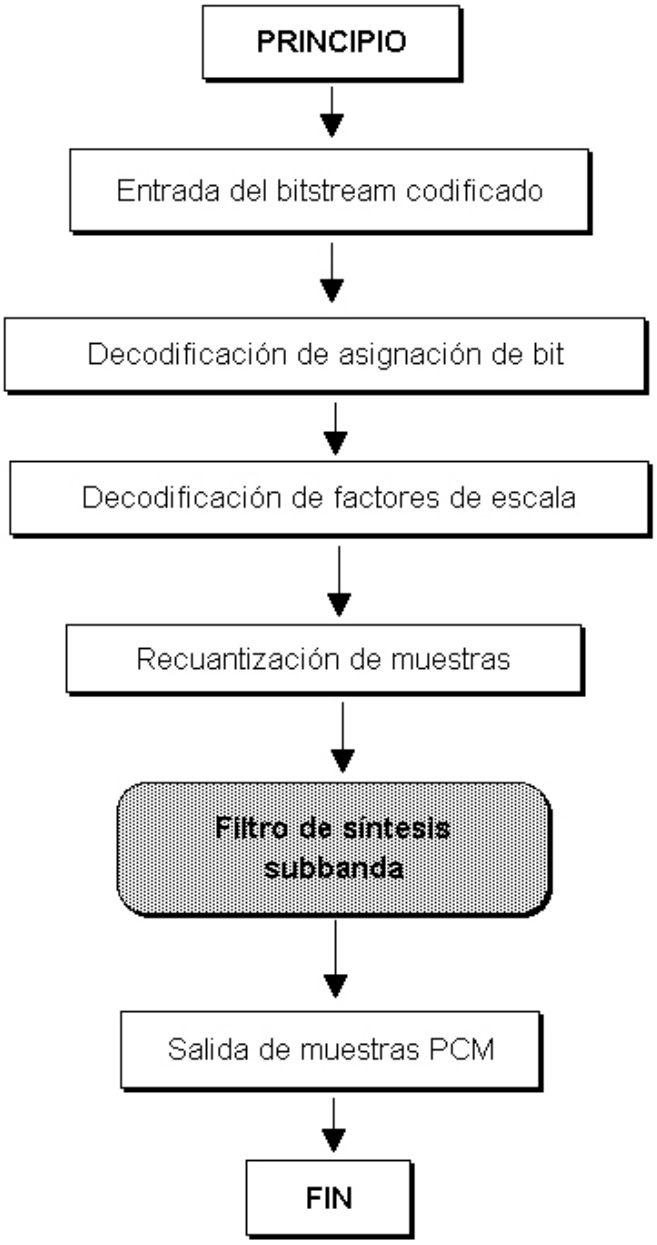


Figura 1.4: Diagrama de flujo del decodificador para esquema-1 y esquema-2 según ISO 11172-3

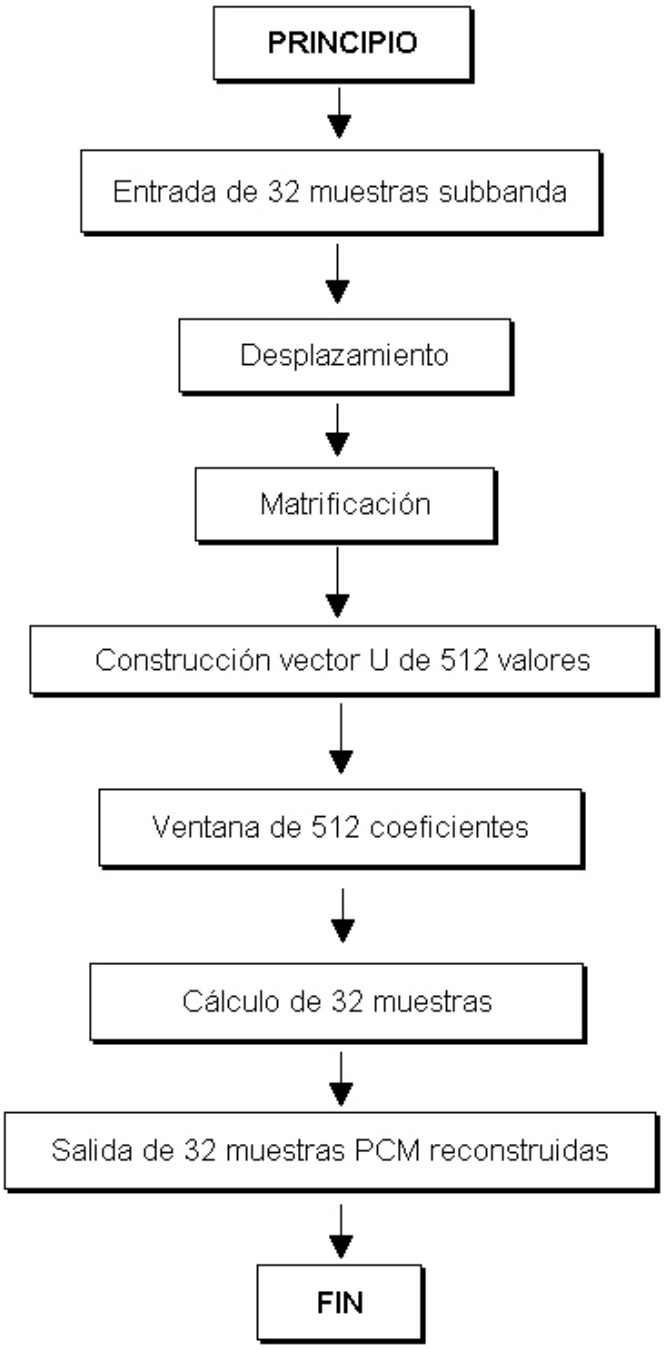


Figura 1.5: Detalle del filtro de síntesis subbanda

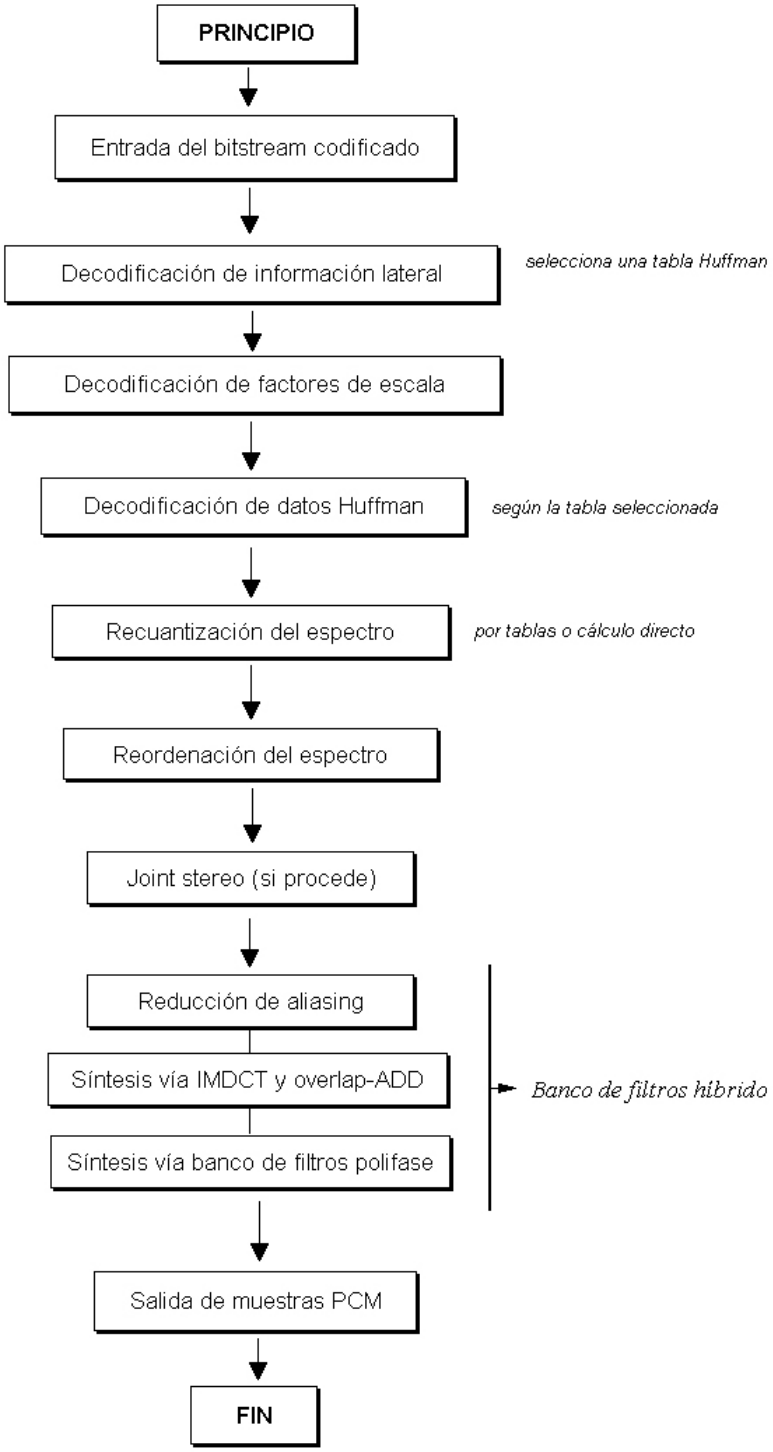


Figura 1.6: Diagrama de flujos del decodificador para esquema-3 según ISO 11172-3

1.5.4. Aplicaciones del estándar MPEG-1.

Ya tenemos una idea medianamente clara de qué es y cómo funciona, pero ¿para qué sirve emplear tiempo y dinero en comprimir el sonido?. Ya hemos visto los diferentes ratios de compresión que alcanzan los tres esquemas:

El esquema-1 obtiene la mayor calidad de sonido a 384 kbps. Las aplicaciones para las que resulta más útil son las relacionadas con la grabación, tanto en cinta como disco duro o discos magneto-ópticos, que aceptan esta tasa de bits sin problemas.

El esquema-2 produce sus mejores resultados de calidad a 256 kbps, pero se mantiene en un nivel aceptable hasta los 64 kbps. Esto hace que se utilice en transmisión de audio, televisión, grabación profesional o doméstica y productos multimedia.

Ciertamente, el mejor miembro de la familia es el esquema-3. Para una determinada calidad de sonido ofrece la menor tasa de bits y viceversa, fijando la tasa de bits ofrece la mejor calidad posible.

El esquema-3 está orientado a aplicaciones donde la necesidad de un ancho de banda reducido justifique el costoso y sofisticado sistema de codificación. La calidad es excelente hasta 64 kbps, de forma que se utiliza, como veremos ahora con más detalle, en telecomunicaciones y sistemas de sonido profesional, así como a nivel de usuario por parte de aficionados con formación informática. Los siguientes puntos se complementan con ejemplos reales de la industria y el mercado del audio.

Conexiones musicales vía ISDN.

Las redes telefónicas digitales (ISDN = Integrated Services Digital Network) ofrecen servicios seguros de conexión con dos canales de datos de 64 kbps por adaptador; en otras redes los canales son de 56 kbps, pero en ambas los costes de transmisión son similares a las líneas telefónicas tradicionales analógicas, que permiten un máximo de 56 kbps (vía módem). Con el esquema-3 una conexión de banda estrecha ISDN de bajo coste permite transmitir sonido con calidad CD. Los estudios de sonido y estaciones de transmisión se benefician de la posibilidad de la 'música por teléfono' de varias maneras. Se ahorra dinero, pues sólo se paga el tiempo de transmisión, a diferencia de la línea telefónica y únicamente se emplea un pequeño conector ISDN para cada canal. Los programas pueden aumentar su atractivo, ofreciendo tomas de alta calidad y noticias en directo sin la pérdida de calidad del sonido telefónico. Aparecen nuevos campos, como el Estudio Virtual, donde artistas en distintas localidades pueden tocar y grabar juntos sin necesidad de viajar hasta el estudio en sí.

Ejemplos:

- En 1992, Radio FFN, una estación privada de Niedersachsen, Alemania, reemplazó sus líneas telefónicas tradicionales por conexiones ISDN y codecs esquema-3, para transmitir 8 programas locales diarios al estudio central. El ahorro declarado en cuotas de transmisión fue de 300.000$ anuales.
- En uno de los primeros ensayos reales de la potencia de este sistema, todas las estaciones de radio privadas en Alemania utilizaron codecs esquema-3 durante los Juegos Olímpicos de Invierno en Albertville para conectar las diferentes localizaciones del evento con la sede central en Meribel, con gran éxito.
- En varios festivales internacionales de música se ha experimentado con éxito en la conexión entre lugares muy separados, uniendo a diferentes artistas en la interpretación de una obra.

Transmisión digital por satélite.

Actualmente se encuentra en pleno funcionamiento un sistema de transmisión de sonido digital a escala mundial por satélite. El nombre del proyecto es Worldstar y utiliza tres satélites en órbita geoestacionaria, llamados AfriStar1 (21 Este), CaribStar1 (95 Oeste) y AsiaStar1 (105 Este), el lanzamiento del primero tuvo lugar a mediados del año 1998, los demás fueron lanzados en los siguientes doce meses. Cada uno está equipado con tres canales de conexión que se pueden multiplexar en hasta 96 subcanales de 16 kbps. Estos son combinables dinámicamente, de manera que se pueden agrupar para formar canales de hasta 128 kbps de capacidad, codificados con el esquema-3. Así, se pueden utilizar cuatro subcanales para formar uno de 64 kbps para transmitir un concierto y al finalizar, utilizar cada uno de ellos para enviar las noticias en cuatro idiomas diferentes.

La empresa responsable del proyecto, Worldspace, ofrece canales en sus tres satélites y ha firmado acuerdos con Voice of America, Radio Nederland, Kenya Broadcasting Corporation, National Broadcasting of Ghana, National Broadcasting of Zimbabwe, New Sky Media of Korea y RCN of Columbia, sumando en total un millón de dólares en inversiones. Alcatel Espace, de Francia, se encargó tanto de la contratación de la lanzadera como del equipo de comunicaciones. Los receptores se han diseñado buscando la máxima simplicidad con los resultados más efectivos. Se han previsto dos millones de estas unidades, que apenas requerirán sintonización y serán totalmente automáticas. El chip principal de estos sistemas ha sido fabricado por ITT Intermetall con tecnología DSP y su nombre es 'MAS3503C'

Audio en Internet.

Como es sabido, Internet es una red mundial de conmutación de paquetes con cientos de miles de máquinas unidas entre sí por medio de varios sistemas de comunicaciones. Los proveedores profesionales normalmente acceden a la red a través de enlaces con un ancho de banda muy elevado (hasta 2 Gbps). El consumidor doméstico, sin embargo, utiliza canales de bajo coste y ancho de banda limitado (ISDN de 64 kbps o conexión telefónica de 56 kbps o ADSL). La tasa de transmisión efectiva varía en función del uso de la parte de la red accedida, situándose en algún punto entre cero y la máxima capacidad del módem.

Sin la codificación de audio, obtener ficheros de audio sin comprimir de un servidor remoto llevaría a unos tiempos de transmisión simplemente inaceptables. Por ejemplo, suponiendo que se usa la tasa de 56 kbps una pista de CD de sólo tres minutos tardaría más de una hora en recibirse. Por tanto, el audio en Internet exige un método de codificación que ofrezca la mayor calidad posible a la vez que permita la decodificación en tiempo real para un amplio número de plataformas sin necesidad de ampliar el hardware, aunque incluya esta posibilidad como elemento de hipotéticas tarjetas de ampliación. Por supuesto, la elección es el esquema-3. Hay varios reproductores en tiempo real, como el WinAmp, que ofrecen el servicio requerido.

En la práctica, las expectativas se han cumplido con creces, de tal manera que el fenómeno 'MP3' está en plena expansión en la telaraña mundial. Ya hay innumerables servidores que ofrecen diferentes piezas en el formato esquema-3 (ficheros de extensión .MP3), de los cuales forman parte tanto aficionados como casas de grabación y grupos independientes. Además, se incorporan temas en este formato a las páginas WEB como elemento para incrementar su atractivo, de forma similar a como se venía haciendo con el MIDI, salvando la barrera de las muy inferiores posibilidades de este.

Llegados a este punto, hay que señalar la importancia de respetar los copyrights a la hora de incluir temas de música en un servidor, así como al almacenarlos en disco duro o CD-ROM. La perspectiva de decuplicar la capacidad de los CD's tradicionales resulta sumamente interesante a la comunidad informática, y dado el auge de las grabadoras domésticas puede decirse que el mercado pirata de CD's conteniendo las discografías al completo de diversos grupos o compositores es una realidad, sea con ánimo de lucro o no. El auge del DVD-ROM como estándar no supone sino un agravamiento del problema.

Las aplicaciones legales que se conocen hasta ahora son, por ejemplo, las de Opticom y Cerberus Sound. La primera ofrece soluciones para que las casas ofrezcan a los clientes audio por demanda, enviando los temas seleccionados al ordenador remoto del usuario.

Cerberus se dedica a la comercialización directa de estos temas como un sistema más de venta electrónica. Asimismo se avanza en el concepto de Internet Radio, dado que se obtiene calidad superior a la de la onda corta con un ancho de banda tan escaso como 16 kbps. Opticom de nuevo está a la cabeza en este campo, junto a Telos, compañía que asociada con Apple presentó en Septiembre de 1996 la tecnología Audioactive. Por último, el gigante Microsoft anunció en Diciembre de ese mismo año su intención de incluir el esquema-3 como parte de la tecnología multimedia Netserver.

Electrónica de consumo.

Por último pero no por ello menos importante, está el campo de la electrónica de consumo. Hoy en día son innumerables los aparatos que ofrecen la posibilidad de reproducir ficheros comprimidos en MP3, desde reproductores de DVD a autorradios pasando como no por los diminutos walkmans que en tan sólo unos pocos centímetros, incorporan todo lo necesario para almacenar y reproducir cientos, por no decir miles de canciones.Podemos decir que la posibilidad de reproducir ficheros comprimidos MP3 se ha convertido en un valor añadido en todos los dispositivos de audio. MP3 se ha convertido en

1.5.5. Especificaciones ISO MPEG.

MPEG-1: Cinco partes, las tres primeras estandarizadas desde 1992. Terminado.

- **IS-11172-1** (Sistema) - Describe la sincronización y multiplexación de señales de audio y vídeo.
- **IS-11172-2** (Vídeo) - Describe la compresión de señales de vídeo, centrándose en el escaneo progresivo y considerando especialmente las aplicaciones de vídeo en CD.
- **IS-11172-3** (Audio) - Describe una familia genérica de codificación de audio, con tres miembros jerárquicamente compatibles, denominados esquema-1, esquema-2 y esquema-3.
- **IS-11172-4** (Tests de conformidad) - Describe los procedimientos para determinar las características de los bitstreams codificados y el proceso de decodificación, así como los tests de conformidad con los requerimientos establecidos en las otras partes.
- **DTR-11172-5** (Simulación por software) - Es un informe técnico sobre la implementación por software de las tres primeras partes de MPEG-1.

MPEG-2: Nueve partes, las tres primeras estandarizadas desde 1994, con algunos añadidos posteriores. En diferentes estados de acabado.

- **IS-13818-1** (Sistema) - Describe la sincronización y multiplexación de señales de audio y vídeo; estandarizado por ITU-T como H.222.

- **IS-13818-2** (Vídeo) - Describe un conjunto genérico de herramientas para codificación de vídeo; estandarizado por ITU-T como H.262.

- **IS-13818-3** (Audio) - Describe una extensión de MPEG-1 compatible hacia atrás, para codificación de audio multicanal (sonido envolvente, sonido multilingüe) y una extensión no compatible hacia atrás para frecuencias de muestreo inferiores, para soportar aplicaciones de sonido con requerimientos de ancho de banda limitado.

- **IS-13818-4** (Tests de concordancia) - Describe los procedimientos para determinar las características de los bitstreams codificados y el proceso de decodificación, así como los tests de conformidad con los requerimientos establecidos en las otras partes.

- **DTR-13818-5** (Simulación por software) - Es un informe técnico sobre la implementación por software de las tres primeras partes de MPEG-2.

- **IS-13818-6** (Extensiones de sistema - Control y comandos para medios de almacenamiento digital) - describe un conjunto de protocolos para aplicaciones cliente-servidor.

- **CD-13818-7** (Audio, codificación no compatible hacia atrás (NBC)) - Describe un esquema de codificación de audio mejorado para señales mono y estéreo, así como para sonido multicanal.

- **13818-8** (Vídeo, extensión para muestras de entrada de 10 bits) - Se ha retirado, debido al escaso interés.

- **IS-13818-9** (Especificación del interface en tiempo real para aplicaciones low-jitter) - Define las restricciones temporales para el envío en tiempo real de bitstreams MPEG-2.

- **WD-13818-10** (Extensiones de concordancia - DSM-CC) - Describe los añadidos a IS-13818-4 para DSM-CC.

1.5.6. El modelo Psicoacústico.

Los modelos psicoacústicos se componen a partir de las percepciones de un grupo de personas entrenadas para rendir al máximo en este campo. Por medio de una serie de experimentos se puede determinar la sensibilidad del oído humano a una serie de fenómenos, de forma que aparezcan resultados útiles para el tratamiento del sonido, como ya hemos visto.

Las tres siguientes características de la audición se acompañan del experimento que sirve para cuantificarlas.

Mínimo umbral auditivo.

Este umbral, también conocido como umbral absoluto, corresponde al sonido de intensidad más débil que se puede escuchar en un ambiente silencioso. El mínimo umbral auditivo no tiene un comportamiento lineal; se representa por una curva de Intensidad (dB) contra Frecuencia (Hz), que posee niveles mínimos entre 2 y 5 KHz, los cuales corresponden a la parte más sensitiva del oído humano. Por lo tanto, en los sistemas de compresión de audio que sacan provecho de la psicoacústica, no es necesario codificar los sonidos situados bajo este umbral (el área por debajo de la curva), ya que éstos no serán percibidos.

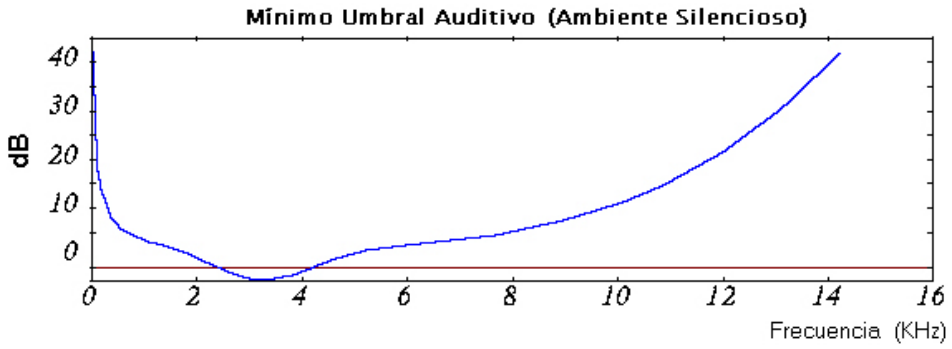


Figura 1.7: Mínimo umbral auditivo

Enmascaramiento.

El efecto de enmascaramiento se basa en las limitaciones del oído humano para responder a todas las componentes de un sonido complejo. Durante los sonidos fuertes, no se pueden oír los sonidos más débiles. Por ejemplo, cuando un músico organista no está tocando, se puede escuchar el resoplido de los tubos; y cuando el músico toca, se pierde el sonido de éstos porque ha sido enmascarado.

Enmascaramiento en frecuencia.

Funciona de manera que un sonido en determinada frecuencia puede enmascarar o disminuir el nivel de otro sonido en las frecuencias adyacentes, siempre y cuando el nivel del sonido enmascarante sea más alto (un sonido más intenso, más fuerte) que el nivel del sonido adyacente.

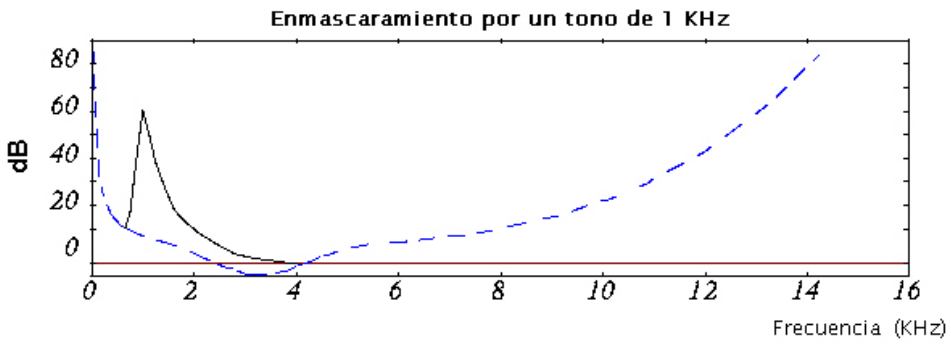


Figura 1.8: Enmascaramiento de un tono

Enmascaramiento temporal.

Se presenta cuando un tono suave está muy cercano en el dominio del tiempo (unos cuantos milisegundos) a un tono fuerte. Si se está escuchando un tono suave y aparece un tono fuerte, el tono suave será enmascarado por el tono fuerte, antes de que el tono fuerte efectivamente aparezca (preenmascaramiento). Posteriormente, cuando el tono fuerte desaparece, el oído necesita un pequeño intervalo de tiempo (entre 50 y 300 ms) para que se pueda seguir escuchando el tono suave (post-enmascaramiento).

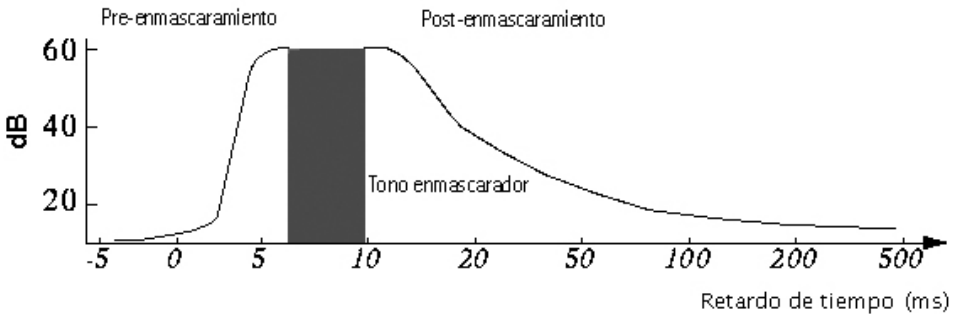


Figura 1.9: Enmascaramiento temporal

Con el post-enmascaramiento no hay problemas; pero el preenmascaramiento sugiere que un tono será enmascarado por otro tono, antes de que el tono enmascarador realmente aparezca, atentando contra el buen juicio de cualquier oyente. Para este fenómeno, se han presentado dos explicaciones:

1. El cerebro integra el sonido sobre un período de tiempo, y procesa la información por ráfagas en la corteza auditiva.

2. Simplemente, el cerebro procesa los sonidos fuertes más rápido que los sonidos suaves.

Sin importar el mecanismo, el caso es que el preenmascaramiento temporal en verdad existe, así sea exageradamente pequeño (se ha calculado con un valor aproximado de 30 ms).

En un sonido cualquiera, se presentan ambos tipos de enmascaramiento. El enmascaramiento en frecuencia es mucho más importante que el enmascaramiento temporal; aunque en ciertos dispositivos para compresión de audio se tienen en cuenta ambos tipos de enmascaramiento, con lo cual se logra mejor compresión de datos. Superponiendo ambas gráficas en una sola que presente tres ejes, se puede ver una curva bajo la cual están todos los sonidos que no pueden ser escuchados.

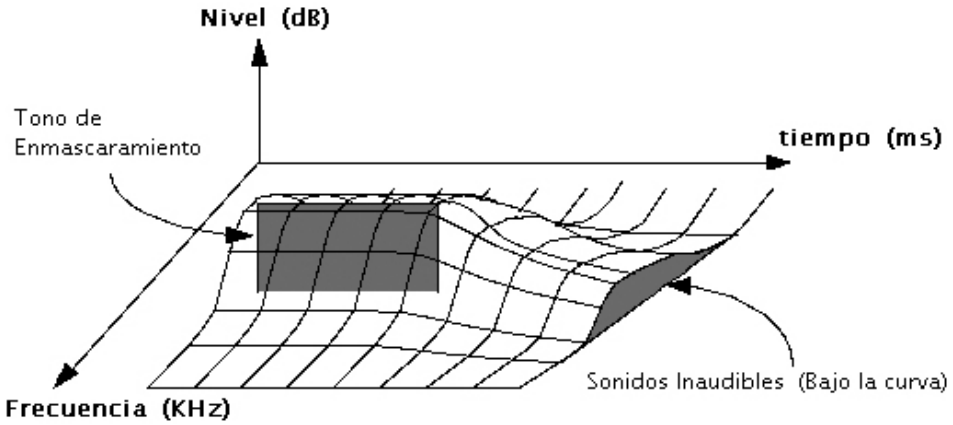


Figura 1.10: Sonidos que no pueden ser escuchados

Joint stereo o estéreo conjunto.

Si una señal de audio es estereofónica se puede lograr comprimirla, con base en la irrelevancia o redundancia entre ambos canales. Por ejemplo, en muchas locuciones de alta fidelidad, existe un único y potente altavoz denominado "BOOMER". Sin embargo, se tiene la impresión de que el sonido proviene de diferentes fuentes como si existieran altavoces en todas las direcciones. En la realidad, por debajo de una frecuencia determinada, el oído ya no es capaz de localizar el origen espacial de los sonidos. De esta manera, algunas frecuencias se pueden grabar como señal monofónica seguida por un pequeño código para lograr restaurar un pequeño porcentaje de espacialización en la decodificación.

Las bandas críticas y el bark.

Bandas críticas.

Estudios de la discriminación en frecuencia del oído han demostrado que en las bajas frecuencias, tonos con unos cuantos hercios de separación pueden ser distinguidos; sin embargo, en las altas frecuencias para poder discriminar los tonos se necesita que estén separados por cientos de hercios. En cualquier caso, el oído responde al estimulo más fuerte que se presente en sus diferentes regiones de frecuencia; a este comportamiento se le da el nombre de bandas críticas. Los estudios muestran que las bandas críticas son mucho más estrechas en las bajas frecuencias que en las altas; el 75 % de las bandas críticas están por debajo de los 5 KHz, lo que implica que el oído recibe más información en las bajas que en las altas frecuencias. Las bandas críticas tienen un ancho de aproximadamente 100 Hz para las frecuencias de 20 a 400 Hz; este ancho aumenta de manera logarítmica a medida que aumenta la frecuencia. Se ha comprobado que el ancho de las bandas críticas se puede aproximar con la fórmula:

$$\textit{Ancho de la banda crítica (Hz)} = 24.7\ (4.37\mathrm{F} + 1)$$

F es la frecuencia central en KHz.

Las bandas críticas son comparables a un analizador de espectro con frecuencia central variable. Más importante aún es el hecho de que las bandas críticas no son fijas; son continuamente variables en frecuencia y cualquier tono audible creará una banda crítica centrada en él. Mirado desde otro punto de vista, el concepto de la banda crítica es un fenómeno empírico: una banda crítica es el ancho de banda al cual las respuestas subjetivas cambian abrúptamente.

El bark.

El bark (en honor al físico alemán Georg Heinrich Barkhausen) es la unidad de frecuencia perceptual; específicamente, un bark mide la tasa de banda crítica, o sea, una banda crítica tiene un ancho de un bark. La escala bark relaciona la frecuencia absoluta (en Hz) con las frecuencias medidas perceptualmente (el caso de las bandas críticas). Usando el bark, un sonido en el dominio de la frecuencia puede ser convertido a sonido en el dominio psicoacústico. De esta manera, un tono puro (representado por una componente en el dominio de la frecuencia) puede ser representado como una curva de enmascaramiento psicoacústico. Eberhard Zwicker modeló el oído con 24 bandas críticas arbitrarias para frecuencias por debajo de 15 KHz, con una banda adicional que ocupa la región entre 15 y

20 KHz. El bark (ancho de una banda crítica) puede calcularse con las siguientes fórmulas:

$$1\ bark\ (Hz) \simeq \frac{f}{100} \qquad Para\ f < 500\ Hz$$
$$1\ bark\ (Hz) \simeq 9 + 4\ \log(\frac{f}{100}) \qquad Para\ f > 500\ Hz$$
$$f = frecuencia.$$

Nota: Para determinar el ancho de una banda crítica, se pueden usar estas fórmulas o la fórmula mostrada anteriormente (*Bandas criticas*).

De las consideraciones anteriores, se deduce que el umbral de enmascaramiento es diferente cuando se tienen en cuenta las bandas críticas. El umbral sin tener en cuenta las bandas críticas y teniéndolas sería, respectivamente:

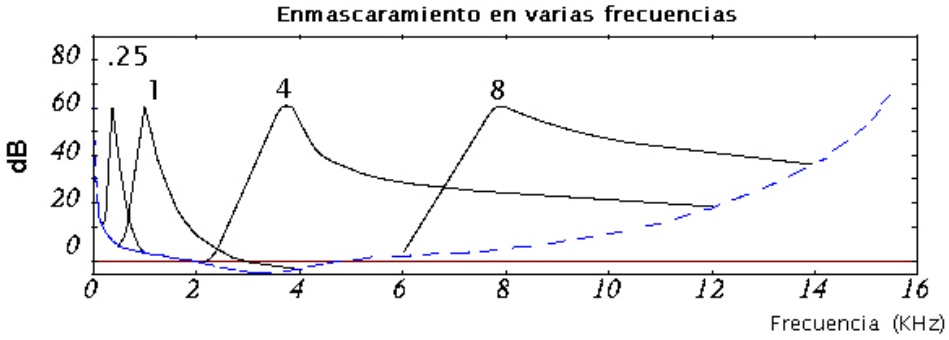


Figura 1.11: Enmascaramiento en varias frecuencias.

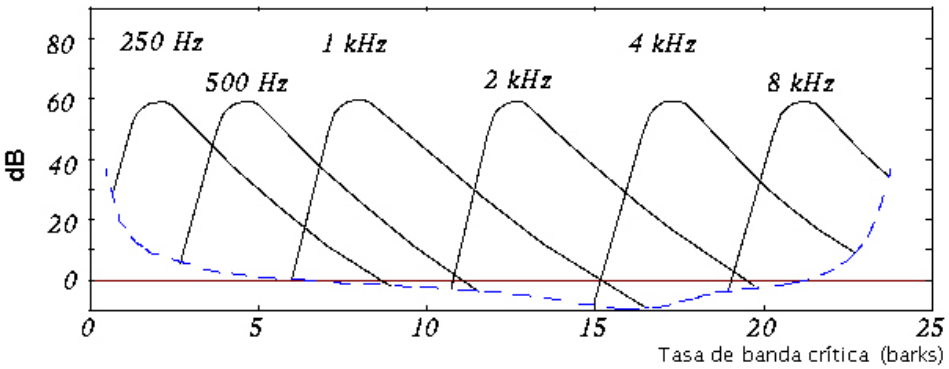


Figura 1.12: Enmascaramiento con bandas críticas.

Capítulo 2

Hardware

En este capítulo se van a describir los componentes hardware de los que consta el proyecto. Primero veremos las distintas partes de las que se compone y luego iremos describiéndolas una a una en profundidad. A continuación puede verse de forma muy genérica,

el diagrama de bloques correspondiente al reproductor de MP3:

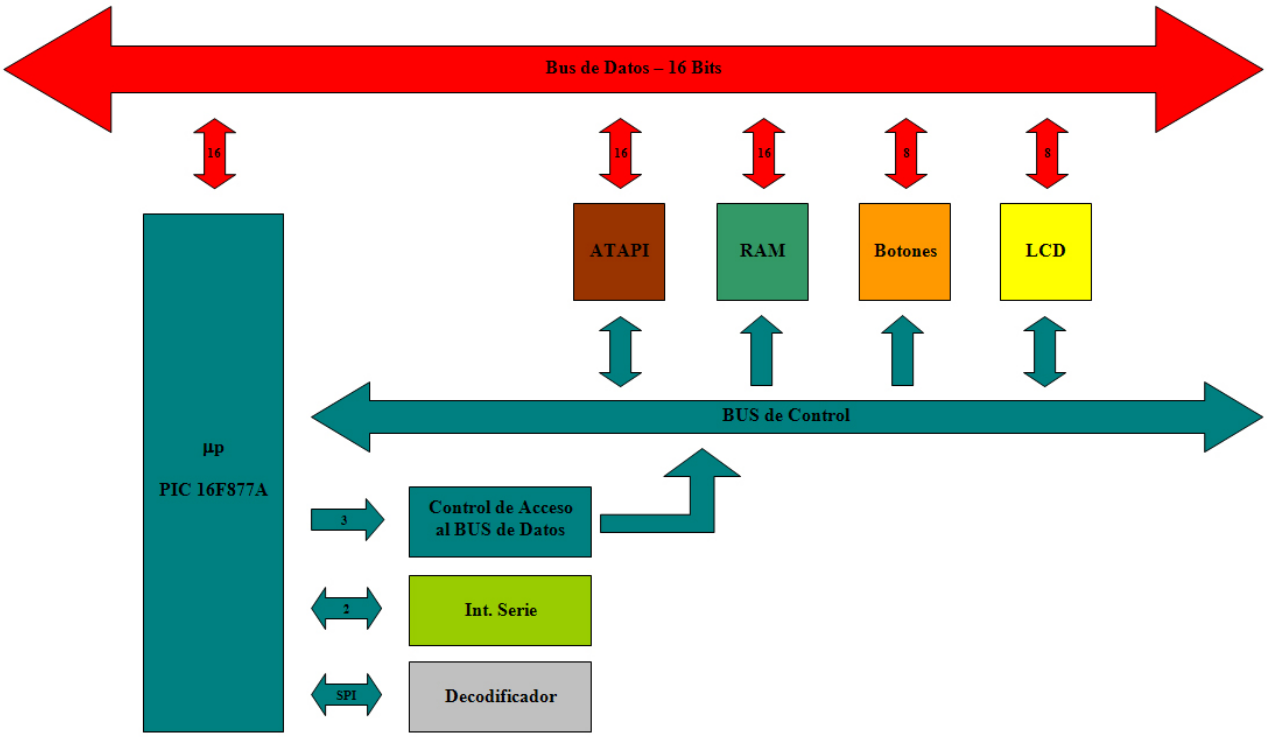


Figura 2.1: Diagrama de bloques

Como vemos esta formado por varios bloques diferenciados, cada uno de los cuales se conecta tanto al BUS de datos como al BUS de control. Su cometido es el siguiente:

- **ATAPI:** Se encarga de facilitar la interconexión con el dispositivo IDE. Contiene la circuitería necesaria para acceder correctamente al disco duro.
- **RAM:** Está formado por los circuitos de memoria externa SRAM. Ofrece el soporte necesario para almacenar los buffers de reproducción y la caché de disco.
- **Botones:** Junto con el display LCD conforma la interfaz de usuario su misión es la de recoger sus decisiones y comunicárselas al controlador.
- **LCD:** En el se muestran desde el nombre de las canciones, hasta el estado de la reproducción.
- **Control de Acceso al BUS de Datos:** Circuitería externa que se encarga de dar acceso a los diferentes bloque al BUS de Datos, de forma que no se produzcan interferencias entre ellos.
- **Interfaz Serie:** Adapta los niveles de tensión TTL a la norma EIA-232 utilizada por el PC.
- **Decodificador:** Este bloque se encarga de realizar la descompresión de las canciones y de su amplificación.

2.1. La circuitería de control y el BUS de Datos

La parte central del dispositivo, es, sin duda alguna, la circuitería de control. Es la encargada de albergar el código de programa y por lo tanto, es la que decide en cada momento que es lo que hay que hacer. Está formada por el microcontrolador de la casa Microchip Technology PIC 16F877A y por una serie de circuitos lógicos que le sirven de apoyo.

PIC 16F877A

Este microcontrolador tiene una serie de características que lo hacen muy adecuado para este proyecto, entre las más importantes están:

- Procesador RISC *(35 instrucciones)*
- Velocidad de reloj de 20 MHz
- Dispone de 8K words de memoria de programa FLASH

- Maneja Inerrupciones
- Soporta ICSP *(In circuit serial programing)*
- 3 temporizadores
- SSP con I2C y SPI
- Hasta 5 puertos de entrada salida
- Bajo coste

A continuación podemos ver el patillaje de este microcontrolador en su versión DIP.

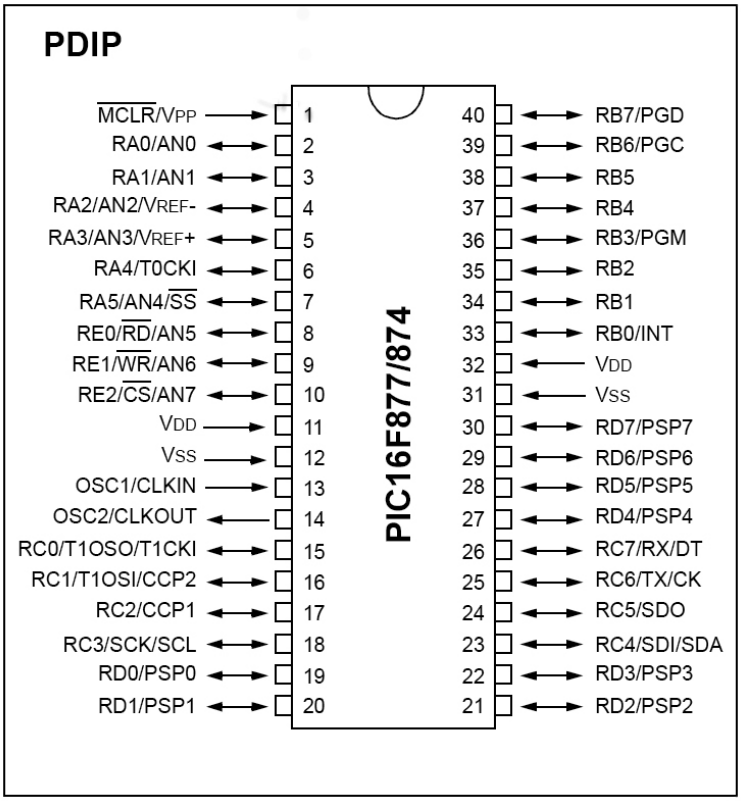


Figura 2.2: Patillaje del PIC 16F877A

74HC138

Este integrado forma parte de la circuitería de control externa que da acceso a los diferentes bloques al BUS de datos. Es un decodificador que activa una de sus ocho salidas en función de lo que tenga en la entrada. Sus características más notables son:

- Salidas compatibles TTL, CMOS y NMOS
- Rango de funcionamiento de 2 a 6 voltios
- Alta inmunidad al ruido

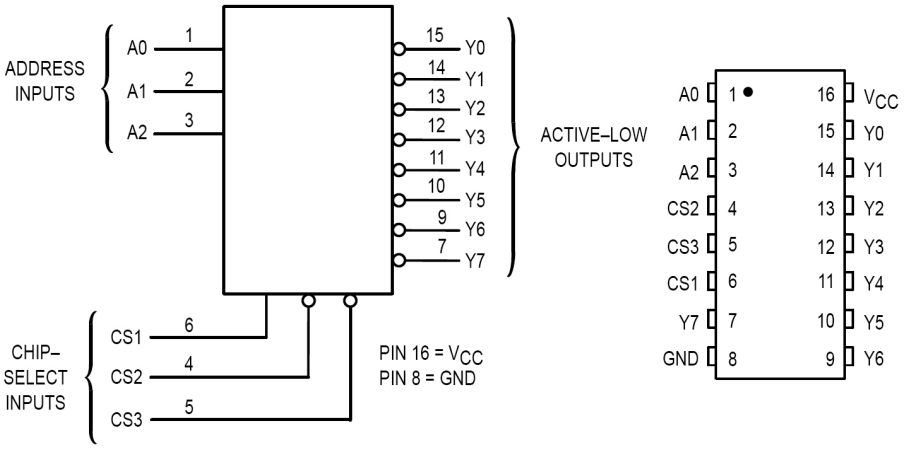


Figura 2.3: Diagrama lógico y patillaje del 74HC138

Como podemos ver, las salidas que ofrece este decodificador están complementadas, es decir, son activas a nivel bajo, lo cual nos bien muy bien, ya que las señales de habilitación suelen ser también activas a nivel bajo.

74HC04

Desgraciadamente algún módulo, como por ejemplo el LCD tiene su línea de activación activa a nivel alto. Se hace necesaria la inclusión de un inversor tras la salida correspondiente del decodificador, para que adquiera el nivel adecuado. En cuanto a sus características eléctricas, este integrado es similar al decodificador ya visto.

2.1.1. Circuito de control

En el esquema completo del circuito de control podemos ver claramente ya todas las líneas de control.

De los cinco puertos de los que dispone el microcontrolador, se usan el **A** y el **E** para generar todas las señales de control. El decodificador se controla a través del **puerto A**,

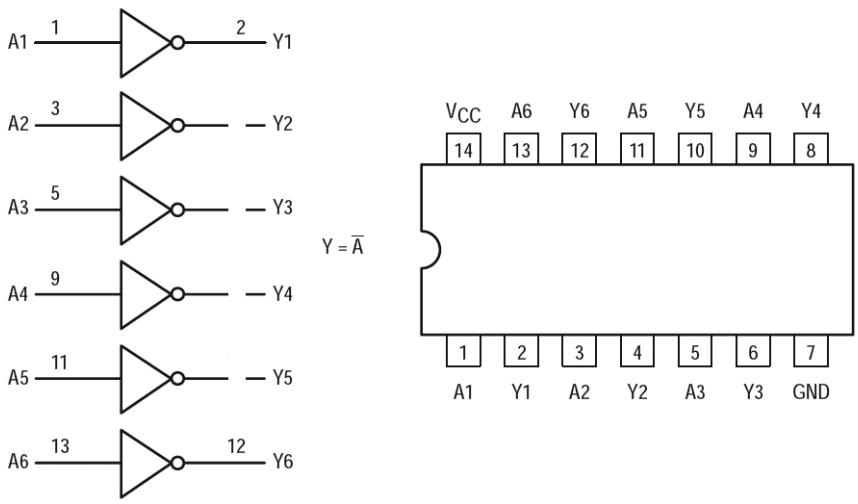


Figura 2.4: Diagrama lógico y patillaje del 74HC04

es el encargado de dar acceso a los distintos módulos al BUS de datos, generando las correspondientes señales de habilitación.

- $\overline{DIOR}$: Activa la lectura del disco IDE.
- $\overline{OE}$: Activa la lectura de memoria SRAM.
- $\overline{LATCH\_MEM\_L}$: Activa la carga del LATCH de memoria.
- LCD_E: Activa el Display LCD.
- $\overline{BUTTON\_E}$: Activa la lectura de los botones.
- $\overline{SERIAL\_E}$: Activa el interfaz serie con el PC.
- $LATCH\_IDE\_L$: Activa la carga del LATCH de disco IDE.

El **puerto A** también controla otra línea de habilitación que no sale del decodificador:

- $\overline{DIOW}$ Activa la escritura en el disco IDE.

Por su parte, el **puerto E** controla dos líneas de habilitación:

- $\overline{WE}$: Activa la escritura en la memoria SRAM.
- INC_ADDR: Activa el incremento del LATCH de memoria SRAM.

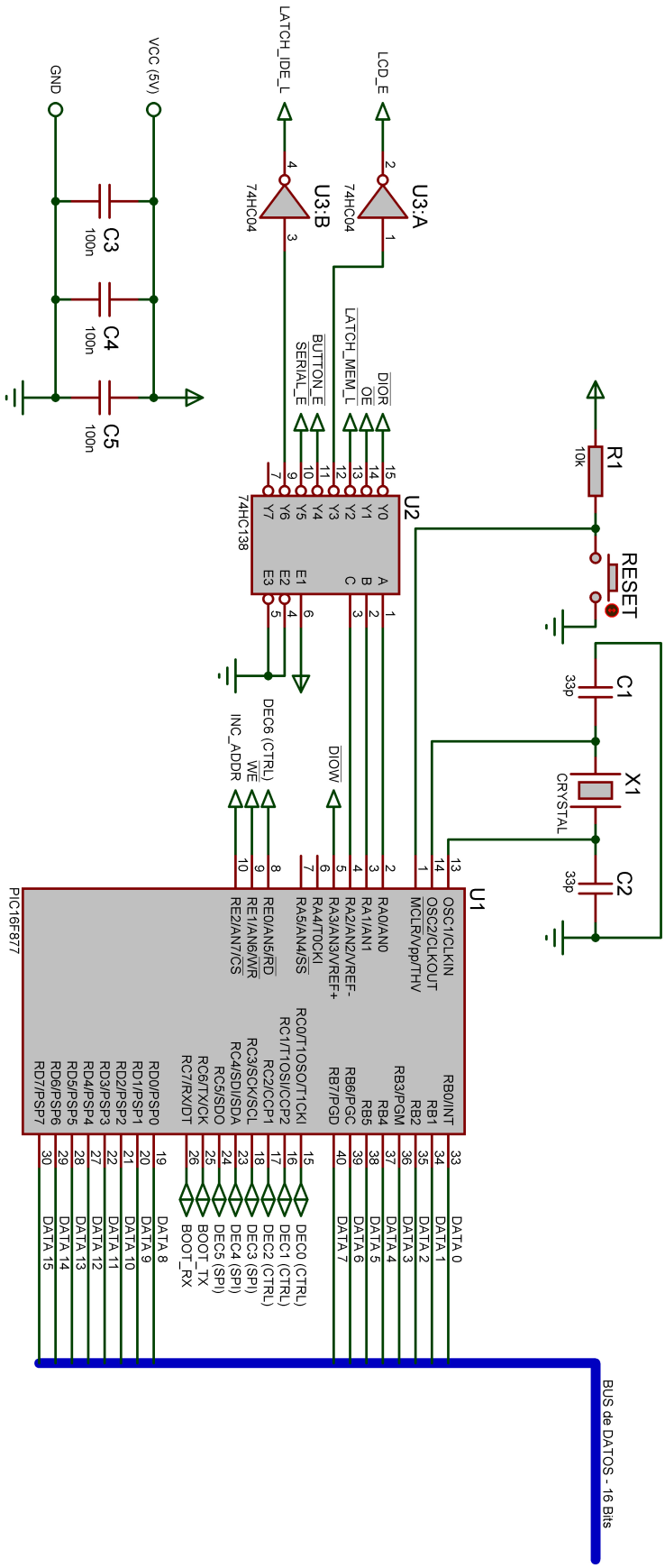


Figura 2.5: Esquema del circuito de control

Como vemos, hay tres líneas de habilitación que no parten del decodificador. El motivo de esto, es que hay dos ocasiones en las que hay que dar acceso a dos módulos al BUS de datos de forma simultánea:

- Transferencia directa de datos del disco IDE a la memoria SRAM. Activamos $\overline{DIOR}$, $\overline{WE}$ y en su caso INC_ADDR.
- Transferencia directa de datos de la memora SRAM al disco IDE.Activamos $\overline{DIOW}$, $\overline{OE}$ y en su caso INC_ADDR.

El **puerto C** se usa, junto con una línea del **puerto E** $DEC0 - DEC6$, para controlar y comunicarse con el módulo decodificador.

El **puerto C** también controla las líneas que van al módulo de Interfaz Serie $BOOT\_TX$ y $BOOT\_RX$.

2.1.2. BUS de Datos

El BUS de Datos es de 16 Bits, y como vemos se corresponde con los **puertos B y D** del microcontrolador. El **puerto B** maneja los ocho bits de menor peso, $DATA0 - DATA7$, mientras que el **puerto D** se encarga de los ocho de mayor peso $DATA8 - DATA15$.

2.1.3. Componentes adicionales

El resto de componentes que pueden verse son:

- **R1** *(10KΩ)* y **Pulsador**: RESET del microcontrolador.
- **C1** *(33pF) Cerámico*, **C2** *(33pF) Cerámico* y **X1** *(20MHz)*: Oscilador del microcontrolador.
- **C3** *(100nF/50V)*, **C4** *(100nF/50V)* y **C5** *(100nF/50V)*: Condensadores de desacoplo.

2.2. El Interfaz Serie

A pesar de que el microcontrolador dispone de un puerto serie integrado, es necesaria algo de circuitería externa para acondicionar los niveles lógicos de las líneas, al estándar EIA-232, que es el que usa el PC.

MAX232

Este integrado es el que se encarga de realizar el cambio de niveles. Sus características principales son:

- Tan sólo requiere una fuente de alimentación de 5V.
- Tiene 2 transmisores y dos receptores.
- Opera correctamente hasta velocidades de 120kbits/s.

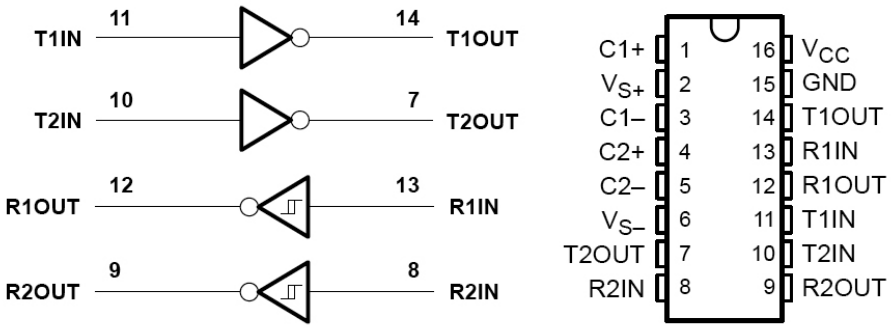


Figura 2.6: Diagrama lógico y patillaje del MAX232

Como podemos ver, este integrado requiere el uso de condensadores externos para poder generar los niveles de tensión adecuados.

74HC245

Es un transceptor de BUS no inversor de tres estados. Lo usamos para aislar las líneas del puerto serie del microcontrolador del adaptador de niveles. Sus características principales son:

- Salidas compatibles TTL, CMOS y NMOS.
- Rango de funcionamiento de 2 a 6 voltios.
- Baja corriente de entrada (1 μA).
- Alta inmunidad al ruido.

En el esquema completo del interfaz se pueden observar las siguientes líneas:

- *BOOT_TX*: Trasmisión de datos series desde el microcontrolador al PC.

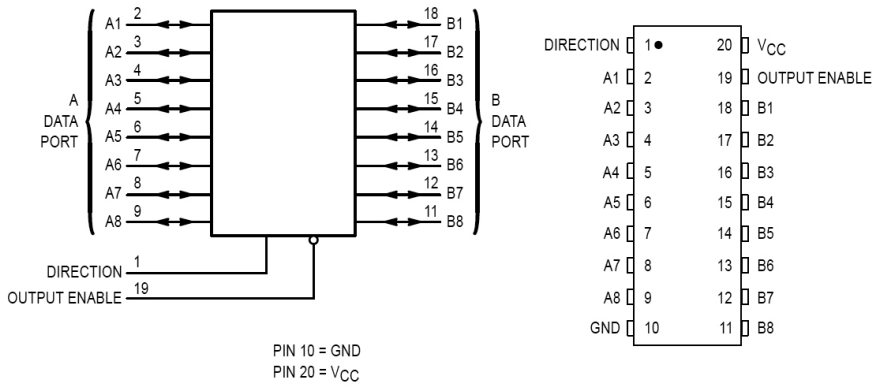


Figura 2.7: Diagrama lógico y patillaje del 74HC245

- *BOOT_RX*: Recepción de datos series desde el PC al microcontrolador.
- $\overline{SERIAL\_E}$: Habilitación del interfaz.

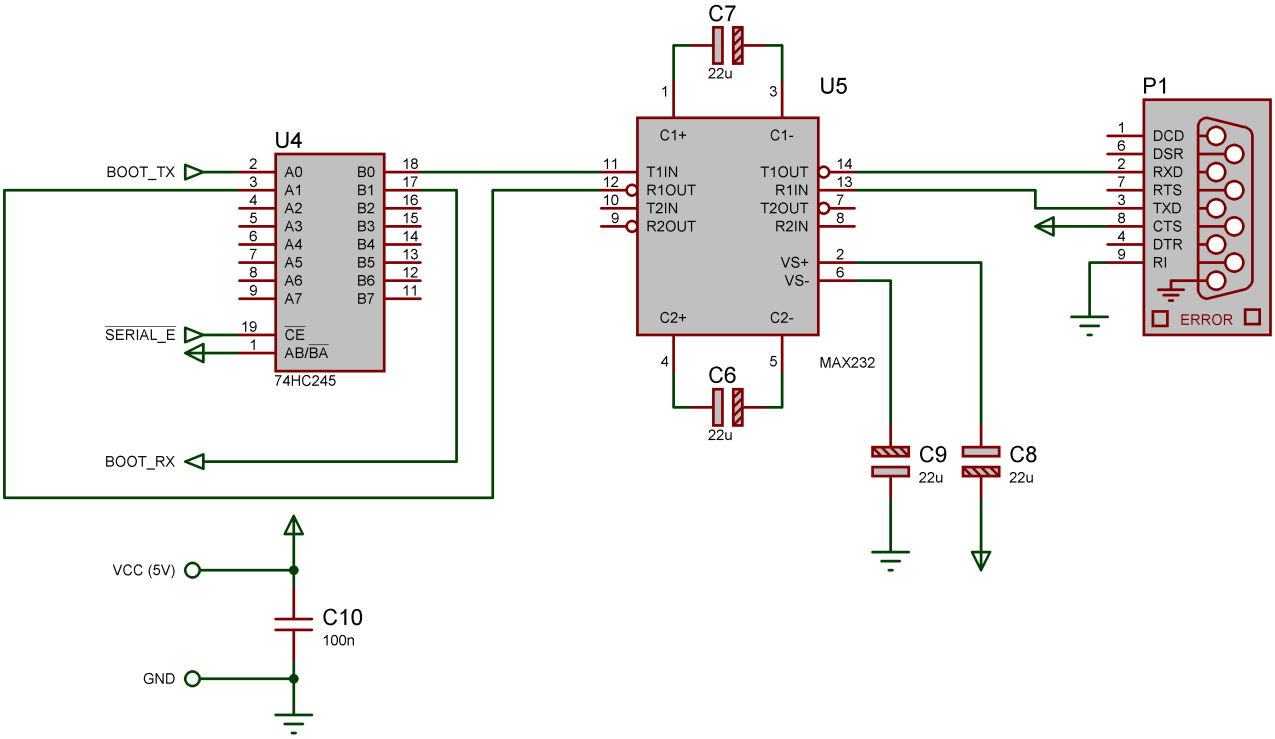


Figura 2.8: Esquema del Interfaz Serie

2.2.1. Componentes adicionales

El resto de componentes que conforman el Interfaz son:

- **C6** *(22μF/25V) Electrolítico*, **C7** *(22μF/25V) Electrolítico*, **C8** *(22μF/25V) Electrolítico* y **C9** *(22μF/25V) Electrolítico*: Condensadores necesarios para generar los niveles adecuados.
- **C10** *(100nF/50V)*: Condensador de desacoplo.

2.3. Interfaz ATAPI

El módulo ATAPI es el encargado de conectarse directamente al dispositivo IDE *(Disco Duro)* por medio de un conector de 40 pines. Almacena los comandos enviados por el microcontrolador y los ejecuta cuando este se lo pide.

2.3.1. AT Attachment ATA-1

El interfaz ATA se ha convertido en el estándar de la industria para la conexión de unidades de disco al PC.

En un principio fue desarrollado para facilitar la conexión de unos cuantos dispositivos de manea simple y barata. Sin embargo, el gran incremento en el desarrollo de programas multimedia distribuidos en CD-ROM y sobre todo el incremento de velocidad de transferencia de los dispositivos, ha creado la necesidad de actualizar continuamente este estándar para adaptarse a las nuevas tecnologías. Actualmente la revisión del estándar se encuentra en su versión ATA/ATAPI-8.

En el se definen los conectores y cables para la conexión física entre el anfitrión *(Host)* y el dispositivo de almacenamiento, así como las características eléctricas y lógicas de las líneas que se emplean para comunicarlos. También se definen los registros operacionales dentro del dispositivo de almacenamiento, y las órdenes y protocolos para su correcto funcionamiento.

En este apartado tan sólo nos centraremos en lo que es estrictamente la conexión Hardware, hablaremos de como han de ser los conectores y de que señal se envía por cada uno de los pines. La parte correspondiente a los registros operacionales, comandos y protocolos, se vera en el correspondiente apartado de Software.

El interfaz que vamos a implementar es el ATA-1, ya que las funciones que este ofrece son más que suficientes para el propósito de este proyecto.

Conectores

El conector que define el estándar para unir el dispositivo IDE al Host, tiene que ser de tipo DIL de 40 contactos.

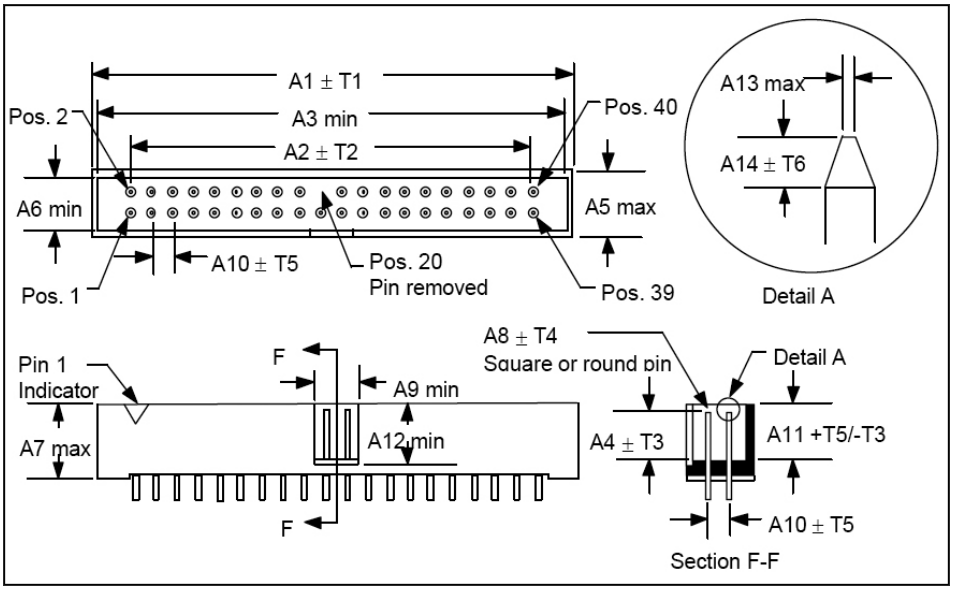


Figura 2.9: Conector ATA (Host) - DIL 40

Como vemos, el estándar prevé la eliminación del pin 20 en el conector, que se corresponderá con la ausencia de agujero para el en el cable, de forma que sea imposible conectarlo del revés.

Líneas de interés

No es necesario usar todas las señales para manejar el dispositivo, algunas de ellas sólo se usan en aplicaciones especiales. Las señales que vamos a usar en este proyecto son:

- Las líneas de selección de chip: $\overline{CS0}$-$\overline{CS1}$. Sirven para seleccionar el **bloque de comandos**, o el **bloque de registros de control**.
- Las líneas de dirección de registro: $DA0 - DA2$. Código de 3 bits mediante el cual se indica el registro al que se quiere acceder.
- Las líneas de lectura y escritura:$\overline{DIOR}$ y $\overline{DIOW}$.
- Las líneas de Datos: $DD0 - DD15$.

Signal name	Connector contact	Conductor		Connector contact	Signal name
RESET-	1	1	2	2	Ground
DD7	3	3	4	4	DD8
DD6	5	5	6	6	DD9
DD5	7	7	8	8	DD10
DD4	9	9	10	10	DD11
DD3	11	11	12	12	DD12
DD2	13	13	14	14	DD13
DD1	15	15	16	16	DD14
DD0	17	17	18	18	DD15
Ground	19	19	20	20	(keypin)
DMARQ	21	21	22	22	Ground
DIOW-:STOP	23	23	24	24	Ground
DIOR-:HDMARDY-:HSTROBE	25	25	26	26	Ground
IORDY:DDMARDY-:DSTROBE	27	27	28	28	CSEL
DMACK-	29	29	30	30	Ground
INTRQ	31	31	32	32	Obsolete (see note)
DA1	33	33	34	34	PDIAG-:CBLID-
DA0	35	35	36	36	DA2
CS0-	37	37	38	38	CS1-
DASP-	39	39	40	40	Ground
NOTE – Pin 32 was defined as IOCS16 in ATA-2, ANSI X3.279-1996.					

Figura 2.10: Conector ATA - Correspondencia PIN-Línea

Conector del Cable IDE

El conector que se une a la base DIL-40, está también definido por el estándar.

Como ya hemos dicho, la ausencia de PIN en la posición numero 20 de la base, tiene que corresponderse con la ausencia de agujero en el lado del conector, de forma que no se puedan acoplar del reves.

Aun así el estándar prevé, de forma opcional, colocar una muesca en uno de los lados del conector, que, de nuevo se corresponde con una hendidura en la base, para evitar que sea insertado del revés.

2.3.2. Dispositivo IDE - Disco duro

Es el dispositivo de almacenamiento externo que usa el reproductor y del cual extrae las canciones comprimidas.

Por su parte posterior posee un conector DIL-40 igual al que se encuentra en el Host y que le sirve para comunicarse co él a través del cable IDE. Junto a el podemos podemos encontrar una serie de pines que sirven para configurar el dispositivo como maestro o esclavo y un conector más ancho de 4 pines por el que recibe la alimentación.

El conector de alimentación es del tipo molex y por el recibe dos tensiones, una de 12V y otra de 5V.

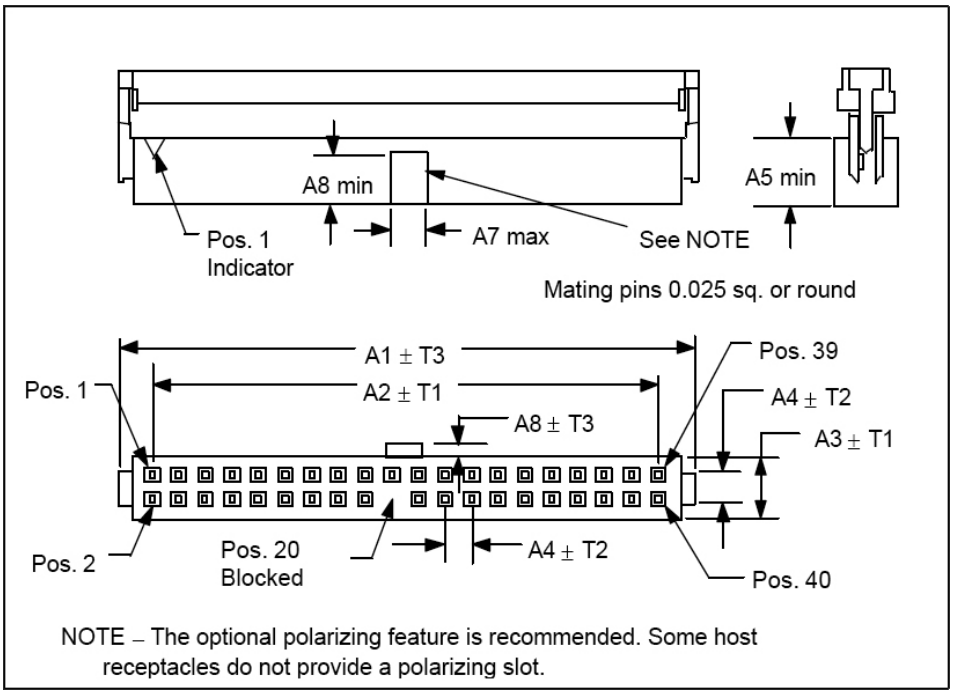


Figura 2.11: ATA - Conector del cable IDE

- El **Pin 1** se corresponde con una tensión de **+12 Voltios**.
- Por el **Pin 4** se recibe la alimentación de **+5 Voltios**.
- Los dos pines centrales, el **2** y el **3**, son las tensiones de retorno del **1** el **4**, es decir **GND**.

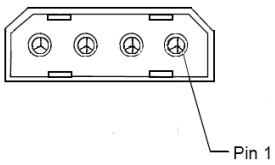


Figura 2.12: Terminal de alimentación

Para su correcto funcionamiento el dispositivo debe ser configurado como maestro. Para ello, basta con colocar un jumper en el lugar adecuado.

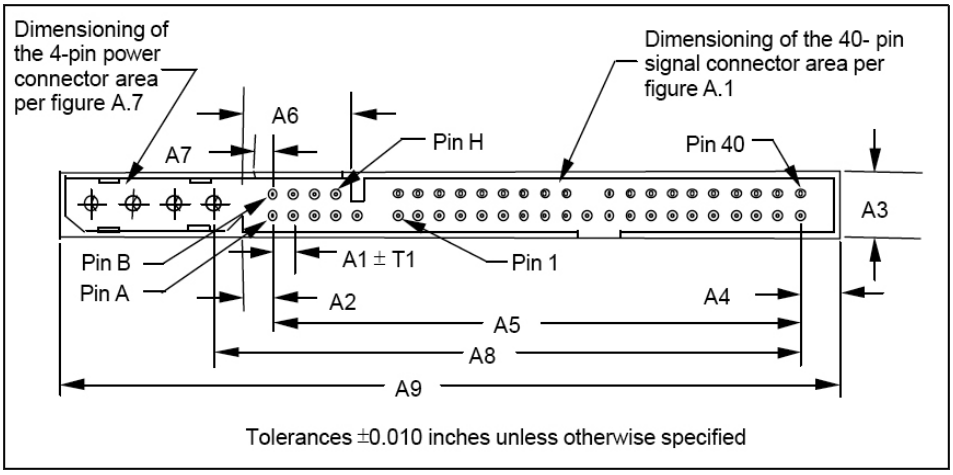


Figura 2.13: Conector del dispositivo IDE

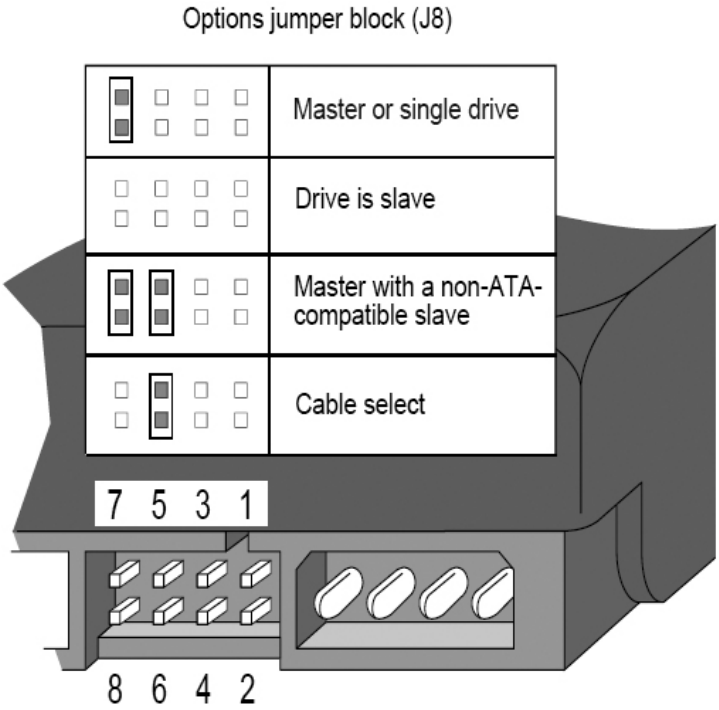


Figura 2.14: Configuración del dispositivo IDE

2.3.3. Componentes adicionales

El proceso para acceder al dispositivo IDE es muy sencillo. Basta con seleccionar el registro al que se quiere leer o escribir por medio de las líneas $DA0$-$DA2$ *(Selección de registro)* y de $CS0$-$CS1$ *(Selección de banco)*, poner el dato que se quiere leer o escribir en las líneas de datos $DD0$-$DD15$ y activar la lectura o escritura del dispositivo $\overline{DIOW}$ o $\overline{DIOR}$. Sin embargo, es necesario que el dato a leer o escribir y la selección de registro estén presentes en las líneas simultáneamente cuando se active la señal de lectura o escritura. Ello nos obliga a utilizar un LATCH que mantenga la selección de registro y su banco, mientras que ponemos el dato en el BUS, ya que todos ellos se cargan a través del BUS de datos.

74HC573

Este integrado es un LATCH transparente de ocho bits. Sus características más importantes son similares a las de otros dispositivos de la serie 74HC.

- Salidas compatibles TTL, CMOS y NMOS.
- Rango de funcionamiento de 2 a 6 voltios.
- Baja corriente de entrada (1 μA).
- Alta inmunidad al ruido.

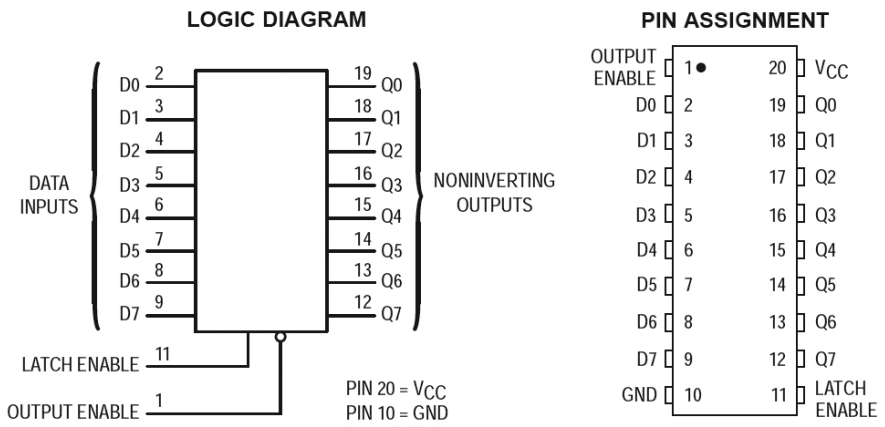


Figura 2.15: Diagrama lógico y patillaje del 74HC573

El resto de los componentes necesarios para completar este módulos son:

- **C11** *(100nF/50V)*: Condensador de desacoplo.

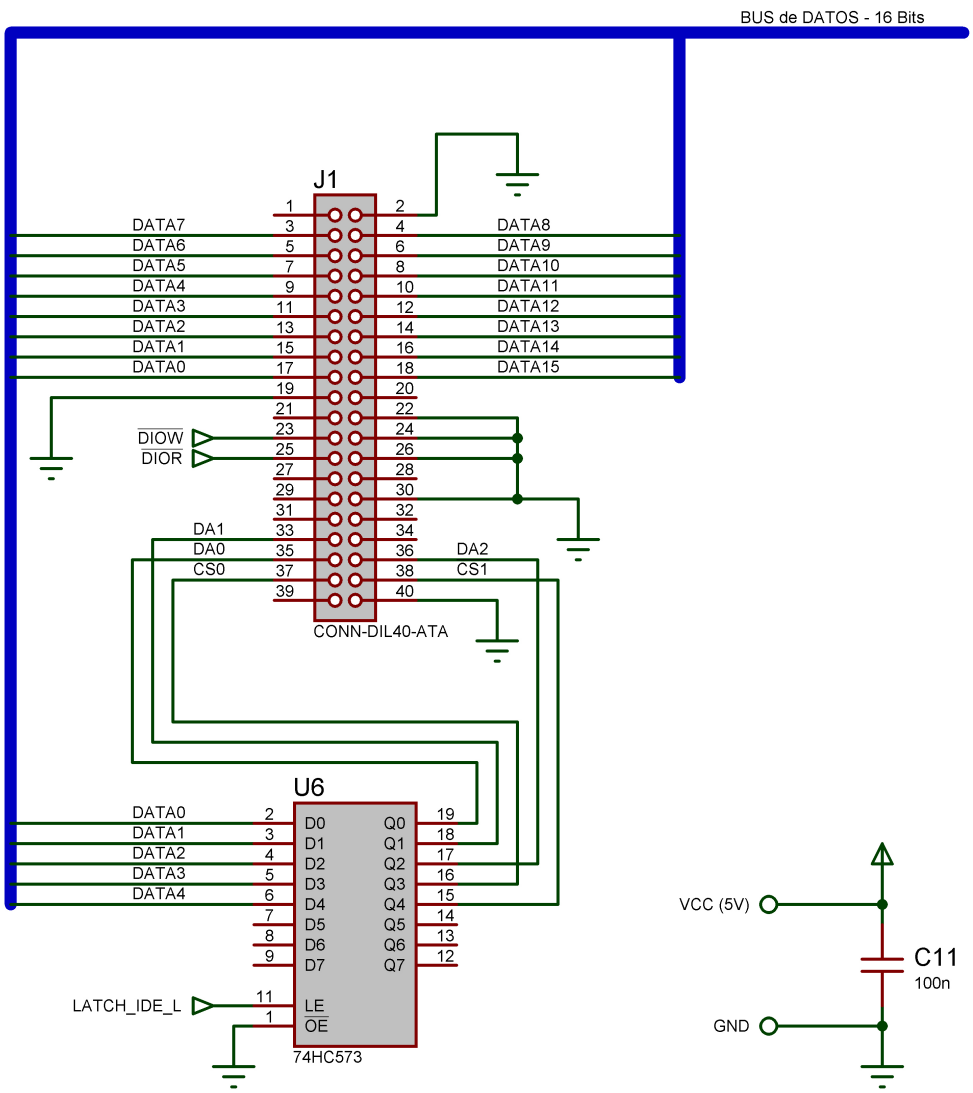


Figura 2.16: Esquema del módulo ATAPI

Podemos observar como este módulo se controla con tan sólo 3 líneas:

- $\overline{DIOW}$: Orden de escritura del dispositivo IDE conectado.
- $\overline{DIOR}$: Orden de lectura del dispositivo IDE conectado.
- $LATCH\_IDE\_L$: Carga del registro a leer o escribir.

Esto es posible gracias a que el propio dispositivo, deja en alta impedancia el BUS de Datos, cuando no están activas $\overline{DIOW}$ o $\overline{DIOR}$.

2.4. Memoria RAM

Al usar un disco duro como fuente de datos y debido a las latencias en el flujo de datos que esto conlleva, se hace necesaria la inclusión de una memoria RAM *(SRAM)* externa, que sirva como **buffer intermedio** entre el disco y el decodificador, de forma que siempre tengamos datos disponibles datos para ser procesador y que no se detenga la reproducción.

La memoria RAM, además nos servirá para almacenar otra serie de datos de gran interés, como:

- Caché de la FAT.
- Almacenamiento de sectores de disco para su posterior tratamiento.
- Almacenamiento de los nombres de archivo y directorio. *(En FAT32 pueden tener hasta 255 caracteres.)*

CY62256

Como chips de memoria, usaremos el model de la casa CYPRESS CY62256.

Estos chips de memoria, son del tipo SRAM, y tiene una capacidad de 32Kx8bits. Como nuestro BUS de Datos es de 16 bits, usaremos dos módulos, de forma que tengamos una memoria total de 32Kx16bits.

Las características más importantes de estos integrados son:

- Funcionan con una alimentación de entre 4.5 y 5.5 voltios.
- Sus salidas son compatibles TTL.
- Tiempo de acceso de entre 55 y 70 ns.
- Facilidad para realizar expansiones de memoria por medio de sus líneas $\overline{CE}$ y $\overline{OE}$.

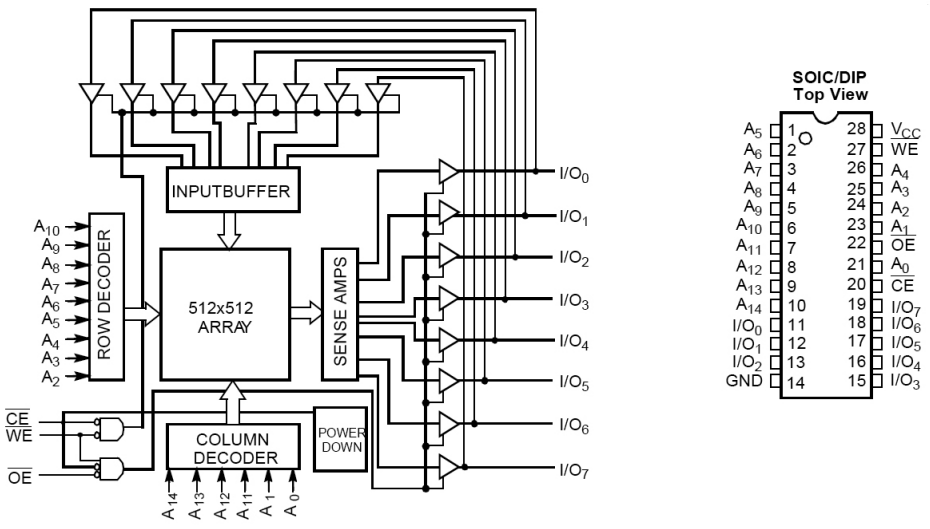


Figura 2.17: Diagrama lógico y patillaje del CY62256

74HC193

Cuando accedemos a la memoria SRAM, es necesario poner en los bits $A0$-$A14$ la dirección de memoria que queremos leer o escribir, y en los bits $I/O0$-$I/O15$ el dato. En nuestro caso tanto los bits de dirección como los de datos, salen del BUS de Datos, por lo que es necesario poner un LATCH que almacene la dirección para poder hacer uso de la memoria.

Si nos fijamos en el uso que le vamos a dar a la memoria, nos damos cuenta de que la mayoría de las veces vamos a acceder a posiciones consecutivas por lo que si como LATCH usamos un contador con carga en paralelo, basta una sola línea *(incremento del contador)* para cambiar la dirección. De esta forma conseguimos que las transferencias **Microcontrolador** $\rightleftarrows$ **SRAM** de varios datos seguidos sean muy rápidas.

El integrado 74HC193, es un contador de 4 bits con carga en paralelo y líneas de incremento y decremento. Podemos asociar fácilmente 4 de estos integrados, para construir un contador de 16 bits, que es lo que necesitamos ya que el BUS de dirección de memoria SRAM es de 15 bits.

Las características principales de este contador son:

- Salidas compatibles TTL.
- Rango de funcionamiento de 2 a 6 voltios.
- Baja corriente de entrada (1 μA).

- Carga en paralelo asíncrona.
- Borrado asíncrono.
- Facilidad de escalado.

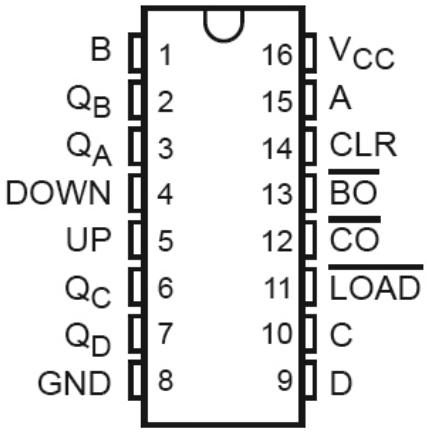


Figura 2.18: Patillaje del 74HC193

Como podemos ver en el esquema, el módulo de memoria se controla por medio de las líneas:

- $\overline{OE}$: Habilita la lectura de memoria.
- $\overline{OE}$: Habilita la escritura en memoria.
- $LATCH\_MEM\_L$: Carga de la dirección en el LATCH de memoria.
- INC_ADDR: Incrementa la dirección de memoria del LATCH.

Cuando las líneas de escritura y lectura de memoria $\overline{OE}$, $\overline{OE}$, están desactivadas *(1 lógico)*, el BUS de Datos se mantiene en alta impedancia, quedando libre para que lo use cualquier otro módulo.

2.4.1. Componentes adicionales

El resto de componentes que se pueden ver son:

- **C12** *(100nF/50V)*, **C13** *(100nF/50V)*, **C14** *(100nF/50V)*, **C15** *(100nF/50V)*, **C16** *(100nF/50V)*, **C17** *(100nF/50V)*: Condensadores de desacoplo.

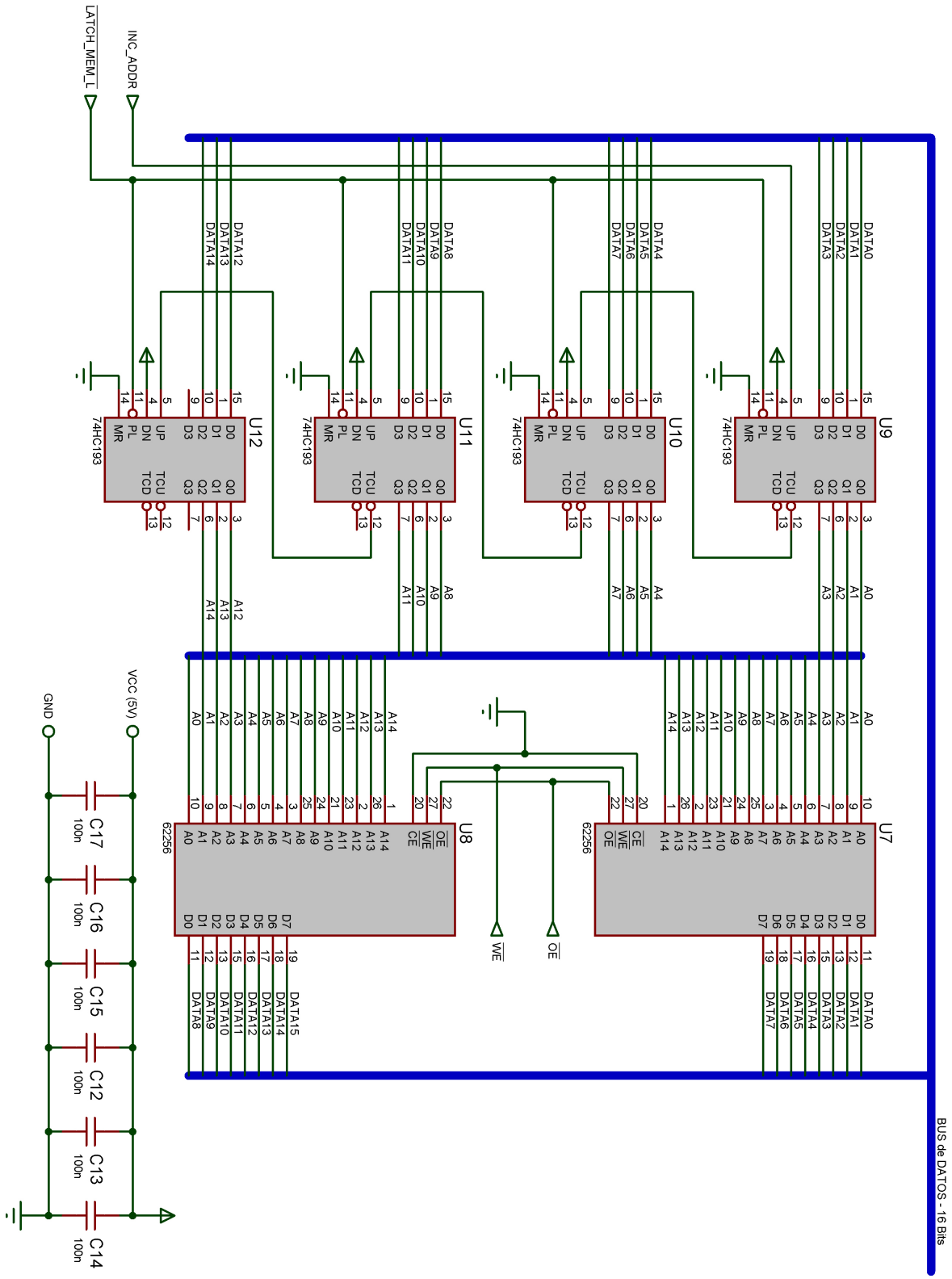


Figura 2.19: Esquema del módulo de memoria SRAM

2.5. Interfaz de usuario - Botones

La interfaz de usuario está formada por dos módulos, los botones y el display LCD.

Los botones se conectan al BUS mediante un transceptor de BUS *(74HC245)* igual al usado en el módulo de **Interfaz Serie**, por lo que no se va a volver a describir aquí.

Cuando se activa el transceptor, en el Byte bajo del BUS $DATA0$-$DATA7$, se recoge el estado de la pulsación o no de los botones. Un botón pulsado, se corresponde con la línea correspondiente del BUS de datos a 0 *(0 lógico)*, mientras que la ausencia de pulsación, aparece como un 1 *(1 lógico)*.

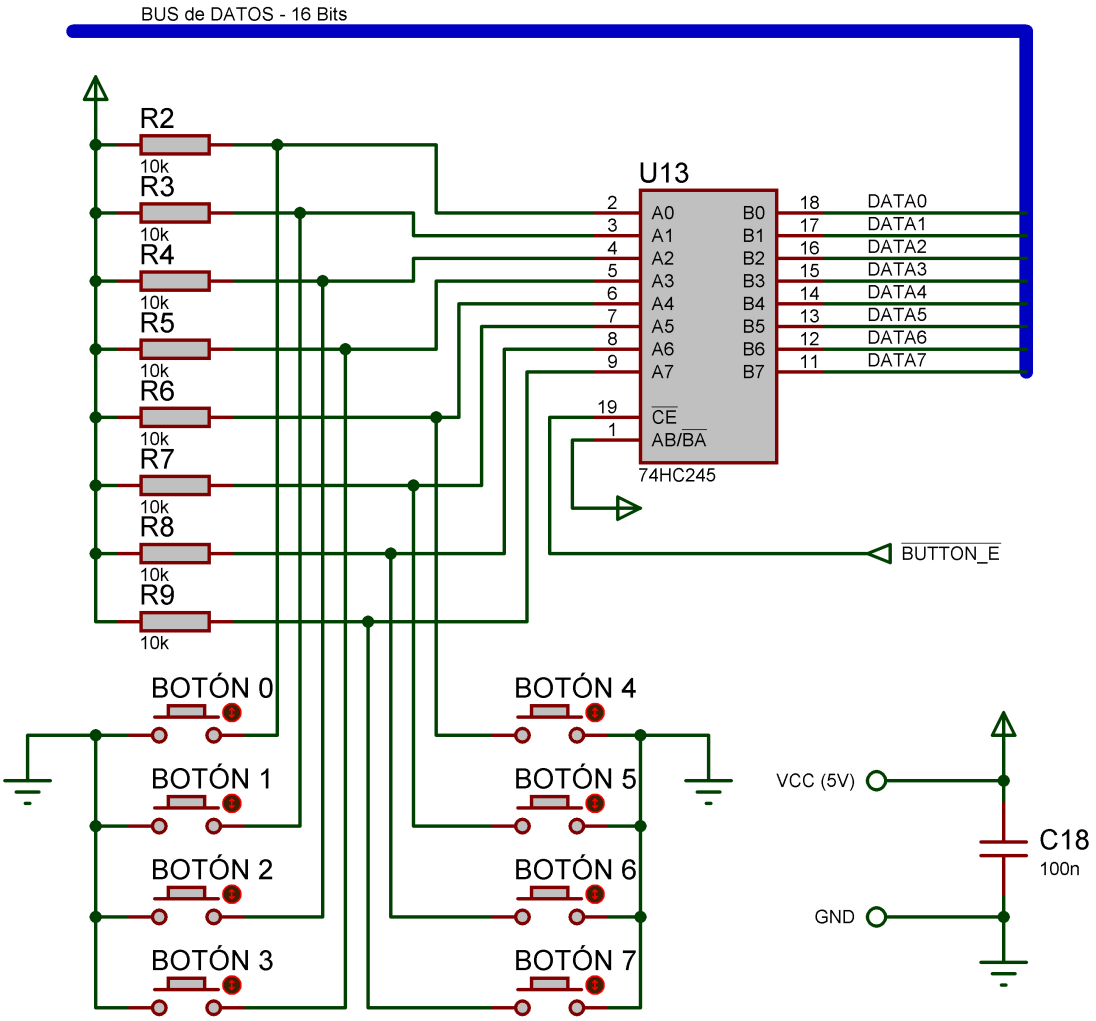


Figura 2.20: Esquema del módulo de Botones

El módulo de botones, se gobierna con tan sólo una línea de control que les da acceso al BUS de Datos.

- $\overline{BUTTON\_E}$: Habilitación del módulo de botones.

2.5.1. Componentes adicionales

Es necesario añadir una serie de resistencias para evitar provocar un cortocircuito cuando se produzca una pulsación en alguno de los botones.

- **R2** *(10KΩ)*, **R3** *(10KΩ)*, **R4** *(10KΩ)*, **R5** *(10KΩ)*, **R6** *(10KΩ)*, **R7** *(10KΩ)*, **R8** *(10KΩ)*, **R9** *(10KΩ)*: Resistencias de botón.
- **C18** *(100nF/50V)*: Condensador de desacoplo.

2.6. Interfaz de usuario - Display LCD

La otra parte de la interfaz de usuario, es el display de cristal liquido. En el se muestran los nombres de las canciones y los datos de reproducción entre otras cosas.

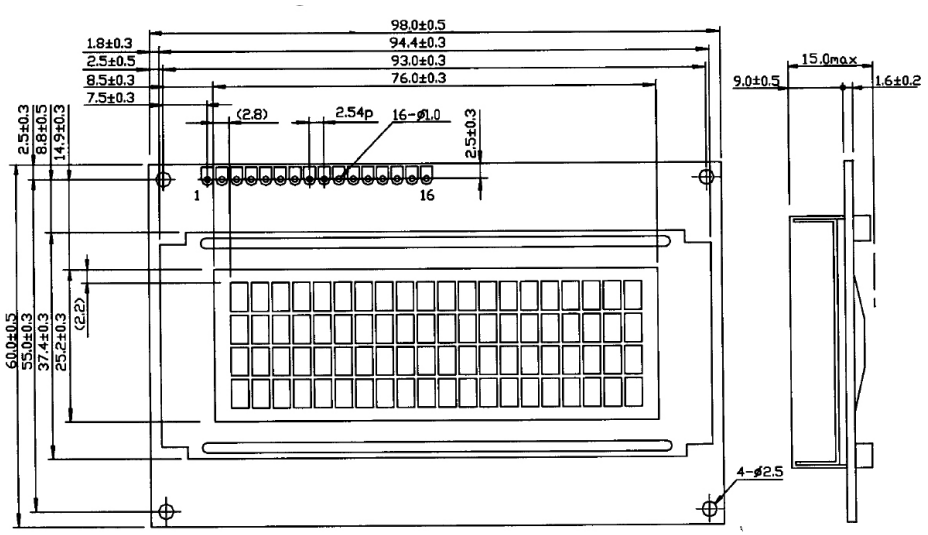


Figura 2.21: Display LCD

El display tiene un microcontrolador incorporado y por lo tanto la forma de escribir en el será mediante comandos que le serán enviados desde el PIC.

El microcontrolador del display es el HD44780U fabricado por Hitachi, sus características más relevantes son:

- Interfaz de 4 o de 8 Bits
- Velocidad máxima del BUS de 2MHz.
- Amplio y variado conjunto de instrucciones.
- Variado conjunto de fuentes.
- Bajo consumo.

Por su parte el Display fabricado por la casa Tianma es el modelo TM204AFF6 se caracteriza por:

- Tipo transmisivo/negativo.
- Caracteres amarillos sobre fondo azul.
- Iluminación por LED.
- Fuentes de 5x7 puntos *(Caracteres)* + 5x1 puntos *(Cursor)*
- Dispone de 4 líneas y 20 caracteres por línea.

El display se conecta a través de 16 pines.

Pin Nº	**Línea**	**Descripción**
1	Vss	Masa
2	Vcc	Alimentación
3	Vee	Contraste
4	RS	Selección de registro (H: Datos L: Instrucción)
5	R/W	Selecciona Lectura o Escritura
6	E	Habilitación
7	DB0	Bit de datos 0
8	DB1	Bit de datos 1
9	DB2	Bit de datos 2
10	DB3	Bit de datos 3
11	DB4	Bit de datos 4
12	DB5	Bit de datos 5
13	DB6	Bit de datos 6
14	DB7	Bit de datos 7
15	LED+	Alimentación para la iluminación
16	LED-	Masa para la iluminación

Cuadro 2.1: Descripción de líneas del Display

Como vemos, son necesarias tres líneas para controlar el display a parte del BUS de datos. La línea de habilitación, se corresponde con la línea LCD_E que sale del circuito de control. Las otras dos líneas RS y $R/\overline{W}$ las conectaremos a la parte alta del BUS de datos, ya que para escribir en el LCD sólo se usa la parte baja del mismo.

- $RS \rightarrow DATA8$
- $R/\overline{W} \rightarrow DATA9$

Cuando la línea LCD_E esta desactivada *(0 lógico)*, el display mantiene las líneas del BUS de Datos en alta impedancia.

Según esto, el esquema completo queda:

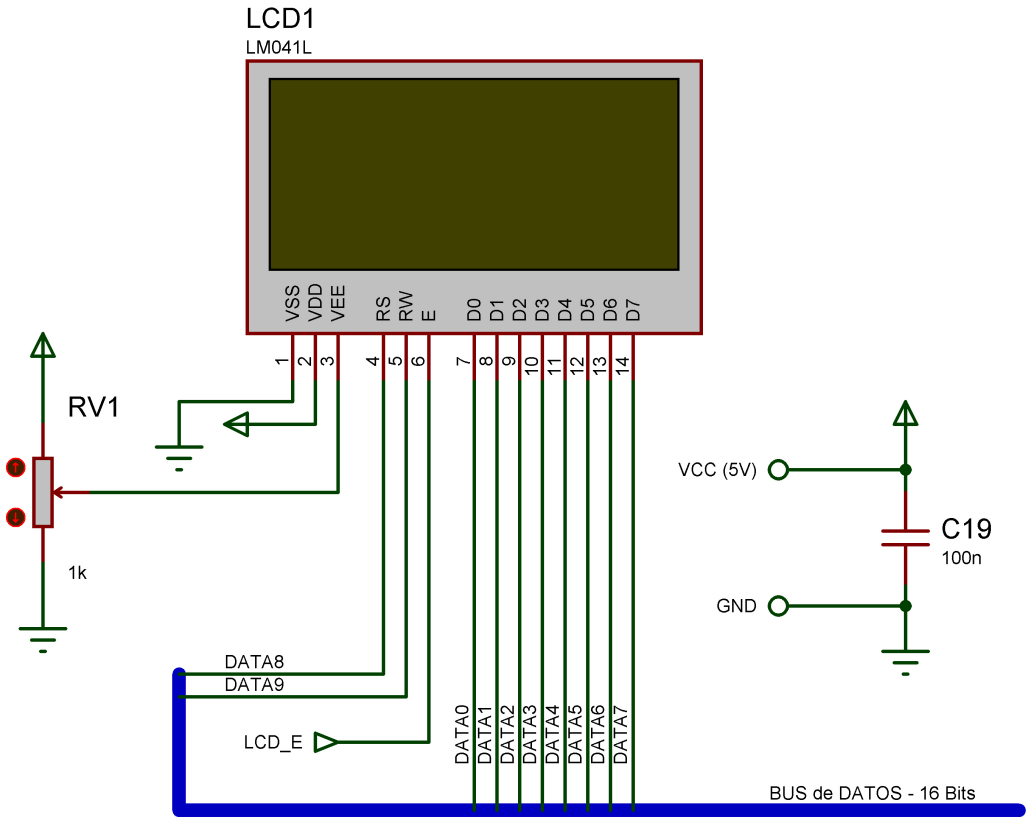


Figura 2.22: Esquema del módulo de Display

2.6.1. Componentes adicionales

En el esquema podemos observar un potenciómetro conectado a la línea de contraste. Su misión es regular el nivel de contraste pero su uso es opcional, se puede suprimir y conectar la línea Vee directamente a masa *(Contraste máximo)*.

- **RV1** *(1KΩ)*: Potenciómetro lineal.
- **C19** *(100nF/50V)*: Condensador de desacoplo.

2.7. Decodificador de MPEG-1 Layer III

El decodificador elegido para el proyecto es el VS1001K fabricado por VLSI Solution Oy. Este decodificador funciona con una tensión de 3.3 voltios. Esto nos obliga a tener que añadir circuitería externa para generar ese voltaje y además, deberemos adaptar las tensiones de las señales provenientes del microprocesador de 5 voltios a los 3.3 requeridos.

74LVC4245

Este integrado es un transceptor de BUS triestado de ocho bits, pero que además actúa como convertidor de niveles lógicos. Para ello cuenta con dos alimentaciones, una de ellas es la estándar de 5 voltios y la otra es la requerida de 3.3 voltios.

Sus características más importantes son:

- Alimentación de 5 y de 3.3 voltios.
- Salidas compatibles TTL.
- Bajo consumo.

Como vemos tiene dos pines de alimentación V_{CCA} y V_{CCB}. V_{CCA} se corresponde con los niveles de tensión presentes en las entradas/salidas A, mientras que V_{CCB} se corresponde con las B. Por lo demás, se maneja exactamente igual que cualquier otro transceptor de BUS.

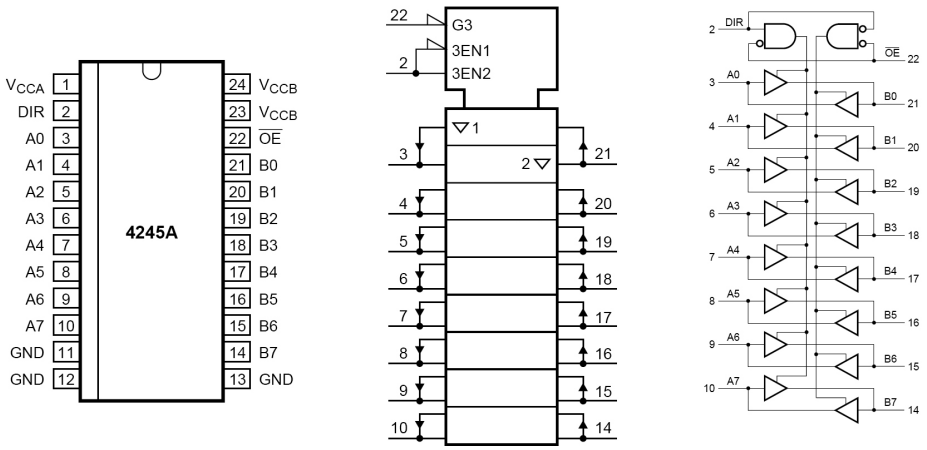


Figura 2.23: Diagrama lógico y patillaje del 74LVC245A

74HCT08

Este chip integra 4 puertas lógicas AND de dos entradas cada una.

El chip decodificador de MPEG-1 se controla a través de dos puertos SPI, uno se usa para transmitir datos y el otro para controlarlo. Como el microcontrolador sólo dispone de un puerto SPI vamos a usar una puerta lógica AND para poder conectar los dos puertos del decodificador con el del microcontrolador.

Este integrado al ser de la serie 74HC posee las mismas características que el resto de componentes de esta serie que ya hemos usado.

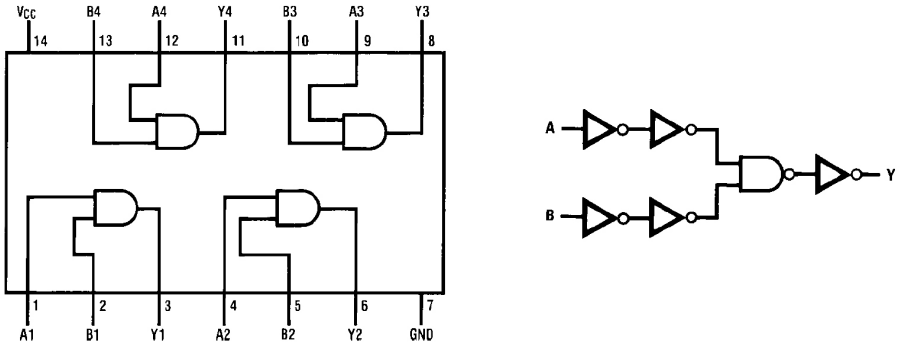


Figura 2.24: Diagrama lógico y patillaje del 74HCT08

LM1086

Para conseguir la tensión de alimentación de 3.3 voltios es necesario este regulador de tensión positiva. Entre sus características más importantes están:

- Disponible para multiples voltajes de salida *(1.8V, 2.5V, 2.85V, 3.3V, 3.45V, 5V y también en versión ajustable)*
- Limitador de corriente y protección térmica.
- Corriente máxima de salida 1.5 amperios.

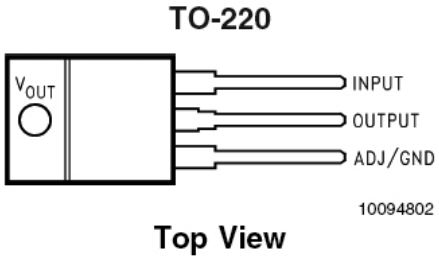


Figura 2.25: Diagrama lógico y patillaje del LM1086

VS1001K

El chip decodificador es el circuito que más componentes adicionales requiere para su funcionamiento. Ello se debe a que funciona a 3.3 voltios y a que no sólo integra el descompresor de MPEG-1 sino que también integra un decodificador digital/analógico y un amplificador.

Como vemos se maneja a través de dos puertos SPI. Las líneas que podemos encontrar en estos puertos son: En el **Puerto de Datos**:

- $\leftarrow DREQ$: Línea de petición de datos.
- $\rightarrow DCLK$: Entrada del reloj.
- $\rightarrow SDATA$: Entrada serie de datos.
- $\rightarrow BSYNC$: Línea de sincronismo de byte.

Por su parte en el **Puerto de Control** tenemos:

- $\leftarrow SO$: Salida de datos.

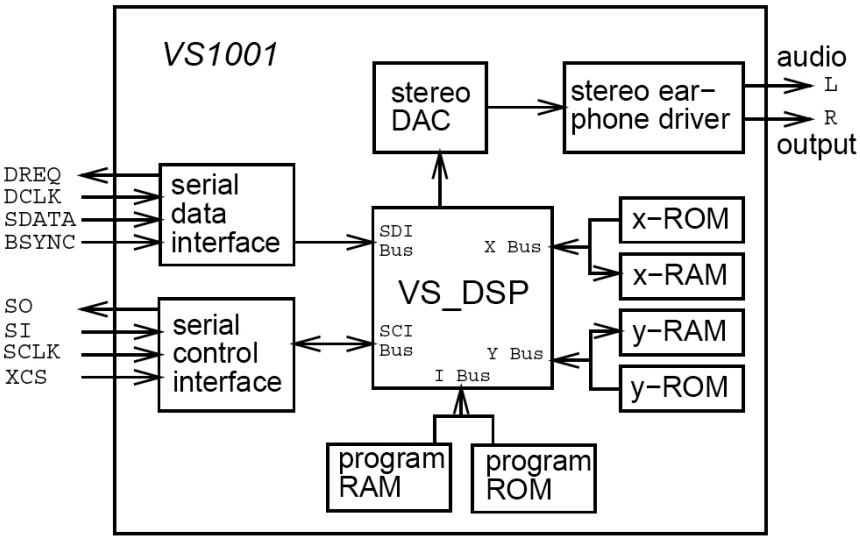


Figura 2.26: Diagrama de bloques del VS1001K

- $\rightarrow SI$: Entrada de datos.
- $\rightarrow SCLK$: Entrada de reloj.
- $\rightarrow \overline{XCS}$: Selección de Chip.

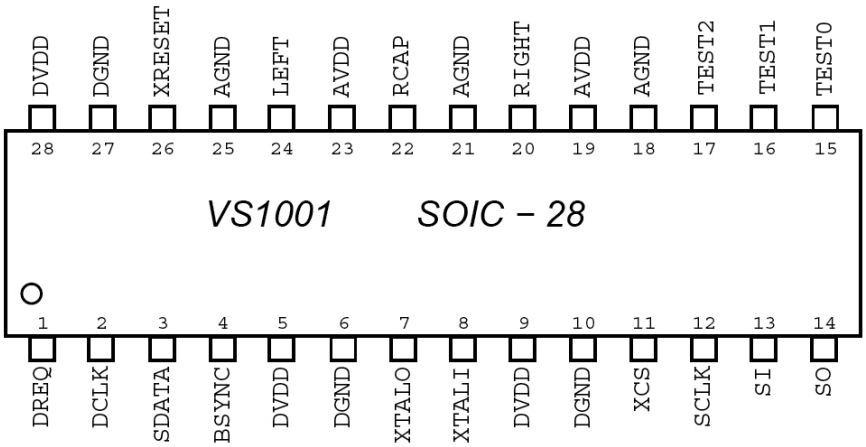


Figura 2.27: Diagrama lógico y patillaje del VS1001K

La estrategia para juntar los dos Buses en uno sólo se basa en la forma en la que se mandan datos de control. Tenemos que mantener activo $\overline{XCS}$ *(en nivel bajo)* todo el tiempo que dure la transmisión de datos de control y luego desactivarlo.

Podemos dividir la salida de datos serie del microcontrolador en las dos entradas de datos en el decodificador $SDATA$ y SI, sin problemas ya que en cada instante tan sólo una como mucho esta activa. Para dividir la línea de reloj que sale del microcontrolador en dos, usamos una puerta AND controlada por la línea $\overline{XCS}$, de forma que cuando transmitamos datos de control el reloj salga hacia $SCLK$ y no hacia $DCLK$ y cuando transmitamos datos de música salga hacia $DCLK$ y hacia $SCLK$, Esto en contra de los que nos pueda parecer no causa ningún problema, ya que en este caso $\overline{XCS}$ esta desactivado y no importa que el reloj llegue a $SCLK$.

El chip decodificador dispone de otra línea de control adicional, $\overline{XRESET}$ que sirve para provocar un RESET asíncrono en el dispositivo. Es lo que comúnmente se llama **RESET por Hardware**.

Con respectos a los tres pines de $TEST$ de los que dispone, el **0** debe conectarse a $DVDD$ mediante una resistencia de *10KΩ*, mientras que el **1** y el **2** no deben conectarse.

2.7.1. Componentes adicionales

Para la alimentación:

- **C27** *(10μF) Tántalo*, **C28** *(10μF) Tántalo*: Condensadores del regulador de tensión.
- **L1** *(10μH)*, **L2** *(10μH)*: Autoinducciones para el filtro de alimentación.
- **C20** *(10μF/16V) Electrolítico*, **C21** *(10μF/16V) Electrolítico*: Condensadores del filtro de tensión.
- **C29** *(100nF/50V)*, **C30** *(100nF/50V)*, **C31** *(100nF/50V)*, **C32** *(100nF/50V)*, **C33** *(100nF/50V)* Condensadores de desacoplo.

Para el decodificador:

- **C25** *(33pF) Cerámico*, **C26** *(33pF) Cerámico*, **X1** *(20MHz)* y **R11** *(1MΩ)*: Oscilador externo del decodificador.
- **C24** *(100nF/50V)*: Condensador para la referencia.
- **C22** *(100μF/25V) Electrolítico*, **C21** *(10μF/25V) Electrolítico*: Condensadores de la salida de audio.
- **R10** *(10KΩ)*: Resistencia para limitar la corriente.

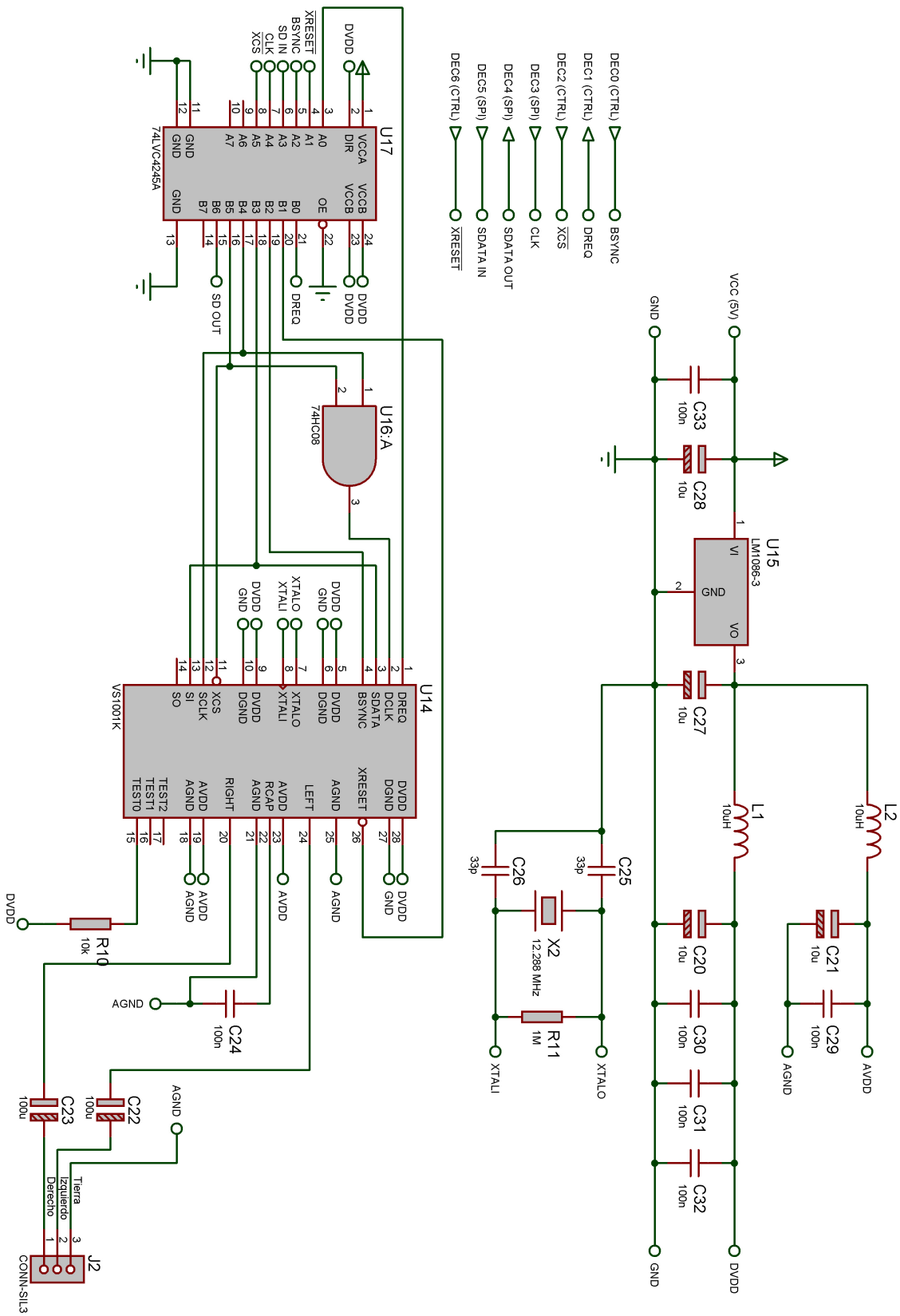


Figura 2.28: Esquema del módulo Decodificador

2.8. Fuente de alimentación

La fuente de alimentación debe de entregar una tensión estabilizada de 5 voltios para el funcionamiento de los circuitos integrados y otra de 12 voltios para la alimentación del disco duro.

Debe ser capaz de ofrecer una corriente máxima de 1.5 amperios en su salida de 12 voltios y 0.5 amperios en la de 5 voltios, ya que es el pico de corriente que necesita el disco duro para arrancar correctamente *(Spinup)*.

Power Mode	Typical Watts RMS	Typical Amps RMS	
		5V	12V
Spinup	—	0.5	1.5
Seeking (Random, no read/write)	7.0	0.4	0.42
Operating (read/write)	6.5	0.42	0.367
Idle	3.5	0.28	0.170
Standby/Sleep	0.8	0.123	0.015

Figura 2.29: Consumos de corriente del disco duro.

En la tabla y en la gráfica adjunta podemos observar esto

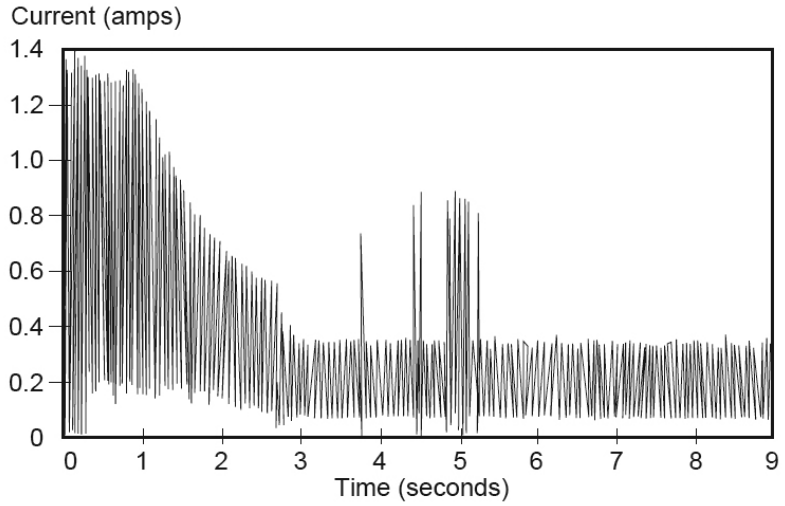


Figura 2.30: Gráfica del consumo de corriente del disco duro.

Una vez que el disco ha alcanzado su velocidad normal de rotación *(5.400 R.P.M)*, el consumo del disco duro se estabiliza entorno a los 0.4 amperios de máximo.

Por su parte, el resto del circuito tiene un consumo de corriente mucho menor, no sobrepasando en ningún caso los 200 miliamperios.

78XX

Los integrados de la serie 78XX son reguladores de tensión positiva. Los que vamos a usar en este proyecto son concretamente de la serie 78TXX que a diferencia de los 78XX soportan hasta 3 amperios de corriente.

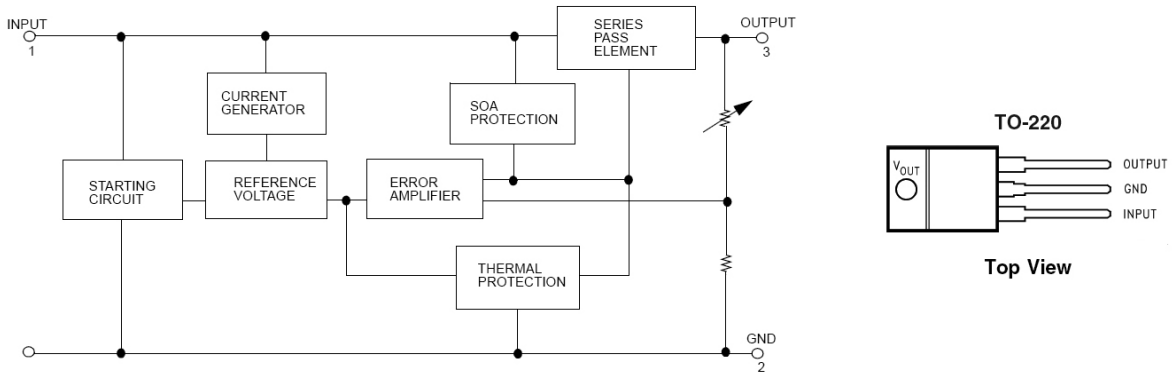


Figura 2.31: Diagrama lógico y patillaje del 78XX.

Usaremos dos reguladores, el 78T05 para conseguir la tensión de 5 voltios regulada y estabilizada y el 78T12 para conseguir la tensión de 12 voltios.

Sus características más importantes son:

- Soportan hasta 3 amperios de corriente.
- Disipan una potencia de 25 W.
- Protegidos internamente contra cortocircuitos y contra sobrecalentamientos.
- La tensión de salida se ofrece con un 4 % de precisión.
- Disponibles para 5, 12 y 15 voltios de salida.

Transformador

Como transformador para la fuente de corriente se ha usado uno de la casa Roqmo que dispone de dos devanados en el secundario, ofreciendo por cada uno de ellos 7.5 voltios y 4 amperios de corriente máxima en cada uno de ellos. Los hemos dispuesto en serie, de forma que obtenemos una tensión de salida de 15 voltios y una corriente de 2 amperios, que es suficiente para alimentar nuestro circuito y el disco duro.

Fuente completa

Esta fuente de alimentación se basa en el empleo de un transformador de tensión adecuada y un rectificador de onda completa con su regulador de tensión acoplado.

T1 reduce la tensión de la línea de 220 V C.A. a 15 voltios además de aislar eléctricamente la parte de baja tensión de la tensión de línea.

La tensión alterna extraída del **secundario de T1** se aplica a un **rectificador de onda completa** formada por 4 diodos en configuración de puente de GRAETZ.

Por medio del condensador **C36** se filtra esta tensión para obtener una tensión prácticamente continua. Esto sucede ya que durante el período ascendente del pulso el condensador se carga y cuando llega el pulso descendente comienza a descargarse, pero como el período de tiempo es muy corto y el condensador tiene una capacidad elevada el mismo se descargará muy poco. Se obtiene aquí una tensión de CC sin regular que ha de aplicarse al circuito integrado regulador **U18**, el cual entregará en su salida un nivel de tensión perfectamente regulado y estabilizado de **12 voltios**.

Esta tensión es filtrada por el condensador **C37** y desacoplada para ruidos de alta frecuencia mediante **C38**.

La salida regulada y estabilizada de U18 se deriva también al integrado regulador **U19**, que aportará un nivel de tensión perfectamente regulado y estabilizado de **5 voltios**, que serán a su vez filtrados y desacoplados mediante **C39** y **C40**.

C34 y **C35** derivan a masa los transitorios provenientes de la línea de **220V** producidos por la conexión/desconexión de cargas inductivas.

Los circuitos integrados de regulación están protegidos contra cortocircuitos y sobrecargas gracias a un sistema de limitación de corriente internamente dispuesto, así también protegido térmicamente contra exceso de disipación.

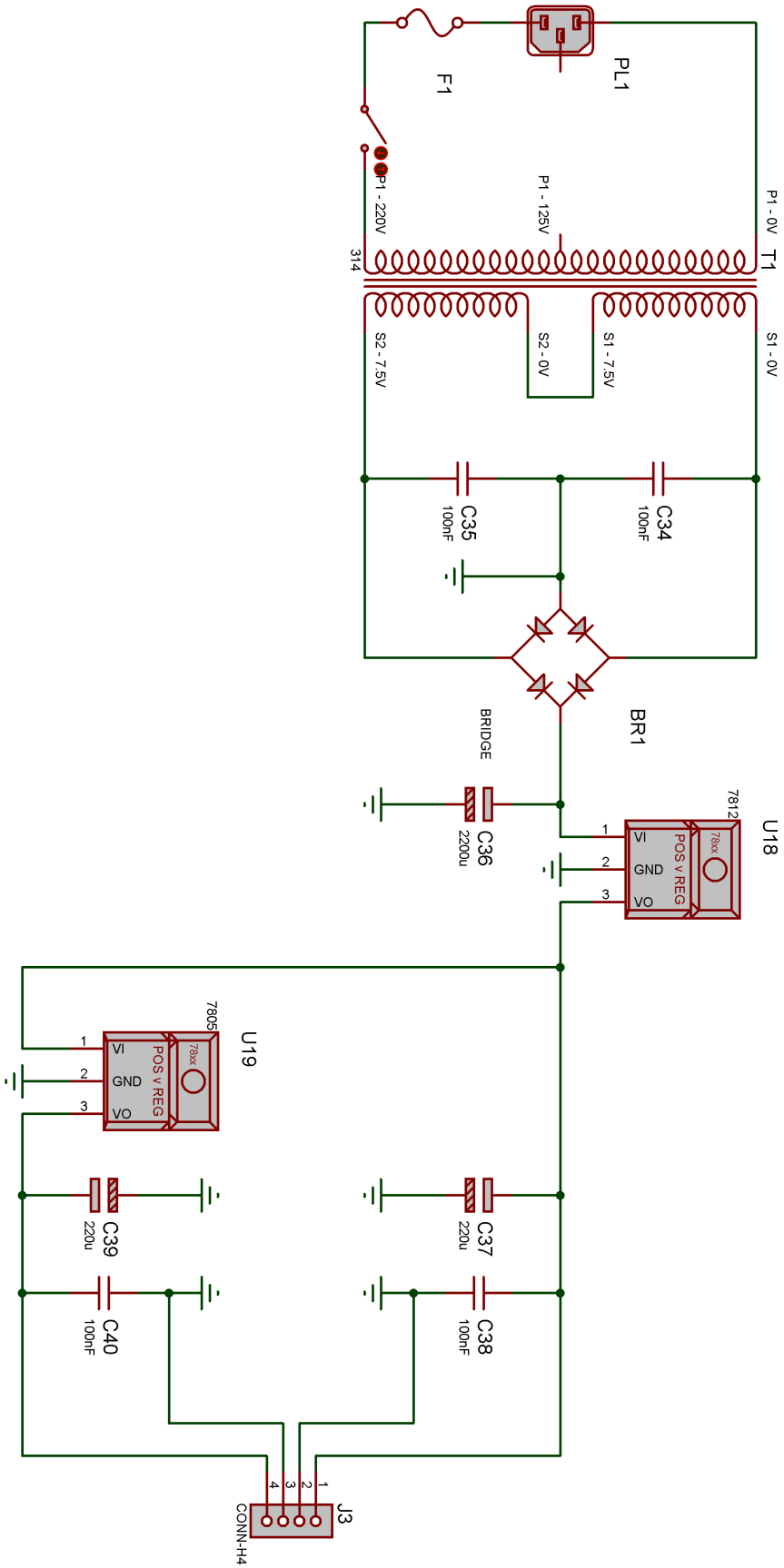


Figura 2.32: Esquema de la fuente de alimentación

2.8.1. Componentes adicionales

- **D1** *1N4007*, **D2** *1N4007*, **D3** *1N4007* y **D4** *1N4007*: Diodos para construir el puente de GRAETZ.
- **C34** *(100nF/50V)* y **C35** *(100nF/50V)*: Condensadores para derivar a masa los transitorios.
- **C36** *(2200μF/25V) Electrolítico*: Condensador para filtrar la tensión de entrada al regulador.
- **C37** *(220μF/25V) Electrolítico* y **C39** *(220μF/25V) Electrolítico*: Condensadores para filtrar la tensión de salida de los reguladores.
- **C38** *(100nF/50V)* y **C40** *(100nF/50V)*: Condensadores de desacoplo.

Capítulo 3

Software

En este capitulo describiremos todo lo relativo al software que usa el dispositivo. Comenzaremos examinado los protocolos de comunicación del dispositivo IDE *(ATA-1)* y del display LCD *(HD44780)* para pasar luego a examinar en profundidad el código fuente que controla el funcionamiento del dispositivo y por lo tanto de cada uno de los módulos.

3.1. AT Attachment ATA-1

En este apartado vamos a ver como se produce la comunicación con el dispositivo IDE.

Como ya quedó claro en el apartado de Hardware, la forma de comunicarnos con el dispositivo IDE es mediante sus **líneas de control** ($\overline{CS0}$-$\overline{CS1}$, $DA0$-$DA2$, $\overline{DIOW}$ y $\overline{DIOR}$) y el **BUS de Datos** ($DATA0$-$DATA15$).

El proceso es muy sencillo, se basa el la lectura y escritura de registros. Podría resumirse en tres pasos:

1. Indicar en las líneas $\overline{CS0}$-$\overline{CS1}$ si vamos a leer o escribir un **registro de control** ó un **registro de comando**.

2. Indicar el registro a leer o escribir en las líneas $DA0$-$DA2$.

3. Ejecutar la operación deseada: **Lectura** $\rightarrow \overline{DIOR}$, **Escritura** $\rightarrow \overline{DIOW}$.

Como ya ha quedado claro hay dos bloques de registros, el **bloque de registros de control** y el **bloque de registros de comando**. Los seleccionamos ajustando las líneas $\overline{CS0}$-$\overline{CS1}$. Cada bloque tiene una serie de registros a los que podremos acceder para leerlos, escribirlos o ambas cosas.

Por su parte con $DA0$-$DA2$ seleccionamos el registro que queremos leer o escribir dentro del bloque elegido.

Con $\overline{DIOR}$ y $\overline{DIOW}$ ejecutamos la acción. No todos los registros se pueden leer y escribir, hay algunos que sólo se pueden leer o sólo se pueden escribir. De hecho hay veces que el registro modificado cambia en función de la operación que hagamos sobre él, es decir, puede darse el caso de que dos registros compartan el mismo bloque y la misma dirección pero que como uno sólo se puede leer y el otro sólo se puede escribir, es precisamente la elección de esta operación la que de hecho selecciona el registro al que se accede.

Dirección					**Función**	
$\overline{CS1}$	$\overline{CS0}$	$DA2$	$DA1$	$DA0$	**READ - $\overline{DIOR}$**	**WRITE - $\overline{DIOW}$**
					Registros del bloque de Control	
A	N	1	1	0	Alternate Status	Device Control
A	N	1	1	1	Drive Address	No Usado
					Registros del bloque de Comandos	
N	A	0	0	0	Data	Data
N	A	0	0	1	Error Register	Features
N	A	0	1	0	Sector Count	Sector Count
N	A	0	1	1	Sector Number	Sector Number
					LBA bits 0-7	LBA bits 0-7
N	A	1	0	0	Cylinder Low	Cylinder Low
					LBA bits 8-15	LBA bits 8-15
N	A	1	0	1	Cylinder High	Cylinder High
					LBA bits 16-23	LBA bits 16-23
N	A	1	1	0	Drive/Head	Drive/Head
					LBA bits 24-27	LBA bits 24-27
N	A	1	1	1	Status	Command

Cuadro 3.1: Selección de registros en la interfaz ATA

Como vemos hay direcciones en las que la función asociada cambia dependiendo si usamos el direccionamiento de sectores LBA o no.

Si por ejemplo quisiéramos leer el registro de estado, tendríamos que poner en las líneas de control el código correspondiente a **status**: **N(1), A(0), 1, 1, 1**, activar $\overline{DIOR}$ y leer el registro del **BUS de Datos**.

Si quisiéramos mandar un comando al dispositivo, tendríamos que poner en las líneas de control el código correspondiente a **command**: **N(1), A(0), 1, 1, 1**, poner el código del comando en el **BUS de Datos** y activar $\overline{DIOW}$.

3.1.1. Descripción de los registros

STATUS

Este registro contiene el estado del dispositivo. Su contenido es actualizado cada vez que la ejecución de un comando se completa cuando BSY es puesto a cero, el resto de bits alcanzan su valor correcto en 400 ns. Si $BSY = 1$ el resto de los bits no son válidos.

La lectura de este registro por parte del *Host* cuando hay una interrupción pendiente , es considerada como un reconocimiento de interrupción. Cualquier interrupción pendiente es borrada cuando este registro se lee.

Bit Nº	7	6	5	4	3	2	1	0
Significado	BSY	$DRDY$	DWF	DSC	DRQ	$CORR$	IDX	ERR

Cuadro 3.2: Registro STATUS

- BSY (BUSY): Ocupado. El Host no debe acceder al bloque de registros de comando.
- $DRDY$ (DRIVE READY): El dispositivo es capaz de responder a un comando.
- DWF (DRIVE WRITE FAULT): Indica un fallo de escritura.
- DSC (DRIVE SEEK COMPLETE): Indica que las cabezas del dispositivo se han situado sobre la pista.
- DRQ (DATA REQUEST): Indica que el dispositivo está listo para transferir una palabra o un byte de datos entre el *Host* y el dispositivo.
- $CORR$ (CORRECTED DATA): Indica que se ha encontrado y corregido un error de datos. No aborta la transmisión de datos.
- IDX (INDEX): Se pone a 1 con cada vuelta del disco.
- ERR (ERROR): Indica que se ha producido un error en la ejecución del comando previo. El registro de **ERROR** especifica el tipo de error que se ha producido.

ALTERNATE STATUS

Contiene la misma información que el registro **STATUS**, pero su lectura no implica un reconocimiento de interrupción o un borrado de la interrupción pendiente.

COMMAND

Contiene el código de comando que va a ser enviado al dispositivo. La ejecución del comando comienza en cuanto este registro es escrito.

DATA

Este registro de 16-bits se usa para transferir bloques de datos entre el Buffer de datos del dispositivo y el *Host*.

DEVICE CONTROL

Sirve para controlar el dispositivo.

Bit Nº	7	6	5	4	3	2	1	0
Significado	X	X	X	X	1	$SRST$	$\overline{IEN}$	0

Cuadro 3.3: Registro DEVICE CONTROL

- $SRST$ (SOFTWARE RESET): El dispositivo se resetea cuando el Host pone este bit a 1.
- $\overline{IEN}$ (INTERRUPT ENABLE): Cuando esta a 0 se activa el uso de INTRQ. Cuando está a 1 INTRQ permanece en alta impedancia.

DRIVE ADDRESS

Este registro indica que dispositivo está activo, y que cabeza se está usando.

Bit Nº	7	6	5	4	3	2	1	0
Significado	HiZ	$\overline{WTG}$	$\overline{HS3}$	$\overline{HS2}$	$\overline{HS1}$	$\overline{HS0}$	$\overline{DS1}$	$\overline{DS0}$

Cuadro 3.4: Registro DRIVE ADDRESS

- HiZ (HIGH IMPEDANCE): Deberá estar siempre en alta impedancia.
- $\overline{WGT}$ (WRITE GATE BIT): Cuando se esta produciendo una escritura en el disco, permanece a 0.

- $\overline{HS3}$-$\overline{HS0}$ (HEAD SELECT): Cabeza seleccionada en complemento a 1.
- $\overline{DS1}$ (DRIVE SELECT 1): Cuando está a 0 indica que el dispositivo 1 está seleccionado y activo.
- $\overline{DS0}$ (DRIVE SELECT 0): Cuando está a 0 indica que el dispositivo 0 está seleccionado y activo.

DRIVE/HEAD

Permite seleccionar el dispositivo usado, la cabeza y el modo LBA.

Bit Nº	7	6	5	4	3	2	1	0
Significado	1	L	1	DRV	$HS3$	$HS2$	$HS1$	$HS0$

Cuadro 3.5: Registro DRIVE/HEAD

- L (LBA): Si vale 0 se selecciona el modo **CHS** de direccionamiento, si vale 1 se selecciona el modo **LBA**.
- DRV (DRIVE): Indica que dispositivo está seleccionado, el 0 o el 1.
- $HS3$-$HS0$ (HEAD SELECT): Si $L = 0$ indica la cabeza seleccionada. Cuando se completa un comando estos bits son actualizados para reflejar la cabeza actual. Si $L = 1$ contienen los bits $24 - 27$ de la dirección **LBA**.

ERROR

Este registro contiene el estado del último comando ejecutado por el dispositivo. El contenido de este registro sólo es valido si al finalizar la ejecución de un comando $ERR = 1$ en el registro **STATUS**.

Bit Nº	7	6	5	4	3	2	1	0
Significado	BBK	UNC	MC	$IDNF$	MCR	$ABRT$	$TK0NF$	$AMNF$

Cuadro 3.6: Registro ERROR

- BBK → BAD BLOCK DETECTED
- UNC → UNCORRECTABLE DATA ERROR
- MC → MEDIA CHANGED
- $IDNF$ → ID NOT FOUND
- MCR → MEDIA CHANGE REQUESTED
- $ABRT$ → ABORTED COMMAND
- $TK0NF$ → TRACK 0 NOT FOUND
- $AMNF$ → ADDRESS MARK NOT FOUND

Si se ejecuta un diagnóstico de dispositivo, los bits de este registro se interpretan como el resultado del diagnóstico según la siguiente tabla:

Código	**Significado**
0x01	No error detected
0x02	Formatter device error
0x03	Sector buffer error
0x04	ECC circuitry error
0x05	Controlling microprocessor error
0x8X	Drive 1 failed

Cuadro 3.7: Tabla de errores de diagnóstico

FEATURES

Este registro se usa específicamente para comandos, permite habilitar o inhibir características especiales del interfaz *(permite por ejemplo habilitar el cacheado)*. Para algunos dispositivos este registro no tiene ninguna función.

SECTOR COUNT

En este registro se indican el número de sectores que van a ser transferidos en una operación de lectura o escritura entre el *Host* y el dispositivo. Si su valor es 0, se transfieren un total de 256 sectores. Si al finalizar un comando su valor es 0, es que este se ejecutó de forma satisfactoria. Si su valor es distinto de 0, indica el número de sectores que todavía están pendientes de ser transferidos.

Valor	Significado
$0x01$	Enable 8-bit data transfers
$0x02$	Enable write cache
$0x03$	Set transfer mode based on value in sector count register
$0x33$	Disable retry
$0x44$	Vendor unique length of ECC on read long/write long commands
$0x54$	Set cache segments to sector count register value
$0x55$	Disable read look-ahead feature
$0x66$	Disable reverting to power on defaults
$0x77$	Disable ECC
$0x81$	Disable 8-bit data transfers
$0x82$	Disable write cache
$0x88$	Enable ECC
$0x99$	Enable retries
$0xAA$	Enable read look-ahead feature
$0xAB$	Set maximum prefetch using sector count register value
$0xBB$	4 bytes of ECC apply on read long/write long commands
$0xCC$	Enable reverting to power on defaults

Cuadro 3.8: Posibles valores de FEATURES

SECTOR NUMBER

Indica el numero del sector de comienzo para cualquier comando que acceda al disco. Puede valer desde 1 hasta el número máximo de **sectores por pista**.

Si el modo **LBA** está activo, este registro contiene los bits 0-7 de la dirección LBA.

CYLINDER LOW

Este registro contiene los 8 bits más bajos de la dirección del cilindro de comienzo para cualquier acceso al disco.

Al finalizar un comando, este registro se actualiza automáticamente para indicar la parte baja de la dirección del cilindro actual.

Si el modo **LBA** está activo, este registro contiene los bits 8-15 de la dirección LBA.

CYLINDER HIGH

Este registro contiene los 8 bits más significativos de la dirección del cilindro de comienzo para cualquier acceso al disco.

Al finalizar un comando, este registro se actualiza automáticamente para indicar la parte alta de la dirección del cilindro actual.

Si el modo **LBA** está activo, este registro contiene los bits 16-23 de la dirección LBA.

3.1.2. COMANDOS

La lista de comandos que se pueden ejecutar sobre el dispositivo es muy larga y variada. Hay que tener en cuenta que muchos de los comandos pueden tener o no sentido dependiendo del dispositivo que se trate, así por ejemplo no tiene sentido ejecutar sobre un disco duro un comando para recoger la bandeja.

Los comandos se dividen en tres grupos: **obligatorios - M** (MANDATORY), **opcionales - O** (OPTIONAL) y los **específicos del fabricante - V** (VENDOR SPECIFIC).

Comando	Tipo	Código	Registros usados
Acknowledge media chge	O	$0xDB$	DH(D)
Boot - post-boot	O	$0xDC$	DH(D)
Boot - pre-boot	O	$0xDD$	DH(D)
Check power mode	O	$0x98$ $0xE5$	DH(D), SC
Door lock	O	$0xDE$	DH(D)
Door unlock	O	$0xDF$	DH(D)
Execute drive diagnostic	M	$0x90$	DH(D)
Format track	M	$0x50$	DH, SC, FR, CY
Identify drive	O	$0xEC$	DH(D)
Idle	O	$0x97$ $0xE3$	DH(D), SC
Idle immediate	O	$0x95$ $0xE1$	DH(D)
Initialize drive parameters	M	$0x91$	DH, SC
NOP	O	$0x00$	DH
Read buffer	O	$0xE4$	DH(D)
Read DMA (w/retry)	O	$0xC8$	DH, SC, SN, CY
Read DMA (w/o retry)	O	$0xC9$	DH, SC, SN, CY
Read long (w/retry)	M	$0x22$	DH, SC, SN, CY
Read long (w/o retry)	M	$0x23$	DH, SC, SN, CY
Read multiple	O	$0xC4$	DH, SC, SN, CY
Read sector(s) (w/retry)	M	$0x20$	DH, SC, SN, CY
Read sector(s) (w/o retry)	M	$0x21$	DH, SC, SN, CY
Read verify sector(s) (w/retry)	M	$0x40$	DH, SC, SN, CY
Read verify sector(s) (w/o retry)	M	$0x41$	DH, SC, SN, CY
Recalibrate	M	$0x1X$	DH(D)
Seek	M	$0x7X$	DH, SN, CY
Set features	O	$0xEF$	DH(D), FR
Set multiple mode	O	$0xC6$	DH(D), SC
Sleep	O	$0x99$ $0xE6$	DH(D)

Comando	Tipo	Código	Registros usados
Standby	O	$0x96\ 0xE2$	DH(D), SC
Standby immediate	O	$0x94\ 0xE0$	DH(D)
Write buffer	O	$0xE8$	DH(D)
Write DMA (w/retry)	O	$0xCA$	DH, SC, SN, CY
Write DMA (w/o retry)	O	$0xCB$	DH, SC, SN, CY
Write long (w/retry)	M	$0x32$	DH, SC, FR, SN, CY
Write long (w/o retry)	M	$0x33$	DH, SC, FR, SN, CY
Write multiple	O	$0xC5$	DH, SC, FR, SN, CY
Write same	O	$0xE9$	DH, SC, FR, SN, CY
Write sector(s) (w/retry)	M	$0x30$	DH, SC, FR, SN, CY
Write sector(s) (w/o retry)	M	$0x31$	DH, SC, FR, SN, CY
Write verify	O	$0x3C$	DH, SC, FR, SN, CY
Vendor unique	V	$0x9A$	
Vendor unique	V	$0xC0\ 0xC3$	
Vendor unique	V	$0x8X$	
Vendor unique	V	$0xF0\ 0xFF$	
Reserved: all remaining codes			
DH - Drive/Head, **SC** - Sector Count, **FR** - Features, **CY** - Cylinder, **SN** - Sector Number			

Cuadro 3.9: Resumen de comandos ATA-1

3.1.3. Protocolos ATA-1

Los comandos pueden ser agrupados en distintas clases, según el protocolo que siga a la ejecución del comando.

En todos los comandos, lo primero que se hace es comprobar si $BSY = 1$. Sólo se puede proceder con la ejecución de un comando cuando $BSY = 0$.

Para la mayoría de comandos, además el *Host* debe esperar hasta que $DRDY = 1$.

Comandos de salida de datos del dispositivo

Esta clase incluye los siguientes comandos:

- Identify Drive
- Read Buffer
- Read Long
- Read Sector(s)

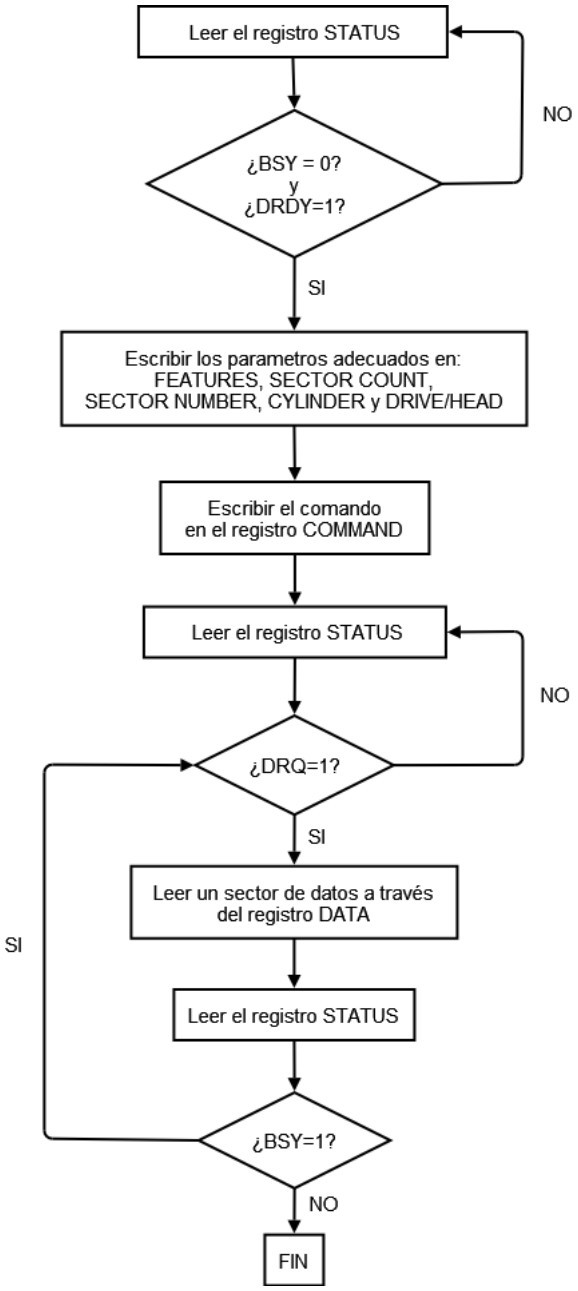


Figura 3.1: Protocolo de salida de datos del dispositivo

Su ejecución implica la transferencia de un bloque o más de 512 bytes (> 512 bytes si se ejecuta Read Long) de sector del dispositivo al *Host*.

Comandos de entrada de datos al dispositivo

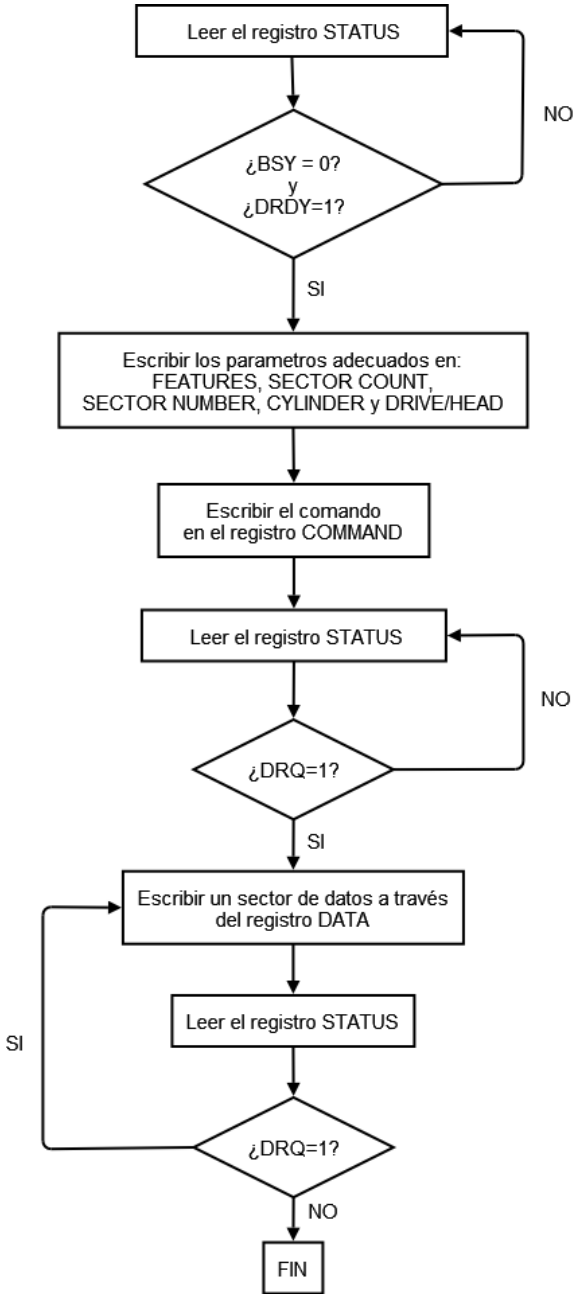


Figura 3.2: Protocolo de entrada de datos al dispositivo

Esta clase incluye los siguientes comandos:

- Format

- Write Buffer
- Write Long
- Write Sector(s)

Su ejecución implica la transferencia de un bloque o más de 512 bytes (> 512 bytes si se ejecuta Write Long) de sector del *Host* al dispositivo.

Comandos sin datos

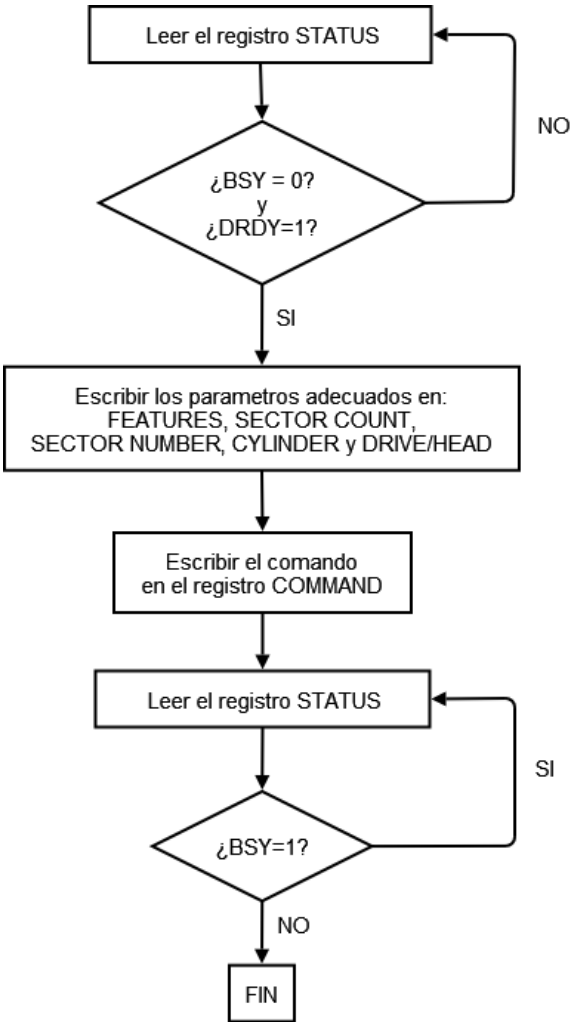


Figura 3.3: Protocolo para la ejecución de comandos sin transferencia de datos

Esta clase incluye los siguientes comandos:

- Execute Drive Diagnostic

- Idle
- Initialize Drive Parameters
- Read Power Mode
- Read Verify Sector(s)
- Recalibrate
- Seek
- Set Features
- Set Multiple Mode
- Standby

Su ejecución no implica la transferencia de datos.

Otro tipo de comandos

Esta clase incluye los siguientes comandos:

- Read Multiple
- Sleep
- Write Multiple
- Write Same

Estos comandos tienen cada uno un protocolo específico. Su descripción puede encontrarse en la documentación del CD adjunto.

Transferencias en modo DMA

Esta clase incluye los siguientes comandos:

- Read DMA
- Write DMA

El protocolo para esta clase puede encontrarse en la documentación del CD adjunto.

3.2. El controlador del Display - HD44780

Como ya hemos dicho, el display se controla con 3 líneas a parte del BUS de Datos, estas líneas (RS, RW y EN) nos van a permitir llevar a cabo todas las operaciones necesarias para inicializar el display y escribir en el.

- RS: Esta línea se usa para seleccionar que registro queremos modificar. Si vale **1**, se modifica el **Registro de Datos (DR)**, mientras que si vale **0**, se selecciona el **Registro de Instrucción (IR)**.
- RW: Con ella indicamos si la operación que queremos realizar es de **Lectura (1)** o de **Escritura (0)**.
- EN: Se usa como señal de reloj. Poniéndola a 1 y luego a 0, le indicamos al display que lleve a cabo la operación que contiene en su registro de instrucción **(IR)**.

3.2.1. Operaciones de lectura y escritura

Lo primero que debemos ver es como se llevan a cabo las operaciones más básicas, esto es las operaciones de lectura y escritura.

Lectura

Para llevar a cabo una operación de lectura y asumiendo que inicialmente $EN = 0$, se deben llevar a cabo los siguientes pasos:

1. Ajustar RS y RW a los valores deseados.
2. Esperar un mínimo de 60 ns.
3. Poner $EN = 1$.
4. Esperar un mínimo de 360 ns.
5. Leer el valor del BUS de Datos.
6. Poner $EN = 0$.

Escritura

Para llevar a cabo una operación de lectura y asumiendo que inicialmente $EN = 0$, se deben llevar a cabo los siguientes pasos:

1. Ajustar RS y RW a los valores deseados.
2. Esperar un mínimo de 60 ns.
3. Poner $EN = 1$.
4. Poner el valor a escribir en el BUS de Datos.
5. Esperar un mínimo de 195 ns.
6. Poner $EN = 0$.

3.2.2. Inicialización

Para poder trabajar con el display es necesario inicializarlo correctamente. En la fase de inicialización se definen varios parámetros básicos necesarios para el correcto funcionamiento del LCD como por ejemplo: si usamos interfaz de 8 o de 4 bits, el número de líneas que mostramos, si ponemos cursor o no...

Todos los parámetros que podemos ajustar, se encuentran en las siguientes instrucciones:

CLEAR DISPLAY

Borra la pantalla del display entera y pone la dirección DDRAM 0 en el contador de direcciones.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

Cuadro 3.10: Instrucción LCD - CLEAR DISPLAY

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

Cuadro 3.11: Instrucción LCD - ENTRY MODE SET

ENTRY MODE SET

Fija la dirección de movimiento del cursor y el desplazamiento del display.

- I/D: $I/D = 1$ (INCREMENT) El cursor se incrementa en 1. $I/D = 0$ (DECREMENT) El cursor se decrementa en 1.
- S: $S = 1$ (SHIFT) El display se desplaza. $S = 0$ (NO SHIFT) El display no se desplaza.

DISPLAY ON/OFF CONTROL

Fija el encendido o apagado de todo el display, la presencia del cursor y su parpadeo.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

Cuadro 3.12: Instrucción LCD - DISPLAY ON/OFF CONTROL

- D: $D = 1$ DISPLAY encendido. $D = 0$ DISPLAY apagado.
- C: $C = 1$ CURSOR presente. $C = 0$ CURSOR ausente.
- B: $B = 1$ (BLINKING) Parpadeo del cursor activo. $B = 0$ (BLINKING) Sin parpadeo en el cursor.

FUNCTION SET

Fija el tipo de interfaz, el número de líneas del display y la fuente de caracteres.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	D/L	N	F	*	*

Cuadro 3.13: Instrucción LCD - FUNCTION SET

- DL: $DL = 1$ Interfaz de 8 bits. $DL = 0$ Interfaz de 4 bits.
- N: $N = 1$ Dos líneas. $N = 0$ Una línea.
- F: $F = 1$ Fuente de 5x10 puntos. $F = 0$ Fuente de 5x8 puntos.

Secuencia de inicialización

Inicialización por RESET

Cuando la alimentación del display se conecta, se produce automáticamente un RESET interno que inicializa el display con unos parámetros determinados.

Para que se produzca esta inicialización automática, es necesario que se cumplan dos condiciones:

1. El tiempo que tarda la alimentación en pasar de **0.2 V** a **4.5V** (con una alimentación de 5V) debe de ser como **mínimo** de **0.1 ms** y como **máximo** de **10 ms**.
2. El tiempo que la alimentación esta en OFF (0.2 V) es un mínimo de **1 ms**. Esto es para compensar las oscilaciones de la alimentación en el encendido.

Las instrucciones que se ejecutan durante el proceso de inicialización automática y sus parámetros son las siguientes:

1. DISPLAY CLEAR
2. FUNCTION SET: $DL = 1$ (Interfaz de 8 bits), $N = 0$ (Una línea) y $F = 0$ (Fuente de 5x8 puntos).
3. DISPLAY ON/OFF CONTROL: $D = 0$ (Display apagado), $C = 0$ (Cursor ausente) y $B = 0$ (Parpadeo del cursor apagado).
4. ENTRY MODE SET: $I/D = 1$ (El cursor se incrementa en 1) y $S = 0$ (El display no se desplaza).

Si los parámetros de inicialización no se ajustan a lo deseado, pueden ser cambiados una vez inicializado el Display, tan sólo hay que tener en cuenta, que si se desean cambiar algún parámetro de FUNCTION SET, esta instrucción ha de ser ejecutado antes que cualquier otra. A partir de ese punto, los parámetros de FUNCTION SET no pueden ser modificados a no ser que se cambie el tamaño del interfaz. El resto de parámetros de inicialización pueden ser cambiados en cualquier momento.

Si las condiciones para la inicialización automática no se cumplen, entonces el display no funcionará correctamente. Será necesario proceder a una inicialización manual del mismo.

Para proceder a la inicialización manual del display LCD, es necesario ejecutar los comandos anteriores siguiendo un protocolo preestablecido.

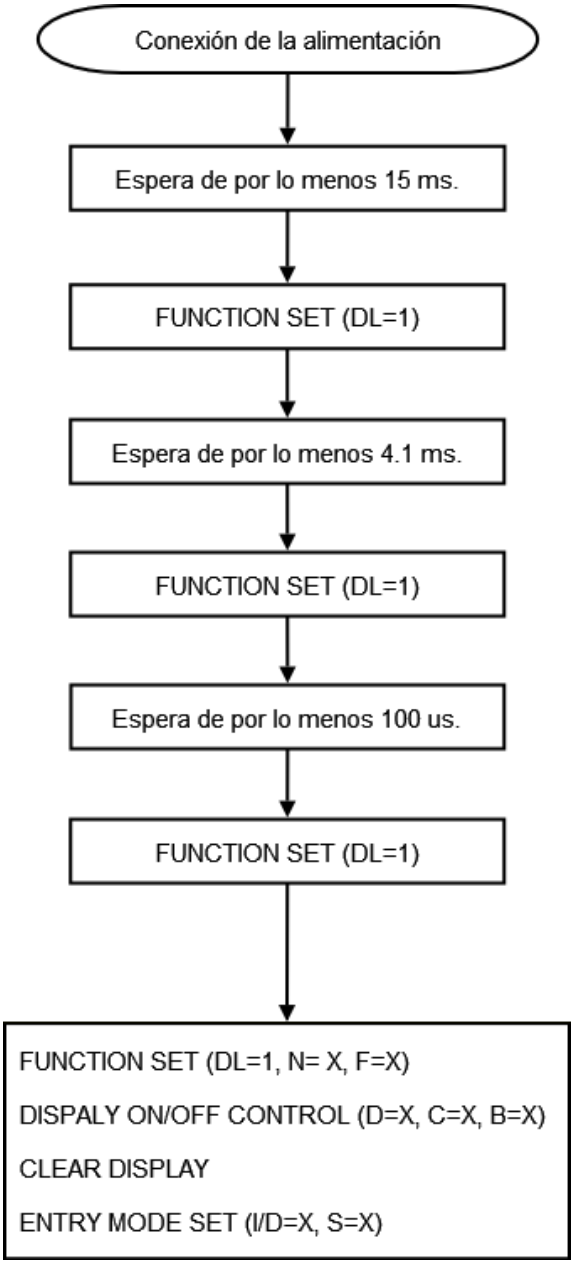


Figura 3.4: Inicialización manual del Display LCD

3.2.3. Uso del Display LCD

A continuación veremos las instrucciones básicas necesaria para manejar el LCD.

Escribir en el Display LCD

Escribir un carácter en el Display es tan fácil como ejecutar una instrucción de escritura. Para ello deberemos poner $RW = 0$ (Escritura), $RS = 1$ (Registro de datos) y $DB7$-$DB0$ = Carácter a escribir.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	Código del carácter							

Cuadro 3.14: Instrucción LCD - Escribir Carácter

Poner el cursor en una posición determinada

De esta forma conseguimos escribir un carácter en la posición que queramos, sin modificar el resto de la pantalla.

Todos los carácteres escritos en el LCD están almacenados en una memoria RAM (DDRAM) interna que este posee, cada posición tiene una dirección. Lo que haremos será ejecutar la instrucción de ajustar la dirección de memoria DDRAM.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	Dirección DDRAM						

Cuadro 3.15: Instrucción LCD - Ajustar la dirección DDRAM

Las direcciones de memoria DDRAM no son continuas en las 4 líneas del Display como puede verse en el siguiente mapa de memoria DDRAM.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

Figura 3.5: Posiciones de memoria del Display 4x20

Borrar el Display

Esta instrucción ya la vimos en el apartado de inicialización. Basta con escribir un 1 en el registro de Instrucción. $RS = 0$, $RW = 0$ y $DB7$-$DB0 = 1$

Esta instrucción borra el display entero y deja el cursor al principio de la primera línea.

Cursor al inicio

Actuamos sobre el registro de instrucción para ajustar la posición de memoria DDRAM a 0, pero no altera el contenido del LCD.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	*

Cuadro 3.16: Instrucción LCD - Cursor al inicio

A continuación podemos ver un resumen con todas las operaciones que se pueden llevar a cabo en el Display LCD.

En el CD que acompaña a la memoria se puede encontrar información detallada acerca de todos los aspectos del manejo del LCD.

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.
Return Home	0	0	0	0	0	0	0	0	1	*	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor and shifts display without changing DDRAM contents.
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (N), and character font (F).
Set CGRAM address	0	0	0	1	CGRAM address						Sets CGRAM address. CGRAM data is sent and received after this setting.
Set DDRAM address	0	0	1	DDRAM address							Sets DDRAM address. DDRAM data is sent and received after this setting.
Read busy flag & address	0	1	BF	CGRAM / DDRAM address							Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.
Write data to CG or CCRAM	1	0	Write data								Writes data into DDRAM or CGRAM.
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.

I/D = 1: Increment I/D = 0: Decrement	S/C = 1: Display shift S/C = 0: Cursor move	R/L = 1: Shift to the right R/L = 0: Shift to the left
S = 1: Accompanies display shift	N = 1: 2 lines, N = 0: 1 line	BF = 1: Internally operating BF = 0: Instructions acceptable
DL = 1: 8 bits, DL = 0: 4 bits	F = 1: 5 ´ 10 dots, F = 0: 5 ´ 8 dots	
DDRAM: Display data RAM	CGRAM: Character generator RAM	

Figura 3.6: Resumen de instrucciones del Display LCD

3.3. El sistema de ficheros FAT-32

En este apartado vamos a ver como es el sistema de ficheros elegido para el almacenamiento de la información en el disco duro. Para este propósito se ha elegido el sistema de ficheros FAT-32 desarrollado por Microsoft, ya que junto con el NTFS es el sistema que más se usa en la actualidad en PC's de uso doméstico.

3.3.1. El Master Boot Record

La estructura de datos más importante de un disco duro es el Master Boot Record. El *MBR* es el sector en el que toda la información de las particiones se almacena. El sistema tiene que conocer perfectamente su ubicación, por eso se le da una localización fija, se encuentra SIEMPRE en el primer sector del disco duro: **(HEAD:0 CYLINDER:0 SECTOR:1)**.

Cuando el PC arranca, la **BIOS** carga el *MBR* en memoria. Es el que se encarga de ejecutar las líneas de código necesarias para proceder a la carga del software del sistema operativo. También contiene la tabla de particiones la cual define las diferentes secciones en las que está dividido el disco duro. Baste con decir, que si algo malo les ocurriese a estos 512 bytes el disco duro sería inaccesible.

Dirección	Contenido
$0x000$	Código de partición
$0x1BE$	Tabla de partición
$0x1FE$	Marca de sector *"bootable"*

Cuadro 3.17: Estructura del Master Boot Record

Lo primero que nos encontramos en el *MBR* (Offset → 0x000)es el código de partición. Este es el código que será ejecutado por la BIOS del ordenador si el disco duro es identificado como *"bootable"*, para ello la ultima palabra (16 bits) de este sector (Offset → 0x1FE) debe ser **0xAA55**.

La tabla de particiones se encuentra a partir de la dirección $0x1BE$ del Master Boot Record. Tiene 4 entradas de 16 bytes cada una.

Una de las particiones contendrá el Sistema Operativo, esta partición es la partición **ACTIVA**. En cada momento sólo podemos tener una partición activa. El cometido del **'código de partición'** es identificar la partición activa, cargar el sector de arranque de esta partición en memoria (BOOT SECTOR) y ejecutarlo.

Dirección	Contenido
$0x1BE$	Primera entrada de la tabla de particiones
$0x1CE$	Segunda entrada de la tabla de particiones
$0x1DE$	Tercera entrada de la tabla de particiones
$0x1EE$	Cuarta entrada de la tabla de particiones

Cuadro 3.18: Estructura de la tabla de particiones

Cada entrada de partición tiene una estructura de 16 bytes que la describe y en la que podemos encontrar todos los datos necesarios para saber donde empieza, acaba y si es la partición activa o no.

Offset	Contenido
$0x00$	Estado de la partición: $0x00$ - No activa, $0x80$ - Activa
$0x01$	Cabeza donde comienza la partición
$0x02$	Sector y Cilindro donde comienza la partición
$0x04$	Tipo de partición
$0x05$	Cabeza donde acaba la partición
$0x06$	Sector y Cilindro donde acaba la partición
$0x08$	Distancia en sectores desde el sector de partición al primer sector de la partición
$0x0C$	Número de sectores de los que se compone la partición

Cuadro 3.19: Estructura de una entrada de partición

El byte colocado en el offset $0x04$ de la entrada de partición nos indica el **Tipo de partición**. Los tipos de partición más comunes se pueden encontrar en la siguiente tabla:

Número de referencia	Tipo
$0x00$	Vacía o Nada
$0x01$	DOS 12-bits FAT
$0x02$	XENIX root
$0x03$	XENIX usr
$0x04$	DOS 16-bits <32 Mb
$0x05$	Partición extendida MS-DOS
$0x06$	DOS 16-bits $\geq$32 Mb
$0x07$	OS/2 HPFS
$0x08$	AIX
$0x09$	AIX inicializable
$0x0A$	OS/2 Boot Manager
$0x0B$	32-bit FAT (Partición hasta 2048GB)
$0x0C$	32-bit FAT (Partición hasta 2048GB) LBA
$0x0E$	DOS 16-bits $\geq$32 Mb LBA
$0x0F$	Partición extendida MS-DOS LBA
$0x40$	Venix 80286

Número de referencia	Tipo
$0x51$	Novell
$0x52$	Microport
$0x63$	GNU HURD
$0x64$	Novell
$0x75$	PC/IX
$0x80$	Old MINIX
$0x81$	MINIX/Linux
$0x82$	Linux Swap
$0x83$	Linux Native
$0x93$	Amoeba
$0x94$	Amoeba BBT
$0xA5$	BSD/386
$0xB7$	BSDI fs
$0xB8$	BSDI swap
$0xC7$	Syrinx
$0xDB$	CP/M
$0xE1$	Acceso a DOS
$0xE3$	DOS R/O
$0xF2$	DOS secundaria

Cuadro 3.20: Tipos de partición más comunes

3.3.2. FAT-32 Boot Record - Boot Sector

Esta estructura de datos se encuentra localizada en el primer sector de cada partición, en ella se describe perfectamente la partición, datos clave como el número de bytes por sector, el número de FAT's o el número de cluster en el que comienza el directorio raíz se encuentran almacenados en esta estructura. Tiene una longitud de 512 bytes.

Offset	Tamaño	Significado
$0x000$	3 Bytes	Jump Code + NOP
$0x003$	8 Bytes	OEM Name
$0x00B$	1 Word	Bytes Per Sector
$0x00D$	1 Byte	Sectors Per Cluster
$0x00E$	1 Word	Reserved Sectors
$0x010$	1 Byte	Number of Copies of FAT
$0x011$	1 Word	Maximum Root Directory Entries (N/A for FAT32)
$0x013$	1 Word	Number of Sectors in Partition < 32MB (N/A for FAT32)
$0x015$	1 Byte	Media Descriptor (F8h for Hard Disks)
$0x016$	1 Word	Sectors Per FAT in Older FAT Systems (N/A for FAT32)
$0x018$	1 Word	Sectors Per Track
$0x01A$	1 Word	Number of Heads
$0x01C$	1 Double Word	Number of Hidden Sectors in Partition

Offset	Tamaño	Significado
$0x020$	1 Double Word	Number of Sectors in Partition
$0x024$	1 Double Word	Number of Sectors Per FAT
$0x028$	1 Word	Flags
$0x02A$	1 Word	Version of FAT32 Drive
$0x02C$	1 Double Word	Cluster Number of the Start of the Root Directory
$0x030$	1 Word	Sector Number of the File System Information Sector
$0x032$	1 Word	Sector Number of the Backup Boot Sector
$0x034$	12 Bytes	Reserved
$0x040$	1 Byte	Logical Drive Number of Partition
$0x041$	1 Byte	Unused (Could be High Byte of Previous Entry)
$0x042$	1 Byte	Extended Signature ($0x29$)
$0x043$	1 Double Word	Serial Number of Partition
$0x047$	11 Bytes	Volume Name of Partition
$0x052$	8 Bytes	FAT Name (FAT32)
$0x05A$	420 Bytes	Executable Code
$0x1FE$	2 Bytes	Boot Record Signature ($0x55AA$)

Cuadro 3.21: Estructura del Boot Sector

Sector de información del sistema de ficheros

Generalmente es el segundo sector de la partición, pero puesto que hay una referencia directa a su localización en el Boot Sector en el offset $0x30$, puede ser relocalizado en cualquier otro sector.

La información que podemos encontrar en este sector es la siguiente:

Offset	Tamaño	Significado
$0x000$	1 Double Word	First Signature ($0x52526141$)
$0x004$	480 Bytes	Unknown, Currently (Might just be Null)
$0x1E4$	1 Double Word	Signature of FSInfo Sector ($0x72724161$)
$0x1E8$	1 Double Word	Number of Free Clusters (Set to -1 if Unknown)
$0x1EC$	1 Double Word	Cluster Number of Cluster that was Most Recently Allocated
$0x1F0$	12 Bytes	Reserved
$0x1FC$	2 Bytes	Unknown or Null
$0x1FE$	2 Bytes	Boot Record Signature ($0x55AA$)

Cuadro 3.22: Estructura del sector de información del sistema de ficheros

Toda esta información nos sirve para localizar, entre otras cosas, varias zonas claves en la estructura del disco duro.

A partir del **Master Boot Record**, podemos determinar si la estructura del sistema

de ficheros es **FAT32** o no.

Del **Boot Sector** es de donde sacamos la mayoría de la información. Podemos determinar donde comienzan las **tablas FAT** sin más que sumar a la dirección de comienzo de la partición, el número de sectores reservados.

También obtendremos de el el **número de sectores por cluster**, dato que será vital para calcular la dirección de comienzo de un cluster dado.

El comienzo del **área de datos** también lo obtendremos con información del Boot Sector. Tan sólo deberemos sumar a la dirección de inicio de las tablas FAT, el número de sectores que ocupa cada una de ellas multiplicado por el número total de tablas que tengamos (generalmente 2). El comienzo del área de datos se corresponde con el cluster número 2, no con el 0 como cabría esperar) **NOTA:** Debido a que el primer cluster válido

para almacenar datos es el 2 y no el 0, si queremos saber en que sector comienza un cluster dado **N** deberemos realizar la siguiente operación:

*Primer sector del cluster N = ((N - 2) * Sectores por cluster)) + Comienzo del área de datos*

NOTA: Como el valor que puede contener **Sectores por cluster** está limitado a potencias de 2 (1, 2, 4, 8,16, 32...), la operación de multiplicación necesaria para calcular el primer sector de un cluster dado , se puede realizar fácil y rápidamente mediante rotaciones, evitando así las engorrosas multiplicaciones.

Los Clusters

Un cluster es un grupo de sectores del disco duro que contiene información. Como los sectores del disco son de 512 bytes, un cluster de por ejemplo 4KB contendrá 8 sectores (512 * 8 = 4096).

Si la información almacenada ocupa más de un cluster, se crea una cadena de clusters que contendrán la información.

Cada cluster tiene una entrada en la tabla FAT. En realidad las entradas de la tabla FAT no son de 32 bits, son sólo de 28 bits, por lo que deberemos ignorar los 4 bits más altos de la misma. Cuando inspeccionamos la entrada de la FAT correspondiente a un cluster, esta nos dice:

- Si tiene o no datos almacenados en ella. *(Si vale* $0x00000000$ *es que no hay datos almacenados en ella). Lo mismo indicarían los valores* $0x10000000$ *o* $0xF0000000Q$ *ya que ignoraremos los 4 bits más altos.*
- En caso de que tenga datos, nos puede indicar si es el ultimo cluster de datos de la cadena (EOC) o si hay más clusters con datos a continuación. (*Si vale* $0x0FFFFFFF$ *es que es el último cluster de la cadena)*
- Si el cluster está dañado y no puede ser usado para almacenar datos, el valor que nos encontramos en la tabla FAT es $0x0FFFFF7$
- Si hay más clusters con datos, nos indica el número del siguiente cluster de la cadena.

3.3.3. Estructura de directorio FAT

Un directorio FAT no es otra cosa que un fichero compuesto por una lista lineal de entradas de 32 bytes. Tan sólo hay un directorio que debe estar siempre presente, el **directorio raíz**. En los sistemas de ficheros FAT12 y FAT16, el directorio raíz está localizado siempre a continuación de las tablas FAT y tiene un tamaño fijo que se guarda en el Boot Sector en la entrada **Maximum Root Directory Entries**.

En FAT32 el directorio raíz está formado por una cadena de clusters y por lo tanto es de tamaño variable. El primer cluster del directorio raíz se almacena en el Boot Sector en la entrada **Cluster Number of the Start of the Root Directory**.

A diferencia del resto de directorios, el directorio raíz no tiene fecha ni hora de creación, no tiene nombre *(tan sólo "\")* y tampoco tiene como las dos primeras entradas del directorio "." y "..".

El directorio raíz es el único al que se le permite tener activo el atributo de ATTR_VOLUME_ID.

Los 32 bytes de cada entrada de directorio se distribuyen según la siguiente tabla:

Offset	Descripción	Tamaño
$0x00$	Nombre	11 Bytes
$0x0B$	Atributos	1 Byte
$0x0C$	Reservado para su uso con Windows NT	1 Byte
$0x0D$	Décimas de segundo en el momento de la creación	1 Byte
$0x0E$	Hora de creación	2 Bytes
$0x10$	Fecha de creación	2 Bytes

Offset	Descripción	Tamaño
$0x12$	Fecha del último acceso	2 Bytes
$0x14$	Bytes altos del cluster de comienzo	2 Bytes
$0x16$	Hora de la última actualización	2 Bytes
$0x18$	Fecha de la última actualización	2 Bytes
$0x1A$	Bytes bajos del cluster de comienzo	2 Bytes
$0x1C$	Tamaño en Bytes	4 Bytes

Cuadro 3.23: Estructura de una entrada de directorio corta.

Lo primero que nos encontramos son 11 bytes reservados para el nombre, los ocho primeros se corresponden con el nombre propiamente dicho, mientras que los tres últimos se reservan para la extensión. El punto que usualmente separa el nombre de la extensión no se almacena, se considera implícito. Si el nombre tiene menos de ocho caracteres, las posiciones no ocupadas por el nombre se rellena con espacios ($0x20$).

El primer carácter del nombre (posición 0) no puede ser un espacio, sin embargo puede tener los siguientes valores especiales:

- $0xE5$: Indica que la entrada esta libre.
- $0x00$: Indica que la entrada está libre y que no hay más entradas de directorio a continuación de esta. No es necesario examinar el resto de las entradas de directorio ya que están libres.
- $0x05$: Único carácter por debajo del $0x20$ válido. (Símbolo KANJI japonés)

Hay una serie de caracteres que no se pueden usar para el nombre:

- Valores por debajo de $0x20$ con excepción del ya mencionado $0x05$.
- $0x22$, $0x2A$, $0x2B$, $0x2C$, $0x2E$, $0x2F$, $0x3A$, $0x3B$, $0x3C$, $0x3D$, $0x3E$, $0x3F$, $0x5B$, $0x5C$, $0x5D$, y $0x7C$.

Los atributos definen el tipo de "fichero.ªlmacenado y determinan que se puede y que no se puede hacer con el.

- **ATTR_FILE** ($0x00$): Entrada normal.
- **ATTR_READ_ONLY** ($0x01$): No se puede escribir en la entrada.

- **ATTR_HIDDEN** ($0x02$): Entrada oculta. No debe ser mostrada en inspecciones "normales".
- **ATTR_SYSTEM** ($0x04$): Entrada que contiene datos del sistema operativo.
- **ATTR_VOLUME_ID** ($0x08$): Entrada que contiene la etiqueta del volumen.
- **ATTR_DIRECTORY** ($0x10$): Entrada que contiene otras entradas.
- **ATTR_ARCHIVE** ($0x20$): Entrada nueva o que ha sido modificada. Sirve para dar soporte a las utilidades de BACKUP.

El sistema de fichero FAT32 se caracteriza por dar soporte a nombres de entrada de hasta 255 caracteres. Para que esto sea posible y poder mantener la compatibilidad con FAT16, se crea un nuevo atributo que en realidad es una combinación de los ya existentes que indique que la entrada actual es parte de una entrada de nombre largo.

Este atributo, a partir de ahora **ATTR_LONGNAME** se indica con la siguiente combinación:

ATTR_READ_ONLY |ATTR_HIDDEN |ATTR_SYSTEM |ATTR_VOLUME_ID

Entradas de directorio largas

Como ya hemos visto, las entradas de directorio largo han sido diseñadas como una entrada de directorio normal que tiene un atributo especial. Esto tienes muchas ventajas, pero la principal es que al ser de hecho entradas de directorio normales, se asegura la compatibilidad con los sistemas FAT16 y lo que es más importante, con las herramientas de reparación de disco anteriores a la creación de este sistema de ficheros.

Para que esto sea posible es necesario que cada conjunto de entradas de directorio largas este asociado a una entrada de directorio corto, de forma que las entradas de directorio largas no sustituyen la información contenida en la entrada de directorio corta, sino que la complementan.

A continuación podemos ver la estructura de una entrada de directorio larga:

Offset	Descripción	Tamaño
$0x00$	Orden de la entrada	1 Byte
$0x01$	Caracteres 1 a 5 del nombre largo en esta entrada	10 Bytes
$0x0B$	Atributos (debe estar fijado a ATTR_LONGNAME)	1 Byte

Offset	Descripción	Tamaño
$0x0C$	Tipo: 0 → entrada que forma parte de un nombre largo	1 Byte
$0x0D$	CHECKSUM del nombre en la entrada de directorio corta	1 Byte
$0x0E$	Caracteres 6 a 11 del nombre largo en esta entrada	12 Bytes
$0x1A$	Debe ser 0 (compatibilidad con las utilidades de disco)	1 Byte
$0x1C$	Caracteres 12 y 13 del nombre largo en esta entrada	2 Bytes

Cuadro 3.24: Estructura de una entrada de directorio larga.

Organización y asociación de las entradas de nombre largo y corto

Cada serie de entradas de nombre largo esta asociado con la entrada de nombre corto a la que preceden. Hacen referencia al mismo "fichero". Esto es así para asegurar la compatibilidad con las versiones previas de MS-DOS o Windows, ya que para estos sistemas sólo son visibles las entradas de nombre corto.

Una entrada de nombre largo (o una serie de ellas) no tiene ningún sentido si no va acompañada de una entrada de nombre corto.

Cada miembro de una secuencia de nombres largos esta numerado unívocamente y el último miembro de la cadena se marca con un FLAG LAST_LONG_ENTRY que indica que ya no hay más. Este FLAG se elabora haciendo un **OR** entre el número de **orden de la entrada** en la cadena y $0x40$, el resultado obtenido se almacena de nuevo en el **Orden de la entrada**.

Las entradas son almacenadas en orden inverso, de forma que lo que primero nos encontraremos será la entrada con la marca de LAST_LONG_ENTRY con lo que inmediatamente sabremos de cuantas entradas consta la cadena, sin más que eliminar del **orden de entrada** el séptimo bit correspondiente al FLAG de LAST_LONG_ENTRY.

La primera entrada de nombre largo, la que precede a la entrada de nombre corto con la que esta asociada, se identifica con el número 1 no con el 0, ya que si no se confundiría con una entrada libre.

Cada una de las entradas de nombre largo asociadas a la entrada de nombre corto se marca con el CHEKSUM del nombre corto de la entrada. Este CHEKSUM se elabora en el momento de la creación de las entradas a partir de los 11 caracteres que conforman el nombre corto. Si alguna de las entradas de nombre largo tiene un CHEKSUM distinto al del nombre corto al que van asociadas, entonces todas las entradas de nombre largo se consideran inválidas.

Entrada	**Orden de la entrada**
Enésima entrada de nombre largo	LAST_LONG_ENTRY |N
...Entradas de nombre largo adicionales	...
Primera entrada de nombre largo	1
Entrada de nombre corto	**No aplicable**

Cuadro 3.25: Secuencia de entradas de nombre largo

Una entrada de nombre largo puede contener más caracteres de los que son necesarios para almacenar el nombre. Si esto ocurre, las posiciones correspondientes a los caracteres que no son necesarios se rellenan con NULL ($0xFFFF$).

Los caracteres de las entradas de nombre largo son almacenadas en estas en formato UNICODE (16 bits).

Los nombres largos están limitados a 255 caracteres. Los caracteres validos para los nombres son los mismos que para las entradas de nombre corto, pero se incluye el punto (.) que puede usarse varias veces dentro del nombre, el espacio ( ) también puede usarse aunque este ya se podía usar en entradas de nombre corto, sólo que entonces no tenía demasiado sentido.

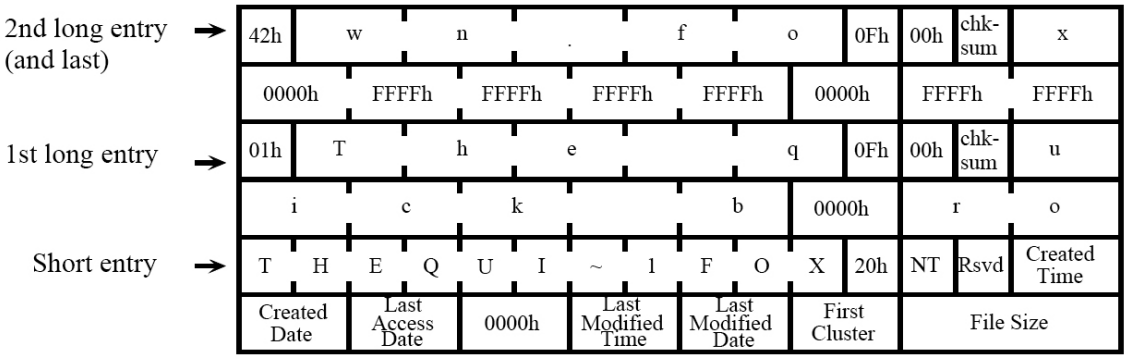


Figura 3.7: Ejemplo de almacenamiento de entradas de nombre largo y corto

3.4. BOOTLOADER - *BOOT CODE*

Una de las características de la familia de microprocesadores **PIC 16F87X** es que tienen la posibilidad de escribir directamente en la memoria de programa, de forma que con un pequeño programa residente en memoria (*BOOT CODE*) podemos actualizar el firmware del dispositivo (*USER CODE*) sin vernos obligados a extraer el integrado para programarlo.

El Bootloader (*BOOT CODE*) se encargará de ejecutar el código de usuario (*USER CODE*) a no ser que detecte que una nueva actualización de este está disponible, en cuyo caso se encargará de obtener los datos y escribirlos en la memoria de programa.

Las características más importantes del Bootloader son las siguientes:

- No es necesaria ninguna modificación especial en el código de usuario (*USER CODE*)
- El código de arranque (*BOOT CODE*) tan sólo ocupa una pequeña cantidad de memoria de programa.
- Comprueba de forma automática si hay disponible una nueva versión del código de usuario.
- Ejecuta el código de usuario de forma transparente, si no hay disponible una nueva versión de este.
- Recibe el nuevo código a programar a través del puerto serie (tan sólo se requieren dos líneas para su conexión con el PC).
- Programa el nuevo código de usuario en memoria IN-LINE. No es necesario extraer el microcontrolador.

3.4.1. Funcionamiento del Bootloader

Lo primero que hace el *BOOT CODE* es chequear si hay disponible una nueva versión del *USER CODE* para ser programada. Se activa la recepción de datos por el puerto serie durante una pequeña fracción de tiempo y se comprueba si se ha recibido la identificación del programa residente en el PC encargado de efectuar la transmisión de los nuevos datos. Si pasado el tiempo establecido no se recibe la identificación por el puerto serie, se concluye que no hay una nueva versión del *USER CODE* disponible y se ejecuta de forma normal la "versión antigua". Si se recibe el código de identificación del programa cargador, se ejecuta el proceso de recepción de datos a través del puerto serie y de escritura en la memoria de programa.

Integración del *BOOT CODE* y el *USER CODE*

Para el correcto funcionamiento del Bootloader, el *BOOT CODE* hace uso de dos zonas de memoria de programa:

- VECTOR DE RESET: Zona de memoria a la que accede el microcontrolador cada vez que es reseteado (0x0000).
- Una pequeña cantidad de memoria situada en la zona más alta de la memoria de programa de forma que no interfiera con el *USER CODE*.

El BOOT CODE no necesita para su funcionamiento el uso de interrupciones, de forma que el *USER CODE* puede usar el vector de interrupciones (0x0004) de forma normal.

Para evitar que el *BOOT CODE* use el vector de interrupciones, es necesario situar en el vector de RESET (0x0000 - 0x0003) un salto incondicional a la zona de memoria ocupada por el *BOOT CODE*.

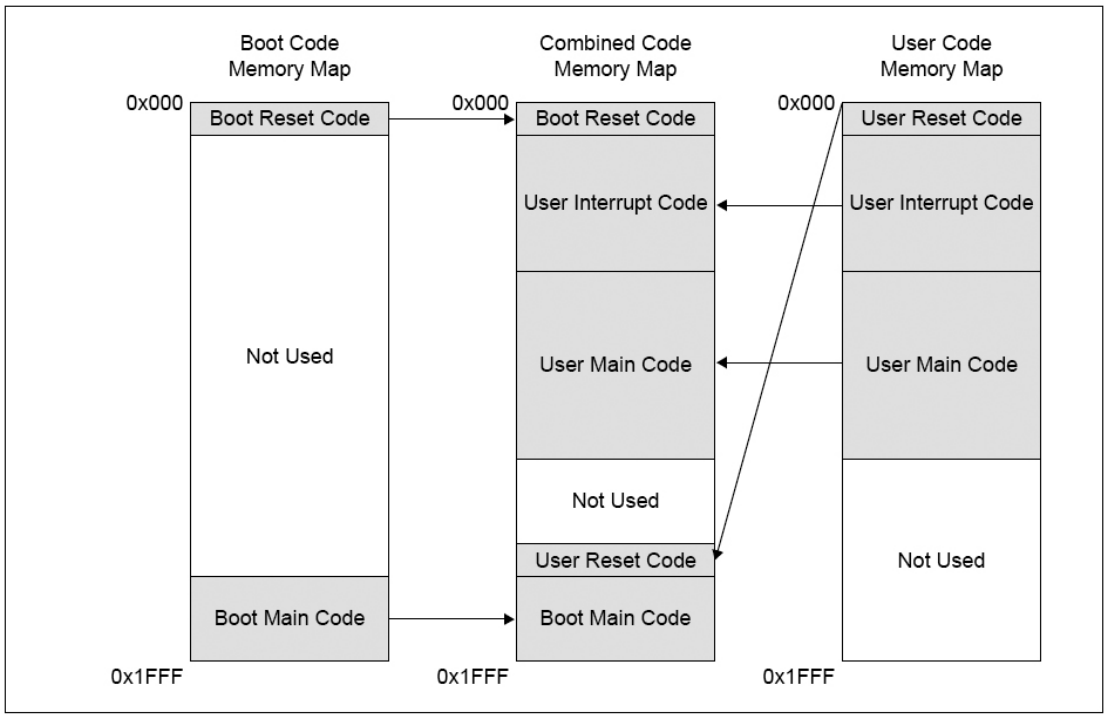


Figura 3.8: Integración del BOOT CODE y el USER CODE

Para que el *BOOT CODE* pueda ejecutar el *USER CODE* es necesario que sepa perfectamente donde se encuentra localizado. Como el *BOOT CODE* comienza en el vector de RESET, el *USER CODE* no puede comenzar en el vector de RESET. Sin embargo queremos que el Bootloader sea transparente, que no tengamos que hacer ninguna modificación especial en el *USER CODE*. Para conseguir esto, basta con hacer que el *USER CODE* empiece de forma normal en el vector de RESET pero que una de sus cuatro primeras instrucciones sea un *goto* que salte el vector de interrupciones. Estas cuatro instrucciones serán

relocalizadas por el *BOOT CODE* en otra zona de memoria ala que este accederá cuando quiera ejecutar el *USER CODE*. El *BOOT CODE* se encargará de asegurar que tanto la página como el banco de memoria usados son los correctos en el momento de ejecutar estas cuatro instrucciones.

BITS de Configuración

El *BOOT CODE* se debe programar en el microcontrolador usando las técnicas habituales (programador externo) y los bits de configuración deben ajustarse correctamente en este momento. El *BOOT CODE* no puede acceder a los BITS de configuración, ya que estos no están mapeados en memoria y por lo tanto no puede cambiarlos.

- **CPx**: Habilitación de la protección de memoria de programa
- **DEBUG**: Habilitación de la depuración In-Circuit
- **WRT**: Habilitación de la escritura de memoria de programa
- **CPD**: Habilitación de la protección de datos EEPROM
- **LVP**: Habilitación de la programación In-Circuit de bajo voltaje
- **BODEN**: Habilitación del Brown-out Reset
- **PWRTE**: Habilitación del temporizador de Power-up
- **WDTE**: Habilitación del Watchdog
- **FOSCx**: Selección del oscilador empleado

Muchas de estas opciones de configuración dependen del hardware o del diseño en general del dispositivo empleado y por lo tanto no van a cambiar.

La protección de todo el código, evita que este pueda ser leído de forma externa, pero no evita su sobrescritura.

La protección de Datos EEPROM sólo se usaría si tuviésemos datos del *BOOT CODE* que no quisiéramos que fueran sobrescritos por el *USER CODE*.

La escritura en memoria de programa tiene que se habilitada por razones obvias.

La programación de bajo voltaje sólo es necesario si se quiere programar el microcontrolador In-Circuit pero a través de RB3, RB6 y RB7.

Si el Watchdog está habilitado entonces tanto el *BOOT CODE* como el *USER CODE* deben de soportar el empleo del Watchdog.

Recepción del nuevo código de usuario

El nuevo código de usuario (*USER CODE*) se recibe via serie desde el PC por medio del puerto USART del PIC.

El *BOOT CODE* debe ser capaz de controlar la recepción de datos, ya que mientras está escribiendo en la memoria de programa no puede procesar ningún dato enviado desde el PC.

Los datos recibidos no sólo contienen el código de programa. Normalmente también se envía la dirección de memoria en la que deben ser almacenados y algún tipo de cheksum para detectar posibles errores. El BOOT CODE debe decodificar, verificar en su caso y almacenar los datos antes de escribirlos en la memoria de programa.

Programación de la memoria FLASH

La memoria de programa es de tipo FLASH. El microcontrolador dispone de unos registros de función especiales que se usan para escribir en esta memoria.

Para evitar escrituras no intencionadas en la memoria de programa hay establecida una secuencia específica de escrituras en estos registros que debe seguirse.

El código de programa no puede ser ejecutado mientras se está llevando a cabo una escritura en la memoria FLASH, así que el microcontrolador detiene la ejecución del código de programa hasta que el ciclo de escritura ha finalizado.

3.4.2. Implementación del Bootloader

El código fuente del Bootloader comienza con la declaración de cuatro constantes de configuración. A partir de estas constantes, el *BOOT CODE* obtiene todos los datos necesarios para establecer correctamente la conexión con el PC. **(Apéndice A.1 - Línea 14)**

- FOSC: Velocidad del oscilador usado para el microcontrolador
- BAUD: Velocidad en baudios de la conexión $PIC \leftrightarrow PC$
- BAUD_ERROR: Porcentaje de error máximo en la velocidad de conexión

- TIMEOUT: Tiempo máximo de espera para combrobar si hay nuevo Firmware disponible (multiplos de 0.1 segundos)

A continuación se calculan las constantes necesarias para configurar correctamente la USART (BRGH y SPBRG). **(Apéndice A.1 - Línea 25)**

El PIC no es capaz de generar cualquier velocidad de conexión con una precisión del 100 %, por ello es necesario realizar una comprobación para ver si se la velocidad de conexión seleccionada se puede generar con un error menor del indicado. **(Apéndice A.1 - Línea 39)**

Después se generan en función de la velocidad de conexión fijada las constantes necesarias para configurar el temporizador. **(Apéndice A.1 - Línea 59)**

Completamos la zona de definiciones fijando la dirección de memoria de programa en la que comienza el Bootloader, el tamaño del mismo, el número de reintentos de escritura y establecemos los códigos de la comunicación entre el PC y el PIC. **(Apéndice A.1 - Línea 95)**

La forma en la que funciona el *BOOT CODE* es muy sencilla y a grandes rasgos ya se ha expuesto anteriormente: una vez conectado el dispositivo el microcontrolador salta al vector de RESET donde reside la primera zona del *BOOT CODE*, al ejecutarla se genera un salto incondicional al comienzo del *BOOT CODE* en la zona alta de memoria. **(Apéndice A.1 - Línea 148)**

Lo primero que hace el *BOOT CODE* es habilitar el módulo de conexión serie **(Apéndice A.1 - Línea 194)** para luego pasar a configurar el puerto serie (USART) **(Apéndice A.1 - Línea 217)** y el temporizador 1 (TIMER1). **(Apéndice A.1 - Línea 244)**

Ahora es cuando comprobamos durante el tiempo prefijado en los datos de configuración si se recibe algún dato por el puerto serie **(Apéndice A.1 - Línea 250)** de ser así se comprueba si proviene de la aplicación que se encarga de transmitir los datos desde el PC y en caso afirmativo se comienza el proceso de recepción y grabado del nuevo código de usuario desde el PC. **(Apéndice A.1 - Línea 263)**

Si durante el tiempo de espera no se consigue establecer la conexión entre el PC y el PIC se pasa a desconectar el temporizador, resetear el puerto serie USART y desconectar el módulo de comunicación serie **(Apéndice A.1 - Línea 289)** ejecutando a continuación, las cuatro instrucciones correspondientes al vector de RESET del programa de usuario que fueron relocalizadas. **(Apéndice A.1 - Línea 168)**

En el proceso de recepción analizamos el código proveniente del PC y en función de este optamos por recibir y escribir una nueva línea de código de usuario **(Apéndice A.1 - Línea 266)** o si ya hemos terminado con todas por pasar a ejecutar el nuevo código de usuario. **(Apéndice A.1 - Línea 274)**

Para recibir una nueva línea de código **(Apéndice A.1 - Línea 320)** lo primero que descargamos es la dirección de destino en memoria, el número de bytes que conforman el comando y el cheksum del mismo. Los datos de comando son entonces descargados a un buffer intermedio y a medida que se descargan se calcula su cheksum. **(Apéndice A.1 - Línea 361)** Si el cheksum es correcto se lo indicamos al PC **(Apéndice A.1 - Línea 375)** y pasamos a escribir el comando en memoria. **(Apéndice A.1 - Línea 440)** Si el cheksum calculado no coincide con el recibido **(Apéndice A.1 - Línea 371)** se lo notificamos al PC para que proceda a su reenvío.

3.4.3. Funciones

En el código del Bootloader (*BOOT CODE*) usamos tres funciones:

Función - putbyte

(Apéndice A.1 - Línea 396) Se encarga de mandar al PC el código correspondiente al resultado de la última operación realizada sobre los datos recibidos.

- WR_OK: Escritura del comando en memoria de programa realizada con éxito.
- WR_BAD: Fallo en la escritura del comando en memoria de programa.
- DATA_OK: Datos enviados del PC descargados con éxito. (Cheksum correcto)
- DATA_BAD: Cheksum incorrecto, datos recibidos erróneos.
- IDACK: Programa de envío de datos identificado correctamente. Comunicación $PC \leftrightarrow PIC$ establecida.

Para evitar mandar un código cuando todavía no se ha terminado de enviar el anterior, se comprueban FLAGS y se hace una espera activa en su caso.

Función - getbyte

(Apéndice A.1 - Línea 410) Recibe un byte de datos desde el PC. Este byte puede corresponder a un código de comunicación: IDENT, WRITE ó DONE, o a parte de una línea de comando: parte de la dirección de destino, cheksum, número de bytes ó parte del comando.

Función - write_eeprom

(Apéndice A.1 - Línea 440) Su misión es la de escribir la línea de comando recibida del PC en la posición adecuada de la memoria de programa. La línea de comando se encuentra almacenada en el buffer intermedio y la dirección de destino en las variables correspondientes. En función de la dirección de destino decide si la línea de comando pertenece al vector de RESET del código de usuario y por lo tanto tiene que ser relocalizada o si es una línea "normalz ha de ser localizada en la dirección transmitida.

También realiza las comprobaciones necesarias para verificar que la línea de comando se escribió correctamente en la memoria de programa.

3.5. Programa Principal - *USER CODE*

El programa principal está ubicado en la **Página 0** de código de programa. Se podría decir que es el 'cerebro' del dispositivo, es el que se encarga de inicializar todos los periféricos que se usan, tanto los internos del microprocesador como los externos a él. También se encarga de interpretar las decisiones del usuario para desempeñar su principal objetivo, reproducir canciones comprimidas en 'MP3'.

Podemos distinguir cuatro zonas de funcionamiento en él:

- Inicialización de periféricos.
- Rutina de interrupción.
- Zona de navegación.
- Funciones de apoyo.

Comienza con la declaración de una serie de constantes que nos van a facilitar la programación. Estas constantes se corresponden con:

- Asignación de puertos: **(Apéndice A.2.2 - Línea 15)** Asignamos nombres relevantes a los puertos del microprocesador en función del cometido que desempeñan, así por ejemplo a los **puertos B** y **D** que se usan para el **BUS de Datos** les asignamos los nombres *DATOS_BAJOS* y *DATOS_ALTOS* respectivamente.

- Asignación de pines: **(Apéndice A.2.2 - Línea 37)** Asignamos nombres relevantes a los pines del microprocesador que se usan para controlar el dispositivo. Por ejemplo, el **bit 3** del **puerto A** se usa para dar la orden de escritura en el dispositivo IDE, por lo tanto asignamos el número 3 a la constante de programa *DIOW*.

- Códigos de habilitación del demultiplexor: **(Apéndice A.2.2 - Línea 60)** Asignamos nombres relevantes a los distintos códigos que han de ser introducidos en el demultiplexor para activar los distintos periféricos. Tenemos que tener en cuenta que el demultiplexor se controla a través de los **bits 0, 1** y **2** del **puerto A**, y que el **bit 3** se usa para generar la señal $\overline{DIOW}$. Siempre que conectemos un periférico, $\overline{DIOW}$ debe estar desactivada, es decir a '1', por lo tanto para, por ejemplo, activar la línea $\overline{DIOR}$ (conectada a la salida 0 del demultiplexor), debemos generar el código '001000' (el puerto A tiene seis bits). Por lo tanto asignaremos el código **'001000'** a la constante *DIOR*.

3.5.1. Inicialización de periféricos

Lo primero que hace el programa principal es inicializar los periféricos que va a usar. Y como no podía ser de otra forma, comienza con la configuración del propio microcontrolador.

Lo primero a configurar son los puertos del microcontrolador. Hasta ahora sólo les hemos asignado nombres, pero no hemos definido que pines dentro de cada puerto son de entrada de datos y cuales de salida.

(Apéndice A.2.2 - Línea 307) El **puerto A** se usa como **puerto de control 1**, lo configuramos como 'digital' (el microprocesador dispone de un convertidor anológico/digital conectado al puerto A) y configuramos todos sus pines como de salida.

- Bit0: Control del multiplexor. (**SALIDA**)
- Bit1: Control del multiplexor. (**SALIDA**)
- Bit2: Control del multiplexor. (**SALIDA**)
- Bit3: $\overline{DIOW}$.(**SALIDA**)
- Bit4: NO ASIGNADO. (**SALIDA**)
- Bit5: NO ASIGNADO. (**SALIDA**)

(Apéndice A.2.2 - Línea 324) Los **puertos B** y **D** se usan para el **BUS de Datos**, y por lo tanto actúan tanto de entrada como de salida. Los configuramos por defecto como de salida.

Puerto B: parte baja del BUS de Datos.

- Bit0: DATA0 (**SALIDA**)
- Bit1: DATA1 (**SALIDA**)
- Bit2: DATA2 (**SALIDA**)
- Bit3: DATA3 (**SALIDA**)
- Bit4: DATA4 (**SALIDA**)
- Bit5: DATA5 (**SALIDA**)
- Bit6: DATA6 (**SALIDA**)
- Bit7: DATA7 (**SALIDA**)

Puerto D: parte alta del BUS de Datos.

- Bit0: DATA8 (**SALIDA**)
- Bit1: DATA9 (**SALIDA**)
- Bit2: DATA10 (**SALIDA**)
- Bit3: DATA11 (**SALIDA**)
- Bit4: DATA12 (**SALIDA**)
- Bit5: DATA13 (**SALIDA**)
- Bit6: DATA14 (**SALIDA**)
- Bit7: DATA15 (**SALIDA**)

(Apéndice A.2.2 - Línea 330) El **puerto E** se usa como **puerto de control 2**, por lo tanto debemos configurarlo como de salida.

- Bit0: $\overline{XRESET}$ (**SALIDA**)

- Bit1: $\overline{WE}$ (**SALIDA**)
- Bit2: *INC_ADDR* (**SALIDA**)

Además de configurarlo lo inicializamos desactivando todas las líneas.

(Apéndice A.2.2 - Línea 341) El **puerto C** se usa para la comunicación y control del **decodificador**, lo configuramos en función de las líneas.

- Bit0: *BSYNC* (**SALIDA**)
- Bit1: *DREQ* (**ENTRADA**)
- Bit2: $\overline{XCS}$ (**SALIDA**)
- Bit3: *SCLK* (**SALIDA**)
- Bit4: *SO* (**ENTRADA**)
- Bit5: *SI* (**SALIDA**)
- Bit6: *DCLK* (**SALIDA**)
- Bit7: *SDATA* (**SALIDA**)

También en este punto inicializamos alguna de las líneas. Desactivamos la línea de sincronismo de bit (*BSYNC*) y la de selección de chip ($\overline{XCS}$).

Una vez configurados los puertos del microcontrolador pasamos a configurar uno de los tres temporizadores internos de los que dispone. El temporizador nos va a servir para activar a través de la rutina de interrupción los FLAGS que indican cuando es el momento de realizar diversas tareas.

El principal cometido del temporizador va a ser el de generar una señal de interrupción que marque el momento de recargar el buffer interno del decodificador, de forma que este no se vacíe y la reproducción no se corte.

El buffer del decodificador MP3 es de 16384 bits, como la velocidad de reproducción maxima es de 320 Kbits por segundo, tenemos que recargar el buffer del decodificador como máximo cada:

$$\frac{16384\ bits}{320 * 1024\ \frac{bits}{segundo}} = 0,05\ segundos = 50\ milisegundos$$

Como vemos obtenemos un tiempo máximo para la temporización de 50 milisegundos, por lo que el tiempo que tenemos que ajustar para la interrupción ha de ser algo menor. La interrupción del temporizador también se va a usar para otras cosas como por ejemplo para actualizar el contador de tiempo de reproducción transcurrido, este temporizador debe actualizarse a un ritmo de una vez por segundo, por lo que sería aconsejable que un múltiplo entero del tiempo entre interrupciones nos diera la latencia de 1 segundo deseada. Si usamos un valor para el temporizador de $\frac{1}{40}$ de segundo (**25 milisegundos**) conseguimos los dos objetivos:

- Que el tiempo entre interrupciones sea adecuado para evitar que el buffer se vacíe.
- Que el tiempo entre interrupciones sea un submúltiplo de un segundo.

Como el microcontrolador funciona a 20 MHz el ritmo del contador para el temporizador será de $\frac{FOsc}{4}$ es decir de 5MHz, a este ritmo y usando la preescala máxima, el temporizador 0 (TIMER0) nos generará una interrupción cada:

$$Retardo\ entre\ interrupciones\ (TMR0) = \frac{(256 - InitTMR0) * preescala}{\frac{FOsc}{4}}$$

El retardo máximo lo obtendremos inicializando el temporizador a 0 y con la preescala de 256. Con estos datos obtenemos un retardo máximo de 0,0131072 segundos, es decir unos **13 milisegundos**, claramente insuficiente.

Si usamos el temporizador 1 (TIMER1) la cosa cambia, el contador es de 16 Bits y la preescala máxima es de 8. Con este temporizador el retardo máximo entre interrupciones es:

$$Retardo\ entre\ interrupciones\ (TMR1) = \frac{(65536 - InitTMR1) * preescala}{\frac{FOsc}{4}}$$

Con estos datos el retardo máximo es de 0,1048576 segundos, es decir, unos **105 milisegundos**, más que suficiente.

El valor con el que tenemos que inicializar el temporizador se puede calcular fácilmente sin más que despejar *InitTMR1* de la fórmula anterior:

$$InitTMR1 = 65536 - \frac{Retardo\ entre\ interrupciones\ (TMR1) * \frac{FOsc}{4}}{preescala}$$

Si ajustamos el valor de la preescala a 4, obtenemos un valor de inicialización del temporizador de **34286 = 0x85EE**. Como vemos está muy próximo a 0x8600 que tiene su parte baja a 0 con lo que eso supone a la hora de reajustar el contador. Si usamos **0x8600** para inicializar el temporizador, conseguimos una interrupción cada **0,0249856 milisegundos** que se aproxima mucho a los 25 milisegundos buscados.

(Apéndice A.2.2 - Línea 398) Procedemos pues a inicializar el temporizador 1 con los siguientes parámetros:

- Preescala = 4
- Oscilador parado.
- Fuente de reloj interna.
- Temporizador detenido.
- Contador inicializado a 0x8600

Hasta que no entremos en la zona de navegación no es necesario conectar el temporizador, ya que para la zona de inicialización no son necesarias las interrupciones, es puramente lineal.

(Apéndice A.2.2 - Línea 414) El temporizador generará una interrupción cada 25 milisegundos, pero para que esto sea posible primero tenemos que configurar y habilitar las interrupciones.

Para configurar las interrupciones llevamos a cabo las siguientes tareas:

- Desactivar las interrupciones globales.
- Desactivar todas las interrupciones.
- Borrar todos los FLAGS de interrupción.
- Activar las interrupciones del temporizador 1.
- Activar las interrupciones de los periféricos.

Cuando hayamos entrado en la zona de navegación, procederemos a activar las interrupciones globales.

Una vez configurado el microcontrolador, pasamos a configurar los periféricos externos.

(Apéndice A.2.2 - Línea 428) Lo primero que configuraremos será el display LCD, para ello hacemos uso de la función que que desempeña esta tarea dentro del código de control del display *InicializaLCD*.

(Apéndice A.2.2 - Línea 445) El decodificador de MP3 también necesita ser correctamente configurado. Como en el caso del display LCD, usamos la función que para este propósito ha sido incluida en el código de control del decodificador.

(Apéndice A.2.2 - Línea 475) El último dispositivo que necesita ser configurado es el dispositivo IDE. En este caso el proceso de configuración es muy sencillo, basta con resetear por software el dispositivo mediante la función *IDE_Reset* y seleccionar el modo de direccionamiento LBA con *IDE_LBAon*. Ambas funciones están presentes en el código de control del dispositivo IDE.

(Apéndice A.2.2 - Línea 475) Llegados a este punto ya tenemos inicializados todos los periféricos, pero el proceso de navegación no puede iniciarse hasta que el sistema de ficheros no sea reconocido y se obtengan todos los parámetros necesarios para interpretar correctamente el contenido del disco duro. Esto se lleva a cabo por medio de la función presente en el código de interfaz FAT32 *Inicializa_FAT32*.

Tan sólo resta configurar adecuadamente diversas variables de programa para poder entrar en la zona de navegación, así por ejemplo tenemos que inicializar el contador de interrupciones para generar el flag de 1 segundo, los offsets y flags de scroll, tenemos que borrar la zona de memoria que almacena los nombres de directorio y fichero...

Además durante el proceso de configuración de los periféricos se va mostrando a través del display LCD diversa información relevante: mensaje de bienvenida, versión del firmware, inicialización del dispositivo IDE, sistema de ficheros detectado...

(Apéndice A.2.2 - Línea 625) La zona de inicialización termina con la activación del temporizador 1 y la activación de las interrupciones globales con lo que pasaremos a ejecutar el código de navegación y de la rutina de interrupción.

3.5.2. Rutina de Interrupción

Una vez que se activan las interrupciones, en el momento que se desborde el contador del temporizador 1 (cada 25 milisegundos) se pasa a ejecutar la rutina de interrupción.

La rutina de interrupción tiene dos misiones principales:

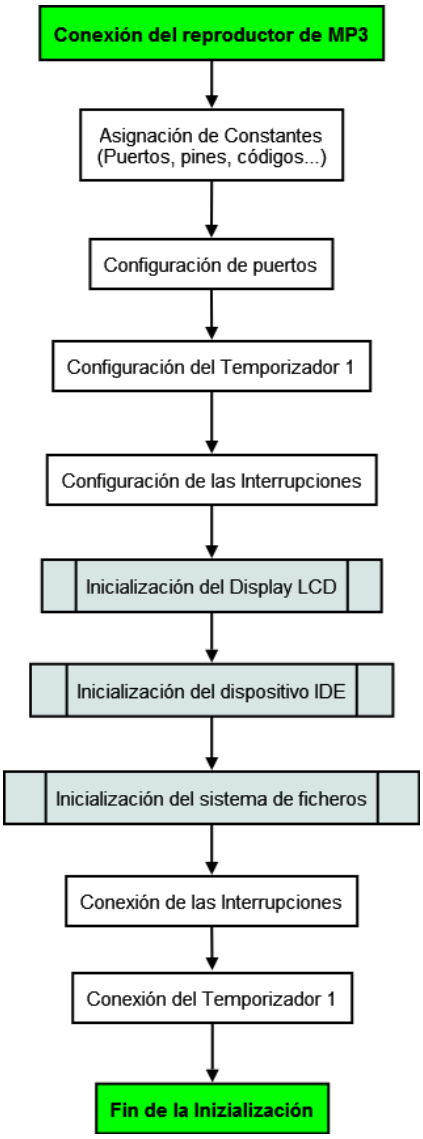


Figura 3.9: Fase de inicialización del dispositivo

1. Recargar el buffer interno del decodificador de MP3 para evitar que se vacíe y se detenga la reproducción.

2. Activar los FLAGS encargados de indicar cuando hay que realizar ciertas tareas como:

   - Realizar el scroll del nombre de directorio.
   - Realizar el scroll del nombre de fichero.
   - Incrementar el contador de tiempo.
   - Permutar en la pantalla la información de Bit o Sample Rate.
   - Aceptar una pulsación de botón como buena.

El funcionamiento de la rutina de interrupción es muy sencillo. Tiene tres zonas claramente diferenciadas, la zona de entrada a la rutina, la zona de modificación de FLAGS y recarga del buffer y la zona de salida de la rutina.

Entrada a la rutina de interrupción

(Apéndice A.2.2 - Línea 91) Lo primero que hacemos nada más entrar en la rutina de interrupción es deshabilitar las interrupciones globales para que no se produzcan posibles anidamientos de interrupciones.

A continuación nos ocupamos de salvaguardar las variables de contexto. El contenido de todas las variables de programa que se usen tanto en la rutina como en el resto del código ha de ser preservado, de forma que cuando salgamos de la rutina, la ejecución del código principal pueda continuar como si no hubiese pasado nada.

Las principales variables que debemos preservar son:

- El registro *W*
- El registro de estado: *STATUS*
- Los 5 bits más altos del contador de programa: *PCLATH*
- El puntero *FSR*
- Los registros de control: *CONTROL1* y *CONTROL2*
- El BUS de Datos: *DATOS_ALTOS* y *DATOS_BAJOS*
- La configuración del BUS de Datos (entrada/salida): *TRISB* y *TRISD*

Modificación de FLAGS y recarga del buffer del decodificador

(Apéndice A.2.2 - Línea 146) La interrupción se produce con el objetivo principal de recargar el buffer del decodificador, por eso se establece un tiempo entre interrupciones de 25 milisegundos, pero también es necesario llevar a cabo otra serie de tareas que controlan el funcionamiento del dispositivo.

Lo primero que hacemos es recargar el contador del temporizador 1 con el valor necesario para que cuente 35 milisegundos. Lo hacemos ahora y no al final de la rutina de interrupción, para que el tiempo que pase entre interrupciones sea lo más proximo posible a

25 milisegundos y para ello deberemos incluir en el tiempo que pasa entre las interrupciones el tiempo de la propia interrupción.

En esta zona además se activan los indicadores que desencadenan la ejecución de las partes del código que tienen que ver con aspectos temporales.

- **Activación del scroll del nombre de directorio:** Se activa el FLAG encargado de comunicar al programa principal que ha de llevar a cabo el desplazamiento de la línea 1 que contiene el nombre del directorio actual almacenado en memoria y que proceda a su escritura en el display LCD.
- **Activación del scroll del nombre de fichero:** Se activa el FLAG encargado de comunicar al programa principal que ha de llevar a cabo el desplazamiento de la línea 2 que contiene el nombre del fichero actual almacenado en memoria y que proceda a su escritura en el display LCD.
- **Actualización del contador de tiempo:** Mediante la activación de este FLAG, se le dice al programa principal que ha de llevar a cabo la actualización del tiempo de reproducción transcurrido (se incrementa en 1 segundo) y que escriba el nuevo tiempo en el display LCD.
- **Activación de la recepción de botón:** Una vez pulsado un botón, se desactiva la recepción de pulsaciones de botón durante un tiempo (500 milisegundos) para que no se origine una cadena de pulsaciones. En este punto, reactivamos la recepción de las pulsaciones mediante la activación de un FLAG.
- **Permutación entre el Bit y el Sample Rate:** Cuando se está reproduciendo una canción, la activación de este FLAG le dice al programa principal que ya ha transcurrido el tiempo prefijado para permutar la información mostrada entre Bit Rate y Sample Rate.

(Apéndice A.2.2 - Línea 212) Para llevar a cabo la recarga del buffer del decodificador, hacemos uso de la función que para este propósito se ha incluido en el código: *CargarBufferMP3*. Pero antes debemos comprobar si se está reproduciendo o no un fichero (testando los FLAGS), ya que las interrupciones se activan nada más entrar en la zona de navegación y es posible que cuando se ejecute la rutina de interrupción no se esté reproduciendo ninguna canción.

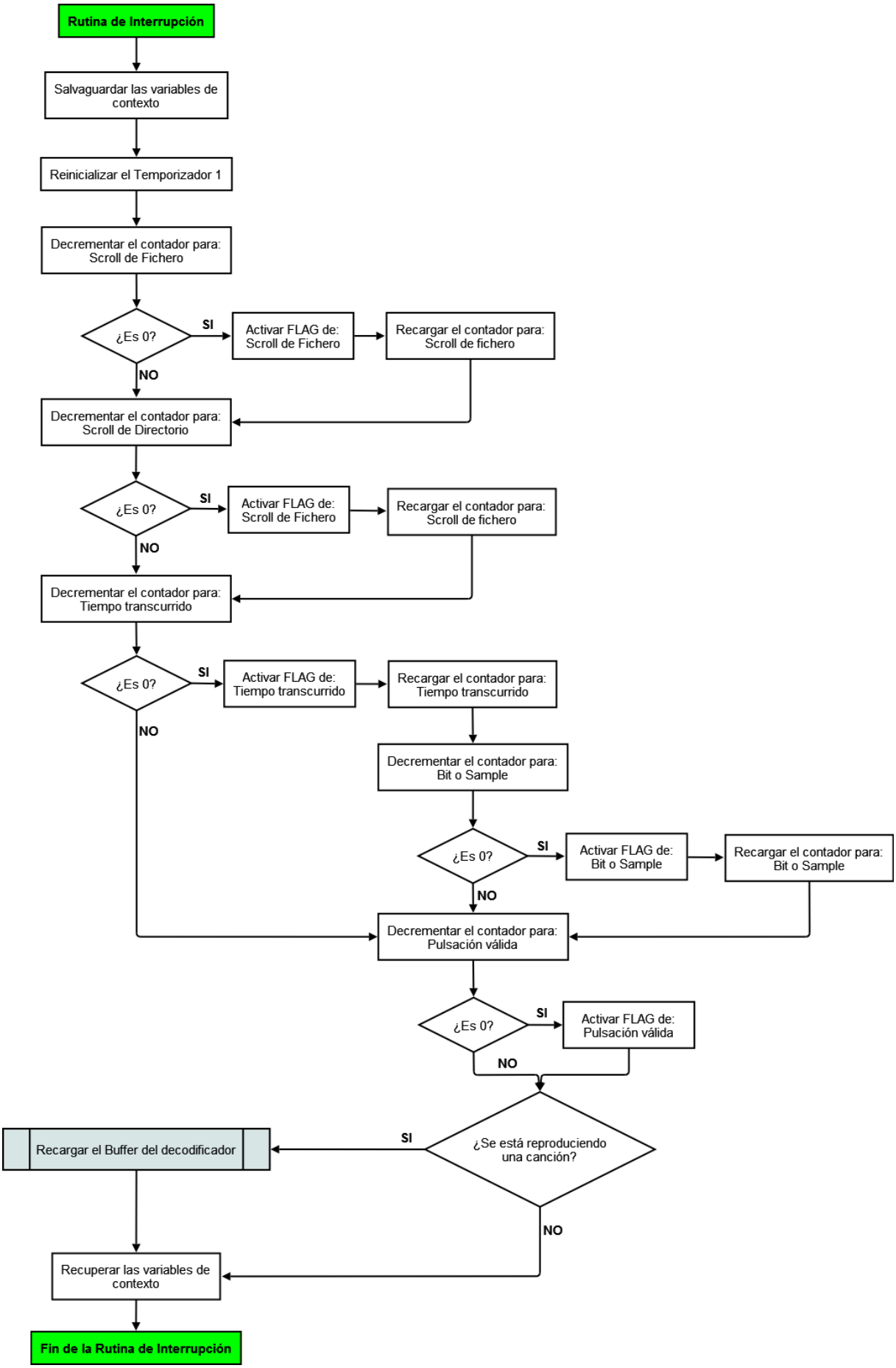


Figura 3.10: Rutina de Interrupción

Salida de la subrutina de interrupción

(Apéndice A.2.2 - Línea 226) A la hora de salir de la rutina debemos asegurarnos de que todo se queda en el mismo estado en que estaba antes de entrar en ella. Para ello debemos restaurar todas las variables que hayan sido modificadas dentro de la rutina (las que habíamos salvaguardado previamente), esto lo debemos hacer en un orden preciso, ya que si no es posible que no se recuperen correctamente. (P.ej. debemos de configurar el BUS de Datos correctamente antes de restaurar las palabras de control, no sea que se active algún periférico y se produzca un cortocircuito)

3.5.3. Zona de navegación

En esta sección del código es donde el programa principal implementa la interfaz con el usuario. Una vez configurados e inicializados todos los periféricos, podemos comenzar a recibir a través de los botones, las decisiones del usuario.

En la zona de inicialización de periféricos, además de configurar e inicializar los periféricos (internos y externos al microcontrolador) también se inicializaron las variables de navegación, de forma que cuando entramos en la zona de navegación, el dispositivo se encuentra situado en el directorio raíz del disco duro.

(Apéndice A.2.2 - Línea 633) Lo primero que nos encontramos al entrar en la zona de navegación es la recogida de una pulsación de botón por parte del usuario mediante el uso de la función *Escanea_Botones*.

En este punto, el usuario tiene la opción de pulsar 5 botones:

- *Directorio siguiente:* Hace que el dispositivo busque el siguiente directorio al actual dentro del directorio raíz, de forma que la pulsación de los botones *fichero siguiente* y *fichero anterior* muestren el contenido de este directorio. Si no existiese un directorio siguiente, permanece en el actual.
- *Directorio anterior:* El dispositivo vuelve al directorio que precede al actual dentro del directorio raíz. Si existiese, los botones de *fichero siguiente* y *fichero anterior* mostrarán el contenido del directorio raíz.
- *Fichero siguiente:* Hace que el dispositivo muestre el fichero siguiente al actual dentro del directorio actual. Si no existe un fichero siguiente, permanece en el actual.
- *Fichero anterior:* Hace que el dispositivo muestre el fichero anterior al actual dentro del directorio actual. Si no existe un fichero anterior, permanece en el actual.

- *Reproducir/Parar:* Inicia la reproducción del fichero actual (el que se muestra en pantalla). Si no se ha seleccionado ningún fichero (la línea de fichero en la pantalla está en blanco) la pulsación de este botón no tiene ningún efecto. Si ya se está reproduciendo un fichero, la pulsación de este botón detiene la reproducción.

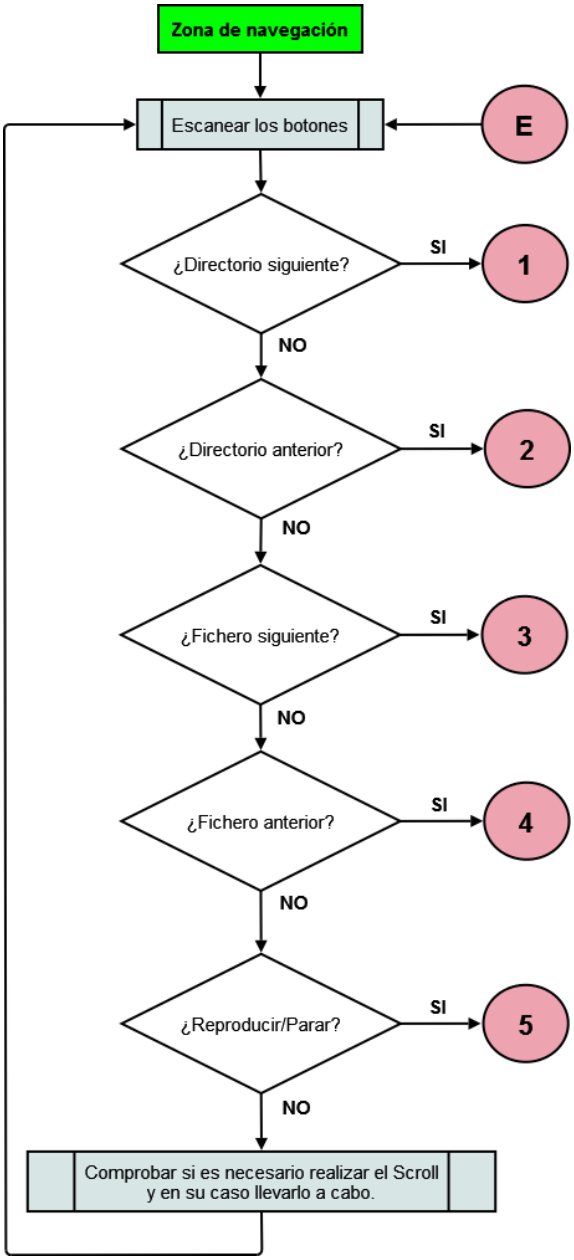


Figura 3.11: Escaneo de los botones

En función del botón que pulsemos, el programa salta a la zona de código encargada de tratar la acción requerida.

En el caso de que se pulse uno de los cuatro primeros botones (*Directorio siguiente, Directorio anterior, Fichero siguiente, Fichero anterior*) las acciones que se desencadenan son muy parecidas:

1. Se borra el nombre de directorio/fichero almacenado en la memoria SRam.
2. Se reinician las variables de offset del scroll de nombre de directorio/fichero.
3. Se busca el nuevo directorio/fichero en el disco duro.
4. Se pone el nuevo nombre de directorio/fichero en la pantalla.
5. En el caso de que no quepa completo el nombre en la pantalla, se activan los FLAGS que indican que debe realizarse el scroll de la línea. (Estos FLAGS son distintos a los que indican CUANDO tiene que realizarse el scroll de línea, que son los que modifica la rutina de interrupción.)

Una vez que se ha actualizado la información el programa vuelve al punto en el que realiza el escaneado de los botones y permanece en este bucle hasta que el usuario decide pulsar el botón de reproducción.

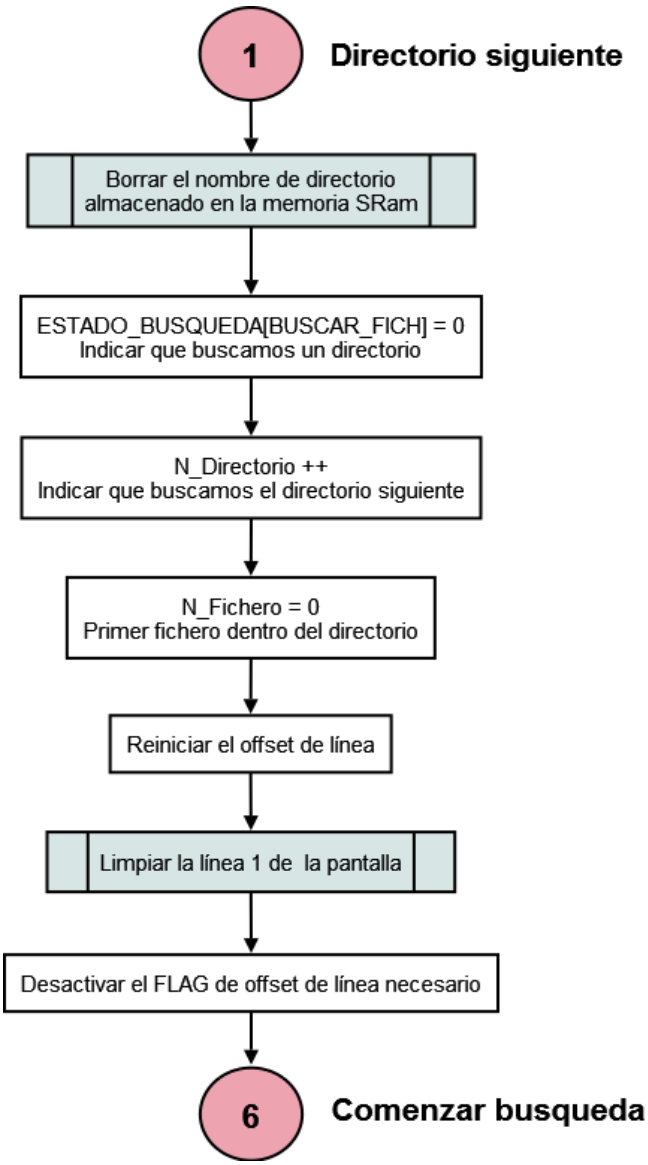


Figura 3.12: Directorio siguiente

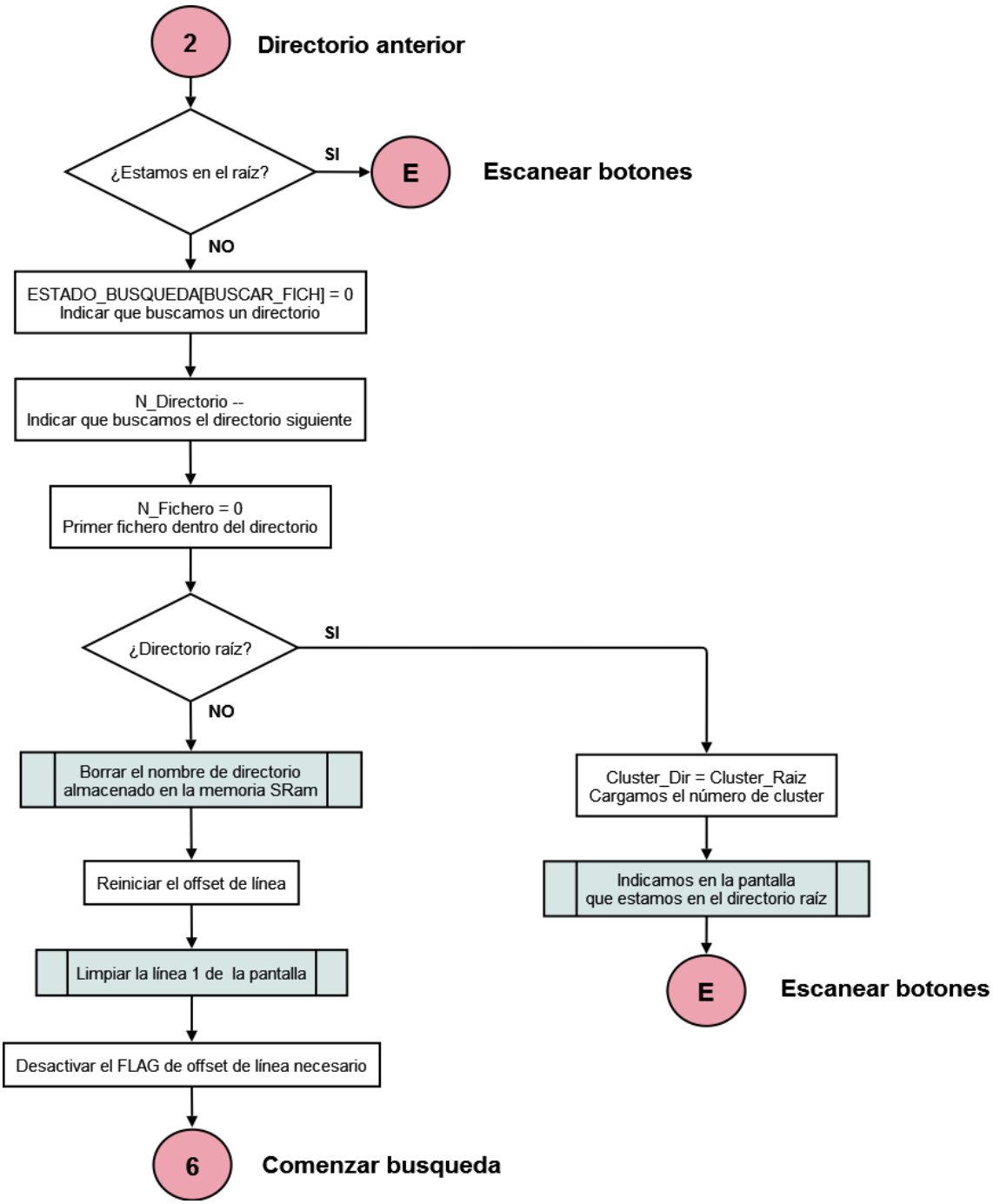


Figura 3.13: Directorio anterior

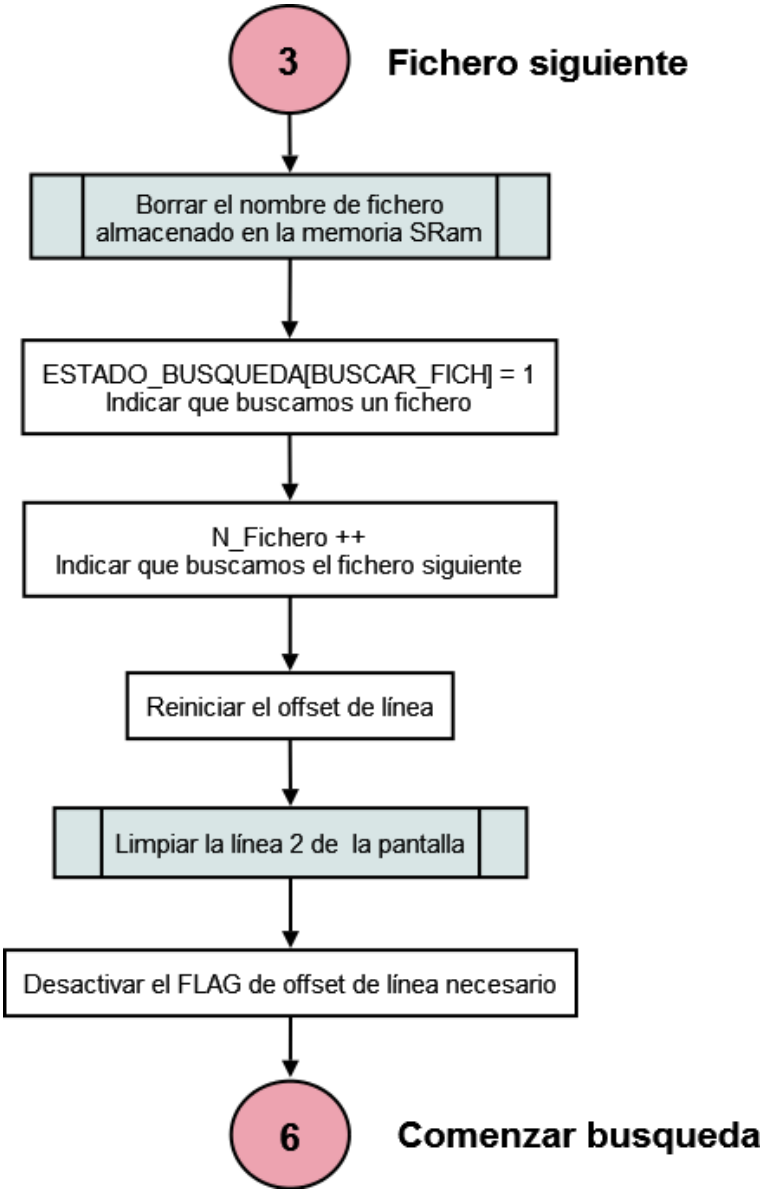


Figura 3.14: Fichero siguiente

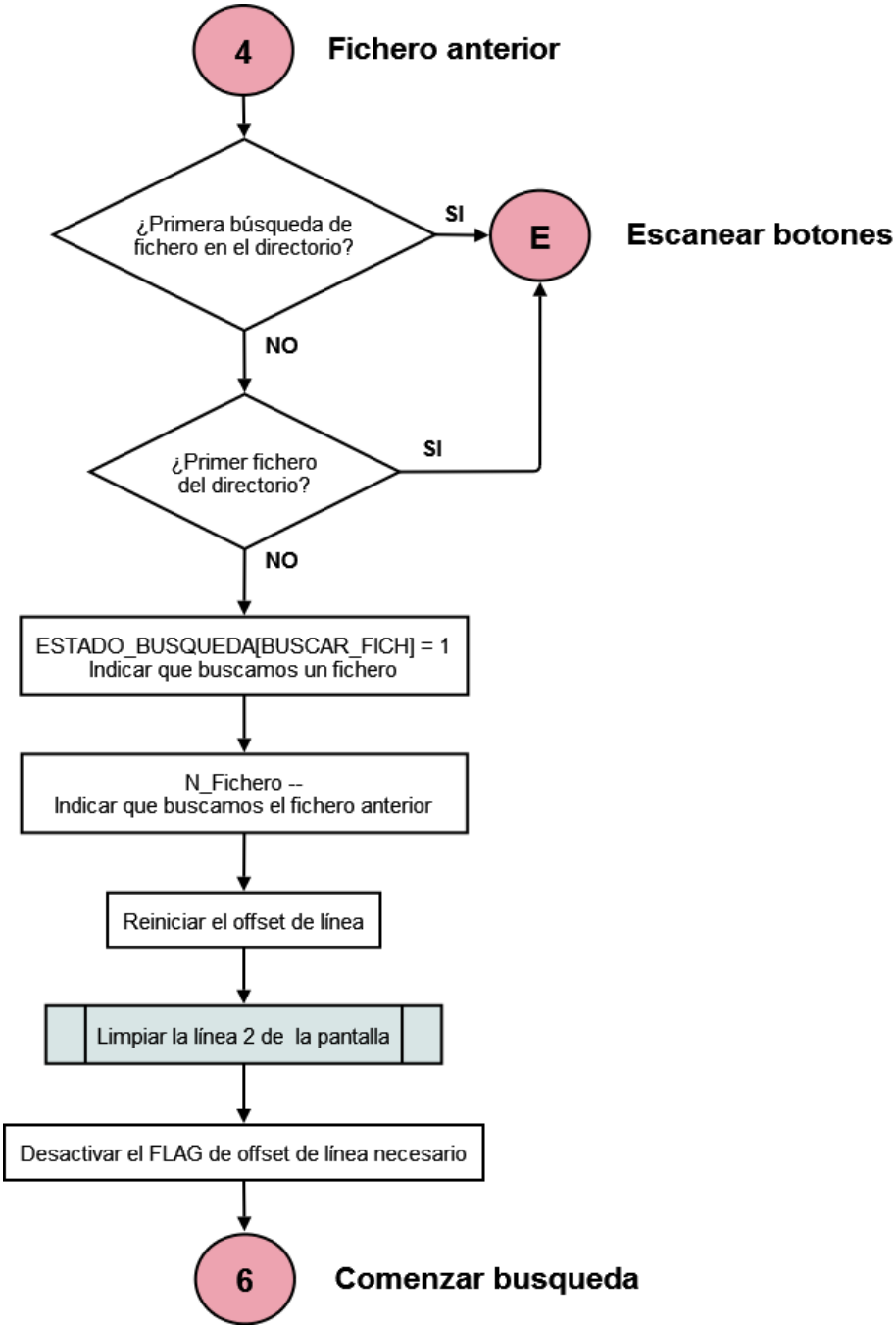


Figura 3.15: Fichero anterior

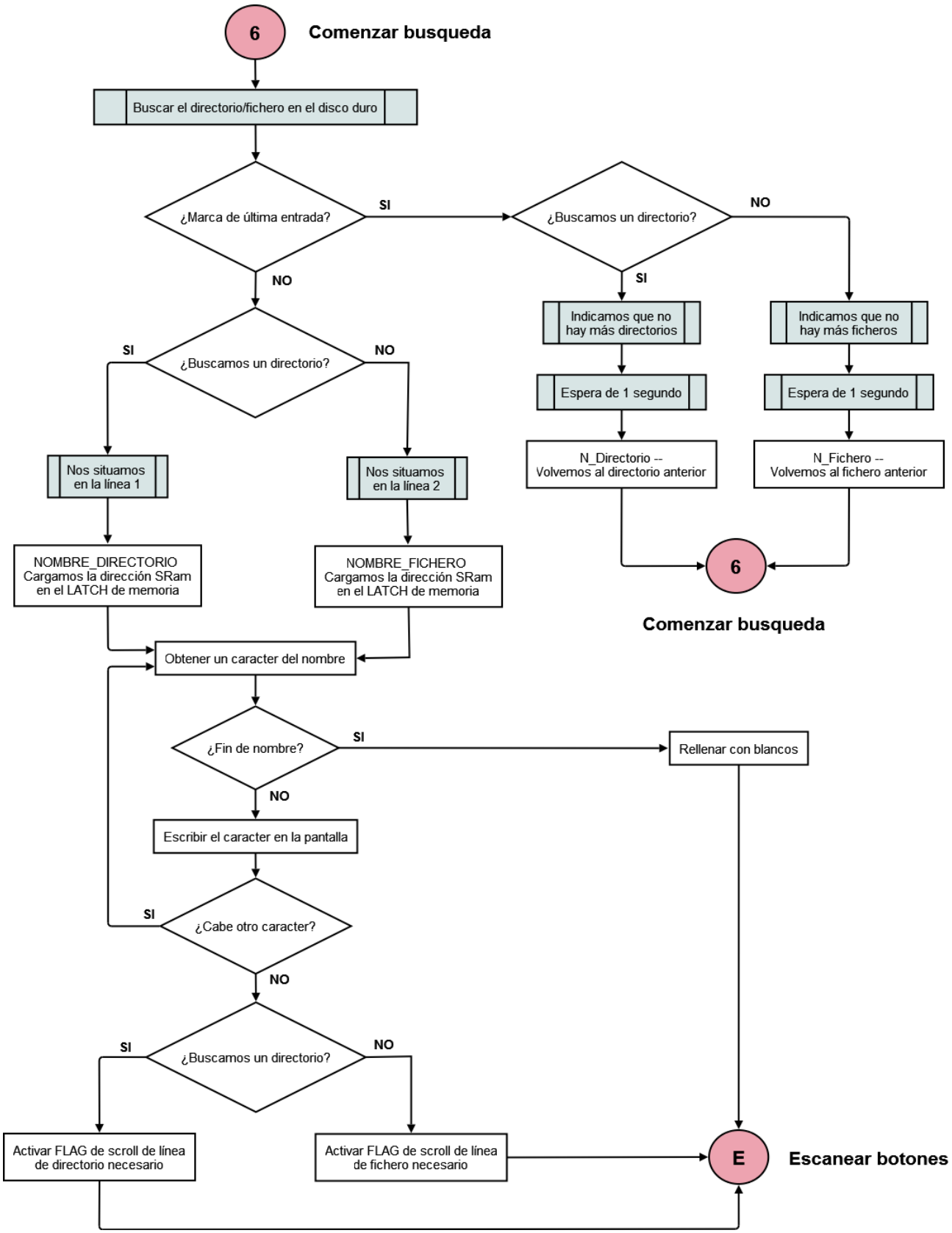


Figura 3.16: Comenzar búsqueda

(Apéndice A.2.2 - Línea 955) Cuando el usuario oprime el botón de reproducción con un fichero seleccionado, el programa principal salta a la zona de código encargada de realizar esta tarea.

Para reproducir un fichero se hace uso de un buffer de memoria intermedio (entre el disco duro y el decodificador) ubicado en la memoria SRam a partir de la dirección contenida en la constante de programa *BUFFER_REPRO*. Este buffer tiene una capacidad de 48 KBytes y para su manejo lo dividiremos en bloques de 2 KBytes. Tenemos pues un total de 24 bloques de memoria para usar en el buffer.

El rango de direcciones de memoria validas es de 0x2000 (comienzo del buffer) a 0x7FFF (dirección más alta de memoria), como cada dirección de memoria almacena 2 bytes de información (16 bits), cada bloque contendrá 0x400 posiciones de memoria.

El buffer se maneja de forma circular, de modo que la siguiente posición valida a la 0x7FFF sea la 0x2000.

Para su manejo mediante dos punteros, uno de lectura (*PuntBufferLec_H:PuntBufferLec_L*) que será modificado exclusivamente por la función encargada de recargar el buffer interno del decodificador (*CargarBufferMP3*) y otro de escritura (*PuntBufferEsc_H:PuntBufferEsc_L*) que será modificado por la función encargada de recargar el buffer intermedio SRam (*CargarBloque*).

La función que se encarga de escribir en el buffer intermedio (*CargarBloque*) es invocada desde el programa principal y la carga se efectúa por bloques, es decir se escriben los datos extraídos del disco duro en conjuntos de 2 KBytes. Para ello utiliza los punteros de lectura y escritura en el buffer para asegurarse de que hay un espacio libre de por lo menos 2 KBytes (1 bloque). El puntero de escritura se mueve de bloque en bloque, ya que se escriben de golpe 2 Kbytes, es decir, siempre apunta a la dirección de comienzo del siguiente bloque que va a escribir. Para evitar que el proceso de escritura sobrescriba el bloque del que todavía se están extrayendo los datos para su reproducción, debemos comprobar tan sólo que ambos punteros, no se encuentran en el mismo bloque. Para hacer esto basta con enmascarar los dos BITS de menor peso de de la parte alta de la dirección de cada puntero y hacer un *XOR* entre ellos, si da 0 están en el mismo bloque, si da 1 están en bloques distintos.

Al comenzar la reproducción de la canción, todo el Buffer está vació, por lo que los punteros de lectura y de escritura valen lo mismo. En este caso no podríamos escribir ningún bloque según lo que hemos dicho hasta que el puntero de lectura nos libere un

		H		**L**		
Puntero de Lectura	→	01000101		11001001		(Bloque 10)
Puntero de Escritura	→	01000100		00000000		(Comienzo del Bloque 10)
		H		**MÁSCARA**		
Puntero de Lectura	→	01000101	AND	11111100	→	01000100
Puntero de Escritura	→	01000100	AND	11111100	→	01000100
		01000100	XOR	01000100	→	00000000 (Mismo bloque)
		H		**L**		
Puntero de Lectura	→	00110100		10001000		(Bloque 06)
Puntero de Escritura	→	01110100		00000000		(Comienzo del Bloque 22)
		H		**MÁSCARA**		
Puntero de Lectura	→	00110100	AND	11111100	→	00110100
Puntero de Escritura	→	01110100	AND	11111100	→	01110100
		00110100	XOR	01110100	→	01000000 (Distintos Bloques)

Cuadro 3.26: Ejemplo de la comprobación antes de la escritura en el buffer

bloque, pero el puntero de lectura no puede liberar ningún bloque por que no hay nada en ellos. La solución a este problema es tan simple como realizar una precarga del buffer intermedio, de forma que cuando la función de lectura sea invocada por primera vez, tenga datos listos para ser enviados al decodificador.

Si el proceso de escritura es más rápido que el de lectura, la situación normal será tener el proceso de escritura parado esperando a que se libere un bloque para rellenarlo.

En el caso de que el proceso de lectura por alguna razón sea más rápido que el proceso de escritura, podría darse el caso de que los dos punteros coincidiesen en algún momento. Si el proceso de lectura no hace ninguna comprobación, seguirá mandando datos (INCORRECTOS) al decodificador (No han sido cargados todavía) y el proceso de escritura se quedaría bloqueado hasta que el de lectura abandone su bloque... esto nos llevaría a un total de 48KB (todo el Buffer) de datos incorrectos mandados al decodificador.

El buffer interno del decodificador tiene una capacidad de 16384 bits o lo que es lo mismo 2 KBytes. Esto nos dice que en el proceso de recarga del buffer interno del decodificador (*CargarBufferMP3*), se van a transferir como mucho 2 KBytes (1 bloque) aunque

		H		**L**		
Puntero de Lectura	→	01000101		11001001		(Bloque 10)
Puntero de Escritura	→	01001000		00000000		(Comienzo del Bloque 11)
		H		**MÁSCARA**		
Puntero de Lectura	→	01000101	AND	11111100	→	01000100
Sumamos 4	→	01000100	+	00000100	→	01001000
		H		**MÁSCARA**		
Puntero de Escritura	→	01001000	AND	11111100	→	01001000
		01001000	XOR	01001000	→	00000000 (Mismo bloque)
		H		**L**		
Puntero de Lectura	→	01000101		11001001		(Bloque 10)
Puntero de Escritura	→	01010000		00000000		(Comienzo del Bloque 13)
		H		**MÁSCARA**		
Puntero de Lectura	→	01000101	AND	11111100	→	01000100
Sumamos 4	→	01000100	+	00000100	→	01001000
		H		**MÁSCARA**		
Puntero de Escritura	→	01010000	AND	11111100	→	01010000
		01001000	XOR	01010000	→	00011000 (Distintos Bloques)

Cuadro 3.27: Ejemplo de la comprobación antes de la lectura del buffer

la transferencia se hace en conjuntos de 2 bytes. Si por ejemplo el puntero de Lectura esta en la mitad del bloque 10 y el puntero de escritura esta a comienzo del bloque 11, entonces cuando se inicie el proceso de lectura, en el peor de los casos, va a avanzar un bloque entero, es decir, se va a quedar a la mitad del bloque 11 y por lo tanto va a sobrepasar al puntero de escritura. Para evitar esto, lo que tenemos que comprobar es que el puntero de escritura no se encuentra en el bloque siguiente al bloque en el que esta el puntero de lectura. Esto lo podemos hacer fácilmente sumando 4 a la parte alta de la dirección del puntero de lectura enmascarado y comprobando que el resultado no coincide con la parte alta del puntero de escritura enmascarado.

Una vez hemos precargado el buffer intermedio entre el disco duro y el decodificador entramos en el bucle de reproducción.

En este bucle se lleva a cabo la carga del buffer intermedio, pero también se realizan otras funciones:

- Se muestra el tiempo de reproducción.
- Se muestra la información relativa al Bit Rate y el Sample Rate.
- Se atiende a la pulsación de los botones: durante la reproducción de la canción podemos actuar sobre tres de los botones del dispositivo.
    - *Subir el volumen*: Sube el volumen del audio de salida del reproductor.
    - *Bajar el volumen*: Baja el volumen de salida del audio del reproductor.
    - *Reproducir/Parar*: Detiene la reproducción de la canción.

Una vez que se detecta el final de la canción mediante la lectura del indicador de *último cluster de la cadena*, se procede a cargar el buffer intermedio de 0's, ya que cuando leamos el indicador, todavía quedarán datos sin mandar al decodificador y además, una vez que todos los datos de la canción hayan sido enviados al decodificador, todavía quedarán datos en el buffer interno de este sin reproducir. Al rellenar el buffer de 0's nos aseguramos de que incluso el buffer intermedio se vacié antes de parar la reproducción.

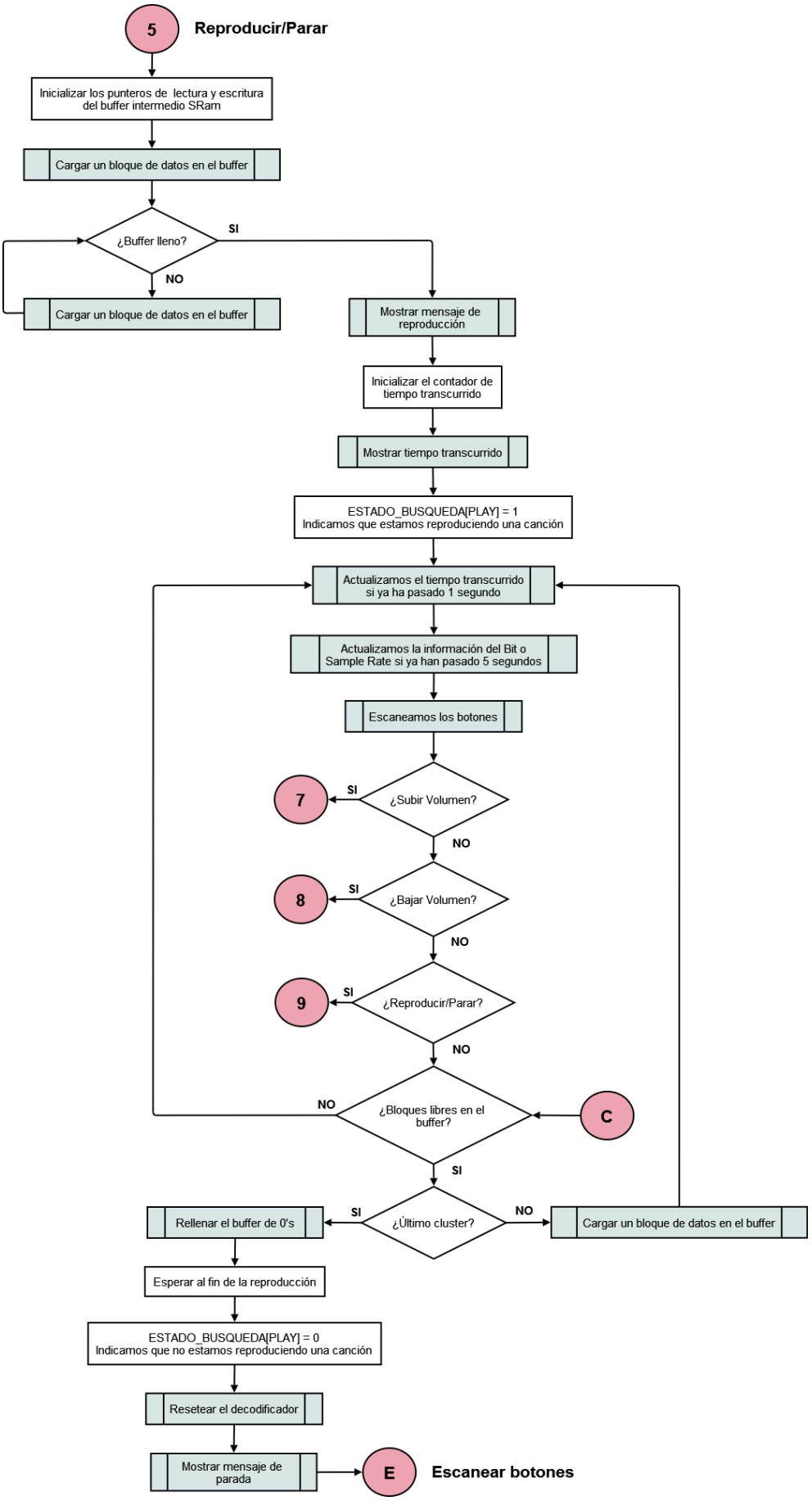


Figura 3.17: Reproducción de un fichero

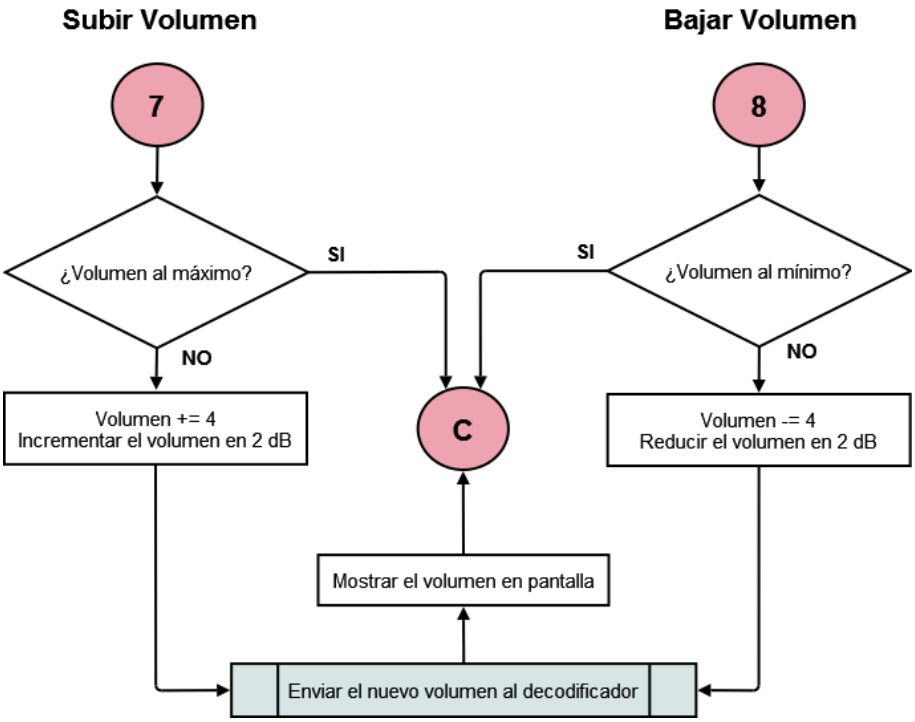


Figura 3.18: Modificación del volumen

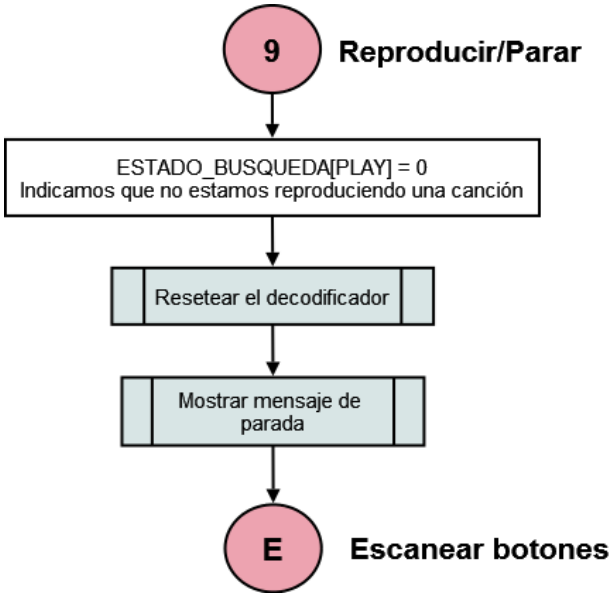


Figura 3.19: Parada de la reproducción

3.5.4. Funciones

La mayoría de las funciones que usa el programa principal están incluidas en ficheros de librería en función de la parte del dispositivo al que hacen referencia. Así tenemos un fichero de funciones que engloba todo lo referente al manejo del dispositivo IDE, otro para el display LCD, etc.

Sin embargo hay una serie de funciones que están incluidas en el código del programa principal:

- Funciones que acceden al buffer intermedio.
- Funciones de configuración del BUS de Datos.
- Funciones que proporcionan lapsos temporales.

CargarCeros

(Apéndice A.2.2 - Línea 1219) Esta función se encarga de escribir en el buffer intermedio un total de 2048 ceros, es decir un bloque entero.

Se usa para evitar que una vez que se han acabado de mandar todos los datos de la canción al decodificador, se corte la canción, lo cual ocurriría si regleteáramos el decodificador cuando este todavía no ha acabado de vaciar el buffer interno que posee.

Una vez que hayamos mandado todos los ceros al decodificador, podremos resetearlo, ya que nos habremos asegurado de que por lo menos los datos válidos de la canción ha sido reproducidos. Si se reproducen los ceros no importa, ya que se interpretan como silencio.

CargarBloque

(Apéndice A.2.2 - Línea 1296) Esta función se encarga del proceso de escritura en el buffer. Realiza una transferencia de un bloque entero (2048 bytes) de datos de la canción del disco duro al buffer intermedio.

Para ello usa el puntero de escritura en el buffer como dirección de destino de los datos.

Lleva continuamente la cuenta de los sectores que quedan por transferir del cluster actual y en caso de que se agoten los sectores del cluster leído da la orden al dispositivo IDE de leer el siguiente cluster.

Una vez a transferido el sector al buffer intermedio, modifica el puntero de escritura a su nueva posición sumando 4 a su parte alta (0x0400 = 1024 posiciones de memoria de 16 bits = 2048 bytes) teniendo en cuenta siempre que el buffer intermedio es circular.

CargarBufferMP3

(Apéndice A.2.2 - Línea 1367) Esta es la función que se encarga de extraer los datos del buffer intermedio para mandarlos al decodificador.

Es invocada desde la rutina de interrupción cada 25 milisegundos y para su funcionamiento emplea el puntero de lectura del buffer intermedio como origen de los datos.

Una vez que es invocada recarga el buffer interno del decodificador hasta los topes, examinando la línea de petición de datos de la que el decodificador dispone (*DREQ*). Si *DREQ* está activa entonces el buffer del decodificador es capaz de recibir al menos 32 bytes de datos. Las transferencias se hacen pues de 32 en 32 bytes, comprobando cada vez si *DREQ* esta activa o no.

Los datos son mandados a través del puerto SPI en grupos de 2 bytes. Con cada transferencia se modifica el puntero de lectura, incrementándolo en una posición y teniendo en cuenta que el buffer intermedio es circular.

BUS_Conf...

(Apéndice A.2.2 - Línea 1476) Configura adecuadamente las líneas del BUS de Datos en el microprocesador para permitir la lectura de datos del mismo *BUS_ConfLec* o la escritura en él *BUS_ConfEsc*.

Espera_L

(Apéndice A.2.2 - Línea 1514) Bucle de retardo que genera una espera activa de un múltiplo de 20 milisegundos.

La función recibe un número a través de W que se interpreta com el número de veces que se realiza la espera de 20 milisegundos.

Tiempo de espera = W * 20 milisegundos

Espera_C

(Apéndice A.2.2 - Línea 1543) Bucle de retardo que genera una espera activa de un múltiplo de 1 milisegundo.

La función recibe un número a través de W que se interpreta com el número de veces que se realiza la espera de 1 milisegundo.

Tiempo de espera = W * 1 milisegundo

3.6. Interfaz IDE - *USER CODE*

Parte del código de usuario que se encarga de manejar el módulo IDE-ATA. Contiene todas las funciones necesarias para acceder al dispositivo de almacenamiento masivo IDE según el interfaz ATA-1.

Se encuentra ubicado en la **Página 0** de la memoria de programa.

Esta parte del código comienza con la declaración de las constantes que almacenan las palabras de control que han de ser mandadas al dispositivo para acceder a los registros de control IDE. Para acceder a los distintos registros del dispositivo IDE, ya sea para leerlos o para escribirlos, tenemos que poner en las líneas de control ($DA0$, $DA1$, $DA2$, $\overline{CS0}$ y $\overline{CS1}$) unos valores determinados. **(Apéndice A.3 - Línea 37)**

Las líneas de control se conectan al BUS de Datos a través de un LATCH de memoria que almacenara la palabra de control. La correspondencia entre líneas es la siguiente:

BUS de Datos		Líneas de control
BIT 0	→	$DA0$
BIT 1	→	$DA1$
BIT 2	→	$DA2$
BIT 3	→	$\overline{CS0}$
BIT 4	→	$\overline{CS1}$

Cuadro 3.28: Correspondencia entre Líneas de control IDE y el BUS de Datos

También se declaran como constantes todos los códigos de comando que podemos ejecutar sobre el dispositivo IDE junto con su correspondiente código, que sera el que tengamos que introducir en el registro que para tal efecto provee el dispositivo (Command → $0x17$). **(Apéndice A.3 - Línea 67)**

3.6.1. Funciones

La forma en la que se accede a los registros IDE en las funciones es la siguiente:

1. se carga el código del registro que se quiere leer o escribir en el LATCH de control IDE.
2. Se pone el nuevo valor en el BUS de Datos (**sólo escritura**)
3. Se activa la línea de lectura ($\overline{DIOR}$) o de escritura ($\overline{DIOW}$)
4. Se reciben los datos por el BUS de DATOS (**sólo lectura**)

IDE_ObtenerEstado

(Apéndice A.3 - Línea 193) Lee el registro de estado del dispositivo IDE (Status $\rightarrow 0x17$) y deja almacenado su valor en W para su posterior tratamiento.

IDE_EsperarListo

(Apéndice A.3 - Línea 229) Invoca a la función *IDE_ObtenerEstado* y comprueba si el dispositivo está listo para recibir comandos (BSY=0 y $DRDY$=1). Si el dispositivo no esta listo, permanece en un bucle hasta que lo esté, momento en el cual abandona la función.

IDE_EsperarDatos

(Apéndice A.3 - Línea 252) Invoca a la función *IDE_ObtenerEstado* y comprueba si el dispositivo está listo para transmitir datos al PIC (BSY=0, $DRDY$=1 y DRQ=1). Si el dispositivo no esta listo, permanece en un bucle hasta que lo esté, momento en el cual abandona la función.

IDE_Reset

(Apéndice A.3 - Línea 275) Invoca a la función *IDE_ObtenerEstado* y comprueba si el dispositivo está listo para recibir un comando, momento en el cual se le envía la orden a través del registro de control correspondiente (Device Control $\rightarrow 0x0E$) de provocar un RESET por Software y de inhibir las interrupciones al HOST. ($SRST$=1 y $\overline{IEN}$=0).

IDE_LBAon

(Apéndice A.3 - Línea 321) Invoca a la función *IDE_ObtenerEstado* y comprueba si el dispositivo está listo para recibir un comando. Selecciona como dispositivo de trabajo el 0 y activa el modo de direccionamiento LBA. (DEV=0 y L=1) (Device Register → $0x16$)

IDE_Identifica

(Apéndice A.3 - Línea 359) Invoca a la función *IDE_ObtenerEstado* y comprueba si el dispositivo está listo para recibir un comando. Le da la orden al dispositivo IDE de identificación, y transfiere los 512 bytes generados a la zona de memoria **SRam** que comienza en la posición almacenada en las variables de programa *SRam_Hi:SRam_Lo*. (Command → $0x17$)

IDE_Standby

(Apéndice A.3 - Línea 407) Invoca a la función *IDE_ObtenerEstado* y comprueba si el dispositivo está listo para recibir un comando. Pasamos el dispositivo IDE a estado de STANDBY, deja de girar para ahorrar energía pero sigue respondiendo a comandos de forma normal.

La latencia de los datos cuando esta en STANDBY aumenta considerablemente. El tiempo que tarda el dispositivo en entrar en modo STANDBY, queda determinado por el valor del registro *Sector Count* del dispositivo IDE. (Sector Count → $0x12$)

Valor en Sector Count	Timeout asignado
0	Timeout deshabilitado
1-240	(valor * 5) segundos
241-251	((valor - 240) * 30) minutos
252	21 minutos
253	Periodo de entre 8 y 12 horas
254	Reservado
255	21 minutos y 15 segundos

Cuadro 3.29: Tiempos de Standby automáticos

IDE_CargarNSector

(Apéndice A.3 - Línea 454) Carga el numero de sector que se quiere leer, o en el que se quiere escribir en los registros adecuados del dispositivo IDE. (LBA bits 0-7 → $0x13$, LBA bits 8-15 → $0x14$, LBA bits 16-23 → $0x15$ y LBA bits 24-27 → $0x16$)

El numero del primer sector a leer o escribir está almacenado en las variables de programa: *IDE_Sec0* - *IDE_Sec3*.

El numero de sectores a leer o escribir ha de está almacenado en la variable de programa: *NumSectRW* y ha de ser transferido al registro IDE *Sector Count*. (Sector Count $\rightarrow 0x12$)

IDE_LeerSectores

(Apéndice A.3 - Línea 543) Recibe en W el numero de sectores a leer. Invoca a la función *IDE_ObtenerEstado* y comprueba si el dispositivo está listo para recibir un comando.

A continuación carga el número del primer sector que es fuente de los datos y que está almacenado en las variables de programa *IDE_Sec0* - *IDE_Sec3*, en el dispositivo IDE mediante la invocación de la función *IDE_CargarNSector*.

Por último da la orden al dispositivo de leer los sectores activando su línea de control $\overline{DIOR}$.

IDE_EscSectores

(Apéndice A.3 - Línea 582) Recibe en W el numero de sectores a escribir. Invoca a la función *IDE_ObtenerEstado* y comprueba si el dispositivo está listo para recibir un comando.

A continuación carga el número del primer sector que es destino de los datos y que está almacenado en las variables de programa *IDE_Sec0* - *IDE_Sec3*, en el dispositivo IDE mediante la invocación de la función *IDE_CargarNSector*.

Por último da la orden al dispositivo de escribir los sectores activando su línea de control $\overline{DIOW}$.

IDE_TransSectaSRam

(Apéndice A.3 - Línea 630) Transfiere un número determinado de sectores del dispositivo IDE a la memoria SRam.

El número de sectores a transferir se recibe por W y se almacena en la variable de programa *NumSectTR*. Como es lógico, este número ha de ser menor o igual al número de sectores que previamente se han solicitado leer del dispositivo IDE *NumSectRW*.

Los sectores transferidos son almacenados el la memoria SRam a partir de la dirección contenida en *SRam_Hi:SRam_Lo*.

La transferencia se hace en bloques, cada bloque esta formado por 32 bytes (16 Words) y como un sector contiene 16 bloques se transfieren: (16 * *NumSectTR*) bloques.

1 Sector = 16 Bloques
1 Bloque = 32 bytes (16 words)
1 Sector = 512 bytes (256 words)

IDE_TransSectaIDE

(Apéndice A.3 - Línea 721) Transfiere un número determinado de sectores de la memoria SRam al dispositivo IDE.

El número de sectores a transferir se recibe por W y se almacena en la variable de programa *NumSectTR*. Como es lógico, este número ha de ser menor o igual al número de sectores que previamente se han solicitado escribir en el dispositivo IDE *NumSectRW*.

Los sectores transferidos son leídos de la memoria SRam a partir de la dirección contenida en *SRam_Hi:SRam_Lo*.

La transferencia se hace en bloques, cada bloque esta formado por 32 bytes (16 Words) y como un sector contiene 16 bloques se transfieren: (16 * *NumSectTR*) bloques.

3.7. Interfaz FAT32 - *USER CODE*

Esta parte del código de usuario es la encargada de interpretar los datos extraídos del disco duro y que están almacenados en formato FAT32.

Es encuentra ubicada en la **Página 0** de memoria de programa y está dividido en dos ficheros fuente:

- Apéndice A.4.1: Definición de los offsets de las entradas más relevantes del disco duro. (Master Boot Record, Boot Sector, Estructura de directorio FAT...)

- Apéndice A.4.2: Definición de las funciones necesarias para acceder a los datos.

3.7.1. Offsets

Lo primero que nos encontramos en el disco duro es el MBR que nos da información acerca de las particiones presentes en el disco permitiéndonos localizar cada una de ellas. También posibilita al sistema operativo arrancar correctamente desde la partición activa. Para tratarlo adecuadamente definimos:

- Estructura del MBR. **(Apéndice A.4.1 - Línea 33)**
- Estructura de la tabla de particiones del MBR. **(Apéndice A.4.1 - Línea 46)**
- Offsets dentro de cada entrada de la tabla de particiones del MBR. **(Apéndice A.4.1 - Línea 55)**

Gracias al MBR también podemos identificar el tipo de partición que tenemos presente en cada una de las particiones, gracias a los códigos de tipo de partición. **(Apéndice A.4.1 - Línea 106)**

En partición el primera sector que nos encontramos es el Boot Sector. En este sector se describe perfectamente la partición, y de el sacamos datos clave como el número de bytes por sector, el número de FAT's o donde comienza el directorio raíz. Tiene una longitud de 512 bytes y los offsets dentro de el se pueden encontrar en **(Apéndice A.4.1 - Línea 160)** .

Con esta información podemos obtener fácilmente las direcciones de comienzo de las tablas FAT y del área de datos, que serán claves para poder extraer la información.

Dentro del área de datos, la información se encuentra almacenada en formato FAT32. Esta se encuentra dividida en entradas de 32 bytes que a su vez pueden ser:

- Entrada de directorio corta (Compatible DOS). **(Apéndice A.4.1 - Línea 244)**
- Entrada de directorio larga. **(Apéndice A.4.1 - Línea 289)**

Con toda esta información de localización y mediante las funciones de acceso podemos navegar y extraer la información que deseemos del disco duro en formato FAT32.

3.7.2. Funciones

Las funciones de acceso nos van a permitir:

1. Extraer toda la información relevante del disco duro (MBR, BS...)
2. Usar esa información para poder acceder a los datos almacenados en el disco.

La forma normal de proceder de muchas de las funciones va a ser cargar en la memoria SRam un sector (512 bytes) de información del disco duro y tratarlo para extraer de el la información que deseemos, almacenándola en la memoria del PIC. A su vez, muchos de los datos almacenados en las distintas entradas del disco duro ocupan 4 bytes, por lo que va a ser muy interesante disponer de una función que se encargue de extraer bloques de 4 bytes de estos sectores.

Lee_4BYTES

(Apéndice A.4.2 - Línea 40) Lee 4 bytes consecutivos de la dirección de memoria apuntada por *SRam_Hi:SRam_Lo* y los deposita en la memoria del PIC a partir de la dirección apuntada por *FSR*.

Inicializa_FAT32

(Apéndice A.4.2 - Línea 96) Esta función tan sólo se ejecuta una vez nada más arrancar el dispositivo, pero de ella depende el correcto funcionamiento del mismo, ya que es la encargada de extraer del disco toda la información necesaria para poder navegar correctamente por el sistema de ficheros.

Para ello descarga y procesa el Master Boot Record y el Boot Sector obteniendo de cada uno de ellos la siguiente información:

Del procesado del Master Boot Record obtenemos:

- La **dirección de comienzo de la partición** en la que se encuentran almacenados los datos.
- El **formato del sistema de ficheros** en el que se encuentran almacenados los datos. Nos sirve para comprobar que el sistema de ficheros es FAT32.
- La dirección de comienzo del **Boot Sector**.

Del procesado del Boot Sector obtenemos:

- El número de **sectores por cluster**.
- El cluster en el que comienza el **directorio Raíz**.
- El numero de **sectores reservados**.
- El número del primer sector de las **tablas FAT**.
- El número del primer sector del **área de datos**.

Calcula_SectorLBA

(Apéndice A.4.2 - Línea 486) Cuando accedemos a una entrada de directorio para extraer la dirección en la que comienzan los datos a los que hace referencia, lo que obtenemos es el número de cluster en el que comienzan dichos datos.

A nosotros lo que nos interesa es saber en que sector comienzan, por lo que necesitamos una función que a partir del número de cluster y los datos obtenidos anteriormente nos calcule el sector en el que comienzan los datos.

$$\frac{Número\ de\ cluster}{Sectores\ por\ cluster} + Primer\ sector\ del\ área\ de\ datos$$

CargaEntradaFAT32

(Apéndice A.4.2 - Línea 564) Lee una entrada completa FAT32 del sector transferido a la SRAM y la copia a la memoria del PIC a partir de la dirección *ENTRADA_FAT32*. El numero de entrada que se transfiere del sector (0 a 15) debe ser previamente cargado en la variable de programa *NumEntradaFAT32*.

Para calcular la posición en memoria SRam en la que se encuentra la entrada que se quiere copiar hacemos:

$$Dirección\ del\ sector\ en\ SRam + (16 * Número\ de\ entrada\ FAT32)$$

En nuestro caso el sector se encuentra situado en la memoria SRam a partir de la dirección $0x0000$.

ProcesaUnicode

(Apéndice A.4.2 - Línea 637) En las entradas de directorio de nombre largo, los caracteres están almacenados en formato UNICODE. Esta función se encarga de extraer un número determinado de caracteres de una entrada almacenados en formato unicode y que se encuentran seguidos en memoria y de almacenarlos en la memoria SRam en formato ASCII.

El número de caracteres a extraer se le pasa a la función a través de *W*. Los caracteres en formato UNICODE están apuntados por *FSR*. La posición destino de los datos en formato ASCII ha de ser previamente cargada en el LATCH de memoria SRam. Los caracteres en formato ASCII son almacenados en memoria en posiciones consecutivas y en la parte baja de la palabra de memoria (0 a 7), poniendo los otros 8 bits (8 a 15) a '0'.

Llamamos codificación de caracteres al procedimiento que asocia un símbolo a cada número, entendiéndose que los números, que en este contexto se llaman çódigos", deben ser enteros, no negativos, dentro de un conjunto finito (p.ej. 0 a 127, o 0 a 255).

Cada fabricante de maquinaria tenía su propio código, lo cual dificultaba en gran medida el intercambio de datos, hasta que en 1968 se llegó a una norma que hasta el día de hoy se usa y se conoce como .ASCII"(American Standard Code for Information Interchange).

El problema es que el ASCII, usa palabras de código de 8 bits por lo que puede representar un total de 128 símbolos. 128 es el número total de diferentes configuraciones que se pueden conseguir con 7 dígitos binarios (0000000, 0000001, ..., 1111111), y el octavo dígito de cada octeto ("dígito de paridad") se usa para detectar algún error de transmisión. Un cupo de 128 es suficiente para incluir mayúsculas y minúsculas del abecedario inglés, además de cifras, puntuación, y algunos çaracteres de control"(por ejemplo, uno que dice a una impresora que pase a la hoja siguiente), pero el ASCII no incluye ni los caracteres acentuados ni el comienzo de interrogación que se usa en castellano, ni tantos otros símbolos (matemáticos, letras griegas, ...) que son necesarios en muchos contextos. Por esta razón se vio que era insuficiente y se definieron varios códigos de caracteres de 8 bits.

Sin embargo, el problema de estos códigos de 8 bits es que cada uno de ellos se define para un conjunto de lenguas con escrituras semejantes y por tanto no dan una solución unificada a la codificación de todas las lenguas del mundo. Es decir, no son suficientes 8 bits para codificar todos los alfabetos y escrituras del mundo.

La solución que se ha venido en adoptar, internacionalmente, es separar la norma de codificación (elección de símbolos, y asignación de un código fijo a cada símbolo) de

la norma de transmisión, usándose ambas en conjunto. La norma de codificación más universal en la actualidad, y desde 1991, se llama **Unicode**: es una gran tabla, que en la actualidad asigna un código a cada uno de los más de cincuenta mil símbolos, los cuales abarcan todos los alfabetos europeos, ideogramas chinos, japoneses, coreanos, muchas otras formas de escritura, y más de un millar de símbolos especiales. Normas de transmisión de Unicode hay varias, pero la más relevante en Internet es UTF-8.

Unicode asigna los enteros del 0 al 127 (un total de 128) a exactamente los mismos caracteres que ASCII, el octavo dígito de la palabra de código siempre en cero. (Recordemos que ese octavo dígito, en ASCII llamado "dígito de paridad,"modernamente ya no se usa para detección de error, aunque sí se usa como parte de la codificación en ISO 8859.)

ExtraeNL_Entrada

(Apéndice A.4.2 - Línea 688) Extrae una parte del nombre del fichero o directorio de una entrada de nombre largo y lo almacena en la memoria SRam.

Para ello hace uso de la variable de programa *ESTADO_BUSQUEDA*, testando el Bit 1 (*BUSCAR_FICH*)de la misma, que nos indica si lo que estamos procesando es el nombre de un fichero o de un directorio.

- *ESTADO_BUSQUEDA[BUSCAR_FICH]* = 0: Estamos procesando el nombre de un directorio.
- *ESTADO_BUSQUEDA[BUSCAR_FICH]* = 1: Estamos procesando el nombre de un fichero.

En caso de que estemos procesando el nombre de un directorio, depositará los caracteres extraidos a partir de la dirección de SRam *NOMBRE_DIRECTORIO*, si lo que estas procesando es un nombre de fichero, usará como dirección destino de los datos *NOMBRE_FICHERO*.

Los caracteres de un nombre largo están almacenados en grupos de 13 repartidos en distintas entradas, pero las entradas están almacenadas de última a primera, por lo que necesitamos realizar algunos cálculos para colocar los caracteres en su posición correcta.

Para colocar los caracteres extraídos de cada una de las entradas usaremos el índice de nombre largo contenido en la entrada. El cálculo a realizar es el siguiente:

$$Posición\ del\ primer\ caracter\ extraído = (Indice\ de\ nombre\ largo - 1) * 13$$

Con este calculo calculamos la posición en la que ha de ir el primer carácter de cada grupo en función del índice de nombre largo de la entrada.

ExtraeNC_Entrada

(Apéndice A.4.2 - Línea 772) Extrae el nombre corto de una entrada y lo almacena en la memoria SRam.

Para ello hace uso de la variable de programa *ESTADO_BUSQUEDA*, testando el Bit 1 (*BUSCAR_FICH*)de la misma, que nos indica si lo que estamos procesando es el nombre de un fichero o de un directorio.

En caso de que estemos procesando el nombre de un directorio, depositará los caracteres extraidos a partir de la dirección de SRam *NOMBRE_DIRECTORIO*, si lo que estas procesando es un nombre de fichero, usará como dirección destino de los datos *NOMBRE_FICHERO*.

BorrarNombre

(Apéndice A.4.2 - Línea 881) Borra el nombre del último directorio o fichero leído y almacenada en la SRam.

Para ello hace uso de la variable de programa *ESTADO_BUSQUEDA*, testando el Bit 1 (*BUSCAR_FICH*)de la misma, que nos indica si lo que estamos borrando es el nombre de un fichero o de un directorio.

En caso de que queramos borrar el nombre de un directorio, eliminará los caracteres almacenados a partir de la dirección de SRam *NOMBRE_DIRECTORIO* poniendo las posiciones de memoria '0', si lo que queremos borrar es un nombre de fichero, hará lo mismo pero a partir de la dirección de memoria SRam *NOMBRE_FICHERO*.

ProcesarEntrada

(Apéndice A.4.2 - Línea 939) Analiza una entrada de 32 bytes para determinar si se trata de una entrada válida de directorio o fichero, si forma parte de una entrada de nombre largo o si indica el final de las entradas validas de directorio.

Lo primero que hace es comprobar si la entrada leída es la que sirve de marca de última entrada de directorio valida, en cuyo caso sale de la función poniendo a '1' el bit que lo indica (*ULT_ENTRADA)* en la variable de programa *ESTADO_BUSQUEDA* (bit 3).

Si no era la última entrada, lo siguiente a comprobar es si está vacía o no. En caso de que este vacía salimos de la función sin más para que se procese una nueva entrada. **(Apéndice A.4.2 - Línea 951)**

Llegados a este punto, la entrada no esta vacía. Es hora de mirar los atributos de la misma para ver si se trata de parte de un nombre largo, y, en su caso, si es la primera entrada de nombre largo. En cualquier caso extraemos los caracteres de nombre largo contenidos en la entrada, pero además, si es la primera entrada de nombre largo, primero borramos el nombre de fichero o directorio anteriormente almacenado en la memoria SRam. **(Apéndice A.4.2 - Línea 960)**

Si no se trata de parte de un nombre largo, es que es la entrada de nombre corto. Es el momento de comprobar a través de sus atributos de si se trata de una entrada de fichero o de directorio y de si lo encontrado coincide con lo que buscábamos. Si buscábamos un fichero y hemos encontrado un directorio o si buscábamos un directorio y hemos encontrado un fichero, salimos de la función para que se siga buscando. **(Apéndice A.4.2 - Línea 991)**

Si lo que buscábamos y hemos encontrado es un fichero, comprobamos que se trata de un fichero de audio comprimido (extensión 'MP3'). Si se trata de otro tipo de fichero, abandonamos la función para que se procese la siguiente entrada. **(Apéndice A.4.2 - Línea 1012)**

Si lo que buscábamos y hemos encontrado es un nombre de directorio o es un fichero 'MP3', extraemos su número de cluster de comienzo y comprobamos si disponemos o no de nombre largo asociado. En caso de no disponer de nombre largo asociado, extraemos el nombre corto de la entrada. **(Apéndice A.4.2 - Línea 1061)**

Por último indicamos a través de la variable de programa *ESTADO_BUSQUEDA* que ya disponemos de todos los datos del fichero o directorio buscado, poniendo a '1' su Bit 2 (*NC_ENCONTRADO*), y salimos de la función. **(Apéndice A.4.2 - Línea 1076)**

LeerSigCluster

(Apéndice A.4.2 - Línea 1128) Busca en el disco duro el siguiente cluster al actual.

El número de cluster ocupa 32 bits y está almacenado en la memoria del PIC en las variables de programa *Cluster_Actual0..Cluster_Actual4*.

Para buscar el cluster siguiente al actual es necesario acceder a las tablas FAT del disco duro. El número buscado se encuentra en la posición de las tablas FAT correspondiente a la indicada por el número de cluster actual.

Para agilizar la búsqueda en las tablas FAT, se mantiene en la memoria SRam una pequeña caché de las tablas FAT (a partir de la dirección *FAT_CACHEADA*) de 512 bytes, como cada entrada de la tabla FAT es de 4 bytes, almacenamos un total de 128 entradas en la caché. El bloque que se almacena se corresponde con el bloque en el que se encontraba la última entrada de la FAT buscada. En las variables de programa *FAT_Inicio0..FAT_Inicio3*, tenemos almacenado el numero de sector en el que comienzan las tablas FAT.

Lo primero que haremos sera mirar a ver si la entrada buscada está en el bloque que tenemos almacenado en la SRam, si está la leemos y si no, cargamos el bloque de las tablas FAT en la que esté la entrada buscada en la memoria SRam y la leemos.

Sabemos que cada entrada de la FAT es de 4 bytes, que el disco esta dividido en sectores de 512 BYTES, luego en un sector tenemos un bloque entero (128 entradas) de FAT. Sabemos el sector en el que comienza la FAT y por lo tanto, podemos saber el sector de FAT en el que se encuentra la entrada buscada y por lo tanto el sector del disco duro que tenemos que descargar.

Una vez obtenido el numero del siguiente cluster, tenemos que comprobar que se trata de un numero valido.

Número leído de la FAT	**Significado**
0x00000000	Libre (**FREE**)
0x00000001 0x..... 0x0FFFFEF	Entradas válidas
0x0FFFFFF0 0x..... 0x0FFFFFF6	Reservados
0x0FFFFFF7	Sector Erróneo (**BAD SECTOR**)
0x0FFFFFF8 0x..... 0x0FFFFFFF	Último Cluster del Fichero/Directorio

Cuadro 3.30: Posibles entradas de la Tabla FAT

Para ello debemos tener en cuenta que de los 32 bits obtenidos, sólo 28 conforman el numero buscado. Los 4 bits de mayor peso son de uso reservado y no deben modificarse "nunca". Únicamente son puestos a 0 en el momento de formatear la unidad, debemos pues enmascararlos siempre. De esta forma los siguientes numeros de cluster obtenidos: $0x00000000$, $0x10000000$ o $0xF0000000$, significan en realidad lo mismo, *'Cluster Libre'*, ya que debemos ignorar los 4 bits de mayor peso a la hora de leer el numero de Cluster.

De la misma forma, si el valor actual de un Cluster fuera $0x30000000$ y quisiéramos marcarlo como invalidó (*BAD CLUSTER*) escribiendo $0x0FFFFFF7$ en el, deberíamos dejar finalmente un valor en la entrada correspondiente de $0x3FFFFFF7$.

Buscar_Entrada

(Apéndice A.4.2 - Línea 1326) Esta es la función que se encarga de realizar la búsqueda de una entrada determinada.

En las variables de programa *EntBuscada_Hi:EntBuscada_Lo*, se encuentra almacenado el número de entrada buscada, mientras que en la variable *ESTADO_BUSQUEDA* y a través de su bit 1 (*BUSCAR_FICH*), indicaremos si lo que queremos buscar es un fichero o un directorio. Así si por ejemplo en *EntBuscada* hay almacenado un 2 y *ESTADO_BUSQUEDA[BUSCAR_FICH]* esta a '0', la función buscará el segundo directorio almacenado dentro del directorio raíz. Si lo que hay en *EntBuscada* es un 4 y *ESTADO_BUSQUEDA[BUSCAR_FICH]* esta a '1', la función buscará el cuarto fichero dentro del directorio actual, cuyo cluster de comienzo está almacenado en las variables de programa *Cluster_Dir0:...:Cluster_Dir3*.

Durante el proceso la función va procesando todas las entradas de un sector hasta que obtiene todos los datos referentes a un fichero/directorio usando la función *ProcesarEntrada*. Si se acaban las entradas de un sector, descarga el siguiente y si se acaban los sectores del cluster leído, obtiene el siguiente cluster a través de la función *LeerSigCluster*.

El proceso se acaba cuando encuentra el enésimo fichero o directorio buscado o cuando encuentra la marca de último cluster o de última entrada.

Una vez ha encontrado el fichero/directorio buscado abandona la función.

3.8. Interfaz con el Display LCD - *USER CODE*

Esta parte del código se corresponde con el módulo LCD y se encarga de activarlo y manejarlo adecuadamente. Forma parte de la interfaz de usuario.

Se encuentra ubicado al comienzo de la **Página 1** de la memoria de programa.

Permite controlar un display LCD de 4 líneas y 20 columnas, compatible con el controlador HD44780U de Hitachi.

El código comienza con una zona de configuración (declaración de constantes) que permite seleccionar a que líneas y/o puertos del microprocesador se conecta el display para su correcto funcionamiento. **(Apéndice A.6 - Línea 38)**

Los datos son transmitidos al Display a través de la parte baja del BUS de Datos del dispositivo ($DATA0$-$DATA7$), mientras que las líneas de control se conectan a la parte alta del BUS de Datos, ya que esta no es necesaria en la transferencia de datos PIC $\leftrightarrow$ DISPLAY.

BUS de Datos		Líneas del Display
DATA0	$\leftrightarrow$	$D0$
DATA1	$\leftrightarrow$	$D1$
DATA2	$\leftrightarrow$	$D2$
DATA3	$\leftrightarrow$	$D3$
DATA4	$\leftrightarrow$	$D4$
DATA5	$\leftrightarrow$	$D5$
DATA6	$\leftrightarrow$	$D6$
DATA7	$\leftrightarrow$	$D7$
DATA8	$\rightarrow$	RS
DATA9	$\rightarrow$	RW

Cuadro 3.31: Correspondencia entre Líneas del BUS de Datos y el Display

3.8.1. Funciones

Las funciones básicas de control permiten:

- La inicialización y configuración del display.
- El posicionamiento del cursor en una línea determinada.
- El posicionamiento del cursor en una posición determinada.
- Escribir caracteres en el display.

LCD_Conf...

(Apéndice A.6 - Línea 95) Configura adecuadamente las líneas del BUS de Datos en el microprocesador para permitir la lectura de datos del Display *LCD_ConfLec* o la escritura en él *LCD_ConfEsc*.

LCD_Conf...(C/D)

(Apéndice A.6 - Línea 125) Manipula adecuadamente las líneas de control del Display para leer o escribir comandos o datos.

- *LCD_ConfEscC*: **Escritura** de **comandos** en el Display.
- *LCD_ConfEscD*: **Escritura** de **datos** en el Display.
- *LCD_ConfLecC*: **Lectura** de **comandos** del Display.
- *LCD_ConfLecD*: **Lectura** de **datos** del Display.

DaLCD

(Apéndice A.6 - Línea 201) Manda el dato o el comando almacenado en W al Display LCD. Presupone que el BUS de Datos y el Display están correctamente configurados.

EsperarLCDlisto

(Apéndice A.6 - Línea 231) Realiza una espera activa hasta que el Display este listo para recibir comandos.

Configura el Display para proceder a la lectura de un comando mediante la invocación de *LCD_ConfLecC*.

Manda al Display la orden (RS=0 y RW=1) de leer el BIT de ocupado (BIT 7) y el contador de dirección (BIT 0 - BIT 6), analiza el bit de ocupado (BUSY) y permanece en un bucle infinito hasta que este se pone a 0, momento en el cual abandona la función.

LCD_EnviaDato

(Apéndice A.6 - Línea 259) Manda el carácter que tengamos almacenado en W al Display LCD.

Para ello primero se asegura de que el Display está listo para recibir datos invocando a la función *EsperarLCDlisto*, luego lo configura para recibir un carácter con *LCD_ConfEscD* y por último manda el carácter usando la función *DaLCD*.

InicializaLCD

(Apéndice A.6 - Línea 279) Esta función inicializa el Display LCD según el protocolo de inicialización manual. (Página - 102)

Hace uso de la función *LCD_ConfEscC* para mandar los comandos.

Las opciones de inicialización usadas son las siguientes:

- Interfaz de 8 bits.
- 4 líneas.
- Fuente para de 5x8.
- Incremento de la posición del cursor.
- Sin desplazamiento del Display.

LCD_Linea1

(Apéndice A.6 - Línea 334) Sitúa el cursor dentro de la primera línea, en la posición que se le pasa por W.

Hace uso de las funciones *EsperarLCDlisto* y *LCD_ConfEscC* para mandar adecuadamente el comando.

Sitúa el cursor en la posición resultante de sumar W (posición dentro de la línea) a $0x80$ (comienzo de la primera línea).

LCD_Linea2

(Apéndice A.6 - Línea 357) Sitúa el cursor dentro de la segunda línea, en la posición que se le pasa por W.

Hace uso de las funciones *EsperarLCDlisto* y *LCD_ConfEscC* para mandar adecuadamente el comando.

Sitúa el cursor en la posición resultante de sumar W (posición dentro de la línea) a $0xC0$ (comienzo de la segunda línea).

LCD_Linea3

(Apéndice A.6 - Línea 377) Sitúa el cursor dentro de la tercera línea, en la posición que se le pasa por W.

Hace uso de las funciones *EsperarLCDlisto* y *LCD_ConfEscC* para mandar adecuadamente el comando.

Sitúa el cursor en la posición resultante de sumar W (posición dentro de la línea) a $0x94$ (comienzo de la tercera línea).

LCD_Linea4

(Apéndice A.6 - Línea 397) Sitúa el cursor dentro de la cuarta línea, en la posición que se le pasa por W.

Hace uso de las funciones *EsperarLCDlisto* y *LCD_ConfEscC* para mandar adecuadamente el comando.

Sitúa el cursor en la posición resultante de sumar W (posición dentro de la línea) a $0xD4$ (comienzo de la cuarta línea).

BorraLinea

(Apéndice A.6 - Línea 417) Borra completamente la línea en la que se encuentra el cursor en el momento de su invocación.

Para ello se limita a llenarla de espacios en blanco.

Presupone que la posición actual del cursor dentro de la línea a borrar es 0.

LCD_LimpiaLinea1

(Apéndice A.6 - Línea 439) Borra completamente la primera línea del Display, dejando el cursor al comienzo de la misma (posición 0).

Para ello usa la función *LCD_Linea1* con W=0 para situarse al comienzo de la línea y de *BorraLinea* para borrarla. Luego vuelve a usar *LCD_Linea1* con W=0 para dejar el cursor al comienzo.

También salvaguarda y recupera el valor de W previo a la llamada a la función.

LCD_LimpiaLinea2

(Apéndice A.6 - Línea 462) Borra completamente la segunda línea del Display, dejando el cursor al comienzo de la misma (posición 0).

Para ello usa la función *LCD_Linea2* con W=0 para situarse al comienzo de la línea y de *BorraLinea* para borrarla. Luego vuelve a usar *LCD_Linea2* con W=0 para dejar el cursor al comienzo.

También salvaguarda y recupera el valor de W previo a la llamada a la función.

LCD_LimpiaLinea3

(Apéndice A.6 - Línea 483) Borra completamente la tercera línea del Display, dejando el cursor al comienzo de la misma (posición 0).

Para ello usa la función *LCD_Linea3* con W=0 para situarse a l comienzo de la línea y de *BorraLinea* para borrarla. Luego vuelve a usar *LCD_Linea3* con W=0 para dejar el cursor al comienzo.

También salvaguarda y recupera el valor de W previo a la llamada a la función.

LCD_LimpiaLinea4

(Apéndice A.6 - Línea 504) Borra completamente la cuarta línea del Display, dejando el cursor al comienzo de la misma (posición 0).

Para ello usa la función *LCD_Linea4* con W=0 para situarse al comienzo de la línea y de *BorraLinea* para borrarla. Luego vuelve a usar *LCD_Linea4* con W=0 para dejar el cursor al comienzo.

También salvaguarda y recupera el valor de W previo a la llamada a la función.

LCD_PonMensaje

(Apéndice A.6 - Línea 524) Escribe en la pantalla LCD y en la línea y a partir de la posición en la que se encuentre actualmente el cursor uno de los mensajes predeterminados y almacenados en la memoria de programa.

La selección del mensaje a escribir se realiza por medio de la variable de programa *mensajes* que tiene que ser previamente fijada.

Los mensajes se dividen en tres grupos: A, B y C, usándose los 2 bits de mayor peso de *mensajes* (6 y 7) para determinar el grupo.

BIT 7	BIT 6		Grupo seleccionado
0	0	→	GRUPO A
0	1	→	GRUPO B
1	0	→	GRUPO C

Cuadro 3.32: Selección del grupo de mensajes

Dentro de cada grupo podemos seleccionar uno de entre seis mensajes posibles. Para ello basta con poner un '1' en el Bit correspondiente al mensaje.

BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0		Mensaje seleccionado
0	0	0	0	0	1	→	Mensaje 0
0	0	0	0	1	0	→	Mensaje 1
0	0	0	1	0	0	→	Mensaje 2
0	0	1	0	0	0	→	Mensaje 3
0	1	0	0	0	0	→	Mensaje 4
1	0	0	0	0	0	→	Mensaje 5

Cuadro 3.33: Selección del mensaje dentro del grupo

Los mensajes se almacenan en memoria de programa en forma de tablas, a partir de la dirección $0x1000$, es decir en la **Página 2** de memoria de programa, por lo que esta función tiene que cambiar constantemente de página de código mediante el uso de los Bits de selección de página presentes PCLATH (Bits 3 y 4).

También hace uso de la función *LCD_EnviaDato* para mandar los caracteres extraídos de las tablas de mensajes al Display.

LCD_PonBitRate

(Apéndice A.6 - Línea 832) Escribe en el Display en la línea y a partir de la posición en la que actualmente se encuentra el cursor, el Bit Rate con el que se está reproduciendo la canción.

El Bit Rate está íntimamente ligado al esquema (LAYER I, II o III). La variable de programa *BitRate* almacena el Bit Rate obtenido del decodificador, mientras que la variable de programa *Layer* almacena el esquema empleado.

BitRate	**Layer I**	**Layer II**	**Layer III**
0000	Formato Libre	Formato Libre	Formato Libre
0001	32 Kbit/s	32 Kbit/s	32 Kbit/s
0010	64 Kbit/s	48 Kbit/s	40 Kbit/s
0011	96 Kbit/s	56 Kbit/s	48 Kbit/s
0100	128 Kbit/s	64 Kbit/s	56 Kbit/s
0101	160 Kbit/s	80 Kbit/s	64 Kbit/s
0110	192 Kbit/s	96 Kbit/s	80 Kbit/s
0111	224 Kbit/s	112 Kbit/s	96 Kbit/s
1000	256 Kbit/s	128 Kbit/s	112 Kbit/s
1001	288 Kbit/s	160 Kbit/s	128 Kbit/s
1010	320 Kbit/s	192 Kbit/s	160 Kbit/s
1011	352 Kbit/s	224 Kbit/s	192 Kbit/s
1100	384 Kbit/s	256 Kbit/s	224 Kbit/s
1101	416 Kbit/s	320 Kbit/s	256 Kbit/s
1110	448 Kbit/s	384 Kbit/s	320 Kbit/s

Cuadro 3.34: Posibles valores del Bit Rate

Los posibles valores para el Bit Rate los tenemos almacenados en tablas y separados por esquemas junto con los mensajes predeterminados. Cada posible valor del Bit Rate ocupa un total de 4 caracteres (tres caracteres ASCII del Bit Rate y un '0' de fin de cadena. Podemos usar el valor de *BitRate* como índice:

Posición a leer de la tabla = (*BitRate* * 4) + 1

Como es una multiplicación por un múltiplo de 2 la podemos realizar fácilmente con desplazamientos.

La forma de acceder a las tablas es similar a como hacíamos para extraer los mensajes predeterminados.

LCD_BitRate

(Apéndice A.6 - Línea 912) Escribe en el Display en la tercera línea y a partir de la posición '0', el mensaje predeterminado para indicar el Bit Rate junto con el esquema (LAYER I, II o III) con el que se está reproduciendo la canción.

El esquema empleado en la codificación de la canción ha sido previamente extraído del decodificador y almacenado en la variable de programa *Layer*.

Layer	**Esquema Empleado**
3	LAYER I
2	LAYER II
1	LAYER III
0	Reservado

Cuadro 3.35: Interpretación de *Layer*

LCD_SampleRate

(Apéndice A.6 - Línea 969) Escribe en el Display en la tercera línea y a partir de la posición '0', el mensaje predeterminado para indicar el Sample Rate.

LCD_PonSampleRate

(Apéndice A.6 - Línea 993) Escribe en el Display en la línea y a partir de la posición en la que actualmente se encuentra el cursor, el Sample Rate con el que se está reproduciendo la canción.

El Sample Rate ha sido previamente extraído del decodificador y almacenado en la variable de programa *SampleRate*. La interpretación de esta variable se hace de acuerdo al contenido de otra variable de program *ID* que también se extrae del decodificador y que almacena el algoritmo de codificación empleado.

ID	**Algoritmo Empleado**
3	ISO 11172-3 1.0
2	MPG 2.0 (1/2-rate)
1	MPG 2.5 (1/4-rate)
0	MPG 2.5 (1/4-rate)

Cuadro 3.36: Interpretación de *ID*

Los posibles valores para el Sample Rate están almacenados en tablas junto a los mensajes predeterminados y los valores del Bit Rate en la **Página 2** de memoria de programa.

El acceso a las tablas que almacenan el Sample Rate se hace de forma similar a como se extraían los valores del Bit Rate (manejando los Bits de selección de página).

SampleRate	**ID = 0**	**ID = 1**	**ID = 2 ó 3**
3	Reservado	Reservado	Reservado
2	16 kHz	32 kHz	8 kHz
1	24 kHz	48 kHz	12 kHz
0	22 kHz	44 kHz	11 kHz

Cuadro 3.37: Posibles valore para el Sample Rate

PonTitulo

(Apéndice A.6 - Línea 1073) Al arrancar el dispositivo se muestra en la pantalla 'MP3' en letras grandes ocupando las cuatro líneas del Display. Esta función es la encargada de pintarlo. El anagrama esta almacenado de nuevo en forma de tablas en la **página 2**, por lo que el cometido de esta función es acceder a las mismas para extraer y escribir los caracteres que conforman el anagrama.

LCD_ScrollLinea1

(Apéndice A.6 - Línea 1148) Los nombres de directorio pueden ocupar más de los 17 caracteres que hay disponibles en la pantalla para escribirlos por lo que es necesario realizar un Scroll de una sola línea cuando esto ocurre.

Los nombres de directorio se escriben en la primera línea, por lo que esta función actúa sólo sobre la primera línea.

Esta función se encarga de:

- Ajustar el offset para realizar el scroll de línea.
- Invocar a la función que realiza el scroll de línea en memoria.
- Pintar la línea con el scroll realizado.

LCD_PreparaLinea1

(Apéndice A.6 - Línea 1185) Esta es la función que se encarga de realizar el scroll de la primera línea (nombre de directorio) en memoria.

El nombre de directorio está almacenado completamente (255 caracteres como máximo) en la memoria SRam a partir de la dirección *NOMBRE_DIRECTORIO*. Para realizar el scroll extraemos 17 caracteres del nombre de directorio usando para calcular la posición de inicio la variable de programa *OffsetD*.

En caso de que no haya caracteres suficientes para ocupar las 17 posiciones (por que lleguemos al final del nombre) se ponen espacios en blanco (hasta 3), si con eso no es suficiente, se completan las 17 posiciones con el inicio del nombre.

Los 17 caracteres extraídos se almacenan en la memoria del microprocesador para ser posteriormente escritos en la pantalla.

LCD_ScrollLinea2

(Apéndice A.6 - Línea 1283) Los nombres de fichero pueden ocupar más de los 17 caracteres que hay disponibles en la pantalla para escribirlos por lo que es necesario realizar un Scroll de una sola línea cuando esto ocurre.

Los nombres de fichero se escriben en la segunda línea, por lo que esta función actúa sólo sobre la segunda línea.

Esta función se encarga de:

- Ajustar el offset para realizar el scroll de línea.
- Invocar a la función que realiza el scroll de línea en memoria.
- Pintar la línea con el scroll realizado.

LCD_PreparaLinea2

(Apéndice A.6 - Línea 1320) Esta es la función que se encarga de realizar el scroll de la segunda línea (nombre de fichero) en memoria.

El nombre de fichero está almacenado completamente (255 caracteres como máximo) en la memoria SRam a partir de la dirección *NOMBRE_FICHERO*. Para realizar el scroll extraemos 17 caracteres del nombre de fichero usando para calcular la posición de inicio la variable de programa *OffsetD*.

En caso de que no haya caracteres suficientes para ocupar las 17 posiciones (por que lleguemos al final del nombre) se ponen espacios en blanco (hasta 3), si con eso no es suficiente, se completan las 17 posiciones con el inicio del nombre.

Los 17 caracteres extraídos se almacenan en la memoria del microprocesador para ser posteriormente escritos en la pantalla.

Scroll

(Apéndice A.6 - Línea 1422) Esta función se controla con los FLAGS de dos variables de programa.

Por una parte tenemos la variable *INTerrupciones*, que contiene todos los FLAGS de eventos relacionados con el tiempo, en este caso nos interesan los Bits 0 y 1 (*FICH* y *DIRE*) que nos indican si ya ha pasado el tiempo indicado para proceder al desplazamiento de la línea de fichero (*FICH* → Línea 2) o de directorio (*DIRE* → Línea 1).

La otra variable de programa es *ESTADO_BUSQUEDA* que contiene FLAGS relacionados de alguna forma con el proceso de búsqueda de la canción en el dispositivo IDE. En este caso los FLAGS de interés son el 5 y el 6 (*SCROLL_F* y *SCROLL_D*) que se ponen a '1' si durante el proceso de búsqueda de fichero o directorio y en el momento de extraer el nombre de los mismos, se detecta que estos tienen más de los 17 caracteres que caben en la pantalla LCD.

Tanto *INTerrupciones* como *ESTADO_BUSQUEDA* son accesibles desde los cuatro bancos de memoria por lo que no es necesario conmutar de banco para acceder a ellas.

Esta función hace pues, una doble comprobación: comprueba primero el FLAG que indica si el scroll es necesario por que el nombre ocupe más de 17 caracteres y en caso afirmativo comprueba que ya haya transcurrido el tiempo necesario para realizarlo. Si se cumplen las dos condiciones, ejecuta el scroll de línea invocando a la función *LCD_ScrollLinea1* o *LCD_ScrollLinea2* según el caso. Además y sólo en caso de que el scroll se lleve a cabo, desactiva el FLAG que indica que ya ha transcurrido el tiempo entre desplazamientos.

MuestraTiempo

(Apéndice A.6 - Línea 1451) El tiempo transcurrido desde que se comenzó a reproducir la canción, se encuentra almacenado en formato ASCII en las variables de programa *DMinutos* → decenas de minutos, *UMinutos* → unidades de minutos, *DSegundos* → decenas de segundos y *USegundos* → unidades de segundo.

Esta función se limita a escribir en la pantalla en la línea 4 y a partir de la posición 15, el tiempo transcurrido en el formato 'XX:XX'.

También se encarga de poner a '0' el FLAG correspondiente al Bit 2 de *INTerrupcion*, *TIEM*, que nos indica cuando se debe actualizar el tiempo mostrado en la pantalla.

ActualizarTiempo

(Apéndice A.6 - Línea 1491) En la variable de programa *INTerrupcíon* y en su Bit 2 se encuentra el FLAG (*TIEM*) que nos indica cuando ya ha transcurrido un segundo y por lo tanto la información acerca del tiempo transcurrido tiene que ser actualizada.

El FLAG *TIEM* es puesto a '1' por la rutina de interrupción cada vez que transcurre un segundo, mientras que es puesto a '0' por la función que se encarga de mostrar el tiempo en pantalla una vez que ha completado su tarea.

Cuando esta función es invocada lo primero que hace es comprobar el FLAG *TIEM* para ver si ya hay que actualizar el tiempo mostrado. En caso de que no haya llegado el momento de la actualización, sale de la función sin más.

Para realizar la actualización se opera directamente sobre las variables *DMinutos* → decenas de minutos, *UMinutos* → unidades de minutos, *DSegundos* → decenas de segundos y *USegundos* → unidades de segundo, es decir trabajamos directamente en formato ASCII.

Comenzamos por las unidades de segundo, comprobamos si tenemos que pasar de 9 a 0 en caso negativo incrementamos las unidades de segundo y mostramos el nuevo tiempo. Si tenemos que pasar de 9 a 0, ponemos '0' en las unidades de segundo y pasamos a incrementar las decenas de segundo.

Comprobamos si tenemos que pasar de 5 a 0 en caso negativo incrementamos las decenas de segundo y mostramos el nuevo tiempo. Si tenemos que pasar de 5 a 0, ponemos '0' en las decenas de segundo y pasamos a incrementar las unidades de minuto.

Comprobamos si tenemos que pasar de 9 a 0 en caso negativo incrementamos las unidades de minuto y mostramos el nuevo tiempo. Si tenemos que pasar de 9 a 0, ponemos '0' en las unidades de minuto y pasamos a incrementar las decenas de minuto.

Comprobamos si tenemos que pasar de 5 a 0 en caso negativo incrementamos las decenas de minuto y mostramos el nuevo tiempo. Si tenemos que pasar de 5 a 0, ponemos '0' en las decenas de segundo y mostramos el nuevo tiempo.

MostrarBSR

(Apéndice A.6 - Línea 1577) En la línea 3 de pantalla y durante la reproducción, se muestran alternativamente con el paso del tiempo las informaciones referidas al Bit Rate y el Sample Rate de la canción. Esta función se encarga de llevar a cabo este cometido.

Para ello hace uso de las dos variables de programa que contienen los FLAG's *INTerrupcion* y *ESTADO_BUSQUEDA*.

De *INTerrupcion* usa el Bit 3 (*BoS*) que es puesto a 1 por la rutina de interrupción cuando ya ha pasado el tiempo preestablecido para conmutar la información que se muestra por pantalla.

De *ESTADO_BUSQUEDA* usamos el bit 7 (*BIToSAMPLE*) que cuando esta a '1' indica que hay que mostrar el Sample Rate y cuando está a '0' el Bit Rate.

La función comprueba el FLAG BoS para ver si ya hay que actualizar la información y en caso negativo abandona la función sin más.

Si hay que actualizar la información comprueba examinado el FLAG *BIToSAMPLE* que información hay que mostrar, la muestra invocando a la función correspondiente (*LCD_BitRate* o *LCD_SampleRate*)y conmuta el FLAG *BIToSAMPLE* para que la proxima vez se muestre la otra información.

Bin2Dec

(Apéndice A.6 - Línea 1674) Subrutina que se encarga de convertir un número que se encuentra en formato binario a código ASCII.

Se parte de un dato binario almacenado en "Dato4:Dato3:Dato2:Dato1:Dato0z que por lo tanto puede tener hasta un máximo de 40 bits y una vez realizada la conversión se consiguen los dígitos decimales en código ASCII correspondientes al dato de entrada en los Bytes '*Digitos:...:Digitos+12*'.

Para su correcto funcionamiento requiere que se ajuste además la variable de programa *Cifras* que contiene el número de cifras deseadas para el dato de salida.

3.9. Mensajes predeterminados del Display LCD - *USER CODE*

En la pantalla LCD se muestran una serie de mensajes predeterminados en distintos momentos de la ejecución del código. Estos mensajes son almacenados en forma de tablas a partir de la dirección $0x1000$ de memoria de programa, es decir en la **Página 2**.

Junto a los mensajes predeterminados también se almacena los posibles valores que pueden tomar el Bit Rate y el Sample Rate y el volcado en memoria del anagrama que se muestra en el momento del encendido del dispositivo.

3.9.1. Mensajes predeterminados

Tenemos tres grupos de mensajes predeterminados posibles: el A, el B y el C.

Mensajes del GRUPO A

Los mensajes que encontramos en este grupo son los correspondientes al proceso de arranque del dispositivo.

- MA0: **(Apéndice A.7 - Línea 20)** 'Reproductor de MP3.'
- MA1: **(Apéndice A.7 - Línea 26)** 'V X.X - En Pruebas'
- MA2: **(Apéndice A.7 - Línea 32)** 'IDE: DEV 0 - Listo'
- MA3: **(Apéndice A.7 - Línea 38)** 'Sistema: No es FAT32.'
- MA4: **(Apéndice A.7 - Línea 44)** 'Sistema: FAT32.'
- MA5: **(Apéndice A.7 - Línea 50)** 'Modelo: '

Mensajes del GRUPO B

Los mensajes que encontramos en este grupo son los correspondientes a la navegación por el disco duro en busca de canciones.

- MB0: **(Apéndice A.7 - Línea 56)** 'Ultima Entrada. '
- MB1: **(Apéndice A.7 - Línea 62)** 'No Encontrado.'
- MB2: **(Apéndice A.7 - Línea 68)** 'D: '
- MB3: **(Apéndice A.7 - Línea 74)** 'F: '
- MB4: **(Apéndice A.7 - Línea 80)** 'Raíz.'
- MB5: **(Apéndice A.7 - Línea 86)** 'Volumen: /35 '

Mensajes del GRUPO C

Los mensajes que encontramos en este grupo son los que se generan en el momento de la reproducción de la canción.

- MC0: **(Apéndice A.7 - Línea 92)** 'STOP '
- MC1: **(Apéndice A.7 - Línea 98)** 'PLAY '
- MC2: **(Apéndice A.7 - Línea 104)** 'Sample Rate: '
- MC3: **(Apéndice A.7 - Línea 110)** 'Layer '
- MC4: **(Apéndice A.7 - Línea 116)** ' KBit/s'
- MC5: **(Apéndice A.7 - Línea 122)** ' KHz'

3.9.2. Anagrama de inicio

Cuando arranca el dispositivo se muestra durante un lapso de tiempo un anagrama. La función que lo genera accede a esta parte del código para extraer los caracteres que ha de colocar en cada una de las cuatro líneas. **(Apéndice A.7 - Línea 135)**

3.9.3. Valores para el Bit y el Sample Rate

Todos los posibles valores que pueden tomar el Bit Rate y el Sample Rate están almacenados en tablas en esta parte del código.

Valores del Bit Rate

Los valores que puede tomar el Bit Rate se dividen en tres tablas en función del esquema empleado:

- LAYER I: **(Apéndice A.7 - Línea 221)** Valores para el esquema I
- LAYER II: **(Apéndice A.7 - Línea 241)** Valores para el esquema II
- LAYER III: **(Apéndice A.7 - Línea 261)** Valores para el esquema III

Valores del Sample Rate

Los valores que puede tomar el Sample Rate se dividen en tres tablas en función del algoritmo empleado (*ID*):

- SR_H: **(Apéndice A.7 - Línea 281)** Valores para *ID*=1
- SR_M: **(Apéndice A.7 - Línea 290)** Valores para *ID*=0
- SR_L: **(Apéndice A.7 - Línea 299)** Valores para *ID*=2 ó 3

3.10. Manejo de los Botones - *USER CODE*

Esta parte del código de usuario es la encargada de activar y manejar el módulo de botones que completa la interfaz de usuario.

Se encuentra ubicado en la **Página 0** de la memoria de programa.

3.10.1. Funciones

La función principal de este módulo es *Escanea_Botones* que es la encargada de capturar las pulsaciones, pero también se incluyen otras que aportan diversas funcionalidades.

Escanea_Botones

(Apéndice A.5 - Línea 39) Se en carga de comprobar si se ha pulsado algún botón y devuelve el código con el botón pulsado.

Para ello lo primero que hace es comprobar si se puede aceptar ya la pulsación de un botón comprobando que el FLAG correspondiente está activo. **(Apéndice A.5 - Línea 41)** Para evitar que una pulsación larga se interprete como una cadena de pulsaciones, cada vez que se recoge una pulsación se desactiva este FLAG durante un tiempo prefijado de forma que una pulsación larga se traducirá en una serie de pulsaciones separadas por un intervalo de tiempo. No se ha optado por detectar cuando se suelta el botón ya que esto ocasionaría la parada total en la ejecución del programa.

El FLAG se encuentra en la variable de programa *INTerrupcion* junto a los demás FLAGS que son ajustados en la rutina de interrupción. Cuando ha pasado el tiempo preestablecido, la rutina de interrupción pone el FLAG de *BOTON* a 1.

A continuación se procede a conectar el módulo de botones al BUS de Datos **(Apéndice A.5 - Línea 50)** y se guarda el contenido del mismo en una variable auxiliar.

En caso de haberse producido una pulsación de botón, el bit correspondiente a ese botón en el byte capturado del BUS de Datos aparecerá marcado con un '0', los botones no pulsados se representan con un '1'.

Por último el byte capturado es devuelto y la detección de pulsación se desactiva durante el tiempo prefijado. **(Apéndice A.5 - Línea 64)**

Si no se pulsó ningún botón **(Apéndice A.5 - Línea 67)** o si la detección de pulsación estaba desactivada **(Apéndice A.5 - Línea 71)** se devuelve un byte de 1's '11111111'.

QuitaRebotes

(Apéndice A.5 - Línea 67) Realiza una espera activa de un tiempo predeterminado para volver a chequear los botones y así eliminar los posibles rebotes mecánicos de los pulsadores.

EsperaTecla

(Apéndice A.5 - Línea 107) Cuando está función es invocada, se detiene automáticamente la ejecución del programa principal. El microprocesador permanece en un bucle infinito, que sólo se rompe cuando uno de los botones es pulsado.

3.11. Interfaz con el Decodificador de MPEG-1 Layer-3 - *USER CODE*

Esta parte del código se encuentra ubicada en la **página 1** de memoria de programa y esta formado por el conjunto de funciones que permiten el acceso al chip decodificador de MPEG que se va a encargar de decodificar los datos extraídos del dispositivo de almacenamiento externo IDE.

La comunicación entre el PIC y el decodificador (VS1001K) se realiza mediante dos puertos de comunicación serie SPI en el decodificador, que se conectan al puerto SPI del PIC.

Los dos puertos SPI de los que dispone el decodificador son el *SDI* y el *SCI*.

El puerto *SDI* (Serial Data Interface) sirve principalmente para transmitir los datos de audio comprimidos en formato MPEG al decodificador, pero también se puede usar para realizar TESTS de funcionamiento sobre el chip.

Para enviar datos al puerto de datos *SDI* es muy recomendable hacer uso de la línea de sincronismo de la que dispone el decodificador (*BSYNC*). Esta línea debe ser activada antes de comenzar a mandar los bits del byte y tiene que ser desactivada después de mandar el primer bit y antes de que haya sido enviado el último.

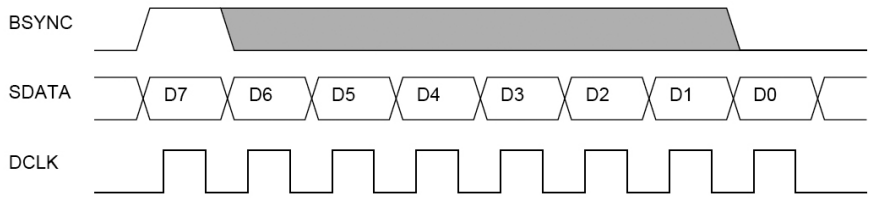


Figura 3.20: Transmisión de datos al puerto *SDI*

El puerto *SCI* (Serial Control Interface) sirve para transmitir y recibir información de control al o del decodificador. Las transferencias de datos en este puerto son siempre de 16 bits. Las principales funciones del puerto *SCI* son las siguientes:

- Controlar el modo de funcionamiento del chip.
- Cargar programas de usuario en el chip.
- Acceder a los datos de cabecera.
- Obtener información acerca del estado del chip y de la reproducción.
- Dar acceso a los datos decodificados.

Para enviar comunicarnos con el puerto *SCI* debemos activar previamente la línea de selección de CHIP ($\overline{XCS}$) poniéndola a '0'

Lo primero que hacemos es definir por que puerto se realiza la comunicación y a que líneas del mismo se conecta cada una de la líneas de control y datos del decodificador. **(Apéndice A.8 - Línea 27)**

También tenemos que definir los comandos de control **(Apéndice A.8 - Línea 52)** y las direcciones de los registros de control del decodificador **(Apéndice A.8 - Línea 63)** junto con el significado de cada uno de sus Bits cuando corresponda. **(Apéndice A.8 - Línea 88)**

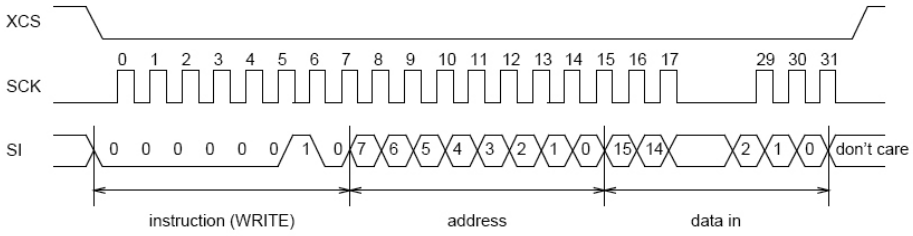


Figura 3.21: Escritura en el puerto *SCI*

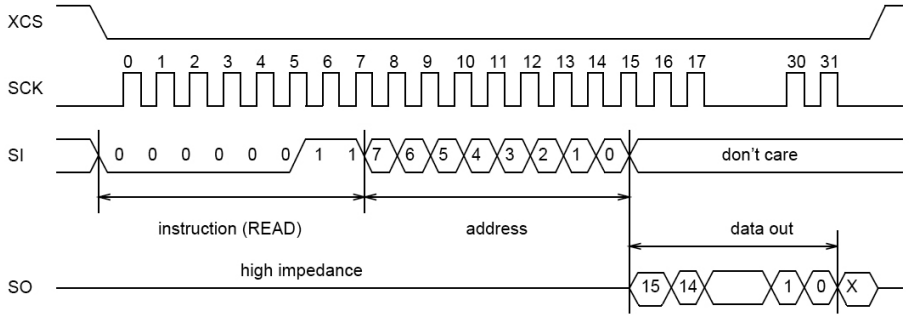


Figura 3.22: Lectura del puerto *SCI*

3.11.1. Funciones

Inicializa_SPI

(Apéndice A.8 - Línea 178) Para la interfaz de control usaremos el puerto MSSP del PIC configurado de forma que ejecute el protocolo SPI.

La velocidad del puerto SPI viene determinada por el decodificador, puede aguantar como máximo, 1/4 de la velocidad de oscilación del reloj del decodificador. Como el crystal del decodificador es de 12,888 MHz: 12,288 MHz/4 = 3,072 MHz.

El microprocesador funciona a 20MHz y puede hacer funcionar el puerto SPI a 1/4, 1/16, ... de su frecuencia maxima. Con 1/4 el puerto SPI funcionaría a 5 MHz (se pasa) y con 1/16 funcionaría a 1,25 MHz, que ya esta por debajo del limite máximo de 3 MHz.

A parte de configurar la velocidad del puerto SPI también tenemos que configurar el estado IDLE y flanco activo.

La configuración del puerto SPI se hace con los siguientes parámetros:

- Velocidad de **1,25 MHz**.
- Estado IDLE en la **parte baja** del reloj.

- Datos transmitidos en el **flanco de subida** del reloj.

Inicializa_VS1001

(Apéndice A.8 - Línea 197) Esta función se encarga de invocar las funciones de RESET y configuración del decodificador en el orden adecuado y siguiendo un protocolo preestablecido.

Primero provoca un RESET Hardware del decodificador. Configura el chip a sus valores correctos y lo resetea de nuevo, pero esta vez por Software mandando acto seguido 32 ceros al puerto de datos como indica el protocolo.

Configura_VS1001

(Apéndice A.8 - Línea 241) Esta función se encarga de configurar el decodificador para que funcione de forma correcta.

Lo primero que hacemos es indicar al decodificador el cristal oscilador que hemos puesto y si queremos activar el doblador interno de frecuencia del que dispone. Para ello debemos escribir en su registro $CLOCKF$ el valor correspondiente a dividir la frecuencia del cristal por 2000. Si además queremos activar el doblador interno de frecuencia debemos sumar al resultado obtenido $0x8000$ (32768).

A continuación pasamos a ajustar el volumen de los canales derecho e izquierdo. Esto se hace fácilmente escribiendo en el registro VOL de 16 bits el valor para cada uno de los canales. Los 8 bits más altos se corresponden con el canal izquierdo, mientras que los 8 bits más bajos son el canal derecho. Un valor de 0 ajusta el volumen al máximo. Cada unidad en que incrementemos la parte baja o la parte alta de este registro, se corresponde con una reducción de 0.5 dB en el canal modificado.

Reset_H_VS1001

(Apéndice A.8 - Línea 306) Realiza un RESET por HARDWARE del chip decodificador.

Para ello activa la línea de RESET del mismo ($\overline{XRESET}$), hace una espera de 1ms y la desactiva, esperando después 5ms para que el chip se estabilice.

Reset_S_VS1001

(Apéndice A.8 - Línea 329) Si la anterior función provocaba un RESET por Hardware del decodificador, esta origina un RESET por Software del mismo.

El RESET por Software debe hacerse siempre entre la reproducción de dos canciones para que la información del fichero que se reproduce se actualice y para que se ajuste correctamente la velocidad de reproducción en función del Bit Rate de la canción.

Si queremos asegurarnos de que no se corta el final de una canción con un Bit Rate bajo, deberemos enviar 2048 ceros al SDI antes de activar el RESET.

Para activar el RESET por Software debemos poner a '1' el Bit *SM_RESET* del registro *MODE* del decodificador. Esperamos 1 ms y lo desactivamos poniéndolo a '0'.

Después del RESET por Software debemos esperar 2 μs y testar **DREQ**. **DREQ** estará a '0' durante unos 6000 ciclos de reloj, funcionando a 24.576MHz esto supone unos 250 μs. Cuando **DREQ** se ponga a '1', debemos escribir al menos un cero en SDI antes de continuar con el funcionamiento normal. Para evitar hacer todo esto esperamos 1 ms en vez de los 2 μs, con lo que es de suponer que tras la espera **DREQ** ya estará a '1'.

Extrae_Info

(Apéndice A.8 - Línea 388) Una vez que se ha empezado a reproducir la canción en los registros del decodificador *HDAT0* y *HDAT1* aparece reflejada toda una serie de información sobre el fichero que se está reproduciendo. En concreto a nosotros nos interesa conocer los siguientes datos:

- El bit rate.
- El sample rate.
- El esquema empleado (Layer I, II o III).
- El algoritmo de codificación empleado (ID).

Esta función accede al decodificador y lee los registros de información *HDAT1* y *HDAT0*. De *HDAT1* sólo nos interesa el byte bajo, que es el que contiene *ID* y *layer*. De *HDAT0* sólo nos interesa el byte alto en el que se encuentran almacenados *bit rate* y *sample rate*.

Una vez recibidos los bytes de información, separamos los datos contenidos en ellos mediante simples rotaciones y máscaras.

BIT	Información
HDAT1[15:5]	syncword
HDAT1[4:3]	ID
HDAT1[2:1]	layer
HDAT1[0]	protect bit
HDAT0[15:12]	bit rate
HDAT0[11:10]	sample rate
HDAT0[9]	pad bit
HDAT0[8]	private bit
HDAT0[7:6]	mode
HDAT0[5:4]	extension
HDAT0[3]	copyright
HDAT0[2]	original
HDAT0[1:0]	emphasis

Cuadro 3.38: Contenido de *HDAT1:HDAT0*

SCI_Out

(Apéndice A.8 - Línea 468) Esta es la función que se encarga de mandar un dato al decodificador a su puerto de control SCI.

Para ello pone el dato a mandar en el registro de salida del puerto serie del microprocesador *SSPBUF* y espera hasta que el dato ha sido enviado completamente.

Previamente ha de ser activado el puerto de control del decodificador poniendo a '0' la línea de selección de chip $\overline{XCS}$.

SDI_Out

(Apéndice A.8 - Línea 487) Esta es la función que se encarga de mandar un dato al decodificador a su puerto de datos SDI.

Para ello activa la señal de sincronismo del decodificador ($BSYNC$) y pone el dato a mandar en el registro de salida del puerto serie del microprocesador *SSPBUF*, desactivando la señal de sincronismo una vez que se haya transmitido al menos el primer bit y siempre antes de que se hayan transmitido todos. Luego espera hasta que el dato haya sido enviado completamente.

Capítulo 4

Manual de usuario

El dispositivo es un reproductor de audio de ficheros comprimidos en el formato MPEG-1 Layer III (ISO 11172-3).

Es capaz de reproducir canciones comprimidas en una gran variedad de formatos dentro del estándar 'MP3'.

Como fuente de almacenamiento para las canciones usa un disco duro IDE de 8 gigabytes de capacidad, espacio suficiente para almacenar aproximadamente 2000 canciones con calidad CD (depende de la duración de la canción).

4.1. Características principales

- Reproduce canciones comprimidas en formato MPEG Layer III.
- Soporta las extensiones MPEG 1, 2 y 2.5, con todos sus sample rate y bit rate posibles, ya sea en mono o en estéreo.
- Es capaz de reproducir canciones en formato VBR (Variable Bit Rate).
- Usa un disco duro IDE como soporte de almacenamiento.
- El formato para el almacenamiento de las canciones es FAT 32.
- Dispone de un amplio display LCD en el que se muestra información detallada acerca de la canción que se está reproduciendo.
- El manejo es sencillo e intuitivo.

4.2. Conexión del dispositivo

4.2.1. Fuente de alimentación

Para el correcto funcionamiento del reproductor este dispone de una fuente de alimentación que ha de ser conectada a una toma de corriente de 220 voltios.

En la parte posterior de la fuente de alimentación se encuentra el interruptor de alimentación, que ha de ser activado para encender el reproductor y un compartimiento para el fusible de protección.

4.2.2. Disco duro

El disco duro IDE es el dispositivo encargado de almacenar las canciones.

Con el reproductor se suministra un disco duro IDE de 8 gigabytes de capacidad, pero puede ser sustituido por otro de mayor capacidad si así se desea. Tan sólo hay que asegurarse de que sea compatible IDE.

El disco duro se conecta a la fuente de alimentación a través del conector molex del que esta dispone.

Además el disco duro ha de ser conectado a la placa principal por medio de un cable plano de 40 hilos, para la transmisión de los datos.

4.3. Salida de audio

El reproductor dispone de una salida de audio stereo tipo jack.

A este clavija se pueden conectar directamente unos auriculares si se desea, ya que la salida de audio está preamplificada. En este caso el volumen se regulará a través de los controles de volumen de los que dispone el reproductor.

Si lo que se quiere es conectar el reproductor a un amplificador externo (equipo de música), conectaremos la salida de audio del reproductor, con una entrada de línea o auxiliar del amplificador externo. En este caso no hace falta tocar el control del volumen del reproductor, ya que al encenderlo este se ajusta automáticamente a un nivel adecuado para ser conectado a un amplificador. El control de volumen se efectuará a través del amplificador.

4.3.1. Conexión al PC

El reproductor puede ser conectado a un PC para la actualización de su firmware.

Para ello dispone de una clavija DB-9 que ha de ser conectada a uno de los puertos serie del PC: COM1, COM2, COM3...

4.4. Funcionamiento del dispositivo

Una vez conectada la alimentación del reproductor por medio del interruptor situado en la parte posterior de la fuente de alimentación, el dispositivo ejecutará el proceso de arranque del mismo, durante el cual detecta y configura el dispositivo de almacenamiento. Pasados unos segundos el reproductor puede usarse normalmente.

4.4.1. Navegación por el disco duro

Una vez ha arrancado correctamente, el reproductor mostrará en la parte superior de la pantalla la información relativa al directorio en el que se encuentra dentro del disco duro y al fichero que se está examinando.

Por medio de los botones *Directorio Siguiente* y *Directorio Anterior*, podemos cambiar durante el proceso de navegación el directorio en el que buscar las canciones. Nada más conectar el reproductor esté mostrará que se encuentra situado en el directorio *raíz* que es el primer directorio del disco duro.

Por medio de los botones *Fichero Siguiente* y *Fichero Anterior*, podemos ir seleccionando las canciones que se encuentren almacenadas dentro del directorio elegido.

4.5. Reproducción de una canción

Una vez hayamos seleccionado una canción para su reproducción por medio del los botones de navegación, podemos iniciar su reproducción pulsando el botón *Reproducir/Parar*. En cuanto iniciemos la reproducción de una canción, aparecerá en la pantalla el tiempo de reproducción y pasados unos segundos podremos ver además información acerca del Bit Rate y el Sample Rate con el que está codificada la canción. Durante la reproducción de la canción no es posible navegar por el disco duro.

Si deseamos interrumpir la reproducción de la canción, basta con volver a pulsar el botón de *Reproducir/Parar*. Una vez parada la canción, podemos volver a navegar por el disco en busca de otra.

4.6. Volumen de reproducción

Durante la reproducción de la canción es posible ajustar el volumen del audio de salida. Para ello debemos pulsar el botón de *Subir Volumen* si deseamos que la canción se oiga más alto o *Bajar Volumen* si queremos que se oiga más bajo.

4.7. Actualización del Firmware

El firmware del dispositivo puede ser actualizado mediante el uso de la conexión al PC del que el reproductor dispone.

Para actualizar el firmware basta con conectar, con la alimentación desconectada, el reproductor a uno de los puertos serie del PC y ejecutar la aplicación de actualización incluida en el CD adjunto.

Una vez ejecutada la aplicación, tan sólo debemos indicarla la ubicación del fichero que contiene la actualización para el firmware y seleccionar el puerto serie al que hallamos conectado el reproductor.

1. Apagar el reproductor.
2. Conectar el reproductor a un puerto serie del PC.
3. Iniciar, en el PC, la aplicación de actualización.
4. Indicar en el campo *File* la ubicación del fichero que contiene la actualización.
5. Seleccionar el puerto serie al que hayamos conectado el reproductor y elegir la máxima velocidad '*56000*'.
6. Pulsar sobre *Write*.
7. Conectar la alimentación del reproductor.

Una vez actualizada la versión del firmware, el reproductor continuará operando de forma normal.

Capítulo 5

Presupuesto

Para realizar el presupuesto económico para el desarrollo y la producción del reproductor, nos centraremos primero en los costes derivados del desarrollo del prototipo y luego pasaremos a evaluar los de producción.

5.1. Costes del prototipo

En el desarrollo del prototipo se originan dos tipos de costes, por una parte está el coste del material empleado y por otra tenemos el coste de la realización del diseño.

5.1.1. Coste del material empleado

En este apartado se incluyen los costes de todos los materiales usados durante la elaboración del prototipo.

Componente	Precio por unidad	Cantidad	Precio total
Circuitos Integrados			
Microcontrolador PIC 16F877A	6.44	1	6.44
74HC04	0.18	1	0.18
74HC08	0.18	1	0.18
74HC138	0.31	1	0.31
74HC193	0.69	4	2.76
74HC245	0.37	2	0.74
74HC573	0.38	1	0.38
74LVC4245	0.69	1	0.69
78T05	1.96	1	1.96
78T12	1.68	1	1.68
LM1086 - 3.3	1.47	1	1.47
MAX232	2.0	1	2.0
CY62256	2.58	2	5.16

Componente	Precio por unidad	Cantidad	Precio total
VS1001K	19.98	1	19.98
Zócalos			
DIL14	0.3	2	0.6
DIL16	0.33	6	1.98
DIL20	0.42	3	1.26
DIL28	0.58	2	1.16
DIL40	0.82	1	0.82
Adaptador SOIC24/DIL24	1.38	1	1.38
Adaptador SOIC28/DIL28	1.54	1	1.54
Tira de pines 25	0.53	2	1.06
Tira de zócalo 25	0.40	2	0.80
Resistencias			
10kΩ	0.03	10	0.3
1MΩ	0.03	1	0.03
Condensadores			
100nF/50V (Cerámico)	0.08	23	1.84
33pF/50V (Cerámico)	0.05	4	0.2
10μF/16V (Tántalo)	0.39	2	0.78
10μF/16V (Electrolítico)	0.07	2	0.14
22μF/25V (Electrolítico)	0.07	4	0.28
100μF/25V (Electrolítico)	0.1	2	0.2
220μF/35V (Electrolítico)	0.20	2	0.40
2200μF/35V (Electrolítico)	1.54	1	1.54
Osciladores			
20 MHz	0.77	1	0.77
12.288 MHz	0.77	1	0.77
Otros			
TM204AFF6 (Display LCD)	24.80	1	24.80
Disco Duro	15.72	1	15.72
1N4007 (Diodos)	0.05	4	0.2
Pulsador C.I.	0.14	9	1.26
Interruptor 220V	0.53	1	0.53
Portafusibles	0.2	1	0.2
Enchufe IEC/CEE22	0.69	1	0.69
Conector ATA-40	1.23	1	1.23
Conector DB9 Macho	2.3	1	2.3
Conector Molex hembra	0.35	1	0.35
220V - 15V/2A (Transformador)	17.9	1	17.9
10μH/2.3A (Autoinducción)	0.88	2	1.76
Placa de pruebas	15.75	1	15.75
Cables	4.5		4.5
Tornillos, tuercas y separadores	2.5		2.5
Presupuesto total del material			151.47 €

Cuadro 5.1: Coste del material

5.1.2. Coste del diseño

En este apartado calculamos el coste de la realización del diseño del reproductor, teniendo en cuenta el tiempo empleado en el desarrollo del mismo y el salario que, aproximadamente, recibiría un ingeniero por cada hora de trabajo empleada.

En total se han empleado 6 meses en el diseño y construcción del prototipo del reproductor, contando con jornadas de 8 horas y trabajando 5 días a la semana.

$$Número\ de\ horas = 6\ Meses * 20\ \frac{Días}{Mes} * 8\ \frac{Horas}{Día} = \mathbf{960\ Horas}$$

El salario aproximado que recibiría un ingeniero por este trabajo estaría entorno a los 15 € por hora trabajada. Con estos datos el coste aproximado del diseño del reproductor sería:

$$Coste\ del\ diseño = Número\ de\ horas * \frac{Coste}{Hora} = 960 * 15 = \mathbf{14400\ €}$$

5.2. Coste de producción

Una vez evaluado el coste del prototipo vamos a calcular ahora el coste de producción, analizando factores como el número de unidades producidas que va a determinar el coste del material empleado y el coste de la mano de obra necesaria para realizar el montaje de las unidades.

5.2.1. Coste del material

El número de unidades producidas va a ser un factor determinante a la hora de calcular el coste del material necesario para su producción, ya que es normal la aplicación de importantes descuentos en el precio de los componentes en función del número de estos que se adquieran.

Haremos el estudio para tres cantidades de unidades producidas y aplicaremos un descuento para cada una de ellas.

Coste del material del prototipo			151.47 €
Unidades producidas	100	1000	10000
Descuento aplicado	10 %	20 %	30 %
Coste del material	13632.3 €	121176 €	1060290 €

Cuadro 5.2: Coste del material en función del número de unidades

5.2.2. Coste de la mano de obra

Ahora tenemos que evaluar el coste que tendría producir las unidades. Para ello debemos tener en cuenta dos factores: por una parte tenemos que tener evaluar el tiempo que se tardaría en montar una unidad del reproductor pero además debemos contar con el salario que percibiría cada uno de los operarios que se van a encargar de su montaje.

Debido a la cantidad de componentes que tiene el reproductor el tiempo de montaje de una unidad del reproductor es de aproximadamente 3 horas.

Para realizar el montaje de las unidades no hace falta que las personas que se van a encargar de ello tengan una cualificación muy alta, nos basta con simples operarios de montaje. El salario estipulado para este tipo de puesto es de aproximadamente 8 € la hora.

Los costes derivados de la mano de obra serían los siguientes:

$$\textit{Coste de la mano de obra por unidad} = 3 * 8 = \mathbf{24\ €} \text{ por unidad producida.}$$

Teniendo en cuenta el número de unidades producidas, el coste de la mano de obra sería:

Unidades producidas	100	1000	10000
Coste de la mano de obra	2400 €	24000 €	240000 €

Cuadro 5.3: Coste de la mano de obra en función del número de unidades

5.3. Precio de salida al mercado

Para calcular el precio de salida al mercado, debemos tener en cuenta todos los costes: el coste del diseño, el coste del material y el coste de la mano de obra. Además, como también queremos obtener beneficios, tendremos que incluir estos como un coste añadido al precio final de salida al mercado.

Unidades producidas	**100**	**1000**	**10000**
Coste del diseño del prototipo	14400 €		
Coste de la mano de obra	2400 €	24000 €	240000 €
Coste del material	13632.3 €	121176 €	1060290 €
Coste por unidad	304.3 €	159.6 €	131.5 €
Porcentaje de beneficios	18 %		
Coste final por unidad	**359.1 €**	**188.3 €**	**155.2 €**

Cuadro 5.4: Cálculo del precio de salida al mercado

Capítulo 6

Conclusiones y otras cosas

6.1. Conclusiones

Como proyecto de fin de carrera quise elegir un proyecto en el que se abordara la electrónica desde un punto de vista práctico y en el que se abordaran diversos aspectos, y el diseño de un dispositivo basado en el uso de un microcontrolador cumplía todas mis expectativas.

Dejando a un lado la utilidad práctica del reproductor, en él se han incluido una serie de características, que si bien no son estrictamente necesarias para reproducir canciones, si que lo hacen mucho más versátil y atractivo de cara al diseño del mismo.

Hoy en día se pueden encontrar en las tiendas multitud de reproductores 'MP3' de dimensiones muy pequeñas y ello se debe ha que prescinden de muchas de las características de este reproductor. Son reproductores que no disponen de disco duro, como medio de almacenamiento usan memorias estáticas de acceso directo, por lo que no es necesario incluir la circuitería de control requerida para gobernar el dispositivo IDE, pero además al usar directamente una memoria para almacenar canciones, el tiempo de latencia se reduce considerablemente y no es necesario incluir memoria adicional. Tampoco disponen de conexión serie y la fuente de alimentación es una simple batería.

Si hubiésemos optado por realizar un dispositivo de este tipo, el objetivo final (reproducir canciones 'MP3') se habría cumplido de la misma forma pero en el camino habríamos perdido todo lo interesante para acabar haciendo algo 'como los demás'.

La inclusión de todos estos aspectos prescindibles (disco duro, memoria adicional SRAM, bootloader,...) hacen que el reproductor se convierta en una plataforma versátil para el desarrollo de diversos dispositivos en el que la capacidad de reproducción 'MP3' pasa a ser una 'característica adicional'.

En el desarrollo del reproductor se han tenido que abordar diversos temas, muchos de los cuales han supuesto un proceso de investigación previo y que por lo tanto se han visto reflejados en un enriquecimiento personal tanto en conocimientos adquiridos como en la forma de lidiar con los problemas que van surgiendo.

Algunos de los más importantes son:

- Diseño de la fuente de alimentación
- Incorporación de un interfaz serie al PC
- Elaboración del bootloader
- Manejo de dispositivos IDE
- Implementación del sistema de ficheros FAT32
- Inclusión de un interfaz con el usuario (pantalla de visualización + botones)
- Añadir una memoria SRam
- Emplear un decodificador de 'MP3' dedicado

6.2. Posibles mejoras

Con un dispositivo de este tipo la relación de mejoras que pueden efectuarse en él son casi ilimitadas, sin embargo hay algunas que darían mucho 'juego' al reproductor, las más destacadas son:

- Incluir una unidad de CD-ROM o DVD-ROM.
- Añadir una conexión USB para permitir la grabación de canciones en el disco duro sin tener que extraerlo del reproductor.
- Dotar de mayor funcionalidad al reproductor.

6.2.1. Incluir una unidad de CD-ROM o DVD-ROM

Debido al diseño del reproductor, esta mejora es 'muy fácil' de llevar a cabo. El reproductor está dotado de interfaz IDE-ATA, por lo que si incluimos un lector de CD's o DVD's con este formato no tendremos ningún problema en manejarlo correctamente.

Sin embargo el almacenamiento de los datos en el dispositivo (CD o DVD) ya no se hace según el sistema de ficheros FAT32, para almacenar datos en CD's o DVD's se usa el estándar ISO-9660, por lo que habría que incluir una librería con el código de soporte de este estándar y modificar ligeramente el código del programa principal para dar al usuario la capacidad de seleccionar la fuente de los datos: CD/DVD o HD.

Se podría ir un poco más allá y posibilitar la copia de datos del CD/DVD al HD desde el propio reproductor, lo que serviría para que el usuario fuera almacenando en el HD las canciones que más le gustan. Para haver esto, además de todo lo anterior, habría que incluir en la librería de FAT32, funciones para formatear los datos leídos del CD/DVD y modificar las tablas FAT. Las funciones de escritura en el dispositivo IDE ya están soportadas.

6.2.2. Añadir una conexión USB

La principal 'pega' del reproductor tal y como está es que es necesario extraer el disco duro para grabarlo, aunque una vez grabado y debido a la gran capacidad que tiene, es de esperar que no sea necesario volver a grabarlo en mucho tiempo.

Para dar al reproductor la capacidad de recibir las canciones directamente desde un PC, lo más cómodo es usar un interfaz USB. El reproductor dispone de un interfaz serie, pero la 'lentitud' de este interfaz hace que sea incluso más rápido extraer el disco duro, grabarlo y volver a ponerlo, que usar el puerto serie para transferir las canciones. (siempre y cuando el volumen de estas sea considerable)

A la hora de incluir soporte USB tenemos tres opciones:

- Solución software
- Solución hardware
- Solución mixta

Solución software

Sin lugar a dudas está es la opción más barata. Tan sólo es necesario incluir un conector USB y posiblemente un transceptor de BUS para aislarlo como elementos hardware, sin embargo el software necesario para hacer funcionar el interfaz sería considerable.

- Es necesario implementar el protocolo USB.
- Hay que incluir en la librería de FAT32, funciones para formatear los datos obtenidos por el puerto USB y modificar las tablas FAT
- Hay que programar un driver para el SO de forma que el dispositivo sea reconocido como un sistema de almacenamiento masivo.

Si tenemos en cuenta que el espacio que queda libre para añadir código no es muy grande, habría que hilar muy fino para llevar esta solución a cabo.

Solución hardware

Lo más cómodo y a la vez lo más caro es optar por una solución 100 % hardware.

En el mercado hay multitud de chips que ofrecen la posibilidad de implementar un 'USB-ATA bridge'. Se trata de un integrado que se conecta directamente al puerto ATA del disco duro por una parte, al conector USB por otra, que tiene una línea que cuando se activa deja en alta impedancia la conexión con el puerto ATA y que además es detectada automáticamente por el SO del PC como un dispositivo de almacenamiento masivo, de forma que no es necesario programar ningún driver.

Con una solución de este tipo no tenemos que añadir 'casi' ni una línea. Tan sólo debemos incluir el control sobre la línea de activación de la conexión del 'chip puente' con el puerto ATA del disco duro.

Un firme candidato para este cometido sería el chip de *Cypress Semiconductor Corporation* **CY7C68300C** que además está disponible en formato SSOP-56.

Solución mixta

Una de las partes más tediosas de la solución software es la de programar el protocolo USB. Para evitarlo, en esta solución mixta usamos un integrado de *Future Technology Devices International Ltd. (FTDI)* el **FT8U245BM** que nos proporciona el interfaz USB.

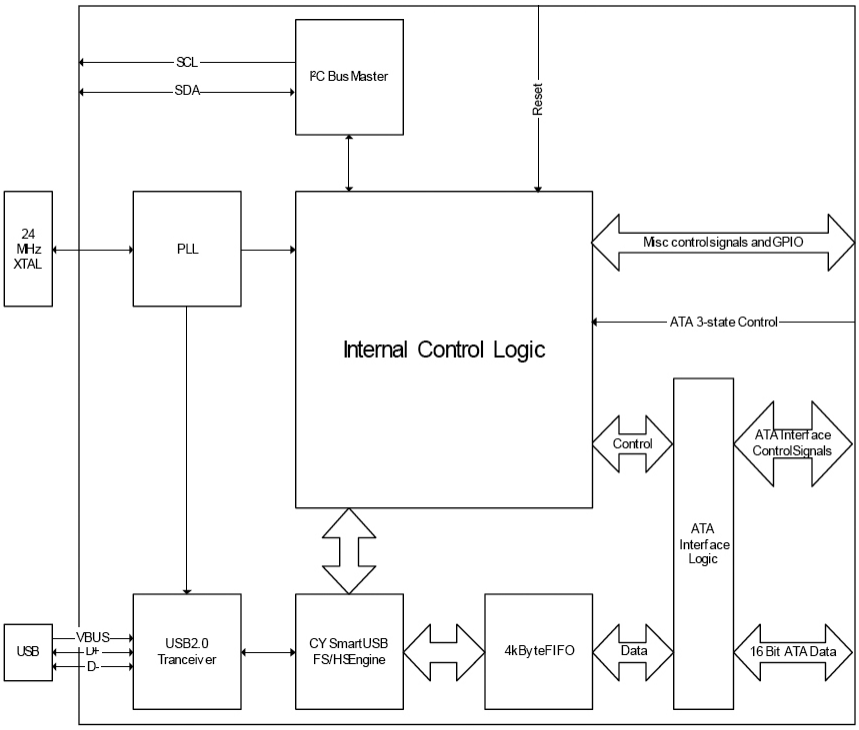


Figura 6.1: Diagrama de bloques del CY7C68300C

Podemos además usar la versión del mismo que viene en formato de mini-módulo con la conexión USB incorporada y listo para ser insertado en un zócalo de 24 pines.

Este módulo ofrece la implementación directa del interfaz USB completo, incluyendo el protocolo de pila y se controla de una forma muy sencilla a través del BUS de datos (8 líneas) y de una línea de lectura y otra de escritura. Además el fabricante posee drivers de libre distribución para los principales sistemas operativos.

La parte de software que sería necesario añadir hace referencia a la necesidad de formatear los datos provenientes del módulo USB para que puedan ser correctamente almacenados en el disco duro en el sistema de ficheros FAT32.

6.2.3. Dotar de mayor funcionalidad al reproductor

Aquí las posibilidades son infinitas. Podemos añadir al reproductor diversas funcionalidades extra, como por ejemplo:

- Control de graves y agudos
- Introducir esquemas sonoros (JAZZ, POP, ROCK...)

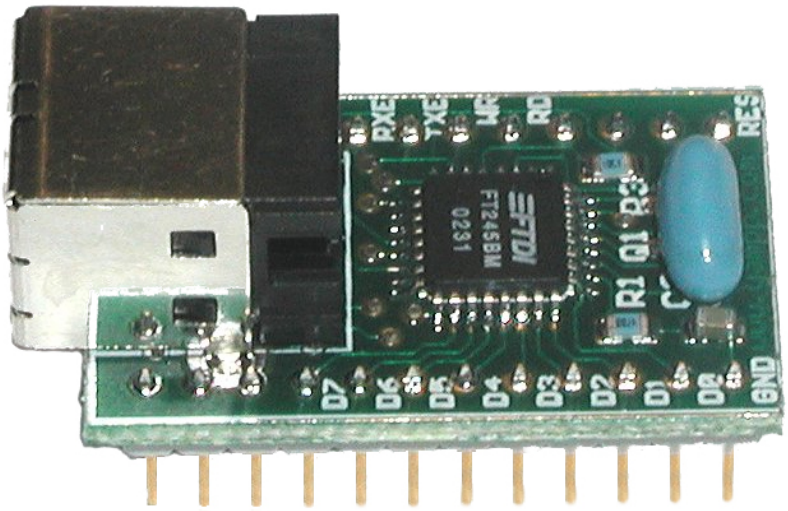


Figura 6.2: Mini-módulo USB

- Añadir funcionalidades de reproducción como: RANDOM, SKIP...
- Permitir al usuario editar los nombres de las canciones almacenadas
- Permitir al usuario borrar canciones del disco duro
- ...

6.3. Algunas Fotos

Por último se muestran algunas fotos de diversas partes del proyecto.

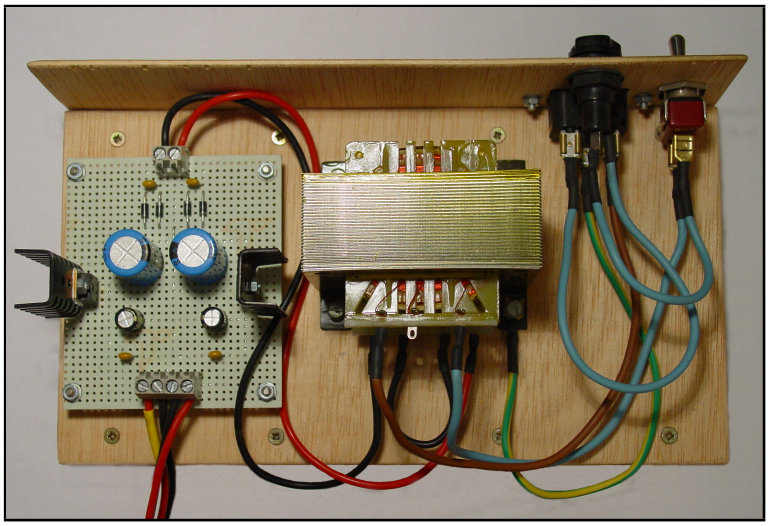


Figura 6.3: Fuente de alimentación

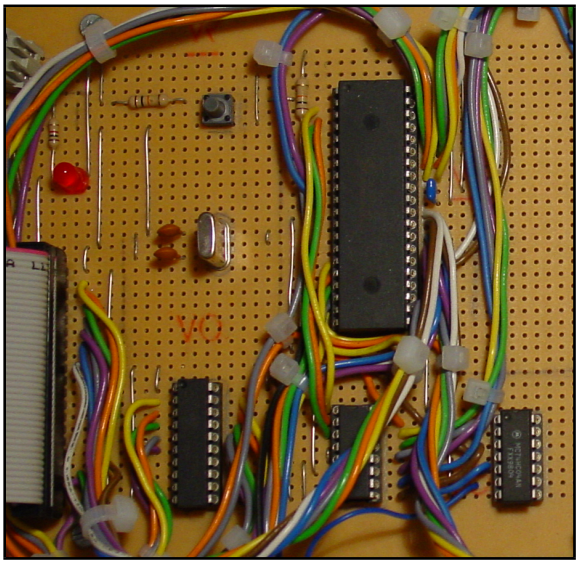


Figura 6.4: Circuiteria de control e interfaz IDE

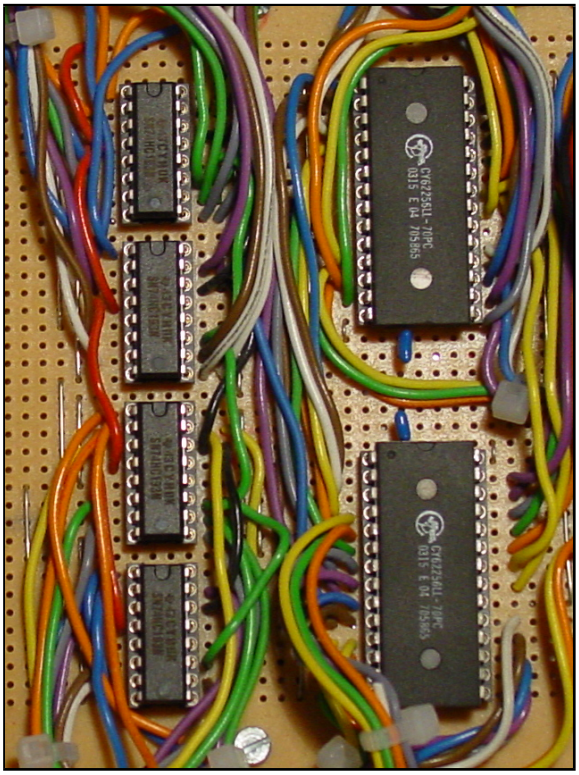


Figura 6.5: Memoria SRam

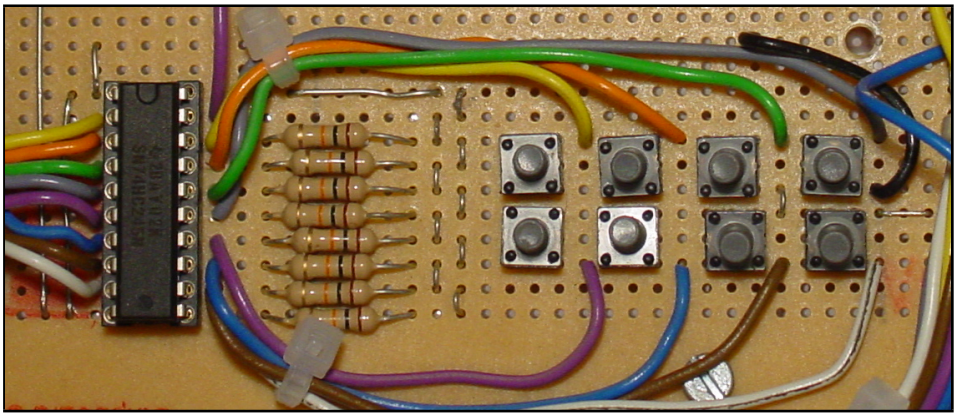


Figura 6.6: Módulo de botones

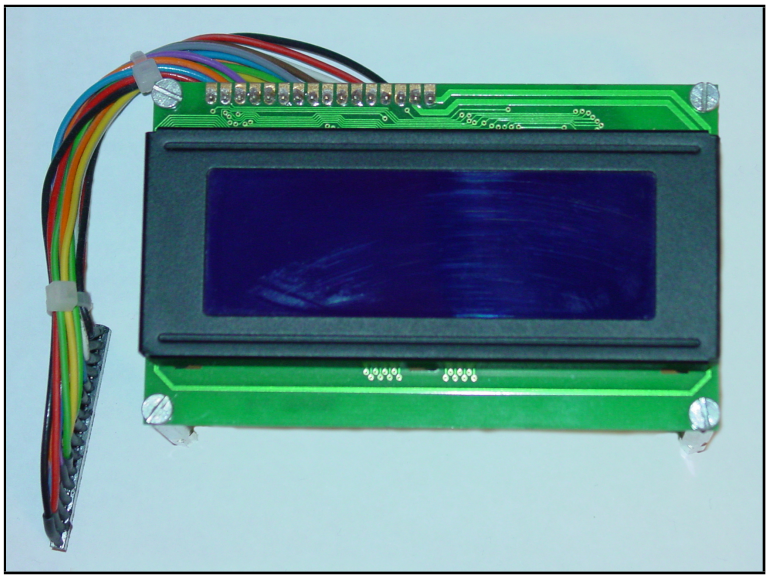


Figura 6.7: Display LCD

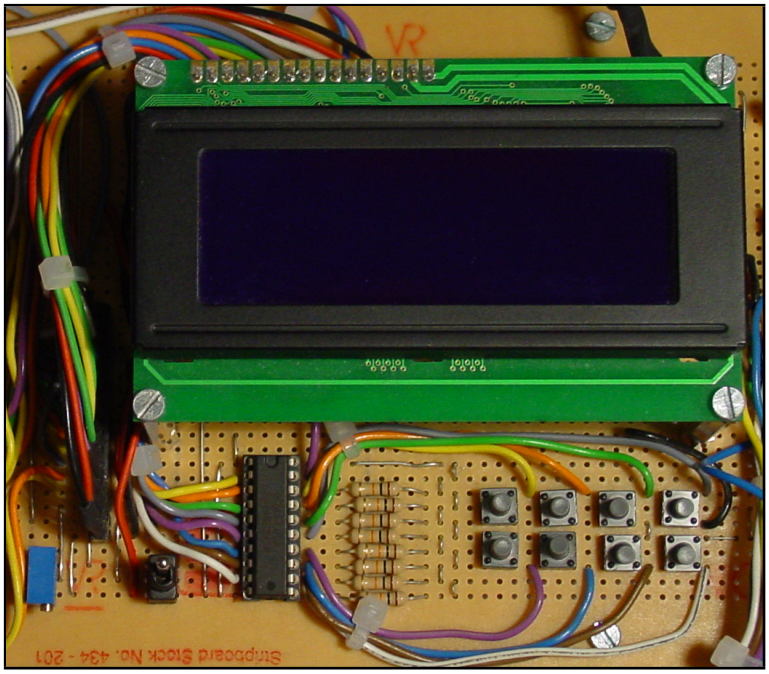


Figura 6.8: Interfaz con el usuario

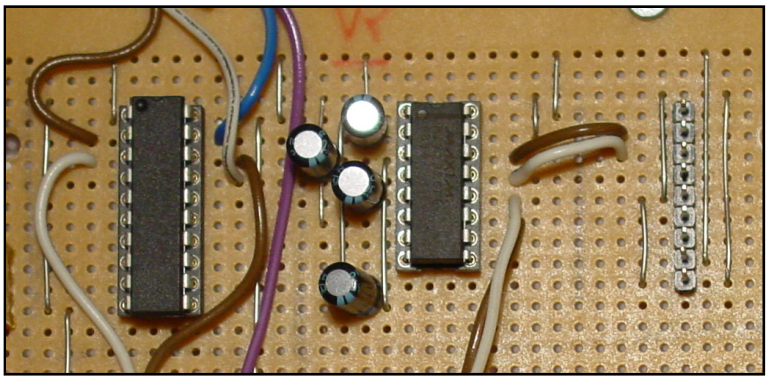


Figura 6.9: Interfaz serie

Figura 6.10: Disco duro

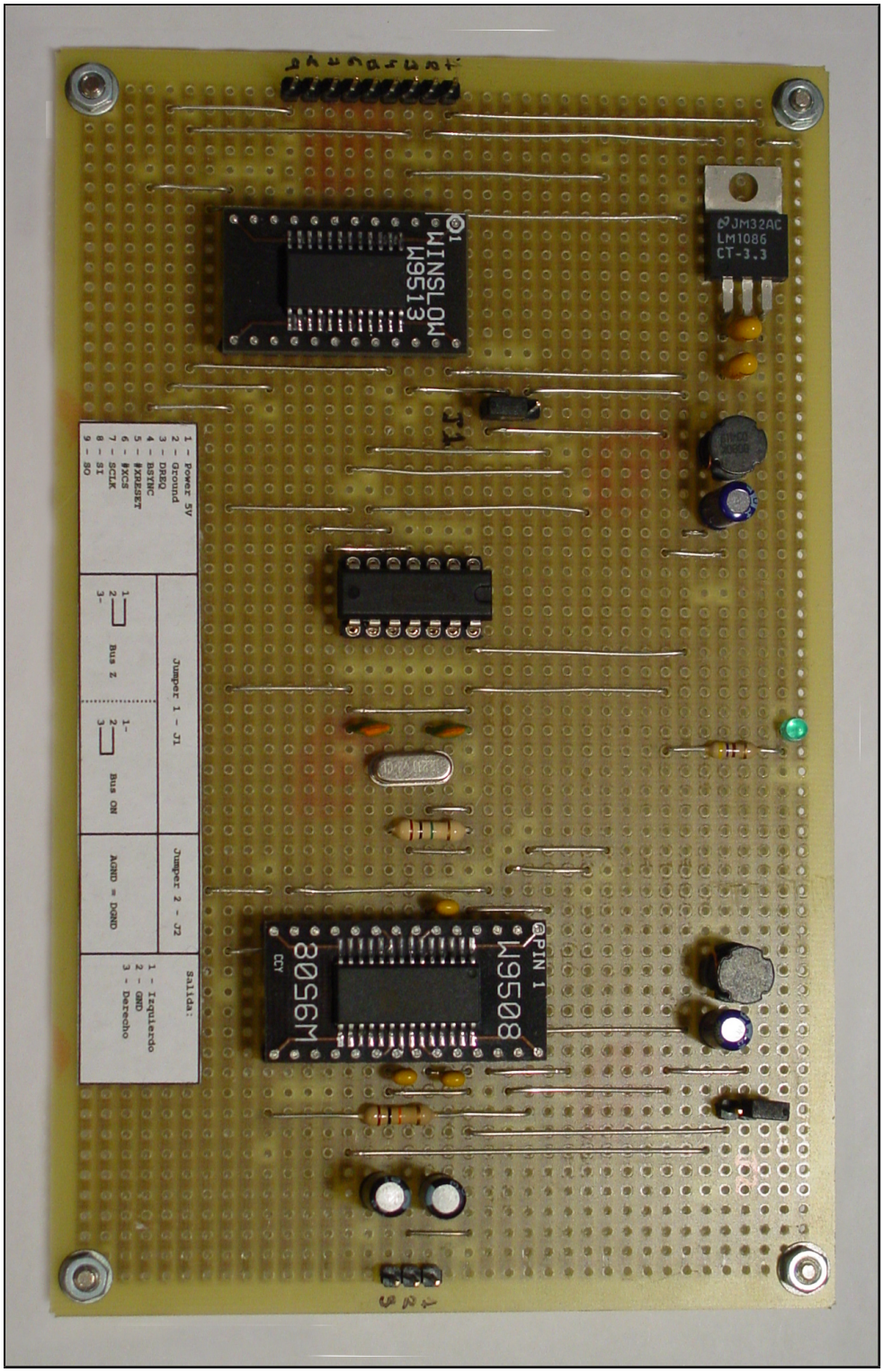


Figura 6.11: Módulo decodificador MPEG-1 Layer III

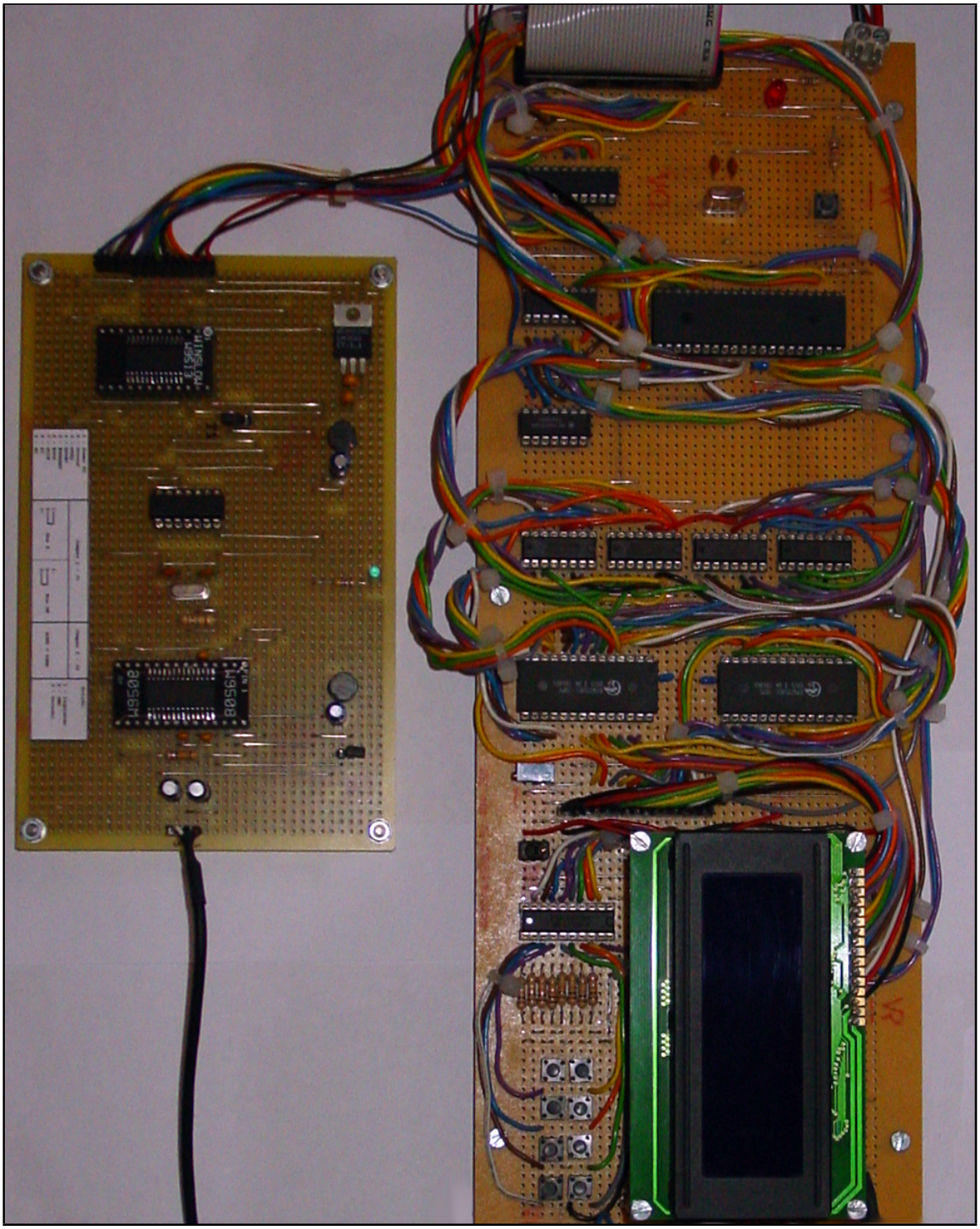


Figura 6.12: Circuito completo

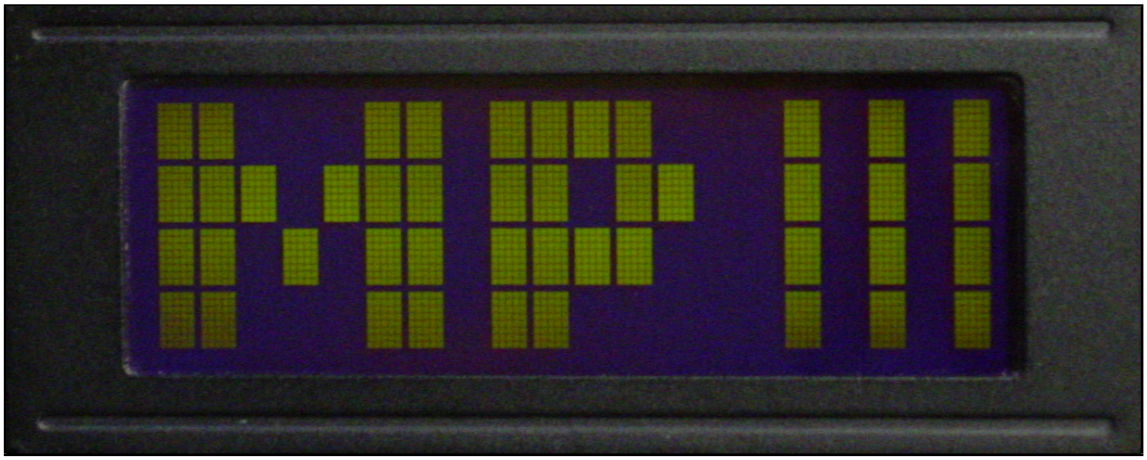


Figura 6.13: Encendido del reproductor

Figura 6.14: Reconocimiento de dispositivos

Figura 6.15: Scroll funcionando

Figura 6.16: Sample rate

Figura 6.17: Bit rate

Figura 6.18: Modificación del volumen

Apéndice A

Codigo Fuente

A.1. Bootloader

Esta parte de código es la encargada de gestionar la recepción de nuevo código desde el PC. Para que funcione correctamente, debe ser compilada de forma independiente y cargada en el microcontrolador mediante un grabador externo (tipo JDM).

El Bootloader se encuentra localizado en la **Página 3** de memoria de programa, concretamente en la parte mas alta de esta, y ocupa solamente 384 palabras de código de las 8192 que puede almacenar el microcontrolador.

```
1        LIST        P=16F877A
2        INCLUDE     "P16F877A.inc"
3        ERRORLEVEL  -302
4        __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _HS_OSC & _PWRTE_ON & _LVP_OFF

6   ;*****************************************************************************
7   ;*****************************************************************************
8   ;**                                                                         **
9   ;**                          Datos de Configuracion                         **
10  ;**                                                                         **
11  ;*****************************************************************************
12  ;*****************************************************************************

14  #define  FOSC       d'20000000'  ; Frecuencia del Oscilador, max. 20 MHz
15  #define  BAUD       d'56000'     ; Baud rate [bit/sec]
16  #define  BAUD_ERROR d'4'          ; Maximo Error en Baud rate [%]
17  #define  TIMEOUT    d'2'          ; Tiempo de chequeo [0.1s], max. 25 sec

19  ;*****************************************************************************
20  ;*                                                                           *
21  ;*               Calculamos y verificamos los Baudios de conexion            *
22  ;*                                                                           *
23  ;*****************************************************************************

25     IF ((FOSC/(d'16' * BAUD))-1) < d'256'

27        #define  DIVIDER     (FOSC/(d'16'*BAUD))-1
28        #define  HIGH_SPEED  1

30     ELSE

32        #define  DIVIDER     (FOSC/(d'64'*BAUD))-1
33        #define  HIGH_SPEED  0

35     ENDIF
```

```
BAUD_REAL   EQU FOSC/((d'64'-(HIGH_SPEED*d'48'))*(DIVIDER+1))

   IF BAUD_REAL > BAUD
      IF (((BAUD_REAL - BAUD)*d'100')/BAUD) > BAUD_ERROR

         ERROR "Baud␣Rate␣Erroneo"

      ENDIF
   ELSE
      IF (((BAUD - BAUD_REAL)*d'100')/BAUD) > BAUD_ERROR

         ERROR "Baud␣Rate␣Erroneo"

      ENDIF
   ENDIF

; ***************************************************************************
; *                                                                         *
; *                      Calculamos el valor del Temporizador               *
; *                                                                         *
; ***************************************************************************

   IF FOSC > d'10240000'

      #define  T1PS  8          ; Preescala 1:8
      #define  T1SU  0x31       ; '00110001' -> Preescala=1:8 TMR1ON=1

   ELSE
      IF FOSC > d'5120000'

         #define  T1PS  4       ; Preescala 1:4
         #define  T1SU  0x21    ; '00100001' -> Preescala=1:4 TMR1ON=1

      ELSE
         IF FOSC > d'2560000'

            #define  T1PS  2    ; Preescala 1:2
            #define  T1SU  0x11 ; '00010001' -> Preescala=1:2 TMR1ON=1

         ELSE

            #define  T1PS  1    ; Preescala 1
            #define  T1SU  0x01 ; '000000001 -> Preescala=1   TMR1ON=1

         ENDIF
      ENDIF
   ENDIF

TIMER       EQU (d'65538'-(FOSC/(d'10'*4*T1PS)))

;****************************************************************************
;****************************************************************************
;**                                                                        **
;**                  Localizacion del Codigo de Arranque                   **
;**                                                                        **
;****************************************************************************
;****************************************************************************

#define     ProgHI      0x1FFF   ; Direccion maxima de memoria
#define     LoaderSize  0x190    ; Tamaño del BootLoader

#define     LoaderTop   ProgHI   ; Direccion maxima del Bootloader

#define     LoaderStart (LoaderTop)-LoaderSize+1   ; Direccion de comienzo del
                                                   ; Bootloader

#define     NumRetries  1        ; Numero de reintentos de escritura


#define     WRITE       0xE3     ; Protocolo de comunicaciones
#define     WR_OK       0xE4
#define     WR_BAD      0xE5
```

```
110  #define      DATA_OK      0xE7
111  #define      DATA_BAD     0xE8

113  #define      IDENT        0xEA
114  #define      IDACK        0xEB

116  #define      DONE         0xED

118  ;*****************************************************************************
119  ;*****************************************************************************
120  ;**                                                                         **
121  ;**                          Variables de Programa                          **
122  ;**                                                                         **
123  ;*****************************************************************************
124  ;*****************************************************************************

126  buff         EQU 0x20
127  amount       EQU 0x71
128  chk1         EQU 0x72
129  chk2         EQU 0x73
130  retry        EQU 0x74
131  address      EQU 0x75
132  tmpaddr      EQU 0x77
133  temp         EQU 0x79
134  time         EQU 0x7A
135  count        EQU 0x7B

137  help         EQU 0x7C
138  lcount       EQU 0x7D

140  ;*****************************************************************************
141  ;*****************************************************************************
142  ;**                                                                         **
143  ;**                            Codigo de Programa                           **
144  ;**                                                                         **
145  ;*****************************************************************************
146  ;*****************************************************************************

148     org       0x0000
149     nop
150     pagesel   Main
151     goto      Main

153  ;*****************************************************************************
154  ;*                                                                           *
155  ;*                              Vector de RESET                              *
156  ;*                                                                           *
157  ;*****************************************************************************

159    org        LoaderStart          ; Pagina 3 de codigo comienzo del Bootloader

161  ; Lo primero que hacemos es detectar las incursiones no deliberadas en el codigo
162  ; del Bootloader, bloqueandolas con un bucle infinito

164  TrapError
165     pagesel   TrapError
166     goto      TrapError

168  UserStart

170     clrf      PCLATH               ; Esta instruccion nunca es sobreescrita

172                                    ; Las siguientes 4 instrucciones son
173                                    ; sobrescritas por el programa de usuario.

175     pagesel   UserStart            ; La primera vez que se ejecuta el Bootloader,
176     goto      UserStart            ; como no hay programa de usuario que ejecutar
177     nop                            ; porque no se ha cargado ninguno, nos
178                                    ; mantenemos en un bucle infinito

180  ;*****************************************************************************
181  ;*                                                                           *
182  ;*                                 Principal                                 *
```

```
183 ;*                                                                          *
184 ;***************************************************************************

186 Main
187    btfss     STATUS,NOT_TO
188    goto      UserStart


192 Start

194 ;***************************************************************************
195 ;*                                                                         *
196 ;*             Configuramos el circuito para usar el puerto serie          *
197 ;*                                                                         *
198 ;***************************************************************************

200    bsf       STATUS,RP0            ; Banco 1
201    movlw     b'00000110'           ; PORTA Todo digital
202    movwf     ADCON1

204    clrf      TRISA                 ; A0-A2  OUT -> CODIGO del demultiplexor
205    bcf       STATUS,RP0
206    movlw     b'00001101'
207    movwf     PORTA

209 ;***************************************************************************
210 ;*                                                                         *
211 ;*                      Configuramos el puerto serie                       *
212 ;*                                                                         *
213 ;* USART SYNC=0 SPEN=1 CREN=1 SREN=0 TX9=0 RX9=0 TXEN=1                   *
214 ;*                                                                         *
215 ;***************************************************************************

217    movlw     b'10010000'           ; SPEN = 1, CREN = 1
218    movwf     RCSTA
219    bsf       STATUS,RP0            ; Banco 1

221    IF HIGH_SPEED == 1

223       bsf       TXSTA,BRGH

225    ELSE

227       bcf       TXSTA,BRGH

229    ENDIF

231    bsf       TXSTA,TXEN
232    movlw     DIVIDER
233    movwf     SPBRG

235 ;***************************************************************************
236 ;*                                                                         *
237 ;*                      Configuramos el Timer 1                            *
238 ;*                                                                         *
239 ;* Debemos esperar durante el tiempo designado en TIMEOUT para ver si se  *
240 ;* la comunicacion desde el PC.                                            *
241 ;*                                                                         *
242 ;***************************************************************************

244    bcf       STATUS,RP0
245    movlw     TIMEOUT+1
246    movwf     time
247    movlw     T1SU                  ; Ajustamos el valor de T1CON
248    movwf     T1CON
249    bsf       PIR1,TMR1IF
250    call      getbyte               ; Esperamos la transmision del  PC
251    xorlw     IDENT                 ; Comprobamos si es del Bootloader
252    btfss     STATUS,Z
253    goto      user_restore          ; No es, ejecutamos el programa de usuario
254    clrf      time                  ; Ya no necesitamos TIMEOUT
255    goto      inst_ident            ; Bootloader identificado, respondemos con un
```

```
                                    ; IDACK
; Recibimos los datos

; Programando
; Obtenemos un byte por el puerto serie y lo identificamos. En funcion de su
; valor, decidimos lo que hacer.

receive
   call     getbyte
   movwf    temp
   xorlw    WRITE                   ; Comprobamos si era la instruccion WRITE
   btfsc    STATUS,Z
   goto     inst_write
   movf     temp,W
   xorlw    IDENT                   ; Comprobamos si era la instruccion IDENT
   btfsc    STATUS,Z
   goto     inst_ident
   movf     temp,W
   xorlw    DONE                    ; Comprobamos si era la instruccion DONE
   btfss    STATUS,Z
   goto     receive

; Hemos acabado la programacion

;****************************** INSTRUCCION DONE ******************************

inst_done
   movlw    WR_OK                   ; Enviamos WR_OK
   call     putbyte
   movlw    TIMEOUT+1
   movwf    time
   call     getbyte

user_restore
   clrf     T1CON                   ; Desconectamos el TIMER 1
   clrf     RCSTA
   bsf      STATUS,RP0
   clrf     TXSTA                   ; Resetamos el puerto serie
   bcf      STATUS,RP0
   clrf     PIR1

;*****************************************************************************
;*                                                                           *
;*              Desactivamos el uso del puerto serie en la placa             *
;*                                                                           *
;*****************************************************************************

   movlw    b'00000111'             ; Desactivamos el MUX
   movwf    PORTA

;*****************************************************************************

   goto     UserStart               ; Ejecutamos el programa de usuario

;*****************************************************************************

;****************************** INSTRUCCION IDENT ******************************

inst_ident
   movlw    IDACK                   ; Enviamos IDACK
   goto     send_byte

;****************************** INSTRUCCION WRITE ******************************

inst_write
   call     getbyte
   movwf    address+1               ; Byte alto de la direccion
   call     getbyte
   movwf    address                 ; Byte bajo de la direccion
   call     getbyte
   movwf    amount                  ; Numero de bytes -> amount -> count
   movwf    count
   call     getbyte                 ; checksum -> chk2
```

```
329     movwf     chk2
330     clrf      chk1                  ; chk1 = 0

332  ;*** *** *** *** *** *** *** *** ADAPTACION 1.2 *** *** *** *** *** *** *** ***

334     movlw     0x21                  ; Si (0x2100 <= tmpaddr <= 0x21FF) ..........
335     banksel   address
336     subwf     address+1,W
337     btfsc     STATUS,Z
338     goto      adapt12_1end          ; Fin de la adaptacion

340     movf      address,W
341     andlw     0x03
342     movwf     help
343     movwf     lcount
344     bcf       STATUS,C
345     rlf       help,F
346     movlw     buff
347     addwf     help,W
348     goto      adapt12_2end
349  adapt12_1end                       ; Fin de la adaptacion
350     movlw     buff

352  adapt12_2end                       ; Fin de la adaptacion
353     movwf     FSR

355  ; *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** **

357  ; FSR pointer = buff

359  ; Recibimos el dato

361  receive_data
362     call      getbyte               ; Recibimos el siguiente byte -> buff[FSR]
363     movwf     INDF
364     addwf     chk1,F                ; chk1 := chk1 + buff[FSR]
365     incf      FSR,F                 ; FSR++
366     decfsz    count,F
367     goto      receive_data          ; Repetimos hasta que count sea 0
368  checksum
369     movf      chk1,W
370     xorwf     chk2,W                ; Si (chk1 != chk2)
371     movlw     DATA_BAD
372     btfss     STATUS,Z
373     goto      send_byte             ; checksum erroneo

375  checksum_ok
376     movlw     DATA_OK               ; checksum OK
377     call      putbyte

379  write_byte
380     call      write_eeprom          ; Escribimos en la eeprom
381     iorlw     0
382     movlw     WR_OK                 ; Escritura CORRECTA
383     btfsc     STATUS,Z
384     movlw     WR_BAD                ; Escritura ERRONEA

386  send_byte
387     call      putbyte               ; Enviamos el codigo de CORRECTO o ERRONEO
388     goto      receive               ; Pasamos a recibir

390  ;*****************************************************************************
391  ;*                                                                           *
392  ;*                                   putbyte                                 *
393  ;*                                                                           *
394  ;*****************************************************************************

396  putbyte
397     clrwdt
398     btfss     PIR1,TXIF             ; Esperamos a que el PIC mande el byte
399                                     ; anterior
400     goto      putbyte
401     movwf     TXREG                 ; Mandamos el nuevo dato
```

```
402     return

404 ;*****************************************************************************
405 ;*                                                                           *
406 ;*                                  getbyte                                  *
407 ;*                                                                           *
408 ;*****************************************************************************

410 getbyte
411     clrwdt
412     movf      time,W
413     btfsc     STATUS,Z              ; Comprobamos si time=0
414     goto      getbyte3
415     btfss     PIR1,TMR1IF           ; Comprobamos si se ha desbordado el TIMER1
416     goto      getbyte3              ; No hay desbordamiento
417     bcf       T1CON,TMR1ON          ; Timeout 0.1 sec
418     decfsz    time,F                ; Decrementamos time
419     goto      getbyte2
420     retlw     0                     ; Si time=0 salimos

422 getbyte2
423     bcf       PIR1,TMR1IF           ; Borramos el FLAG de desbordamiento de TIMER1
424     movlw     high TIMER            ; Reajustamos el valor del contador
425     movwf     TMR1H                 ; Ajustamos TIMER1 para un timeout de 0.1s
426     bsf       T1CON,TMR1ON

428 getbyte3
429     btfss     PIR1,RCIF             ; Mientras(!RCIF) (Datos no recibidos)
430     goto      getbyte
431     movf      RCREG,W               ; RCREG
432     return

434 ;*****************************************************************************
435 ;*                                                                           *
436 ;*                                write_eeprom                               *
437 ;*                                                                           *
438 ;*****************************************************************************

440 write_eeprom

442 ;*** *** *** *** *** *** *** *** ADAPTACION 1.2 *** *** *** *** *** *** *** ***

444     movlw     0x21                  ; Si (0x2100 <= tmpaddr <= 0x21FF) ..........
445     subwf     address+1,W
446     btfsc     STATUS,Z
447     goto      adapt12_3end          ; Fin de la adaptacion

449     banksel   EECON1
450     bsf       EECON1,EEPGD          ; EEPGD = 1 -> Memoria de programa
451     banksel   address
452     movf      address,W
453     andlw     0xfc
454     bsf       STATUS,RP1
455     movwf     EEADR                 ; EEADR = Parte baja de la direccion
456     bcf       STATUS,RP1
457     movf      address+1,W
458     bsf       STATUS,RP1
459     movwf     EEADRH                ; EEADRH = Parte alta de la direccio
460     movlw     buff
461     movwf     FSR

463 ; Leemos de la memoria y escribimos al principio del buffer

465 adapt12_loop1
466     bcf       STATUS,RP1
467     movf      lcount,W
468     btfsc     STATUS,Z
469     goto      adapt12_4end          ; Fin de la adaptacion
470     banksel   EECON1
471     bsf       EECON1,RD
472     nop
473     nop
474     bcf       STATUS,RP0
```

```
475     movf      EEDATH,W
476     movwf     INDF
477     incf      FSR,F
478     movf      EEDATA,W
479     movwf     INDF
480     incf      FSR,F
481     incf      EEADR,F
482     bcf       STATUS,RP1
483     decf      lcount,F
484     decf      address,F
485     incf      amount,F
486     incf      amount,F
487     goto      adapt12_loop1

489 ; Leemos de la memoria y escribimos al final del buffer

491 adapt12_4end
492     movf      amount,W
493     movwf     help
494     addlw     buff
495     movwf     FSR
496     bcf       STATUS,C
497     rrf       help,F

499     movf      address+1,W
500     bsf       STATUS,RP1
501     movwf     EEADRH
502     bcf       STATUS,RP1
503     movf      address,W
504     bsf       STATUS,RP1
505     movwf     EEADR
506     bcf       STATUS,RP1
507     movf      help,W
508     bsf       STATUS,RP1
509     addwf     EEADR,F
510     btfsc     STATUS,Z
511     incf      EEADRH,F

513 adapt12_loop2
514     bsf       STATUS,RP1
515     movf      EEADR,W
516     andlw     0x03                  ; Mientras (EEADR & 0x03) != 0
517     btfsc     STATUS,Z
518     goto      adapt12_3end

520     bsf       STATUS,RP0
521     bsf       EECON1,RD
522     nop
523     nop
524     bcf       STATUS,RP0
525     movf      EEDATH,W
526     movwf     INDF
527     incf      FSR,F                 ; FSR++
528     movf      EEDATA,W
529     movwf     INDF
530     incf      FSR,F                 ; FSR++
531     incf      EEADR,F
532     bcf       STATUS,RP1
533     incf      amount,F
534     incf      amount,F
535     goto      adapt12_loop2

537 adapt12_3end

539 ; *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** **

541     movf      address,W
542     movwf     tmpaddr               ; tmpaddr = address
543     movf      address+1,W
544     movwf     tmpaddr+1
545     clrf      count                 ; count=0

547 write_loop
```

```
548     movlw     NumRetries+1          ; retry = NumRetries+1
549     movwf     retry

551  w_e_l_1
552     movf      amount,W
553     subwf     count,W               ; Mientras (count<amount)
554     btfsc     STATUS,C
555     retlw     1
556     movf      count,W
557     addlw     buff                  ; Ajustamos el puntero al buffer
558     movwf     FSR

560  w_e_l_2
561     movlw     0x21                  ; Si (0x2100 <= tmpaddr <= 0x21FF) ..........
562     subwf     tmpaddr+1,W
563     bsf       STATUS,RP1
564     bsf       STATUS,RP0            ; Banco 3
565     btfsc     STATUS,Z
566     goto      data_eeprom

568  program_eeprom
569     bsf       EECON1,EEPGD          ; EEPGD = 1 -> Memoria de programa
570     clrf      STATUS
571     movlw     high (LoaderStart)    ; Si (tmpaddr >= LoaderStart) ...............
572     subwf     tmpaddr+1,W
573     movlw     low (LoaderStart)
574     btfsc     STATUS,Z
575     subwf     tmpaddr,W
576     btfsc     STATUS,C
577     goto      next_adr              ; Siguiente direccion
578     goto      w_e_l_3

580  data_eeprom
581     bcf       EECON1,EEPGD          ; EEPGD = 0 -> Memoria de datos
582     clrf      STATUS

584  w_e_l_3
585     movf      tmpaddr,W
586     bsf       STATUS,RP1
587     movwf     EEADR                 ; EEADR = low tmpaddr
588     bcf       STATUS,RP1
589     movf      tmpaddr+1,W           ; Si (tmpaddr < 0x0004) .....................
590     btfss     STATUS,Z
591     goto      w_e_l_4
592     movlw     4
593     subwf     tmpaddr,W
594     btfsc     STATUS,C
595     goto      w_e_l_4
596     bsf       STATUS,RP1            ; Banco 3
597     bsf       STATUS,RP0
598     btfss     EECON1,EEPGD          ; Saltar si (EEPGD)
599     goto      w_e_l_31
600     bcf       STATUS,RP0            ; Banco 2

602  ;*** *** *** *** *** *** *** *** ** ADAPTACION *** *** *** *** *** *** *** *** *

604     movlw     low UserStart+1       ; EEADRL + low UserStart+1

606  ; *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** **

608     addwf     EEADR,F               ; Relocalizamos las primeras 4 instrucciones

610  w_e_l_31
611     clrf      STATUS                ; Banco 0
612     movlw     high UserStart        ; EEADRH = high UserStart
613     goto      w_e_l_5

615  w_e_l_4
616     movf      tmpaddr+1,W           ; EEADRH = high tmpaddr

618  w_e_l_5
619     bsf       STATUS,RP1
620     movwf     EEADRH                ; Ajustamos EEADRH
```

```
621     movf      INDF,W
622     movwf     EEDATH                ; EEDATH = buff[count]
623     incf      FSR,F
624     movf      INDF,W
625     movwf     EEDATA                ; EEDATA = buff[count+1]
626     bsf       STATUS,RP0
627     bsf       EECON1,WREN           ; WREN=1
628     movlw     0x55                  ; EECON2 = 0x55
629     movwf     EECON2
630     movlw     0xAA                  ; EECON2 = 0xAA
631     movwf     EECON2
632     bsf       EECON1,WR             ; WR=1
633     nop                             ; Esperamos a que se complete a escritura
634     nop                             ; Esperamos a que se complete a escritura
635     clrf      STATUS

637 ;*** *** *** *** *** *** *** *** ** ADAPTACION *** *** *** *** *** *** *** *** *

639     banksel   EECON1
640     btfsc     EECON1,EEPGD          ; Saltar si (EEPGD=0 / Escribir a EEPROM)
641     goto      ver_flash

643 ; *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** **

645     banksel PIR2

647 wait_write
648     clrwdt
649     btfss     PIR2,EEIF             ; Necesario para escribir en la eeprom de datos
650     goto      wait_write
651     bcf       PIR2,EEIF
652     bsf       STATUS,RP0            ; Banco 3
653     bsf       STATUS,RP1
654     bcf       EECON1,WREN           ; WREN=0


657     bsf       EECON1,RD             ; RD=1
658     nop
659     nop
660     bcf       STATUS,RP0            ; Banco 2
661     decf      FSR,F
662     movf      INDF,W                ; Si ((EEDATH != buff[count]) ||
663     xorwf     EEDATH,W              ; || (EEDATA != buff[count+1]))
664     btfss     STATUS,Z
665     goto      w_e_l_6               ; Repetimos la escritura
666     incf      FSR,F
667     movf      INDF,W
668     xorwf     EEDATA,W
669     btfsc     STATUS,Z
670     goto      next_adr              ; Verificacion correcta, siguiente direccion

672 w_e_l_6
673     clrf      STATUS                ; Banco 0
674     decfsz    retry,F
675     goto      w_e_l_1               ; Si (--retry != 0) repetimos la escritura
676     retlw     0                     ; si no devolvemos 0

678 next_adr

680 ;*** *** *** *** *** *** *** *** ** ADAPTACION *** *** *** *** *** *** *** *** *

682     banksel   count

684 ; *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** **

686     movlw     2                     ; count := count + 2
687     addwf     count,F
688     incf      tmpaddr,F             ; tmpaddr := tmpaddr + 1
689     btfsc     STATUS,Z
690     incf      tmpaddr+1,F
691     goto      write_loop

693 ;*** *** *** *** *** *** *** *** ** ADAPTACION *** *** *** *** *** *** *** *** *
```

```
695  ver_flash
696     banksel  PIR2
697     bcf      PIR2,EEIF
698     banksel  EECON1
699     bcf      EECON1,WREN          ; WREN=0

701     banksel  EEADR
702     movf     EEADR,W
703     andlw    0x03
704     sublw    0x03
705     btfss    STATUS,Z
706     goto     next_adr             ; Si (EEADR & 0x03) = 0x03

708     movlw    3
709     subwf    EEADR,F              ; EEADR <- EEADR - 3

711     banksel  help
712     movlw    4
713     movwf    help                 ; help <- 4

715     movlw    7
716     subwf    FSR,F                ; FSR <- FSR - 7

718  ver_fl_1
719     nop
720     banksel  EECON1
721     bsf      EECON1,RD            ; RD=1
722     nop
723     nop
724     nop
725     nop
726     nop
727     nop

729     movf     INDF,W               ; Si ((EEDATH != buff[count]) ||
730     banksel  EEDATH               ; || (EEDATA != buff[count+1]))
731     xorwf    EEDATH,W
732     btfss    STATUS,Z
733     goto     ver_fl_2             ; Repetimos la escritura
734     incf     FSR,F
735     movf     INDF,W
736     xorwf    EEDATA,W
737     btfsc    STATUS,Z
738     goto     ver_fl_3

740  ver_fl_2                         ; Verificacion erronea
741     banksel  tmpaddr
742     movlw    0x0fc
743     andwf    tmpaddr,F            ; Borramos los dos bits menos significativos de
744                                   ; [tmpaddr]
745     movlw    6
746     subwf    count,F              ; count <- count - 6

748     goto     w_e_l_6

750  ver_fl_3
751     banksel  EEADR
752     incf     EEADR,F
753     incf     FSR,F

755     banksel  help
756     decfsz   help,F
757     goto     ver_fl_1
758     goto     next_adr

760  ; *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** **

762    END
```

A.2. Programa Principal

El programa principal es el encargado de inicializar todos los dispositivos y de gestionar la interacción entre el dispositivo y el usuario.

Se encuentra localizado en la parte mas baja de la memoria de programa, al empezar la **Página 0**.

La definición de variables se hace a través de un fichero separado ya que como se hace uso de los 4 bancos de memoria de los que dispone el microcontrolador para este uso, es necesario tener absolutamente controladas las posiciones de memoria que ocupa cada variable. No podemos dejar que sea el compilador el que las sitúe en memoria.

A.2.1. Cabecera - Variables de Programa

```
;*****************************************************************************
;*****************************************************************************
;**                                                                         **
;**                        Posiciones de memoria SRAM                       **
;**                                                                         **
;*****************************************************************************
;*****************************************************************************

SECTOR            EQU 0x0000      ; Posicion a partir de la cual se almacena un
                                  ; sector de HDD completo para su posterior
                                  ; tratamiento.

                                  ; LONGITUD -> 512 Bytes
                                  ; FIN      -> 0x00FF

FAT_CACHEADA      EQU 0x0100      ; Posicion a partir de la cual se almacena
                                  ; parte de la tabla FAT de acceso a disco.

                                  ; LONGITUD -> 512 Bytes
                                  ; FIN      -> 0x01FF

NOMBRE_DIRECTORIO EQU 0x0200      ; Posicion a partir de la cual se almacena el
                                  ; nombre del directorio en el que se encuentra
                                  ; el fichero que se esta reproduciendo.
                                  ; El tamaño del nombre de directorio esta
                                  ; limitado en FAT32 a 255 Caracteres

                                  ; LONGITUD -> 256 Bytes (Solo usamos los 8
                                  ;             primeros Bits de cada posicion
                                  ;             de memoria.)
                                  ; FIN      -> 0x02FF

NOMBRE_FICHERO    EQU 0x0300      ; Posicion a partir de la cual se almacena el
                                  ; nombre del fichero que se esta reproduciendo.
                                  ; El tamaño del nombre de fichero esta limitado
                                  ; en FAT32 a 255 Caracteres

                                  ; LONGITUD -> 256 Bytes (Solo usamos los 8
                                  ;             primeros Bits de cada posicion de
                                  ;             memoria.)
                                  ; FIN      -> 0x03FF

BUFFER_REPRO      EQU 0x2000      ; Zona de memoria reservada para actuar como
                                  ; buffer intermedio entre el HDD y el
                                  ; decodificador de MP3.

                                  ; LONGITUD -> 48 KBytes
                                  ; FIN      -> 0x7FFF
```

```
50  ;*****************************************************************************
51  ;*****************************************************************************
52  ;**                                                                         **
53  ;**                          Variables de Programa                          **
54  ;**                                                                         **
55  ;*****************************************************************************
56  ;*****************************************************************************

59  ;*******    *********** ***     *** ********* ***********             *********
60  ;*********  *********** ***     *** ********* ***********            ***********
61  ;********** *********** ****    *** ********* ***********            ***********
62  ;***   ***** ****   **** *****   *** ****      ****   ****            ****   ****
63  ;***   ***** ****   **** ******  *** ****      ****   ****            ****   ****
64  ;***   ***** ****   **** ******* *** ****      ****   ****            ****   ****
65  ;********** *********** *** ******* ****      ****   ****  ********  ****   ****
66  ;*********  *********** ***  ****** ****      ****   ****  ********  ****   ****
67  ;********** *********** ***   ***** ****      ****   ****  ********  ****   ****
68  ;***   ***** ****   **** ***    **** ****      ****   ****            ****   ****
69  ;***   ***** ****   **** ***     *** ****      ****   ****            ****   ****
70  ;***   ***** ****   **** ***     *** ****      ****   ****            ****   ****
71  ;********** ****   **** ***     *** ********* ***********            ***********
72  ;*********  ****   **** ***     *** ********* ***********            ***********
73  ;*******    ****   **** ***     *** ********* ***********             *********

75  ;*****************************************************************************
76  ;*****************************************************************************

78  ;                      ZONA DE MEMORIA -> 0x0020 - 0x007F

80  ;*****************************************************************************
81  ;*****************************************************************************

83  ; ****************************************************************************
84  ; *                                                                          *
85  ; *                      Variables del Programa Principal                    *
86  ; *                                                                          *
87  ; ****************************************************************************

89  RespaldoW         EQU 0x0020     ; Registro de Respaldo de W
90  Temp0             EQU 0x0021     ; Variables Temporales
91  Temp1             EQU 0x0022
92  Contador1         EQU 0x0023     ; Registro contador
93  Tempo1            EQU 0x0024
94  Tempo2            EQU 0x0025
95  Tempo3            EQU 0x0026
96  N_Fichero         EQU 0x0027     ; Variables de navegacion por el dispositivo
97  N_Directorio      EQU 0x0028     ; IDE
98  Dato0             EQU 0x0029
99  Dato1             EQU 0x002A
100 Dato2             EQU 0x002B
101 Dato3             EQU 0x002C
102 Dato4             EQU 0x002D
103 Cifras            EQU 0x002E
104 NBit              EQU 0x002F
105 Digito            EQU 0x0030

107 ; ***************************************************************** **********
108 ; *                                                                          *
109 ; *               Variables de las funciones de control del LCD              *
110 ; *                                                                          *
111 ; ***************************************************************** **********

113 TempLCD           EQU 0x0031
114 ContadorLCD       EQU 0x0032
115 WBackup           EQU 0x0033     ; Respaldo de W
116 Caracter          EQU 0x0034     ; Puntero al mensaje
117 OffsetD           EQU 0x0035
118 OffsetF           EQU 0x0036
119 PosLibres         EQU 0x0037
120 CarScroll         EQU 0x0038
121 Mensaje           EQU 0x0039     ; Variable que contiene el Mensaje a pintar
```

```
                                ; Solo puede haber 2 1's en el Byte, el que indica el
                                ; numero de mensaje y el que indica el grupo.
                                ;
                                ; Si los dos BITS de grupo estan a 0 el Grupo
                                ; seleccionado es el GrupoA
                                ;
                                ;            [7]  [6]  [5]  [4]  [3]  [2]  [1]  [0]
                                ; Mensaje ->  X    X    X    X    X    X    X    X
                                ;             |    |    |    |    |    |    |    |_ Mensaje0
                                ;             |    |    |    |    |    |    |_____ Mensaje1
                                ;             |    |    |    |    |    |_________ Mensaje2
                                ;             |    |    |    |    |_____________ Mensaje3
                                ;             |    |    |    |_________________ Mensaje4
                                ;             |    |    |_______________________ Mensaje5
                                ;             |    |
                                ;             |    |_____________________________ GrupoB
                                ;             |_______________________________ GrupoC

; ****************************************************************************
; *                                                                          *
; *                  Variables de las funciones de control IDE               *
; *                                                                          *
; ****************************************************************************

NumSectRW          EQU 0x003A     ; Numero de sectores a leer o escribir
NumSectTR          EQU 0x003B     ; Numero de sectores a transferir
NumWords           EQU 0x003C     ; Numero de Words a transferir
IDE_Sec0           EQU 0x003D     ; Numero de sector a leer o escribir del
IDE_Sec1           EQU 0x003E     ; dispositivo IDE
IDE_Sec2           EQU 0x003F
IDE_Sec3           EQU 0x0040
Contador           EQU 0x0041     ; Pues eso
IDE_Status         EQU 0x0042     ; Copia del registro de estado del IDE
DATOS_BAJOS_B      EQU 0x0043     ; Respaldo de DATOS_BAJOS
SRam_Hi            EQU 0x007C     ; Parte alta de la direccion SRam
SRam_Lo            EQU 0x007D     ; Parte baja de la direccion SRam

; ****************************************************************************
; *                                                                          *
; *                Variables de las funciones de control FAT32               *
; *                                                                          *
; ****************************************************************************

TempFAT0           EQU 0x0046
ContadorFAT0       EQU 0x0047
ContadorFAT1       EQU 0x0048
FAT_Inicio0        EQU 0x0049
FAT_Inicio1        EQU 0x004A
FAT_Inicio2        EQU 0x004B
FAT_Inicio3        EQU 0x004C
FAT_Cacheado0      EQU 0x004D
FAT_Cacheado1      EQU 0x004E
FAT_Cacheado2      EQU 0x004F
FAT_Cacheado3      EQU 0x0050
Datos_Inicio0      EQU 0x0051
Datos_Inicio1      EQU 0x0052
Datos_Inicio2      EQU 0x0053
Datos_Inicio3      EQU 0x0054
Cluster_Raiz0      EQU 0x0055
Cluster_Raiz1      EQU 0x0056
Cluster_Raiz2      EQU 0x0057
Cluster_Raiz3      EQU 0x0058
Cluster_Dir0       EQU 0x0059
Cluster_Dir1       EQU 0x005A
Cluster_Dir2       EQU 0x005B
Cluster_Dir3       EQU 0x005C
Cluster_Ent0       EQU 0x005D
Cluster_Ent1       EQU 0x005E
Cluster_Ent2       EQU 0x005F
Cluster_Ent3       EQU 0x0060
Cluster_Actual0    EQU 0x0061
```

```
195  Cluster_Actual1    EQU 0x0062
196  Cluster_Actual2    EQU 0x0063
197  Cluster_Actual3    EQU 0x0064
198  TamanoArchivo0     EQU 0x0065
199  TamanoArchivo1     EQU 0x0066
200  TamanoArchivo2     EQU 0x0067
201  TamanoArchivo3     EQU 0x0068
202  EntBuscada_Hi      EQU 0x0069
203  EntBuscada_Lo      EQU 0x006A
204  EntEncontrada_Hi   EQU 0x006B
205  EntEncontrada_Lo   EQU 0x006C
206  EntPendientes      EQU 0x006D
207  SecCluster         EQU 0x006E
208  SecPendientes      EQU 0x006F
209  FactorRol          EQU 0x0045
210  NumEntradaFAT32    EQU 0x0044
211  I_NombreLargo      EQU 0x0072
212  T_NombreLargo      EQU 0x0073
213  ESTADO_BUSQUEDA    EQU 0x0074

215     PLAY            EQU 0x00
216     BUSCAR_FICH     EQU 0x01
217     NC_ENCONTRADO   EQU 0x02
218     ULT_ENTRADA     EQU 0x03
219     ULT_CLUSTER     EQU 0x04
220     SCROLL_F        EQU 0x05
221     SCROLL_D        EQU 0x06
222     BIToSAMPLE      EQU 0x07

224                          ; BYTE de ESTADO de la Busqueda
225                          ; ----------------------------
226                          ;
227                          ;*******************************************************
228                          ; BIT[0] (PLAY)            -> 0 = Reproduccion Detenida
229                          ;                          -> 1 = Reproduccion Activada
230                          ;*******************************************************
231                          ; BIT[1] (BUSCAR_FICH)     -> 0 = Buscamos Directorio
232                          ;                          -> 1 = Buscamos Fichero
233                          ;*******************************************************
234                          ; BIT[2] (NC_ENCONTRADO)   -> 0 = Entrada de NOMBRE CORTO
235                          ;                                no encontrada
236                          ;                          -> 1 = Encontrada
237                          ;*******************************************************
238                          ; BIT[3] (ULT_ENTRADA)     -> 0 = NO es la ultima entrada
239                          ;                          -> 1 = Es la ultima entrada
240                          ;*******************************************************
241                          ; BIT[4] (ULT_CLUSTER)     -> 0 = Hay mas clusters
242                          ;                          -> 1 = No hay mas clusters
243                          ;*******************************************************
244                          ; BIT[5] (SCROLL_F)        -> 0 = Scroll de Nombre de
245                          ;                                Fichero Desactivado
246                          ;                          -> 1 = Scroll Activado
247                          ;*******************************************************
248                          ; BIT[6] (SCROLL_D)        -> 0 = Scroll de Nombre de
249                          ;                                Diretorio Desactivado
250                          ;                          -> 1 = Scroll Activado
251                          ;*******************************************************
252                          ; BIT[7] (BIToSAMPLE)      -> 0 = Mostrar BIT RATE
253                          ;                          -> 1 = Mostrar SAMPLE RATE
254                          ;*******************************************************

256  ; ***************************************************************************
257  ; *                                                                         *
258  ; *             Respaldo del registro de Volumen del decodificador          *
259  ; *                                                                         *
260  ; ***************************************************************************

262  Volumen             EQU 0x0075

264  ; ***************************************************************************
265  ; *                                                                         *
266  ; *                        Manejo de Interrupciones                         *
267  ; *                                                                         *
```

```
; ****************************************************************************

INTerrupcion        EQU 0x007B

   DIRE             EQU 0x00
   FICH             EQU 0x01
   TIEM             EQU 0x02
   BoS              EQU 0x03
   BOTON            EQU 0x04
   INFO             EQU 0x05
   ZEROS            EQU 0x06

                       ; Registro de CONTROL de las INTERRUPCIONES
                       ; ----------------------------------------
                       ;
                       ;*******************************************************
                       ; BIT[0] (DIRE)            -> 1 = Activar Scroll de
                       ;                                 nombre de Directorio
                       ;*******************************************************
                       ; BIT[1] (FICH)            -> 1 = Activar Scroll de
                       ;                                 nombre de Fichero
                       ;*******************************************************
                       ; BIT[2] (TIEM)            -> 1 = Incrementar el contador
                       ;                                 de segundos
                       ;*******************************************************
                       ; BIT[3] (BoS)             -> 1 = Permuta la informacion
                       ;                                 mostrada en pantalla de
                       ;                                 Bit o Sample Rate.
                       ;*******************************************************
                       ; BIT[4] (BOTON)           -> 1 = Aceptar la pulsacion de
                       ;                                 Boton
                       ;*******************************************************
                       ; BIT[5] (INFO)            -> 1 = Extrae la Informacion
                       ;                                 de la cancion
                       ;*******************************************************
                       ; BIT[6] (ZEROS)           -> 1 = Enviar Ceros al Deco
                       ;*******************************************************
                       ; BIT[7] ()                -> 1 =
                       ;*******************************************************

; ****************************************************************************
; *                                                                          *
; *             Respaldo de W y STATUS para las Interrupciones               *
; *                                                                          *
; ****************************************************************************

W_Temp            EQU 0x0076
STATUS_Temp       EQU 0x0077

PuntBufferEsc_H   EQU 0x0070
PuntBufferEsc_L   EQU 0x0071
PuntBufferLec_H   EQU 0x007E
PuntBufferLec_L   EQU 0x007F

DirMem_H          EQU 0x0078
DirMem_L          EQU 0x0079

Boton_B           EQU 0x007A


;*******    *********** ***     *** ********* ***********                   ****
;*********  *********** ***     *** ********* ***********                  *****
;*********** *********** ****    *** ********* ***********                 ******
;***   ***** ****    **** *****   *** ****      ****   ****               *******
;***   ***** ****    **** ******  *** ****      ****   ****              ********
;***   ***** ****    **** ******* *** ****      ****   ****             *********
;********** *********** *** ******* ****      ****   ****   ********        ****
;*********  *********** ***  ****** ****      ****   ****   ********        ****
;*********** *********** ***   ***** ****      ****   ****   ********        ****
;***   ***** ****    **** ***    **** ****      ****   ****                   ****
;***   ***** ****    **** ***     *** ****      ****   ****                   ****
;***   ***** ****    **** ***     *** ****      ****   ****                   ****
```

```
;********** ****   **** ***     *** ********* ***********                   ****
;*********  ****   **** ***     *** ********* ***********                   ****
;*******    ****   **** ***     *** ********* ***********                   ****

;*****************************************************************************
;*****************************************************************************

;                     ZONA DE MEMORIA -> 0x00A0 - 0x00EF

;*****************************************************************************
;*****************************************************************************

ENTRADA_FAT32     EQU 0x00A0     ; [32 BYTES] [A0-BF] Entrada FAT32 completa

Digitos           EQU 0x00C0     ; [13 BYTES] [C0-CC] Valor ASCII convertido por
                                 ;                    Bin2Dec

RespaldoLinea1    EQU 0x00CD     ; [17 BYTES] [CD-DD] Respaldo de la Linea 1 del
                                 ;                    LCD

RespaldoLinea2    EQU 0x00DE     ; [17 BYTES] [DE-EE] Respaldo de la Linea 2 del
                                 ;                    LCD

;XXXXXX           EQU 0x00EF

;*******    *********** ***     *** ********* ***********            ***********
;*********  *********** ***     *** ********* ***********            ***********
;********** *********** ****    *** ********* ***********            ***********
;***  ***** ****   **** *****   *** ****      ****   ****                   ****
;***  ***** ****   **** ******  *** ****      ****   ****                   ****
;***  ***** ****   **** ******* *** ****      ****   ****                   ****
;********** *********** *** ******* ****      ****   ****  ********  ***********
;*********  *********** ***  ****** ****      ****   ****  ********  ***********
;********** *********** ***   ***** ****      ****   ****  ********  ***********
;***  ***** ****   **** ***    **** ****      ****   ****            ****
;***  ***** ****   **** ***     *** ****      ****   ****            ****
;***  ***** ****   **** ***     *** ****      ****   ****            ****
;********** ****   **** ***     *** ********* ***********            ***********
;*********  ****   **** ***     *** ********* ***********            ***********
;*******    ****   **** ***     *** ********* ***********            ***********

;*****************************************************************************
;*****************************************************************************

;                     ZONA DE MEMORIA -> 0x0120 - 0x016F

;*****************************************************************************
;*****************************************************************************

; ****************************************************************************
; *                                                                          *
; *                Variables de manejo del Buffer intermedio                 *
; *                                                                          *
; ****************************************************************************

DMinutos          EQU 0x0120     ; Decenas de Minutos de cancion transcurridos
                                 ; (ASCII)
UMinutos          EQU 0x0121     ; Unidades de Minutos de cancion transcurridos
                                 ; (ASCII)
DSegundos         EQU 0x0122     ; Decenas de Segundos de cancion transcurridos
                                 ; (ASCII)
USegundos         EQU 0x0123     ; Unidades de Segundos de cancion transcurridos
                                 ; (ASCII)

BloquesLibres     EQU 0x0124     ; Numero de Bloques de 2K libres en el BUFFER.

MiniBloques       EQU 0x0125     ; Numero de envios de 32 Bytes que falta por
                                 ; realizar, para liberar completamente un
                                 ; bloque de 2K Bytes

Cont32Bytes       EQU 0x0126

BitRate           EQU 0x0127     ; Bit Rate del fichero reproducido (ISO 1117-3)
```

```
Layer              EQU 0x0128     ; "Layer" (MPEG) del Fichero Reproducido
ID                 EQU 0x0129     ; Algoritmo de codificacion usado
SampleRate         EQU 0x0130     ; Pues eso

IntScrollFich      EQU 0x0131     ; Marcador de Interrupciones (Scroll de
                                  ; Fichero)
IntScrollDir       EQU 0x0132     ; Marcador de Interrupciones (Scroll de
                                  ; Directorio)
IntContTiempo      EQU 0x0133     ; Marcador de Interrupciones (Incremento de 1
                                  ; segundo)
IntBitSample       EQU 0x0140     ; Marcador de Interrupciones (Permuta Bit o
                                  ; Sample Rate)
IntBoton           EQU 0x0141     ; Marcador de Interrupciones (Pulsacion de
                                  ; Boton)

; ****************************************************************************
; *                                                                          *
; * Respaldo de los registros de control, FSR y PCLATH para las interrupciones *
; *                                                                          *
; ****************************************************************************

CTRL1_Temp         EQU 0x0134
CTRL2_Temp         EQU 0x0135
FSR_Temp           EQU 0x0136
PCLATH_Temp        EQU 0x0137
DATOS_HTemp        EQU 0x0138
DATOS_LTemp        EQU 0x0139

;*******    *********** ***      *** ********* ***********             ***********
;*********  *********** ***      *** ********* ***********             ***********
;********** *********** ****     *** ********* ***********             ***********
;***   ***** ****    **** *****    *** ****       ****    ****                    ****
;***   ***** ****    **** ******   *** ****       ****    ****                    ****
;***   ***** ****    **** ******* *** ****       ****    ****                    ****
;********** *********** *** ******* ****       ****    ****   ********  ***********
;*********  *********** ***   ****** ****       ****    ****   ********  ***********
;********** *********** ***    ***** ****       ****    ****   ********  ***********
;***   ***** ****    **** ***     **** ****       ****    ****                    ****
;***   ***** ****    **** ***      *** ****       ****    ****                    ****
;***   ***** ****    **** ***      *** ****       ****    ****                    ****
;********** ****    **** ***      *** ********* ***********             ***********
;*********  ****    **** ***      *** ********* ***********             ***********
;*******    ****    **** ***      *** ********* ***********             ***********

;****************************************************************************
;****************************************************************************

;                     ZONA DE MEMORIA -> 0x01A0 - 0x01EF

;****************************************************************************
;****************************************************************************

; ****************************************************************************
; *                                                                          *
; * Respaldo de TRISB y TRISD (Configuracion del BUS) para las interrupciones *
; *                                                                          *
; ****************************************************************************

TRISB_Temp         EQU 0x01A0
TRISD_Temp         EQU 0x01A1
```

A.2.2. Programa

```
1          LIST        P=16F877A
2          INCLUDE     "P16F877A.inc"
3          ERRORLEVEL  -302
4          __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _HS_OSC & _PWRTE_ON & _LVP_OFF

6          INCLUDE "Var_MP3.inc"
7  ;*****************************************************************************
8  ;*****************************************************************************
9  ;**                                                                         **
10 ;**                          Asignacion de Puertos                          **
11 ;**                                                                         **
12 ;*****************************************************************************
13 ;*****************************************************************************

15 DATOS_BAJOS     EQU PORTB          ; Parte Baja del Bus de Datos
16 DATOS_ALTOS     EQU PORTD          ; Parte Alta del Bus de Datos
17 DIR_SRAM_L      EQU PORTB          ; Parte Baja del Bus de Direcciones
18 DIR_SRAM_H      EQU PORTD          ; Parte Alta del Bus de Direcciones
19 CONTROL1        EQU PORTA          ; Puerto 1 de Control
20 CONTROL2        EQU PORTE          ; Puerto 2 de Control
21 COMANDO_LCD     EQU PORTD          ; Puerto de envio de comandos al LCD
22 COMANDO_IDE     EQU PORTB          ; Puerto de envio de comandos al dispositivo
23                                    ; IDE
24 DECO1           EQU PORTC          ; Puerto 1 de comunicacion con el decodificador
25                                    ; MP3
26 DECO2           EQU PORTE          ; Puerto 2 de comunicacion con el decodificador
27                                    ; MP3

29 ;*****************************************************************************
30 ;*****************************************************************************
31 ;**                                                                         **
32 ;**                          Asignacion de Pines                            **
33 ;**                                                                         **
34 ;*****************************************************************************
35 ;*****************************************************************************
36 ;*** CONTROL 1 *** IDE ***
37 DIOW            EQU 3              ; Dispositiovo IDE (Escritura; BUS -> IDE)

39 ;*** CONTROL 2 *** SRAM ***
40 WE              EQU 1              ; Memoria SRAM (Escritura; BUS -> MEMORIA)
41 INC_ADDR        EQU 2              ; Latch de direccion (Incrementar direccion)

43 ;*** COMANDO_LCD *** LCD ***
44 LCD_RS          EQU 0              ; LCD (0 -> Comando) (1 -> Dato)
45 LCD_RW          EQU 1              ; LCD (0 -> Escritura) (1 -> Lectura)

47 ; *** DECO ***

49 ;*****************************************************************************
50 ;*****************************************************************************
51 ;**                                                                         **
52 ;**                 Codigos de habilitacion del Decodificador               **
53 ;**                                                                         **
54 ;*****************************************************************************
55 ;*****************************************************************************

57 ; Estos son los codigos sirven que hay que introducir en el demultiplexor para
58 ; activar el acceso de los distintos dispositivos al BUS de datos.

60 DIOR            EQU b'001000'      ; Dipositivo IDE (Lectura; IDE -> BUS)
61 OE              EQU b'001001'      ; Memoria SRAM (Lectura; MEMORIA -> BUS)
62 LATCH_MEM_L     EQU b'001010'      ; Latch de direccion de memoria
63 LCD_E           EQU b'001011'      ; Pantalla LCD
64 BUTTON_E        EQU b'001100'      ; Botones de manejo
65 SERIAL_E        EQU b'001101'      ; Puerto serie PC
66 LATCH_IDE_L     EQU b'001110'      ; Latch de comando IDE
67 OCIOSO          EQU b'001111'      ; Ninguna señal activa

69 ;*****************************************************************************
70 ;*****************************************************************************
71 ;**                                                                         **
```

```
72  ;**                    Comienzo del Programa Principal                      **
73  ;**                                                                          **
74  ;******************************************************************************
75  ;******************************************************************************

77     org       0x0000

79  Arranque
80     goto      Inicio

82     org       0x0004

84  ;******************************************************************************
85  ;*                                                                            *
86  ;*                            RUTINA DE INTERRUPCION                          *
87  ;*                                                                            *
88  ;******************************************************************************

90  Interrupcion
91     bcf       INTCON,GIE            ; Deshabilitamos las Interrupciones globales

93  ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

95  ; Salvaguardamos el Contexto

97     movwf     W_Temp                ; W_Temp y STATUS_Temp son accesibles desde
98                                     ; cualquier banco
99     swapf     STATUS,W
100    movwf     STATUS_Temp           ; W y STATUS estan replicados en los 4 Bancos

102    clrf      STATUS                ; 00 -> Banco 0
103    bsf       STATUS,RP1            ; 10 -> Banco 2
104    movf      PCLATH,W              ; PCLATH esta replicado en los 4 Bancos
105    movwf     PCLATH_Temp
106    clrf      PCLATH

108    bcf       STATUS,IRP

110    movf      FSR,W                 ; FSR esta replicado en los 4 Bancos
111    movwf     FSR_Temp

113    bcf       STATUS,RP1            ; 00 -> Banco 0
114    movf      CONTROL1,W
115    bsf       STATUS,RP1            ; 10 -> Banco 2
116    movwf     CTRL1_Temp

118    bcf       STATUS,RP1            ; 00 -> Banco 0
119    movf      CONTROL2,W
120    bsf       STATUS,RP1            ; 10 -> Banco 2
121    movwf     CTRL2_Temp

123    bcf       STATUS,RP1            ; 00 -> Banco 0
124    movf      DATOS_BAJOS,W
125    bsf       STATUS,RP1            ; 10 -> Banco 2
126    movwf     DATOS_LTemp

128    bcf       STATUS,RP1            ; 00 -> Bamco 0
129    movf      DATOS_ALTOS,W
130    bsf       STATUS,RP1            ; 10 -> Banco 2
131    movwf     DATOS_HTemp

133    bcf       STATUS,RP1            ; 00 -> Banco 0
134    bsf       STATUS,RP0            ; 01 -> Banco 1
135    movf      TRISB,W
136    bsf       STATUS,RP1            ; 11 -> Banco 3
137    movwf     TRISB_Temp

139    bcf       STATUS,RP1            ; 01 -> Banco 1
140    movf      TRISD,W
141    bsf       STATUS,RP1            ; 11 -> Banco 3
142    movwf     TRISD_Temp

144 ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **
```

```
146     clrf      STATUS                ; 00 -> Banco 0

148 ;    movlw     OCIOSO                ; Desactivamos los perifericos controlados por
149 ;    movwf     CONTROL1              ; el Multiplexor

151 ;    bsf       CONTROL2,WE           ; Desactivamos la escritura en SRAM
152 ;    bcf       CONTROL2,INC_ADDR     ; Desactivamos el incremento de la direccion
153                                      ; SRAM
154 ;    bsf       CONTROL2,XRESET       ; Desactivamos el RESET del CHIP decodificador

156 ; Ahora reinicializamos el Temporizador

158     movlw     0x86                  ; Reinicializamos el valor del temporizador 1
159     movwf     TMR1H
160     clrf      TMR1L

162     bsf       STATUS,RP1            ; 10 -> Banco 2

164 Fich
165     decf      IntScrollFich,F
166     btfsc     STATUS,Z
167     goto      IntFich

169 Dir
170     decf      IntScrollDir,F
171     btfsc     STATUS,Z
172     goto      IntDir

174 Tiempo
175     decf      IntContTiempo,F
176     btfsc     STATUS,Z
177     goto      IntTiempo

179 Boton
180     decf      IntBoton,F
181     btfsc     STATUS,Z
182     goto      IntBotones
183     goto      CompruebaPlay

185 IntFich
186     bsf       INTerrupcion,FICH
187     movlw     d'16'
188     movwf     IntScrollFich
189     goto      Dir

191 IntDir
192     bsf       INTerrupcion,DIRE
193     movlw     d'26'
194     movwf     IntScrollDir
195     goto      Tiempo

197 IntTiempo
198     bsf       INTerrupcion,TIEM
199     movlw     d'40'
200     movwf     IntContTiempo
201     decfsz    IntBitSample,F
202     goto      Boton
203     bsf       INTerrupcion,BoS
204     movlw     d'5'
205     movwf     IntBitSample
206     goto      Boton

208 IntBotones
209     bsf       INTerrupcion,BOTON


212 CompruebaPlay
213     clrf      STATUS                ; Banco 0
214     btfss     ESTADO_BUSQUEDA,PLAY
215     goto      SalirInterrupcion

217 ; Esta parte de la Interrupcion solo se ejecuta si estamos reproduciendo un
```

```
218 ; Fichero

220     call      CargarBufferMP3

222     clrf      STATUS                ; 00 -> Banco 0

224 SalirInterrupcion

226 ; Recuperamos el valor de los registros de memoria (Contadores)

228     call      BUS_ConfEsc           ; Configuramos el PIC para escribir el LATCH de
229                                     ; memoria
230     movf      DirMem_H,W            ; Ponemos la direccion destino en el BUS de
231     movwf     DIR_SRAM_H            ; DATOS
232     movf      DirMem_L,W
233     movwf     DIR_SRAM_L
234     movlw     LATCH_MEM_L           ; Activamos LATCH_MEM_L
235     movwf     CONTROL1
236     nop                             ; Esperamos
237     movlw     OCIOSO                ; Desactivamos LATCH_MEM_L
238     movwf     CONTROL1

240     bcf       PIR1,TMR1IF           ; Borramos el FLAG del TIMER 1

242 ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

244 ; Recuperamos el valor de las variables de contexto

246     bsf       STATUS,RP0            ; 01 -> Banco 1
247     bsf       STATUS,RP1            ; 11 -> Banco 3
248     movf      TRISD_Temp,W
249     bcf       STATUS,RP1            ; 01 -> Banco 1
250     movwf     TRISD

252     bsf       STATUS,RP1            ; 11 -> Banco 3
253     movf      TRISB_Temp,W
254     bcf       STATUS,RP1            ; 01 -> Banco 1
255     movwf     TRISB

257     bcf       STATUS,RP0            ; 00 -> Banco 0
258     bsf       STATUS,RP1            ; 10 -> Banco 2
259     movf      DATOS_HTemp,W
260     bcf       STATUS,RP1            ; 00 -> Banco 0
261     movwf     DATOS_ALTOS

263     bsf       STATUS,RP1            ; 10 -> Banco 2
264     movf      DATOS_LTemp,W
265     bcf       STATUS,RP1            ; 00 -> Banco 0
266     movwf     DATOS_BAJOS

268     bsf       STATUS,RP1            ; 10 -> Banco 2
269     movf      CTRL2_Temp,W
270     bcf       STATUS,RP1            ; 00 -> Banco 0
271     movwf     CONTROL2

273     bsf       STATUS,RP1            ; 10 -> Banco 2
274     movf      CTRL1_Temp,W
275     bcf       STATUS,RP1            ; 00 -> Banco 0
276     movwf     CONTROL1

278     bsf       STATUS,RP1            ; 10 -> Banco 2
279     movf      FSR_Temp,W
280     movwf     FSR                   ; FSR esta replicado en los 4 Bancos

282     movf      PCLATH_Temp,W
283     movwf     PCLATH                ; PCLATH esta replicado en los 4 Bancos

285     swapf     STATUS_Temp,W         ; W y STATUS estan replicados en los 4 Bancos
286     movwf     STATUS

288     swapf     W_Temp,F              ; W_Temp y STATUS_Temp son accesibles desde
289     swapf     W_Temp,W              ; cualquier Banco
290     retfie                          ; Retornamos activando las interrupciones
```

```
292   ; ******************************************************************************
293   ; ******************************************************************************

295   Inicio

297   ;******************************************************************************
298   ;*                                                                            *
299   ;*                        Configuracion inicial de Puertos                    *
300   ;*                                                                            *
301   ;******************************************************************************

303   ;bcf      ESTADO_BUSQUEDA,7

305   ; ***************************** PORT A - CONTROL 1 ****************************

307      bsf       STATUS,RP0            ; Banco 1
308      bcf       STATUS,RP1
309      movlw     b'00000110'           ; PORTA Todo digital
310      movwf     ADCON1

312      movlw     b'00000000'
313      movwf     TRISA                 ; A0-A2  OUT -> CODIGO del decodificador
314                                      ; A3     OUT -> DIOW
315                                      ; A4     OUT -> Sin Asignar
316                                      ; A5     OUT -> Sin Asignar
317      bcf       STATUS,RP0            ; Banco 0
318      movlw     OCIOSO                ; Inicializamos las señales de control
319      movwf     CONTROL1              ; Ponemos todos los dispositivos en alta
320                                      ; impedancia

322   ; ************************** PORT B:D - BUS DE DATOS *************************

324      bsf       STATUS,RP0            ; Banco 1
325      clrf      TRISB                 ; B0-B7  OUT -> DATA0 - DATA7  (Por Defecto)
326      clrf      TRISD                 ; D0-D7  OUT -> DATA8 - DATA15 (Por Defecto)

328   ; ****************************** PORT E - CONTROL 2 **************************

330      clrf      TRISE                 ; E0     OUT -> DECO8 - #XRESET
331                                      ; E1     OUT -> WE
332                                      ; E2     OUT -> INC_ADDR
333      bcf       STATUS,RP0            ; Banco 0
334      bsf       CONTROL2,WE           ; Desactivamos la escritura en SRAM
335      bcf       CONTROL2,INC_ADDR     ; Desactivamos el incremento de la direccion
336                                      ; SRAM
337      bsf       CONTROL2,XRESET       ; Desactivamos el RESET del CHIP decodificador

339    ; ***************************** PORT C - DECODIFICADOR ************************

341      bsf       STATUS,RP0            ; Banco 1
342      movlw     b'11010010'
343      movwf     TRISC                 ; C0     OUT -> BSYNC
344                                      ; C1     IN  -> DREQ
345                                      ; C2     OUT -> #XCS
346                                      ; C3     OUT -> SCLK
347                                      ; C4     IN  -> SO
348                                      ; C5     OUT -> SI        Por defecto, seran
349                                      ; C6     IN  -> DCLK   \  configuradas
350                                      ; C7     IN  -> SDATA  /  adecuadamente en la
351                                      ;                         configuracion del
352                                      ;                         puerto serie
353                                      ;

355      bcf       STATUS,RP0            ; Banco 0
356      bcf       VSCONTROL,BSYNC       ; Desactivamos la sincronizacion de BIT del
357                                      ; decodificador
358      bsf       VSCONTROL,XCS         ; Desactivamos el CHIP decodificador

360   ; ************************** TEMPORIZACION DE INTERRUPCION ********************

362   ; Ahora vamos a preparar la temporizacion.
363   ; El Buffer del decodificador MP3 es de 16384 bits, como la velocidad de
```

```
364  ; reproduccion maxima es de 320 Kbits por segundo, tenemos que recargar el
365  ; Buffer del decodificador como maximo cada:
366  ;
367  ;                        16384
368  ;                    ------------ = 0,050 segundos = 50 ms
369  ;                     320 * 1024
370  ;
371  ; Como estamos trabajando a 20 MHz el ritmo del contador sera de FOsc/4 es decir
372  ; de 5MHz, a este ritmo y usando la preescala maxima, el TIMER0 nos generara una
373  ; interrupcion cada:
374  ;
375  ;                 (256 - InitTMR0) * preescala
376  ;      Retardo = ------------------------------
377  ;                          FOsc / 4
378  ;
379  ; El retardo maximo lo obtenemos inicializando el temporizador a 0 y con la
380  ; preescala de 256, con estos datos obtenemos un retardo maximo de 0,0131072 s,
381  ; es decir unos 13 ms.
382  ;
383  ; Si usamos el TIMER1 la cosa cambia, el contador es de 16 Bits y la preescala
384  ; maxima es de 8.
385  ;
386  ;                 (65536 - InitTMR1) * preescala
387  ;      Retardo = -------------------------------
388  ;                          FOsc / 4
389  ;
390  ; Con estos datos el retardo maximo es de 0,1048576 s, es decir, unos 105 ms,
391  ; mas que suficiente.
392  ; Si hacemos que el temporizador cuente 31232 ciclos, con una preescala de 4
393  ; conseguimos un retardo de 0,0249856 ms que se aproxima mucho alos 25 ms
394  ; buscados.
395  ; Para conseguir 31232 ciclos tenemos que inicializar el temporizador a
396  ; 65536 - 31232 = 34304 -> 0x8600

398     movlw    b'00100000'          ; [7:6]                 -> No implementados
399                                   ; [5:4] T1CKPS1:T1CKPS0 -> Preescala 1:4
400                                   ; [3]   T1OSCEN         -> Oscilador Parado
401                                   ; [2]   #T1SYNC         -> Ignorado al estar
402                                   ;                          TMR1CS a 0
403                                   ; [1]   TMR1CS          -> Fuente de reloj
404                                   ;                          interna
405                                   ; [0]   TMR1ON          -> Temporizador
406                                   ;                          detenido
407     movwf    T1CON
408     movlw    0x86
409     movwf    TMR1H
410     clrf     TMR1L

412  ; Ahora configuramos las interrupciones

414     clrf     INTCON               ; Desactivamos las interrupciones y borramos l
415                                   ; os Flags
416     clrf     PIR1                 ; Borramos los flags de interrupcion de los
417     clrf     PIR2                 ; perifericos
418     bsf      STATUS,RP0           ; Banco 1
419     clrf     PIE1                 ; Desactivamos todas las interrupciones de los
420     clrf     PIE2                 ; perifericos
421     bsf      PIE1,TMR1IE          ; Activamos las interrupciones del temporizador
422                                   ; 1
423     bsf      INTCON,PEIE          ; Activamos las interrupciones de los
424                                   ; perifericos
425     bcf      STATUS,RP0           ; Banco 0

427  ; ***** INICIALIZAMOS EL LCD *****
428     bsf      PCLATH,3
429     call     InicializaLCD
430  ; ********************************

432  ; ***** MENSAJE DE INICIO *****
433     call     PonTitulo
434     bcf      PCLATH,3
435     movlw    d'150'
436     call     Espera_L
```

```
437     bsf       PCLATH,3
438     call      LCD_LimpiaLinea1
439     call      LCD_LimpiaLinea2
440     call      LCD_LimpiaLinea3
441     call      LCD_LimpiaLinea4
442  ; ****************************

444  ; ***** INICIALIZAMOS EL DECODIFICADOR DE MP3 *****
445     call      Inicializa_SPI
446     call      Inicializa_VS1001
447     bcf       PCLATH,3
448  ; ************************************************

450  ; *****  MENSAJES DE INICIO ****
451     bsf       PCLATH,3
452     clrw                        ; Ponemos el cursor al comienzo de la Linea 1
453     call      LCD_Linea1
454     movlw     b'00000001'       ; Indicamos que mensaje vamos a pintar
455     movwf     Mensaje
456     call      LCD_PonMensaje

458     clrw                        ; Ponemos el cursor al comienzo de la Linea 3
459     call      LCD_Linea4
460     movlw     b'00000010'       ; Indicamos que mensaje vamos a pintar
461     movwf     Mensaje
462     call      LCD_PonMensaje
463     bcf       PCLATH,3

465     movlw     d'150'
466     call      Espera_L

468     bsf       PCLATH,3
469     call      LCD_LimpiaLinea1
470     call      LCD_LimpiaLinea4
471     bcf       PCLATH,3
472  ; ******************************

474  ; ***** INICIALIZAMOS EL DISPOSITIVO IDE *****
475     call      IDE_Reset         ; Reseteamos el dispositivo IDE

477     call      IDE_LBAon         ; Seleccionamos el modo LBA y el dispositivo 0

479     bsf       PCLATH,3
480     clrw                        ; Ponemos el cursor al comienzo de la Linea 1
481     call      LCD_Linea1

483     movlw     b'00000100'       ; Indicamos que mensaje vamos a pintar
484     movwf     Mensaje
485     call      LCD_PonMensaje
486     bcf       PCLATH,3
487  ; *****************************************

489  ; ***** IDENTIFICACION DEL DISPOSITIVO IDE *****
490     movlw     0x00              ; Cargamos la direccion de destino en la SRAM
491     movwf     SRam_Hi
492     movlw     0x00
493     movwf     SRam_Lo

495     call      IDE_Identifica
496  ; *********************************************

498  ; ***** MOSTRAMOS EL NUMERO DE MODELO DE IDE EN EL LCD *****
499     movlw     d'10'
500     call      Espera_L

502     bsf       PCLATH,3
503     clrw                        ; Ponemos el cursor al comienzo de la Linea 2
504     call      LCD_Linea2
505     movlw     b'00100000'
506     movwf     Mensaje
507     call      LCD_PonMensaje
508     bcf       PCLATH,3
```

```
510     movlw     d'50'
511     call      Espera_L

513     call      BUS_ConfEsc           ; Configuramos el BUS para escribir en el LATCH
514                                     ; MEMORIA
515     movlw     0x00
516     movwf     DIR_SRAM_H
517     movlw     0x1B                  ; Nos desplazamos a la direccion 27 (Model
518     movwf     DIR_SRAM_L            ; Number)
519     movlw     LATCH_MEM_L           ; Activamos LATCH_MEM_L
520     movwf     CONTROL1
521     nop                             ; Esperamos
522     movlw     OCIOSO                ; Desactivamos LATCH_MEM_L
523     movwf     CONTROL1
524     movlw     d'10'
525     movwf     Contador1

527 BucleModel
528     call      BUS_ConfLec           ; Configuramos el BUS para leer de la MEMORIA
529     movlw     OE                    ; Leemos la memoria SRam (OE)
530     movwf     CONTROL1
531     movf      DATOS_BAJOS,W
532     movwf     Temp0
533     movf      DATOS_ALTOS,W
534     movwf     Temp1
535     movlw     OCIOSO                ; Desactivamos OE
536     movwf     CONTROL1
537     bsf       PCLATH,3
538     movf      Temp1,W
539     call      LCD_EnviaDato
540     movf      Temp0,W
541     call      LCD_EnviaDato
542     bcf       PCLATH,3
543     bsf       CONTROL2,INC_ADDR     ; Incrementamos la direccion de destino
544     nop
545     bcf       CONTROL2,INC_ADDR
546     decfsz    Contador1,F
547     goto      BucleModel

549     movlw     d'50'
550     call      Espera_L

552     call      Inicializa_FAT32

554     movlw     d'50'
555     call      Espera_L

557     bsf       PCLATH,3
558     call      LCD_LimpiaLinea1
559     call      LCD_LimpiaLinea2
560     call      LCD_LimpiaLinea3
561     call      LCD_LimpiaLinea4

563     movlw     b'10000001'           ; Indicamos que mensaje vamos a pintar
564     movwf     Mensaje

566     call      LCD_PonMensaje
567     bcf       PCLATH,3

569     movf      Cluster_Raiz0,W       ; Indicamos que el directorio actual es el RAIZ
570     movwf     Cluster_Dir0
571     movf      Cluster_Raiz1,W
572     movwf     Cluster_Dir1
573     movf      Cluster_Raiz2,W
574     movwf     Cluster_Dir2
575     movf      Cluster_Raiz3,W
576     movwf     Cluster_Dir3

578     clrf      N_Fichero             ; Borramos los indices de Fichero y Directorio
579     clrf      N_Directorio

581     bsf       PCLATH,3
582     call      LCD_LimpiaLinea1      ;Preparamos la Pantalla
```

```
583     movlw     b'01000100'
584     movwf     Mensaje
585     call      LCD_PonMensaje

587     movlw     d'3'
588     call      LCD_Linea1
589     movlw     b'01010000'
590     movwf     Mensaje
591     call      LCD_PonMensaje

593     call      LCD_LimpiaLinea2
594     movlw     b'01001000'
595     movwf     Mensaje
596     call      LCD_PonMensaje
597     bcf       PCLATH,3

599     movlw     0xFF                  ; Comenzamos con un incremento
600     movwf     OffsetD
601     movwf     OffsetF

603     bsf       ESTADO_BUSQUEDA,BUSCAR_FICH
604     call      BorrarNombre
605     bcf       ESTADO_BUSQUEDA,BUSCAR_FICH
606     bcf       ESTADO_BUSQUEDA,SCROLL_F
607     bcf       ESTADO_BUSQUEDA,SCROLL_D
608     bcf       ESTADO_BUSQUEDA,PLAY

610     clrf      Dato0
611     clrf      Dato1
612     clrf      Dato2
613     clrf      Dato3
614     clrf      Dato4

616     bcf       STATUS,RP0
617     bsf       STATUS,RP1
618     movlw     d'40'
619     movwf     IntContTiempo
620     movlw     d'20'
621     movwf     IntBoton
622     bcf       STATUS,RP1

624 ; ***** Inicializamos las interrupciones *****
625     bsf       INTCON,GIE            ; Activamos las interrupciones globales
626     bsf       T1CON,TMR1ON          ; Activamos el Temporizador 1
627 ; *****************************************

629     bsf       INTerrupcion,BOTON

631 ; ***** Comenzamos la Navegacion *****

633 Escanea

635     call      Escanea_Botones

637     movwf     RespaldoW
638     btfss     RespaldoW,0
639     goto      IncDir
640     btfss     RespaldoW,4
641     goto      DecDir
642     btfss     RespaldoW,1
643     goto      IncFich
644     btfss     RespaldoW,5
645     goto      DecFich
646     btfss     RespaldoW,7
647     goto      Reproducir

649     bsf       PCLATH,3
650     call      Scroll
651     bcf       PCLATH,3

653     goto      Escanea

655 ;********************** BOTON 0 - Incrementar Directorio **********************
```

```
IncDir
   bsf      ESTADO_BUSQUEDA,BUSCAR_FICH   ; Cambio de directorio, el nombre de
   call     BorrarNombre                  ; fichero actual ya no es valido, lo
                                          ; borramos
   bcf      ESTADO_BUSQUEDA,BUSCAR_FICH   ; Indicamos que buscamos un directorio
   clrf     N_Fichero                     ; Primer fichero del directorio
   incf     N_Directorio,F                ; Directorio siguiente al actual

   bsf      PCLATH,3                      ; Limpiamos la linea de Fichero
   movlw    0xFF
   movwf    OffsetD
   call     LCD_LimpiaLinea2
   movlw    b'01001000'
   movwf    Mensaje
   call     LCD_PonMensaje
   bcf      PCLATH,3

   bcf      ESTADO_BUSQUEDA,SCROLL_D      ; Desactivamos el SCROLL de nombre de
                                          ; Directorio
   movf     N_Directorio,W                ; Buscamos la siguiente entrada de
   goto     ComienzaBusqueda              ; directorio
;*******************************************************************************

;********************** BOTON 4 - Decrementar Directorio **********************
DecDir
   bcf      ESTADO_BUSQUEDA,BUSCAR_FICH   ; Indicamos que buscamos un directorio
   clrf     N_Fichero                     ; Primer fichero del directorio
   movf     N_Directorio,F                ; Comprobamos si estamos en el
   btfsc    STATUS,Z                      ; directorio raiz
   goto     Escanea                       ; Estamos en el Raiz, no hacemos nada

   decf     N_Directorio,F                ; No es el Raiz, buscamos el
                                          ; directorio anterior
   movf     N_Directorio,F                ; Comprobamos si el anterior es el
   btfss    STATUS,Z                      ; Raiz
   goto     NoRaiz                        ; El Directorio anterior no es el Raiz

Raiz                                      ; El nuevo directorio es el Raiz,
   movf     Cluster_Raiz0,W               ; cargamos su Cluster  correspondiente
   movwf    Cluster_Dir0
   movf     Cluster_Raiz1,W
   movwf    Cluster_Dir1
   movf     Cluster_Raiz2,W
   movwf    Cluster_Dir2
   movf     Cluster_Raiz3,W
   movwf    Cluster_Dir3

   bsf      PCLATH,3                      ; Indicamos en la pantalla que estamos
   movlw    d'3'                          ; en el directorio Raiz.
   call     LCD_Linea1
   movlw    b'01010000'
   movwf    Mensaje
   call     LCD_PonMensaje
   bcf      PCLATH,3

   goto     Escanea                       ; Hemos acabado

NoRaiz                                    ; El nuevo directorio NO es el Raiz,
                                          ; lo buscamos
   bsf      ESTADO_BUSQUEDA,BUSCAR_FICH   ; Cambio de directorio, el nombre de
   call     BorrarNombre                  ; fichero actual ya no es valido, lo
                                          ; borramos
   bcf      ESTADO_BUSQUEDA,BUSCAR_FICH   ; Indicamos que buscamos un directorio

   bsf      PCLATH,3                      ; Limpiamos la linea de Fichero
   movlw    0xFF
   movwf    OffsetD
   call     LCD_LimpiaLinea2
   movlw    b'01001000'
   movwf    Mensaje
   call     LCD_PonMensaje
   bcf      PCLATH,3
```

```
729     bcf       ESTADO_BUSQUEDA,SCROLL_D       ; Desactivamos el SCROLL de nombre de
730                                              ; Directorio
731     movf      N_Directorio,W                 ; Buscamos la entrada anterior de
732     goto      ComienzaBusqueda               ; directorio
733   ;*******************************************************************************

735   ;************************ BOTON 1 - Incrementar Fichero ************************
736   IncFich
737     bsf       ESTADO_BUSQUEDA,BUSCAR_FICH    ; Indicamos que buscamos un Fichero
738     incf      N_Fichero,F                    ; Fichero siguiente al actual
739     movlw     0xFF
740     movwf     OffsetF
741     bcf       ESTADO_BUSQUEDA,SCROLL_F       ; Desactivamos el SCROLL de nombre de
742                                              ; fichero
743     movf      N_Fichero,W                    ; Buscamos la entrada siguiente de
744     goto      ComienzaBusqueda               ; fichero
745   ;*******************************************************************************

747   ;************************ BOTON 5 - Decrementar Fichero ************************
748    DecFich
749     bsf       ESTADO_BUSQUEDA,BUSCAR_FICH    ; Indicamos que buscamos un Fichero
750     movf      N_Fichero,F                    ; Comprobamos si hemos buscado ya
751     btfsc     STATUS,Z                       ; algun fichero en el directorio
752                                              ; actual
753     goto      Escanea                        ; NO hemos buscado ficheros, no
754                                              ; hacemos nada

756     movf      N_Fichero,W                    ; Comprobamos si estamos en el primer
757     xorlw     d'1'                           ; fichero del directorio
758     btfsc     STATUS,Z
759     goto      Escanea                        ; Estamos en el primero. No hay
760                                              ; anterior

762     decf      N_Fichero,F                    ; Indicamos que buscamos el fichero
763     movlw     0xFF                           ; anterior
764     movwf     OffsetF
765     bcf       ESTADO_BUSQUEDA,SCROLL_F       ; Indicamos que buscamos un fichero
766     movf      N_Fichero,W                    ; Buscamos el fichero
767     goto      ComienzaBusqueda
768   ;*******************************************************************************


771   ;***************************** PARTE COMUN - Busqueda **************************
772   ComienzaBusqueda
773     movwf     EntBuscada_Lo                  ; Indicamos el numero de entrada
774     clrf      EntBuscada_Hi                  ; buscada
775     call      Buscar_Entrada                 ; Buscamos la entrada
776     btfss     ESTADO_BUSQUEDA,ULT_ENTRADA    ; Comprobamos si era la ultima entrada
777                                              ; valida
778     goto      PonNombre                      ; No era la ultima entrada

780   ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

782   UltimaEntrada                              ; Era la Ultima entrada
783     btfsc     ESTADO_BUSQUEDA,BUSCAR_FICH    ; Comprobamos si buscabamos fichero o
784                                              ; directorio
785     goto      UltimoFichero

787   ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

789   UltimoDirectorio                           ; Era la ultima entrada de Directorio
790     bsf       PCLATH,3                       ; Indicamos por pantalla que ya no hay
791     movlw     d'3'                           ; mas directorios
792     call      LCD_Linea1
793     movlw     b'01000001'
794     movwf     Mensaje
795     call      LCD_PonMensaje
796     bcf       PCLATH,3

798     movlw     d'50'                          ; Esperamos 1 segundo
799     call      Espera_L

801     decf      N_Directorio,F                 ; Volvemos al directorio anterior (el
```

```
802     movf      N_Directorio,W                ; ultimo valido)
803     goto      ComienzaBusqueda

805 ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

807 UltimoFichero                               ; Era la ultima entrada de fichero
808     bsf       PCLATH,3                      ; Indicamos por pantalla que ya no hay
809     movlw     d'3'                          ; mas ficheros
810     call      LCD_Linea2
811     movlw     b'01000001'
812     movwf     Mensaje
813     call      LCD_PonMensaje
814     bcf       PCLATH,3

816     movlw     d'50'                         ; Esperamos 1 segundo
817     call      Espera_L

819     decf      N_Fichero,F                   ; Volvemos al fichero anterior (el
820     movf      N_Fichero,W                   ; ultimo valido)
821     goto      ComienzaBusqueda
822 ;*******************************************************************************

825 ;************ PARTE COMUN - Actualizacion del Nombre de la Entrada *************
826 PonNombre                                   ; Busqueda exitosa, ponemos el nombre
827                                             ; de la entrada en la pantalla
828     btfsc     ESTADO_BUSQUEDA,BUSCAR_FICH   ; Comprobamos si buscabamos fichero o
829                                             ; directorio
830     goto      NombreFichero

833 NombreDirectorio                            ; Buscabamos Directorio
834     bsf       PCLATH,3                      ; Situamos el cursor en la Linea 1
835     movlw     d'3'
836     call      LCD_Linea1
837     bcf       PCLATH,3

839     call      BUS_ConfEsc                   ; Configuramos el BUS para escribir en
840                                             ; el LATCH de MEMORIA
841     movlw     (NOMBRE_DIRECTORIO>>8)        ; Parte alta de la direccion del
842                                             ; nombre de directorio
843     goto      SiguePonNombre

845 ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

847 NombreFichero                               ; Buscabamos Fichero
848     bsf       PCLATH,3                      ; Situamos el cursor en la Linea 2
849     movlw     d'3'
850     call      LCD_Linea2
851     bcf       PCLATH,3

853     call      BUS_ConfEsc                   ; Configuramos el BUS para escribir en
854                                             ; el LATCH de MEMORIA
855     movlw     (NOMBRE_FICHERO>>8)           ; Parte alta de la direccion del
856                                             ; nombre de fichero
857 ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

859 SiguePonNombre
860     movwf     DirMem_H
861     movwf     DIR_SRAM_H                    ; Ponemos la parte Alta en el BUS
862     clrw                                    ; La parte Baja es 0
863     movwf     DirMem_L
864     movwf     DIR_SRAM_L

866     bcf       INTCON,GIE
867     movlw     LATCH_MEM_L                   ; Metemos la direccion en el LATCH de
868     movwf     CONTROL1                      ; memoria
869     nop
870     movlw     OCIOSO
871     movwf     CONTROL1
872     bsf       INTCON,GIE

874     movlw     d'17'                         ; Maximo numero de caracteres que
```

```
875     movwf     Contador1                       ; podemos escribir en la pantalla

877  BucleNombre                                  ; Escribimos el Nombre de la entrada
878     call      BUS_ConfLec                     ; Configuramos el BUS para leer de la
879                                               ; MEMORIA

881     bcf       INTCON,GIE
882     movlw     OE                              ; Leemos la memoria SRam (OE)
883     movwf     CONTROL1
884     movf      DATOS_BAJOS,W
885     movwf     Temp0                           ; Almacenamos el dato leido en Temp0
886     movlw     OCIOSO
887     movwf     CONTROL1
888     bsf       INTCON,GIE

890     movf      Temp0,W                         ; Comprobamos si el dato leido es el
891     movf      Temp0,F                         ; numero 0 correspondiente a "Fin de
892     btfsc     STATUS,Z                        ; Nombre"
893     goto      Seguir                          ; Se ha acabado el nombre

895     bsf       PCLATH,3                        ; Todavia queda nombre, enviamos el
896     call      LCD_EnviaDato                   ; caracter leido a la pantalla
897     bcf       PCLATH,3
898     bcf       INTCON,GIE
899     bsf       CONTROL2,INC_ADDR               ; Incrementamos la direccion de origen
900     nop                                       ; para leer el siguiente caracter
901     bcf       CONTROL2,INC_ADDR
902     incf      DirMem_L,F
903     btfsc     STATUS,Z
904     incf      DirMem_H,F
905     bsf       INTCON,GIE
906     decfsz    Contador1,F                     ; Decrementamos el contador de
907                                               ; posiciones libres de pantalla
908     goto      BucleNombre

910  ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

912  ; LLegados a este punto, el nombre no cabe en la linea, activamos el Scroll

914     btfsc     ESTADO_BUSQUEDA,BUSCAR_FICH     ; Comprobamos que SCROLL hay que
915                                               ; activar
916     goto      ScrollFichero

918  ScrollDirectorio                             ; Activamos el SCROLL de Directorio
919     bsf       ESTADO_BUSQUEDA,SCROLL_D
920     bsf       STATUS,RP1
921     movlw     d'255'
922     movwf     IntScrollDir
923     bcf       STATUS,RP1
924     goto      Seguir

926  ScrollFichero                                ; Activamos el SCROLL de Fichero
927     bsf       ESTADO_BUSQUEDA,SCROLL_F
928     bsf       STATUS,RP1
929     movlw     d'128'
930     movwf     IntScrollFich
931     bcf       STATUS,RP1

933  ;****   ****   ****   ****   ****   ****   ****   ****   ****   ****   ****   **

935  Seguir                                       ; Se acabo el nombre de la entrada,
936  PonEspacios                                  ; completamos la linea con blancos.
937     movf      Contador1,F                     ; Comprobamos si queda sitio en la
938     btfsc     STATUS,Z                        ; linea
939     goto      Otro                            ; Linea completa, hemos acabado

941     bsf       PCLATH,3                        ; Pintamos el blanco
942     movlw     '␣'
943     call      LCD_EnviaDato
944     bcf       PCLATH,3

946     decf      Contador1,F                     ; Decrementamos el numero de
947                                               ; posiciones libres
```

```
948     goto      PonEspacios

950  Otro
951     goto      Escanea
952  ;*****************************************************************************

954  ;************************* BOTON 7 - Reproducir Fichero ************************
955  Reproducir
956     clrf      STATUS                ; Banco 0

958  ; Metemos en los PUNTEROS de LECTURA y ESCRITURA la direccion de comienzo del
959  ; BUFFER SRAM

961     movlw     BUFFER_REPRO>>8       ; Parte alta
962     movwf     PuntBufferEsc_H
963     movwf     PuntBufferLec_H
964     movlw     BUFFER_REPRO          ; Parte baja
965     movwf     PuntBufferEsc_L
966     movwf     PuntBufferLec_L

968  ;    bcf       STATUS,RP1            ; Banco 0

970  ; Cargamos el numero de cluster en el que empieza la cancion en los registros de
971  ; Cluster_Actual.


974     movf      Cluster_Ent0,W
975     movwf     Cluster_Actual0
976     movf      Cluster_Ent1,W
977     movwf     Cluster_Actual1
978     movf      Cluster_Ent2,W
979     movwf     Cluster_Actual2
980     movf      Cluster_Ent3,W
981     movwf     Cluster_Actual3

983     bcf       ESTADO_BUSQUEDA,ULT_CLUSTER   ; Indicamos que NO es el ULTIMO
984                                             ; CLUSTER de la cadena
985     clrf      SecPendientes         ; No tenmos sectores pendientes

987  ; A continuacion vamos a precargar el BUFFER INTERMEDIO SRAM, de forma que
988  ; tengamos ya datos disponibles para decodificar

990  CargarBuffer

992  ; Para precargar el BUFFER INTERMEDIO, primero cargamos un bloque para evitar
993  ; que los punteros ded lectura y escritura valgan lo mismo

995     call      CargarBloque

997  ; Ahora continuamos cargando bloques en el Buffer hasta que el puntero de
998  ; escritura valga lo mismo que el de lectura (de una vuelta entera al Buffer
999  ; circular).

1001 ContinuaPrecarga
1002    movf      PuntBufferLec_H,W
1003    andlw     b'11111100'
1004    xorwf     PuntBufferEsc_H,W
1005    btfsc     STATUS,Z
1006    goto      BufferPrecargado
1007    call      CargarBloque
1008    goto      ContinuaPrecarga

1010 ; El Buffer ya esta precargado, entramos pues en el bloque de reproduccion
1011 ; principal.
1012 ; Tenemos que activar el proceso de lectura (INTERRUPCIONES) y secuenciar la
1013 ; carga del Buffer, los Scrolls de pantalla y las pulsaciones de teclas

1015 BufferPrecargado

1017 MensReproduccion
1018    bsf       PCLATH,3
1019    clrw                            ; Ponemos el cursor al comienzo de la Linea 4
1020    call      LCD_Linea4
```

```
1022      movlw     b'10000010'          ; Indicamos que mensaje vamos a pintar
1023      movwf     Mensaje

1025      call      LCD_PonMensaje
1026      bcf       PCLATH,3

1028      bsf       STATUS,RP1           ; Banco 2
1029      movlw     b'00110000'          ; Ponemos a 0 el contador de tiempo
1030      movwf     DMinutos             ; transcurrido
1031      movwf     UMinutos
1032      movwf     DSegundos
1033      movwf     USegundos
1034      movlw     d'40'
1035      movwf     IntContTiempo
1036      bcf       INTerrupcion,TIEM
1037      movlw     d'10'
1038      movwf     IntBitSample
1039      bcf       STATUS,RP1           ; Banco 0
1040      bsf       PCLATH,3
1041      call      MuestraTiempo
1042      bcf       PCLATH,3

1044  Reproduccion
1045  ; Aqui comienza el proceso de Reproduccion, lo primero que hacemos es llamar a
1046  ; la rutina que se encarga de llenar el Buffer del decodificador de MP3. esto lo
1047  ; hacemos cada 25 ms de acuerdo con la activacion del correspondiente BIT del
1048  ; registro de control de interrupciones.

1050      bcf       STATUS,RP1           ; Banco 0

1052      bsf       ESTADO_BUSQUEDA,PLAY ; Indicamos que estamos reproduciendo un
1053                                     ; fichero

1055      bcf       ESTADO_BUSQUEDA,BIToSAMPLE
1056      bcf       INTerrupcion,BoS
1057      bcf       INTerrupcion,INFO
1058  ;    bcf       INTerrupcion,ZEROS

1060  Cargar
1061  ;    call      CargarBufferMP3
1062      bsf       PCLATH,3
1063      call      ActualizarTiempo
1064      call      Scroll
1065      btfsc     INTerrupcion,INFO
1066      call      MostrarBSR
1067      bcf       PCLATH,3

1069      call      Escanea_Botones
1070      movwf     RespaldoW
1071      btfss     RespaldoW,7
1072      goto      FinReproduccion
1073      btfss     RespaldoW,2
1074      goto      SubVolumen
1075      btfss     RespaldoW,6
1076      goto      BajVolumen

1078      btfsc     INTerrupcion,INFO    ; Extraemos la Informacion del fichero MP3,
1079      goto      Cargado              ; INFO debe estar a 0 y BoS a 1
1080      btfss     INTerrupcion,BoS
1081      goto      Cargado
1082      bsf       PCLATH,3
1083      call      Extrae_Info          ; Extraemos la Informacion
1084      bcf       PCLATH,3
1085      bsf       INTerrupcion,INFO    ; Indicamos que ya hemos extraido la
1086                                     ; informacion

1088  ; A continuacion comprobamos si tenemos espacio en el Buffer intermedio para
1089  ; cargar un nuevo bloque.

1091  Cargado
1092      movf      PuntBufferLec_H,W
1093      andlw     b'11111100'
```

```
1094      xorwf     PuntBufferEsc_H,W
1095      btfsc     STATUS,Z
1096      goto      Cargar

1098      btfsc     ESTADO_BUSQUEDA,ULT_CLUSTER
1099      goto      CargaCeros

1101   CargaBloque
1102      call      CargarBloque
1103      goto      Cargar

1105   CargaCeros
1106      call      CargarCeros

1108   EsperaFin
1109   ;    call      CargarBufferMP3
1110      movf      PuntBufferLec_H,W
1111      andlw     b'11111100'
1112      xorwf     PuntBufferEsc_H,W
1113      btfsc     STATUS,Z
1114      goto      FinReproduccion       ; No quedan clusters, paramos la reproduccion
1115      goto      EsperaFin

1117   ; Se ha acabado la cancion, desactivamos las interrupciones y volvemos al bucle
1118   ; inicial

1120   FinReproduccion
1121      bcf       ESTADO_BUSQUEDA,PLAY
1122      bsf       PCLATH,3
1123      call      Reset_S_VS1001



1128      clrw                            ; Ponemos el cursor al comienzo de la Linea 1
1129      call      LCD_Linea4

1131      movlw     b'10000001'           ; Indicamos que mensaje vamos a pintar
1132      movwf     Mensaje
1133      call      LCD_PonMensaje
1134      call      LCD_LimpiaLinea3
1135      bcf       PCLATH,3

1137      goto      Escanea

1139   SubVolumen
1140      movf      Volumen,F
1141      btfsc     STATUS,Z
1142      goto      Cargar
1143      movlw     d'4'                  ; Cantidad en la que se altera el volumen
1144      subwf     Volumen,F
1145      goto      SetVolumen

1147   BajVolumen
1148      movlw     d'140'
1149      xorwf     Volumen,W
1150      btfsc     STATUS,Z
1151      goto      Cargar
1152      movlw     d'4'
1153      addwf     Volumen,F

1155   SetVolumen
1156      bsf       PCLATH,3
1157      movlw     d'2'
1158      movwf     Cifras
1159      movf      Volumen,W
1160      movwf     Dato0
1161      bcf       STATUS,C
1162      rrf       Dato0,F
1163      bcf       STATUS,C
1164      rrf       Dato0,W
1165      sublw     d'35'
1166      movwf     Dato0
```

```
1167      call      Bin2Dec
1168      clrw
1169      call      LCD_Linea3
1170      movlw     b'01100000'
1171      movwf     Mensaje
1172      call      LCD_PonMensaje
1173      movlw     d'9'
1174      call      LCD_Linea3
1175      bsf       STATUS,RP0
1176      movf      Digitos,W
1177      bcf       STATUS,RP0
1178      call      LCD_EnviaDato
1179      bsf       STATUS,RP0
1180      movf      (Digitos+1),W
1181      bcf       STATUS,RP0
1182      call      LCD_EnviaDato
1183      bcf       VSCONTROL,XCS          ; Habilitimas el VS1001K
1184      movlw     SCI_WRITE              ; Comando -> ESCRIBIR
1185      call      SCI_Out
1186      movlw     VS_VOL                 ; Registro -> VOLUMEN
1187      call      SCI_Out
1188      bsf       STATUS,RP1             ; Banco 2
1189      movf      Volumen,W
1190      bcf       STATUS,RP1             ; Banco 0
1191      call      SCI_Out
1192      bsf       STATUS,RP1             ; Banco 2
1193      movf      Volumen,W
1194      bcf       STATUS,RP1             ; Banco 0
1195      call      SCI_Out
1196      bsf       VSCONTROL,XCS          ; Deshabilitamos el VS1001K
1197      bcf       PCLATH,3
1198      movlw     d'1'                   ; Esperamos 1ms
1199      call      Espera_C
1200      goto      Cargar

1202  SinFin
1203      goto      SinFin

1205  ;*****************************************************************************
1206  ;*****************************************************************************

1208  ;*****************************************************************************
1209  ;*                                                                           *
1210  ;*                                  CargarCeros                              *
1211  ;*                                                                           *
1212  ;*****************************************************************************
1213  ;*                                                                           *
1214  ;* Escribe un total de 2K -> 2048 ceros en la zona de Buffer de memoria      *
1215  ;* intermedio.                                                               *
1216  ;*                                                                           *
1217  ;*****************************************************************************

1219  CargarCeros

1221      bcf       STATUS,RP0
1222      bcf       STATUS,RP1

1224      movlw     d'4'
1225      movwf     NumSectTR
1226      call      BUS_ConfEsc
1227      bcf       INTCON,GIE
1228      movf      PuntBufferEsc_H,W
1229      movwf     DirMem_H
1230      movwf     DIR_SRAM_H
1231      movf      PuntBufferEsc_L,W
1232      movwf     DirMem_L
1233      movwf     DIR_SRAM_L
1234      movlw     LATCH_MEM_L
1235      movwf     CONTROL1
1236      nop
1237      movlw     OCIOSO
1238      movwf     CONTROL1
1239      bsf       INTCON,GIE
```

```
1241  Sector_Z
1242     movlw     d'0'
1243     movwf     NumWords

1245  Word_Z
1246     clrw
1247     movwf     DATOS_BAJOS
1248     movwf     DATOS_ALTOS
1249     bcf       INTCON,GIE
1250     bcf       CONTROL2,WE
1251     nop
1252     bsf       CONTROL2,WE
1253  ;   movlw     OCIOSO
1254  ;   movwf     CONTROL1
1255     bsf       CONTROL2,INC_ADDR
1256     nop
1257     bcf       CONTROL2,INC_ADDR
1258     incf      DirMem_L,F
1259     btfsc     STATUS,Z
1260     incf      DirMem_H,F
1261     bsf       INTCON,GIE
1262     decfsz    NumWords,F

1264     goto      Word_Z

1266     decfsz    NumSectTR,F

1268     goto      Sector_Z

1270     bcf       INTCON,GIE
1271     movlw     0x04
1272     addwf     PuntBufferEsc_H,F
1273     btfss     PuntBufferEsc_H,7
1274     goto      ZerosCargados

1276     movlw     BUFFER_REPRO>>8
1277     movwf     PuntBufferEsc_H
1278     movlw     BUFFER_REPRO
1279     movwf     PuntBufferEsc_L

1281  ZerosCargados
1282     bsf       INTCON,GIE
1283     return

1285  ;*****************************************************************************
1286  ;*                                                                           *
1287  ;*                                CargarBloque                               *
1288  ;*                                                                           *
1289  ;*****************************************************************************
1290  ;*                                                                           *
1291  ;* Obtiene un total de 2K (4 sectores) de DATOS de la cancion del disco duro, *
1292  ;* y los almacena en el BUFFER INTERMEDIO situado en memoria SRAM.           *
1293  ;*                                                                           *
1294  ;*****************************************************************************

1296  CargarBloque

1298     clrf      STATUS                ; Banco 0
1299     movf      SecPendientes,F       ; Comprobamos si tenemos sectores pendientes de
1300     btfss     STATUS,Z              ; transferir
1301     goto      ClusterCargado

1303  CargarCluster
1304     call      Calcula_SectorLBA     ; Calculamos el numero de Sector a leer
1305     movf      SecCluster,W          ; Ponemos el numero de sectores por cluster en
1306     movwf     SecPendientes         ; el contador de sectores pendientes
1307     call      IDE_LeerSectores      ; Damos la orden al HD de leer un cluster
1308                                     ; entero

1310  ClusterCargado
1311     movf      PuntBufferEsc_H,W     ; Cargamos la direccion de destino en la SRAM
1312     movwf     SRam_Hi
```

```
1313      movf      PuntBufferEsc_L,W
1314      movwf     SRam_Lo
1315      movlw     d'4'
1316      call      IDE_TransSectaSRam    ; Transferimos 4 sectores a la SRAM -> 2K

1318      bcf       INTCON,GIE            ; Deshabilitamos las interrupciones. Como se
1319                                      ; comparan los punteros de lectura y escritura,
1320                                      ; debemos asegurarnos de que en todo momento
1321                                      ; mantienen un valor correcto.
1322      movlw     0x04                  ; Incrementamos el puntero de escritura en
1323      addwf     PuntBufferEsc_H,F     ; 1Kx16 = 2Kx8

1325      btfss     PuntBufferEsc_H,7     ; El Buffer SRAM es circular. ¨Hemos llegado al
1326                                      ; final?

1328      goto      PunteroEscIncrementado

1330  PunteroEscAlPrincipio
1331      movlw     BUFFER_REPRO>>8       ; Parte alta
1332      movwf     PuntBufferEsc_H
1333      movlw     BUFFER_REPRO          ; Parte baja. Estas dos instrucciones SE PUEDEN
1334      movwf     PuntBufferEsc_L       ; ELIMINAR si la direccion de comienzo del
1335                                      ; Buffer es 0xXX00
1336  PunteroEscIncrementado
1337      bsf       INTCON,GIE            ; Rehabilitamos las interrupciones
1338      movlw     d'4'
1339      subwf     SecPendientes,F       ; Decrementamos en cuatro el numero de sectores
1340                                      ; pendientes de transferir del cluster
1341      btfss     STATUS,Z              ; Comprobamos si hemos transferido ya todos
1342                                      ; sectores
1343      goto      Salida                ; Bloque transferido, salimos

1345  ; Hemos transferido el bloque entero de 2K, pero con ello tambien hemos
1346  ; terminado de transferir lo que quedaba de CLUSTER, por lo que antes de salir,
1347  ; obtenemos la direccion del siguiente CLUSTER.

1349  CargarSiguienteCluster
1350      call      LeerSigCluster

1352  Salida
1353      return

1355  ;*****************************************************************************
1356  ;*                                                                           *
1357  ;*                              CargarBufferMP3                              *
1358  ;*                                                                           *
1359  ;*****************************************************************************
1360  ;*                                                                           *
1361  ;* Recarga el Buffer del CHIP decodificador de MP3 hasta que este esta lleno,  *
1362  ;* usando para ello y como origen de los datos el BUFFER INTERMEDIO situado   *
1363  ;* en memoria SRAM.                                                          *
1364  ;*                                                                           *
1365  ;*****************************************************************************

1367  CargarBufferMP3

1369  ; Comprobamos si hay sitio en el Buffer Intermedio SRAM; buscamos un espacio
1370  ; libre de al menos 2K, ya que en el peor caso (Buffer del decodificador vacio),
1371  ; es lo que va a incrementarse el puntero de lectura.


1374      clrf      STATUS                ; Banco 0
1375      movf      PuntBufferLec_H,W
1376      andlw     b'11111100'
1377      addlw     d'4'
1378      xorwf     PuntBufferEsc_H,W
1379      btfsc     STATUS,Z
1380      goto      BufferIntVacio

1382  Trans32Bytes

1384  ; Comprobamos si hay sitio en el Buffer del Decodificador de MP3
```

```
1386      bcf       STATUS,RP1            ; Banco 0
1387      btfss     VSCONTROL,DREQ
1388      goto      BufferDecLLeno

1390      bsf       STATUS,RP1            ; Banco 2
1391      movlw     d'16'
1392      movwf     Cont32Bytes           ; Reseteamos el contador de Bytes a 16,
1393                                      ; mandamos los datos en grupitos de 2 Bytes
1394                                      ; 2*16 = 32
1395      call      BUS_ConfEsc           ; Configuramos el BUS de DATOS para escribir en
1396                                      ; el.

1398      bcf       STATUS,RP1            ; Banco 0
1399      movf      PuntBufferLec_H,W
1400      movwf     DIR_SRAM_H
1401      movwf     DirMem_H
1402      movf      PuntBufferLec_L,W
1403      movwf     DIR_SRAM_L
1404      movwf     DirMem_L

1406      bcf       INTCON,GIE
1407      movlw     LATCH_MEM_L           ; Activamos LATCH_MEM_L
1408      movwf     CONTROL1
1409      nop                             ; Esperamos
1410      movlw     OCIOSO                ; Desactivamos LATCH_MEM_L
1411      movwf     CONTROL1
1412      bsf       INTCON,GIE

1414  Trans2Bytes
1415      call      BUS_ConfLec           ; Configuramos el BUS de DATOS para leer de la
1416                                      ; memoria
1417      bsf       PCLATH,3

1419      bcf       INTCON,GIE
1420      movlw     OE                    ; Leemos la memoria SRam (OE)
1421      movwf     CONTROL1
1422      movf      DATOS_BAJOS,W         ; Mandamos el Byte BAJO al decodificador
1423      call      SDI_Out
1424      movf      DATOS_ALTOS,W         ; Mandamos el Byte ALTO al decodificador
1425      call      SDI_Out
1426      movlw     OCIOSO                ; Deactivamos OE
1427      movwf     CONTROL1
1428      bsf       CONTROL2,INC_ADDR     ; Incrementamos la direccion de origen de la
1429      nop                             ; SRAM
1430      bcf       CONTROL2,INC_ADDR
1431      incf      DirMem_L,F
1432      btfsc     STATUS,Z
1433      incf      DirMem_H,F
1434      bsf       INTCON,GIE
1435      bcf       PCLATH,3

1437      incf      PuntBufferLec_L,F     ; Incrementamos el puntero de lectura del
1438                                      ; Buffer en 1
1439      btfsc     STATUS,Z
1440      incf      PuntBufferLec_H,F     ; Incrementamos el Puntero de lectura del
1441                                      ; Buffer en 1
1442      btfss     PuntBufferLec_H,7     ; Comprobamos si hemos llegado al final del
1443                                      ; Buffer
1444      goto      PunteroLecIncrementado

1446  PunteroLecAlPrincipio               ; Volvemos al principio (Buffer circular)
1447      movlw     BUFFER_REPRO>>8       ; Parte alta
1448      movwf     PuntBufferLec_H
1449      movlw     BUFFER_REPRO          ; Parte baja. Estas dos instrucciones NO SON
1450      movwf     PuntBufferLec_L       ; NECESARIAS si el Buffer comienza en la
1451                                      ; direccion 0xXX00

1453  PunteroLecIncrementado              ; Ahora comprobamos si se ha liberado ya un
1454                                      ; bloque
1455      bsf       STATUS,RP1            ; Banco 2
1456      decfsz    Cont32Bytes,F
1457      goto      Trans2Bytes
1458      goto      Trans32Bytes
```

```
1460  BufferIntVacio
1461  BufferDecLLeno
1462     clrf     STATUS                ; Banco 0
1463     return

1466  ;*****************************************************************************
1467  ;*                                                                           *
1468  ;*                                  BUS_Conf...                              *
1469  ;*                                                                           *
1470  ;*****************************************************************************
1471  ;*                                                                           *
1472  ;* Configura el PIC para leer o escribir en el BUS de datos                  *
1473  ;*                                                                           *
1474  ;*****************************************************************************

1476  BUS_ConfLec

1478     movlw    0xFF                  ; Cargamos 0xFF en W para configurar luego los
1479                                    ; puertos como de entrada
1480     goto     LEComun               ; Saltamos a la zona comun

1482  BUS_ConfEsc                       ; Configuracion para escribir
1483     clrw                           ; Cargamos 0x00 en W para configurar luego los
1484                                    ; puertos como de salida
1485  LEComun                           ; Zona de configuracion comun
1486     bcf      STATUS,RP1
1487     bsf      STATUS,RP0            ; Banco 1
1488     movwf    TRISB                 ; Configuramos DATA0-DATA7 como de entrada o
1489                                    ; salida
1490     movwf    TRISD                 ; Configuramos DATA8-DATA15 como de entrada o
1491                                    ; salida
1492     bcf      STATUS,RP0            ; Banco 0
1493     return

1495  ;*****************************************************************************
1496  ;*****************************************************************************
1497  ;**                                                                         **
1498  ;**                              Bucles de retardo                          **
1499  ;**                                                                         **
1500  ;*****************************************************************************
1501  ;*****************************************************************************

1503  ;*****************************************************************************
1504  ;*                                                                           *
1505  ;*                                    Espera_L                               *
1506  ;*                                                                           *
1507  ;*****************************************************************************
1508  ;*                                                                           *
1509  ;* Bucle de retardo que realiza una espera de multiplos de 20ms segun lo que  *
1510  ;* reciba por W.                                                             *
1511  ;*                                                                           *
1512  ;*****************************************************************************

1514  Espera_L:
1515     movwf    Tempo1
1516  Espera_L_Ext                      ; Bucle de 20 ms
1517     movlw    d'130'
1518     movwf    Tempo2
1519     movlw    d'0'
1520     movlw    Tempo3
1521  Espera_L_Int
1522     decfsz   Tempo3,F
1523     goto     Espera_L_Int          ; 768

1525     decfsz   Tempo2,F
1526     goto     Espera_L_Int          ; (768 + 3) * 13 = 100230 veces -- 10023 *
1527                                    ; * 0.2us = 0,020046s
1528     decfsz   Tempo1,F              ; Repetimos la espera de 20 ms Temp1 veces
1529     goto     Espera_L_Ext
1530     return
```

```
1532 ;*****************************************************************************
1533 ;*                                                                           *
1534 ;*                                  Espera_C                                 *
1535 ;*                                                                           *
1536 ;*****************************************************************************
1537 ;*                                                                           *
1538 ;* Bucle de retardo que realiza una espera de multiplos de 1ms segun lo que  *
1539 ;* reciba por W.                                                             *
1540 ;*                                                                           *
1541 ;*****************************************************************************

1543 Espera_C:
1544    movwf    Tempo1
1545 Espera_C_Ext                      ; Bucle de 1 ms
1546    movlw    d'4'
1547    movwf    Tempo2
1548    movlw    d'0'
1549    movlw    Tempo3
1550 Espera_C_Int
1551    nop
1552    nop
1553    decfsz   Tempo3,F
1554    goto     Espera_C_Int          ; 1280

1556    decfsz   Tempo2,F
1557    goto     Espera_C_Int          ; (1280 + 3) * 4 = 5132 veces -- 5132 * 0.2us =
1558                                   ; = 0,0010264s
1559    decfsz   Tempo1,F              ; Repetimos la espera de 1 ms Temp1 veces
1560    goto     Espera_C_Ext
1561    return

1563    VARIABLE       FileStartC1 = $

1565     INCLUDE I_FAT32.asm
1566     INCLUDE I_IDE.asm
1567     INCLUDE I_LCD.asm
1568     INCLUDE Mensajes.asm
1569     INCLUDE I_VS1001.asm
1570     INCLUDE I_Boton.asm

1572    END
```

A.3. Interfaz IDE

Esta parte del código tiene el objetivo de implementar la interfaz con el disco duro. Contiene todas las funciones necesarias para inicializarlo, leer y escribir sectores y transferir el contenido de los mismos al PIC o al dispositivo IDE.

Todos los comandos que se implementan pertenecen al estandar AT Attachment *ATA1*.

Esta parte del codigo se encuentra localizada en la **Página 0**.

```
1   ;*****************************************************************************
2   ;*****************************************************************************
3   ;*****************************************************************************
4   ;***                                                                       ***
5   ;**                                I_IDE.asm                                **
6   ;**                                                                         **
7   ;** Contiene todas las funciones necesarias para controlar el dispositivo IDE **
8   ;** conectado al PIC: resetear, controlar su estado, ponerlo en Stand By,     **
9   ;** transferir sectores a la memoria SRAM ...                                **
10  ;**                                                                         **
11  ;***                                                                       ***
12  ;*****************************************************************************
13  ;*****************************************************************************
14  ;*****************************************************************************

16  ;*****************************************************************************
17  ;*****************************************************************************
18  ;**                                                                         **
19  ;**                        Declaracion de Constantes                        **
20  ;**                                                                         **
21  ;*****************************************************************************
22  ;*****************************************************************************

24  ;*****************************************************************************
25  ;*                                                                           *
26  ;*                              Palabras de Control                          *
27  ;*                                                                           *
28  ;* Para acceder a los distintos registros del dispositivo IDE, ya sea para   *
29  ;* leerlos o para escribirlos, tenemos que poner en los pines de control     *
30  ;* (DA0 - DA2, #CS0, #CS1) los siguientes valores:                           *
31  ;*                                                                           *
32  ;*****************************************************************************

34  ;                                                BIT4 BIT3  BIT2 BIT1 BIT0

36  ;                                                #CS1 #CS0  DA2  DA1  DA0
37  IDE_REG_ALTSTATUS       EQU   0x0E  ; XXX01110    A    N    1    1    0
38  IDE_REG_CTRL            EQU   0x0E  ; XXX01110    A    N    1    1    0
39  IDE_REG_DATA            EQU   0x10  ; XXX10000    N    A    0    0    0
40  IDE_REG_ERROR           EQU   0x11  ; XXX10001    N    A    0    0    1
41  IDE_REG_FEATURES        EQU   0x11  ; XXX10001    N    A    0    0    1
42  IDE_REG_SECTORCOUNT     EQU   0x12  ; XXX10010    N    A    0    1    0
43  IDE_REG_SEC0            EQU   0x13  ; XXX10011    N    A    0    1    1
44  IDE_REG_SEC1            EQU   0x14  ; XXX10100    N    A    1    0    0
45  IDE_REG_SEC2            EQU   0x15  ; XXX10101    N    A    1    0    1
46  IDE_REG_DEVHEAD         EQU   0x16  ; XXX10110    N    A    1    1    0
47  IDE_REG_DEVICE          EQU   0x16  ; XXX10110    N    A    1    1    0
48  IDE_REG_STATUS          EQU   0x17  ; XXX10111    N    A    1    1    1
49  IDE_REG_CMD             EQU   0x17  ; XXX01110    N    A    1    1    1

51  ;*****************************************************************************
52  ;*                                                                           *
53  ;*                                Codigos de comado                          *
54  ;*                                                                           *
55  ;* Lista de comados que podemos ejecutar sobre el dispositivo IDE (ATA6),    *
56  ;* junto con su correspondiente codigo, que sera el que tengamos que         *
57  ;* introducir en el registro que para tal efecto provee el dispositivo:      *
58  ;* IDE_REG_CMD (17h)                                                         *
59  ;* Dada la gran variedad de dispositivos IDE diferentes que puede haber, NO  *
```

```
60  ;* TODOS estos comandos pueden ser ejecutados por un mismo dispositivo, de     *
61  ;* hecho, hay comandos cuya ejecucion esta prohibida en funcion del            *
62  ;* dispositivo que sea. Es altamente recomendable leer el datasheet del        *
63  ;* dispositivo y el de ATA6 antes de usarlos.                                  *
64  ;*                                                                             *
65  ;*******************************************************************************

67  CMD_Nop                                EQU   0x00  ;  Nop
68  CMD_Cfa_Request_Extended_Error         EQU   0x03  ;  Cfa Request Extended Error
69  CMD_Device_Reset                       EQU   0x08  ;  Device Reset
70  CMD_Read_Sector                        EQU   0x20  ;  Read Sector(S)
71  CMD_Read_Sector_Ext                    EQU   0x24  ;  Read Sector(S) Ext
72  CMD_Read_Dma_Ext                       EQU   0x25  ;  Read Dma Ext
73  CMD_Read_Dma_Queued_Ext                EQU   0x26  ;  Read Dma Queued Ext
74  CMD_Read_Native_Max_Address_Ext        EQU   0x27  ;  Read Native Max Address
75                                                     ;  Ext
76  CMD_Read_Multiple_Ext                  EQU   0x29  ;  Read Multiple Ext
77  CMD_Read_Log_Ext                       EQU   0x2F  ;  Read Log Ext
78  CMD_Write_Sector                       EQU   0x30  ;  Write Sector(S)
79  CMD_Write_Sector_Ext                   EQU   0x34  ;  Write Sector(S) Ext
80  CMD_Write_Dma_Ext                      EQU   0x35  ;  Write Dma Ext
81  CMD_Write_Dma_Queued_Ext               EQU   0x36  ;  Write Dma Queued Ext
82  CMD_Set_Max_Address_Ext                EQU   0x37  ;  Set Max Address Ext
83  CMD_Cfa_Write_Sectors_WO_Erase         EQU   0x38  ;  Cfa Write Sectors W/Out
84                                                     ; Erase
85  CMD_Write_Multiple_Ext                 EQU   0x39  ;  Write Multiple Ext
86  CMD_Write_Log_Ext                      EQU   0x3F  ;  Write Log Ext
87  CMD_Read_Verify_Sector                 EQU   0x40  ;  Read Verify Sector(S)
88  CMD_Read_Verify_Sector_Ext             EQU   0x42  ;  Read Verify Sector(S) Ext
89  CMD_Seek                               EQU   0x70  ;  Seek
90  CMD_Cfa_Translate_Sector               EQU   0x87  ;  Cfa Translate Sector
91  CMD_Execute_Device_Diagnostic          EQU   0x90  ;  Execute Device Diagnostic
92  CMD_Download_Microcode                 EQU   0x92  ;  Download Microcode
93  CMD_Acket                              EQU   0xA0  ;  Acket
94  CMD_Identify_Packet_Device             EQU   0xA1  ;  Identify Packet Device
95  CMD_Ervice                             EQU   0xA2  ;  Ervice
96  CMD_Smart_Disable_Operations           EQU   0xB0  ;  Smart Disable Operations
97  CMD_Smart_EnableDisable_Autosave       EQU   0xB0  ;  Smart Enable/Disable
98                                                     ;  Autosave
99  CMD_Smart_Enable_Operations            EQU   0xB0  ;  Smart Enable Operations
100 CMD_Smart_Execute_Off_Line             EQU   0xB0  ;  Smart Execute Off_Line
101 CMD_Smart_Read_Data                    EQU   0xB0  ;  Smart Read Data
102 CMD_Smart_Read_Log_Sector              EQU   0xB0  ;  Smart Read Log Sector
103 CMD_Smart_Return_Status                EQU   0xB0  ;  Smart Return Status
104 CMD_Smart_Write_Log_Sector             EQU   0xB0  ;  Smart Write Log Sector
105 CMD_Device_Configuration_Freeze_Lock   EQU   0xB1  ;  Device Configuration
106                                                    ;  Freeze Lock
107 CMD_Device_Configuration_Identify      EQU   0xB1  ;  Device Configuration
108                                                    ;  Identify
109 CMD_Device_Configuration_Restore       EQU   0xB1  ;  Device Configuration
110                                                    ;  Restore
111 CMD_Device_Configuration_Set           EQU   0xB1  ;  Device Configuration Set
112 CMD_Cfa_Erase_Sectors                  EQU   0xC0  ;  Cfa Erase Sectors
113 CMD_Read_Multiple                      EQU   0xC4  ;  Read Multiple
114 CMD_Write_Multiple                     EQU   0xC5  ;  Write Multiple
115 CMD_Set_Multiple_Mode                  EQU   0xC6  ;  Set Multiple Mode
116 CMD_Read_Dma_Queued                    EQU   0xC7  ;  Read Dma Queued
117 CMD_Read_Dma                           EQU   0xC8  ;  Read Dma
118 CMD_Write_Dma                          EQU   0xCA  ;  Write Dma
119 CMD_Write_Dma_Queued                   EQU   0xCC  ;  Write Dma Queued
120 CMD_Cfa_Write_Multiple_WO_Erase        EQU   0xCD  ;  Cfa Write Multiple W/Out
121                                                    ;  Erase
122 CMD_Check_Media_Card_Type              EQU   0xD1  ;  Check Media Card Type
123 CMD_Get_Media_Status                   EQU   0xDA  ;  Get Media Status
124 CMD_Media_Lock                         EQU   0xDE  ;  Media Lock
125 CMD_Media_Unlock                       EQU   0xDF  ;  Media Unlock
126 CMD_Standby_Immediate                  EQU   0xE0  ;  Standby Immediate
127 CMD_Idle_Immediate                     EQU   0xE1  ;  Idle Immediate
128 CMD_Standby                            EQU   0xE2  ;  Standby
129 CMD_Idle                               EQU   0xE3  ;  Idle
130 CMD_Read_Buffer                        EQU   0xE4  ;  Read Buffer
131 CMD_Check_Power_Mode                   EQU   0xE5  ;  Check Power Mode
132 CMD_Sleep                              EQU   0xE6  ;  Sleep
```

```
133  CMD_Flush_Cache                          EQU   0xE7  ;  Flush Cache
134  CMD_Write_Buffer                         EQU   0xE8  ;  Write Buffer
135  CMD_Flush_Cache_Ext                      EQU   0xEA  ;  Flush Cache Ext
136  CMD_Identify_Device                      EQU   0xEC  ;  Identify Device
137  CMD_Media_Eject                          EQU   0xED  ;  Media Eject
138  CMD_Set_Features                         EQU   0xEF  ;  Set Features
139  CMD_Security_Set_Password                EQU   0xF1  ;  Security Set Password
140  CMD_Security_Unlock                      EQU   0xF2  ;  Security Unlock
141  CMD_Security_Erase_Prepare               EQU   0xF3  ;  Security Erase Prepare
142  CMD_Security_Erase_Unit                  EQU   0xF4  ;  Security Erase Unit
143  CMD_Security_Freeze_Lock                 EQU   0xF5  ;  Security Freeze Lock
144  CMD_Security_Disable_Password            EQU   0xF6  ;  Security Disable Password
145  CMD_Read_Native_Max_Address              EQU   0xF8  ;  Read Native Max Address
146  CMD_Set_Max_Address                      EQU   0xF9  ;  Set Max Address
147  ;
148  ;  Vendor Specific                              0x9A
149  ;                                               0xC0-0xC3
150  ;                                               0x8x
151  ;                                               0xF0
152  ;                                               0xF7
153  ;                                               0xFA-0xFF
154  ;  Retired                                      0x11-0x1F
155  ;                                               0x71-0x7F
156  ;                                               0x94-0x99
157  ;                                               0xDB-0xDD
158  ;                                               0xE9
159  ;  Obsolete                                     0x10
160  ;                                               0x21-0x23
161  ;                                               0x31-0x33
162  ;                                               0x3C
163  ;                                               0x41
164  ;                                               0x50
165  ;                                               0xC9
166  ;                                               0xCB
167  ;                                               0xEE
168  ;

170  Interfaz_IDE   EQU FileStartC1


173  ;*****************************************************************************
174  ;*****************************************************************************
175  ;**                                                                         **
176  ;**                          Funciones de acceso                            **
177  ;**                                                                         **
178  ;*****************************************************************************
179  ;*****************************************************************************

181     org      Interfaz_IDE

183  ;*****************************************************************************
184  ;*                                                                           *
185  ;*                          IDE_ObtenerEstado                                *
186  ;*                                                                           *
187  ;*****************************************************************************
188  ;*                                                                           *
189  ;* Lee el registro de estado del IDE y lo deja en W                          *
190  ;*                                                                           *
191  ;*****************************************************************************

193  IDE_ObtenerEstado

195     call     BUS_ConfEsc          ; Configuramos el PIC para escribir el LATCH de
196                                   ; control
197     movlw    IDE_REG_STATUS       ; #CS1 #CS0 DA2 DA1 DA0
198     movwf    DATOS_BAJOS          ;   N    A   1   1   1    --> Status
199     bcf      INTCON,GIE
200     movlw    LATCH_IDE_L          ; Almacenamos la palabra de control en el
201     movwf    CONTROL1             ; registro de control
202     nop
203     movlw    OCIOSO
204     movwf    CONTROL1
205     call     BUS_ConfLec          ; Configuramos el PIC para leer del IDE
```

```
207     movlw     DIOR                    ; Activamos DIOR
208     movwf     CONTROL1
209     nop                               ; Esperamos un ciclo (por seguridad)
210     movf      DATOS_BAJOS,W           ; Ponemos el resultado (STATUS) en IDE_Status
211     movwf     IDE_Status
212     movlw     OCIOSO                  ; Desactivamos DIOR
213     bsf       INTCON,GIE
214     movwf     CONTROL1
215 ;    call      BUS_ConfEsc             ; <-------------------------
216     return

218 ;*****************************************************************************
219 ;*                                                                           *
220 ;*                                IDE_Esperar_Listo                          *
221 ;*                                                                           *
222 ;*****************************************************************************
223 ;*                                                                           *
224 ;* Chequea el estado del registro STATUS hasta que este listo (BSY y DRDY),  *
225 ;* permaneciendo en estado IDLE mientras espera                              *
226 ;*                                                                           *
227 ;*****************************************************************************

229 IDE_EsperarListo

231     call      IDE_ObtenerEstado       ; Leemos el registro STATUS del IDE
232     movf      IDE_Status,W
233     andlw     b'11000000'             ; Aplicamos una mascara para quedarnos
234                                       ; con BSY = 7 y DRDY = 6
235     xorlw     b'01000000'             ; Comprobamos si el dispositivo esta listo para
236                                       ; aceptar un comando (BSY = 0; DRDY = 1)
237     btfss     STATUS,Z
238     goto      IDE_EsperarListo        ; El bit 6 (DRDY) sigue a 0 lo testeamos
239                                       ; otra vez
240     return                            ; DRDY = 1, dispositivo listo, salimos

242 ;*****************************************************************************
243 ;*                                                                           *
244 ;*                              IDE_EsperarDatos                             *
245 ;*                                                                           *
246 ;*****************************************************************************
247 ;*                                                                           *
248 ;* Espera a que el IDE devuelva algun dato                                   *
249 ;*                                                                           *
250 ;*****************************************************************************

252 IDE_EsperarDatos

254     call      IDE_ObtenerEstado       ; Leemos el registro de estado del IDE
255     movf      IDE_Status,W
256     andlw     b'11001000'             ; Aplicamos una mascara para leer BSY(7),
257                                       ; DRDY(6) y DRQ(3)
258     xorlw     b'01001000'             ; Comprueba que no este BSY (BUSY) que este
259                                       ; DRDY (Drive ready) y que tengamos un dato
260                                       ; esperando DRQ (Data Request)
261     btfss     STATUS,Z                ; Si se cumple, sale
262     goto      IDE_EsperarDatos        ; No se cumple, sigue testeando
263     return

265 ;*****************************************************************************
266 ;*                                                                           *
267 ;*                                 IDE_Reset                                 *
268 ;*                                                                           *
269 ;*****************************************************************************
270 ;*                                                                           *
271 ;* Provoca un RESET por SOFTWARE en el dispositivo IDE                       *
272 ;*                                                                           *
273 ;*****************************************************************************

275 IDE_Reset

277     call      IDE_EsperarListo        ; Esperamos a que el dispositivo este listo
278     call      BUS_ConfEsc             ; Configuramos el PIC para escribir el LATCH de
```

```
                                        ; control
    movlw     IDE_REG_CTRL              ; #CS1 #CS0 DA2 DA1 DA0
    movwf     DATOS_BAJOS               ;   A    N   1   1   0    --> Device Control
    movlw     LATCH_IDE_L               ; Almacenamos la palabra de control en el
    movwf     CONTROL1                  ; registro de control
    nop
    movlw     OCIOSO
    movwf     CONTROL1
    movlw     b'00000110'               ; Activamos el Reset por Software y inhibimos
                                        ; las interrupciones al HOST
                                        ; SRST(2) = 1 --> Software Reset
                                        ; nIEN(1) = 1 --> MUXEnable bit for Drive
                                        ; Interrupt to the Host
    movwf     DATOS_BAJOS
    bcf       CONTROL1,DIOW             ; Activamos DIOW
    nop                                 ; Esperamos
    bsf       CONTROL1,DIOW             ; Desactivamos DIOW
    movlw     d'5'
    call      Espera_L
    movlw     b'00000010'               ; Desactivamos el Reset por Software y
                                        ; habilitamos las interrupciones al HOST
                                        ; SRST(2) = 0 --> Software Reset
                                        ; nIEN(1) = 1 --> MUXEnable bit for Drive
                                        ; Interrupt to the Host
    movwf     DATOS_BAJOS
    bcf       CONTROL1,DIOW             ; Activamos DIOW
    nop                                 ; Esperamos
    bsf       CONTROL1,DIOW             ; Desactivamos DIOW
    movlw     d'5'                      ; Esperamos 5 * 20 ms = 100 ms
    call      Espera_L
    return

;*****************************************************************************
;*                                                                           *
;*                                IDE_LBAon                                  *
;*                                                                           *
;*****************************************************************************
;*                                                                           *
;* Seleciona el modo de direccionamiento LBA y el dispositivo maestro (0).   *
;*                                                                           *
;*****************************************************************************

IDE_LBAon

    call      IDE_EsperarListo          ; Esperamos a que este listo y lo seleccionamos
    call      BUS_ConfEsc               ; Configuramos el BUS para escribir en el LATCH
                                        ; IDE
    movlw     IDE_REG_DEVICE            ; #CS1 #CS0 DA2 DA1 DA0
    movwf     DATOS_BAJOS               ;   N    A   1   1   0    --> Device Register
    movlw     LATCH_IDE_L               ; Almacenamos la palabra de control en el
    movwf     CONTROL1                  ; registro de control
    nop
    movlw     OCIOSO
    movwf     CONTROL1
    movlw     b'11100000'               ; Seleccionamos el dispositivo 0 y de paso
    movwf     DATOS_BAJOS               ; seleccionamos el modo LBA
                                        ;  7     6     5     4     3     2     1     0
                                        ; Obs    L    Obs   DEV   HS3   HS2   HS1   HS0
                                        ;  X     1     X     0     X     X     X     X
                                        ; Los Bits 7 y 5 son ignorados por el
                                        ; dispositivo, los Bits 3-0 completan la
                                        ; direccion LBA
    bcf       CONTROL1,DIOW             ; Activamos DIOW
    nop                                 ; Esperamos
    bsf       CONTROL1,DIOW             ; Desactivamos DIOW

    call      IDE_EsperarListo
    return

;*****************************************************************************
;*                                                                           *
;*                              IDE_Identifica                               *
;*                                                                           *
```

```
352 ;*****************************************************************************
353 ;*                                                                           *
354 ;* Le da la orden al dispositivo IDE de identificacion, y transfiere los 512  *
355 ;* bytes generados a la zona de memoria SRam que comienza en SRam_Hi:SRam_Lo  *
356 ;*                                                                           *
357 ;*****************************************************************************

359 IDE_Identifica

361     call      BUS_ConfEsc           ; Configuramos el BUS para escribir en el LATCH
362                                     ; IDE
363     movlw     IDE_REG_CMD           ; #CS1 #CS0 DA2 DA1 DA0
364     movwf     DATOS_BAJOS           ;   N    A   1   1   1    --> Command
365     movlw     LATCH_IDE_L           ; Almacenamos la palabra de control en el
366     movwf     CONTROL1              ; registro de control
367     nop
368     movlw     OCIOS0
369     movwf     CONTROL1
370     movlw     CMD_Identify_Device   ; Mandamos el commando IDENTIFY DEVICE
371     movwf     DATOS_BAJOS
372     bcf       CONTROL1,DIOW         ; Activamos DIOW
373     nop                             ; Esperamos
374     bsf       CONTROL1,DIOW         ; Desactivamos DIOW
375     movlw     d'1'                  ; Indicamos el numero de sectores a transferir
376     call      IDE_TransSectaSRam    ; Transferimos el sector a la SRAM
377     return

379 ;*****************************************************************************
380 ;*                                                                           *
381 ;*                                  IDE_Standby                              *
382 ;*                                                                           *
383 ;*****************************************************************************
384 ;*                                                                           *
385 ;* Pasamos el dispositivo IDE a estado de STANDBY, deja de girar para ahorrar *
386 ;* energia pero sigue respondiendo a comandos de forma normal.               *
387 ;* La latencia de los datos cuando esta en STANDBY aumenta considerablemente *
388 ;*                                                                           *
389 ;* El tiempo que tarda el dispositivo en entrar en modo STANDBY, queda       *
390 ;* determinado por el valor del registro SECTORCOUNT del IDE, segun la       *
391 ;* siguiente tabla:                                                          *
392 ;*                                                                           *
393 ;* Sector Count register       Corresponding timeout period                  *
394 ;*        contents                                                           *
395 ;* ---------------------       ----------------------------                  *
396 ;*                                                                           *
397 ;* 0          (00h)            Timeout disabled                              *
398 ;* 1-240      (01h-F0h)        (value * 5) s                                 *
399 ;* 241-251    (F1h-FBh)        ((value - 240) *30) min                       *
400 ;* 252        (FCh)            21 min                                        *
401 ;* 253        (FDh)            Period between 8 and 12                       *
402 ;* 254        (FEh)            Reserved                                      *
403 ;* 255        (FFh)            21 min 15 s                                   *
404 ;*                                                                           *
405 ;*****************************************************************************

407 IDE_Standby

409     call      IDE_EsperarListo      ; Esperamos hasta que el IDE este listo
410     call      BUS_ConfEsc           ; Configuramos el PIC para escribir el LATCH de
411                                     ; control
412     movlw     IDE_REG_SECTORCOUNT   ; #CS1 #CS0 DA2 DA1 DA0
413     movwf     DATOS_BAJOS           ;   N    A   0   1   0    --> Sector Count
414     movlw     LATCH_IDE_L           ; Almacenamos la palabra de control en el
415     movwf     CONTROL1              ; registro de control
416     nop
417     movlw     OCIOS0
418     movwf     CONTROL1
419     movlw     d'60'                 ; 60 * 5s = 300s = 5 Minutos para entrar en
420     movwf     DATOS_BAJOS           ; STANDBY
421     bcf       CONTROL1,DIOW         ; Activamos DIOW
422     nop                             ; Esperamos
423     bsf       CONTROL1,DIOW         ; Desactivamos DIOW
424     movlw     IDE_REG_CMD           ; #CS1 #CS0 DA2 DA1 DA0
```

```
425     movwf     DATOS_BAJOS           ;   N    A   1   1   1    --> Command
426     movlw     LATCH_IDE_L           ; Almacenamos la palabra de control en el
427     movwf     CONTROL1              ; registro de control
428     nop
429     movlw     OCIOSO
430     movwf     CONTROL1
431     movlw     CMD_Standby           ; Mandamos el commando STANDBY
432     movwf     DATOS_BAJOS
433     bcf       CONTROL1,DIOW         ; Activamos DIOW
434     nop                             ; Esperamos
435     bsf       CONTROL1,DIOW         ; Desactivamos DIOW
436     call      IDE_EsperarListo      ; Esperamos a que el IDE este listo
437     return

439 ;*****************************************************************************
440 ;*                                                                           *
441 ;*                              IDE_CargarNSector                            *
442 ;*                                                                           *
443 ;*****************************************************************************
444 ;*                                                                           *
445 ;* Carga el numero de sector que se quiere leer, o en el que se quiere       *
446 ;* escribir en los registros adecuados del dispositivo IDE.                  *
447 ;* El numero del primer sector a leer o escribir ha de estar contenido en las *
448 ;* variables: IDE_Sec0 - IDE_Sec3                                            *
449 ;* El numero de sectores a leer o escribir ha de estar contenido en la       *
450 ;* variable: NumSectRW                                                       *
451 ;*                                                                           *
452 ;*****************************************************************************

454 IDE_CargarNSector

456     call      BUS_ConfEsc           ; Configuramos el PIC para escribir el LATCH de
457                                     ; control
458     clrf      DATOS_ALTOS           ; Borramos la parte alta del BUS DATA8-DATA15
459     movlw     d'3'                  ; Cargamos 3 en el contador
460     movwf     Contador              ; hay que cargar 3 (IDE_Sec0 - IDE_Sec2)
461     movlw     IDE_REG_SEC0          ; #CS1 #CS0 DA2 DA1 DA0
462     movwf     DATOS_BAJOS           ;   N    A   0   1   1    --> LBA bits 0-7
463     movwf     DATOS_BAJOS_B
464     bcf       INTCON,GIE
465     movlw     LATCH_IDE_L           ; Almacenamos la palabra de control en el
466     movwf     CONTROL1              ; registro de control
467     nop
468     movlw     OCIOSO
469     movwf     CONTROL1
470     bsf       INTCON,GIE
471     movlw     IDE_Sec0              ; Ponemos IDE_Sec0 en LBA bits 0-7
472     movwf     FSR                   ; Hacemos que FSR apunte a la zona de memoria
473                                     ; donde comienza enl numero de sector
474 CNS1
475     movf      INDF,W                ; Ponemos IDE_Sec0 en DATA0-DATA7
476     movwf     DATOS_BAJOS
477     bcf       INTCON,GIE
478     bcf       CONTROL1,DIOW         ; Activamos DIOW
479     nop                             ; Esperamos
480     bsf       CONTROL1,DIOW         ; Desactivamos DIOW
481     bsf       INTCON,GIE
482     incf      FSR,F                 ; Apuntamos a la siguiente parte del numero de
483                                     ; sector a transferir IDE_Sec1, IDE_Sec2
484     incf      DATOS_BAJOS_B,F       ; Incrementamos el DATOS_BAJOS para apuntar
485                                     ; a LBA bits 8-15, LBA bits 16-23
486     movf      DATOS_BAJOS_B,W
487     movwf     DATOS_BAJOS
488     bcf       INTCON,GIE
489     movlw     LATCH_IDE_L           ; Almacenamos la palabra de control en el
490     movwf     CONTROL1              ; registro de control
491     nop
492     movlw     OCIOSO
493     movwf     CONTROL1
494     bsf       INTCON,GIE

496     decfsz    Contador,F            ; Decrementamos el contador, si no es 0 carga
497                                     ; otro registro
```

```
498     goto      CNS1                  ; Carga otro registro
499                                     ; Ahora tratamos a parte IDE_Sec3 ya que solo 4
500                                     ; bits HS0-HS3 son de direccion LBA 24-27
501     movf      INDF,W                ; Pasamos IDE_Sec3 a W
502     andlw     b'00001111'           ; Aplicamos una mascara para quedarnos con los
503                                     ; 4 bits de menor peso
504     iorlw     b'11100000'           ; Elegimos el modo LBA, L = 1
505                                     ;  7     6     5     4     3     2     1     0
506                                     ;        L          DRV   HS3   HS2   HS1   HS0
507                                     ;  1     1     1     X     X     X     X     X
508     movwf     DATOS_BAJOS           ; Ponemos el registro en DATA0-DATA7
509     bcf       INTCON,GIE
510     bcf       CONTROL1,DIOW         ; Activamos DIOW
511     nop                             ; Esperamos
512     bsf       CONTROL1,DIOW         ; Desactivamos DIOW
513     movlw     IDE_REG_SECTORCOUNT   ; #CS1 #CS0 DA2 DA1 DA0
514     movwf     DATOS_BAJOS           ;   N    A   N   A   N    --> Sector count
515     movlw     LATCH_IDE_L           ; Almacenamos la palabra de control en el
516     movwf     CONTROL1              ; registro de control
517     nop
518     movlw     OCIOSO
519     bsf       INTCON,GIE
520     movwf     CONTROL1
521     movf      NumSectRW,W           ; Ponemos el numero de sectores a transferir
522     movwf     DATOS_BAJOS
523     bsf       INTCON,GIE
524     bcf       CONTROL1,DIOW         ; Activamos DIOW
525     nop                             ; Esperamos
526     bsf       CONTROL1,DIOW         ; Desactivamos DIOW
527     bsf       INTCON,GIE
528     return

530 ;*****************************************************************************
531 ;*                                                                           *
532 ;*                              IDE_LeerSectores                             *
533 ;*                                                                           *
534 ;*****************************************************************************
535 ;*                                                                           *
536 ;* Da al dispositivo IDE la orden de leer un numero determinado de sectores  *
537 ;* consecutivos que se le pasa a la funcion por W.                           *
538 ;* El numero del primer sector a leer ha de cargarse previamente en los      *
539 ;* registros adecuados del PIC (IDE_Sec0 - IDE_Sec3).                       *
540 ;*                                                                           *
541 ;*****************************************************************************

543 IDE_LeerSectores

545     movwf     NumSectRW             ; Guardamos el numero de sectores a leer
546     call      IDE_EsperarListo      ; Esperamos a que el dispositivo IDE este listo
547     call      IDE_CargarNSector     ; Cargamos el numero de sector en el
548                                     ; dispositivo
549     call      BUS_ConfEsc           ; Configuramos el PIC para escribir el LATCH de
550                                     ; control
551     movlw     IDE_REG_CMD           ; #CS1 #CS0 DA2 DA1 DA0
552     movwf     DATOS_BAJOS           ;   N    A   1   1   1    --> Command
553     bcf       INTCON,GIE
554     movlw     LATCH_IDE_L           ; Almacenamos la palabra de control en el
555     movwf     CONTROL1              ; registro de control
556     nop
557     movlw     OCIOSO
558     movwf     CONTROL1
559     movlw     CMD_Read_Sector       ; Comando --> Read Sector
560     movwf     DATOS_BAJOS
561     bcf       CONTROL1,DIOW         ; Activamos DIOW
562     nop                             ; Esperamos
563     bsf       CONTROL1,DIOW         ; Desactivamos DIOW
564     bsf       INTCON,GIE
565 ;    movlw     d'1'                 ; Esperamos 1ms para asegurar que STATUS es
566 ;    call      Espera_C             ; correcto
567     return

569 ;*****************************************************************************
570 ;*                                                                           *
```

```
571 ;*                                IDE_EscSectores                                *
572 ;*                                                                               *
573 ;*******************************************************************************
574 ;*                                                                               *
575 ;* Da al dispositivo IDE la orden de escribir un numero determinado de          *
576 ;* sectores consecutivos  que se le pasa a la funcion por W.                    *
577 ;* El numero del primer sector a escribir ha de cargarse previamente en los     *
578 ;* registros adecuados del PIC (IDE_Sec0 - IDE_Sec3).                           *
579 ;*                                                                               *
580 ;*******************************************************************************

582 IDE_EscSectores

584     movwf    NumSectRW            ; Guardamos el numero de sectores a escribir
585     call     IDE_EsperarListo     ; Esperamos a que el dispositivo IDE este listo
586     call     IDE_CargarNSector    ; Cargamos el numero de sector en el
587                                   ; dispositivo
588     call     BUS_ConfEsc          ; Configuramos el PIC para escribir el LATCH de
589                                   ; control
590     movlw    IDE_REG_CMD          ; #CS1 #CS0 DA2 DA1 DA0
591     movwf    DATOS_BAJOS          ;   N    A   A   A   A    --> Command
592     movlw    LATCH_IDE_L          ; Almacenamos la palabra de control en el
593     movwf    CONTROL1             ; registro de control
594     nop
595     movlw    OCIOSO
596     movwf    CONTROL1
597     movlw    CMD_Write_Sector     ; Comando --> Write Sector
598     movwf    DATOS_BAJOS
599     bcf      CONTROL1,DIOW
600     nop                           ; Esperamos
601     bsf      CONTROL1,DIOW
602 ;   movlw    d'1'                 ; Esperamos 1ms para asegurar que STATUS es
603 ;   call     Espera_C             ; correcto
604     return

606 ;*******************************************************************************
607 ;*                                                                               *
608 ;*                              IDE_TransSectaSRam                               *
609 ;*                                                                               *
610 ;*******************************************************************************
611 ;*                                                                               *
612 ;* Transfiere NumSectTR sectores de los NumSectRW sectores leidos del            *
613 ;* dispositivo IDE a la zona de la memoria SRam que comienza en la direccion     *
614 ;* SRam_Hi:SRam_Lo.                                                              *
615 ;* El numero de sectores a tranferir NumSectTR se le pasa a la funcion a         *
616 ;* traves de W.                                                                  *
617 ;* Para su correcto funcionamiento, es necesario que se le haya dado al          *
618 ;* dispositivo IDE previamente, la orden de leer como minimo, el mismo numero    *
619 ;* de sectores de los que se van a transferir.                                   *
620 ;* HAY QUE TENER MUY BIEN CONTROLADOS NumSectTR y NumSectRW.                     *
621 ;* La transferencia se hace en bloques, cada bloque esta formado por 32 bytes    *
622 ;* (16 Words) y  un sector contiene 16 bloques, por lo que se transfieren        *
623 ;* (16 * NumSectTR) bloques                                                      *
624 ;* 1 Sector = 16 Bloques                                                         *
625 ;* 1 bloque = 32 bytes (16 words)                                                *
626 ;* 1 Sector = 512 bytes (256 words)                                              *
627 ;*                                                                               *
628 ;*******************************************************************************

630 IDE_TransSectaSRam
631     clrf     STATUS               ; Seleccionamos el Banco 0


634     movwf    NumSectTR            ; Guardamos el numero de sectores a transferir
635     call     BUS_ConfEsc          ; Configuramos el PIC para escribir el LATCH de
636                                   ; memoria
637     movf     SRam_Hi,W            ; Ponemos la direccion destino en el LATCH de
638     movwf    DirMem_H             ; memoria
639     movwf    DIR_SRAM_H
640     movf     SRam_Lo,W
641     movwf    DirMem_L
642     movwf    DIR_SRAM_L
643     bcf      INTCON,GIE
```

```
644     movlw     LATCH_MEM_L           ; Activamos LATCH_MEM_L
645     movwf     CONTROL1
646     nop                             ; Esperamos
647     movlw     OCIOSO                ; Desactivamos LATCH_MEM_L
648     movwf     CONTROL1
649     bsf       INTCON,GIE
650                                     ; Vamos a transferir NumSectTR sectores
651 Sector_L
652     call      IDE_EsperarDatos      ; Hemos transferido el sector entero, esperamos
653                                     ; hasta que el IDE tenga listo el siguiente
654     movlw     d'0'                  ; --> 256 Numero de words a transferir por
655     movwf     NumWords              ; sector
656     call      BUS_ConfEsc           ; Configuramos el PIC para escribir el LATCH
657                                     ; IDE
658     movlw     IDE_REG_DATA          ; #CS1 #CS0 DA2 DA1 DA0
659     movwf     DATOS_BAJOS           ;   N    A   0   0   0    --> Data
660     bcf       INTCON,GIE
661     movlw     LATCH_IDE_L           ; Almacenamos la palabra de control en el
662     movwf     CONTROL1              ; registro de control
663     nop
664     movlw     OCIOSO
665     movwf     CONTROL1
666     bsf       INTCON,GIE
667     call      BUS_ConfLec           ; Configuramos el PIC para leer del IDE

669 Word_L
670     bcf       INTCON,GIE
671     movlw     DIOR                  ; Activamos DIOR
672     movwf     CONTROL1
673     bcf       CONTROL2,WE           ; Escribimos en la SRAM
674     nop
675     bsf       CONTROL2,WE
676     movlw     OCIOSO                ; Desactivamos DIOR
677     movwf     CONTROL1
678     bsf       CONTROL2,INC_ADDR     ; Incrementamos la direccion de destino
679     nop
680     bcf       CONTROL2,INC_ADDR
681     incf      DirMem_L,F
682     btfsc     STATUS,Z
683     incf      DirMem_H,F
684     bsf       INTCON,GIE
685     decfsz    NumWords,F            ; Decrementamos en 1 el numero de words
686                                     ; transferidas y comprobamos si hemos acabado
687     goto      Word_L                ; No hemos acabado, movemos otra word

689     decfsz    NumSectTR,F           ; Decrementamos en 1 el numero de sectores
690                                     ; pendientes de transferir
691     goto      Sector_L              ; Si quedan Sectores por transferir se
692                                     ; transfieren, si no, hemos acabado
693     return

695 ;*****************************************************************************
696 ;*                                                                           *
697 ;*                              IDE_TransSectaIDE                            *
698 ;*                                                                           *
699 ;*****************************************************************************
700 ;*                                                                           *
701 ;* Transfiere NumSectTR sectores leidos de la memoria SRam a la zona del     *
702 ;* dispositivo IDE que comienza en el numero de sector que contiene el       *
703 ;* dispositivo y que previamente ha de ser cargado en el mismo.              *
704 ;* El numero de sectores a tranferir NumSectTR se le pasa a la funcion a     *
705 ;* traves de W.                                                              *
706 ;* La direccion de la que se transfieren los datos esta contenida en         *
707 ;* SRam_Hi:SRam_Lo.                                                          *
708 ;* Para su correcto funcionamiento, es necesario que se le haya dado al      *
709 ;* dispositivo IDE previamente, la orden de escribir el mismo numero de      *
710 ;* sectores de los que se van a transferir.                                  *
711 ;* HAY QUE TENER MUY BIEN CONTROLADOS NumSectTR y NumSectRW.                 *
712 ;* La transferencia se hace en bloques, cada bloque esta formado por 32 bytes *
713 ;* (16 Words) y  un sector contiene 16 bloques, por lo que se transfieren    *
714 ;* (16 * NumSectTR) bloques                                                  *
715 ;* 1 Sector = 16 Bloques                                                     *
716 ;* 1 bloque = 32 bytes (16 words)                                            *
```

```
717  ;* 1 Sector = 512 bytes (256 words)                                    *
718  ;*                                                                      *
719  ;************************************************************************

721  IDE_TransSectaIDE

723     movwf    NumSectTR            ; Guardamos el numero de sectores a transferir
724     call     BUS_ConfEsc          ; Configuramos el PIC para escribir el LATCH de
725                                   ; memoria
726     movf     SRam_Hi,W            ; Ponemos la direccion destino en el LATCH de
727     movwf    DirMem_H             ; memoria
728     movwf    DIR_SRAM_H
729     movf     SRam_Lo,W
730     movwf    DirMem_L
731     movwf    DIR_SRAM_L
732     movlw    LATCH_MEM_L          ; Activamos LATCH_MEM_L
733     movwf    CONTROL1
734     nop                           ; Esperamos
735     movlw    OCIOSO               ; Desactivamos LATCH_MEM_L
736     movwf    CONTROL1
737                                   ; Vamos a transferir NumSectTR sectores
738  Sector_E
739     call     IDE_EsperarDatos     ; Hemos transferido el sector entero, esperamos
740                                   ; hasta que el IDE tenga listo el siguiente
741     movlw    d'0'                 ; --> 256 Numero de words a transferir por
742     movwf    NumWords             ; bloque
743     call     BUS_ConfEsc          ; Configuramos el PIC para escribir el LATCH de
744                                   ; memoria
745     movlw    IDE_REG_DATA         ; #CS1 #CS0 DA2 DA1 DA0
746     movwf    DATOS_BAJOS          ;   N    A   0   0   0    --> Data
747     movlw    LATCH_IDE_L          ; Almacenamos la palabra de control en el
748     movwf    CONTROL1             ; registro de control
749     nop
750     movlw    OCIOSO
751     movwf    CONTROL1
752     call     BUS_ConfLec          ; Configuramos el PIC para leer de la memoria
753                                   ; SRam
754  Word_E
755     movlw    OE                   ; Leemos la memoria SRam (OE)
756     movwf    CONTROL1
757     bcf      CONTROL1,DIOW        ; Activamos DIOW
758     nop
759     bsf      CONTROL1,DIOW        ; Desactivamos DIOW
760     movlw    OCIOSO               ; Desactivamos OE
761     movwf    CONTROL1
762     bcf      INTCON,GIE
763     bsf      CONTROL2,INC_ADDR    ; Incrementamos la direccion de destino
764     nop
765     bcf      CONTROL2,INC_ADDR
766     incf     DirMem_L,F
767     btfsc    STATUS,Z
768     incf     DirMem_H,F
769     bsf      INTCON,GIE
770     decfsz   NumWords,F           ; Decrementamos en 1 el numero de words
771                                   ; transferidas y comprobamos si hemos acabado
772     goto     Word_E               ; No hemos acabado, movemos otra word

774     decfsz   NumSectTR,F          ; Decrementamos en 1 el numero de sectores
775                                   ; pendientes de transferir
776     goto     Sector_E             ; Si quedan Sectores por transferir se
777                                   ; transfieren, si no, hemos acabado
778     call     BUS_ConfEsc
779     return

781     VARIABLE       FileStartC1 = $
```

A.4. Interfaz FAT32

Una vez extraída la información en forma de bytes del dispositivo IDE, esta tiene que ser interpretada de acuerdo al formato FAT32. Esta parte del código, es la que se encarga de esta tarea.

Se encuentra localizado en la **Página 0** de memoria de programa del microcontrolador.

A.4.1. Cabecera - Definición de Constantes

```
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************
;***                                                                         ***
;**                    MANEJO DEL SISTEMA DE FICHEROS FAT32                   **
;***                                                                         ***
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************

;*******************************************************************************
;*******************************************************************************
;**                                                                           **
;**                              MASTER BOOT RECORD                           **
;**                                                                           **
;*******************************************************************************
;*******************************************************************************

; El MBR es el sector en el que toda la informacion de la particion es
; almacenada, el sistema tiene que conocer perfectamente su ubicacion, por eso
; se le da una localizacion fija, se encuentra SIEMPRE en el primer sector del
; disco duro:
; (HEAD:0 CYLINDER:0 SECTOR:1).
;
; Cuando el PC arranca, la BIOS carga el MBR en memoria.
; Para identificar el sector como "bootable" la ultima palabra de este sector
; debe ser 0xAA55, esto implicara que si el disco duro es el "primero", los
; primeros bytes del MBR seran instrucciones.
; La BIOS se encargara pues, de buscar este codigo y ejecutarlo.
;
; Estructura del MBR:
;
;  0x000 -- Codigo de Particion
;  0x1BE -- Tabla de Particion
;  0x1FE -- Bytes que determinan si el sector puede ser ejecutado ("bootable")
;
; Una de las particiones contendra el Sistema Operativo, esta particion es la
; ACTIVA.
; En cada momento solo podemos tener una particion activa.
; El cometido de el "codigo de particion" es identificar la particion activa,
; cargar el sector de arranque de esta particion en memoria (BOOT SECTOR) y
; ejecutarlo
;
; Estructura de la tabla de particiones:
;
;  0x1BE -- [16 BYTES] Primera entrada de la tabla de particiones
;  0x1CE -- [16 BYTES] Segunda entrada de la tabla de particiones
;  0x1DE -- [16 BYTES] Tercera entrada de la tabla de particiones
;  0x1EE -- [16 BYTES] Cuarta entrada de la tabla de particiones
;
; Cada entrada de la tabla de particiones contiene:
;
;  Offset
;  ------
;   0x00 -- Estado de la particion
;   0x01 -- HEAD en la que comienza la particion
```

```
57  ;   0x02 -- SECTOR y CYLINDER donde comienza la particion
58  ;   0x04 -- Tipo de particion
59  ;   0x05 -- HEAD donde acaba la particion
60  ;   0x06 -- SECTOR y CYLINDER donde acaba la particion
61  ;   0x08 -- Distancia en sectores hasta el primer sector de la particion
62  ;   0x0C -- Numero de sectores de la particion

64  ;*****************************************************************************
65  ;*****************************************************************************
66  ;*****************************************************************************
67  ;**                                                                         **
68  ;**   Offsets en la primera entrada de particion (Master Boot Record)       **
69  ;**                                                  512 Bytes              **
70  ;**                                                                         **
71  ;**     La primera entrada de particion (First Partition Entry) esta situada **
72  ;**     a partirde la direcion 1BEh                                         **
73  ;**                                                                         **
74  ;**   PEOFFSET --> Partition Entry Offset                                   **
75  ;**                                                                         **
76  ;*****************************************************************************
77  ;*****************************************************************************

79  PEOFFSET_IsActive       equ   0x1BE ; [1 BYTE], Indica si la particion esta
80                                      ; activa
81     ACTIVE               equ   0x80  ; Esta activa
82     NOT_ACTIVE           equ   0x00  ; NO esta activa

84  PEOFFSET_StartHead      equ   0x1BF ; [1 BYTE], Cabeza de comienzo para la
85                                      ; particion
86  PEOFFSET_StartCylSect   equ   0x1C0 ; [2 BYTE], Cilindro y sector de comienzo
87  PEOFFSET_PartType       equ   0x1C2 ; [1 BYTE], Tipo de particion
88  PEOFFSET_EndHead        equ   0x1C3 ; [1 BYTE], Ultima cabeza para la particion
89  PEOFFSET_EndCylSect     equ   0x1C4 ; [2 BYTE], Cilindro y sector de fin
90  PEOFFSET_StartLBA       equ   0x1C6 ; [4 BYTE], Distancia en sectores al primer
91                                      ; sector de la particion
92  PEOFFSET_Size           equ   0x1CA ; [4 BYTE], Tamaño de la particion en
93                                      ; sectores

95  ; Los posibles tipos de particion (mas comunes) con los que nos podemos
96  ; encontrar son:

98  ;*****************************************************************************
99  ;*****************************************************************************
100 ;**                                                                         **
101 ;**                  Tipo de particion del disco duro mas comunes           **
102 ;**                                                                         **
103 ;*****************************************************************************
104 ;*****************************************************************************

106 PART_TYPE_UNKNOWN       equ   0x00  ; --> Vacia
107 PART_TYPE_FAT12         equ   0x01  ; --> DOS 12-bits FAT
108 PART_TYPE_XENIX_R       equ   0x02  ; --> XENIX Root
109 PART_TYPE_XENIX_U       equ   0x03  ; --> XENIX User
110 PART_TYPE_DOSFAT16      equ   0x04  ; --> DOS 16-bits < 32 Mb
111 PART_TYPE_EXTDOS        equ   0x05  ; --> Particion de MS-DOS Extendida
112 PART_TYPE_FAT16         equ   0x06  ; --> DOS 16-bits >= 32 Mb
113 PART_TYPE_NTFS          equ   0x07  ; --> NTFS
114 PART_TYPE_AIX           equ   0x08  ; --> AIX
115 PART_TYPE_AIX_I         equ   0x09  ; --> AIX Inicializable
116 PART_TYPE_OS2           equ   0x0A  ; --> OS/2 Boot Manager
117 PART_TYPE_FAT32         equ   0x0B  ; --> 32-bit FAT (Hasta 2048GB)
118 PART_TYPE_FAT32LBA      equ   0x0C  ; --> 32-bit FAT (Hasta 2048GB) LBA
119 PART_TYPE_FAT16LBA      equ   0x0E  ; --> DOS 16-bits >= 32 Mb LBA
120 PART_TYPE_EXTDOSLBA     equ   0x0F  ; --> Particion extendida MS-DOS LBA
121 PART_TYPE_ONTRACK       equ   0x33  ; --> Ontrack
122 PART_TYPE_VENIX         equ   0x40  ; --> Venix 80286
123 PART_TYPE_PCIX          equ   0x4B  ; --> PCIX
124 PART_TYPE_NOVELL        equ   0x51  ; --> Novell
125 PART_TYPE_MICRO         equ   0x52  ; --> Microport
126 PART_TYPE_GNU           equ   0x63  ; --> GNU HURD
127 PART_TYPE_NOVELL_B      equ   0x64  ; --> Novell
128 PART_TYPE_PCIX          equ   0x75  ; --> PC/IX
129 PART_TYPE_O_MINIX       equ   0x80  ; --> Old MINIX
```

```
130  PART_TYPE_MINIX          equ   0x81  ; --> MINIX/Linux
131  PART_TYPE_LINUX_S        equ   0x82  ; --> Linux Swap
132  PART_TYPE_LINUX_N        equ   0x83  ; --> Linux Native
133  PART_TYPE_AMOEBA         equ   0x93  ; --> Amoeba
134  PART_TYPE_AMOEBA_B       equ   0x94  ; --> Amoeba BBT
135  PART_TYPE_PHOENIXSAVE    equ   0xA0  ; --> Phoenix
136  PART_TYPE_BSD            equ   0xA5  ; --> BSD/386
137  PART_TYPE_BSDI           equ   0xB7  ; --> BSDI fs
138  PART_TYPE_BSDI_S         equ   0xB8  ; --> BSDI swap
139  PART_TYPE_SYRINX         equ   0xC7  ; --> Syrinx
140  PART_TYPE_CPM            equ   0xDB  ; --> CP/M
141  PART_TYPE_DBFS           equ   0xE0  ; --> DBFS
142  PART_TYPE_DOS            equ   0xE1  ; --> Acceso a DOS
143  PART_TYPE_DOS_R          equ   0xE3  ; --> DOS R/O
144  PART_TYPE_DOS_S          equ   0xF2  ; --> DOS secundaria
145  PART_TYPE_BBT            equ   0xFF  ; --> BBT

147  ;*****************************************************************************
148  ;*****************************************************************************
149  ;**                                                                         **
150  ;**      Offsets dentro del sector de arranque FAT32 (FAT32 BOOT Record).   **
151  ;**                                                                         **
152  ;**   Esta informacion esta localizada en el primer sector de cada particion. **
153  ;**   (512 bytes)                                                           **
154  ;**                                                                         **
155  ;**   BROFFSET --> Boot Record Offset                                       **
156  ;**                                                                         **
157  ;*****************************************************************************
158  ;*****************************************************************************

160  BROFFSET_jmpBoot         equ   0x00  ; [3 BYTE] Instruccion de salto al codigo de
161                                        ; arranque, puede ser 0xE9:0xXX:0xXX o
162                                        ; 0xEB:0xXX:0x90
163  BROFFSET_OEMName         equ   0x03  ; [8 BYTE] Nombre OEM y version
164  BROFFSET_BytsPerSec      equ   0x0B  ; [2 BYTE] Bytes por sector
165  BROFFSET_SecPerClus      equ   0x0D  ; [1 BYTE] Sectores por cluster
166  BROFFSET_RsvdSecCnt      equ   0x0E  ; [2 BYTE] Numero de sectores reservados
167  BROFFSET_NumFATs         equ   0x10  ; [1 BYTE] Numero de FATs
168  BROFFSET_RootEntCnt      equ   0x11  ; [2 BYTE] FAT12 y FAT16 --> Numero de
169                                        ; entradas de directorio de 32 bytes en el
170                                        ; directorio raiz.
171                                        ; FAT32 --> 0
172  BROFFSET_TotSec16        equ   0x13  ; [2 BYTE] FAT12 y FAT16 --> Numero total de
173                                        ; sectores.
174                                        ; FAT32 --> 0
175  BROFFSET_Media           equ   0x15  ; [1 BYTE] Descriptor del tipo de
176                                        ; dispositivo usado(fijo, removible)
177  BROFFSET_FATSz16         equ   0x16  ; [2 BYTE] FAT12 y FAT16 --> Numero de
178                                        ; sectores por FAT.
179                                        ; FAT32 --> 0
180  BROFFSET_SecPerTrk       equ   0x18  ; [2 BYTE] Numero de sectores por pista
181  BROFFSET_NumHeads        equ   0x1A  ; [2 BYTE] Numero de cabezas
182  BROFFSET_HiddSec         equ   0x1C  ; [4 BYTE] Numero de sectores ocultos que
183                                        ; hay precediendo a la particion que
184                                        ; contiene el volumen FAT
185  BROFFSET_TotSec32        equ   0x20  ; [4 BYTE] Numero total de sectores (solo si
186                                        ; BROFFSET_TotSec16 = 0)
187  BROFFSET_FATSz32         equ   0x24  ; [4 BYTE] Numero de sectores por FAT (solo
188                                        ; si BROFFSET_FATSz16 = 0)
189  BROFFSET_ExtFlags        equ   0x28  ; [2 BYTE] Flags. Solo para FAT32

191     FATNUM                equ   0x0F  ; Mascara para el numero de la FAT activa (
192                                        ; solo con FATMIRROR = 1)
193     FATMIRROR             equ   0x80  ; 0 --> La FAT esta duplicada (mirrored at
194                                        ; runtime).
195                                        ; 1 --> Solo una FAT esta activa y es la
196                                        ; referenciada en los bits FATNUM

198  BROFFSET_FSVer           equ   0x2A  ; [2 BYTE] Solo para FAT32. Version del
199                                        ; dispositivo FAT32 (HighByte = Major
200                                        ; revision number, Low Byte = Minor revision
201                                        ; number)
202     FSVERS                equ   0x00  ; Actualmente solo el dispositivo 0:0 esta
```

```
                                        ; permitido

BROFFSET_RootClus        equ    0x2C    ; [4 BYTE] Solo para FAT32. Numero de
                                        ; cluster de comienzo del directorio raiz
BROFFSET_FSInfo          equ    0x30    ; [2 BYTE] Solo para FAT32. Numero de sector
                                        ; de la estructura FSINFO en el area
                                        ; reservada del volumen FAT32
BROFFSET_BkBootSec       equ    0x32    ; [2 BYTE] Solo para FAT32. Si no es 0
                                        ; indica el numero de sector en el que se
                                        ; encuentar la copia del MBR
BROFFSET_Reserved        equ    0x34    ; [12 BYTE] Solo para FAT32. Reservado para
                                        ; futuras expansiones, debe ser 0
BROFFSET_DrvNum          equ    0x40    ; [1 BYTE] Int 0x13 drive number
BROFFSET_Reserved1       equ    0x41    ; [1 BYTE] Reservado para su uso con
                                        ; Windows NT
BROFFSET_BootSig         equ    0x42    ; [1 BYTE] Extended Boot signature
BROFFSET_VolID           equ    0x43    ; [4 BYTE] Numero de serie del volumen
BROFFSET_VolLab          equ    0x47    ; [11 BYTE] Nombre del volumen debe
                                        ; coincidir con la entrada del directorio
                                        ; raiz que tiene atributo de 'Volume label'
BROFFSET_FilSysType      equ    0x52    ; [8 BYTE] Tipo de sistema de ficheros

BROFFSET_BootCode        equ    0x5A    ; [420 BYTE] Codigo de arranque ejecutable
BROFFSET_BootSectSig0    equ    0x1FE   ; [1 BYTE] Primer byte de la firma
                                        ; (signature) del codigo de arranque
   BOOTSIG0              equ    0x55    ; Mascara para chequear el primer byte

BROFFSET_BootSectSig1    equ    0x1FF   ; [1 BYTE] Segundo byte de la firma
                                        ; (signature) del codigo de arranque
   BOOTSIG1              equ    0xAA    ; Mascara para chequear el segundo byte

;*****************************************************************************
;*****************************************************************************
;**                                                                         **
;**              Offsets en una entrada de directorio DOS                   **
;**                                                                         **
;**   DEOFFSET --> Directory Entry Offset                                   **
;**                                                                         **
;*****************************************************************************
;*****************************************************************************

DEOFFSET_Name            equ    0x00    ; [8 BYTES] Nombre del fichero
   DE_END                equ    0x00    ; Libre y ultima entrada de directorio
   DE_CHRxE5             equ    0x05
   DE_FREE               equ    0xE5    ; Entrada libre (borrada)

DEOFFSET_NameExt         equ    0x08    ; [3 BYTES] Extension del fichero
DEOFFSET_Attr            equ    0x0B    ; [1 BYTE ] Atributos del fichero
   ATTR_FILE             equ    0x00    ; Fichero normal
   ATTR_READ_ONLY        equ    0x01    ; Fichero de solo lectura
   ATTR_HIDDEN           equ    0x02    ; Fichero oculto
   ATTR_SYSTEM           equ    0x04    ; Fichero de systema
   ATTR_VOLUME_ID        equ    0x08    ; La entrada es el nombre de volumen
   ATTR_DIRECTORY        equ    0x10    ; La entrada es el nombre de directorio
   ATTR_ARCHIVE          equ    0x20    ; El fichero es nuevo o ha sido modificado
   ATTR_LONG_NAME        equ    0x0F    ; La entrada es de nombre largo

   ATTR_MASK             equ    0x0F    ; Mascara para detectar un fichero normal
   ATTR_HID_SYS          equ    0x06    ; Mascara para detectar un fichero oculto y
                                        ; de sistema

DEOFFSET_ntRes           equ    0x0C    ; [1 BYTE] Reservado para su uso con NT VFAT
                                        ; (lower case flags)
   LCASE_BASE            equ    0x08    ; Base del nombre del fichero (lower case)
   LCASE_EXT             equ    0x10    ; Extension del nombre del fichero (lower
                                        ; case)

DEOFFSET_CreateTimeTenth   equ    0x0D   ; [1 BYTE] Decimas de segundos en el
                                         ; momento de lacreacion del fichero
DEOFFSET_CreateTime      equ    0x0E    ; [2 BYTE] Hora de creacion
DEOFFSET_CreateDate      equ    0x10    ; [2 BYTE] Fecha de creacion
DEOFFSET_LastAccDate     equ    0x12    ; [2 BYTE] Fecha del ultimo acceso
DEOFFSET_FirstClusHi     equ    0x14    ; [2 BYTE] Bytes altos del numero de cluster
```

```
DEOFFSET_WriteTime      equ   0x16  ; [2 BYTE] Hora de la ultima actualizacion
DEOFFSET_WriteDate      equ   0x18  ; [2 BYTE] Fecha de la ultima actualizacion
DEOFFSET_FirstClusLo    equ   0x1A  ; [2 BYTE] Cluster de comienzo del fichero
DEOFFSET_FileSize       equ   0x1C  ; [4 BYTE] Tamaño del fichero en bytes

;*******************************************************************************
;*******************************************************************************
;**                                                                           **
;**           Offsets en una entrada de directorio largo FAT32                **
;**                                                                           **
;*******************************************************************************
;*******************************************************************************

LNOFFSETDIR_Ord         equ   0x00  ; [ 1 BYTE] Orden de la entrada en la
                                    ; secuencia de de entradas de nombre largo
   MASK_LAST_LONG_ENTRY equ   0x40  ; Mascara para detectar la ultima entrada
                                    ; larga
   MASK_NTH_LONG_ENTRY  equ   0x3F  ; Mascara para leer el numero de orden de la
                                    ; entrada larga
LNOFFSETDIR_Name1       equ   0x01  ; [10 BYTES] Caracteres 1 a 5 del nombre
                                    ; largo (UNICODE --> 2 bytes por caracter)
LNOFFSETDIR_Attr        equ   0x0B  ; [1 BYTE] Atributos (debe estar fijado a
                                    ; ATTR_LONG_NAME)
LNOFFSETDIR_Type        equ   0x0C  ; [1 BYTES] 0 --> Entrada de directorio que
                                    ; es una  subcomponente de un nombre largo.
                                    ; Cualquier otro valor implica un tipo
                                    ; diferente
LNOFFSETDIR_Chksum      equ   0x0D  ; [1 BYTES] CHECKSUM del nombre en la
                                    ; entrada de directorio corta al final del
                                    ; set de entradas de directorio largas
LNOFFSETDIR_Name2       equ   0x0E  ; [12 BYTES] Caracteres 6 a 11 del nombre
                                    ; largo (UNICODE --> 2 bytes por caracter)
LNOFFSETDIR_FstClusLO   equ   0x1A  ; [2 BYTES] Debe ser 0 para mantener la
                                    ; compatibilidad con las utilidades
                                    ; existentes de disco
LNOFFSETDIR_Name3       equ   0x1C  ; [4 BYTES] Caracteres 12 y 13 del nombre
                                    ; largo (UNICODE --> 2 bytes por caracter)
```

A.4.2. Programa

```
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************
;***                                                                         ***
;**                                I_FAT32.asm                              **
;**                                                                         **
;** Permite la navegacion por el dispositivo IDE en formato FAT32.          **
;**                                                                         **
;***                                                                         ***
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************

   INCLUDE  "I_FAT32.inc"

Interfaz_FAT32    EQU FileStartC1

;*******************************************************************************
;*******************************************************************************
;**                                                                         **
;**                            Funciones de acceso LCD                      **
;**                                                                         **
;*******************************************************************************
;*******************************************************************************

   org      Interfaz_FAT32

;*******************************************************************************
;*                                                                             *
;*                                  Lee_4BYTES                                 *
;*                                                                             *
;*******************************************************************************
;*                                                                             *
;* Lee 4 BYTES consecutivos de la direccion de memoria apuntada por S          *
;* Ram_Hi:SRam_Lo y los deposita en la memoria del PIC a partir de la          *
;* direccion apuntada por FSR                                                  *
;*                                                                             *
;*******************************************************************************

Lee_4BYTES
   call     BUS_ConfEsc        ; Vamos a escribir en el LATCH de MEMORIA
   movf     SRam_Hi,W          ; Cargamos la direccion en el Latch
   movwf    DirMem_H
   movwf    DIR_SRAM_H
   movf     SRam_Lo,W
   movwf    DirMem_L
   movwf    DIR_SRAM_L

   bcf      INTCON,GIE
   movlw    LATCH_MEM_L        ; Activamos LATCH_MEM_L
   movwf    CONTROL1
   nop                         ; Esperamos
   movlw    OCIOSO             ; Desactivamos LATCH_MEM_L
   movwf    CONTROL1
   bsf      INTCON,GIE

   movlw    d'2'               ; Cargamos el contador con 2, leemos los 4 BYTES
   movwf    Contador1          ; de dos en dos
   call     BUS_ConfLec        ; Configuramos el BUS para leer la MEMORIA SRAM

Lee_2BYTES
   bcf      INTCON,GIE
   movlw    OE                 ; Leemos la memoria SRam (OE)
   movwf    CONTROL1
   movf     DATOS_BAJOS,W
   movwf    INDF               ; Movemos el byte a su destino
   incf     FSR,F
   movf     DATOS_ALTOS,W
   movwf    INDF
   incf     FSR,F
   movlw    OCIOSO             ; Deactivamos OE
```

```
72      movwf     CONTROL1
73      bsf       CONTROL2,INC_ADDR ; Incrementamos la direccion de destino
74      nop
75      bcf       CONTROL2,INC_ADDR
76      incf      DirMem_L,F
77      btfsc     STATUS,Z
78      incf      DirMem_H,F
79      bsf       INTCON,GIE
80      decfsz    Contador1,F
81      goto      Lee_2BYTES
82      return

84   ;*****************************************************************************
85   ;*                                                                           *
86   ;*                                Inicializa_FAT32                           *
87   ;*                                                                           *
88   ;*****************************************************************************
89   ;*                                                                           *
90   ;* Se encarga de obtener y procesar el MBR y el BOOT SECTOR para obtener las *
91   ;* tablas FAT y dar acceso a los archivos y directorios del dispositivo.     *
92   ;* Devuelve 0 en W si todo ha ido bien, si ocurrio algun erro devuelve 1     *
93   ;*                                                                           *
94   ;*****************************************************************************

96   Inicializa_FAT32

98      ; ***** LECTURA DEL PRIMER SECTOR FAT32 *****

100     clrw                          ; Indicamos el numero de sector LBA a leer
101     movwf     IDE_Sec0
102     movwf     IDE_Sec1
103     movwf     IDE_Sec2
104     movwf     IDE_Sec3
105     movlw     d'1'                ; Indicamos que queremos leer 1 sector
106     call      IDE_LeerSectores    ; Leemos el sector

108     movlw     0x00                ; Cargamos la direccion de destino en la SRAM
109     movwf     SRam_Hi
110     movlw     0x00
111     movwf     SRam_Lo
112     movlw     d'1'                ; Indicamos que queremos transferir 1 sector
113     call      IDE_TransSectaSRam  ; Transferimos el sector a la SRAM

115  ;*****************************************************************************

117  ;************************** COMPROBAMOS SI ES FAT32 **************************

119  ; Tipos de Particiones
120  ; --------------------
121  ;
122  ; Numero de referencia     Tipo
123  ; --------------------    -------------------------------------
124  ;        00h               Vacia o Nada
125  ;        01h               DOS 12-bits FAT
126  ;        02h               XENIX root
127  ;        03h               XENIX usr
128  ;        04h               DOS 16-bits < 32 Mb
129  ;        05h               Particion extendida MS-DOS
130  ;        06h               DOS 16-bits >= 32 Mb
131  ;        07h               OS/2 HPFS
132  ;        08h               AIX
133  ;        09h               AIX inicializable
134  ;        0Ah               OS/2 Boot Manager
135  ;        0Bh               32-bit FAT (Particion hasta 2048GB)
136  ;        0Ch               32-bit FAT (Particion hasta 2048GB) LBA
137  ;        0Eh               DOS 16-bits >= 32 Mb LBA
138  ;        0Fh               Particion extendida MS-DOS LBA
139  ;        40h               Venix 80286
140  ;        51h               Novell
141  ;        52h               Microport
142  ;        63h               GNU HURD
143  ;        64h               Novell
144  ;        75h               PC/IX
```

```
145  ;        80h                Old MINIX
146  ;        81h                MINIX/Linux
147  ;        82h                Linux Swap
148  ;        83h                Linux Native
149  ;        93h                Amoeba
150  ;        94h                Amoeba BBT
151  ;        A5h                BSD/386
152  ;        B7h                BSDI fs
153  ;        B8h                BSDI swap
154  ;        C7h                Syrinx
155  ;        DBh                CP/M
156  ;        E1h                Acceso a DOS
157  ;        E3h                DOS R/O
158  ;        F2h                DOS secundaria
159  ;        FFh                BBT

161     call     BUS_ConfEsc           ; Vamos escribir en el LATH de MEMORIA
162                                    ; Verificamos que es una particion FAT32
163     movlw    PEOFFSET_PartType>>1  ; Parte baja del offset del byte que nos dice
164     movwf    DIR_SRAM_L            ; el tipo de particion  dividimos por 2, ya
165                                    ; que tenemos 2 BYTES por direccion
166     movlw    PEOFFSET_PartType>>9  ; Parte alta del offset del byte que nos dice
167     movwf    DIR_SRAM_H            ; el tipo de particiondividimos igualmente por
168                                    ; 2 y luego desplazamos 8 posiciones
169     bcf      INTCON,GIE
170     movlw    LATCH_MEM_L           ; Activamos LATCH_MEM_L
171     movwf    CONTROL1
172     nop                            ; Esperamos
173     movlw    OCIOSO                ; Desactivamos LATCH_MEM_L
174     movwf    CONTROL1
175     bsf      INTCON,GIE

177     call     BUS_ConfLec           ; Configuramos para leer la MEMORIA SRAM

179     bcf      INTCON,GIE
180     movlw    OE                    ; Leemos la memoria SRam (OE)
181     movwf    CONTROL1
182     movf     DATOS_BAJOS,W
183     movwf    TempFAT0
184     movlw    OCIOSO                ; Deactivamos OE
185     movwf    CONTROL1
186     bsf      INTCON,GIE

188     movf     TempFAT0,W
189     xorlw    0x0C                  ; Comprobamos si es FAT32LBA
190     btfsc    STATUS,Z
191     goto     FAT32_LBA
192     movf     TempFAT0,W
193     xorlw    0x0B
194     btfsc    STATUS,Z
195     goto     FAT32_LBA

197     bsf      PCLATH,3
198     clrw
199     call     LCD_Linea3
200     movlw    b'00001000'           ; No es Fat32, lo indicamos y acabamos
201     movwf    Mensaje
202     call     LCD_PonMensaje
203     bcf      PCLATH,3
204     return

206  FAT32_LBA                         ; El sistema de ficheros es FAT32LBA
207     bsf      PCLATH,3
208     clrw                           ; Ponemos el cursor al comienzo de la Linea 1
209     call     LCD_Linea3

211     movlw    b'00010000'           ; Indicamos que mensaje vamos a pintar
212     movwf    Mensaje

214     call     LCD_PonMensaje
215     bcf      PCLATH,3
216  ; ******************************************************************************
```

```
218  ; ************************** CARGAMOS EL BOOT SECTOR *************************

220     movlw    PEOFFSET_StartLBA>>1 ; Parte baja del offset del byte que nos dice
221     movwf    SRam_Lo              ; la distancia en sectores del primer sector de
222                                   ; la particion. Dividimos por 2, ya que tenemos
223                                   ; 2 BYTES por direccion de memoria SRAM
224     movlw    PEOFFSET_StartLBA>>9 ; Parte alta del offset del byte que nos dice
225     movwf    SRam_Hi              ; la distancia en sectores del primer sector de
226                                   ; la particion. Dividimos igualmente por 2 y
227                                   ; luego desplazamos 8 posiciones
228     movlw    FAT_Inicio0
229     movwf    FSR                  ; Hacemos que FSR apunte a la zona de memoria
230                                   ; de las variables FAT_Inicio
231     call     Lee_4BYTES           ; Introducimos en estas variables el numero de
232                                   ; sectores que hay hasta el primer secror LBA
233                                   ; Leemos el primer sector LBA -> Primer sector
234                                   ; de la particion -> BOOT SECTOR
235     movf     FAT_Inicio0,W
236     movwf    IDE_Sec0
237     movf     FAT_Inicio1,W
238     movwf    IDE_Sec1
239     movf     FAT_Inicio2,W
240     movwf    IDE_Sec2
241     movf     FAT_Inicio3,W
242     movwf    IDE_Sec3
243     movlw    d'1'                 ; Indicamos que queremos leer 1 sector
244     call     IDE_LeerSectores     ; Leemos el sector

246     movlw    0x00                 ; Cargamos la direccion de destino en la SRAM
247     movwf    SRam_Hi
248     movlw    0x00
249     movwf    SRam_Lo
250     movlw    d'1'                 ; Indicamos que queremos transferir un sector
251     call     IDE_TransSectaSRam   ; Transferimos el sector a la SRAM

253  ;*****************************************************************************

255  ;*****************************************************************************
256  ;* Una vez que el primer sector de la particion esta en la SRam              *
257  ;* (0x0000..0x0100) hay que  extraer la informacion relevante: RsvdSecCnt,    *
258  ;* FATSz32, BytsPerSec, SecCluster.                                          *
259  ;*                                                                           *
260  ;* Obtencion de la direccion del AREA DE DATOS                               *
261  ;* ------------------------------------------------------------------------- *
262  ;* OFFSET                                        DESCRIPCCION                *
263  ;* ------------------------------------------------------------------------- *
264  ;* Comienzo de la Particion                      Boot Sector                 *
265  ;* -> StartLBA                                                               *
266  ;* ------------------------------------------------------------------------- *
267  ;* Comienzo + Numero de Sectores Reservados      Tablas FAT                  *
268  ;* -> StartLBA + RsvdSecCnt                      FAT_Inicio0..FAT_Inicio3    *
269  ;* ------------------------------------------------------------------------- *
270  ;* Comienzo + Numero de Sectores Reservados +    Area de Datos (Cluster #2)  *
271  ;* + (Numero de Sectores por FAT * 2)                                        *
272  ;* -> StartLBA + RsvdSecCnt + (FATSz32 * 2)      Datos_Inicio0..Datos_Inicio3 *
273  ;* ------------------------------------------------------------------------- *
274  ;*                                                                           *
275  ;*****************************************************************************

277     call     BUS_ConfEsc          ; Configuramos para escribir el LATCH MEMORIA
278                                   ; Obtenemos primero el numero de sectores por
279                                   ; cluster (1 BYTE)
280     movlw    BROFFSET_SecPerClus>>1
281     movwf    DIR_SRAM_L           ; Dividimos por 2, ya que tenemos 2 BYTES por
282     clrw                          ; direccion
283     movwf    DIR_SRAM_H

285     bcf      INTCON,GIE
286     movlw    LATCH_MEM_L          ; Activamos LATCH_MEM_L
287     movwf    CONTROL1
288     nop                           ; Esperamos
289     movlw    OCIOSO               ; Desactivamos LATCH_MEM_L
290     movwf    CONTROL1
```

```
291     call      BUS_ConfLec           ; Configuramos el BUS para leer la MEMORIA SRAM
292     movlw     OE                    ; Leemos la memoria SRam (OE)
293     movwf     CONTROL1
294     movf      DATOS_ALTOS,W
295     movwf     SecCluster
296     movlw     OCIOSO                ; Deactivamos OE
297     movwf     CONTROL1
298     bsf       INTCON,GIE
299 ;    call      BUS_ConfEsc
300                                     ; Ahora obtenemos el cluster del directorio
301                                     ; raiz(4 BYTES)
302     movlw     BROFFSET_RootClus>>1
303     movwf     SRam_Lo               ; dividimos por 2, ya que tenemos 2 BYTES por
304     clrw                            ; direccion
305     movwf     SRam_Hi

307     movlw     Cluster_Raiz0
308     movwf     FSR
309     call      Lee_4BYTES
310                                     ; Vamos a sumar: FAT_Inicio3:FAT_Inicio2:...
311                                     ; ...FAT_Inicio1:FAT_Inicio0 +
312                                     ; + RsvdSecCnt[2 BYTES]
313                                     ; Pasamos primero el numero de sectores
314                                     ; reservados (RsvdSecCnt) desde la SRam a
315                                     ; IDE_Sec3..IDE_Sec0 que seran usados como
316                                     ; registros temporales

318     movlw     BROFFSET_RsvdSecCnt>>1
319     movwf     SRam_Lo               ; Dividimos por 2, ya que tenemos 2 BYTES por
320     clrw                            ; direccion
321     movwf     SRam_Hi

323     movlw     IDE_Sec0
324     movwf     FSR
325     call      Lee_4BYTES

327     clrf      IDE_Sec2              ; Como solo eran 2 bytes, borramos IDE_Sec2 y
328     clrf      IDE_Sec3              ; IDE_Sec3  que tienen valores erroneos

330     movf      IDE_Sec0,W            ; IDE_Sec0 + FAT_Inicio0 --> FAT_Inicio0
331     addwf     FAT_Inicio0,F
332     movlw     d'1'                  ; Metemos en W el posible acarreo
333     btfsc     STATUS,C              ; Comprobamos si hay acarreo
334     addwf     FAT_Inicio1,F         ; Hay acarreo, sumamos 1 a FAT_Inicio1
335     btfsc     STATUS,C              ; Comprobamos si hay acarreo
336     addwf     FAT_Inicio2,F         ; Hay acarreo, sumamos 1 a FAT_Inicio2
337     btfsc     STATUS,C              ; Comprobamos si hay acarreo
338     addwf     FAT_Inicio3,F         ; Hay acarreo, sumamos 1 a FAT_Inicio3
339     bcf       STATUS,C              ; Borramos el posible acarreo

341     movf      IDE_Sec1,W            ; IDE_Sec1 + FAT_Inicio1 --> FAT_Inicio1
342     addwf     FAT_Inicio1,F
343     movlw     d'1'                  ; Metemos en W el posible acarreo
344     btfsc     STATUS,C              ; Comprobamos si hay acarreo
345     addwf     FAT_Inicio2,F         ; Hay acarreo, sumamos 1 a FAT_Inicio2
346     btfsc     STATUS,C              ; Comprobamos si hay acarreo
347     addwf     FAT_Inicio3,F         ; Hay acarreo, sumamos 1 a FAT_Inicio3

349                                     ; A partir de ahora FAT_Inicio3..FAT_Inicio0
350                                     ; contendran el numero del primer sector de la
351                                     ; FAT (Tablas Fat).
352                                     ; Ahora hay que calcular el primer sector de la
353                                     ; zona de  datos.
354                                     ; Primero pasamos desde la SRam a
355                                     ; Datos_Inicio0...Datos_Inicio3 el numero de
356                                     ; sectores de la FAT (FATSz32)

358     movlw     BROFFSET_FATSz32>>1
359     movwf     SRam_Lo               ; Dividimos por 2, ya que tenemos 2 BYTES por
360     clrw                            ; direccion
361     movwf     SRam_Hi

363     movlw     Datos_Inicio0
```

```
364     movwf     FSR
365     call      Lee_4BYTES

367                                       ; Ahora multiplicamos por 2 el numero de
368                                       ; sectores de la FAT (ya que esta duplicada)
369     bcf       STATUS,C                ; Borramos el bit de acarreo (rotacion)
370     rlf       Datos_Inicio0,F         ; Rotamos una vez todos los registros, el bit
371     rlf       Datos_Inicio1,F         ; que sale de uno, entra por el siguiente a
372     rlf       Datos_Inicio2,F         ; traves del bit C
373     rlf       Datos_Inicio3,F

375     bcf       STATUS,C                ; Por ultimo sumamos FAT_Inicio0..FAT_Inicio3 +
376                                       ; Datos_Inicio0..Datos_Inicio3
377     movf      FAT_Inicio0,W           ; FAT_Inicio0 + Datos_Inicio0 --> Datos_Inicio0
378     addwf     Datos_Inicio0,F
379     movlw     d'1'                    ; Metemos el posible acarreo en W
380     btfsc     STATUS,C                ; Comprobamos si hay acarreo
381     addwf     Datos_Inicio1,F         ; Hay acarreo, sumamos 1 a Datos_Inicio1
382     btfsc     STATUS,C                ; Comprobamos si hay acarreo
383     addwf     Datos_Inicio2,F         ; Hay acarreo, sumamos 1 a Datos_Inicio2
384     btfsc     STATUS,C                ; Comprobamos si hay acarreo
385     addwf     Datos_Inicio3,F         ; Hay acarreo, sumamos 1 a Datos_Inicio3

387     bcf       STATUS,C                ; Borramos el acarreo
388     movf      FAT_Inicio1,W           ; FAT_Inicio1 + Datos_Inicio1 --> Datos_Inicio1
389     addwf     Datos_Inicio1,F
390     movlw     d'1'                    ; Metemos el posible acarreo en W
391     btfsc     STATUS,C                ; Comprobamos si hay acarreo
392     addwf     Datos_Inicio2,F         ; Hay acarreo, sumamos 1 a Datos_Inicio2
393     btfsc     STATUS,C                ; Comprobamos si hay acarreo
394     addwf     Datos_Inicio3,F         ; Hay acarreo, sumamos 1 a Datos_Inicio3

396     bcf       STATUS,C                ; Borramos el acarreo
397     movf      FAT_Inicio2,W           ; FAT_Inicio2 + Datos_Inicio2 --> Datos_Inicio2
398     addwf     Datos_Inicio2,F
399     movlw     d'1'                    ; Metemos el posible acarreo en W
400     btfsc     STATUS,C                ; Comprobamos si hay acarreo
401     addwf     Datos_Inicio3,F         ; Hay acarreo, sumamos 1 a Datos_Inicio3

403     bcf       STATUS,C                ; Borramos el acarreo
404     movf      FAT_Inicio3,W           ; FAT_Inicio3 + Datos_Inicio3 --> Datos_Inicio3
405     addwf     Datos_Inicio3,F

407 ;*****************************************************************************
408 ;* Cuando busquemos una entrada en la FAT, esta nos dira en que cluster      *
409 ;* comienza, nosotros lo que queremos saber es en que sector comienza. Como  *
410 ;* sabemos en que sector comienza el Area de Datos y tambien sabemos el numero *
411 ;* de Sectores por cluster, tan solo deberemos hacer: Datos_Inicio +         *
412 ;* + (N°Cluster * SecCluster). Sin embargo esto solo seria correcto si   el  *
413 ;* primer cluster fuese el 0, como es el 2 tendremos que restar 2 al N° de   *
414 ;* cluster siempre, para evitarlo restamos ahora (2 * SecCluster) a          *
415 ;* Datos_Inicio.                                                             *
416 ;*****************************************************************************

418     bcf       STATUS,C                ; Borramos C (rotacion)
419     rlf       SecCluster,W            ; Multiplicamos los sectores por cluster por 2
420     subwf     Datos_Inicio0,F         ; Calculamos el sector de comienzo de los datos
421                                       ; menos 2
422     btfss     STATUS,C                ; Comprobamos si hay acarreo
423     subwf     Datos_Inicio1,F         ; Hay acarreo, restamos 1 a Datos_Inicio1
424     btfss     STATUS,C                ; Comprobamos si hay acarreo
425     subwf     Datos_Inicio2,F         ; Hay acarreo, restamos 1 a Datos_Inicio2
426     btfss     STATUS,C                ; Comprobamos si hay acarreo
427     subwf     Datos_Inicio3,F         ; Hay acarreo, restamos 1 a Datos_Inicio3

429 ;*****************************************************************************
430 ;*                                                                           *
431 ;* Datos_Inicio3..Datos_Inicio0 ahora contiene el numero del primer sector del *
432 ;* Area de  Datos menos 2.                                                   *
433 ;* Para calcular el sector en el que comienza un Cluster dado usaremos:      *
434 ;*                                                                           *
435 ;*     Datos_Inicio3..Datos_Inicio0 + (N°Cluster * SecCluster)               *
436 ;*                                                                           *
```

```
437  ;* Como el numero de sectores por cluster (SecCluster) siempre sera una        *
438  ;* potencia de 2, podemos hacer uso de las rotaciones para hacer el producto   *
439  ;* de forma rapida:                                                            *
440  ;*                                                                             *
441  ;* Si SecCluster es:                                                           *
442  ;*                                                                             *
443  ;*      2 entonces FactorRol es  1   |   2 -> 00000010                         *
444  ;*      4 entonces FactorRol es  2   |   4 -> 00000100                         *
445  ;*      8 entonces FactorRol es  3   |   8 -> 00001000                         *
446  ;*     16 entonces FactorRol es  4   |  16 -> 00010000                         *
447  ;*     32 entonces FactorRol es  5   |  32 -> 00100000                         *
448  ;*     64 entonces FactorRol es  6   |  64 -> 01000000                         *
449  ;*                                                                             *
450  ;* La formula que ahora nos da el Numero de Sector a partir del Numero de      *
451  ;* Cluster es:                                                                 *
452  ;*                                                                             *
453  ;* (N°Cluster << FactorRol) + Datos_Inicio3..Datos_Inicio0                     *
454  ;*                                                                             *
455  ;*******************************************************************************

457                                  ; Calculamos el factor de desplazamiento
458      movf     SecCluster,W        ; Ponemos el numero de sectores por cluster en
459      movwf    TempFAT0            ; un registro temporal
460      clrf     FactorRol           ; Ponemos el factor a 0

462  Otro0
463      rrf      TempFAT0,F          ; El numero de desplazamientos posibles de
464      incf     FactorRol,F         ; SecCluster hasta que el 1 llegue al BIT 0 es
465      btfss    TempFAT0,0          ; igual al factor de desplazamiento
466      goto     Otro0

468      clrw
469      return

471  ;*******************************************************************************
472  ;*                                                                             *
473  ;*                              Calcula_SectorLBA                              *
474  ;*                                                                             *
475  ;*******************************************************************************
476  ;*                                                                             *
477  ;* A partir del numero de cluster calcula el numero de sector que ha de leerse *
478  ;* del HD mediante la aplicacion de la formula:                                *
479  ;*                                                                             *
480  ;*     (N°Cluster << FactorRol) + Datos_Inicio3..Datos_Inicio0                 *
481  ;*                                                                             *
482  ;* Deja el resultado en IDE_Sec0..IDE_Sec3                                     *
483  ;*                                                                             *
484  ;*******************************************************************************

486  Calcula_SectorLBA

488     movf      Cluster_Actual0,W   ; Pasamos el numero de cluster a las variables
489     movwf     IDE_Sec0            ; de numero de sector
490     movf      Cluster_Actual1,W
491     movwf     IDE_Sec1
492     movf      Cluster_Actual2,W
493     movwf     IDE_Sec2
494     movf      Cluster_Actual3,W
495     movwf     IDE_Sec3


498     movf      FactorRol,W         ; Multiplicamos el numero de cluster por el
499     movwf     ContadorFAT0        ; numero de sectores por cluster

501  rota_otra
502     bcf       STATUS,C
503     rlf       IDE_Sec0,F
504     rlf       IDE_Sec1,F
505     rlf       IDE_Sec2,F
506     rlf       IDE_Sec3,F
507     decfsz    ContadorFAT0,F
508     goto      rota_otra
```

```
510     movf      Datos_Inicio0,W       ; Le sumamos el sector de inicio de la zona de
511     addwf     IDE_Sec0,F            ; datos
512     movlw     d'1'                  ; Ponemos en W el posible acarreo
513     btfsc     STATUS,C
514     addwf     IDE_Sec1,F
515     btfsc     STATUS,C
516     addwf     IDE_Sec2,F
517     btfsc     STATUS,C
518     addwf     IDE_Sec3,F

520     bcf       STATUS,C
521     movf      Datos_Inicio1,W
522     addwf     IDE_Sec1,F
523     movlw     d'1'
524     btfsc     STATUS,C
525     addwf     IDE_Sec2,F
526     btfsc     STATUS,C
527     addwf     IDE_Sec3,F

529     bcf       STATUS,C
530     movf      Datos_Inicio2,W
531     addwf     IDE_Sec2,F
532     movlw     d'1'
533     btfsc     STATUS,C
534     addwf     IDE_Sec3,F

536     bcf       STATUS,C
537     movf      Datos_Inicio3,W
538     addwf     IDE_Sec3,F
539     return

541 ;*****************************************************************************
542 ;*                                                                           *
543 ;*                                CargaEntradaFAT32                          *
544 ;*                                                                           *
545 ;*****************************************************************************
546 ;*                                                                           *
547 ;* Lee una entrada completa FAT32 del sector transferido a la SRAM y la      *
548 ;* transfiere a la memoria del PIC a partir de la direccion ENTRADA_FAT32    *
549 ;* El numero de entrada que se transfiere del sector (0 a 15) estara         *
550 ;* previamente indicado en la variable NumEntradaFAT32                      *
551 ;*                                                                           *
552 ;* Para calcular la posicion en memoria SRam en la que se encuentra la entrada *
553 ;* que se quiere transferir hacemos:                                         *
554 ;*                                                                           *
555 ;* DirComienzoSector + (16 * NumEntrada)                                     *
556 ;*                                                                           *
557 ;* En nuestro caso el sector se encuentra situado en la memoria SRam a partir *
558 ;* de la direccion 0x0000.                                                   *
559 ;* Multiplicamos por 16 y no por 32 ya que en cada posicion de memoria       *
560 ;* almacenamos 2 BYTES.                                                      *
561 ;*                                                                           *
562 ;*****************************************************************************

564 CargaEntradaFAT32
565     movf      NumEntradaFAT32,W
566     movwf     ContadorFAT0
567     clrf      SRam_Hi
568     clrf      SRam_Lo

570 Bucle_Entrada
571     movf      ContadorFAT0,F
572     btfsc     STATUS,Z
573     goto      Carga
574     movlw     d'16'
575     addwf     SRam_Lo,F
576     decf      ContadorFAT0,F
577     goto      Bucle_Entrada

579 Carga
580     call      BUS_ConfEsc           ; Vamos a escribir el LATCH de MEMORIA
581     movf      SRam_Hi,W
582     movwf     DirMem_H
```

```
583     movwf     DIR_SRAM_H
584     movf      SRam_Lo,W
585     movwf     DirMem_L
586     movwf     DIR_SRAM_L

588     bcf       INTCON,GIE
589     movlw     LATCH_MEM_L          ; Activamos LATCH_MEM_L
590     movwf     CONTROL1
591     nop                            ; Esperamos
592     movlw     OCIOSO               ; Desactivamos LATCH_MEM_L
593     movwf     CONTROL1
594     bsf       INTCON,GIE

596     movlw     d'16'                ; Leemos 16 WORDS
597     movwf     ContadorFAT0
598     movlw     ENTRADA_FAT32
599     movwf     FSR
600     call      BUS_ConfLec          ; Configuramos el BUS para leer la MEMORIA SRAM

602 Otra_Word
603     bcf       INTCON,GIE
604     movlw     OE                   ; Leemo la memoria SRam (OE)
605     movwf     CONTROL1
606     movf      DATOS_BAJOS,W
607     movwf     INDF
608     incf      FSR,F
609     movf      DATOS_ALTOS,W
610     movwf     INDF
611     incf      FSR,F
612     movlw     OCIOSO               ; Deactivamos OE
613     movwf     CONTROL1
614     bsf       CONTROL2,INC_ADDR    ; Incrementamos la direccion de origen
615     nop
616     bcf       CONTROL2,INC_ADDR
617     incf      DirMem_L,F
618     btfsc     STATUS,Z
619     incf      DirMem_H,F
620     bsf       INTCON,GIE
621     decfsz    ContadorFAT0,F
622     goto      Otra_Word
623 ;    call      BUS_ConfEsc
624     return

626 ;*****************************************************************************
627 ;*                                                                           *
628 ;*                                ProcesaUnicode                             *
629 ;*                                                                           *
630 ;*****************************************************************************
631 ;*                                                                           *
632 ;* Procesa un numero determinado (se le pasa por W) de caracteres UNICODE    *
633 ;* apuntados por FSR y los almacena en la memoria SRam.                      *
634 ;*                                                                           *
635 ;*****************************************************************************

637 ProcesaUnicode
638     movwf     ContadorFAT0
639     call      BUS_ConfEsc          ; Vamos a escribir en la MEMORIA SRAM

641 Bucle_Unicode
642     movf      INDF,W
643     movwf     DATOS_BAJOS
644     clrf      DATOS_ALTOS

646     bcf       INTCON,GIE
647     bcf       CONTROL2,WE          ; Escribimos en la SRAM
648     nop
649     bsf       CONTROL2,WE
650     incf      FSR,F
651     incf      FSR,F
652     bsf       CONTROL2,INC_ADDR    ; Incrementamos la direccion de destino
653     nop
654     bcf       CONTROL2,INC_ADDR
655     incf      DirMem_L,F
```

```
656     btfsc     STATUS,Z
657     incf      DirMem_H,F
658     bsf       INTCON,GIE
659     decfsz    ContadorFAT0,F
660     goto      Bucle_Unicode
661     return

663 ;*****************************************************************************
664 ;*                                                                           *
665 ;*                              ExtraeNL_Entrada                             *
666 ;*                                                                           *
667 ;*****************************************************************************
668 ;*                                                                           *
669 ;* Extrae una parte del nombre del fichero o directorio de una entrada de    *
670 ;* nombre largo y lo almacena en la memoria SRam.                            *
671 ;* Testea el BYTE EstadoBusqueda para decidir el destino del nombre en la    *
672 ;* SRam en funcion de si es una entrada de:                                  *
673 ;* Directorio -> EstadoBusqueda[1] = 0                                       *
674 ;* Fichero ->    EstadoBusqueda[1] = 1                                       *
675 ;*                                                                           *
676 ;*****************************************************************************

678 ; Hacemos el calculo:
679 ; I_NombreLargo <- (I_NombreLargo - 1) * 13

681 ; Los caracteres de LONG_NAME estan almacenados en grupos
682 ; de 13 pero los grupos de 13 estan almacenados de ultimo
683 ; a primero.
684 ; Con este calculo calculamos la posicion en la que ha de
685 ; ir el primer caracter de cada grupo en funcion del
686 ; indice del grupo

688 ExtraeNL_Entrada
689     decf      I_NombreLargo,F       ; I_NombreLargo <- I_NombreLargo -1

691     movf      I_NombreLargo,W
692     movwf     TempFAT0              ; TempFAT0 <- I_NombreLargo - 1
693     bcf       STATUS,C
694     rlf       TempFAT0,F
695     rlf       TempFAT0,F            ; TempFAT0 <- 4 * (I_NombreLargo - 1)
696     movf      TempFAT0,W
697     addwf     I_NombreLargo,F       ; I_NombreLargo <- 5 * (I_NombreLargo - 1)
698     bcf       STATUS,C
699     rlf       TempFAT0,F            ; TempFAT0 <- 8 * (I_NombreLargo - 1)
700     movf      TempFAT0,W
701     addwf     I_NombreLargo,F       ; I_NombreLargo <- 13 * (I_NombreLargo - 1)

703     call      BUS_ConfEsc           ; Vamos a escribir el LATCH de MEMORIA

705                                     ; Determinamos la direccion de destino SRAM en
706                                     ; funcion de si es el nombre de un FICHERO o de
707                                     ; un DIRECTORIO

709     btfss     ESTADO_BUSQUEDA,BUSCAR_FICH
710     goto      Busco_Directorio

712     movlw     (NOMBRE_FICHERO>>8)
713     goto      Extrae_Nombre

715 Busco_Directorio
716     movlw     (NOMBRE_DIRECTORIO>>8)

718 Extrae_Nombre
719     movwf     DirMem_H
720     movwf     DIR_SRAM_H
721     movf      I_NombreLargo,W
722     movwf     DirMem_L
723     movwf     DIR_SRAM_L

725     bcf       INTCON,GIE
726     movlw     LATCH_MEM_L           ; Activamos LATCH_MEM_L
727     movwf     CONTROL1
728     nop                             ; Esperamos
```

```
729     movlw     OCIOSO                  ; Desactivamos LATCH_MEM_L
730     movwf     CONTROL1
731     bsf       INTCON,GIE

733     movlw     (ENTRADA_FAT32)+LNOFFSETDIR_Name1   ; Apuntamos FSR al primer
734     movwf     FSR                                 ; carcter contenido en los
735     movlw     d'5'                                ; primeros 16 bytes de la
736                                                   ; entrada larga
737     call      ProcesaUnicode                      ; Copiamos parte del nombre (la
738                                                   ; mitad de una entrada larga)
739     movlw     (ENTRADA_FAT32)+LNOFFSETDIR_Name2   ; Apuntamos FSR al primer
740     movwf     FSR                                 ; caracter contenido en los
741     movlw     d'6'                                ; segundos 16 bytes de la
742                                                   ; entrada larga
743     call      ProcesaUnicode                      ; Copiamos parte del nombre (la
744                                                   ; mitad de una entrada larga)
745     movlw     (ENTRADA_FAT32)+LNOFFSETDIR_Name3   ; Apuntamos FSR a los dos
746     movwf     FSR                                 ; ultimos caracteres de los
747     movlw     d'2'                                ; segundos 16 bytes de la
748                                                   ; entrada larga
749     call      ProcesaUnicode                      ; Copiamos parte del nombre (los
750                                                   ; ultimos dos caracteres de una
751                                                   ; entrada larga)
752     movlw     d'13'                               ; Sumamos 13 a la longitud del
753     addwf     T_NombreLargo,F                     ; nombre largo
754     return

756 ;*****************************************************************************
757 ;*                                                                           *
758 ;*                              ExtraeNC_Entrada                             *
759 ;*                                                                           *
760 ;*****************************************************************************
761 ;*                                                                           *
762 ;* Extrae el nombre corto de una entrada y lo almacena en la memoria SRam en *
763 ;* el lugar que corresponda en funcion de si es un nombre de FICHERO o de   *
764 ;* DIRECTORIO.                                                               *
765 ;* Testea el BYTE EstadoBusqueda para decidir el destino del nombre en la SRam *
766 ;* en funcion de si es una entrada de:                                       *
767 ;*                                                                           *
768 ;* Directorio -> EstadoBusqueda[0] = 0                                       *
769 ;* Fichero ->    EstadoBusqueda[0] = 1                                       *
770 ;*****************************************************************************

772 ExtraeNC_Entrada
773     call      BUS_ConfEsc                 ; Vamos a escribir el LATCH de MEMORIA
774     btfss     ESTADO_BUSQUEDA,BUSCAR_FICH ; Miramos si estamos buscando ficheros
775     goto      Buscamos_Directorios

777 Buscamos_Ficheros
778     movlw     (NOMBRE_FICHERO>>8)         ; Parte alta de la direccion del
779                                           ; nombre del fichero en SRAM
780     goto      Carga_Direccion

782 Buscamos_Directorios
783     movlw     (NOMBRE_DIRECTORIO>>8)      ; Parte alta de la direccion del
784                                           ; nombre del directorio en SRAM
785 Carga_Direccion
786     movwf     DirMem_H
787     movwf     DIR_SRAM_H
788     clrw
789     movwf     DirMem_L
790     movwf     DIR_SRAM_L

792     bcf       INTCON,GIE
793     movlw     LATCH_MEM_L             ; Activamos LATCH_MEM_L
794     movwf     CONTROL1
795     nop                               ; Esperamos
796     movlw     OCIOSO                  ; Desactivamos LATCH_MEM_L
797     movwf     CONTROL1
798     bsf       INTCON,GIE

800     movlw     ENTRADA_FAT32           ; Apuntamos FSR al primer carcter del nombre
801     movwf     FSR                     ; corto
```

```
802     movlw     d'8'                  ; Extraemos los 8 primeros caracteres
803     movwf     ContadorFAT0

805 Nombre_Archivo
806     movf      INDF,W
807     movwf     DATOS_BAJOS
808     clrf      DATOS_ALTOS
809     bcf       INTCON,GIE
810     bcf       CONTROL2,WE           ; Escribimos en la SRAM
811     nop
812     bsf       CONTROL2,WE
813     bsf       CONTROL2,INC_ADDR     ; Incrementamos la direccion de destino
814     nop
815     bcf       CONTROL2,INC_ADDR
816     incf      DirMem_L,F
817     btfsc     STATUS,Z
818     incf      DirMem_H,F
819     bsf       INTCON,GIE
820     incf      FSR,F
821     decfsz    ContadorFAT0,F
822     goto      Nombre_Archivo

824     movlw     '.'                   ; Ponemos el punto
825     movwf     DATOS_BAJOS
826     clrf      DATOS_ALTOS
827     bcf       INTCON,GIE
828     bcf       CONTROL2,WE           ; Escribimos en la SRAM
829     nop
830     bsf       CONTROL2,WE
831     bsf       CONTROL2,INC_ADDR     ; Incrementamos la direccion de destino
832     nop
833     bcf       CONTROL2,INC_ADDR
834     incf      DirMem_L,F
835     btfsc     STATUS,Z
836     incf      DirMem_H,F
837     bsf       INTCON,GIE
838     movlw     d'3'                  ; Extraemos los 3 ultimos caracteres 'MP3'
839     movwf     ContadorFAT0

841 Extension_Archivo
842     movf      INDF,W
843     movwf     DATOS_BAJOS
844     clrf      DATOS_ALTOS
845     bcf       INTCON,GIE
846     bcf       CONTROL2,WE           ; Escribimos en la SRAM
847     nop
848     bsf       CONTROL2,WE
849     bsf       CONTROL2,INC_ADDR     ; Incrementamos la direccion de destino
850     nop
851     bcf       CONTROL2,INC_ADDR
852     incf      DirMem_L,F
853     btfsc     STATUS,Z
854     incf      DirMem_H,F
855     bsf       INTCON,GIE
856     incf      FSR,F
857     decfsz    ContadorFAT0,F
858     goto      Extension_Archivo

860     clrw                            ; Ponemos la marca de fin de nombre (0x0000)
861     movwf     DATOS_BAJOS
862     movwf     DATOS_ALTOS
863     bcf       INTCON,GIE
864     bcf       CONTROL2,WE           ; Escribimos en la SRAM
865     nop
866     bsf       CONTROL2,WE
867     bsf       INTCON,GIE
868     return

870 ;*****************************************************************************
871 ;*                                                                           *
872 ;*                                BorrarNombre                               *
873 ;*                                                                           *
874 ;*****************************************************************************
```

```
875  ;*                                                                      *
876  ;* Borra el nombre de la ultima entrada de directorio o fichero leida y  *
877  ;* almacenada en la SRam                                                *
878  ;*                                                                      *
879  ;************************************************************************

881  BorrarNombre
882     call      BUS_ConfEsc           ; Vamos a escribir el LATCH de MEMORIA

884     btfss     ESTADO_BUSQUEDA,BUSCAR_FICH
885     goto      Borro_Directorio

887     movlw     (NOMBRE_FICHERO>>8)
888     goto      Borra_Nombre

890  Borro_Directorio
891     movlw     (NOMBRE_DIRECTORIO>>8)

893  Borra_Nombre
894     movwf     DirMem_H
895     movwf     DIR_SRAM_H
896     clrw
897     movwf     DirMem_L
898     movwf     DIR_SRAM_L
899     bcf       INTCON,GIE
900     movlw     LATCH_MEM_L           ; Activamos LATCH_MEM_L
901     movwf     CONTROL1
902     nop                             ; Esperamos
903     movlw     OCIOSO                ; Desactivamos LATCH_MEM_L
904     movwf     CONTROL1
905     bsf       INTCON,GIE

907     movlw     d'127'
908     movwf     ContadorFAT0

910  Bucle_Borrar
911     clrf      DATOS_BAJOS
912     clrf      DATOS_ALTOS
913     bcf       INTCON,GIE
914     bcf       CONTROL2,WE           ; Escribimos en la SRAM
915     nop
916     bsf       CONTROL2,WE
917     bsf       CONTROL2,INC_ADDR     ; Incrementamos la direccion de destino
918     nop
919     bcf       CONTROL2,INC_ADDR
920     incf      DirMem_L,F
921     btfsc     STATUS,Z
922     incf      DirMem_H,F
923     bsf       INTCON,GIE
924     decfsz    ContadorFAT0,F
925     goto      Bucle_Borrar
926     return

928  ;************************************************************************
929  ;*                                                                      *
930  ;*                                ProcesarEntrada                       *
931  ;*                                                                      *
932  ;************************************************************************
933  ;*                                                                      *
934  ;*b                                                                 *
935  ;*                                                                      *
936  ;************************************************************************

938  ProcesarEntrada
939     bcf       ESTADO_BUSQUEDA,NC_ENCONTRADO ; Ponemos a 0 el FLAG de encontrado
940     movlw     ENTRADA_FAT32                 ; Hacemos que FSR apunte a la
941     movwf     FSR                           ; direccion que marca el primer byte
942                                             ; transferido
943     movf      INDF,W                        ; Obtenemos el primer byte de la
944                                             ; entrada del fichero
945     xorlw     DE_END                        ; Comprobamos si es la ultima entrada
946     btfss     STATUS,Z                      ; de directorio DE_END = 0x00
947     goto      NoEsLaUltima                  ; No es la ultima
```

```
948     bsf       ESTADO_BUSQUEDA,ULT_ENTRADA    ; Pone a 1 el FLAG que nos indica que
949     return                                   ; no hay mas directorios

951  NoEsLaUltima
952     movf      INDF,W                 ; Comprobamos si es una entrada vacia
953     xorlw     DE_FREE                ; DE_FREE = 0xE5
954     btfss     STATUS,Z
955     goto      NoEstaLibre            ; No esta vacia, la procesamos
956     clrf      T_NombreLargo          ; Borramos la variable de tamaño del nombre
957                                      ; largo
958     return

960  NoEstaLibre
961     movlw     (ENTRADA_FAT32)+DEOFFSET_Attr     ; Cargamos el byte de atributo
962     movwf     FSR
963     movf      INDF,W
964     andlw     b'00111111'                       ; Nos quedamos con 6 BITS
965     xorlw     ATTR_LONG_NAME                    ; Y comprobamos si es una entrada
966     btfss     STATUS,Z                          ; de nombre largo
967                                                 ; ATTR_LONG_NAME = h'0F'
968     goto      NoEsNombreLargo                   ; No es un nombre largo
969     movlw     (ENTRADA_FAT32)+LNOFFSETDIR_Ord   ; Cargamos el primer byte LDIR_Ord
970     movwf     FSR
971     movf      INDF,W
972     andlw     b'01000000'            ; Comprobamos si es el LAST_LONG_ENTRY = 0x4X
973     btfsc     STATUS,Z               ; testeando si el BIT 6 esta a 1
974     goto      No_Primera_Entrada

976  Primera_Entrada
977     clrf      T_NombreLargo          ; Si es borramos el campo de tamaño del nombre
978                                      ; largo, ya que es la primera entrada de nombre
979                                      ; largo
980     call      BorrarNombre           ; Borramos el nombre de la entrada anterior en
981                                      ; SRam

983  No_Primera_Entrada
984     movf      INDF,W                 ; Quitamos el indicador de LAST_LONG_ENTRY del
985     andlw     b'00111111'            ; indice
986     movwf     I_NombreLargo
987     call      ExtraeNL_Entrada       ; Obtenemos la parte del nombre largo contenida
988     return                           ; en la entrada y lo almacenamos en la SRAM


991  NoEsNombreLargo                     ; Hemos llegado a la SHORT_ENTRY asociada con
992                                      ; las LONG_ENTRY anteriores

994     btfsc     ESTADO_BUSQUEDA,BUSCAR_FICH    ; Comprobamos si estamos buscando
995                                              ; ficheros
996     goto      Comprueba_Fichero              ; Estamos buscando ficheros

998  Comprueba_Directorio
999     movf      INDF,W                 ; Enmascaramos el BIT de 'solo lectura'
1000    andlw     b'11111110'
1001    xorlw     ATTR_DIRECTORY         ; Comprobamos si es un directorio
1002    btfsc     STATUS,Z
1003    goto      CargarNumCluster
1004    return

1006 Comprueba_Fichero
1007    movf      INDF,W                 ; Comprobamos si es un archivo normal
1008    andlw     b'00011000'            ; Para ello los bits ATTR_VOLUME_ID y
1009    btfss     STATUS,Z               ; ATTR_DIRECTORY han de ser 0
1010    return

1012 Archivo_Normal                      ; Ahora comprobamos que la estension es "MP3"
1013    movlw     (ENTRADA_FAT32)+DEOFFSET_NameExt     ; Apuntamos al primer byte de
1014    movwf     FSR                                  ; extension
1015    movf      INDF,W
1016    xorlw     'M'
1017    btfss     STATUS,Z
1018    return                           ; En el momento que una de las letras falle,
1019                                     ; salimos
1020    incf      FSR,F
```

```
1021      movf      INDF,W
1022      xorlw     'P'
1023      btfss     STATUS,Z
1024      return

1026      incf      FSR,F
1027      movf      INDF,W
1028      xorlw     '3'
1029      btfss     STATUS,Z
1030      return

1032   Archivo_MP3                        ; El archivo es de tipo MP3, pasamos a obtener
1033                                      ; todos los datos de interes

1035      movlw     (ENTRADA_FAT32)+DEOFFSET_FileSize    ; Obtenemos el tamaño del
1036      movwf     FSR                                  ; fichero
1037      movf      INDF,W
1038      movwf     TamanoArchivo0
1039      incf      FSR,F
1040      movf      INDF,W
1041      movwf     TamanoArchivo1
1042      incf      FSR,F
1043      movf      INDF,W
1044      movwf     TamanoArchivo2
1045      incf      FSR,F
1046      movf      INDF,W
1047      movwf     TamanoArchivo3

1049   ;  movlw     d'2'                   ; Dividimos el tamaño del fichero entre 4 con
1050   ;  movwf     ContadorFAT0           ; dos rotaciones
1051   ;
1052   ;Dividir
1053   ;  bcf       STATUS,C
1054   ;  rrf       TamanoArchivo3,F
1055   ;  rrf       TamanoArchivo2,F
1056   ;  rrf       TamanoArchivo1,F
1057   ;  rrf       TamanoArchivo0,F
1058   ;  decfsz    ContadorFAT0,F
1059   ;  goto      Dividir

1061   CargarNumCluster
1062      movlw     (ENTRADA_FAT32)+DEOFFSET_FirstClusHi    ; Cargamos el valor del
1063      movwf     FSR                                     ; primer cluster del archivo
1064      movf      INDF,W                                  ; o directorio
1065      movwf     Cluster_Ent2
1066      incf      FSR,F
1067      movf      INDF,W
1068      movwf     Cluster_Ent3
1069      movlw     (ENTRADA_FAT32)+DEOFFSET_FirstClusLo
1070      movwf     FSR
1071      movf      INDF,W
1072      movwf     Cluster_Ent0
1073      incf      FSR,F
1074      movf      INDF,W
1075      movwf     Cluster_Ent1
1076      movf      T_NombreLargo,F                      ; Comprobamos si tenemos nombre
1077      btfss     STATUS,Z                             ; largo asociado o no
1078      goto      NombreLargo_Asociado

1080      call      ExtraeNC_Entrada                     ; Extraemos el Nombre Corto
1081      bsf       ESTADO_BUSQUEDA,NC_ENCONTRADO        ; Ponemos el FLAG de encontrado
1082      return

1084   NombreLargo_Asociado                              ; Tenemos nombre largo
1085      bsf       ESTADO_BUSQUEDA,NC_ENCONTRADO        ; Ponemos el FLAG de encontrado
1086      clrf      T_NombreLargo                        ; Borramos el contador de
1087                                                     ; longitud
1088      return

1090   ;*****************************************************************************
1091   ;*                                                                           *
1092   ;*                              LeerSigCluster                               *
1093   ;*                                                                           *
```

```
1094  ;*****************************************************************************
1095  ;*                                                                           *
1096  ;* Busca en el disco duro el siguiente cluster al actual y lo deja apuntado  *
1097  ;* por:                                                                      *
1098  ;*                                                                           *
1099  ;* Cluster_Actual0..Cluster_Actual4                                          *
1100  ;*                                                                           *
1101  ;*****************************************************************************

1103  ; En las variables FAT_Inicio0..FAT_Inicio3, tenemos almacenado el numero de
1104  ; Sector en el que comienzan las tablas FAT. Por otra parte tenemos almacenado
1105  ; en SRam, un bloque de estas tablas, concretamente 512 bytes es decir 128
1106  ; entradas. A la hora de buscar el siguiente cluster lo que tenemos que hacer es
1107  ; buscar en la tabla FAT la entrada correspondiente al cluster actual, su
1108  ; contenido es el numero del cluster siguiente.

1110  ; Como tenemos cargado una parte de la FAT en la SRam (para aumentar la
1111  ; velocidad), lo primero que haremos sera mirar a ver si la entrada buscada esta
1112  ; en la parte que tenemos en SRam, si esta la leemos y si no, cargamos la parte
1113  ; de las tablas FAT en la que este la entrada buscada que corresponda en SRam y
1114  ; la leemos.

1116  ; En la memoria SRAM tenemos almacenado un sector entero de tabla FAT a modo de
1117  ; cache. Un sector de disco duro son 512 BYTES, como cada entrada de la tabla
1118  ; FAT esta formada por 4 BYTES, tendremos un total de 128 entradas de la tabla
1119  ; FAT cacheadas.

1121  ; Lo primero que debemos hacer es comprobar si tenemos cargado el  bloque de FAT
1122  ; en el que se encuentra la entrada que buscamos o no. Para ello sabemos que
1123  ; cada entrada de la FAT es de 4 BYTES, que el disco esta dividido en sectores
1124  ; de 512 BYTES, luego en un sector tenemos 128 entradas de FAT. Sabemos el
1125  ; sector en el que comienza la FAT (FAT_Inicio) y por lo tanto, podemos saber el
1126  ; sector de FAT en el que se encuentra la entrada buscada.

1128  LeerSigCluster                    ; Vamos a realizar esta operacion:
1129     movf      Cluster_Actual0,W    ; IDE_Sec <- (Cluster_Actual/128) + FAT_Inicio
1130     movwf     IDE_Sec0             ; Esto nos dara el numero de sector
1131     movf      Cluster_Actual1,W    ; correspondiente al comienzo del bloque de FAT
1132     movwf     IDE_Sec1             ; en el que se encuentra la entrada buscada.
1133     movf      Cluster_Actual2,W
1134     movwf     IDE_Sec2
1135     movf      Cluster_Actual3,W
1136     movwf     IDE_Sec3

1138     movlw     d'7'                 ; Dividimos el cluster actual por 128
1139     movwf     ContadorFAT0

1141  Divide128                        ; 7 rotaciones a izquierda = Dividir por 128
1142     bcf       STATUS,C
1143     rrf       IDE_Sec3,F
1144     rrf       IDE_Sec2,F
1145     rrf       IDE_Sec1,F
1146     rrf       IDE_Sec0,F
1147     decfsz    ContadorFAT0,F
1148     goto      Divide128

1150     bcf       STATUS,C             ; Le sumamos FAT_Inicio
1151     movf      FAT_Inicio0,W
1152     addwf     IDE_Sec0,F
1153     movlw     d'1'
1154     btfsc     STATUS,C
1155     addwf     IDE_Sec1,F
1156     btfsc     STATUS,C
1157     addwf     IDE_Sec2,F
1158     btfsc     STATUS,C
1159     addwf     IDE_Sec3,F

1161     bcf       STATUS,C
1162     movf      FAT_Inicio1,W
1163     addwf     IDE_Sec1,F
1164     movlw     d'1'
1165     btfsc     STATUS,C
1166     addwf     IDE_Sec2,F
```

```
1167      btfsc     STATUS,C
1168      addwf     IDE_Sec3,F

1170      bcf       STATUS,C
1171      movf      FAT_Inicio2,W
1172      addwf     IDE_Sec2,F
1173      movlw     d'1'
1174      btfsc     STATUS,C
1175      addwf     IDE_Sec3,F

1177      bcf       STATUS,C
1178      movf      FAT_Inicio3,W
1179      addwf     IDE_Sec3,F
1180                                  ; Comprobamos si tenemos o no cargado el bloque
1181      movf      IDE_Sec0,W        ; de FAT en el que se encuentra nuestra entrada
1182      xorwf     FAT_Cacheado0,W
1183      btfss     STATUS,Z
1184      goto      CargarFAT
1185      movf      IDE_Sec1,W
1186      xorwf     FAT_Cacheado1,W
1187      btfss     STATUS,Z
1188      goto      CargarFAT
1189      movf      IDE_Sec2,W
1190      xorwf     FAT_Cacheado2,W
1191      btfss     STATUS,Z
1192      goto      CargarFAT
1193      movf      IDE_Sec3,W
1194      xorwf     FAT_Cacheado3,W
1195      btfsc     STATUS,Z
1196      goto      LeerFAT

1198  CargarFAT                       ; No tenemos cargado el bloque; lo cargamos
1199      movf      IDE_Sec0,W        ; Marcamos el "numero de bloque" que se va a
1200      movwf     FAT_Cacheado0     ; cachear para futuras comprobaciones
1201      movf      IDE_Sec1,W
1202      movwf     FAT_Cacheado1
1203      movf      IDE_Sec2,W
1204      movwf     FAT_Cacheado2
1205      movf      IDE_Sec3,W
1206      movwf     FAT_Cacheado3
1207      movlw     d'1'              ; Indicamos que queremos leer 1 sector
1208      call      IDE_LeerSectores

1210      movlw     (FAT_CACHEADA>>8) ; Ponemos la direccion en la que vamos a
1211      movwf     SRam_Hi           ; cachear el bloque FAT
1212      movlw     FAT_CACHEADA      ; Ponemos la direccion en la que vamos a
1213      movwf     SRam_Lo           ; cachear el bloque FAT
1214      movlw     d'1'
1215      call      IDE_TransSectaSRam ; Transferimos el sector a la SRAM

1217  LeerFAT                         ; Ya tenemos cargado el bloque
1218      clrf      SRam_Hi           ; Vamos a calcular el desplazamiento dentro del
1219                                  ; bloque cacheado en Sram_Hi:SRam_Lo
1220      movf      Cluster_Actual0,W ; El desplazamiento nos lo dan los 7 ultimos
1221                                  ; bits de Cluster_Actual0 --> 127
1222      andlw     b'01111111'
1223      movwf     SRam_Lo
1224                                  ; Como las entradas son de 4 bytes y la memoria
1225                                  ; tiene una anchura de 16 BITS, tenemos que
1226                                  ; multiplicar el desplazamiento por 2
1227      bcf       STATUS,C
1228      rlf       SRam_Lo,F
1229      rlf       SRam_Hi,F         ; Una vez calculado el desplazamiento, se lo
1230                                  ; sumamos al comienzo del bloque de FAT
1231                                  ; cacheado
1232      movlw     FAT_CACHEADA
1233      addwf     SRam_Lo,F
1234      btfsc     STATUS,C
1235      incf      SRam_Hi,F
1236      bcf       STATUS,C
1237      movlw     (FAT_CACHEADA>>8)
1238      addwf     SRam_Hi,F
```

```
1240      movlw     Cluster_Actual0       ; Leemos los 4 bytes de la entrada buscada y
1241      movwf     FSR                   ; los almacenamos en Cluster_Actual0:...
1242      call      Lee_4BYTES            ; ...Cluster_Actual3

1244                                      ; Comprobamos si es el ultimo cluster del disco
1245                                      ; duro
1246                                      ; El ultimo cluster del disco duro es el
1247                                      ; 0x0FFFFFFF

1249  ; Una vez obtenido el numero del siguiente Cluster, tenemos que comprobar que se
1250  ; trata de un numero valido

1252  ; Para ello debemos tener en cuenta que de los 32 Bits obtenidos, solo 28
1253  ; conforman el numero buscado, los 4 Bits de mayor peso son de uso reservado y
1254  ; no deben modificarse "nunca". Unicamente son puestos a 0 en el momento de
1255  ; formatear la unidad, debemos pues enmascararlos siempre.
1256  ; De esta forma los siguientes numeros leidos: 0x00000000, 0x10000000 o
1257  ; 0xF0000000, significan en realidad lo mismo "Cluster Libre", ya que debemos
1258  ; ignorar los 4 Bits de mayor peso a la hora de leer el numero de Cluster.
1259  ; De la misma forma, si el valor actual de un Cluster es 0x30000000 y queremos
1260  ; marcarlo como invalido (BAD CLUSTER) escribiendo 0x0FFFFFF7 en el, deberemos
1261  ; dejar finalmente un valor en la entrada correspondiente de 0x3FFFFFF7

1263  ; Numero leido FAT                 Significado
1264  ; ----------------                -----------
1265  ;
1266  ;    0x00000000     -------->     Libre (FREE)
1267  ;
1268  ;    0x0FFFFFF0 \
1269  ;     ::::::::   |  -------->     Reservados
1270  ;    0x0FFFFFF6 /
1271  ;
1272  ;    0x0FFFFFF7     -------->     Sector Erroneo, inusable (BAD SECTOR)
1273  ;
1274  ;    0x0FFFFFF8 \
1275  ;     ::::::::   |  -------->     Ultimo Cluster del Fichero / Directorio
1276  ;    0x0FFFFFFF /


1279  ;incfsz   Cluster_Actual0,W
1280  ;return

1282  ; Comprobamos si el Byte bajo se encuentra en el rango 0xF8..0xFF

1284      movlw     b'11111000'           ; La variacion 0xF8 a 0xFF nos la dan los 3
1285      andwf     Cluster_Actual0,W     ; Bits de menor peso, asi que los eliminamos
1286      xorlw     b'11111000'           ; Comprobamos los otros 5 Bits.
1287      btfss     STATUS,Z
1288      return                          ; No es una marca EOC (End Of Chain)

1290  ; Testeamos los valores de 0xFF

1292      incfsz    Cluster_Actual1,W     ; Si es 0xFF al incrementar en uno se pone en
1293                                      ; 0x00 y Z=1
1294      return                          ; No es una marca EOC (End Of Chain)

1296      incfsz    Cluster_Actual2,W     ; Si es 0xFF al incrementar en uno se pone en
1297                                      ; 0x00 y Z=1
1298      return                          ; No es una marca EOC (End Of Chain)

1300  ;movf     Cluster_Actual3,W
1301  ;xorlw      0x0F
1302  ;btfsc      STATUS,Z

1304  ; Enmascaramos el Byte alto y testeamos su valor

1306      movlw     0x0F
1307      andwf     Cluster_Actual3,F     ; Enmascaramos los Bits 28..31
1308      xorwf     Cluster_Actual3,W     ; Testeamos los Bits 24.27
1309      btfsc     STATUS,Z

1311  Ultimo_Cluster
1312      bsf       ESTADO_BUSQUEDA,ULT_CLUSTER
```

```
1313      return

1315  ;*****************************************************************************
1316  ;*                                                                           *
1317  ;*                                Buscar_Entrada                             *
1318  ;*                                                                           *
1319  ;*****************************************************************************
1320  ;*                                                                           *
1321  ;* Busca en el disco duro la entrada con el numero de orden contenido en:    *
1322  ;* EntBuscada_Hi:EntBuscada_Lo                                               *
1323  ;*                                                                           *
1324  ;*****************************************************************************

1326  Buscar_Entrada
1327     bcf       ESTADO_BUSQUEDA,ULT_ENTRADA    ; Borramos el FLAG de Ultima Entrada
1328     btfss     ESTADO_BUSQUEDA,BUSCAR_FICH    ; Miramos que buscamos, Fichero o
1329                                              ; Directorio
1330     goto      Busco_Directorios

1332  Busco_Ficheros                     ; Buscamos Ficheros, cargamos el directorio
1333     movlw     Cluster_Dir0           ; actual como origen de busqueda.
1334     movwf     FSR
1335     goto      Cargar_Cluster

1337  Busco_Directorios                  ; Buscamos Directorios, cargamos el directorio
1338     movlw     Cluster_Raiz0          ; raiz como origen de busqueda.
1339     movwf     FSR

1341  Cargar_Cluster                     ; Preparamos las variables para buscar el
1342     movf      INDF,W                 ; cluster
1343     movwf     Cluster_Actual0
1344     incf      FSR,F
1345     movf      INDF,W
1346     movwf     Cluster_Actual1
1347     incf      FSR,F
1348     movf      INDF,W
1349     movwf     Cluster_Actual2
1350     incf      FSR,F
1351     movf      INDF,W
1352     movwf     Cluster_Actual3

1354     bcf       ESTADO_BUSQUEDA,ULT_CLUSTER    ; Borramos el indicador de ultimo
1355                                              ; cluster
1356     clrf      EntEncontrada_Hi               ; Borramos los registros temporales de
1357     clrf      EntEncontrada_Lo               ; busqueda

1359  Leer_Cluster
1360     call      Calcula_SectorLBA      ; Calculamos el numero de Sector a leer
1361     movf      SecCluster,W           ; Ponemos el numero de sectores por cluster en
1362     movwf     SecPendientes          ; el contador de sectores pendientes
1363     call      IDE_LeerSectores       ; Leemos un cluster entero del HD

1365  Transfiere_Sector
1366     movlw     d'16'                  ; 1 Sector = 16 Entradas de 32 BYTES
1367     movwf     EntPendientes
1368     movlw     0x00                   ; Cargamos la direccion de destino en la SRAM
1369     movwf     SRam_Hi
1370     movlw     0x00
1371     movwf     SRam_Lo
1372     movlw     d'1'
1373     call      IDE_TransSectaSRam     ; Transferimos el sector a la SRAM
1374     decf      SecPendientes,F        ; Decrementamos en uno el numero de sectores
1375                                      ; pendientes de transferir del cluster
1376     clrw
1377     movwf     NumEntradaFAT32        ; Inicializamos el contador de entradas
1378                                      ; examinadas
1379     goto      Carga_Entrada          ; Cargamos y analizamos la primera entrada del
1380                                      ; sector

1382  Transfiere_Otro_Sector              ; Comprobamos si quedan sectores por transferir
1383     movf      SecPendientes,F
1384     btfss     STATUS,Z
1385     goto      Transfiere_Sector      ; Si que quedan, transferimos el siguiente
```

```
                                        ; No quedan, tenemos que leer el siguiente
                                        ; cluster

                                        ; Vamos a leer el siguiente cluster. Como
                                        ; estamos buscando entradas y no leyendo
                                        ; un fichero MP3, debemos de pasar como
                                        ; Cluster_Actual, el numero de Cluster del
                                        ; directorio en el que estamos buscando las
                                        ; entradas.
    call      LeerSigCluster
    btfss     ESTADO_BUSQUEDA,ULT_CLUSTER    ; Comprobamos si se acabaron los
                                             ; clusters
    goto      Leer_Cluster                   ; Quedan clusters, leemos el siguiente
    goto      Fin_Busqueda

Carga_Entrada
    call      CargaEntradaFAT32      ; Transfiere 32 bytes (1 entrada de directorio)
                                     ; de la  memoria SRam al PIC
    call      ProcesarEntrada        ; Procesamos la entrada de directorio leida

    btfsc     ESTADO_BUSQUEDA,ULT_ENTRADA       ; Comprobamos si hemos acabado ya c
                                                ; on todas las entradas
    goto      Fin_Busqueda                      ; Hemos acabado con todos las
                                                ; entradas
    btfss     ESTADO_BUSQUEDA,NC_ENCONTRADO     ; Testeamos si hemos encontrado una
                                                ; entrada corta
    goto      Siguiente_Entrada                 ; No hemos encontrado una entrada
                                                ; corta todavia

NC_Encontrado
    incf      EntEncontrada_Lo,F     ; Hemos encontrado una entrada corta, i
                                     ; ncrementamos el contador
    btfsc     STATUS,Z
    incf      EntEncontrada_Hi,F

    movf      EntBuscada_Lo,W        ; Comprobamos si es la entrada que buscabamos.
    xorwf     EntEncontrada_Lo,W     ; Parte 1.
    btfss     STATUS,Z
    goto      Siguiente_NC           ; No es la buscada, buscamos la siguiente
                                     ; entrada de Nombre Corto
    movf      EntBuscada_Hi,W        ; Comprobamos si es la entrada que buscabamos.
    xorwf     EntEncontrada_Hi,W     ; Parte 2.
    btfss     STATUS,Z
    goto      Siguiente_NC

Buscada_Encontrada
    btfss     ESTADO_BUSQUEDA,BUSCAR_FICH    ; Comprobamos si buscamos ficheros
    goto      CargarClust_Dir                ; Buscamos directorios

CargarClust_Fich
;   movlw     Cluster_Actual0        ; Buscamos ficheros, FSR apunta al CLUSTER
;   movwf     FSR                    ; actual
;   goto      Cargar_Cluster2
    goto      Fin_Busqueda

CargarClust_Dir
    movlw     Cluster_Dir0           ; FSR apunta al cluster de directorio actual
    movwf     FSR

Cargar_Cluster2
    movf      Cluster_Ent0,W
    movwf     INDF
    incf      FSR,F
    movf      Cluster_Ent1,W
    movwf     INDF
    incf      FSR,F
    movf      Cluster_Ent2,W
    movwf     INDF
    incf      FSR,F
    movf      Cluster_Ent3,W
    movwf     INDF

Fin_Busqueda
```

```
1459      return

1461   Siguiente_NC
1462      clrf      T_NombreLargo

1464   Siguiente_Entrada
1465      incf      NumEntradaFAT32,F          ; Incrementamos el contador de entradas
1466                                           ; examinadas
1467      decfsz    EntPendientes,F            ; Tenemos en principio 16 entradas de 32
1468                                           ; bytes en el sector leido del HD,
1469                                           ; decrementamos en 1 el numero de
1470                                           ; entradas que quedan por procesar antes
1471                                           ; de leer otro sectorentrada
1472      goto      Carga_Entrada              ; Cargamos la siguiente entrada
1473      goto      Transfiere_Otro_Sector     ; No hay mas entradas cargamos un nuevo
1474                                           ; sector

1476      VARIABLE       FileStartC1 = $
```

A.5. Manejo de Botones

Todas las funciones necesarias para recoger las pulsaciones de los botones del dispositivo se gestionan en esta parte del código.

Esta ubicado en la **Página 0** de la memoria de código.

```
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************
;***                                                                         ***
;**                              I_Botones.asm                              **
;**                                                                         **
;** Permite definir la forma en la que se accede al conjunto de 8 botones  **
;** para explorarlos y determinar si alguno de ellos se ha pulsado.         **
;** Incluye funciones para eliminar rebotes.                                **
;**                                                                         **
;***                                                                         ***
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************

Interfaz_Botones     EQU FileStartC1

;*******************************************************************************
;*******************************************************************************
;**                                                                         **
;**                            Funciones de acceso                          **
;**                                                                         **
;*******************************************************************************
;*******************************************************************************

   org      Interfaz_Botones

;*******************************************************************************
;*                                                                             *
;*                              Escanea_Botones                                *
;*                                                                             *
;*******************************************************************************
;*                                                                             *
;* Comprueba si se pulsa un boton (elimina rebotes) y devuelve el boton        *
;* pulsado en W.                                                               *
;*                                                                             *
;*******************************************************************************

Escanea_Botones

   btfss    INTerrupcion,BOTON    ; Comprobamos si podemos aceptar una pulsacion
   goto     Pulsacion_Desactivada

   bsf      STATUS,RP1            ; Banco 2
   movlw    d'20'
   movwf    IntBoton

   bcf      STATUS,RP1            ; Banco 0
   bcf      INTCON,GIE
   call     BUS_ConfLec           ; Configuramos el BUS para leer los BOTONES
   movlw    BUTTON_E              ; Activamos Button_E
   movwf    CONTROL1
   movf     DATOS_BAJOS,W
   movwf    Boton_B
   movlw    OCIOSO                ; No se pulso ningun boton
   movwf    CONTROL1              ; Desactivamos el MUX
   bsf      INTCON,GIE

   movlw    b'11111111'
   xorwf    Boton_B,W
   btfsc    STATUS,Z
   goto     BotonNoPulsado
```

```
64  BotonPulsado
65     bcf       INTerrupcion,BOTON    ; Boton pulsado, Inhibimos la pulsacion de otro

67  BotonNoPulsado
68     movf      Boton_B,W
69     return

71  Pulsacion_Desactivada
72     movlw     b'11111111'
73     return



77  ;*****************************************************************************
78  ;*                                                                           *
79  ;*                                QuitaRebotes                               *
80  ;*                                                                           *
81  ;*****************************************************************************
82  ;*                                                                           *
83  ;* Hace una espera para eliminar los rebotes                                 *
84  ;*                                                                           *
85  ;*****************************************************************************

87  QuitaRebotes

89     movlw     0x00
90     movwf     Tempo3

92  QR_Bucle
93     decfsz    Tempo3,F
94     goto      QR_Bucle
95     return

97  ;*****************************************************************************
98  ;*                                                                           *
99  ;*                                EsperaTecla                                *
100 ;*                                                                           *
101 ;*****************************************************************************
102 ;*                                                                           *
103 ;* Se detiene la ejecucion del programa hasta que se pulsa una tecla.        *
104 ;*                                                                           *
105 ;*****************************************************************************

107 EsperaTecla

109    call      Escanea_Botones       ; Escaneamos a ver si se ha pulsado un boton
110    xorlw     b'11111111'           ; Chequeamos el valor devuelto en W
111    btfsc     STATUS,Z              ; Si es un '0' no se ha pulsado ningun boton
112    goto      EsperaTecla           ; Volvemos a escanear los botones
113    return                          ; Salimos cuando W sea distinto de '0'

115    VARIABLE       FileStartC1 = $
```

A.6. Interfaz con el Display LCD

El display LCD elegido para mostrar la información al usuario, dispone de su propio microcontrolador *HITACHI HD44780U*. Esta parte del código es la encargada de establecer el diálogo entre el PIC y el display para inicializarlo y escribir en el.

La zona de la memoria de programa elegida para ubicar esta parte del código es él comienzo (*zona mas baja*) de la **Página 1**.

```
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************
;***                                                                         ***
;**                                I_LCD.asm                                 **
;**                                                                          **
;** Permite controlar un display LCD de cuatro lineas y 20 columnas,         **
;** compatible con el controlador HD44780U.                                  **
;** Permite realizar operaciones basicas con el, como son:                   **
;** - Inicializacion y configuracion.                                        **
;** - Posicionamiento del cursor en una linea determinada.                   **
;** - Posicionamiento del cursor en una posicion determinada.                **
;** - Escribir un caracter.                                                  **
;**                                                                          **
;***                                                                         ***
;*******************************************************************************
;*******************************************************************************
;*******************************************************************************

;*******************************************************************************
;*******************************************************************************
;**                                                                          **
;**                    Configuracion de la conexion LCD - PIC                **
;**                                                                          **
;*******************************************************************************
;*******************************************************************************

; Para controlar el LCD es necesario definir a que pines del PIC estan
; conectadas las siguientes lineas:
;
;     - Bus de Datos:                DATA0 - DATA7
;     - Seleccion de comando/dato:   RS
;     - Escritura/Lectura:           RW
;     - Habilitacion:                LCD_E

; Indica en que puerto has el BUS de datos. (DATA0 - DATA7)

LCD_DATOS       EQU PORTB
LCD_DATOS_IO    EQU TRISB

; Indica en que puerto y en que posicion has conectado la linea de seleccion de
; Comando/Dato. RS

LCD_RS_PORT     EQU PORTD
LCD_RS_IO       EQU TRISD
LCD_RS          EQU 0

; Indica en que puerto y en que posicion has conectado la linea de seleccion de
; Escritura/Lectura. RW

LCD_RW_PORT     EQU PORTD
LCD_RW_IO       EQU TRISD
LCD_RW          EQU 1

; La habilitacion del LCD corre a cargo de un decodificador controlado por el
; PIC, a continuacion puede verse en que puerto esta conectado el decodificador
; y los codigos de habilitacion y deshabilitacion del LCD.

; Puerto ->           CONTROL1
```

```
60  ; Habilitacion ->    LCD_E
61  ; Deshabilitacion -> OCIOSO

63  Interfaz_LCD      equ 0x0800

65  ;*****************************************************************************
66  ;*****************************************************************************
67  ;**                                                                         **
68  ;**                        Funciones de acceso LCD                          **
69  ;**                                                                         **
70  ;*****************************************************************************
71  ;*****************************************************************************

73     org      Interfaz_LCD

75  ;*****************************************************************************
76  ;*****************************************************************************
77  ;**                                                                         **
78  ;**                     Configuracion de puertos y PIC                      **
79  ;**                                                                         **
80  ;*****************************************************************************
81  ;*****************************************************************************

83  ;*****************************************************************************
84  ;*
85  ;*                              LCD_Conf... (BUS)
86  ;*
87  ;*****************************************************************************
88  ;*                                                                           *
89  ;* Configura el PIC para leer o escribir en el dispositivo LCD               *
90  ;* Realiza la misma funcion que BUS_ConfLec y BUS_ConfEsc, pero nos evita    *
91  ;* tener que cambiar de banco de codigo.                                     *
92  ;*                                                                           *
93  ;*****************************************************************************

95  LCD_ConfLec

97     movlw    0xFF                ; Cargamos 0xFF en W para configurar luego los
98                                  ; puertos como de entrada
99     goto     LCDComun            ; Saltamos a la zona comun

101 LCD_ConfEsc                     ; Configuracion para escribir
102    clrw                         ; Cargamos 0x00 en W para configurar luego los
103                                 ; puertos como de salida
104 LCDComun                        ; Zona de configuracion comun
105    bsf      STATUS,RP0          ; Banco 1
106    bcf      STATUS,RP1
107    movwf    TRISB               ; Configuramos DATA0-DATA7 como de entrada o
108                                 ; salida
109    movwf    TRISD               ; Configuramos DATA8-DATA15 como de entrada o
110                                 ; salida
111    bcf      STATUS,RP0          ; Banco 0
112    return

114 ;*****************************************************************************
115 ;*                                                                           *
116 ;*                              LCD_Conf... (C/D)                            *
117 ;*                                                                           *
118 ;*****************************************************************************
119 ;*                                                                           *
120 ;* Configura el PIC para leer o escribir comandos o datos en el dispositivo  *
121 ;* LCD                                                                       *
122 ;*                                                                           *
123 ;*****************************************************************************

125 LCD_ConfEscC

127    bsf      STATUS,RP0
128    clrf     LCD_RS_IO
129    clrf     LCD_RW_IO
130    bcf      STATUS,RP0
131    bcf      LCD_RS_PORT,LCD_RS  ; RS = 0 --> Comando
132    bcf      LCD_RW_PORT,LCD_RW  ; RW = 0 --> Escritura
```

```
133     goto      ComunCLL1

135 LCD_ConfEscD

137     bsf       STATUS,RP0
138     clrf      LCD_RS_IO
139     clrf      LCD_RW_IO
140     bcf       STATUS,RP0
141     bsf       LCD_RS_PORT,LCD_RS    ; RS = 1 --> Dato
142     bcf       LCD_RW_PORT,LCD_RW    ; RW = 0 --> Escritura

144 ComunCLL1

146     clrw                            ; DATA0-DATA7 = OUT
147     goto      ComunCLL

149 LCD_ConfLecC

151     bsf       STATUS,RP0
152     clrf      LCD_RS_IO
153     clrf      LCD_RW_IO
154     bcf       STATUS,RP0
155     bcf       LCD_RS_PORT,LCD_RS    ; RS = 0 --> Comando
156     bsf       LCD_RW_PORT,LCD_RW    ; RW = 1 --> Lectura
157     goto      ComunCLL2

159 LCD_ConfLecD

161     bsf       STATUS,RP0
162     clrf      LCD_RS_IO
163     clrf      LCD_RW_IO
164     bcf       STATUS,RP0
165     bsf       LCD_RS_PORT,LCD_RS    ; RS = 1 --> Dato
166     bsf       LCD_RW_PORT,LCD_RW    ; RW = 1 --> Lectura

168 ComunCLL2

170     movlw     0xFF                  ; DATA0-DATA7 = IN
171     goto      ComunCLL

173 ComunCLL

175     bsf       STATUS,RP0
176     movwf     LCD_DATOS_IO          ; Configuramos el PUERTO B como entrada o
177     bcf       STATUS,RP0            ; salida
178     return

180 ;*****************************************************************************
181 ;*****************************************************************************
182 ;**                                                                         **
183 ;**                              Manejo del LCD                             **
184 ;**                                                                         **
185 ;*****************************************************************************
186 ;*****************************************************************************

188 ;*****************************************************************************
189 ;*                                                                           *
190 ;*                                   DaLCD                                   *
191 ;*                                                                           *
192 ;*****************************************************************************
193 ;*                                                                           *
194 ;* Rutina de Control del LCD, manda el dato o el comando almacenado en W al  *
195 ;* LCD.                                                                      *
196 ;* Presupone que el BUS esta configurado para escribir en el.                *
197 ;* NO DEBE USARSE FUERA DE ESTE LIBRERIA.                                    *
198 ;*                                                                           *
199 ;*****************************************************************************

201 DaLCD
202     movwf     LCD_DATOS             ; Ponemos el dato en el puerto de Salida
203                                     ; DATA0-DATA7
204     bcf       INTCON,GIE
205     movlw     LCD_E                 ; Activamos LCD_E
```

```
206     movwf     CONTROL1
207     nop                           ; Esperamos
208     nop
209     movlw     OCIOSO
210     movwf     CONTROL1
211     bsf       INTCON,GIE

213     movlw     d'60'
214     movwf     Tempo3

216 Espera_SCC
217     decfsz    Tempo3,F
218     goto      Espera_SCC
219     return

221 ;*****************************************************************************
222 ;*                                                                           *
223 ;*                                EsperarLCDlisto                            *
224 ;*                                                                           *
225 ;*****************************************************************************
226 ;*                                                                           *
227 ;* Esperamos hasta que el LCD este listo para recibir comandos               *
228 ;*                                                                           *
229 ;*****************************************************************************

231 EsperarLCDlisto
232     movlw     OCIOSO              ; Desactivamos el MUX
233     movwf     CONTROL1
234     call      LCD_ConfLecC        ; Configuramos el PIC y el MUX para leer del
235                                   ; LCD
236     bcf       INTCON,GIE
237     movlw     LCD_E               ; Activamos LCD_E
238     movwf     CONTROL1
239     nop                           ; Esperamos

241 ELCD
242     btfsc     LCD_DATOS,7         ; Testeamos DATA7 = 0
243     goto      ELCD                ; DATA7 = 1 seguimos esperando
244     movlw     OCIOSO              ; Desactivamos LCD_E
245     movwf     CONTROL1
246     bsf       INTCON,GIE
247     return

249 ;*****************************************************************************
250 ;*                                                                           *
251 ;*                                LCD_EnviaDato                              *
252 ;*                                                                           *
253 ;*****************************************************************************
254 ;*                                                                           *
255 ;* Manda el dato que tengamos almacenado en W al LCD                         *
256 ;*                                                                           *
257 ;*****************************************************************************

259 LCD_EnviaDato

261     movwf     WBackup             ; Guardamos el dato de momento
262     call      EsperarLCDlisto     ; Esperamos a que el LCD este listo
263     call      LCD_ConfEscD        ; Configuramos el PIC y el MUX para escribir en
264                                   ; el LCD
265     movf      WBackup,W           ; Extraemos el dato
266     call      DaLCD               ; Enviamos el dato
267     return

269 ;*****************************************************************************
270 ;*                                                                           *
271 ;*                                InicializaLCD                              *
272 ;*                                                                           *
273 ;*****************************************************************************
274 ;*                                                                           *
275 ;* Inicializamos el LCD                                                      *
276 ;*                                                                           *
277 ;*****************************************************************************
```

```
279 InicializaLCD
280                                     ; Comienzo de la Inicializacion del LCD
281     call      LCD_ConfEscC          ; Configuramos el PIC para usar el LCD
282     movlw     d'15'                 ; Hacemos un espera de 15ms
283     bcf       PCLATH,3
284     call      Espera_C
285     bsf       PCLATH,3
286     movlw     0x30                  ; Function Set
287     call      DaLCD                 ; LCDFuncion 8 bits long (0011****)
288     movlw     d'5'                  ; Hacemos un espera de 5ms
289     bcf       PCLATH,3
290     call      Espera_C
291     bsf       PCLATH,3
292     movlw     0x30                  ; Function Set
293     call      DaLCD                 ; LCDFuncion 8 bits long (0011****)
294     movlw     d'1'                  ; Hacemos un espera de 1ms
295     bcf       PCLATH,3
296     call      Espera_C
297     bsf       PCLATH,3
298     movlw     0x30                  ; Function Set
299     call      DaLCD                 ; LCDFuncion 8 bits long (0011****)
300     movlw     d'1'                  ; Hacemos un espera de 1ms
301     bcf       PCLATH,3
302     call      Espera_C
303     bsf       PCLATH,3
304     movlw     0x38                  ; Function Set
305     call      DaLCD                 ; LCDFuncion (8 Bits, 2 Linea, font 5x8)
306     movlw     0x08                  ; Display Off, LCDOff
307     call      DaLCD
308     movlw     0x01                  ; Display Clear, LCD_Clear
309     call      DaLCD
310     movlw     d'1'                  ; Hacemos un espera de 1ms
311     bcf       PCLATH,3
312     call      Espera_C
313     bsf       PCLATH,3
314     movlw     0x06                  ; Entry Mode Set
315     call      DaLCD                 ; LCD_Inc (Inc. Posicion, No despl. Display)
316     movlw     d'1'                  ; Hacemos un espera de 1ms
317     bcf       PCLATH,3
318     call      Espera_C
319     bsf       PCLATH,3
320     movlw     0x0C                  ; CursOff
321     call      DaLCD
322     return                          ; Fin de la Inicializacion del LCD

324 ;*****************************************************************************
325 ;*                                                                           *
326 ;*                                LCD_Linea1                                 *
327 ;*                                                                           *
328 ;*****************************************************************************
329 ;*                                                                           *
330 ;* Pone en el cursor en la posicion marcada por W dentro de la primera linea *
331 ;*                                                                           *
332 ;*****************************************************************************

334 LCD_Linea1

336     movwf     WBackup               ; Guardamos el valor de la posicion de momento
337     call      EsperarLCDlisto       ; Esperamos a que el LCD este listo para recibi
338                                     ; un comando
339     call      LCD_ConfEscC          ; Configuramos el PIC y el MUX para escribir en
340                                     ; el LCD
341     movlw     0x80                  ; Comando --> Poner el cursor en el comienzo de
342                                     ; la primera linea
343     addwf     WBackup,W             ; Le sumamos el desplazamiento en la linea
344     call      DaLCD
345     return

347 ;*****************************************************************************
348 ;*                                                                           *
349 ;*                                LCD_Linea2                                 *
350 ;*                                                                           *
351 ;*****************************************************************************
```

```
352 ;*                                                                          *
353 ;* Pone en el cursor en la posicion marcada por W dentro de la segunda linea   *
354 ;*                                                                          *
355 ;****************************************************************************

357 LCD_Linea2

359     movwf     WBackup
360     call      EsperarLCDlisto
361     call      LCD_ConfEscC
362     movlw     0xC0
363     addwf     WBackup,W
364     call      DaLCD
365     return

367 ;****************************************************************************
368 ;*                                                                          *
369 ;*                                  LCD_Linea3                              *
370 ;*                                                                          *
371 ;****************************************************************************
372 ;*                                                                          *
373 ;* Pone en el cursor en la posicion marcada por W dentro de la tercera linea   *
374 ;*                                                                          *
375 ;****************************************************************************

377 LCD_Linea3

379     movwf     WBackup
380     call      EsperarLCDlisto
381     call      LCD_ConfEscC
382     movlw     0x94
383     addwf     WBackup,W
384     call      DaLCD
385     return

387 ;****************************************************************************
388 ;*                                                                          *
389 ;*                                  LCD_Linea4                              *
390 ;*                                                                          *
391 ;****************************************************************************
392 ;*                                                                          *
393 ;* Pone en el cursor en la posicion marcada por W dentro de la cuarta linea    *
394 ;*                                                                          *
395 ;****************************************************************************

397 LCD_Linea4

399     movwf     WBackup
400     call      EsperarLCDlisto
401     call      LCD_ConfEscC
402     movlw     0xD4
403     addwf     WBackup,W
404     call      DaLCD
405     return

407 ;****************************************************************************
408 ;*                                                                          *
409 ;*                                  BorraLinea                              *
410 ;*                                                                          *
411 ;****************************************************************************
412 ;*                                                                          *
413 ;* Borra la linea en la que esta el cursor, llenandola de espacios en blanco   *
414 ;*                                                                          *
415 ;****************************************************************************

417 BorraLinea
418     movlw     d'20'
419     movwf     ContadorLCD          ; Numero de posiciones a escribir

421 BucleBorra
422     movlw     '␣'
423     call      LCD_EnviaDato
424     decfsz    ContadorLCD,F
```

```
425      goto      BucleBorra
426      return

428   ;*****************************************************************************
429   ;*                                                                           *
430   ;*                                LCD_LimpiaLinea1                           *
431   ;*                                                                           *
432   ;*****************************************************************************
433   ;*                                                                           *
434   ;* Pone en el cursor al comienzo de la Linea 1, borrando previamente toda la *
435   ;* linea                                                                     *
436   ;*                                                                           *
437   ;*****************************************************************************

439   LCD_LimpiaLinea1
440      movwf     TempLCD                ; Guardamos el valor de W para luego
441                                       ; recuperarlo
442      clrw
443      call      LCD_Linea1             ; Nos ponemos al comienzo de la linea
444      call      BorraLinea             ; Borramos la linea
445      clrw                             ; Cursor al inicio de la linea
446      call      LCD_Linea1             ; Seleccionamos la linea y ponemos el cursor al
447                                       ; inicio
448      movf      TempLCD,W              ; Recuperamos W
449      return

451   ;*****************************************************************************
452   ;*                                                                           *
453   ;*                                LCD_LimpiaLinea2                           *
454   ;*                                                                           *
455   ;*****************************************************************************
456   ;*                                                                           *
457   ;* Pone en el cursor al comienzo de la Linea 2, borrando previamente toda la *
458   ;* linea                                                                     *
459   ;*                                                                           *
460   ;*****************************************************************************

462   LCD_LimpiaLinea2
463      movwf     TempLCD
464      clrw
465      call      LCD_Linea2
466      call      BorraLinea
467      clrw
468      call      LCD_Linea2
469      movf      TempLCD,W
470      return

472   ;*****************************************************************************
473   ;*                                                                           *
474   ;*                                LCD_LimpiaLinea3                           *
475   ;*                                                                           *
476   ;*****************************************************************************
477   ;*                                                                           *
478   ;* Pone en el cursor al comienzo de la Linea 3, borrando previamente toda la *
479   ;* linea                                                                     *
480   ;*                                                                           *
481   ;*****************************************************************************

483   LCD_LimpiaLinea3
484      movwf     TempLCD
485      clrw
486      call      LCD_Linea3
487      call      BorraLinea
488      clrw
489      call      LCD_Linea3
490      movf      TempLCD,W
491      return

493   ;*****************************************************************************
494   ;*                                                                           *
495   ;*                                LCD_LimpiaLinea4                           *
496   ;*                                                                           *
497   ;*****************************************************************************
```

```
498  ;*                                                                      *
499  ;* Pone en el cursor al comienzo de la Linea 4, borrando previamente toda la   *
500  ;* linea                                                                *
501  ;*                                                                      *
502  ;************************************************************************

504  LCD_LimpiaLinea4
505     movwf     TempLCD
506     clrw
507     call      LCD_Linea4
508     call      BorraLinea
509     clrw
510     call      LCD_Linea4
511     movf      TempLCD,W
512     return

514  ;************************************************************************
515  ;*                                                                      *
516  ;*                              LCD_PonMensaje                          *
517  ;*                                                                      *
518  ;************************************************************************
519  ;*                                                                      *
520  ;* Escribe en la pantalla el mensaje seleccionado                       *
521  ;*                                                                      *
522  ;************************************************************************

524  LCD_PonMensaje:
525     movlw     d'1'
526     movwf     Caracter              ; Inicializamos el puntero de caracteres

528  B_Mensaje
529     movf      Caracter,W
530     movwf     TempLCD
531     btfsc     Mensaje,7             ; Se comprueba de que grupo es el mensaje
532     goto      Mens_GrupoC
533     btfsc     Mensaje,6
534     goto      Mens_GrupoB

536  Mens_GrupoA
537     btfsc     Mensaje,0
538     goto      MensajeA0
539     btfsc     Mensaje,1
540     goto      MensajeA1
541     btfsc     Mensaje,2
542     goto      MensajeA2
543     btfsc     Mensaje,3
544     goto      MensajeA3
545     btfsc     Mensaje,4
546     goto      MensajeA4
547     btfsc     Mensaje,5
548     goto      MensajeA5

550  ;  **** Mensajes del Grupo B ****
551  Mens_GrupoB
552     btfsc     Mensaje,0
553     goto      MensajeB0
554     btfsc     Mensaje,1
555     goto      MensajeB1
556     btfsc     Mensaje,2
557     goto      MensajeB2
558     btfsc     Mensaje,3
559     goto      MensajeB3
560     btfsc     Mensaje,4
561     goto      MensajeB4
562     btfsc     Mensaje,5
563     goto      MensajeB5

565  ;  **** Mensajes del Grupo C ****
566  Mens_GrupoC
567    btfsc     Mensaje,0
568    goto      MensajeC0
569    btfsc     Mensaje,1
570    goto      MensajeC1
```

```
571     btfsc     Mensaje,2
572     goto      MensajeC2
573     btfsc     Mensaje,3
574     goto      MensajeC3
575     btfsc     Mensaje,4
576     goto      MensajeC4
577     btfsc     Mensaje,5
578     goto      MensajeC5

580   MensajeA0
581      movlw     MA0
582      addwf     TempLCD,F
583      movlw     MA0>>8
584      btfsc     STATUS,C
585      addlw     d'1'
586      movwf     PCLATH
587      movf      TempLCD,W
588      call      MA0
589      bsf       PCLATH,3
590      bcf       PCLATH,4
591      goto      Mens_Comun

593   MensajeA1
594      movlw     MA1
595      addwf     TempLCD,F
596      movlw     (MA1>>8)
597      btfsc     STATUS,C
598      addlw     d'1'
599      movwf     PCLATH
600      movf      TempLCD,W
601      call      MA1
602      bsf       PCLATH,3
603      bcf       PCLATH,4
604      goto      Mens_Comun

606   MensajeA2
607      movlw     MA2
608      addwf     TempLCD,F
609      movlw     (MA2>>8)
610      btfsc     STATUS,C
611      addlw     d'1'
612      movwf     PCLATH
613      movf      TempLCD,W
614      call      MA2
615      bsf       PCLATH,3
616      bcf       PCLATH,4
617      goto      Mens_Comun

619   MensajeA3
620      movlw     MA3
621      addwf     TempLCD,F
622      movlw     (MA3>>8)
623      btfsc     STATUS,C
624      addlw     d'1'
625      movwf     PCLATH
626      movf      TempLCD,W
627      call      MA3
628      bsf       PCLATH,3
629      bcf       PCLATH,4
630      goto      Mens_Comun

632   MensajeA4
633      movlw     MA4
634      addwf     TempLCD,F
635      movlw     (MA4>>8)
636      btfsc     STATUS,C
637      addlw     d'1'
638      movwf     PCLATH
639      movf      TempLCD,W
640      call      MA4
641      bsf       PCLATH,3
642      bcf       PCLATH,4
643      goto      Mens_Comun
```

```
645  MensajeA5
646     movlw     MA5
647     addwf     TempLCD,F
648     movlw     (MA5>>8)
649     btfsc     STATUS,C
650     addlw     d'1'
651     movwf     PCLATH
652     movf      TempLCD,W
653     call      MA5
654     bsf       PCLATH,3
655     bcf       PCLATH,4
656     goto      Mens_Comun

658  MensajeB0
659     movlw     MB0
660     addwf     TempLCD,F
661     movlw     (MB0>>8)
662     btfsc     STATUS,C
663     addlw     d'1'
664     movwf     PCLATH
665     movf      TempLCD,W
666     call      MB0
667     bsf       PCLATH,3
668     bcf       PCLATH,4
669     goto      Mens_Comun

671  MensajeB1
672     movlw     MB1
673     addwf     TempLCD,F
674     movlw     (MB1>>8)
675     btfsc     STATUS,C
676     addlw     d'1'
677     movwf     PCLATH
678     movf      TempLCD,W
679     call      MB1
680     bsf       PCLATH,3
681     bcf       PCLATH,4
682     goto      Mens_Comun

684  MensajeB2
685     movlw     MB2
686     addwf     TempLCD,F
687     movlw     (MB2>>8)
688     btfsc     STATUS,C
689     addlw     d'1'
690     movwf     PCLATH
691     movf      TempLCD,W
692     call      MB2
693     bsf       PCLATH,3
694     bcf       PCLATH,4
695     goto      Mens_Comun

697  MensajeB3
698     movlw     MB3
699     addwf     TempLCD,F
700     movlw     (MB3>>8)
701     btfsc     STATUS,C
702     addlw     d'1'
703     movwf     PCLATH
704     movf      TempLCD,W
705     call      MB3
706     bsf       PCLATH,3
707     bcf       PCLATH,4
708     goto      Mens_Comun

710  MensajeB4
711     movlw     MB4
712     addwf     TempLCD,F
713     movlw     (MB4>>8)
714     btfsc     STATUS,C
715     addlw     d'1'
716     movwf     PCLATH
```

```
717      movf      TempLCD,W
718      call      MB4
719      bsf       PCLATH,3
720      bcf       PCLATH,4
721      goto      Mens_Comun

723  MensajeB5
724    movlw     MB5
725    addwf     TempLCD,F
726    movlw     (MB5>>8)
727    btfsc     STATUS,C
728    addlw     d'1'
729    movwf     PCLATH
730    movf      TempLCD,W
731    call      MB5
732    bsf       PCLATH,3
733    bcf       PCLATH,4
734    goto      Mens_Comun

736  MensajeC0
737    movlw     MC0
738    addwf     TempLCD,F
739    movlw     (MC0>>8)
740    btfsc     STATUS,C
741    addlw     d'1'
742    movwf     PCLATH
743    movf      TempLCD,W
744    call      MC0
745    bsf       PCLATH,3
746    bcf       PCLATH,4
747    goto      Mens_Comun

749  MensajeC1
750    movlw     MC1
751    addwf     TempLCD,F
752    movlw     (MC1>>8)
753    btfsc     STATUS,C
754    addlw     d'1'
755    movwf     PCLATH
756    movf      TempLCD,W
757    call      MC1
758    bsf       PCLATH,3
759    bcf       PCLATH,4
760    goto      Mens_Comun

762  MensajeC2
763    movlw     MC2
764    addwf     TempLCD,F
765    movlw     (MC2>>8)
766    btfsc     STATUS,C
767    addlw     d'1'
768    movwf     PCLATH
769    movf      TempLCD,W
770    call      MC2
771    bsf       PCLATH,3
772    bcf       PCLATH,4
773    goto      Mens_Comun

775  MensajeC3
776    movlw     MC3
777    addwf     TempLCD,F
778    movlw     (MC3>>8)
779    btfsc     STATUS,C
780    addlw     d'1'
781    movwf     PCLATH
782    movf      TempLCD,W
783    call      MC3
784    bsf       PCLATH,3
785    bcf       PCLATH,4
786    goto      Mens_Comun

788  MensajeC4
789    movlw     MC4
```

```
790     addwf     TempLCD,F
791     movlw     (MC4>>8)
792     btfsc     STATUS,C
793     addlw     d'1'
794     movwf     PCLATH
795     movf      TempLCD,W
796     call      MC4
797     bsf       PCLATH,3
798     bcf       PCLATH,4
799     goto      Mens_Comun

801   MensajeC5
802     movlw     MC5
803     addwf     TempLCD,F
804     movlw     (MC5>>8)
805     btfsc     STATUS,C
806     addlw     d'1'
807     movwf     PCLATH
808     movf      TempLCD,W
809     call      MC5
810     bsf       PCLATH,3
811     bcf       PCLATH,4
812     goto      Mens_Comun

814   Mens_Comun
815      iorlw     d'0'                  ; Se comprueba la finalizacion del mensaje
816      btfsc     STATUS,Z
817      return
818      call      LCD_EnviaDato
819      incf      Caracter,F
820      goto      B_Mensaje

822   ;*****************************************************************************
823   ;*                                                                           *
824   ;*                                LCD_PonBitRate                             *
825   ;*                                                                           *
826   ;*****************************************************************************
827   ;*                                                                           *
828   ;* Escribe en la pantalla el Bit Rate de la cancion                          *
829   ;*                                                                           *
830   ;*****************************************************************************

832   LCD_PonBitRate:
833      bcf       STATUS,RP0
834      bsf       STATUS,RP1
835      movf      BitRate,W
836      bcf       STATUS,RP1
837      movwf     Caracter              ; Inicializamos el puntero de caracteres
838      bcf       STATUS,C
839      rlf       Caracter,F
840      rlf       Caracter,F
841      incf      Caracter,F

843   B_BitRate
844      movf      Caracter,W
845      movwf     TempLCD
846      bsf       STATUS,RP1
847      btfss     Layer,1               ; Se comprueba de que grupo es el mensaje
848      goto      LIII
849      btfss     Layer,0
850      goto      LII

852   LI
853      bcf       STATUS,RP1
854      movlw     LAYERI
855      addwf     TempLCD,F
856      movlw     (LAYERI>>8)
857      btfsc     STATUS,C
858      addlw     d'1'
859      movwf     PCLATH
860      movf      TempLCD,W
861      call      LAYERI
862      bsf       PCLATH,3
```

```
863     bcf       PCLATH,4
864     goto      L_Comun

866  LII
867     bcf       STATUS,RP1
868     movlw     LAYERII
869     addwf     TempLCD,F
870     movlw     (LAYERII>>8)
871     btfsc     STATUS,C
872     addlw     d'1'
873     movwf     PCLATH
874     movf      TempLCD,W
875     call      LAYERII
876     bsf       PCLATH,3
877     bcf       PCLATH,4
878     goto      L_Comun

880  LIII
881     bcf       STATUS,RP1
882     movlw     LAYERIII
883     addwf     TempLCD,F
884     movlw     (LAYERIII>>8)
885     btfsc     STATUS,C
886     addlw     d'1'
887     movwf     PCLATH
888     movf      TempLCD,W
889     call      LAYERIII
890     bsf       PCLATH,3
891     bcf       PCLATH,4
892     goto      L_Comun

894  L_Comun
895     iorlw     d'0'                  ; Se comprueba la finalizacion del mensaje
896     btfsc     STATUS,Z
897     return
898     call      LCD_EnviaDato
899     incf      Caracter,F
900     goto      B_BitRate

902  ;*****************************************************************************
903  ;*                                                                           *
904  ;*                                 LCD_BitRate                               *
905  ;*                                                                           *
906  ;*****************************************************************************
907  ;*                                                                           *
908  ;*                                                                           *
909  ;*                                                                           *
910  ;*****************************************************************************

912  LCD_BitRate:
913     bcf       STATUS,RP0
914     bcf       STATUS,RP1
915     clrw
916     call      LCD_Linea3
917     movlw     b'10001000'           ; Indicamos que mensaje vamos a pintar
918     movwf     Mensaje
919     call      LCD_PonMensaje
920     movlw     'I'
921     call      LCD_EnviaDato
922     bsf       STATUS,RP1
923     btfsc     Layer,0
924     goto      L_IoL_III

926  L_II
927     bcf       STATUS,RP1
928     movlw     'I'
929     call      LCD_EnviaDato
930     movlw     '␣'
931     call      LCD_EnviaDato
932     movlw     '␣'
933     call      LCD_EnviaDato
934     goto      BRate
```

```
936  L_IoL_III
937     btfsc     Layer,1
938     goto      BRate

940  L_III
941     bcf       STATUS,RP1
942     movlw     'I'
943     call      LCD_EnviaDato
944     movlw     'I'
945     call      LCD_EnviaDato
946     movlw     '␣'
947     call      LCD_EnviaDato

949  BRate
950     bcf       STATUS,RP1
951     movlw     d'10'
952     call      LCD_Linea3
953     call      LCD_PonBitRate
954     movlw     b'10010000'          ; Indicamos que mensaje vamos a pintar
955     movwf     Mensaje
956     call      LCD_PonMensaje
957     return

959  ;*****************************************************************************
960  ;*                                                                           *
961  ;*                                LCD_SampleRate                             *
962  ;*                                                                           *
963  ;*****************************************************************************
964  ;*                                                                           *
965  ;*                                                                           *
966  ;*                                                                           *
967  ;*****************************************************************************

969  LCD_SampleRate:
970     bcf       STATUS,RP0
971     bcf       STATUS,RP1
972     clrw
973     call      LCD_Linea3
974     movlw     b'10000100'          ; Indicamos que mensaje vamos a pintar
975     movwf     Mensaje
976     call      LCD_PonMensaje
977     call      LCD_PonSampleRate
978     movlw     b'10100000'          ; Indicamos que mensaje vamos a pintar
979     movwf     Mensaje
980     call      LCD_PonMensaje
981     return

983  ;*****************************************************************************
984  ;*                                                                           *
985  ;*                              LCD_PonSampleRate                            *
986  ;*                                                                           *
987  ;*****************************************************************************
988  ;*                                                                           *
989  ;* Escribe en la pantalla el Sample Rate de la cancion                       *
990  ;*                                                                           *
991  ;*****************************************************************************

993  LCD_PonSampleRate:
994     bcf       STATUS,RP0
995     bsf       STATUS,RP1
996     movf      SampleRate,W
997     bcf       STATUS,RP1
998     movwf     Caracter             ; Inicializamos el puntero de caracteres
999     bcf       STATUS,C
1000    rlf       Caracter,F
1001    rlf       Caracter,F
1002    incf      Caracter,F

1004 B_SampleRate
1005    movf      Caracter,W
1006    movwf     TempLCD
1007    bsf       STATUS,RP1
1008    btfsc     ID,0                 ; Se comprueba de que grupo es el mensaje
```

```
1009        goto      SH
1010        btfsc     ID,1
1011        goto      SL

1013    SM
1014        bcf       STATUS,RP1
1015        movlw     SR_M
1016        addwf     TempLCD,F
1017        movlw     (SR_M>>8)
1018        btfsc     STATUS,C
1019        addlw     d'1'
1020        movwf     PCLATH
1021        movf      TempLCD,W
1022        call      SR_M
1023        bsf       PCLATH,3
1024        bcf       PCLATH,4
1025        goto      S_Comun

1027    SH
1028        bcf       STATUS,RP1
1029        movlw     SR_H
1030        addwf     TempLCD,F
1031        movlw     (SR_H>>8)
1032        btfsc     STATUS,C
1033        addlw     d'1'
1034        movwf     PCLATH
1035        movf      TempLCD,W
1036        call      SR_H
1037        bsf       PCLATH,3
1038        bcf       PCLATH,4
1039        goto      S_Comun

1041    SL
1042        bcf       STATUS,RP1
1043        movlw     SR_L
1044        addwf     TempLCD,F
1045        movlw     (SR_L>>8)
1046        btfsc     STATUS,C
1047        addlw     d'1'
1048        movwf     PCLATH
1049        movf      TempLCD,W
1050        call      SR_L
1051        bsf       PCLATH,3
1052        bcf       PCLATH,4
1053        goto      S_Comun

1055    S_Comun
1056        iorlw     d'0'                    ; Se comprueba la finalizacion del mensaje
1057        btfsc     STATUS,Z
1058        return
1059        call      LCD_EnviaDato
1060        incf      Caracter,F
1061        goto      B_SampleRate

1063    ;*****************************************************************************
1064    ;*                                                                           *
1065    ;*                                  PonTitulo                                *
1066    ;*                                                                           *
1067    ;*****************************************************************************
1068    ;*                                                                           *
1069    ;*                                                                           *
1070    ;*                                                                           *
1071    ;*****************************************************************************

1073    PonTitulo
1074        clrf      Temp0
1075        movlw     d'1'
1076        movwf     Caracter
1077        clrw
1078        call      LCD_Linea1
1079        clrf      Temp1
1080        bsf       Temp1,2
```

```
1082  BucleTit
1083     movf      Temp0,W
1084     xorlw     d'20'
1085     btfss     STATUS,Z
1086     goto      Sigue_Linea
1087     btfsc     Temp1,2
1088     goto      Linea2
1089     btfsc     Temp1,3
1090     goto      Linea3
1091     btfsc     Temp1,4
1092     goto      Linea4
1093     return

1095  Linea2
1096     clrw
1097     call      LCD_Linea2
1098     bcf       Temp1,2
1099     bsf       Temp1,3
1100     clrf      Temp0
1101     goto      Sigue_Linea

1103  Linea3
1104     clrw
1105     call      LCD_Linea3
1106     bcf       Temp1,3
1107     bsf       Temp1,4
1108     clrf      Temp0
1109     goto      Sigue_Linea

1111  Linea4
1112     clrw
1113     call      LCD_Linea4
1114     bcf       Temp1,4
1115     clrf      Temp0
1116     goto      Sigue_Linea

1118  Sigue_Linea
1119     movf      Caracter,W
1120     movwf     TempLCD

1122     movlw     T1
1123     addwf     TempLCD,F
1124     movlw     (T1>>8)
1125     btfsc     STATUS,C
1126     addlw     d'1'
1127     movwf     PCLATH
1128     movf      TempLCD,W
1129     call      T1
1130     bsf       PCLATH,3
1131     bcf       PCLATH,4

1133     call      LCD_EnviaDato
1134     incf      Caracter,F
1135     incf      Temp0,F
1136     goto      BucleTit

1138  ;*****************************************************************************
1139  ;*                                                                           *
1140  ;*                              LCD_ScrollLinea1                             *
1141  ;*                                                                           *
1142  ;*****************************************************************************
1143  ;*                                                                           *
1144  ;* Realiza parte del Scroll del nombre de fichero en la linea1               *
1145  ;*                                                                           *
1146  ;*****************************************************************************

1148  LCD_ScrollLinea1
1149     bcf       STATUS,RP0          ; Banco 0
1150     bcf       STATUS,RP1
1151     incf      OffsetD,F
1152     call      LCD_PreparaLinea1

1154  ComienzaScrollL1
```

```
1155      movlw     d'3'
1156      call      LCD_Linea1
1157      movlw     d'17'
1158      movwf     PosLibres
1159      movlw     RespaldoLinea1        ; Apuntamos a la zona de origen
1160      movwf     FSR

1162   BucleLinea1
1163      movf      INDF,W
1164      call      LCD_EnviaDato
1165      incf      FSR,F
1166      decfsz    PosLibres,F
1167      goto      BucleLinea1
1168      return

1170   ;*****************************************************************************
1171   ;*                                                                           *
1172   ;*                              LCD_PreparaLinea1                            *
1173   ;*                                                                           *
1174   ;*****************************************************************************
1175   ;*                                                                           *
1176   ;* Elabora la linea 1 en memoria para realizar el Scroll                     *
1177   ;*                                                                           *
1178   ;*****************************************************************************


1181   ;Offset         -> Global
1182   ;PosLibres      -> Local
1183   ;CarScroll      -> Local

1185   LCD_PreparaLinea1
1186      call      LCD_ConfEsc           ; Configuramos el BUS para escribir en el LATH
1187                                      ; MEMORIA
1188      movlw     (NOMBRE_DIRECTORIO>>8)  ; Cargamos la direccion del primer caracter
1189      movwf     DIR_SRAM_H            ; Cargamos la parte alta
1190      movwf     DirMem_H
1191      clrw                            ; Calculamos la parte baja: Base = 0x00
1192      addwf     OffsetD,W             ; Calculamos la parte baja: Base + Offset -> W
1193      movwf     DIR_SRAM_L
1194      movwf     DirMem_L
1195      bcf       INTCON,GIE
1196      movlw     LATCH_MEM_L           ; Lo cargamos en el registro de direccion
1197      movwf     CONTROL1
1198      nop
1199      movlw     OCIOSO
1200      movwf     CONTROL1
1201      bsf       INTCON,GIE
1202                                      ; Seleccionamos el destino de los datos
1203      movlw     RespaldoLinea1        ; Apuntamos a la zona en la que vamos a empezar
1204      movwf     FSR                   ; a construir la linea

1206                                      ; Origen y destino listos
1207      movlw     d'17'                 ; Inicializamos el contador
1208      movwf     PosLibres

1210   BuclePrepLinea1
1211      call      LCD_ConfLec           ; Configuramos el BUS para leer la MEMORIA SRAM
1212      bcf       INTCON,GIE
1213      movlw     OE                    ; Leemos un caracter y lo almacenamos en
1214      movwf     CONTROL1              ; CarScroll
1215      nop
1216      movf      DATOS_BAJOS,W
1217      movwf     CarScroll
1218      movlw     OCIOSO
1219      movwf     CONTROL1
1220      bsf       INTCON,GIE

1222      movf      CarScroll,F           ; Comprobamos si es la marca de fin de nombre
1223      btfss     STATUS,Z              ; 0x00
1224      goto      PonCaracterL1

1226      movf      PosLibres,W           ; Comprobamos si tenemos que resetear el offset
1227      xorlw     d'17'
```

```
1228      btfsc     STATUS,Z
1229      clrf      OffsetD
1230      movlw     ' '                    ; Copiamos el caracter en destino
1231      movwf     INDF

1233      incf      FSR,F                  ; Incrementamos el puntero de destino
1234      decf      PosLibres,F            ; Decrementamos el puntero de posiciones libres
1235      btfsc     STATUS,Z               ; Si PosLibres llega a cero salimos
1236      return
1237                                       ; Ahora tenemos que empezar a poner el nombre
1238                                       ; desde el principio
1239      call      LCD_ConfEsc            ; Configuramos el BUS para escribir en el LATCH
1240                                       ; MEMORIA
1241      movlw     (NOMBRE_DIRECTORIO>>8)  ; Cargamos la direccion del primer caracter
1242      movwf     DIR_SRAM_H
1243      movwf     DirMem_H
1244      clrw
1245      movwf     DIR_SRAM_L
1246      movwf     DirMem_L
1247      bcf       INTCON,GIE
1248      movlw     LATCH_MEM_L
1249      movwf     CONTROL1
1250      nop
1251      movlw     OCIOSO
1252      movwf     CONTROL1
1253      bsf       INTCON,GIE
1254      goto      BuclePrepLinea1        ; Ponemos el caracter

1256  PonCaracterL1
1257      movf      CarScroll,W            ; Copiamos el caracter en destino
1258      movwf     INDF
1259      bcf       INTCON,GIE
1260      bsf       CONTROL2,INC_ADDR      ; Incrementamos la direccion de origen
1261      nop
1262      bcf       CONTROL2,INC_ADDR
1263      incf      DirMem_L,F
1264      btfsc     STATUS,Z
1265      incf      DirMem_H,F
1266      bsf       INTCON,GIE
1267      incf      FSR,F                  ; Incrementamos el puntero de destino

1269      decfsz    PosLibres,F            ; Decrementamos el puntero de posiciones libres
1270      goto      BuclePrepLinea1        ; Tratamos el siguiente caracter
1271      return

1273  ;*****************************************************************************
1274  ;*                                                                           *
1275  ;*                              LCD_ScrollLinea2                             *
1276  ;*                                                                           *
1277  ;*****************************************************************************
1278  ;*                                                                           *
1279  ;* Realiza parte del Scroll del nombre de fichero en la linea2               *
1280  ;*                                                                           *
1281  ;*****************************************************************************

1283  LCD_ScrollLinea2
1284      bcf       STATUS,RP0             ; Banco 0
1285      bcf       STATUS,RP1
1286      incf      OffsetF,F
1287      call      LCD_PreparaLinea2

1289  ComienzaScrollL2
1290      movlw     d'3'
1291      call      LCD_Linea2
1292      movlw     d'17'
1293      movwf     PosLibres
1294      movlw     RespaldoLinea2         ; Apuntamos a la zona de origen
1295      movwf     FSR

1297  BucleLinea2
1298      movf      INDF,W
1299      call      LCD_EnviaDato
1300      incf      FSR,F
```

```
1301      decfsz    PosLibres,F
1302      goto      BucleLinea2
1303      return

1305  ;*****************************************************************************
1306  ;*                                                                           *
1307  ;*                              LCD_PreparaLinea2                            *
1308  ;*                                                                           *
1309  ;*****************************************************************************
1310  ;*                                                                           *
1311  ;* Elabora la linea 2 en memoria para realizar el Scroll                     *
1312  ;*                                                                           *
1313  ;*****************************************************************************


1316  ;Offset         -> Global
1317  ;PosLibres      -> Local
1318  ;CarScroll      -> Local

1320  LCD_PreparaLinea2
1321      call      LCD_ConfEsc           ; Configuramos el BUS para escribir en el LATH
1322                                      ; MEMORIA
1323      movlw     (NOMBRE_FICHERO>>8)   ; Cargamos la direccion del primer caracter
1324      movwf     DIR_SRAM_H            ; Cargamos la parte alta
1325      movwf     DirMem_H
1326      clrw                            ; Calculamos la parte baja: Base = 0x00
1327      addwf     OffsetF,W             ; Calculamos la parte baja: Base + Offset -> W
1328      movwf     DIR_SRAM_L
1329      movwf     DirMem_L
1330      bcf       INTCON,GIE
1331      movlw     LATCH_MEM_L           ; Lo cargamos en el registro de direccion
1332      movwf     CONTROL1
1333      nop
1334      movlw     OCIOSO
1335      movwf     CONTROL1
1336      bsf       INTCON,GIE
1337                                      ; Seleccionamos el destino de los datos
1338      movlw     RespaldoLinea2        ; Apuntamos a la zona en la que vamos a empezar
1339      movwf     FSR                   ; a construir la linea

1341                                      ; Origen y destino listos
1342      movlw     d'17'                 ; Inicializamos el contador
1343      movwf     PosLibres

1345  BuclePrepLinea2
1346      call      LCD_ConfLec           ; Leemos un caracter y lo almacenamos en
1347                                      ; CarScroll
1348      bcf       INTCON,GIE
1349      movlw     OE
1350      movwf     CONTROL1
1351      nop
1352      movf      DATOS_BAJOS,W
1353      movwf     CarScroll
1354      movlw     OCIOSO
1355      movwf     CONTROL1
1356      bsf       INTCON,GIE
1357      call      LCD_ConfEsc

1359      movf      CarScroll,F           ; Comprobamos si es la marca de fin de nombre
1360      btfss     STATUS,Z              ; 0x00
1361      goto      PonCaracterL2

1363      movf      PosLibres,W           ; Comprobamos si tenemos que resetear el offset
1364      xorlw     d'17'
1365      btfsc     STATUS,Z
1366      clrf      OffsetF
1367      movlw     '␣'                   ; Copiamos el caracter en destino
1368      movwf     INDF

1370      incf      FSR,F                 ; Incrementamos el puntero de destino
1371      decf      PosLibres,F           ; Decrementamos el puntero de posiciones libres
1372      btfsc     STATUS,Z              ; Si PosLibres llega a cero salimos
1373      return
```

```
                                    ; Ahora tenemos que empezar a poner el nombre
                                    ; desde el principio
    call      LCD_ConfEsc           ; Configuramos el BUS para escribir en el LATCH
                                    ; MEMORIA
    movlw     (NOMBRE_FICHERO>>8)   ; Cargamos la direccion del primer caracter
    movwf     DIR_SRAM_H
    movwf     DirMem_H
    clrw
    movwf     DIR_SRAM_L
    movwf     DirMem_L
    bcf       INTCON,GIE
    movlw     LATCH_MEM_L
    movwf     CONTROL1
    nop
    movlw     OCIOSO
    movwf     CONTROL1
    bsf       INTCON,GIE
    goto      BuclePrepLinea2       ; Ponemos el caracter

PonCaracterL2
    movf      CarScroll,W           ; Copiamos el caracter en destino
    movwf     INDF

    bcf       INTCON,GIE
    bsf       CONTROL2,INC_ADDR     ; Incrementamos la direccion de origen
    nop
    bcf       CONTROL2,INC_ADDR
    incf      DirMem_L,F
    btfsc     STATUS,Z
    incf      DirMem_H,F
    bsf       INTCON,GIE
    incf      FSR,F                 ; Incrementamos el puntero de destino

    decfsz    PosLibres,F           ; Decrementamos el puntero de posiciones libres
    goto      BuclePrepLinea2       ; Tratamos el siguiente caracter
    return

;*******************************************************************************
;*                                                                             *
;*                                    Scroll                                   *
;*                                                                             *
;*******************************************************************************
;*                                                                             *
;* Comprueba si es necesario o no relaizar el Scroll de fichero o directorio   *
;* en funcion de los BITS de estado y en caso de ser necesario lo realiza.     *
;*                                                                             *
;*******************************************************************************

Scroll

ScrollFich
    btfsc     ESTADO_BUSQUEDA,SCROLL_F
    btfss     INTerrupcion,FICH
    goto      ScrollDir
    bcf       INTerrupcion,FICH
    call      LCD_ScrollLinea2

ScrollDir
    btfsc     ESTADO_BUSQUEDA,SCROLL_D
    btfss     INTerrupcion,DIRE
    return

    bcf       INTerrupcion,DIRE
    call      LCD_ScrollLinea1
    return


;*******************************************************************************
;*                                                                             *
;*                                 MuestraTiempo                               *
;*                                                                             *
;*******************************************************************************
;*                                                                             *
```

```
1447  ;* Muestra el tiempo transcurrido en la pantalla                          *
1448  ;*                                                                        *
1449  ;**************************************************************************

1451  MuestraTiempo

1453     bcf      STATUS,RP1          ; Banco 0
1454     movlw    d'15'               ; Situamos el Cursor
1455     call     LCD_Linea4
1456     bsf      STATUS,RP1          ; Banco 2
1457     movf     DMinutos,W
1458     bcf      STATUS,RP1          ; Banco 0
1459     call     LCD_EnviaDato
1460     bsf      STATUS,RP1          ; Banco 2
1461     movf     UMinutos,W
1462     bcf      STATUS,RP1          ; Banco 0
1463     call     LCD_EnviaDato
1464     movlw    ':'
1465     call     LCD_EnviaDato
1466     bsf      STATUS,RP1          ; Banco 2
1467     movf     DSegundos,W
1468     bcf      STATUS,RP1          ; Banco 0
1469     call     LCD_EnviaDato
1470     bsf      STATUS,RP1          ; Banco 2
1471     movf     USegundos,W
1472     bcf      STATUS,RP1          ; Banco 0
1473     call     LCD_EnviaDato

1475     bcf      INTerrupcion,TIEM   ; Ponemos a 0 el marcador

1477     return

1479  ;**************************************************************************
1480  ;*                                                                        *
1481  ;*                              ActualizarTiempo                          *
1482  ;*                                                                        *
1483  ;**************************************************************************
1484  ;*                                                                        *
1485  ;* Comprueba si es necesario o no actualizar el contador de tiempo        *
1486  ;* transcurrido. Si es necesario lo actualiza.                            *
1487  ;* Trabaja directamente con caracteres ASCII.                             *
1488  ;*                                                                        *
1489  ;**************************************************************************

1491  ActualizarTiempo

1493     btfss   INTerrupcion,TIEM    ; Comprobamos si tenemos que actualizar el
1494                                  ; contador de tiempo trancurrido
1495     return

1497     bsf      STATUS,RP1          ; Banco 2

1499  ModificaUSegundos               ; Modificamos las Unidades de segundos
1500     movlw    b'00111001'         ; Comprobamos si tenemos que pasar de 9 a 0
1501     xorwf    USegundos,W
1502     btfss    STATUS,Z
1503     goto     IncUSegundos        ; NO tenemos que pasar de 9 a 0
1504     movlw    b'00110000'         ; PONEMOS 0 en las unidades de segundos y
1505     movwf    USegundos           ; pasamos a modificar las decenas de segundos.
1506     goto     ModificaDSegundos

1508  IncUSegundos                    ; Incrementamos las unidades de segundos
1509     incf     USegundos,F
1510  ;   bsf     PCLATH,3
1511     call     MuestraTiempo       ; Mostramos el nuevo tiempo transcurrido
1512  ;   bcf     PCLATH,3
1513     return

1515  ModificaDSegundos               ; Modificamos las Decenas de segundos
1516     movlw    b'00110101'         ; Comprobamos si tenemos que pasar de 5 a 0
1517     xorwf    DSegundos,W
1518     btfss    STATUS,Z
1519     goto     IncDSegundos        ; NO tenemos que pasar de 5 a 0
```

```
1520      movlw     b'00110000'          ; PONEMOS 0 en las decenas de segundos y
1521      movwf     DSegundos            ; pasamos a modificar las unidades de minuto.
1522      goto      ModificaUMinutos

1524   IncDSegundos                      ; Incrementamos las decenas de segundos
1525      incf      DSegundos,F
1526   ;   bsf      PCLATH,3
1527      call      MuestraTiempo        ; Mostramos el nuevo tiempo transcurrido
1528   ;   bcf      PCLATH,3
1529      return

1531   ModificaUMinutos                  ; Modificamos las Unidades de minutos
1532      movlw     b'00111001'          ; Comprobamos si tenemos que pasar de 9 a 0
1533      xorwf     UMinutos,W
1534      btfss     STATUS,Z
1535      goto      IncUMinutos          ; NO tenemos que pasar de 9 a 0
1536      movlw     b'00110000'          ; PONEMOS 0 en las unidades de minutos y
1537      movwf     UMinutos             ; pasamos a modificar las decenas de minutos.
1538      goto      ModificaDMinutos

1540   IncUMinutos                       ; Incrementamos las unidades de minutos
1541      incf      UMinutos,F
1542   ;   bsf      PCLATH,3
1543      call      MuestraTiempo        ; Mostramos el nuevo tiempo transcurrido
1544   ;   bcf      PCLATH,3
1545      return

1547   ModificaDMinutos                  ; Modificamos las Decenas de minutos
1548      movlw     b'00111001'          ; Comprobamos si tenemos que pasar de 9 a 0
1549      xorwf     DMinutos,W
1550      btfss     STATUS,Z
1551      goto      IncDMinutos          ; NO tenemos que pasar de 9 a 0
1552      movlw     b'00110000'          ; PONEMOS 0 en las decenas de minutos
1553      movwf     DMinutos             ; y pasamos a mostrar el tiempo transcurrido.
1554   ;   bsf      PCLATH,3
1555      call      MuestraTiempo
1556   ;   bcf      PCLATH,3
1557      return

1559   IncDMinutos                       ; Incrementamos las decenas de minutos
1560      incf      DMinutos,F
1561   ;   bsf      PCLATH,3
1562      call      MuestraTiempo        ; Mostramos el nuevo tiempo transcurrido
1563   ;   bcf      PCLATH,3
1564      return

1566   ;*****************************************************************************
1567   ;*                                                                           *
1568   ;*                                 MostrarBSR                                *
1569   ;*                                                                           *
1570   ;*****************************************************************************
1571   ;*                                                                           *
1572   ;* Muestra el Sample Rate o el Bit Rate por pantalla en funcion del valor de *
1573   ;* los FLAGS                                                                 *
1574   ;*                                                                           *
1575   ;*****************************************************************************

1577   MostrarBSR

1579      btfss     INTerrupcion,BoS
1580      return

1582      bcf       INTerrupcion,BoS
1583      clrf      STATUS               ; Banco 0
1584      btfss     ESTADO_BUSQUEDA,BIToSAMPLE
1585      goto      MostrarBitRate

1587   MostrarSampleRate
1588      call      LCD_SampleRate
1589      bcf       ESTADO_BUSQUEDA,BIToSAMPLE
1590      goto      SalidaMostrar

1592   MostrarBitRate
```

```
1593      call      LCD_BitRate
1594      bsf       ESTADO_BUSQUEDA,BIToSAMPLE

1596   SalidaMostrar
1597      clrf      STATUS
1598      return

1600   ;*****************************************************************************
1601   ;*                                                                           *
1602   ;*                                Print_Status                               *
1603   ;*                                                                           *
1604   ;*****************************************************************************
1605   ;                                                                            *
1606   ; Imprime la palabra de estado en binario                                   *
1607   ;                                                                            *
1608   ;*****************************************************************************

1610   Print_Status
1611      movlw     '1'
1612      btfss     IDE_Status,7
1613      movlw     '0'
1614      call      LCD_EnviaDato
1615      movlw     '␣'
1616      call      LCD_EnviaDato
1617      movlw     '1'
1618      btfss     IDE_Status,6
1619      movlw     '0'
1620      call      LCD_EnviaDato
1621      movlw     '␣'
1622      call      LCD_EnviaDato
1623      movlw     '1'
1624      btfss     IDE_Status,5
1625      movlw     '0'
1626      call      LCD_EnviaDato
1627      movlw     '␣'
1628      call      LCD_EnviaDato
1629      movlw     '1'
1630      btfss     IDE_Status,4
1631      movlw     '0'
1632      call      LCD_EnviaDato
1633      movlw     '␣'
1634      call      LCD_EnviaDato
1635      movlw     '1'
1636      btfss     IDE_Status,3
1637      movlw     '0'
1638      call      LCD_EnviaDato
1639      movlw     '␣'
1640      call      LCD_EnviaDato
1641      movlw     '1'
1642      btfss     IDE_Status,2
1643      movlw     '0'
1644      call      LCD_EnviaDato
1645      movlw     '␣'
1646      call      LCD_EnviaDato
1647      movlw     '1'
1648      btfss     IDE_Status,1
1649      movlw     '0'
1650      call      LCD_EnviaDato
1651      movlw     '␣'
1652      call      LCD_EnviaDato
1653      movlw     '1'
1654      btfss     IDE_Status,0
1655      movlw     '0'
1656      call      LCD_EnviaDato
1657      movlw     '␣'
1658      call      LCD_EnviaDato
1659      return

1661   ;*****************************************************************************
1662   ;*                                                                           *
1663   ;*                                   Bin2Dec                                 *
1664   ;*                                                                           *
1665   ;*****************************************************************************
```

```
1666  ;*                                                                      *
1667  ;* Subrutina de conversion de binario a codigo ASCII                    *
1668  ;* Se parte de un dato binario almacenado en "Dato4:Dato3:Dato2:Dato1:Dato0"   *
1669  ;* y se consiguen los digitos decimales en codigo ASCII correspondientes al    *
1670  ;* dato de entrada en los BYTES "Digitos:...:Digitos+12"                *
1671  ;*                                                                      *
1672  ;************************************************************************

1674  Bin2Dec
1675     movlw    Digitos              ; Se guarda en "W" la posicion de memoria que
1676                                   ; ocupa "Digitos"
1677     addwf    Cifras,W             ; Se añade "Cifras" a la posicion de "Digitos"
1678     movwf    FSR                  ; Se apunta con "FSR" a la posicion donde se
1679     decf     FSR,F                ; encuentra el digito de menor peso (Digitos+2)

1681  OtroMas
1682     call     Div10                ; Llamada a la subrrutina de obtencion de un
1683                                   ; digito
1684     movf     Digito,W             ; Se guarda en "W" el digito decimal obtenido
1685     andlw    b'00001111'          ; Se enmascara la parte alta de "Digito"
1686     addlw    b'00110000'          ; Se convierte el digito decimal a codigo ASCII
1687     movwf    INDF                 ; Se almacena en "Digito+N" el correspondiente
1688                                   ; termino ASCII
1689     decf     FSR,F                ; Se apunta al digito inmediato superior
1690     decfsz   Cifras,F             ; Se decrementa el numero de cifras restantes
1691     goto     OtroMas              ; Se salta a la etiqueta indicada

1693     return                        ; Retorna de la subrrutina

1695  Div10:
1696     clrf     Digito               ; Se inicializa la variable "Digito"
1697     movlw    d'40'                ; Se carga en "NBit" el n° de bits de todos los
1698     movwf    NBit                 ; registos de entrada (DatoX:...:DatoZ)

1700  NuevoBit
1701     bcf      STATUS,C             ; Se borra el bit de acarreo
1702     rlf      Dato0,F
1703     rlf      Dato1,F              ; Se realiza un desplazamiento a derechas del
1704     rlf      Dato2,F              ; dato binario de entrada, introducciendo el
1705     rlf      Dato3,F              ; "MSB" de cada BYTE en suinmediato superior
1706     rlf      Dato4,F              ; por medio del bit de acarreo
1707     rlf      Digito,F             ; Se introduce el "MSB" del dato resultante en
1708                                   ; "Digito"
1709     movlw    d'10'                ; Se guarda en "W" el dato '10'
1710     subwf    Digito,W             ; Se realiza la resta (Digito - 10)
1711     btfss    STATUS,C             ; C = 1 cuando "Digito" supere al '9'
1712     goto     Menor10              ; Se salta a la etiqueta indicada
1713     movwf    Digito               ; Se guarda en "Digito" el valor de las
1714                                   ; unidades resultantes
1715     incf     Dato0,F              ; Se guarda en "Dato0" el valor de las decenas

1717  Menor10
1718     decfsz   NBit,F               ; Se decrementa el numero de bits restantes
1719     goto     NuevoBit             ; Se salta a la etiqueta indicada

1721     return                        ; Retorna de la subrrutina

1723     VARIABLE       FileStartC2 = $
```

A.7. Mensajes de Pantalla

Parte de los mensajes que se muestran al usuario durante la interacción con el dispositivo, son almacenados en la memoria de programa en forma de tablas. En esta parte del código se declaran estas tablas.

Para interferir lo menos posible con el resto del código, se ha optado por situar las tablas de mensajes al comienzo de la **Página 2** de memoria de programa.

```
;*****************************************************************************
;*****************************************************************************
;**                                                                         **
;**                                  MENSAJES                               **
;**                                                                         **
;*****************************************************************************
;*****************************************************************************

;*****************************************************************************
;*                                                                           *
;* Tablas con los mensajes que se mostraran por el LCD                       *
;* Comienzan en el BANCO de MEMORIA numero 3.                                *
;*                                                                           *
;*****************************************************************************

Mensajes  equ 0x1000

   org Mensajes

MA0:
   movwf     PCL
;            "00000000001111111111222222222233333333334"
;            "01234567890123456789012345678901234567890"
   dt        "Reproductor␣de␣MP3.",0

MA1:
   movwf     PCL
;            "00000000001111111111222222222233333333334"
;            "01234567890123456789012345678901234567890"
   dt        "V␣1.1␣-␣En␣Pruebas.",0

MA2:
   movwf     PCL
;            "00000000001111111111222222222233333333334"
;            "01234567890123456789012345678901234567890"
   dt        "IDE:␣DEV␣0␣-␣Listo",0

MA3:
   movwf     PCL
;            "00000000001111111111222222222233333333334"
;            "01234567890123456789012345678901234567890"
   dt        "Sistema:␣No␣es␣FAT32.",0

MA4:
   movwf     PCL
;            "00000000001111111111222222222233333333334"
;            "01234567890123456789012345678901234567890"
   dt        "Sistema:␣FAT32.",0

MA5:
   movwf     PCL
;            "00000000001111111111222222222233333333334"
;            "01234567890123456789012345678901234567890"
   dt        "Modelo:␣",0

MB0:
   movwf     PCL
;            "00000000001111111111222222222233333333334"
;            "01234567890123456789012345678901234567890"
```

```
60      dt        "Ultima␣Entrada.␣␣",0

62  MB1:
63      movwf     PCL
64  ;             "00000000001111111111222222222233333333334"
65  ;             "01234567890123456789012345678901234567890"
66      dt        "No␣Encontrado.",0

68  MB2:
69      movwf     PCL
70  ;             "00000000001111111111222222222233333333334"
71  ;             "01234567890123456789012345678901234567890"
72      dt        "D:␣",0

74  MB3:
75      movwf     PCL
76  ;             "00000000001111111111222222222233333333334"
77  ;             "01234567890123456789012345678901234567890"
78      dt        "F:␣",0

80  MB4:
81      movwf     PCL
82  ;             "00000000001111111111222222222233333333334"
83  ;             "01234567890123456789012345678901234567890"
84      dt        "Raiz.",0

86  MB5:
87      movwf     PCL
88  ;             "00000000001111111111222222222233333333334"
89  ;             "01234567890123456789012345678901234567890"
90      dt        "Volumen:␣␣␣/35␣␣␣␣␣␣",0

92  MC0:
93      movwf     PCL
94  ;             "00000000001111111111222222222233333333334"
95  ;             "01234567890123456789012345678901234567890"
96      dt        "STOP␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣",0

98  MC1:
99      movwf     PCL
100 ;             "00000000001111111111222222222233333333334"
101 ;             "01234567890123456789012345678901234567890"
102     dt        "PLAY␣␣␣␣␣␣",0

104 MC2:
105     movwf     PCL
106 ;             "00000000001111111111222222222233333333334"
107 ;             "01234567890123456789012345678901234567890"
108     dt        "Sample␣Rate:␣",0

110 MC3:
111     movwf     PCL
112 ;             "00000000001111111111222222222233333333334"
113 ;             "01234567890123456789012345678901234567890"
114     dt        "Layer␣",0

116 MC4:
117     movwf     PCL
118 ;             "00000000001111111111222222222233333333334"
119 ;             "01234567890123456789012345678901234567890"
120     dt        "␣KBit/s",0

122 MC5:
123     movwf     PCL
124 ;             "00000000001111111111222222222233333333334"
125 ;             "01234567890123456789012345678901234567890"
126     dt        "␣KHz",0

128 ;              ##   ## ####   # # #
129 ;              ### ### ## ##  # # #
130 ;              ## # ## ####   # # #
131 ;              ##   ## ##     # # #
132 ;             "00000000001111111111"
```

```
133 ;           "01234567890123456789"

135 T1:
136     movwf     PCL
137     dt        b'11111111'
138     dt        b'11111111'
139     dt        b'00100000'
140     dt        b'00100000'
141     dt        b'00100000'
142     dt        b'11111111'
143     dt        b'11111111'
144     dt        b'00100000'
145     dt        b'11111111'
146     dt        b'11111111'
147     dt        b'11111111'
148     dt        b'11111111'
149     dt        b'00100000'
150     dt        b'00100000'
151     dt        b'00100000'
152     dt        b'11111111'
153     dt        b'00100000'
154     dt        b'11111111'
155     dt        b'00100000'
156     dt        b'11111111'

158     dt        b'11111111'
159     dt        b'11111111'
160     dt        b'11111111'
161     dt        b'00100000'
162     dt        b'11111111'
163     dt        b'11111111'
164     dt        b'11111111'
165     dt        b'00100000'
166     dt        b'11111111'
167     dt        b'11111111'
168     dt        b'00100000'
169     dt        b'11111111'
170     dt        b'11111111'
171     dt        b'00100000'
172     dt        b'00100000'
173     dt        b'11111111'
174     dt        b'00100000'
175     dt        b'11111111'
176     dt        b'00100000'
177     dt        b'11111111'

179     dt        b'11111111'
180     dt        b'11111111'
181     dt        b'00100000'
182     dt        b'11111111'
183     dt        b'00100000'
184     dt        b'11111111'
185     dt        b'11111111'
186     dt        b'00100000'
187     dt        b'11111111'
188     dt        b'11111111'
189     dt        b'11111111'
190     dt        b'11111111'
191     dt        b'00100000'
192     dt        b'00100000'
193     dt        b'00100000'
194     dt        b'11111111'
195     dt        b'00100000'
196     dt        b'11111111'
197     dt        b'00100000'
198     dt        b'11111111'

200     dt        b'11111111'
201     dt        b'11111111'
202     dt        b'00100000'
203     dt        b'00100000'
204     dt        b'00100000'
205     dt        b'11111111'
```

```
206     dt        b'11111111'
207     dt        b'00100000'
208     dt        b'11111111'
209     dt        b'11111111'
210     dt        b'00100000'
211     dt        b'00100000'
212     dt        b'00100000'
213     dt        b'00100000'
214     dt        b'00100000'
215     dt        b'11111111'
216     dt        b'00100000'
217     dt        b'11111111'
218     dt        b'00100000'
219     dt        b'11111111'

221 LAYERI:
222     movwf     PCL
223 ;             "00000000001"
224 ;             "01234567890"
225     dt        "---",0
226     dt        "␣32",0
227     dt        "␣64",0
228     dt        "␣96",0
229     dt        "128",0
230     dt        "160",0
231     dt        "192",0
232     dt        "224",0
233     dt        "256",0
234     dt        "288",0
235     dt        "320",0
236     dt        "352",0
237     dt        "384",0
238     dt        "416",0
239     dt        "448",0

241 LAYERII:
242     movwf     PCL
243 ;             "00000000001"
244 ;             "01234567890"
245     dt        "---",0
246     dt        "␣32",0
247     dt        "␣48",0
248     dt        "␣56",0
249     dt        "␣64",0
250     dt        "␣80",0
251     dt        "␣96",0
252     dt        "112",0
253     dt        "128",0
254     dt        "160",0
255     dt        "192",0
256     dt        "224",0
257     dt        "256",0
258     dt        "320",0
259     dt        "384",0

261 LAYERIII:
262     movwf     PCL
263 ;             "00000000001"
264 ;             "01234567890"
265     dt        "---",0
266     dt        "␣32",0
267     dt        "␣40",0
268     dt        "␣48",0
269     dt        "␣56",0
270     dt        "␣64",0
271     dt        "␣80",0
272     dt        "␣96",0
273     dt        "112",0
274     dt        "128",0
275     dt        "160",0
276     dt        "192",0
277     dt        "224",0
278     dt        "256",0
```

```
279     dt          "320",0

281 SR_H:
282     movwf       PCL
283 ;               "00000000001"
284 ;               "01234567890"
285     dt          "␣44",0
286     dt          "␣48",0
287     dt          "␣32",0
288     dt          "␣--",0

290 SR_M:
291     movwf       PCL
292 ;               "00000000001"
293 ;               "01234567890"
294     dt          "␣22",0
295     dt          "␣24",0
296     dt          "␣16",0
297     dt          "␣--",0

299 SR_L:
300     movwf       PCL
301 ;               "00000000001"
302 ;               "01234567890"
303     dt          "␣11",0
304     dt          "␣12",0
305     dt          "␣␣8",0
306     dt          "␣--",0

308     VARIABLE    FileStartC3 = $
```

A.8. Decodificador de MP3

El PIC se comunica con el decodificador de MP3 *VS1001K* por medio del BUS SPI. Sin embargo, el decodificador debe ser previamente configurado, y el envío de la información tiene que seguir un protocolo establecido.

Se encuentra ubicado en la Página 1 de memoria de programa del microcontrolador.

```
;*****************************************************************************
;*****************************************************************************
;*****************************************************************************
;***                                                                       ***
;**                                I_VS1001.asm                             **
;**                                                                         **
;** Conjunto de funciones que posibilitan el acceso al chip decodificador de  **
;** MPEG que va a encargar de decodificar los datos extraidos del dispositivo **
;** IDE.                                                                    **
;** La comunicacion entre el PIC y el decodificador (VS1001K) se realiza    **
;** mediante dos puertos de comunicacion serie SPI en el decodificador, que  **
;** se conectan al puerto SPI del PIC.                                      **
;**                                                                         **
;***                                                                       ***
;*****************************************************************************
;*****************************************************************************
;*****************************************************************************

;*****************************************************************************
;*****************************************************************************
;**                                                                         **
;**                            Control del VS1001K                          **
;**                                                                         **
;*****************************************************************************
;*****************************************************************************

VSCONTROL         EQU PORTC

; BUS de DATOS SDI

BSYNC             EQU 0    ; Señal de sincronizacion de BYTE
DREQ              EQU 1    ; Peticion de dato
DCLK              EQU 6    ; Reloj del BUS
SDATA             EQU 7    ; Entrada de datos

; BUS de CONTROL SCI

XCS               EQU 2    ; Habilitacion del CHIP Decodificador. Activo en baja
SCLK              EQU 3    ; Reloj del BUS
SI                EQU 5    ; Entrada de datos
SO                EQU 4    ; Salida de datos
XRESET            EQU 0    ; Reset (PUERTO E - Activo en baja)

;*****************************************************************************
;*****************************************************************************
;**                                                                         **
;**                           Comandos de Control SCI                       **
;**                                                                         **
;*****************************************************************************
;*****************************************************************************

SCI_READ          EQU 0x03 ; Comando de lectura de registro
SCI_WRITE         EQU 0x02 ; Comando de escritura de registro

;*****************************************************************************
;*****************************************************************************
;**                                                                         **
;**                 Direcciones de los Registros de control SCI             **
;**                                                                         **
;*****************************************************************************
;*****************************************************************************
```

```
63  VS_MODE           EQU 0x00  ; RW - Control de Modo
64  VS_STATUS         EQU 0x01  ; RW - Estado del VS1001K
65  VS_FCTLH          EQU 0x02  ; RW - "No debe usarse"
66  VS_CLOCKF         EQU 0x03  ; RW - Frecuencia de reloj + Doblador
67  VS_DECODE_TIME    EQU 0x04  ; R  - Tiempo decodificado (sg)
68  VS_AUDATA         EQU 0x05  ; R  - Misc. Datos de Audio
69  VS_WRAM           EQU 0x06  ; W  - RAM de escritura de programa
70  VS_WRAMADDR       EQU 0x07  ; W  - Direccion base para WRAM
71  VS_HDAT0          EQU 0x08  ; R  - Cabecera de datos
72  VS_HDAT1          EQU 0x09  ; R  - Cabecera de datos
73  VS_ALADDR         EQU 0x0A  ; RW - Direccion de la aplicacion
74  VS_VOL            EQU 0x0B  ; RW - Control de volumen
75  VS_AICTRL         EQU 0x0C  ; RW - Control de la aplicacion

77  ;*****************************************************************************
78  ;*                                                                           *
79  ;*                              BITS de VS_MODE                              *
80  ;*                                                                           *
81  ;* La forma de usarlos sera haciendo OR's sucesivos de las etiquetas que     *
82  ;* queramos mandar sobre W antes de mandarlo.                                *
83  ;*                                                                           *
84  ;*****************************************************************************

86  ; PRIMER BYTE DE VS_MODE
87                            ; BIT 8
88  SM_DACT           EQU 1   ; 0 -> Flanco de reloj activo en la SUBIDA
89                            ; 1 -> Flanco de reloj activo en la BAJADA
90                            ; BIT 9
91  SM_BYTEORD        EQU 2   ; 0 -> Orden de Bits en la entrada  - MSB Primero
92                            ; 1 -> Orden de Bits en la entrada  - MSB Ultimo
93                            ; BIT 10
94  SM_IBMODE         EQU 4   ; 0 -> Funcionamiento del BUS de entrada - ESCLAVO
95                            ; 1 -> Funcionamiento del BUS de entrada - MAESTRO
96                            ; BIT 11
97  SM_IBCLK          EQU 8   ; 0 -> Reloj del BUS cuando  es maestro -  512 kHz
98                            ; 1 -> Reloj del BUS cuando  es maestro - 1024 kHz

100 ; SEGUNDO BYTE DE VS_MODE

102                           ; BIT 0
103 SM_DIFF           EQU 1   ; 0 -> Audio en fase normal
104                           ; 1 -> Canal izquierdo invertido (Surround)
105                           ; BIT 1
106 SM_FFWD           EQU 2   ; 0 -> Reproduccion normal
107                           ; 1 -> Reproduccion a velocidad alta
108                           ; BIT 2
109 SM_RESET          EQU 4   ; 0 -> RESET por Software inactivo
110                           ; 1 -> RESET por Software
111                           ; BIT 4
112 SM_PDOWN          EQU 16  ; 0 -> Encendido
113                           ; 1 -> Apagado (Bajo consumo)
114                           ; BIT 7
115 SM_BASS           EQU 128 ; 0 -> Acentuador de graves apagado
116                           ; 1 -> Acentuador de graves encendido

118 ;*****************************************************************************
119 ;*                                                                           *
120 ;*                             BITS de VS_AUDATA                             *
121 ;*                                                                           *
122 ;*****************************************************************************

124 ; Los BITS 0..8 contienen el BITRRATE de la cancion en kbits/seg

126 ; Los BITS 9..12 contienen un indice para determinar la frecuencia de muestreo.

128 SR_MASK           EQU b'00011110'   ; Mascara para extraer el SAMPLE RATE del
129                                     ; segundo byte de VS_AUDATA

131 SR_DESCONOCIDO    EQU b'00000000'
132 SR_44100          EQU b'00000010'
133 SR_48000          EQU b'00000100'
134 SR_32000          EQU b'00000110'
```

```
135  SR_22050          EQU b'00001000'
136  SR_24000          EQU b'00001010'
137  SR_16000          EQU b'00001100'
138  SR_11025          EQU b'00001110'
139  SR_12000          EQU b'00010000'
140  SR_8000           EQU b'00010010'

142  BIT_ESTEREO       EQU 7             ; BIT del segundo BYTE de VS_AUDATA que nos
143                                      ; indica si la cancion es mono o Estereo
144                                      ; 0 -> MONO
145                                      ; 1 -> ESTEREO

147  Interfaz_VS1001   equ FileStartC2

149  ;*****************************************************************************
150  ;*****************************************************************************
151  ;**                                                                         **
152  ;**                          Funciones de acceso                            **
153  ;**                                                                         **
154  ;*****************************************************************************
155  ;*****************************************************************************

157     org      Interfaz_VS1001

159  ;*****************************************************************************
160  ;*                                                                           *
161  ;*                                Inicializa_SPI                             *
162  ;*                                                                           *
163  ;*****************************************************************************
164  ;*                                                                           *
165  ;* Para la interfaz de control, usaremos el puerto MSSP del PIC configurado de *
166  ;* forma que ejecute el protocolo SPI                                        *
167  ;*                                                                           *
168  ;* La velocidad del puero SPI viene determinada por el decodificador, puede  *
169  ;* aguantar como MAXIMO, una velocidad de oscilacion del reloj de 1/4 de su  *
170  ;* frecuencia, como el crystal del decodificador es de 12,888 MHz;           *
171  ;* 12,288 MHz/4 = 3,072 MHz el Microprocesador funciona a 20MHz y puede hacer *
172  ;* funcionar el puwerto SPI a 1/4, 1/16, ... de su frecuencia maxima, con 1/4 *
173  ;* el puerto SPI funcionaria a 5 MHz (se pasa) y con 1/16 funcionaria a      *
174  ;* 1,25 MHz, que ya esta por debajo del limite maximo de 3 MHz               *
175  ;*                                                                           *
176  ;*****************************************************************************

178  Inicializa_SPI
179     bcf      SSPCON,CKP        ; Estado IDLE en la parte baja del reloj
180     bsf      SSPCON,SSPM0      ; Velocidad del puerto 1/16 = 1,25 MHz
181     bsf      STATUS,RP0        ; Banco 1
182     bsf      SSPSTAT,CKE       ; Datos transmitidos en el FLANCO DE SUBIDA de SCK
183     bcf      STATUS,RP0        ; Banco 0
184     bsf      SSPCON,SSPEN      ; Habilita el puerto serie
185     return

187  ;*****************************************************************************
188  ;*                                                                           *
189  ;*                              Inicializa_VS1001                            *
190  ;*                                                                           *
191  ;*****************************************************************************
192  ;*                                                                           *
193  ;* Reinicia el Chip Decodificador                                            *
194  ;*                                                                           *
195  ;*****************************************************************************

197  Inicializa_VS1001
198     bcf      PCLATH,3
199     movlw    d'3'              ; Esperamos 3 ms
200     call     Espera_C
201     bsf      PCLATH,3
202     call     Reset_H_VS1001    ; Reseteamos por Hardware el CHIP
203     bcf      PCLATH,3
204     movlw    d'3'              ; Esperamos 3 ms
205     call     Espera_C
206     bsf      PCLATH,3
207     call     Configura_VS1001  ; Configuramos el CHIP
```

```
208      call      Reset_S_VS1001
209      movlw     d'32'
210      movwf     Contador1

212   Manda_Ceros
213      movlw     d'0'
214      call      SDI_Out
215      decfsz    Contador1,F
216      goto      Manda_Ceros
217      return



221   ;*****************************************************************************
222   ;*                                                                           *
223   ;*                              Configura_VS1001                             *
224   ;*                                                                           *
225   ;*****************************************************************************
226   ;*                                                                           *
227   ;* Realiza la configuracion inicial del CHIP decodificador VS1001K           *
228   ;*                                                                           *
229   ;* Con respecto a la velocidad maxima de envio de datos, una vez configurado y *
230   ;* activado el doblador de frecuencia del decodificador, la frecuencia de    *
231   ;* trabajo sera:  12.288 MHz * 2 = 24,576 MHz con lo que la velocidad maxima  *
232   ;* del puerto sera: 24,576 MHz /4 = 6,144 MHz, luego podemos hacer funcionar  *
233   ;* el puerto serie a la velocidad maxima de 5MHz.                            *
234   ;*                                                                           *
235   ;* Ademas como la frecuencia del decodificador ya esta doblada, podriamos    *
236   ;* incluso reconfigurar el puerto SPI para que el BUS de control, tambien    *
237   ;* funcionara a 5 MHz                                                        *
238   ;*                                                                           *
239   ;*****************************************************************************

241   Configura_VS1001
242                                 ; Ajustamos el reloj, tenemos un cristal de
243                                 ; 12.288 MHz luego:

245                                 ; Doblador   Crystal           Valor a meter
246                                 ; --------   -------------    ---------------
247                                 ; 0x8000  +  12288000/2000 =      0x9800
248                                 ;  32768  +           6144 =       38912

250      bcf       VSCONTROL,XCS    ; Habilitimas el VS1001K
251      movlw     SCI_WRITE        ; Comando -> ESCRIBIR
252      call      SCI_Out
253      movlw     VS_CLOCKF        ; Registro -> RELOJ
254      call      SCI_Out
255      movlw     0x98             ; Parte Alta
256      call      SCI_Out
257      movlw     0x00             ; Parte Baja
258      call      SCI_Out
259      bsf       VSCONTROL,XCS    ; Deshabilitamos el VS1001K
260      bcf       PCLATH,3
261      movlw     d'1'             ; Esperamos 1ms
262      call      Espera_C
263   ;  bsf       PCLATH,3
264   ;  bcf       VSCONTROL,XCS    ; Habilitimas el VS1001K
265   ;  movlw     SCI_WRITE        ; Comando -> ESCRIBIR
266   ;  call      SCI_Out
267   ;  movlw     VS_FCTLH         ; Registro -> XXX
268   ;  call      SCI_Out
269   ;  movlw     0x80             ; Parte Alta
270   ;  call      SCI_Out
271   ;  movlw     0x08             ; Parte Baja
272   ;  call      SCI_Out
273   ;  bsf       VSCONTROL,XCS    ; Deshabilitamos el VS1001K
274   ;  bcf       PCLATH,3
275   ;  movlw     d'1'             ; Esperamos 1ms
276   ;  call      Espera_C
277   ;  bcf       SSPCON,SSPM0     ; Subimos la velocidad del puerto. 1/4 = 5 MHz
278      bsf       PCLATH,3         ; Ajustamos el volumen
279      bcf       VSCONTROL,XCS    ; Habilitimas el VS1001K
280      movlw     SCI_WRITE        ; Comando -> ESCRIBIR
```

```
281     call      SCI_Out
282     movlw     VS_VOL              ; Registro -> VOLUMEN
283     call      SCI_Out
284     movlw     d'40'               ; IZQUIERDA -> 40 * 0.5 = -20 dB
285     call      SCI_Out
286     movlw     d'40'               ; DERECHA -> 40 * 0.5 = -20 dB
287     call      SCI_Out
288     movwf     Volumen             ; Valor de volumen ajustado
289     bsf       VSCONTROL,XCS       ; Deshabilitamos el VS1001K
290     bcf       PCLATH,3
291     movlw     d'1'                ; Esperamos 1ms
292     call      Espera_C
293     bsf       PCLATH,3
294     return

296 ;*****************************************************************************
297 ;*                                                                           *
298 ;*                                Reset_H_VS1001                             *
299 ;*                                                                           *
300 ;*****************************************************************************
301 ;*                                                                           *
302 ;* Provoca un RESET por Hardware del CHIP VS1001K                            *
303 ;*                                                                           *
304 ;*****************************************************************************

306 Reset_H_VS1001
307     bcf       PORTE,XRESET        ; Activamos la linea de RESET
308     bcf       PCLATH,3
309     movlw     d'1'                ; Esperamos 1ms
310     call      Espera_C
311     bsf       PCLATH,3
312     bsf       PORTE,XRESET        ; Desactivamos la linea de RESET
313     bcf       PCLATH,3
314     movlw     d'5'                ; Esperamos 5 ms
315     call      Espera_C
316     bsf       PCLATH,3
317     return

319 ;*****************************************************************************
320 ;*                                                                           *
321 ;*                                Reset_S_VS1001                             *
322 ;*                                                                           *
323 ;*****************************************************************************
324 ;*                                                                           *
325 ;* Provoca un RESET por Software del CHIP VS1001K y lo configuramos          *
326 ;*                                                                           *
327 ;*****************************************************************************

329 Reset_S_VS1001
330     bcf       VSCONTROL,XCS
331     movlw     SCI_WRITE           ; Comando -> ESCRIBIR
332     call      SCI_Out
333     movlw     VS_MODE             ; Registro -> MODO
334     call      SCI_Out
335     clrw                          ; BYTE ALTO
336     call      SCI_Out
337     clrw                          ; BYTE BAJO
338     iorlw     SM_RESET            ; Activamos el RESET
339     call      SCI_Out
340     bsf       VSCONTROL,XCS
341     bcf       PCLATH,3
342     movlw     d'1'                ; Esperamos 1ms
343     call      Espera_C
344     bsf       PCLATH,3
345     bcf       VSCONTROL,XCS
346     movlw     SCI_WRITE           ; Comando -> ESCRIBIR
347     call      SCI_Out
348     movlw     VS_MODE             ; Registro -> MODO
349     call      SCI_Out
350     clrw                          ; BYTE ALTO
351     call      SCI_Out
352     clrw                          ; BYTE BAJO
353     call      SCI_Out             ; Todo desactivado, incluido el RESET
```

```
354     bsf       VSCONTROL,XCS
355     bcf       PCLATH,3
356     movlw     d'5'                ; Esperamos 100ms
357     call      Espera_L
358     bsf       PCLATH,3

360  ; Despues del RESET esperamos 2us y testeamos DREQ. DREQ estara baja durante
361  ; unos 6000 ciclos de reloj, funcionando a 24.576MHz esto supone unos 250us.
362  ; Cuando DREQ suba, debemos escribir al menos 1 cero en SDI antes de continuar
363  ; con el funcionamiento normal.
364  ; Como ya hemos esperado 1ms, es de esperar que DREQ ya este en alta.

366  Esperar_Listo
367     btfss     VSCONTROL,DREQ
368     goto      Esperar_Listo
369     clrw
370     call      SDI_Out
371     return

373  ; El RESET por Software debe hacerse siempre entre la reproduccion de dos
374  ; canciones, si queremos asegurarnos de que no se corta el final de una cancion
375  ; con un BITRATE bajo, deberemos enviar 2048 ceros a SDI antes de activar el
376  ; RESET.

378  ;*****************************************************************************
379  ;*                                                                           *
380  ;*                                  Extrae_Info                              *
381  ;*                                                                           *
382  ;*****************************************************************************
383  ;*                                                                           *
384  ;* Obtiene Informacion acerca de la cancion que se esta reproduciendo        *
385  ;*                                                                           *
386  ;*****************************************************************************

388  Extrae_Info:
389     bcf       STATUS,RP0
390     bcf       STATUS,RP1
391     bcf       VSCONTROL,XCS       ; Seleccionamos el Decodificador
392     movlw     SCI_READ            ; Lectura
393     call      SCI_Out
394     movlw     VS_HDAT0            ; Registro 8 (HDAT0)
395     call      SCI_Out
396     movlw     0x00                ; Iniciamos la lectura
397     call      SCI_Out

399     movf      SSPBUF,W            ; Recibimos el Byte Alto (el de interes)
400     bsf       STATUS,RP1          ; Banco 1
401     movwf     BitRate             ; Lo almacenamos en el registro adecuado
402     bcf       STATUS,RP1          ; Banco 0

404     movlw     0x00                ; Recibimos el Byte Bajo y lo ignoramos
405     call      SCI_Out
406     bsf       VSCONTROL,XCS       ; Desseleccionamos el Decodificador

408     bcf       VSCONTROL,XCS       ; Seleccionamos el Decodificador
409     movlw     SCI_READ            ; Lectura
410     call      SCI_Out
411     movlw     VS_HDAT1            ; Registro 8 (HDAT1)
412     call      SCI_Out
413     movlw     0x00                ; Iniciamos la lectura. Recibimos el Byte Alto
414     call      SCI_Out             ; y lo ignoramos.

416     movlw     0x00                ; Recibimos el Byte Bajo (el de interes)
417     call      SCI_Out

419     movf      SSPBUF,W
420     bsf       STATUS,RP1          ; Banco 1
421     movwf     Layer               ; Lo almacenamos en el registro adecuado
422     bcf       STATUS,RP1          ; Banco 0
423     bsf       VSCONTROL,XCS       ; Desseleccionamos el Decodificador

425     bsf       STATUS,RP1          ; Banco 1
```

```
427     movf      BitRate,W
428     movwf     SampleRate

430     movf      Layer,W
431     movwf     ID

433     rrf       BitRate,F          ; Extraemos el Valor del BitRate
434     rrf       BitRate,F
435     rrf       BitRate,F
436     rrf       BitRate,F
437     movlw     0x0F
438     andwf     BitRate,F

440     rrf       SampleRate,F       ; Extraemos el Valor del
441     rrf       SampleRate,F
442     movlw     0x03
443     andwf     SampleRate,F

445     rrf       Layer,F            ; Extraemos el valor del "Layer"
446     movlw     0x03
447     andwf     Layer,F

449     rrf       ID,F               ; Extraemos el valor del "ID"
450     rrf       ID,F
451     rrf       ID,F
452     movlw     0x03
453     andwf     ID,F

455     bcf       STATUS,RP1         ; Banco 0
456     return

458 ;*****************************************************************************
459 ;*                                                                           *
460 ;*                                    SCI_Out                                *
461 ;*                                                                           *
462 ;*****************************************************************************
463 ;*                                                                           *
464 ;* Manda un dato por el puerto SCI                                           *
465 ;*                                                                           *
466 ;*****************************************************************************

468 SCI_Out
469     movwf     SSPBUF

471 Mandando_SCI
472     btfss     PIR1,SSPIF         ; Esperamos hasta que el micro ha enviado todo el
473     goto      Mandando_SCI       ; dato
474     bcf       PIR1,SSPIF         ; Ponemos el FLAG a 0
475     return

477 ;*****************************************************************************
478 ;*                                                                           *
479 ;*                                    SDI_Out                                *
480 ;*                                                                           *
481 ;*****************************************************************************
482 ;*                                                                           *
483 ;* Manda un dato por el puerto SDI                                           *
484 ;*                                                                           *
485 ;*****************************************************************************

487 SDI_Out
488     bsf       VSCONTROL,BSYNC
489     movwf     SSPBUF
490     nop
491     nop
492     nop
493     nop
494     nop
495     nop
496     nop
497     nop
498     nop
499     nop
```

```
500     nop
501     nop
502     bcf       VSCONTROL,BSYNC

504 Mandando_SDI
505     btfss     PIR1,SSPIF
506     goto      Mandando_SDI
507     bcf       PIR1,SSPIF
508     return

510     VARIABLE       FileStartC2 = $
```

Apéndice B

Datasheets

B.1. PIC16F877A

*Microcontrolador **Microchip PIC16F877A**. Debido a la extensión de este datasheet (218 páginas), tan solo se muestran unas pocas a modo de introducción. El datasheet completo puede encontrarse, junto con todos los demás, en el CD adjunto.*

PIC16F87X
Data Sheet

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

PIC16F87X

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

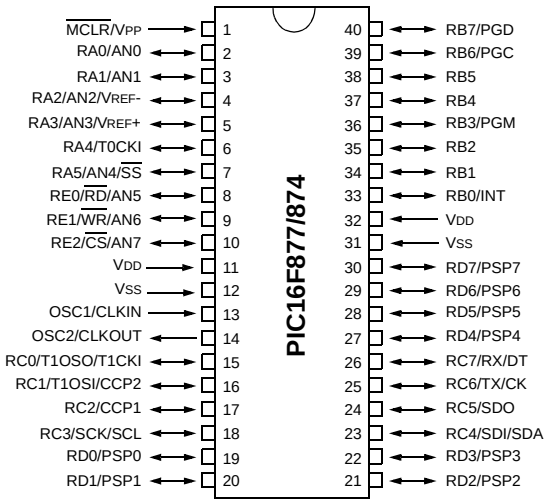
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and
  Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram

PDIP



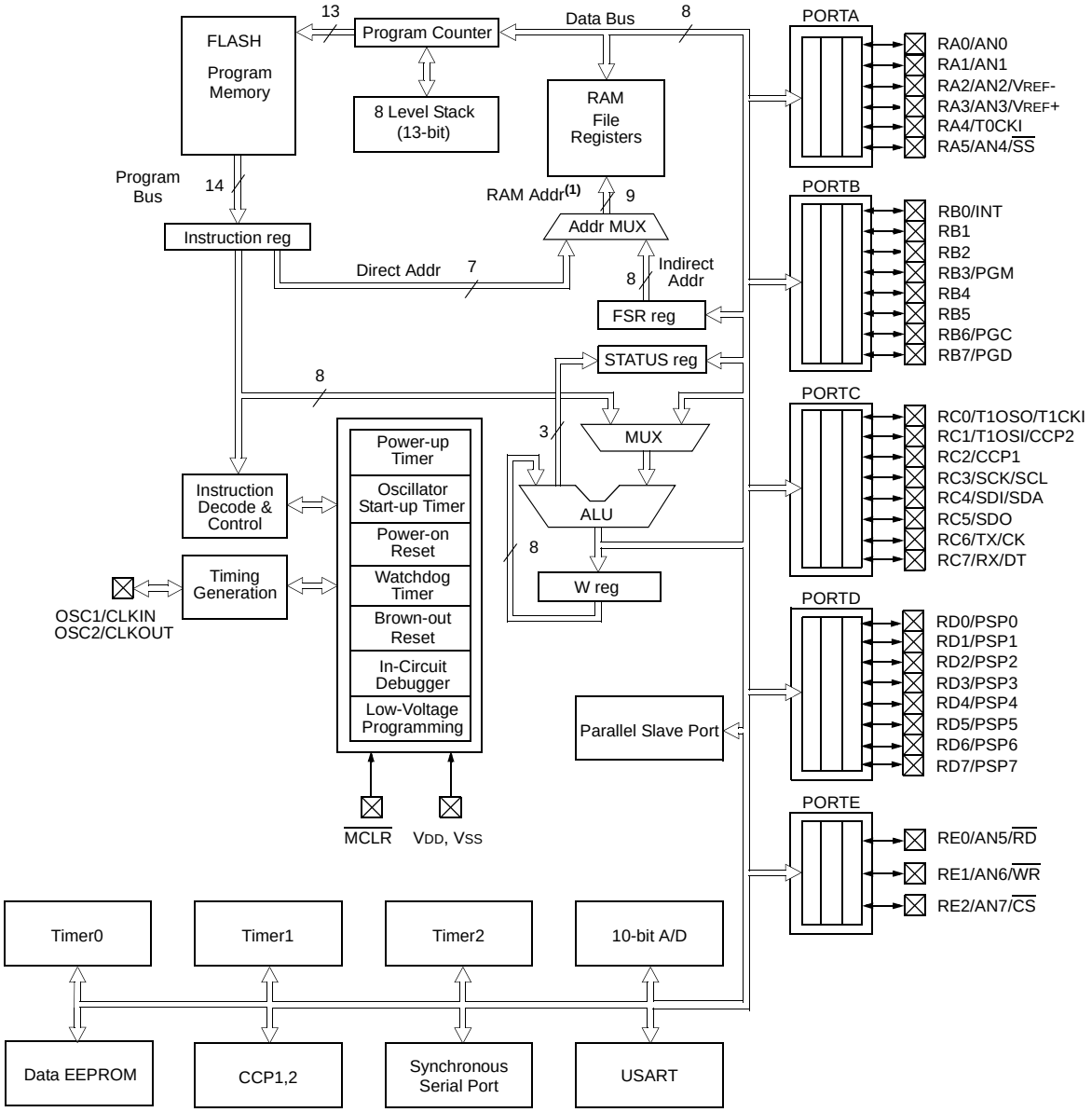
Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

FIGURE 1-2: PIC16F874 AND PIC16F877 BLOCK DIAGRAM

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



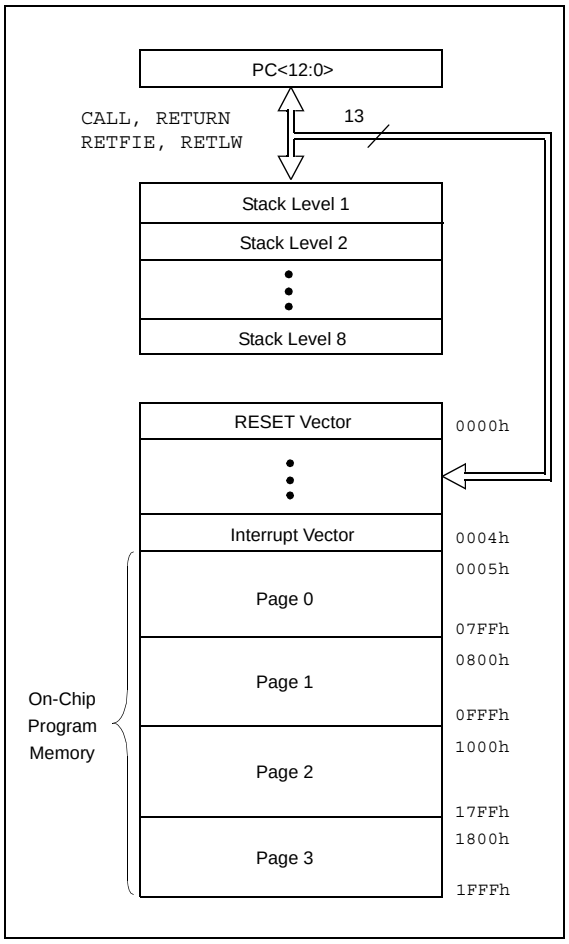
Note 1: Higher order bits are from the STATUS register.

2.0 MEMORY ORGANIZATION

There are three memory blocks in each of the PIC16F87X MCUs. The Program Memory and Data Memory have separate buses so that concurrent access can occur and is detailed in this section. The EEPROM data memory block is detailed in Section 4.0.

Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

FIGURE 2-1: PIC16F877/876 PROGRAM MEMORY MAP AND STACK

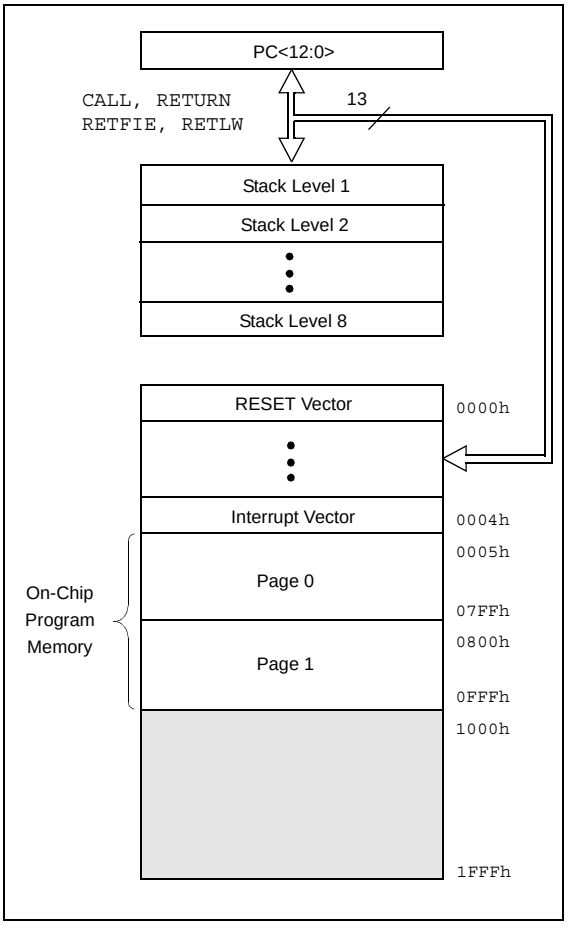


2.1 Program Memory Organization

The PIC16F87X devices have a 13-bit program counter capable of addressing an 8K x 14 program memory space. The PIC16F877/876 devices have 8K x 14 words of FLASH program memory, and the PIC16F873/874 devices have 4K x 14. Accessing a location above the physically implemented address will cause a wraparound.

The RESET vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 2-2: PIC16F874/873 PROGRAM MEMORY MAP AND STACK



2.2 Data Memory Organization

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the Special Function Registers. Bits RP1 (STATUS<6>) and RP0 (STATUS<5>) are the bank select bits.

RP1:RP0	Bank
00	0
01	1
10	2
11	3

Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some frequently used Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

Note: EEPROM Data Memory description can be found in Section 4.0 of this data sheet.

2.2.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly, or indirectly through the File Select Register (FSR).

FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP

Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3	File Address
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD(1)	08h	TRISD(1)	88h		108h		188h
PORTE(1)	09h	TRISE(1)	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved(2)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Reserved(2)	18Fh
T1CON	10h		90h	General Purpose Register 16 Bytes	110h	General Purpose Register 16 Bytes	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h
	7Fh	accesses 70h-7Fh	EFh	accesses 70h-7Fh	16Fh	accesses 70h - 7Fh	1EFh
			F0h		170h		1F0h
			FFh		17Fh		1FFh

Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

2.2.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. A list of these registers is given in Table 2-1.

The Special Function Registers can be classified into two sets: core (CPU) and peripheral. Those registers associated with the core functions are described in detail in this section. Those related to the operation of the peripheral features are described in detail in the peripheral features section.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
01h	TMR0	Timer0 Module Register								xxxx xxxx	47
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	29
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	33
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	35
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	36
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
0Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	22
0Dh	PIR2	—	**(5)**	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0--0	24
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	51
11h	TMR2	Timer2 Module Register								0000 0000	55
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	55
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	70, 73
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	67
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	57
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	57
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	58
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	96
19h	TXREG	USART Transmit Data Register								0000 0000	99
1Ah	RCREG	USART Receive Data Register								0000 0000	101
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	57
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	57
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	58
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	116
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	111

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as ‘0’.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as ‘0’.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
81h	OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	19
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
83h(3)	STATUS	IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C	0001 1xxx	18
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	29
86h	TRISB	PORTB Data Direction Register								1111 1111	31
87h	TRISC	PORTC Data Direction Register								1111 1111	33
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	35
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	37
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
8Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	21
8Dh	PIE2	—	**(5)**	—	EEIE	BCLIE	—	—	CCP2IE	-r-0 0--0	23
8Eh	PCON	—	—	—	—	—	—	$\overline{\text{POR}}$	$\overline{\text{BOR}}$	---- --qq	25
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	68
92h	PR2	Timer2 Period Register								1111 1111	55
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	73, 74
94h	SSPSTAT	SMP	CKE	D/$\overline{\text{A}}$	P	S	R/$\overline{\text{W}}$	UA	BF	0000 0000	66
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	95
99h	SPBRG	Baud Rate Generator Register								0000 0000	97
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	—	Unimplemented								—	—
9Dh	—	Unimplemented								—	—
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	116
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000	112

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.
Shaded locations are unimplemented, read as ‘0’.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.
2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.
3: These registers can be addressed from any bank.
4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as ‘0’.
5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 2											
100h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
101h	TMR0	Timer0 Module Register								xxxx xxxx	47
102h(3)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	26
103h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
104h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
10Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx	41
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx	41
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						xxxx xxxx	41
10Fh	EEADRH	—	—	—	EEPROM Address Register High Byte					xxxx xxxx	41
Bank 3											
180h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	19
182h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
183h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
184h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	31
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
18Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	41, 42
18Dh	EECON2	EEPROM Control Register2 (not a physical register)								---- ----	41
18Eh	—	Reserved maintain clear								0000 0000	—
18Fh	—	Reserved maintain clear								0000 0000	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

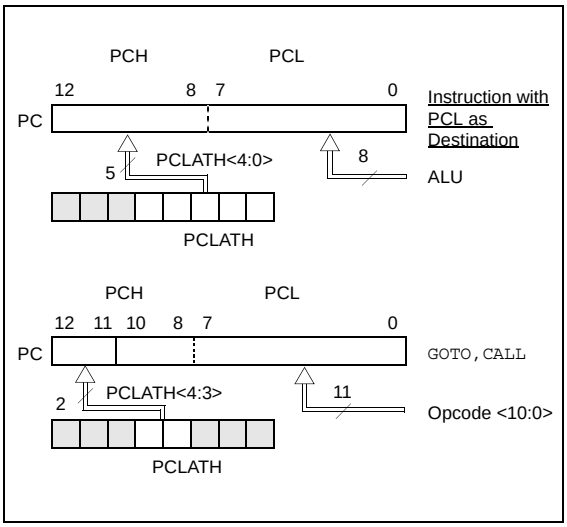
4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

2.3 PCL and PCLATH

The program counter (PC) is 13-bits wide. The low byte comes from the PCL register, which is a readable and writable register. The upper bits (PC<12:8>) are not readable, but are indirectly writable through the PCLATH register. On any RESET, the upper bits of the PC will be cleared. Figure 2-5 shows the two situations for the loading of the PC. The upper example in the figure shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). The lower example in the figure shows how the PC is loaded during a `CALL` or `GOTO` instruction (PCLATH<4:3> → PCH).

FIGURE 2-5: LOADING OF PC IN DIFFERENT SITUATIONS



2.3.1 COMPUTED GOTO

A computed `GOTO` is accomplished by adding an offset to the program counter (`ADDWF PCL`). When doing a table read using a computed `GOTO` method, care should be exercised if the table location crosses a PCL memory boundary (each 256 byte block). Refer to the application note, *"Implementing a Table Read"* (AN556).

2.3.2 STACK

The PIC16F87X family has an 8-level deep x 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. The PC is PUSHed onto the stack when a `CALL` instruction is executed, or an interrupt causes a branch. The stack is POPed in the event of a `RETURN`, `RETLW` or a `RETFIE` instruction execution. PCLATH is not affected by a PUSH or POP operation.

The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

> **Note 1:** There are no status bits to indicate stack overflow or stack underflow conditions.
>
> **2:** There are no instructions/mnemonics called PUSH or POP. These are actions that occur from the execution of the `CALL`, `RETURN`, `RETLW` and `RETFIE` instructions, or the vectoring to an interrupt address.

2.4 Program Memory Paging

All PIC16F87X devices are capable of addressing a continuous 8K word block of program memory. The `CALL` and `GOTO` instructions provide only 11 bits of address to allow branching within any 2K program memory page. When doing a `CALL` or `GOTO` instruction, the upper 2 bits of the address are provided by PCLATH<4:3>. When doing a `CALL` or `GOTO` instruction, the user must ensure that the page select bits are programmed so that the desired program memory page is addressed. If a return from a `CALL` instruction (or interrupt) is executed, the entire 13-bit PC is popped off the stack. Therefore, manipulation of the PCLATH<4:3> bits is not required for the return instructions (which POPs the address from the stack).

> **Note:** The contents of the PCLATH register are unchanged after a `RETURN` or `RETFIE` instruction is executed. The user must rewrite the contents of the PCLATH register for any subsequent subroutine calls or `GOTO` instructions.

Example 2-1 shows the calling of a subroutine in page 1 of the program memory. This example assumes that PCLATH is saved and restored by the Interrupt Service Routine (if interrupts are used).

EXAMPLE 2-1: CALL OF A SUBROUTINE IN PAGE 1 FROM PAGE 0

```
        ORG 0x500
        BCF PCLATH,4
        BSF PCLATH,3   ;Select page 1
                       ;(800h-FFFh)
        CALL SUB1_P1   ;Call subroutine in
        :              ;page 1 (800h-FFFh)
        :
        ORG 0x900      ;page 1 (800h-FFFh)
SUB1_P1
        :              ;called subroutine
                       ;page 1 (800h-FFFh)
        :
        RETURN         ;return to
                       ;Call subroutine
                       ;in page 0
                       ;(000h-7FFh)
```

2.5 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.

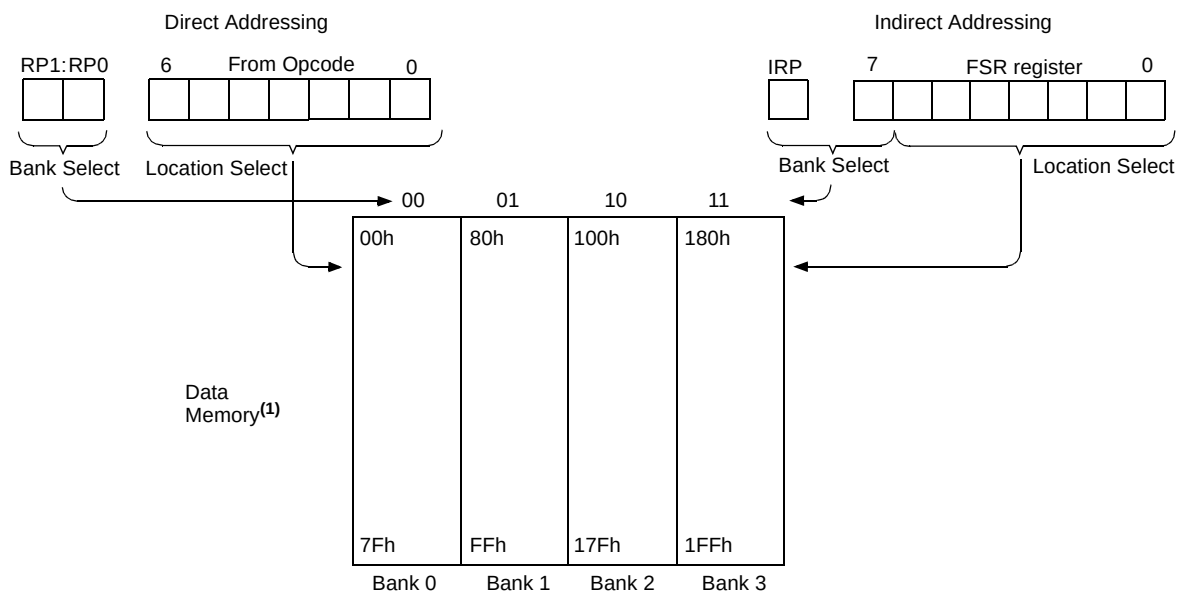
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = '0') will read 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-6.

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

EXAMPLE 2-2: INDIRECT ADDRESSING

```
        MOVLW  0x20    ;initialize pointer
        MOVWF  FSR     ;to RAM
NEXT    CLRF   INDF    ;clear INDF register
        INCF   FSR,F   ;inc pointer
        BTFSS  FSR,4   ;all done?
        GOTO   NEXT    ;no clear next
CONTINUE
        :              ;yes continue
```

FIGURE 2-6: DIRECT/INDIRECT ADDRESSING



Note 1: For register file map detail, see Figure 2-3.

6.0 TIMER1 MODULE

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L), which are readable and writable. The TMR1 Register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 Interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

- As a timer
- As a counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

In Timer mode, Timer1 increments every instruction cycle. In Counter mode, it increments on every rising edge of the external clock input.

Timer1 can be enabled/disabled by setting/clearing control bit TMR1ON (T1CON<0>).

Timer1 also has an internal “RESET input”. This RESET can be generated by either of the two CCP modules (Section 8.0). Register 6-1 shows the Timer1 control register.

When the Timer1 oscillator is enabled (T1OSCEN is set), the RC1/T1OSI/CCP2 and RC0/T1OSO/T1CKI pins become inputs. That is, the TRISC<1:0> value is ignored, and these pins read as ‘0’.

Additional information on timer modules is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Control bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON**: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

6.1 Timer1 Operation in Timer Mode

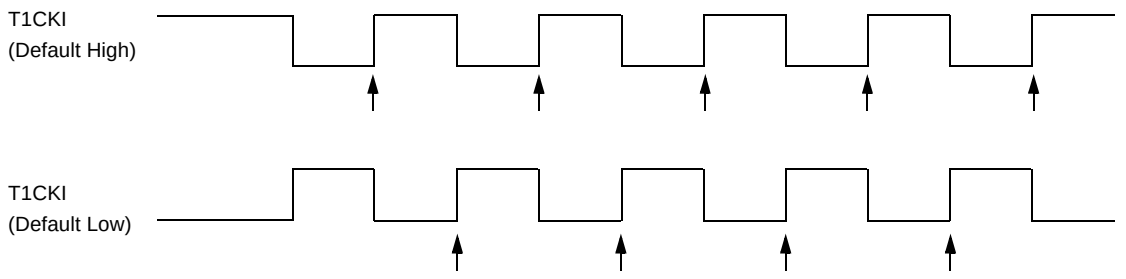
Timer mode is selected by clearing the TMR1CS (T1CON<1>) bit. In this mode, the input clock to the timer is FOSC/4. The synchronize control bit $\overline{\text{T1SYNC}}$ (T1CON<2>) has no effect, since the internal clock is always in sync.

6.2 Timer1 Counter Operation

Timer1 may operate in either a Synchronous, or an Asynchronous mode, depending on the setting of the TMR1CS bit.

When Timer1 is being incremented via an external source, increments occur on a rising edge. After Timer1 is enabled in Counter mode, the module must first have a falling edge before the counter begins to increment.

FIGURE 6-1: TIMER1 INCREMENTING EDGE



Note: Arrows indicate counter increments.

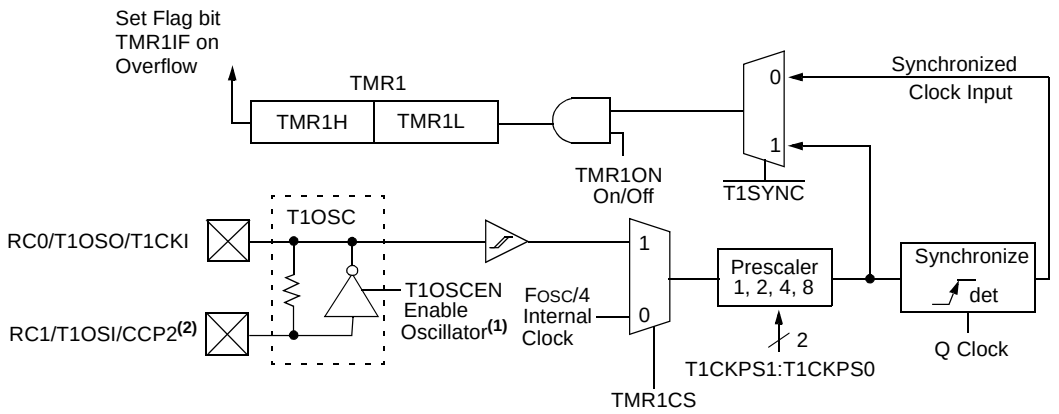
6.3 Timer1 Operation in Synchronized Counter Mode

Counter mode is selected by setting bit TMR1CS. In this mode, the timer increments on every rising edge of clock input on pin RC1/T1OSI/CCP2, when bit T1OSCEN is set, or on pin RC0/T1OSO/T1CKI, when bit T1OSCEN is cleared.

If $\overline{\text{T1SYNC}}$ is cleared, then the external clock input is synchronized with internal phase clocks. The synchronization is done after the prescaler stage. The prescaler stage is an asynchronous ripple-counter.

In this configuration, during SLEEP mode, Timer1 will not increment even if the external clock is present, since the synchronization circuit is shut-off. The prescaler, however, will continue to increment.

FIGURE 6-2: TIMER1 BLOCK DIAGRAM



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

6.4 Timer1 Operation in Asynchronous Counter Mode

If control bit $\overline{\text{T1SYNC}}$ (T1CON<2>) is set, the external clock input is not synchronized. The timer continues to increment asynchronous to the internal phase clocks. The timer will continue to run during SLEEP and can generate an interrupt-on-overflow, which will wake-up the processor. However, special precautions in software are needed to read/write the timer (Section 6.4.1).

In Asynchronous Counter mode, Timer1 cannot be used as a time-base for capture or compare operations.

6.4.1 READING AND WRITING TIMER1 IN ASYNCHRONOUS COUNTER MODE

Reading TMR1H or TMR1L while the timer is running from an external asynchronous clock, will guarantee a valid read (taken care of in hardware). However, the user should keep in mind that reading the 16-bit timer in two 8-bit values itself, poses certain problems, since the timer may overflow between the reads.

For writes, it is recommended that the user simply stop the timer and write the desired values. A write contention may occur by writing to the timer registers, while the register is incrementing. This may produce an unpredictable value in the timer register.

Reading the 16-bit value requires some care. Examples 12-2 and 12-3 in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023) show how to read and write Timer1 when it is running in Asynchronous mode.

6.5 Timer1 Oscillator

A crystal oscillator circuit is built-in between pins T1OSI (input) and T1OSO (amplifier output). It is enabled by setting control bit T1OSCEN (T1CON<3>). The oscillator is a low power oscillator, rated up to 200 kHz. It will continue to run during SLEEP. It is primarily intended for use with a 32 kHz crystal. Table 6-1 shows the capacitor selection for the Timer1 oscillator.

The Timer1 oscillator is identical to the LP oscillator. The user must provide a software time delay to ensure proper oscillator start-up.

TABLE 6-1: CAPACITOR SELECTION FOR THE TIMER1 OSCILLATOR

Osc Type	Freq.	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF
These values are for design guidance only.			
Crystals Tested:			
32.768 kHz	Epson C-001R32.768K-A		± 20 PPM
100 kHz	Epson C-2 100.00 KC-P		± 20 PPM
200 kHz	STD XTL 200.000 kHz		± 20 PPM

Note 1: Higher capacitance increases the stability of oscillator, but also increases the start-up time.

2: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.

6.6 Resetting Timer1 using a CCP Trigger Output

If the CCP1 or CCP2 module is configured in Compare mode to generate a “special event trigger” (CCP1M3:CCP1M0 = `1011`), this signal will reset Timer1.

Note: The special event triggers from the CCP1 and CCP2 modules will not set interrupt flag bit TMR1IF (PIR1<0>).

Timer1 must be configured for either Timer or Synchronized Counter mode to take advantage of this feature. If Timer1 is running in Asynchronous Counter mode, this RESET operation may not work.

In the event that a write to Timer1 coincides with a special event trigger from CCP1 or CCP2, the write will take precedence.

In this mode of operation, the CCPRxH:CCPRxL register pair effectively becomes the period register for Timer1.

6.7 Resetting of Timer1 Register Pair (TMR1H, TMR1L)

TMR1H and TMR1L registers are not reset to 00h on a POR, or any other RESET, except by the CCP1 and CCP2 special event triggers.

T1CON register is reset to 00h on a Power-on Reset, or a Brown-out Reset, which shuts off the timer and leaves a 1:1 prescale. In all other RESETS, the register is unaffected.

6.8 Timer1 Prescaler

The prescaler counter is cleared on writes to the TMR1H or TMR1L registers.

TABLE 6-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.
Note 1: Bits PSPIE and PSPIF are reserved on the PIC16F873/876; always maintain these bits clear.

9.0 MASTER SYNCHRONOUS SERIAL PORT (MSSP) MODULE

The Master Synchronous Serial Port (MSSP) module is a serial interface, useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I^2C)

Figure 9-1 shows a block diagram for the SPI mode, while Figure 9-5 and Figure 9-9 show the block diagrams for the two different I^2C modes of operation.

The Application Note AN734, “Using the PICmicro® SSP for Slave I^2C™ Communication” describes the slave operation of the MSSP module on the PIC16F87X devices. AN735, “Using the PICmicro® MSSP Module for I^2C™ Communications” describes the master operation of the MSSP module on the PIC16F87X devices.

REGISTER 9-1: SSPSTAT: SYNC SERIAL PORT STATUS REGISTER (ADDRESS: 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Sample bit
SPI Master mode:
1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time
SPI Slave mode:
SMP must be cleared when SPI is used in slave mode
In I^2C Master or Slave mode:
1 = Slew rate control disabled for standard speed mode (100 kHz and 1 MHz)
0 = Slew rate control enabled for high speed mode (400 kHz)

bit 6 **CKE**: SPI Clock Edge Select (Figure 9-2, Figure 9-3 and Figure 9-4)
SPI mode:
For CKP = 0
1 = Data transmitted on rising edge of SCK
0 = Data transmitted on falling edge of SCK
For CKP = 1
1 = Data transmitted on falling edge of SCK
0 = Data transmitted on rising edge of SCK
In I^2C Master or Slave mode:
1 = Input levels conform to SMBus spec
0 = Input levels conform to I^2C specs

bit 5 **D/A**: Data/$\overline{\text{Address}}$ bit (I^2C mode only)
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P**: STOP bit
(I^2C mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared.)
1 = Indicates that a STOP bit has been detected last (this bit is '0' on RESET)
0 = STOP bit was not detected last

bit 3 **S**: START bit
(I^2C mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared.)
1 = Indicates that a START bit has been detected last (this bit is '0' on RESET)
0 = START bit was not detected last

bit 2 **R/W**: Read/Write bit Information (I^2C mode only)
This bit holds the R/W bit information following the last address match. This bit is only valid from the address match to the next START bit, STOP bit or not $\overline{\text{ACK}}$ bit.
In I^2C Slave mode:
1 = Read
0 = Write
In I^2C Master mode:
1 = Transmit is in progress
0 = Transmit is not in progress
Logical OR of this bit with SEN, RSEN, PEN, RCEN, or ACKEN will indicate if the MSSP is in IDLE mode.

bit 1 **UA**: Update Address (10-bit I^2C mode only)
1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit **BF**: Buffer Full Status bit
Receive (SPI and I^2C modes):
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty
Transmit (I^2C mode only):
1 = Data transmit in progress (does not include the $\overline{\text{ACK}}$ and STOP bits), SSPBUF is full
0 = Data transmit complete (does not include the $\overline{\text{ACK}}$ and STOP bits), SSPBUF is empty

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 9-2: SSPCON: SYNC SERIAL PORT CONTROL REGISTER (ADDRESS 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL**: Write Collision Detect bit
Master mode:
1 = A write to SSPBUF was attempted while the I2C conditions were not valid
0 = No collision
Slave mode:
1 = SSPBUF register is written while still transmitting the previous word (must be cleared in software)
0 = No collision

bit 6 **SSPOV**: Receive Overflow Indicator bit
In SPI mode:
1 = A new byte is received while SSPBUF holds previous data. Data in SSPSR is lost on overflow. In Slave mode, the user must read the SSPBUF, even if only transmitting data, to avoid overflows. In Master mode, the overflow bit is not set, since each operation is initiated by writing to the SSPBUF register. (Must be cleared in software.)
0 = No overflow
In I^2C mode:
1 = A byte is received while the SSPBUF is holding the previous byte. SSPOV is a "don't care" in Transmit mode. (Must be cleared in software.)
0 = No overflow

bit 5 **SSPEN**: Synchronous Serial Port Enable bit
In SPI mode,
When enabled, these pins must be properly configured as input or output
1 = Enables serial port and configures SCK, SDO, SDI, and SS as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins
In I^2C mode,
When enabled, these pins must be properly configured as input or output
1 = Enables the serial port and configures the SDA and SCL pins as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins

bit 4 **CKP**: Clock Polarity Select bit
In SPI mode:
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level
In I^2C Slave mode:
SCK release control
1 = Enable clock
0 = Holds clock low (clock stretch). (Used to ensure data setup time.)
In I^2C Master mode:
Unused in this mode

bit 3-0 **SSPM3:SSPM0**: Synchronous Serial Port Mode Select bits
0000 = SPI Master mode, clock = FOSC/4
0001 = SPI Master mode, clock = FOSC/16
0010 = SPI Master mode, clock = FOSC/64
0011 = SPI Master mode, clock = TMR2 output/2
0100 = SPI Slave mode, clock = SCK pin. $\overline{SS}$ pin control enabled.
0101 = SPI Slave mode, clock = SCK pin. $\overline{SS}$ pin control disabled. $\overline{SS}$ can be used as I/O pin.
0110 = I^2C Slave mode, 7-bit address
0111 = I^2C Slave mode, 10-bit address
1000 = I^2C Master mode, clock = FOSC / (4 * (SSPADD+1))
1011 = I^2C Firmware Controlled Master mode (slave idle)
1110 = I^2C Firmware Controlled Master mode, 7-bit address with START and STOP bit interrupts enabled
1111 = I^2C Firmware Controlled Master mode, 10-bit address with START and STOP bit interrupts enabled
1001, 1010, 1100, 1101 = Reserved

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

REGISTER 9-3: SSPCON2: SYNC SERIAL PORT CONTROL REGISTER2 (ADDRESS 91h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

bit 7 **GCEN**: General Call Enable bit (In I^2C Slave mode only)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled

bit 6 **ACKSTAT**: Acknowledge Status bit (In I^2C Master mode only)
In Master Transmit mode:
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT**: Acknowledge Data bit (In I^2C Master mode only)
In Master Receive mode:
Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.
1 = Not Acknowledge
0 = Acknowledge

bit 4 **ACKEN**: Acknowledge Sequence Enable bit (In I^2C Master mode only)
In Master Receive mode:
1 = Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit. Automatically cleared by hardware.
0 = Acknowledge sequence idle

bit 3 **RCEN**: Receive Enable bit (In I^2C Master mode only)
1 = Enables Receive mode for I^2C
0 = Receive idle

bit 2 **PEN**: STOP Condition Enable bit (In I^2C Master mode only)
SCK Release Control:
1 = Initiate STOP condition on SDA and SCL pins. Automatically cleared by hardware.
0 = STOP condition idle

bit 1 **RSEN**: Repeated START Condition Enable bit (In I^2C Master mode only)
1 = Initiate Repeated START condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated START condition idle

bit 0 **SEN**: START Condition Enable bit (In I^2C Master mode only)
1 = Initiate START condition on SDA and SCL pins. Automatically cleared by hardware.
0 = START condition idle

Note: For bits ACKEN, RCEN, PEN, RSEN, SEN: If the I^2C module is not in the IDLE mode, this bit may not be set (no spooling), and the SSPBUF may not be written (or writes to the SSPBUF are disabled).

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	’1’ = Bit is set	’0’ = Bit is cleared	x = Bit is unknown

9.1 SPI Mode

The SPI mode allows 8 bits of data to be synchronously transmitted and received simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

- Serial Data Out (SDO)
- Serial Data In (SDI)
- Serial Clock (SCK)

Additionally, a fourth pin may be used when in a Slave mode of operation:

- Slave Select ($\overline{SS}$)

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits (SSPCON<5:0> and SSPSTAT<7:6>). These control bits allow the following to be specified:

- Master mode (SCK is the clock output)
- Slave mode (SCK is the clock input)
- Clock Polarity (Idle state of SCK)
- Data input sample phase
  (middle or end of data output time)
- Clock edge
  (output data on rising/falling edge of SCK)
- Clock Rate (Master mode only)
- Slave Select mode (Slave mode only)

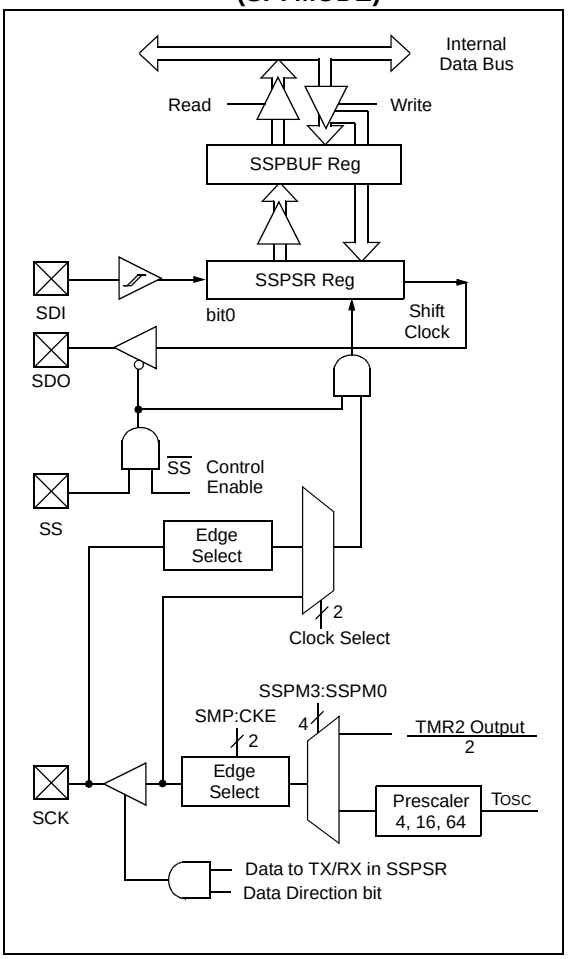
Figure 9-4 shows the block diagram of the MSSP module when in SPI mode.

To enable the serial port, MSSP Enable bit, SSPEN (SSPCON<5>) must be set. To reset or reconfigure SPI mode, clear bit SSPEN, re-initialize the SSPCON registers, and then set bit SSPEN. This configures the SDI, SDO, SCK and $\overline{SS}$ pins as serial port pins. For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS register) appropriately programmed. That is:

- SDI is automatically controlled by the SPI module
- SDO must have TRISC<5> cleared
- SCK (Master mode) must have TRISC<3> cleared
- SCK (Slave mode) must have TRISC<3> set
- $\overline{SS}$ must have TRISA<5> set and register ADCON1 (see Section 11.0: A/D Module) must be set in a way that pin RA5 is configured as a digital I/O

Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

FIGURE 9-1: MSSP BLOCK DIAGRAM (SPI MODE)



9.1.1 MASTER MODE

The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave (Processor 2, Figure 9-5) is to broadcast data by the software protocol.

In Master mode, the data is transmitted/received as soon as the SSPBUF register is written to. If the SPI module is only going to receive, the SDO output could be disabled (programmed as an input). The SSPSR register will continue to shift in the signal present on the SDI pin at the programmed clock rate. As each byte is received, it will be loaded into the SSPBUF register as if a normal received byte (interrupts and status bits appropriately set). This could be useful in receiver applications as a “line activity monitor”.

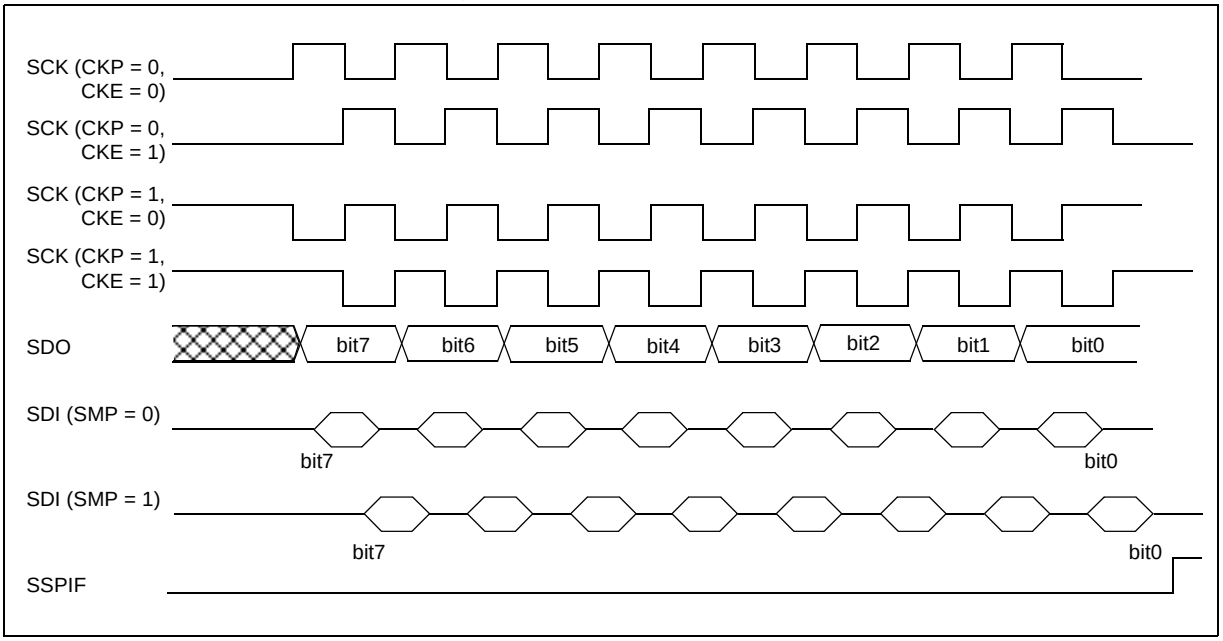
The clock polarity is selected by appropriately programming bit CKP (SSPCON<4>). This then, would give waveforms for SPI communication as shown in Figure 9-6, Figure 9-8 and Figure 9-9, where the MSb is transmitted first. In Master mode, the SPI clock rate (bit rate) is user programmable to be one of the following:

- FOSC/4 (or TCY)
- FOSC/16 (or 4 • TCY)
- FOSC/64 (or 16 • TCY)
- Timer2 output/2

This allows a maximum bit clock frequency (at 20 MHz) of 5.0 MHz.

Figure 9-6 shows the waveforms for Master mode. When CKE = 1, the SDO data is valid before there is a clock edge on SCK. The change of the input sample is shown based on the state of the SMP bit. The time when the SSPBUF is loaded with the received data is shown.

FIGURE 9-2: SPI MODE TIMING, MASTER MODE



9.1.2 SLAVE MODE

In Slave mode, the data is transmitted and received as the external clock pulses appear on SCK. When the last bit is latched, the interrupt flag bit SSPIF (PIR1<3>) is set.

While in Slave mode, the external clock is supplied by the external clock source on the SCK pin. This external clock must meet the minimum high and low times as specified in the electrical specifications.

While in SLEEP mode, the slave can transmit/receive data. When a byte is received, the device will wake-up from SLEEP.

> **Note 1:** When the SPI module is in Slave mode with $\overline{SS}$ pin control enabled (SSPCON<3:0> = `0100`), the SPI module will reset if the $\overline{SS}$ pin is set to VDD.
>
> **2:** If the SPI is used in Slave mode with CKE = '1', then $\overline{SS}$ pin control must be enabled.

FIGURE 9-3: SPI MODE TIMING (SLAVE MODE WITH CKE = 0)

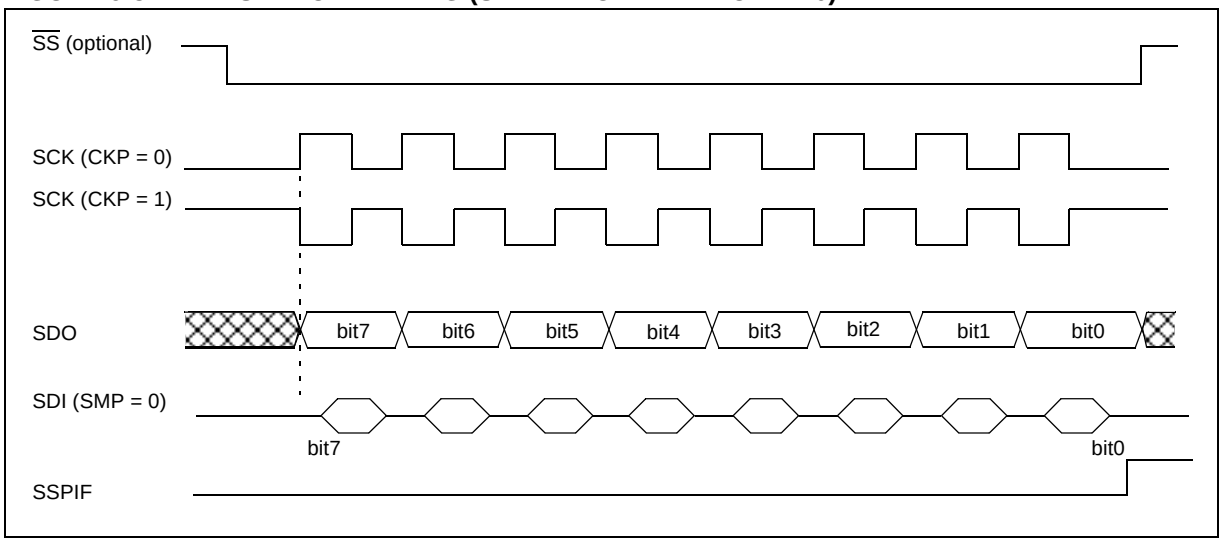


FIGURE 9-4: SPI MODE TIMING (SLAVE MODE WITH CKE = 1)

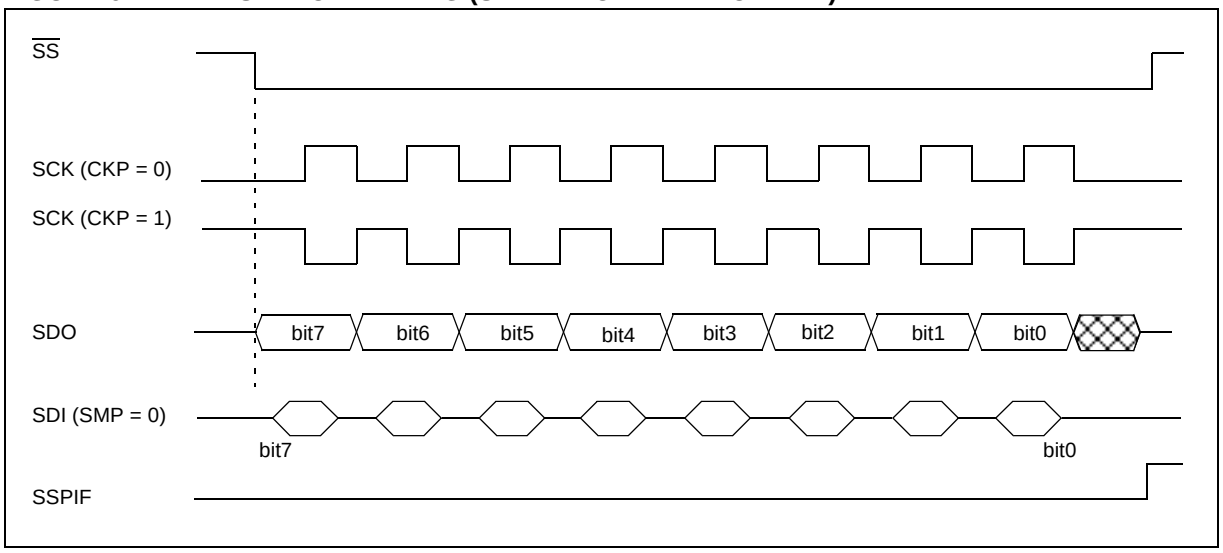


TABLE 9-1: REGISTERS ASSOCIATED WITH SPI OPERATION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on: MCLR, WDT
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the SSP in SPI mode.
Note 1: These bits are reserved on PCI16F873/876 devices; always maintain these bits clear.

B.2. VS1001K

Circuito integrado ***Decodificador MPEG****. Funciona con una alimentación de 3.3 voltios*

VS1001k - MPEG AUDIO CODEC

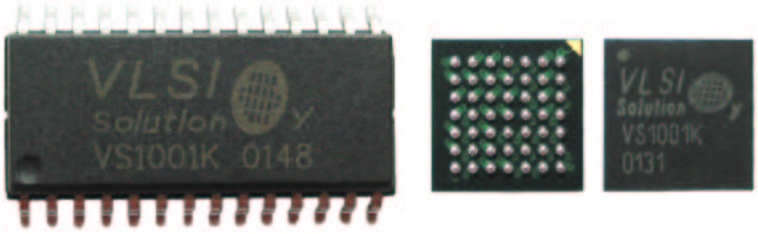
Features

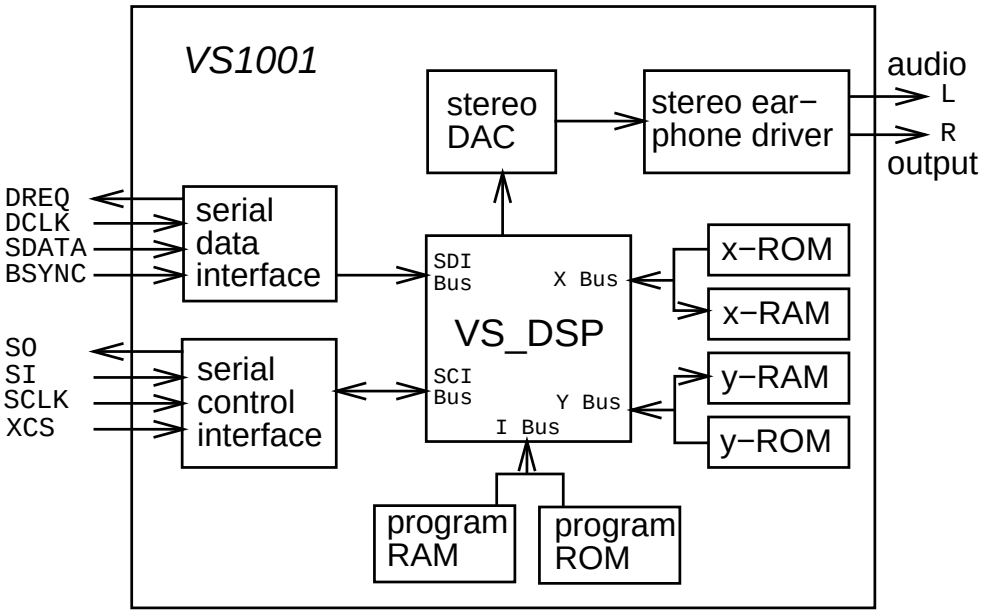
- MPEG audio layer 3 decoder (ISO11172-3)
- Supports MPEG 1 & 2, and 2.5 extensions, all their sample rates and bit rates, in mono and stereo
- Supports PCM input
- Supports VBR (variable bitrate)
- Can be used as a slave co-processor
- Operates with single clock 12..13 MHz or 24..26 MHz
- Extremely low-power operation
- On-chip high-quality stereo DAC with no phase error between channels
- Internal Op-Amp in BGA-49 and LQFP-48 packages
- Stereo earphone driver capable of driving a 30Ω load.
- Separate 2.5 .. 3.6V operating voltages for analog and digital
- 4 KiB On-chip RAM for user code
- Serial control and data interfaces
- New functions may be added with software

Description

VS1001k is a single-chip solution for an MPEG layer 3 audio decoder. The chip contains a high-performance low-power DSP processor (VS_DSP), working memory, 4 KiB program RAM and 0.5 KiB data RAM for user applications, serial control and input data interfaces, and a high-quality oversampling variable-sample-rate stereo DAC, followed by an earphone amplifier and a ground buffer.

VS1001k receives its input bitstream through a serial input bus, which it listens to as a system slave. The input stream is decoded and passed through a analog/digital hybrid volume control to an 18-bit oversampling multi-bit sigma-delta DAC. The decoding is controlled via a serial control bus. In addition to the basic decoding, it is possible to add application specific features, like DSP effects, to the user RAM memory.





3 Packages and Pin Descriptions

3.1 SOIC-28

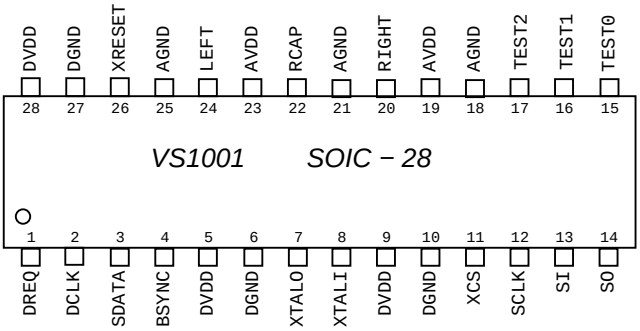


Figure 1: Pin Configuration, SOIC-28.

Pin Name	Pin	Pin Type	Function
DREQ	1	DO	data request, input bus
DCLK	2	DIO	serial input data bus clock
SDATA	3	DI	serial data input
BSYNC	4	DI	byte synchronization signal
DVDD1	5	PWR	digital power supply
DGND1	6	PWR	digital ground
XTALO	7	CLK	crystal output
XTALI	8	CLK	crystal input
DVDD2	9	PWR	digital power supply
DGND2	10	PWR	digital ground
XCS	11	DI	chip select input (active low)
SCLK	12	DI	clock for serial bus
SI	13	DI	serial input
SO	14	DO3	serial output
TEST0	15	DI	reserved for test, connect to DVDD
TEST1	16	DO	reserved for test, *do not connect!*
TEST2	17	DO	reserved for test, *do not connect!*
AGND1	18	PWR	analog ground
AVDD1	19	PWR	analog power supply
RIGHT	20	AO	right channel output
AGND2	21	PWR	analog ground
RCAP	22	AIO	capacitance for reference
AVDD2	23	PWR	analog power supply
LEFT	24	AO	left channel output
AGND3	25	PWR	analog ground
XRESET	26	DI	active low asynchronous reset
DGND3	27	PWR	digital ground
DVDD3	28	PWR	digital power supply

Pin types:

Type	Description
DI	Digital input, CMOS Input Pad
DO	Digital output, CMOS Input Pad
DIO	Digital input/output
DO3	Digital output, CMOS Tri-stated Output Pad

Type	Description
AI	Analog input
AO	Analog output
AIO	Analog input/output
PWR	Power supply pin

SOIC-28 package dimensions can be found at *http://www.vlsi.fi/vs1001/soic28.pdf* .

4 Connection Diagram, SOIC-28

In this connection diagram, a SOIC-28 -packaged VS1001k is used.

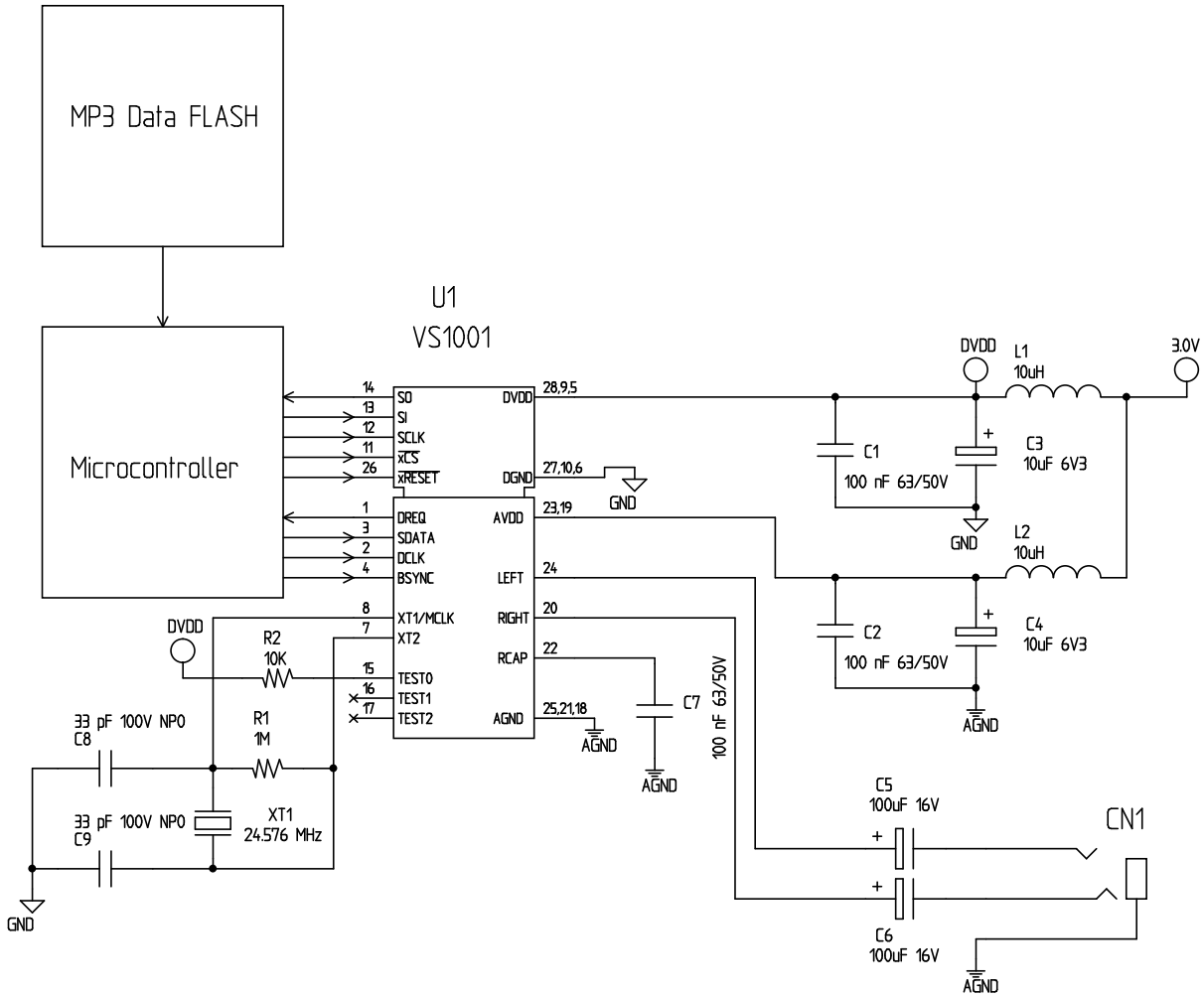


Figure 4: Typical Connection Diagram Using SOIC-28.

Ground buffer is not available for the SOIC-28 package; hence it is not used.

6 SPI Buses

6.1 General

The SPI Bus - that was originally used in some Motorola devices - has been used for both VS1001k's Serial Data Interface SDI (Chapters 6.3 and 7.3) and Serial Control Interface SCI (Chapters 6.4 and 7.4).

6.2 SPI Bus Pin Descriptions

SDI Pin	SCI Pin	Description
-	XCS	Active low chip select input. A high level forces the serial interface into standby mode, ending the current operation. A high level also forces serial output (SO) to high impedance state. There is no chip select for SDI, which is always active.
DCLK	SCK	Serial clock input. The serial clock is also used internally as the master clock for the register interface. SCK can be gated or continuous. In either case, the first rising clock edge after XCS has gone low marks the first bit to be written (clock 0 in the following figures).
SDATA	SI	Serial input. SI is sampled on the rising SCK edge, if XCS is low.
-	SO	Serial output. In reads, data is shifted out on the falling SCK edge. In writes SO is at a high impedance state.

6.3 Serial Protocol for Serial Data Interface (SDI)

The serial data interface can operate in either master or slave mode. In master mode, VS1001k generates the DCLK signal, which can be selected to be either 512 or 1024 kHz. In slave mode, the DCLK signal is generated by an external circuit.

The data (SDATA signal) can be clocked in at either the rising or falling edge of the DCLK. (Chapter 7.5).

The VS1001k chip assumes its input to be byte-sychronized. I.e. the internal operation of the decoder does not search for byte synchronization of the frames from the data stream, but instead assumes the data to be correctly byte-aligned. The bytes can be transmitted either MSB or LSB first, depending of contents of SCI register MODE (Chapter 7.5).

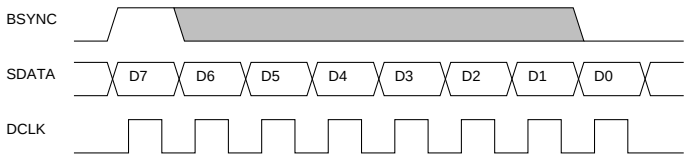


Figure 6: BSYNC Signal.

To ensure correct byte-alignment of the input bitstream, the serial data interface has a BSYNC signal.

The first DCLK sampling edge (rising or falling, depending on selected polarity), during which the BSYNC is high, marks the first bit of a byte (LSB, if LSB-first order is used, MSB, if MSB-first order is used). If BSYNC is not used, it must be tied to VCC externally and the master of the input serial interface must always sustain the correct byte-alignment. Using BSYNC is strongly recommended. For more details, look at the Application Notes for VS10XX.

The DREQ signal of the data interface is used in slave mode to signal if VS1001k's FIFO is capable of receiving more input data. If DREQ is high, VS1001k can take at least 32 bytes of data. When there is less than 32 bytes of free space, DREQ is turned low, and the sender should stop transferring new data. Because of the 32-byte safety area, the sender may send upto 32 bytes of data at a time without checking the status of DREQ, making controlling VS1001k easier for low-speed microcontrollers.

Note: DREQ may turn low or high at any time, even during a byte transmission. Thus, DREQ should only be used to decide whether to send more bytes. It should not abort a transmission that has already started.

6.4 Serial Protocol for Serial Command Interface (SCI)

6.4.1 General

The serial bus protocol for the Serial Command Interface SCI (Chapter 7.4) consists of an instruction byte, address byte and one 16-bit data word. Each read or write operation can read or write a single register. Data bits are read at the rising edge, so the user should not update data at the rising edge.

The operation is specified by an 8-bit instruction opcode. The supported instructions are read and write. See table below.

Instruction		
Name	Opcode	Operation
READ	0000 0011	Read data
WRITE	0000 0010	Write data

Note: After using the Serial Command Interface, it is not allowed to send SCI or SDI data for 5 microseconds.

6.4.2 SCI Read

VS1001k registers are read by the following sequence. First, XCS line is pulled low to select the device. Then the READ opcode (0x3) is transmitted via the SI line followed by an 8-bit word address. After the address has been read in, any further data on SI is ignored. The 16-bit data corresponding to the received address will be shifted out onto the SO line.

XCS should be driven high after the data has been shifted out. In that case, the word address will be incremented and data corresponding to the next address will be shifted out. After the last word has been shifted out, XCS should be driven high to end the READ sequence.

Word read is shown in Figure 7.

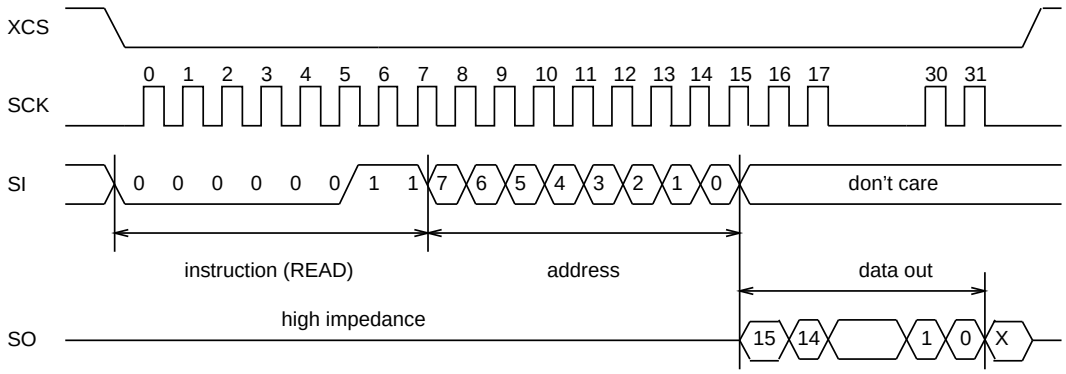


Figure 7: SCI Word Read

6.4.3 SCI Write

VS1001k registers are written by the following sequence. First, XCS line is pulled low to select the device. Then the WRITE opcode (0x2) is transmitted via the SI line followed by an 8-bit word address.

After the word has been shifted in, XCS should be pulled high to end the WRITE sequence. XCS low to high transition must occur after SCLK high to low transition corresponding to LSB of the last word.

Single word write is shown in Figure 8.

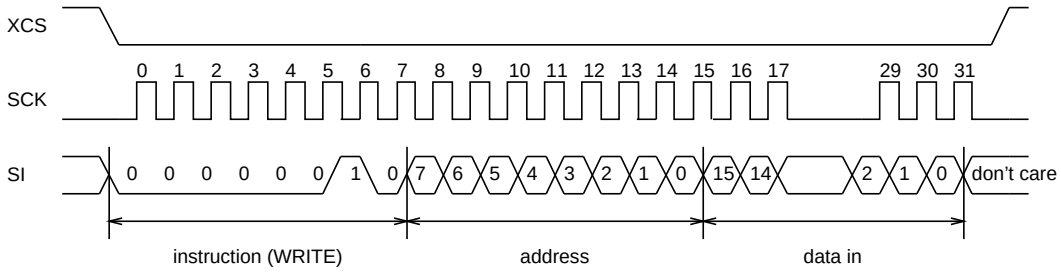


Figure 8: SCI Word Write

6.5 SPI Timing Diagram

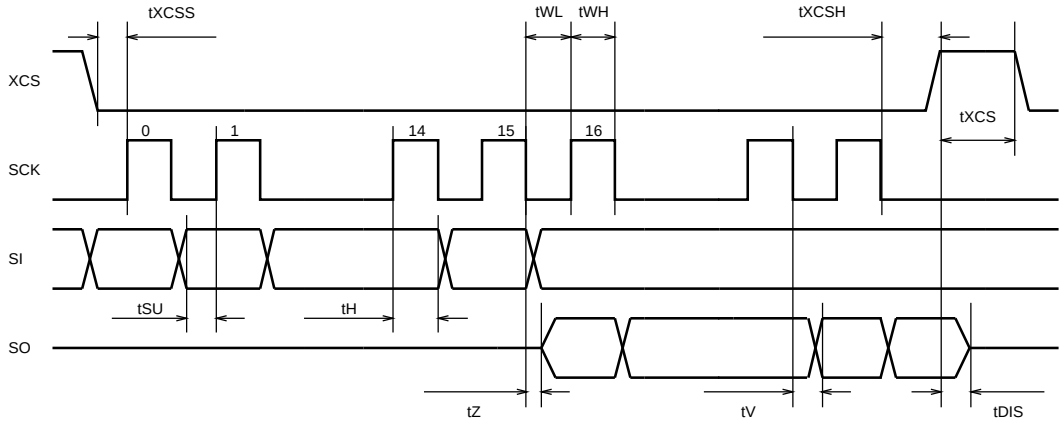


Figure 9: SPI Timing Diagram.

Symbol	Min	Max	Unit
tXCSS	5		ns
tSU	10		ns
tH	42		ns
tZ		42	ns
tWL	100		ns
tWH	100		ns
tV		42	ns
tXCSH	10		ns
tXCS	2		XTALI cycles
tDIS		1	XTALI cycles

Note: As tXCS must be at least 2 clock cycles, the maximum speed for the SPI bus is 1/4 of VS1001k's internal clock speed. For details, see Application Notes for VS10XX.

7 Functional Description

7.1 Main Features

VS1001k is based on a proprietary digital signal processor, VS_DSP. It contains all the code and data memory needed for MPEG audio decoding, together with serial interfaces, a multirate stereo audio DAC and analog output amplifiers and filters.

VS1001k can play all MPEG 1 and 2 layer 3 files, as well as so-called MPEG 2.5 layer 3 extension files with all sample rates and bitrates. In addition, variable bitrate (VBR) is also supported. With VBR, and depending on the song, near-cd quality can be achieved with approximately 100 kbits/s for stereo music sampled at 44100 Hz, whereas old encoders required 128 kbits/s for the same task. As both commercial and free (*http://www.mp3dev.org/*) high-quality VBR encoders are nowadays widely available, MP3 format is getting better as it is maturing.

7.2 Data Flow of VS1001k

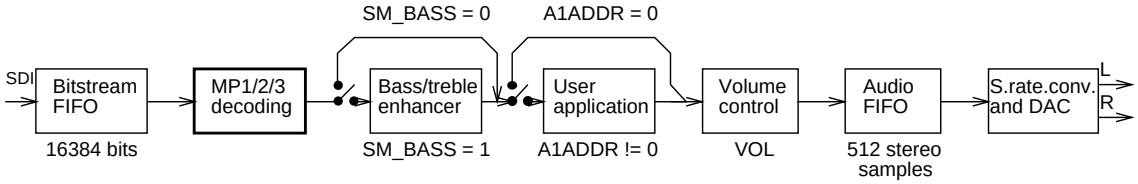


Figure 10: Data Flow of VS1001k.

First, MP3 data is input through the SDI bus.

After decoding, data may be sent to the Bass/treble enhancer depending on SCI register MODE's bit SM_BASS.

Then, if SCI register AIADDR is non-zero, application code is executed from the address pointed to by AIADDR. For more details, see Chapters 7.5.10 and Application Notes for VS10XX.

After the optional user application, the signal is fed to the volume control unit, which also copies the data to the Audio FIFO.

The Audio FIFO holds the data that is read by the Audio interrupt (Chapter 9.5.1) and fed to the sample rate converter and DACs. The size of the audio FIFO is 512 stereo (2×16-bit) samples

The sample rate converter converts all different sample rates to CLKI/512 and feeds the data to the DAC, which in order makes a stereo in-phase signal. This signal is then forwarded to the earphone amplifier.

7.3 Serial Data Interface (SDI)

The serial data interface is meant for transferring compressed MPEG audio data.

Also several different tests may be activated through SDI as described in Chapter 8.

7.4 Serial Control Interface (SCI)

The serial control interface is compatible with the SPI bus specification. Data transfers are always 16-bits. The VS1001k is controlled by writing and reading the registers of the interface.

The main controls of the control interface are:

- control of the operation mode
- uploading user programs
- access to header data
- status information
- access to decoded digital data
- feeding input data

7.5 SCI Registers

Name	Type	addr	Function
MODE	RW	0	mode control
STATUS	RW	1	status of VS1001k
INT_FCTLH	-	2	internal register, never use
CLOCKF	RW	3	clock freq + doubler
DECODE_TIME	R	4	decode time in seconds
AUDATA	R	5	misc. audio data
WRAM	W	6	RAM write program
WRAMADDR	W	7	base address for RAM write
HDAT0	R	8	read header data
HDAT1	R	9	read header data
AIADDR	RW	10	start address of application
VOL	RW	11	volume control
RESERVED	-	12	reserved for VS1002 use, don't touch
AICTRL[x]	RW	13+x	2 application control registers

x = [0 .. 1]

All registers are filled with zeros at hardware reset.

7.5.1 MODE (RW)

MODE is used to control the operation of VS1001k.

Bit	Name	Function	Value	Description
0	SM_DIFF	differential	0 1	normal in-phase audio left channel inverted
1	SM_FFWD	fast forward	0 1	normal playback fast forward on
2	SM_RESET	soft reset	0 1	no reset reset
3	SM_UNUSED1	set to 0	0	Set to 0
4	SM_PDOWN	powerdown	0 1	power on powerdown
5	SM_UNUSED2	set to 0	0	Set to 0
6	SM_UNUSED3	set to 0	0	Set to 0
7	SM_BASS	bass/treble enhancer	0 1	off on
8	SM_DACT	DCLK active edge	0 1	rising falling
9	SM_BYTEORD	Byte order on serial input bus	0 1	MSB first MSB last
10	SM_IBMODE	input bus mode	0 1	slave master
11	SM_IBCLK	input bus clk when VS1001k is master	0 1	512 kHz 1024 kHz

When SM_DIFF is set, the player inverts the left output. For a stereo input, this creates a virtual surround, and for a mono input this effectively creates a differential left/right signal.

By setting SM_FFWD the player starts to accept SCI data at a high speed, and just decodes the audio headers silently without playing any data. This can be used to fast-forward data with safe landing. Register DECODE_TIME is updated during a fast-forward just as normal.

By setting SM_RESET to 1, the player is reset.

SM_UNUSED1 should always be set to 0.

Bit SM_PDOWN overrides any other: it turns VS1001k into powerdown mode, where the only operational part is the control bus.

SM_UNUSED2 and SM_UNUSED3 should always be set to 0.

Bit SM_BASS turns on the built-in Bass and Treble enhancer. The frequency response of the enhancer when the sample rate is 44.1 kHz is shown in Figure 11. For other sample frequencies the response frequence axis must be adjusted accordingly. Example: If the sample rate is 48 kHz, the 1 kHz frequency in the figure is actually 1 kHz $\times$ 48 kHz / 44.1 kHz = 1.09 kHz. For details of how much extra processing

power is needed when activating this feature, see Application Notes for VS10XX.

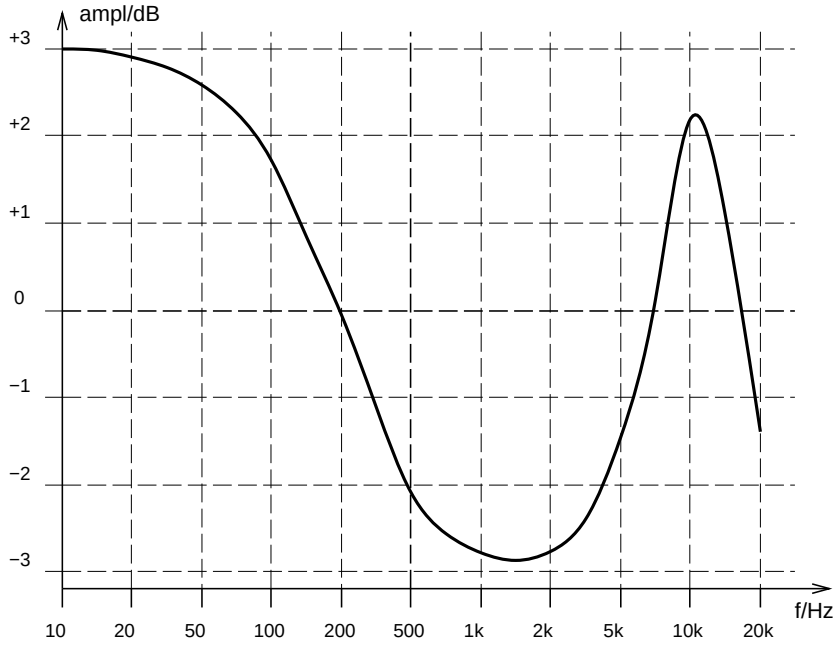


Figure 11: Built-In Bass/Treble Enhancer Frequency Response at 44.1 kHz.

SM_DACT defines the active edge of data clock for SDI.

SM_BYTEORD defines the data order inside a byte for SDI. Bytes are, however, still sent in the default order.

SM_IBMODE sets input bus to master mode. Master mode has not been tested, and its use is not recommended.

SM_IBCLK sets the bus clock speed when VS1001k is the master.

7.5.2 STATUS (RW)

STATUS contains information on the current status of the VS1001k. Bits 1 and 0 are used to control analog output volume: 0 = -0 dB, 1 = -6 dB, 3 = -12 dB. Bit 2 is analog powerdown bit. When set to 1, analog is put to powerdown.

Note: writing to register VOL will automatically set the analog output volume, and muting if necessary. Thus, the user needn't worry about this register.

7.5.3 INT_FCNTLH (-)

INT_FCTLH is not a user-accessible register.

7.5.4 CLOCKF (RW)

CLOCKF is used to tell if the input clock XTALI is running at something else than 24.576 MHz. XTALI is set in 2 kHz steps. Thus, the formula for calculating the correct value for this register is $XTALI/2000$ (XTALI is in Hz). Values may be between 0..32767, although hardware limits the highest allowed speed. Also, with lower-than 24.576 MHz speeds all sample rates and bit-stream widths are no longer available.

Setting the MSB of CLOCKF to 1 activates internal clock-doubling. A clock of upto 15 MHz may be doubled depending on the voltage provided to the chip.

Note: CLOCKF must be set before beginning decoding MP3 data; otherwise the sample rate will not be set correctly.

Example 1: For a 26 MHz clock the value would be $26000000/2000 = 13000$.

Example 2: For a 13 MHz external clock and using internal clock-doubling for a 26 MHz internal frequency, the value would be $0x8000 + (13000000/2000) = 39268$.

Example 3: For a 24.576 MHz clock the value would be either $24576000/2000 = 12288$, or just the default value 0. For this clock frequency, CLOCKF doesn't need to be set at all.

7.5.5 DECODE_TIME (R)

When decoding correct data, current decoded time is shown in this register in full seconds.

7.5.6 AUDATA (R)

When decoding correct data, the current bitrate in kbits/s can be found in bits 8..0 of AUDATA. For a variable bitrate bitstream, the current bitstream width is displayed. Bits 12..9 contains an index to the sample rate. The indices are shown in the table below. Bits 14..13 are not in use and always set to 0. Bit 15 is 0 for mono data and 1 for stereo.

Bits 12..9	Sample Rate/Hz
0b0000	Unknown
0b0001	44100
0b0010	48000
0b0011	32000
0b0100	22050
0b0101	24000
0b0110	16000
0b0111	11025
0b1000	12000
0b1001	8000

7.5.7 WRAM (W)

WRAM is used to upload application programs to program RAM. The start address must be initialized by writing to the WRAMADDR register prior to the first call of WRAM. value will be used. As 16 bits of data can be transferred with one WRAM write, and the program word is 32 bits, two consecutive writes are needed for each program word. The byte order is big-endian (i.e. MSBs first). After each full-word write, the internal pointer is autoincremented.

7.5.8 WRAMADDR (W)

WRAMADDR is used to set the program address for following WRAM writes. User program space is between addresses 0x4000 .. 0x43ff (with addresses 0x4000 .. 0x401f being reserved by the system), but for writes through the WRAM mechanism, they are visible at addresses 0x4000 higher. Thus, if the programmer wish to write his application to address 0x4167, he should write 0x4167 + 0x4000 = 0x8167 to WRAMADDR.

7.5.9 HDAT0 and HDAT1 (R)

Bit	Function	Value	Explanation
HDAT1[15:5]	syncword	2047	stream valid
HDAT1[4:3]	ID	3	ISO 11172-3 1.0
		2	MPG 2.0 (1/2-rate)
		1	MPG 2.5 (1/4-rate)
		0	MPG 2.5 (1/4-rate)
HDAT1[2:1]	layer	3	I
		2	II
		1	III
		0	reserved
HDAT1[0]	protect bit	1	No CRC
		0	CRC protected
HDAT0[15:12]	bitrate		ISO 11172-3
HDAT0[11:10]	sample rate	3	reserved
		2	32/16/8 kHz
		1	48/24/12 kHz
		0	44/22/11 kHz
HDAT0[9]	pad bit	1	additional slot
		0	normal frame
HDAT0[8]	private bit		not defined
HDAT0[7:6]	mode	3	mono
		2	dual channel
		1	joint stereo
		0	stereo
HDAT0[5:4]	extension		ISO 11172-3
HDAT0[3]	copyright	1	copyrighted
		0	free
HDAT0[2]	original	1	original
		0	copy
HDAT0[1:0]	emphasis	3	CCITT J.17
		2	reserved
		1	50/15 microsec
		0	none

When read, HDAT0 and HDAT1 contain header information that is extracted from MPEG stream being currently being decoded. Right after resetting VS1001k, 0 is automatically written to both registers, indicating no data has been found yet.

The “sample rate” field in HDAT0 is interpreted as follows: if the “ID” field in HDAT1 is ’1’, the highest sample rate is used. If “ID” is ’0’, half sample rate is used. For ’2’ and ’3’, the lowest sample rate is used.

Note: The sample rate, stereo/mono and bitrate information can more easily be read from register AUDATA.

7.5.10 AIADDR (RW)

AIADDR indicates the start address of the application code written earlier through WRAMADDR and WRAM registers. If no application code is used, this register should not be initialized, or it should be initialized to zero. For more details, see Application Notes for VS10XX.

7.5.11 VOL (RW)

VOL is a volume control for the player hardware. For each channel, a value in the range of 0 .. 255 may be defined to set its attenuation from the maximum volume level (in 0.5 dB steps). The left channel value is then multiplied by 256 and the values are added. Thus, maximum volume is 0 and total silence if 65535. Example: for a volume of -2.0 dB for the left channel and -3.5 dB for the right channel: (4*256) + 7 = 1031. Note, that at startup volume is set to full volume. Resetting the software does not reset the volume setting.

Note: Setting the volume to total silence (255 for both left and right channels), will turn analog power off. This will save power, but also cause a slight snap in the earphones. If you want to turn the volume off but don't want this snap, turn the volume only to 254 for both channels (0xFEFE).

7.5.12 RESERVED (RW)

This register has been reserved for future use.

7.5.13 AICTRL[x] (RW)

AICTRL[x] -registers (x=[0 .. 1]) can be used to access the user's application program.

7.6 Stereo Audio DAC

The decoded digital data is transformed into analog format by an 18-bit oversampling multi-bit sigma-delta DA-converter. The oversampled output is low-pass filtered by an on-chip analog filter. The output rate of the DA-converter is always 1/4 of the clock rate, or 128 times the highest usable sample rate. For instance for a 24.576 MHz clock, the DA-converter operates at 128x48 kHz, which is 6.144 MHz. If the input sample rate is other than 48 kHz, it is internally converted to 48 kHz by the DAC. This removes the need for complex PLL-based clocking schemes and still allows the use of several sample rates with one fixed master clock frequency.

The outputs can be separately muted by the user. If the output of the decoder is invalid or input data is not received fast enough, analog outputs are automatically muted. The analog outputs have buffers that are capable of driving 30Ω loads with a maximum of 50nF capacitance.

8 Operation

8.1 Clocking

The VS1001k chip operates typically on a single 24.576 MHz fundamental frequency master clock. This clock can be generated by external circuitry (connected to pin XTALI) or by the internal clock chrystal interface (pins XTALI and XTALO). This clock is sufficient to support a high quality audio output for almost all the standard sample rates and bit-rates (see Application Notes for VS10XX).

Note: Oscillators above 24.576 MHz are usually so-called 3rd harmonic clocks, which have a fundamental frequency of 1/3 of the nominal clock frequency. With such an oscillator, VS1001 would be running at the base frequency, if working at all. Thus, for instance, if you run VS1001 with a 32 MHz 3rd harmonic clock, you usually end up running the chip at 32 MHz / 3 = 10.67 MHz.

8.2 Powerdown

In powerdown mode the chip only monitors the control bus. The analog output drivers are turned off and the processor remains in hold-state.

8.3 Hardware Reset

When the XRESET -signal is driven low, VS1001k is reset and all the control registers and internal states are set to the initial values. XRESET-signal is asynchronous to any external clock. The reset mode doubles as a full-powerdown mode, where both digital and analog parts of VS1001k are in minimum power consumption stage, and where clocks are stopped. Also XTALO and XTALI are grounded.

After a hardware reset (or at power-up), set the basic software registers such as VOL for volume (and CLOCKF if the input clock is anything else than 24.576 MHz) before starting decoding.

8.4 Software Reset

Between any two MP3 files, the decoder software has to be reset. This is done by activating bit 2 in SCI's MODE register (Chapter 7.5.1). Then wait for at least 2 μs, then look at DREQ. DREQ will stay down for at least 6000 clock cycles, which means an approximate 250 μs delay if VS1001k is run at 24.576 MHz. When DREQ goes up, write at least one zero to SDI. After this, you may continue playback as usual.

If you want to make sure VS1001k doesn't cut the ending of low-bitrate data streams, it is recommended to feed 2048 zeros to the SDI bus before activating the reset bit (DREQ must be respected just as with normal SDI data). This will make sure all frames have been decoded before resetting the chip.

8.5 Play/Decode

This is the normal operation mode of VS1001k. The SDI data is decoded. Decoded samples are converted to analog domain by the internal DAC, If there are errors in the decoding process, the error flags of SCI's HDAT0 and HDAT1 are set accordingly. In case there are serious errors in the input data, decoding is still continued, but the analog outputs are muted.

When there is no valid input for decoding, VS1001k goes into idle mode (lower power consumption than during decoding) and actively monitors the serial data input for valid data. The data input does not need to be clocked (DCLK) when no data is sent.

The software needs to be reset between MPEG audio stream files. See for the Chapter "Testing" to see how it is done.

8.6 Sanity Checks

Although VS1001k checks extensively for bad MP3 streams, it may happen that it encounters a bitstream that makes the firmware's recovery code fail. This may particularly happen during fast forward and fast backwards operations, where the data where the microcontroller lands the MP3 decoder may not be a valid header.

The microcontroller should keep a look at the data speeds VS1001k requires. If data input either stops completely (DREQ always inactive) for a whole second, or if VS1001k requires more than 60 KiB data in any single second, it is the responsibility of the microcontroller to either reset the software. If that doesn't help, a hardware reset should be issued.

8.7 PCM Mode

VS1001k can be used as a Digital-to-Analog converter (DAC) by feeding PCM data. A convenient way to use VS1001k as a DAC is to load *SDI PCM Extension for VS1001k* software from VLSI Solution's home page at *http://www.vlsi.fi/vs1001/software/*.

The SDI PCM Extension makes it possible for the user to use SDI to feed 8-bit or 16-bit PCM samples in mono or stereo at any sample rate upto 48 kHz (with nominal 24.576 MHz operating frequency).

8.8 Testing

There are several test modes in VS1001k, which allow the user to perform memory tests, SCI bus tests, and several different sine wave tests ranging from 250 Hz to 1500 Hz.

All tests are started in a similar way: VS1001 is hardware reset, and then a test command is sent to the SDI bus. Each test is started by sending a 4-byte special command sequence, followed by 4 zeros. The sequences are described below.

B.3. TM204AFF6

Pantalla de datos LCD ***Tianma TM204AFF6****. Tiene un tamaño de 4 líneas de 20 caracteres cada una. El color del display es amarillo sobre fondo azul. Funciona con una alimentación de 5 voltios*

1. General Specifications:

1.1 Display type: STN

1.2 Display color*:

Display color: Yellow-Green

Background: Blue*

1.3 Polarizer mode: Transmissive/Negative

1.4 Viewing Angle: 6:00

1.5 Driving Method: 1/16 Duty 1/5 Bias

1.6 Backlight: LED

1.7 Display Fonts: 5 x 7 dots (1 Character)＋5 x 1 dots (1 Cursor)

1.8 Data Transfer: 8 Bit Parallel

1.9 Operating Temperature: 0----+50℃

Storage Temperature: -20----+60℃

1.10 Outline Dimensions: Refer to outline drawing on next page

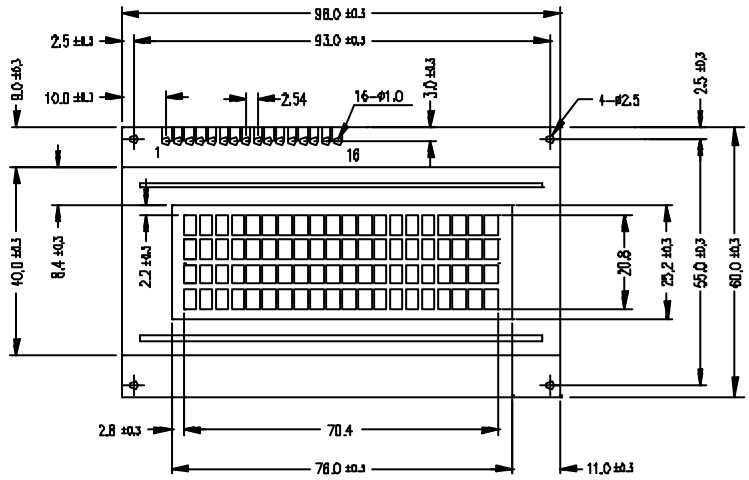
1.11 Dot Matrix: 20 Characters X 4 Lines

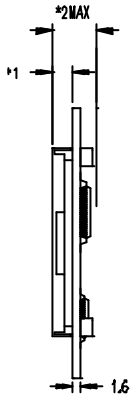
1.12 Dot Size: 0.55X0.55 (mm)

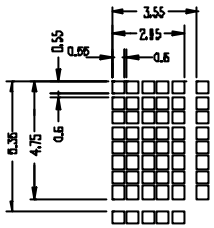
1.13 Dot Pitch: 0.60X0.60 (mm)

1.14 Weight: 75g

* Color tone is changed by backlight, temperature and driving voltage.



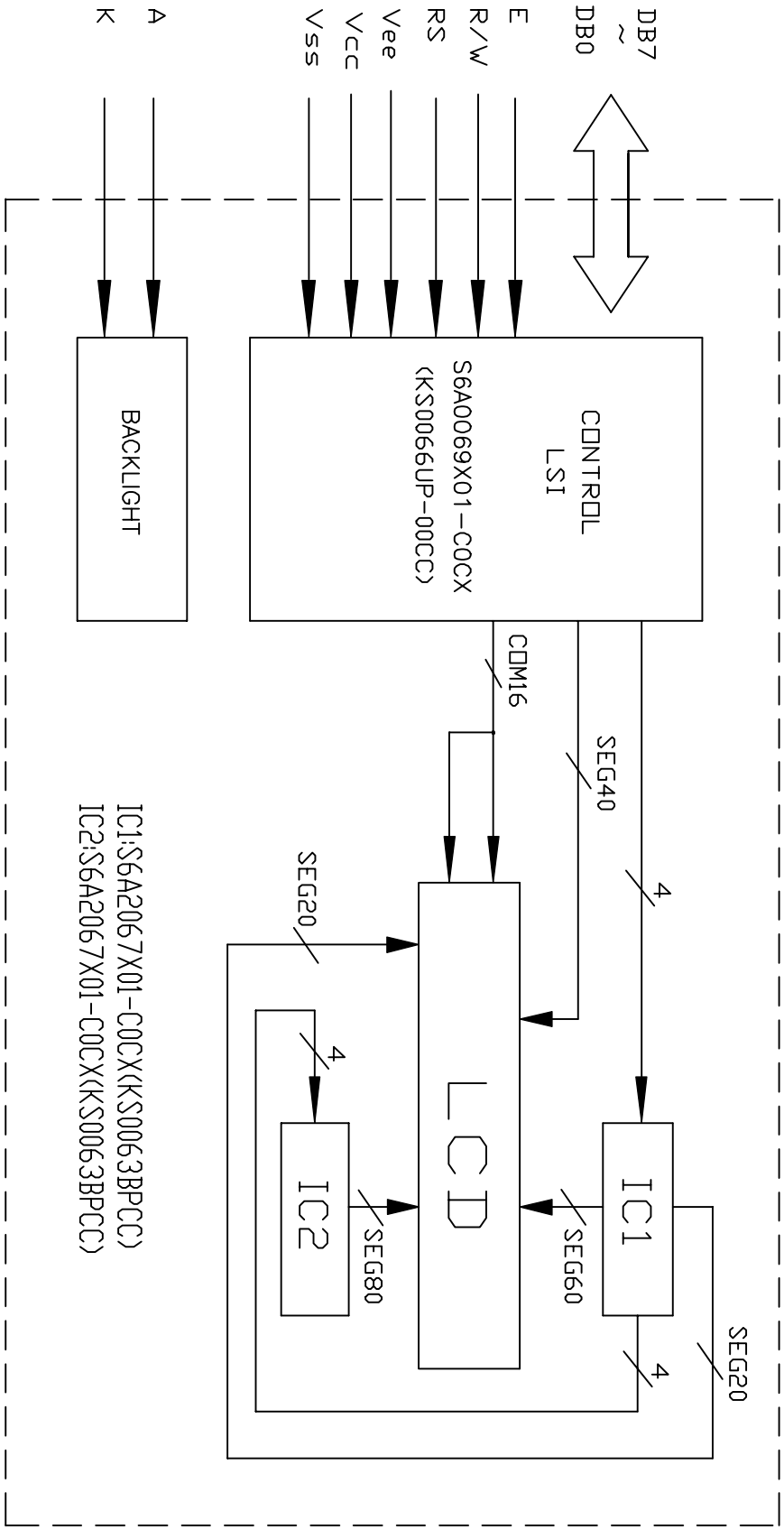




1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Vss	Vcc	Vee	RS	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	LED+	LED-

TYPE	*1	*2
R&EL	4.6	12.0
LED	9.5	15.0

4. Circuit Block Diagram



6.2 Interface Signals

Pin No.	Symbol	Level	Description
1	V_{SS}	0V	Ground
2	V_{CC}	5.0V	Power supply voltage for logic and LCD(+)
3	V_{EE}	0.3V	Power supply voltage for LCD(-)
4	RS	H/L	Selects registers (H: Data L: Instruction)
5	R/W	H/L	Selects read or write
6	E	H/L	Starts data read/write
7	DB0	H/L	Data bit0
8	DB1	H/L	Data bit1
9	DB2	H/L	Data bit2
10	DB3	H/L	Data bit3
11	DB4	H/L	Data bit4
12	DB5	H/L	Data bit5
13	DB6	H/L	Data bit6
14	DB7	H/L	Data bit7
15	LED+	4.2V	Power supply voltage for LED(+)
16	LED-	0V	Power supply voltage for LED(-)

6.4 Instruction Code

Instruction Table

Instruction	Instruction Code										Description	Execution time (fosc= 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write “20H” to DDRAM and set DDRAM address to “00H” from AC	1.53 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to “00H” from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display.	39 μs
Display ON/ OFF Control	0	0	0	0	0	0	1	D	C	B	Set display(D), cursor(C), and blinking of cursor(B) on/off control bit.	39 μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	39 μs
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-bit/4-bit), numbers of display line (N: 2-line/1-line) and, display font type (F:5×11dots/5×8 dots)	39 μs
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	39 μs
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter.	39 μs
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 μs
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43 μs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43 μs

* “-“: don’t care

NOTE: When an MPU program with checking the Busy Flag(DB7) is made, it must be necessary 1/2Fosc is necessary for executing the next instruction by the falling edge of the 'E' signal after the Busy Flag (DB7) goes to “Low”.

6.5 Character generator ROM(KS0066U-00)

Upper 4bit / Lower 4bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	(2)															
LLHL	(3)															
LLHH	(4)															
LHLL	(5)															
LHLH	(6)															
LHHL	(7)															
LHHH	(8)															
HLLL	(1)															
HLLH	(2)															
HLHL	(3)															
HLHH	(4)															
HHLL	(5)															
HHLH	(6)															
HHHL	(7)															
HHHH	(8)															

B.4. MC74HC04

Circuito lógico que contiene ***Seis Inversores Triestatado****. Sus características de entrada salida y su patillaje son compatibles con su versión de la familia 74LS. Tiene salidas estándar CMOS y funciona con una tensión de alimentación de entre 2 y 6 voltios.*

MC74HC04A

Hex Inverter

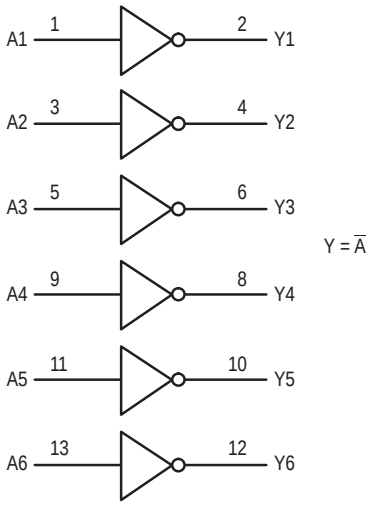
High–Performance Silicon–Gate CMOS

The MC74HC04A is identical in pinout to the LS04 and the MC14069. The device inputs are compatible with Standard CMOS outputs; with pullup resistors, they are compatible with LSTTL outputs.

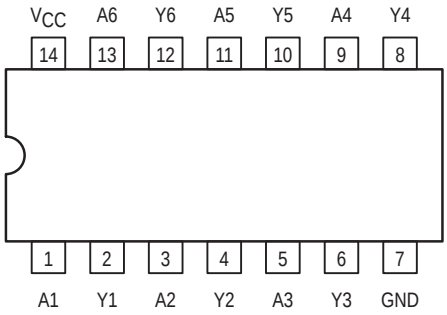
The device consists of six three–stage inverters.

- Output Drive Capability: 10 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS and TTL
- Operating Voltage Range: 2 to 6V
- Low Input Current: 1µA
- High Noise Immunity Characteristic of CMOS Devices
- In Compliance With the JEDEC Standard No. 7A Requirements
- Chip Complexity: 36 FETs or 9 Equivalent Gates

LOGIC DIAGRAM



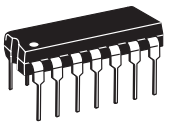
Pinout: 14–Lead Packages (Top View)



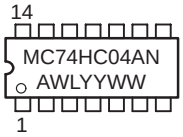
ON Semiconductor

http://onsemi.com

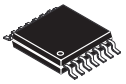
MARKING DIAGRAMS



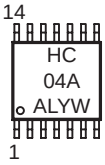
PDIP–14
N SUFFIX
CASE 646



SOIC–14
D SUFFIX
CASE 751A



TSSOP–14
DT SUFFIX
CASE 948G



A = Assembly Location
WL or L = Wafer Lot
YY or Y = Year
WW or W = Work Week

FUNCTION TABLE

Inputs	Outputs
A	**Y**
L	H
H	L

ORDERING INFORMATION

Device	Package	Shipping
MC74HC04AN	PDIP–14	2000 / Box
MC74HC04AD	SOIC–14	55 / Rail
MC74HC04ADR2	SOIC–14	2500 / Reel
MC74HC04ADT	TSSOP–14	96 / Rail
MC74HC04ADTR2	TSSOP–14	2500 / Reel

March, 2000 – Rev. 8

Publication Order Number:
MC74HC04A/D

B.5. MM74HCT08

Circuito lógico que contiene ***Cuatro Puertas AND Dobles****. Sus características de entrada salida y su patillaje son compatibles con su versión de la familia 74LS.*

FAIRCHILD SEMICONDUCTOR™

December 1983
Revised February 1999

MM74HCT08
Quad 2-Input AND Gate

General Description

The MM74HCT08 is a logic function fabricated by using advanced silicon-gate CMOS technology which provides the inherent benefits of CMOS—low quiescent power and wide power supply range. This device is input and output characteristic and pinout compatible with standard 74LS logic families. All inputs are protected from static discharge damage by internal diodes to V_{CC} and ground.

MM74HCT devices are intended to interface between TTL and NMOS components and standard CMOS devices. These parts are also plug-in replacements for LS-TTL devices and can be used to reduce power consumption in existing designs.

Features

- TTL, LS pin-out and threshold compatible
- Fast switching: t_{PLH}, t_{PHL} = 12 ns (typ)
- Low power: 10 μW at DC
- High fan-out, 10 LS-TTL loads

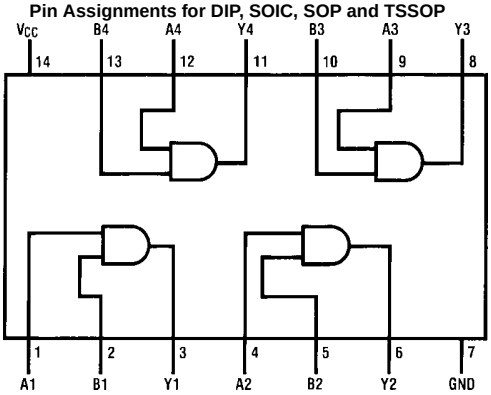
Ordering Code:

Order Number	Package Number	Package Description
MM74HCT08M	M14A	14-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-120, 0.150" Narrow
MM74HCT08SJ	M14D	14-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
MM74HCT08MTC	MTC14	14-Lead Thin Shrink Small Outline Package (TSSOP), JEDEC MO-153, 4.4mm Wide
MM74HCT08N	N14A	14-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide

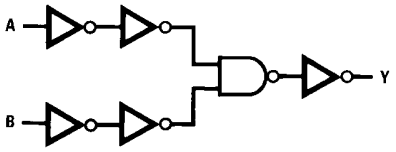
Devices also available in Tape and Reel. Specify by appending the suffix letter "X" to the ordering code.

Connection Diagram

Pin Assignments for DIP, SOIC, SOP and TSSOP



Logic Diagram



Absolute Maximum Ratings (Note 1)

(Note 2)

Supply Voltage (V_{CC})	–0.5 to +7.0V
DC Input Voltage (V_{IN})	–1.5 to V_{CC} +1.5V
DC Output Voltage (V_{OUT})	–0.5 to V_{CC} +0.5V
Clamp Diode Current (I_{IK}, I_{OK})	±20 mA
DC Output Current, per pin (I_{OUT})	±25 mA
DC V_{CC} or GND Current, per pin (I_{CC})	±50 mA
Storage Temperature Range (T_{STG})	–65°C to +150°C
Power Dissipation (P_D)	
(Note 3)	600 mW
S.O. Package only	500 mW
Lead Temperature (T_L)	
(Soldering 10 seconds)	260°C

Recommended Operating Conditions

	Min	Max	Units
Supply Voltage (V_{CC})	4.5	5.5	V
DC Input or Output Voltage			
(V_{IN}, V_{OUT})	0	V_{CC}	V
Operating Temperature Range (T_A)	–40	+85	°C
Input Rise or Fall Times			
(t_r, t_f)		500	ns

Note 1: Absolute Maximum Ratings are those values beyond which damage to the device may occur.

Note 2: Unless otherwise specified all voltages are referenced to ground.

Note 3: Power Dissipation temperature derating — plastic "N" package –12 mW/°C from 65°C to 85°C.

DC Electrical Characteristics

V_{CC} = 5V ± 10% (unless otherwise specified)

Symbol	Parameter	Conditions	T_A = 25°C		T_A = –40 to 85°C	T_A = –55 to 125°C	Units
			Typ	Guaranteed Limits			
V_{IH}	Minimum HIGH Level Input Voltage			2.0	2.0	2.0	V
V_{IL}	Maximum LOW Level Input Voltage			0.8	0.8	0.8	V
V_{OH}	Minimum HIGH Level Output Voltage	$V_{IN} = V_{IH}$ or V_{IL}					
		$\|I_{OUT}\|$ = 20 µA	V_{CC}	V_{CC}– 0.1	V_{CC}– 0.1	V_{CC}– 0.1	V
		$\|I_{OUT}\|$ = 4.0 mA, V_{CC} = 4.5V	4.2	3.98	3.84	3.7	V
		$\|I_{OUT}\|$ = 4.8 mA, V_{CC} = 5.5V	5.2	4.98	4.84	4.7	V
V_{OL}	Maximum LOW Level Voltage	$V_{IN} = V_{IH}$					
		$\|I_{OUT}\|$ = 20 µA	0	0.1	0.1	0.1	V
		$\|I_{OUT}\|$ = 4.0 mA, V_{CC} = 4.5V	0.2	0.26	0.33	0.4	V
		$\|I_{OUT}\|$ = 4.8 mA, V_{CC} = 5.5V	0.2	0.26	0.33	0.4	V
I_{IN}	Maximum Input Current	$V_{IN} = V_{CC}$ or GND, V_{IH} or V_{IL}		±0.1	±1.0	±1.0	µA
I_{CC}	Maximum Quiescent Supply Current	$V_{IN} = V_{CC}$ or GND		2.0	20	40	µA
		I_{OUT} = 0 µA					
		V_{IN} = 2.4V or 0.5V (Note 4)		1.2	1.4	1.5	mA

Note 4: This is measured per input with all other inputs held at V_{CC} or ground.

B.6. MC74HCT138

Circuito lógico que contiene un ***Decodificador-Demultiplexor de Ocho Salidas****. Sus características de entrada salida y su patillaje son compatibles con su versión de la familia 74LS. Funciona con una tensión de alimentación de entre 2 y 6 voltios.*

1-of-8 Decoder/Demultiplexer

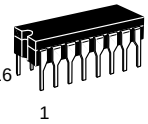
High–Performance Silicon–Gate CMOS

MC54/74HC138A

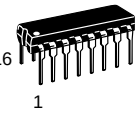
The MC54/74HC138A is identical in pinout to the LS138. The device inputs are compatible with standard CMOS outputs; with pullup resistors, they are compatible with LSTTL outputs.

The HC138A decodes a three–bit Address to one–of–eight active–low outputs. This device features three Chip Select inputs, two active–low and one active–high to facilitate the demultiplexing, cascading, and chip–selecting functions. The demultiplexing function is accomplished by using the Address inputs to select the desired device output; one of the Chip Selects is used as a data input while the other Chip Selects are held in their active states.

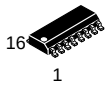
- Output Drive Capability: 10 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS and TTL
- Operating Voltage Range: 2.0 to 6.0 V
- Low Input Current: 1.0 μA
- High Noise Immunity Characteristic of CMOS Devices
- In Compliance with the Requirements Defined by JEDEC Standard No. 7A
- Chip Complexity: 100 FETs or 29 Equivalent Gates



J SUFFIX
CERAMIC PACKAGE
CASE 620–10



N SUFFIX
PLASTIC PACKAGE
CASE 648–08



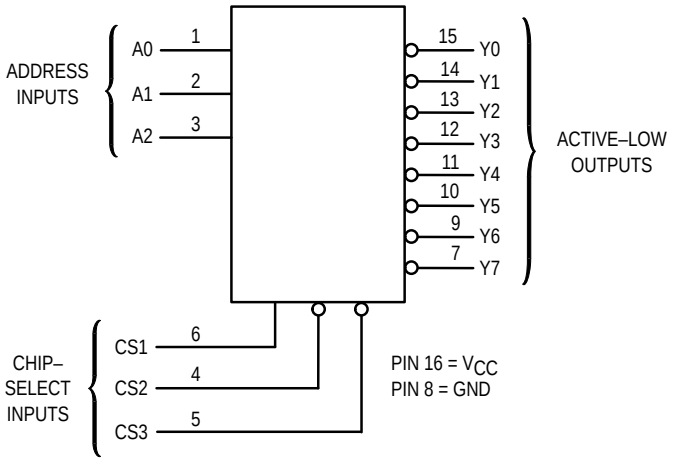
D SUFFIX
SOIC PACKAGE
CASE 751B–05

DT SUFFIX
TSSOP PACKAGE
CASE 948F–01

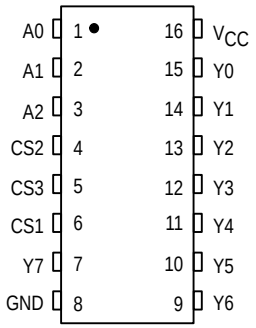
ORDERING INFORMATION

MC54HCXXXAJ	Ceramic
MC74HCXXXAN	Plastic
MC74HCXXXAD	SOIC
MC74HCXXXADT	TSSOP

LOGIC DIAGRAM



PIN ASSIGNMENT



FUNCTION TABLE

Inputs						Outputs							
CS1	CS2	CS3	A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	X	H	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

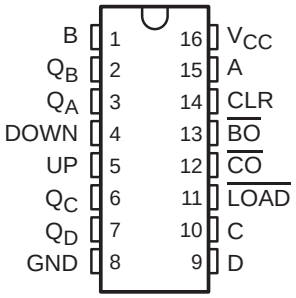
H = high level (steady state); L = low level (steady state);
X = don't care

B.7. SN74HCT193

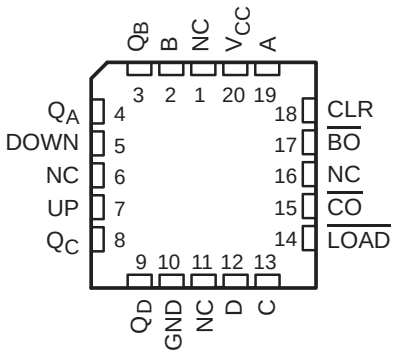
Circuito lógico que contiene un ***Contador Síncrono Up-Down de Cuatro Bits****. Sus características de entrada salida y su patillaje son compatibles con su versión de la familia 74LS. Funciona con una tensión de alimentación de entre 2 y 6 voltios.*

- **Wide Operating Voltage Range of 2 V to 6 V**
- **Outputs Can Drive Up To 10 LSTTL Loads**
- **Low Power Consumption, 80-μA Max I_{CC}**
- **Typical t_{pd} = 20 ns**
- **±4-mA Output Drive at 5 V**
- **Low Input Current of 1 μA Max**
- **Look-Ahead Circuitry Enhances Cascaded Counters**
- **Fully Synchronous in Count Modes**
- **Parallel Asynchronous Load for Modulo-N Count Lengths**
- **Asynchronous Clear**

SN54HC193 . . . J OR W PACKAGE
SN74HC193 . . . D, N, NS, OR PW PACKAGE
(TOP VIEW)



SN54HC193 . . . FK PACKAGE
(TOP VIEW)



NC − No internal connection

description/ordering information

The ’HC193 devices are 4-bit synchronous, reversible, up/down binary counters. Synchronous operation is provided by having all flip-flops clocked simultaneously so that the outputs change coincidentally with each other when so instructed by the steering logic. This mode of operation eliminates the output counting spikes normally associated with asynchronous (ripple-clock) counters.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
−40°C to 85°C	PDIP − N	Tube of 25	SN74HC193N	SN74HC193N
	SOIC − D	Tube of 40	SN74HC193D	HC193
		Reel of 2500	SN74HC193DR	
		Reel of 250	SN74HC193DT	
	SOP − NS	Reel of 2000	SN74HC193NSR	HC193
	TSSOP − PW	Tube of 90	SN74HC193PW	HC193
		Reel of 2000	SN74HC193PWR	
		Reel of 250	SN74HC193PWT	
−55°C to 125°C	CDIP − J	Tube of 25	SNJ54HC193J	SNJ54HC193J
	CFP − W	Tube of 150	SNJ54HC193W	SNJ54HC193W
	LCCC − FK	Tube of 55	SNJ54HC193FK	SNJ54HC193FK

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

description/ordering information (continued)

The outputs of the four flip-flops are triggered on a low-to-high-level transition of either count (clock) input (UP or DOWN). The direction of counting is determined by which count input is pulsed while the other count input is high.

All four counters are fully programmable; that is, each output may be preset to either level by placing a low on the load ($\overline{LOAD}$) input and entering the desired data at the data inputs. The output changes to agree with the data inputs independently of the count pulses. This feature allows the counters to be used as modulo-N dividers simply by modifying the count length with the preset inputs.

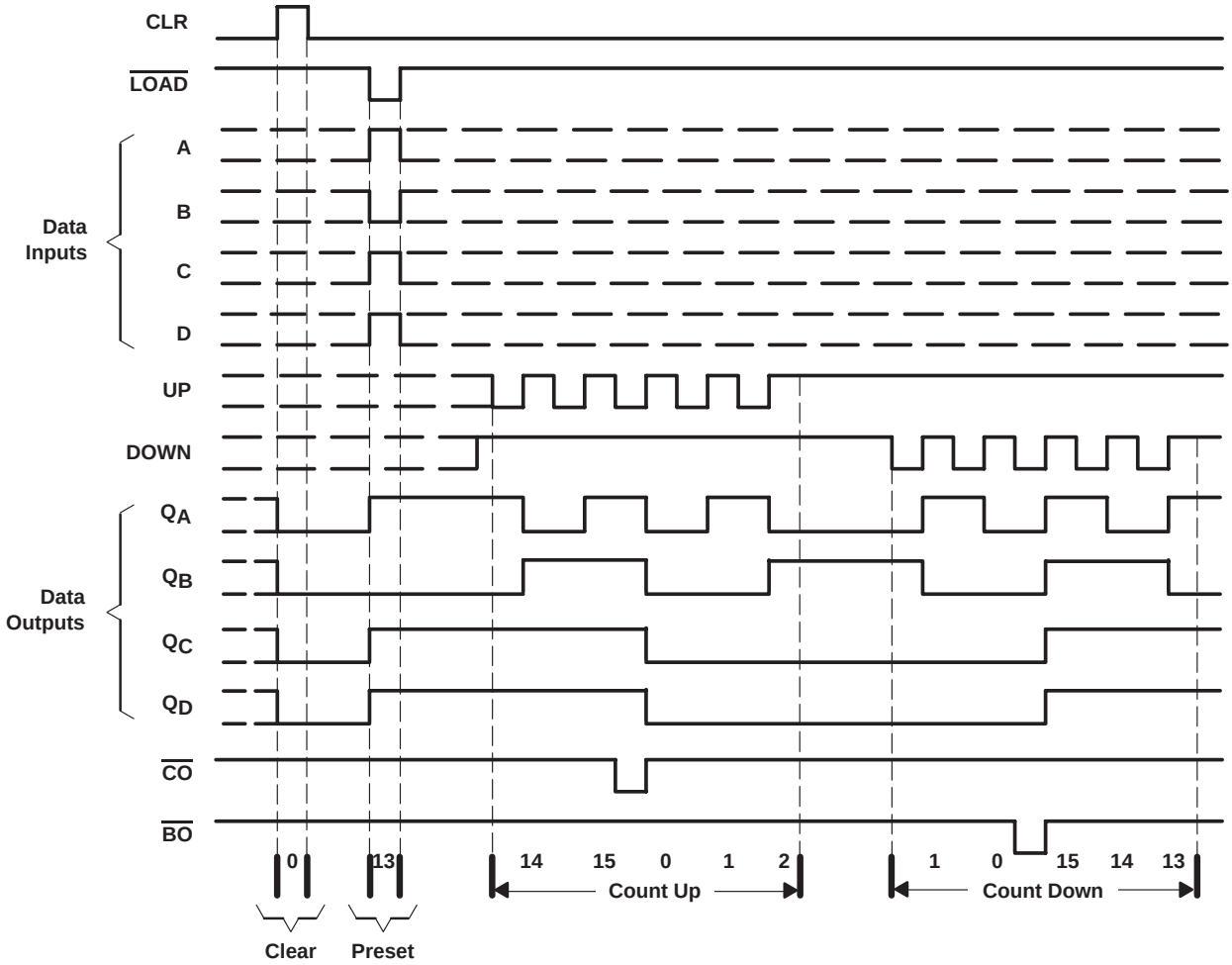
A clear (CLR) input has been provided that forces all outputs to the low level when a high level is applied. The clear function is independent of the count and $\overline{LOAD}$ inputs.

These counters were designed to be cascaded without the need for external circuitry. The borrow ($\overline{BO}$) output produces a low-level pulse while the count is zero (all outputs low) and DOWN is low. Similarly, the carry ($\overline{CO}$) output produces a low-level pulse while the count is maximum (9 or 15), and UP is low. The counters then can be cascaded easily by feeding $\overline{BO}$ and $\overline{CO}$ to DOWN and UP, respectively, of the succeeding counter.

typical clear, load, and count sequence

The following sequence is illustrated below:

1. Clear outputs to 0
2. Load (preset) to binary 13
3. Count up to 14, 15, carry, 0, 1, and 2
4. Count down to 1, 0, borrow, 15, 14, and 13



NOTES: A. CLR overrides $\overline{\text{LOAD}}$, data, and count inputs.
B. When counting up, count-down input must be high; when counting down, count-up input must be high.

B.8. MC74HC245

Circuito lógico que contiene un ***Transceptor de BUS, No Inversor, Triestado de Ocho Bits****. Sus características de entrada salida y su patillaje son compatibles con su versión de la familia 74LS. Funciona con una tensión de alimentación de entre 2 y 6 voltios.*

MC54/74HC245A

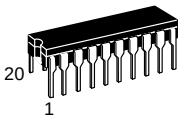
Octal 3-State Noninverting Bus Transceiver

High–Performance Silicon–Gate CMOS

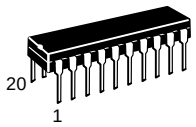
The MC54/74HC245A is identical in pinout to the LS245. The device inputs are compatible with standard CMOS outputs; with pullup resistors, they are compatible with LSTTL outputs.

The HC245A is a 3–state noninverting transceiver that is used for 2–way asynchronous communication between data buses. The device has an active–low Output Enable pin, which is used to place the I/O ports into high–impedance states. The Direction control determines whether data flows from A to B or from B to A.

- Output Drive Capability: 15 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS, and TTL
- Operating Voltage Range: 2 to 6 V
- Low Input Current: 1 μA
- High Noise Immunity Characteristic of CMOS Devices
- In Compliance with the Requirements Defined by JEDEC Standard No. 7A
- Chip Complexity: 308 FETs or 77 Equivalent Gates



J SUFFIX
CERAMIC PACKAGE
CASE 732–03



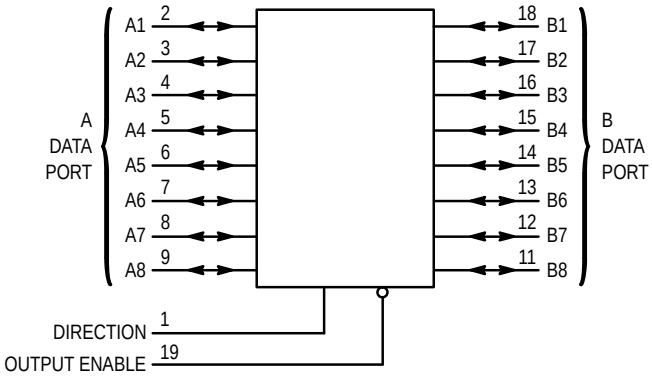
N SUFFIX
PLASTIC PACKAGE
CASE 738–03

DW SUFFIX
SOIC PACKAGE
CASE 751D–04

ORDERING INFORMATION

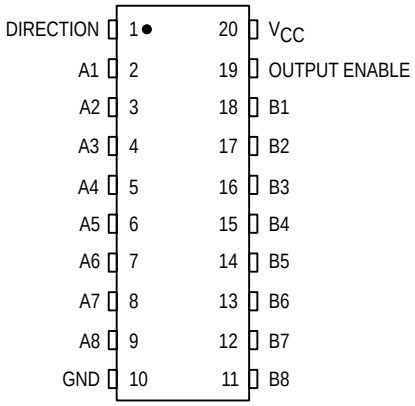
MC54HCXXXAJ	Ceramic
MC74HCXXXAN	Plastic
MC74HCXXXADW	SOIC

LOGIC DIAGRAM



PIN 10 = GND
PIN 20 = V_{CC}

PIN ASSIGNMENT



FUNCTION TABLE

Control Inputs		Operation
Output Enable	Direction	
L	L	Data Transmitted from Bus B to Bus A
L	H	Data Transmitted from Bus A to Bus B
H	X	Buses Isolated (High–Impedance State)

X = don't care

B.9. MC74HC573

Circuito lógico que contiene un ***Latch Transparente, No Inversor, Triestado de Ocho Bits****. Sus características de entrada salida y su patillaje son compatibles con su versión de la familia 74LS. Funciona con una tensión de alimentación de entre 2 y 6 voltios.*

MC74HC573A

Octal 3-State Noninverting Transparent Latch

High–Performance Silicon–Gate CMOS

The MC74HC573A is identical in pinout to the LS573. The devices are compatible with standard CMOS outputs; with pullup resistors, they are compatible with LSTTL outputs.

These latches appear transparent to data (i.e., the outputs change asynchronously) when Latch Enable is high. When Latch Enable goes low, data meeting the setup and hold time becomes latched.

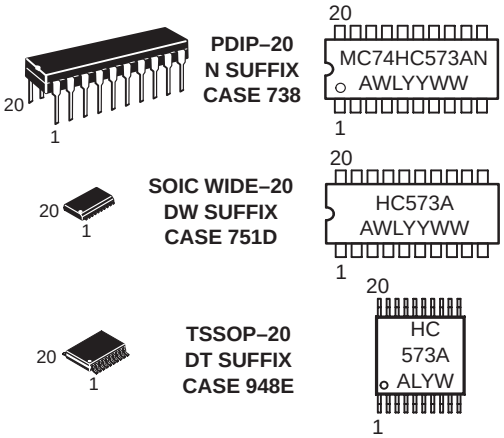
The HC573A is identical in function to the HC373A but has the data inputs on the opposite side of the package from the outputs to facilitate PC board layout.

- Output Drive Capability: 15 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS and TTL
- Operating Voltage Range: 2.0 to 6.0 V
- Low Input Current: 1.0 μA
- In Compliance with the Requirements Defined by JEDEC Standard No. 7A
- Chip Complexity: 218 FETs or 54.5 Equivalent Gates

ON Semiconductor

http://onsemi.com

MARKING DIAGRAMS



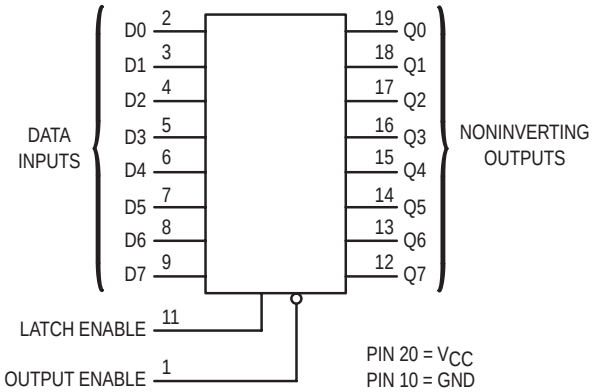
A = Assembly Location
WL = Wafer Lot
YY = Year
WW = Work Week

ORDERING INFORMATION

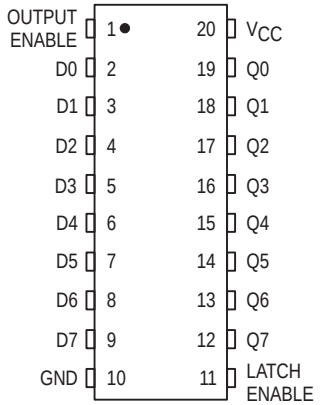
Device	Package	Shipping
MC74HC573AN	PDIP–20	1440 / Box
MC74HC573ADW	SOIC–WIDE	38 / Rail
MC74HC573ADWR2	SOIC–WIDE	1000 / Reel
MC74HC573ADT	TSSOP–20	75 / Rail
MC74HC573ADTR2	TSSOP–20	2500 / Reel

May, 2000 – Rev. 9
Publication Order Number:
MC74HC573A/D

LOGIC DIAGRAM



PIN ASSIGNMENT



FUNCTION TABLE

Inputs			Output
Output Enable	**Latch Enable**	**D**	**Q**
L	H	H	H
L	H	L	L
L	L	X	No Change
H	X	X	Z

X = Don’t Care
Z = High Impedance

Design Criteria	Value	Units
Internal Gate Count*	54.5	ea.
Internal Gate Propagation Delay	1.5	ns
Internal Gate Power Dissipation	5.0	μW
Speed Power Product	0.0075	pJ

*Equivalent to a two–input NAND gate.

B.10. 74LVC4245

Circuito lógico que contiene un ***Transceptor, Conversor de Voltages, Triestado de Ocho Bits****.*

FEATURES

- 5 V tolerant inputs/outputs for interfacing with 5 V logic
- Wide supply voltage range:
  - 3 V port (V_{CCB}): 1.5 V to 3.6 V
  - 5 V port (V_{CCA}): 1.5 V to 5.5 V.
- CMOS low power consumption
- Direct interface with TTL levels
- Inputs accept voltages up to 5.5 V
- High-impedance when V_{CC} = 0 V
- Complies with JEDEC standard no. JESD8B/JESD36
- ESD protection:
  - HBM EIA/JESD22-A114-B exceeds 2000 V
  - MM EIA/JESD22-A115-A exceeds 200 V.
- Specified from –40 °C to +85 °C and –40 °C to +125 °C.

DESCRIPTION

The 74LVC4245A is a high-performance, low-power, low-voltage, Si-gate CMOS device, superior to most advanced CMOS compatible TTL families.

The 74LVC4245A is an octal dual supply translating transceiver featuring non-inverting 3-state bus compatible outputs in both send and receive directions. It is designed to interface between a 3 V and 5 V bus in a mixed 3 V and 5 V supply environment.

The 74LVC4245A features an output enable input (pin $\overline{OE}$) for easy cascading and a send/receive input (pin DIR) for direction control. Pin $\overline{OE}$ controls the outputs so that the buses are effectively isolated.

In suspend mode, when V_{CCA} is zero, there will be no current flow from one supply to the other supply. The A-outputs must be set 3-state and the voltage on the A-bus must be smaller than V_{diode} (typical 0.7 V). $V_{CCA} \geq V_{CCB}$ (except in suspend mode).

QUICK REFERENCE DATA

GND = 0 V; T_{amb} = 25 °C; $t_r = t_f \leq 2.5$ ns.

SYMBOL	PARAMETER	CONDITIONS	TYPICAL	UNIT
t_{PHL}	propagation delay An to Bn	C_L = 50 pF; V_{CCA} = 5.0 V; V_{CCB} = 3.3 V	3.3	ns
	propagation delay Bn to An	C_L = 50 pF; V_{CCA} = 5.0 V; V_{CCB} = 3.3 V	3.4	ns
t_{PLH}	propagation delay An to Bn	C_L = 50 pF; V_{CCA} = 5.0 V; V_{CCB} = 3.3 V	2.8	ns
	propagation delay Bn to An	C_L = 50 pF; V_{CCA} = 5.0 V; V_{CCB} = 3.3 V	3.0	ns
C_I	input capacitance		4.0	pF
$C_{I/O}$	input/output capacitance An and Bn		5.0	pF
C_{PD}	5 V port: power dissipation capacitance Bn to An	V_{CC} = 5.0 V; notes 1 and 2		
		outputs enabled	17	pF
		outputs disabled	5	pF
	3 V port: power dissipation capacitance An to Bn	V_{CC} = 3.3 V; notes 1 and 2		
		outputs enabled	17	pF
		outputs disabled	5	pF

Notes

1. C_{PD} is used to determine the dynamic power dissipation (P_D in μW).

   $P_D = C_{PD} \times V_{CC}^2 \times f_i \times N + \Sigma(C_L \times V_{CC}^2 \times f_o)$ where:

   f_i = input frequency in MHz;

   f_o = output frequency in MHz;

   C_L = output load capacitance in pF;

   V_{CC} = supply voltage in Volts;

   N = total load switching outputs;

   $\Sigma(C_L \times V_{CC}^2 \times f_o)$ = sum of the outputs.
2. The condition is V_I = GND to V_{CC}.

FUNCTION TABLE

See note 1.

INPUT		INPUT/OUTPUT	
$\overline{OE}$	DIR	An	Bn
L	L	A = B	input
L	H	input	B = A
H	X	Z	Z

Note

1. H = HIGH voltage level;
   L = LOW voltage level;
   X = don't care;
   Z = high-impedance OFF-state.

ORDERING INFORMATION

TYPE NUMBER	TEMPERATURE RANGE	PACKAGE			
		PINS	PACKAGE	MATERIAL	CODE
74LVC4245AD	–40 °C to +125 °C	24	SO24	plastic	SOT137-1
74LVC4245ADB	–40 °C to +125 °C	24	SSOP24	plastic	SOT340-1
74LVC4245APW	–40 °C to +125 °C	24	TSSOP24	plastic	SOT355-1

PINNING

PIN	SYMBOL	DESCRIPTION
1	V_{CCA}	supply voltage (5 V bus)
2	DIR	direction control
3	A0	data input or output
4	A1	data input or output
5	A2	data input or output
6	A3	data input or output
7	A4	data input or output
8	A5	data input or output
9	A6	data input or output
10	A7	data input or output
11	GND	ground (0 V)
12	GND	ground (0 V)
13	GND	ground (0 V)
14	B7	data output or input
15	B6	data output or input
16	B5	data output or input
17	B4	data output or input
18	B3	data output or input
19	B2	data output or input
20	B1	data output or input
21	B0	data output or input
22	$\overline{OE}$	output enable input (active LOW)
23	V_{CCB}	supply voltage (3 V bus)
24	V_{CCB}	supply voltage (3 V bus)

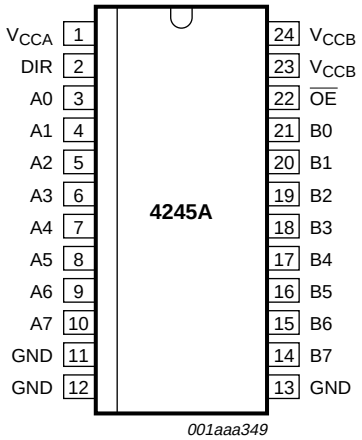


Fig.1 Pin configuration.

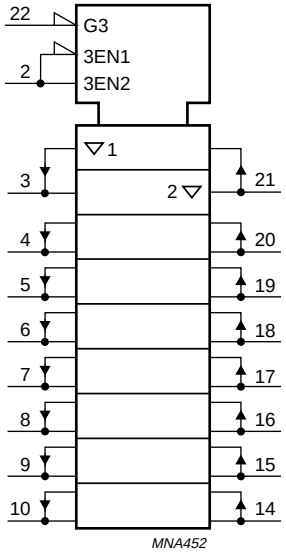


Fig.2 IEC logic symbol.

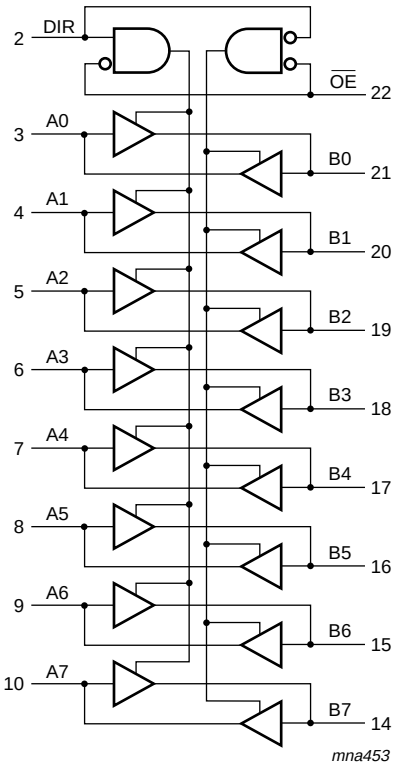


Fig.3 Logic diagram.

B.11. CY62256

Circuito que contiene una ***RAM Estática de 32K x 8 bits****. Sus entradas y salidas son compatibles TTL.*

CY62256

32Kx8 Static RAM

Features

- **4.5V–5.5V Operation**
- **Low active power (70 ns, LL version)**
  — 275 mW (max.)
- **Low standby power (70 ns, LL version)**
  — 28 μW (max.)
- **55, 70 ns access time**
- **Easy memory expansion with CE and OE features**
- **TTL-compatible inputs and outputs**
- **Automatic power-down when deselected**
- **CMOS for optimum speed/power**

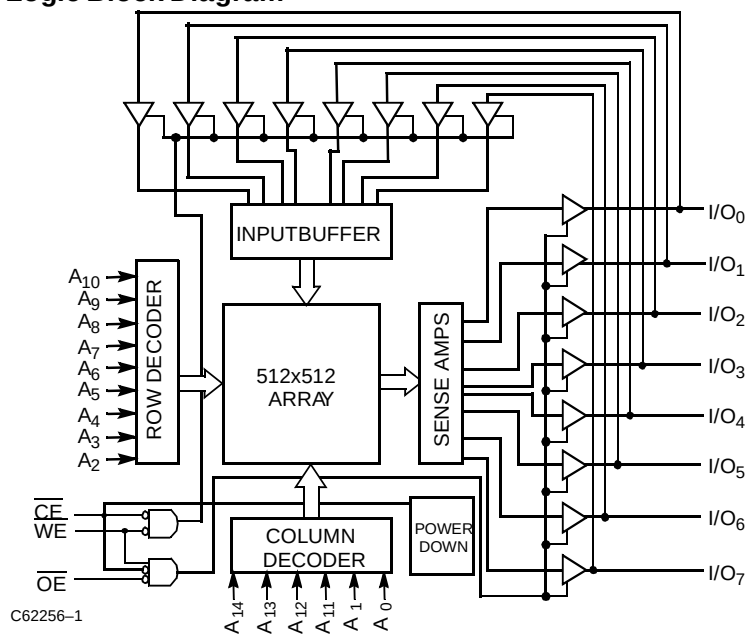
Functional Description

The CY62256 is a high-performance CMOS static RAM organized as 32,768 words by 8 bits. Easy memory expansion is provided by an active LOW chip enable ($\overline{CE}$) and active LOW output enable ($\overline{OE}$) and three-state drivers. This device has an automatic power-down feature, reducing the power consumption by 99.9% when deselected. The CY62256 is in the standard 450-mil-wide (300-mil body width) SOIC, TSOP, and 600-mil PDIP packages.

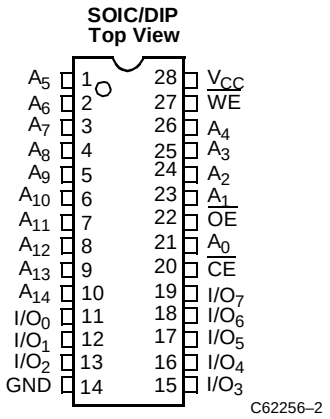
An active LOW write enable signal ($\overline{WE}$) controls the writing/reading operation of the memory. When $\overline{CE}$ and $\overline{WE}$ inputs are both LOW, data on the eight data input/output pins (I/O_0 through I/O_7) is written into the memory location addressed by the address present on the address pins (A_0 through A_{14}). Reading the device is accomplished by selecting the device and enabling the outputs, $\overline{CE}$ and $\overline{OE}$ active LOW, while $\overline{WE}$ remains inactive or HIGH. Under these conditions, the contents of the location addressed by the information on address pins are present on the eight data input/output pins.

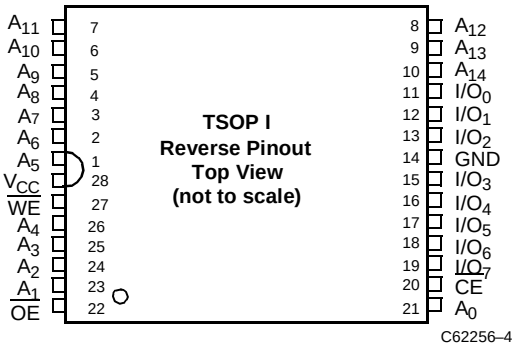
The input/output pins remain in a high-impedance state unless the chip is selected, outputs are enabled, and write enable ($\overline{WE}$) is HIGH.

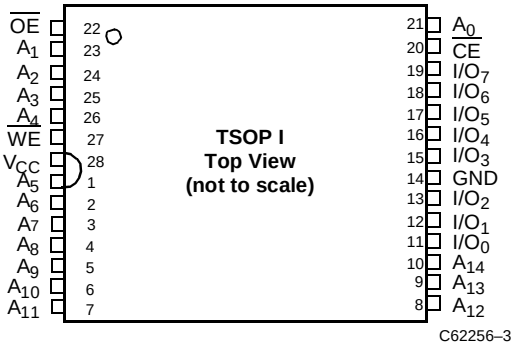
Logic Block Diagram



Pin Configurations





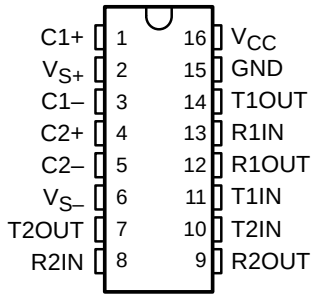


B.12. MAX232

Circuito que contiene un ***Transmisor/Receptor Doble Compatible con la norma EIA-232****. Convierte niveles de tension TTL en niveles EIA-232. Su tensión de alimentación es de tan solo 5 voltios.*

- **Meet or Exceed TIA/EIA-232-F and ITU Recommendation V.28**
- **Operate With Single 5-V Power Supply**
- **Operate Up to 120 kbit/s**
- **Two Drivers and Two Receivers**
- **±30-V Input Levels**
- **Low Supply Current . . . 8 mA Typical**
- **Designed to be Interchangeable With Maxim MAX232**
- **ESD Protection Exceeds JESD 22**
  - **2000-V Human-Body Model (A114-A)**
- **Applications**
  - **TIA/EIA-232-F**
  - **Battery-Powered Systems**
  - **Terminals**
  - **Modems**
  - **Computers**

MAX232 . . . D, DW, N, OR NS PACKAGE
MAX232I . . . D, DW, OR N PACKAGE
(TOP VIEW)



description/ordering information

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply EIA-232 voltage levels from a single 5-V supply. Each receiver converts EIA-232 inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V and a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into EIA-232 levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC™ library.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	PDIP (N)	Tube	MAX232N	MAX232N
	SOIC (D)	Tube	MAX232D	MAX232
		Tape and reel	MAX232DR	
	SOIC (DW)	Tube	MAX232DW	MAX232
		Tape and reel	MAX232DWR	
	SOP (NS)	Tape and reel	MAX232NSR	MAX232
–40°C to 85°C	PDIP (N)	Tube	MAX232IN	MAX232IN
	SOIC (D)	Tube	MAX232ID	MAX232I
		Tape and reel	MAX232IDR	
	SOIC (DW)	Tube	MAX232IDW	MAX232I
		Tape and reel	MAX232IDWR	

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

Function Tables

EACH DRIVER

INPUT TIN	OUTPUT TOUT
L	H
H	L

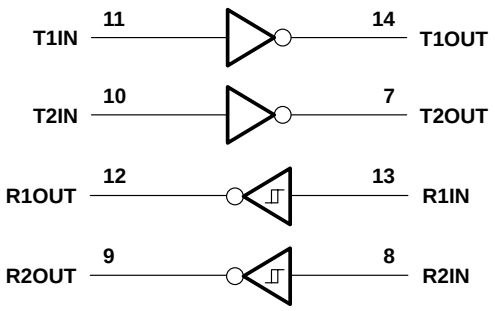
H = high level, L = low level

EACH RECEIVER

INPUT RIN	OUTPUT ROUT
L	H
H	L

H = high level, L = low level

logic diagram (positive logic)



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Input supply voltage range, V_{CC} (see Note 1) . . . . . –0.3 V to 6 V
Positive output supply voltage range, V_{S+} . . . . . V_{CC} – 0.3 V to 15 V
Negative output supply voltage range, V_{S-} . . . . . –0.3 V to –15 V
Input voltage range, V_I: Driver . . . . . –0.3 V to V_{CC} + 0.3 V
Receiver . . . . . ±30 V
Output voltage range, V_O: T1OUT, T2OUT . . . . . V_{S-} – 0.3 V to V_{S+} + 0.3 V
R1OUT, R2OUT . . . . . –0.3 V to V_{CC} + 0.3 V
Short-circuit duration: T1OUT, T2OUT . . . . . Unlimited
Package thermal impedance, θ_{JA} (see Note 2): D package . . . . . 73°C/W
DW package . . . . . 57°C/W
N package . . . . . 67°C/W
NS package . . . . . 64°C/W
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds . . . . . 260°C
Storage temperature range, T_{stg} . . . . . –65°C to 150°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
NOTE 1: All voltage values are with respect to network ground terminal.
2. The package thermal impedance is calculated in accordance with JESD 51-7.

recommended operating conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	V
V_{IH}	High-level input voltage (T1IN,T2IN)		2			V
V_{IL}	Low-level input voltage (T1IN, T2IN)				0.8	V
R1IN, R2IN	Receiver input voltage				±30	V
T_A	Operating free-air temperature	MAX232	0		70	°C
		MAX232I	–40		85	

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see Note 3 and Figure 4)

PARAMETER		TEST CONDITIONS		MIN	TYP‡	MAX	UNIT
I_{CC}	Supply current	V_{CC} = 5.5 V, T_A = 25°C	All outputs open,		8	10	mA

‡ All typical values are at V_{CC} = 5 V and T_A = 25°C.
NOTE 3: Test conditions are C1–C4 = 1 μF at V_{CC} = 5 V ± 0.5 V.

DRIVER SECTION

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 3)

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	T1OUT, T2OUT	R_L = 3 kΩ to GND	5	7		V
V_{OL}	Low-level output voltage‡	T1OUT, T2OUT	R_L = 3 kΩ to GND		–7	–5	V
r_o	Output resistance	T1OUT, T2OUT	$V_{S+} = V_{S-} = 0$, $V_O = \pm 2$ V	300			Ω
I_{OS}§	Short-circuit output current	T1OUT, T2OUT	V_{CC} = 5.5 V, V_O = 0		±10		mA
I_{IS}	Short-circuit input current	T1IN, T2IN	V_I = 0			200	µA

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.
§ Not more than one output should be shorted at a time.
NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 3)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
SR	Driver slew rate	R_L = 3 kΩ to 7 kΩ, See Figure 2			30	V/µs
SR(t)	Driver transition region slew rate	See Figure 3		3		V/µs
	Data rate	One TOUT switching		120		kbit/s

NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

RECEIVER SECTION

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 3)

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	R1OUT, R2OUT	I_{OH} = –1 mA	3.5			V
V_{OL}	Low-level output voltage‡	R1OUT, R2OUT	I_{OL} = 3.2 mA			0.4	V
V_{IT+}	Receiver positive-going input threshold voltage	R1IN, R2IN	V_{CC} = 5 V, T_A = 25°C		1.7	2.4	V
V_{IT-}	Receiver negative-going input threshold voltage	R1IN, R2IN	V_{CC} = 5 V, T_A = 25°C	0.8	1.2		V
V_{hys}	Input hysteresis voltage	R1IN, R2IN	V_{CC} = 5 V	0.2	0.5	1	V
r_i	Receiver input resistance	R1IN, R2IN	V_{CC} = 5, T_A = 25°C	3	5	7	kΩ

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.
NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 3 and Figure 1)

PARAMETER		TYP	UNIT
$t_{PLH(R)}$	Receiver propagation delay time, low- to high-level output	500	ns
$t_{PHL(R)}$	Receiver propagation delay time, high- to low-level output	500	ns

NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

B.13. LM1086

Circuito que contiene un ***Regulador de Tensión positiva de 1.5A****. La versión del circuito empleada tiene una tensión de salida de 3.3 voltios.*

National Semiconductor

November 2002

LM1086
1.5A Low Dropout Positive Regulators

General Description

The LM1086 is a series of low dropout positive voltage regulators with a maximum dropout of 1.5V at 1.5A of load current. It has the same pin-out as National Semiconductor's industry standard LM317.

The LM1086 is available in an adjustable version, which can set the output voltage with only two external resistors. It is also available in six fixed voltages: 1.8V, 2.5V, 2.85V, 3.3V, 3.45V and 5.0V. The fixed versions integrate the adjust resistors.

The LM1086 circuit includes a zener trimmed bandgap reference, current limiting and thermal shutdown.

The LM1086 series is available in TO-220, TO-263, and LLP packages. Refer to the LM1084 for the 5A version, and the LM1085 for the 3A version.

Features

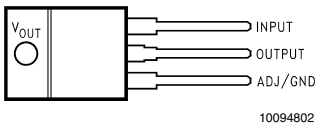
- Available in 1.8V, 2.5V, 2.85V, 3.3V, 3.45V, 5V and Adjustable Versions
- Current Limiting and Thermal Protection
- Output Current 1.5A
- Line Regulation 0.015% (typical)
- Load Regulation 0.1% (typical)

Applications

- SCSI-2 Active Terminator
- High Efficiency Linear Regulators
- Battery Charger
- Post Regulation for Switching Supplies
- Constant Current Regulator
- Microprocessor Supply

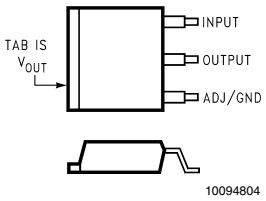
Connection Diagrams

TO-220



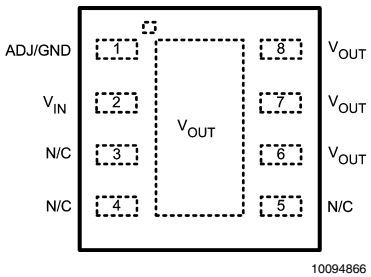
Top View

TO-263



Top View

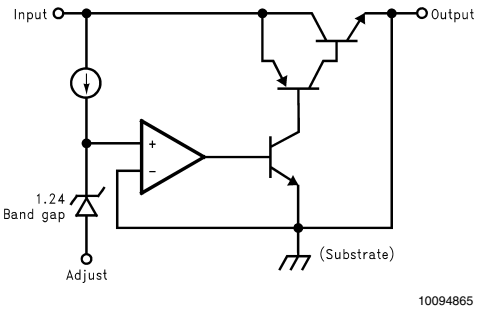
LLP



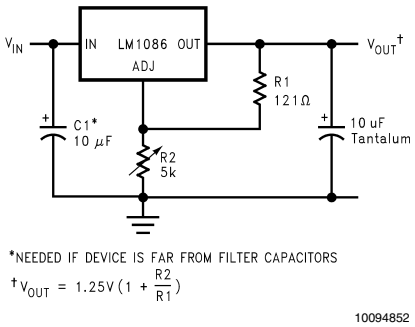
Pins 6, 7, and 8 must be tied together.

Top View

Basic Functional Diagram, Adjustable Version



Application Circuit



1.2V to 15V Adjustable Regulator

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Maximum Input-to-Output Voltage Differential	
LM1086-ADJ	29V
LM1086-1.8	27V
LM1086-2.5	27V
LM1086-2.85	27V
LM1086-3.3	27V
LM1086-3.45	27V
LM1086-5.0	25V
Power Dissipation (Note 2)	Internally Limited
Junction Temperature (T_J)(Note 3)	150˚C
Storage Temperature Range	-65˚C to 150˚C
Lead Temperature	260˚C, to 10 sec
ESD Tolerance (Note 4)	2000V

Operating Ratings (Note 1)

Junction Temperature Range (T_J) (Note 3)	
"C" Grade	
Control Section	0˚C to 125˚C
Output Section	0˚C to 150˚C
"I" Grade	
Control Section	–40˚C to 125˚C
Output Section	–40˚C to 150˚C

Electrical Characteristics

Typicals and limits appearing in normal type apply for T_J = 25˚C. Limits appearing in **Boldface** type apply over the entire junction temperature range for operation.

Symbol	Parameter	Conditions	Min (Note 6)	Typ (Note 5)	Max (Note 6)	Units
V_{REF}	Reference Voltage	LM1086-ADJ I_{OUT} = 10mA, V_{IN}–V_{OUT} = 3V 10mA ≤I_{OUT} ≤ $I_{FULL\ LOAD}$, 1.5V ≤ V_{IN}–V_{OUT} ≤ 15V (Note 7)	1.238 **1.225**	1.250 **1.250**	1.262 **1.270**	V V
V_{OUT}	Output Voltage (Note 7)	LM1086-1.8 I_{OUT} = 0mA, V_{IN} = 5V 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$, 3.3V ≤ V_{IN} ≤ 18V	1.782 **1.764**	1.8 **1.8**	1.818 **1.836**	V
		LM1086-2.5 I_{OUT} = 0mA, V_{IN} = 5V 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$, 4.0V ≤ V_{IN} ≤ 18V	2.475 **2.450**	2.50 2.50	2.525 **2.55**	V
		LM1086-2.85 I_{OUT} = 0mA, V_{IN} = 5V 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$, 4.35V ≤ V_{IN} ≤ 18V	2.82 **2.79**	2.85 **2.85**	2.88 **2.91**	V V
		LM1086-3.3 I_{OUT} = 0mA, V_{IN} = 5V 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$, 4.75V ≤ V_{IN} ≤ 18V	3.267 **3.235**	3.300 **3.300**	3.333 **3.365**	V V
		LM1086-3.45 I_{OUT} = 0mA, V_{IN} = 5V 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$, 4.95V ≤ V_{IN} ≤ 18V	3.415 3.381	3.45 3.45	3.484 3.519	V V
		LM1086-5.0 I_{OUT} = 0mA, V_{IN} = 8V 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$, 6.5V ≤ V_{IN} ≤ 20V	4.950 **4.900**	5.000 **5.000**	5.050 **5.100**	V V
ΔV_{OUT}	Line Regulation (Note 8)	LM1086-ADJ I_{OUT} =10mA, 1.5V≤ (V_{IN}-V_{OUT}) ≤ 15V		0.015 **0.035**	0.2 **0.2**	% %
		LM1086-1.8 I_{OUT} = 0mA, 3.3V ≤ V_{IN} ≤ 18V		0.3 **0.6**	6 **6**	mV
		LM1086-2.5 I_{OUT} = 0mA, 4.0V ≤ V_{IN} ≤ 18V		0.3 **0.6**	6 **6**	mV

Electrical Characteristics (Continued)

Typicals and limits appearing in normal type apply for T_J = 25˚C. Limits appearing in **Boldface** type apply over the entire junction temperature range for operation.

Symbol	Parameter	Conditions	Min (Note 6)	Typ (Note 5)	Max (Note 6)	Units
		LM1086-2.85		0.3	6	mV
		I_{OUT} = 0mA, 4.35V ≤ V_{IN} ≤ 18V		**0.6**	**6**	mV
		LM1086-3.3		0.5	10	mV
		I_{OUT} = 0mA, 4.5V ≤ V_{IN} ≤ 18V		**1.0**	**10**	mV
		LM1086-3.45		0.5	10	mV
		I_{OUT} = 0mA, 4.95V ≤ V_{IN} ≤ 18V		**1.0**	**10**	mV
		LM1086-5.0		0.5	10	mV
		I_{OUT} = 0mA, 6.5V ≤ V_{IN} ≤ 20V		**1.0**	**10**	mV
ΔV_{OUT}	Load Regulation (Note 8)	LM1086-ADJ		0.1	0.3	%
		(V_{IN}-V_{OUT}) = 3V, 10mA ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$		**0.2**	**0.4**	%
		LM1086-1.8 ,2.5, 2.85		3	12	mV
		V_{IN} = 5V, 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$		**6**	**20**	mV
		LM1086-3.3, 3.45		3	15	mV
		V_{IN} = 5V, 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$		**7**	**25**	mV
		LM1086-5.0		5	20	mV
		V_{IN} = 8V, 0 ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$		**10**	**35**	mV
	Dropout Voltage (Note 9)	LM1086-ADJ, 1.8, 2.5,2.85, 3.3, 3.45, 5				
		ΔV_{REF} = 1%, I_{OUT} = 1.5A		**1.3**	**1.5**	V
I_{LIMIT}	Current Limit	LM1086-ADJ				
		V_{IN}–V_{OUT} = 5V	**1.50**	**2.7**		A
		V_{IN}–V_{OUT} = 25V	**0.05**	**0.15**		A
		LM1086-1.8,2.5, 2.85, 3.3, 3.45, V_{IN} = 8V	**1.5**	**2.7**		A
		LM1086-5.0, V_{IN} = 10V	**1.5**	**2.7**		A
	Minimum Load Current (Note 10)	LM1086-ADJ				
		V_{IN} –V_{OUT} = 25V		**5.0**	**10.0**	mA
	Quiescent Current	LM1086-1.8, 2.5, 2.85, V_{IN} ≤ 18V		**5.0**	**10.0**	mA
		LM1086-3.3, V_{IN} ≤ 18V		**5.0**	**10.0**	mA
		LM1086-3.45, V_{IN} ≤ 18V		**5.0**	**10.0**	mA
		LM1086-5.0, V_{IN} ≤ 20V		**5.0**	**10.0**	mA
	Thermal Regulation	T_A = 25˚C, 30ms Pulse		0.008	0.04	%/W
	Ripple Rejection	f_{RIPPLE} = 120Hz, C_{OUT} = 25µF Tantalum, I_{OUT} = 1.5A				
		LM1086-ADJ, C_{ADJ} = 25µF, (V_{IN}–V_O) = 3V	**60**	**75**		dB
		LM1086-1.8, 2.5, 2.85, V_{IN} = 6V	**60**	**72**		dB
		LM1086-3.3, V_{IN}= 6.3V	**60**	**72**		dB
		LM1086-3.45, V_{IN}= 6.3V	**60**	**72**		dB
		LM1086-5.0 V_{IN} = 8V	**60**	**68**		dB
	Adjust Pin Current	LM1086		55	**120**	µA
	Adjust Pin Current Change	10mA ≤ I_{OUT} ≤ $I_{FULL\ LOAD}$, 1.5V ≤ (V_{IN}–V_{OUT}) ≤ 15V		**0.2**	**5**	µA
	Temperature Stability			**0.5**		%
	Long Term Stability	T_A = 125˚C, 1000Hrs		0.3	1.0	%
	RMS Noise (% of V_{OUT})	10Hz ≤ f≤ 10kHz		0.003		%
θ_{JC}	Thermal Resistance Junction-to-Case	3-Lead TO-263: Control Section/Output Section			1.5/4.0	˚C/W
		3-Lead TO-220: Control Section/Output Section			1.5/4.0	˚C/W

B.14. MC78T05 y MC78T12

Circuito que contiene un ***Regulador de Tensión positiva de 3A****. Las versiones del circuito empleadas tiene una tensión de salida de 5 y 12 voltios.*

MC78TXX

3-Terminal 3A Positive Voltage Regulator

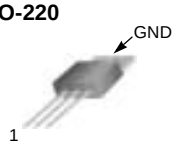
Features

- Output Current in Excess of 3.0A
- Output Transistor Safe Operating Area Compensation
- Power Dissipation :25W
- Internal Short Circuit Current Limiting
- Internal Thermal Overload Protection
- Output Voltage Offered in 4% Tolerance
- No External Components Required
- Output Voltage of 5,12 and 15V

Description

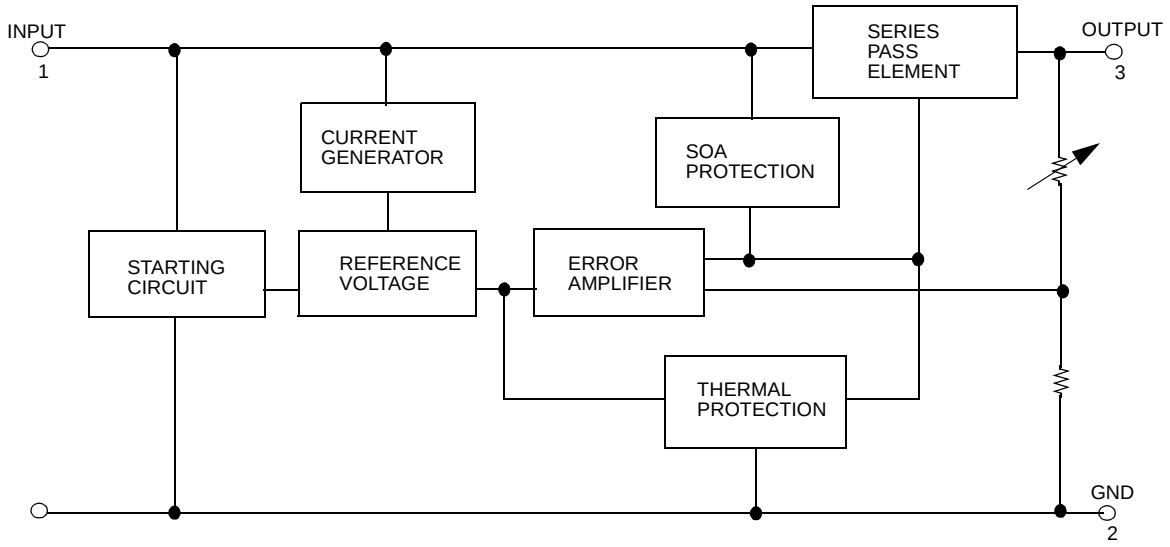
This family of fixed voltage regulators are monolithic integrated circuit capable of driving loads in excess of 3.0 A.

TO-220



1.Input 2. GND 3. Output

Internal Block Diagram



Rev. 1.0.1

Absolute Maximum Ratings

Parameter	Symbol	Value	Unit
Input Voltage (for V_O = 5V to 12V) (for V_O = 15V)	V_I	35 40	V V
Power Dissipation	PD	Internally limited	
Thermal Resistance, Junction to Air (Note1, 2) Ta = +25°C	RθJA	65	°C/W
Thermal Resistance, Junction to Case (Note1) Tc = +25°C	RθJC	2.5	°C/W
Operating Junction Temperature Range	T_J	0 ~ +125	°C
Storage Temperature Range	T_{STG}	-65 ~ +150	°C

Note:

1. Thermal resistance test board
   Size: 76.2mm * 114.3mm * 1.6mm(1S0P)
   JEDEC standard: JESD51-3, JESD51-7
2. Assume no ambient airflow.

Electrical Characteristics(MC78T05)

(V_I = 10V, I_O = 3.0 A, 0°C ≤ T_J ≤ +125°C, Po ≤ P_{MAX} (Note3), unless otherwise specified.)

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Output Voltage	Vo	5mA ≤ Io ≤ 3.0A , T_J = +25°C 7.3V ≤ VI ≤ 20V, 5mA ≤ Io ≤ 2.0A	4.8 4.75	5.0 5.0	5.2 5.25	V
Line Regulation (Note4)	ΔV_O	7.2V ≤ VI ≤ 35V , Io=5mA, T_J =+25°C 7.2V ≤ VI ≤ 35V , Io=1.0A, T_J = +25°C 7.5V ≤ VI ≤ 20V, Io =2.0A, T_J = +25°C 8.0V ≤ VI ≤ 12V, Io =3.0A, T_J = +25°C	-	3.0	25	mV
Load Regulation (Note4)	ΔV_O	5mA ≤ Io ≤ 3.0A , T_J = +25°C 5mA ≤ Io ≤ 3.0A	-	10 15	30 80	mV mV
Thermal Regulation	REG_T	Pulse =10ms, P = 20W T_A = +25°C	-	0.002	0.03	%Vo/W
Quiescent Current	I_Q	5mA ≤ Io ≤ 3.0A , T_J = +25°C 5mA ≤ Io ≤ 3.0A	-	3.5 4.0	5.0 6.0	mA mA
Quiescent Current Change	ΔI_Q	7.2V ≤ VI ≤ 35V, Io = 5mA T_J = +25°C ; 7.5V ≤ VI ≤ 20V, Io =2.0A ; 5mA ≤ Io ≤ 3.0A, T_J = +25°C	-	0.1	0.8	mA
Ripple Rejection	RR	f = 120Hz, 8V ≤ V_I ≤ 18V, I_O = 2.0A T_J = +25°C	-	75	-	dB
Dropout Voltage	V_D	Io = 3A ,T_J = +25°C	-	2.2	2.5	V
Output Noise Voltage	V_N	T_A = +25°C, 10Hz ≤ f ≤ 100kHz	-	10	-	μV/Vo
Peak Output Current	I_{PK}	T_A = +25°C	-	5.0	-	A
Output Resistance	Ro	f = 1.0kHz	-	2.0	-	mΩ
Short Circuit Current Limit	Isc	VI = 35V, T_J =+25°C	-	1.5	2.5	A
Average Temperature Coefficient of Output Voltage	$\Delta V_O/\Delta T$	Io = 5.0mA	-	0.2	-	mV/°C

Note:

3. Although power dissipation is internally limited, specifications apply only for P_O ≤ Pmax, Pmax = 25W
4. Load and line regulation are specified at constant junction temperature. Change in Vo due heating effects must be taken into account separately. Pulse testing with low duty is used.

Electrical Characteristics(MC78T12) (Continued)

(V_I = 19V, I_O = 3.0 A, 0°C ≤ T_J ≤ +125°C, Po ≤ P_{MAX} (Note1), unless otherwise specified.)

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Output Voltage	Vo	5mA ≤ Io ≤ 3.0A , T_J =+25°C 14.5V ≤ VI ≤ 27V, 5mA ≤ Io ≤ 2.0A	11.5 11.4	12 12	12.5 12.8	V
Line Regulation (Note2)	ΔV_O	14.5V ≤ VI ≤ 35V, Io=5mA, T_J =+25°C 14.5V ≤ VI ≤ 35V, Io=1.0A, T_J =+25°C 14.9V ≤ VI ≤ 28V, Io =2.0A, T_J =+25°C 16V ≤ VI ≤ 22V, Io =3.0A, T_J =+25°C	-	6.0	45	mV
Load Regulation (Note2)	ΔV_O	5mA ≤ Io ≤ 3.0A, T_J =+25°C 5mA ≤ Io ≤ 3.0A	-	10 15	30 80	mV mV
Thermal Regulation	REG_T	Pulse =10ms, P = 20W TA = +25°C	-	0.002	0.03	%Vo/W
Quiescent Current	I_Q	5mA ≤ Io ≤ 3.0A, T_J =+25°C 5mA ≤ Io ≤ 3.0A	-	3.5 4.0	5.0 6.0	mA mA
Quiescent Current Change	ΔI_Q	14.5V ≤ VI ≤ 35V, Io = 5mA T_J =+25°C ; 14.9V ≤ VI ≤ 27V, Io =2.0A ; 5mA ≤ Io ≤ 3.0A, T_J =+25°C	-	0.1	0.8	mA
Ripple Rejection	RR	f = 120Hz, 15V ≤ V_I ≤ 25V, Io = 2.0A T_J =+25°C	-	67	-	dB
Dropout Voltage	V_D	Io = 3A,T_J =+25°C	-	2.2	2.5	V
Output Noise Voltage	V_N	T_A =+25°C, 10Hz ≤ f ≤ 100kHz	-	10	-	μV/Vo
Peak Output Current	I_{PK}	T_A =+25°C	-	5.0	-	A
Output Resistance	Ro	f = 1.0kHz	-	2.0	-	mΩ
Short Circuit Current Limit	Isc	VI = 35V, T_J =+25°C	-	1.5	2.5	A
Average Temperature Coefficient of Output Voltage	$\Delta V_O/\Delta T$	Io = 5.0mA	-	0.5	-	mV/°C

Note:

1. Although power dissipation is internally limited, specifications apply only for P_O ≤ Pmax, Pmax = 25W
2. Load and line regulation are specified at constant junction temperature. Change in Vo due heating effects must be taken into account separately. Pulse testing with low duty is used. (P_{MAX} = 25W)

Apéndice C

Contenido del CD

En el CD-Rom que se adjunta con el proyecto fin de carrera, podemos encontrar los siguientes contenidos:

- **Código Fuente**: Ficheros en código ensamblador del software del reproductor.
  - **Bootloader**: Código fuente del bootloader.
  - **Reproductor**: Código fuente del reproductor.
- **Código compilado**: Ficheros compilados listos para ser grabados en el PIC.
- **Documentación Adicional**: Documentación acerca de diversos aspectos relacionados con el proyecto.
  - **Latex**: Ficheros de ayuda para generar documentos en LaTeX.
  - **Proyecto**: Documentos sobre FAT32, ISO9660, MBR,..
- **Esquemas**: Ficheros en formato Proteus con los esquemas del reproductor.
- **Memoria**: Memoria del proyecto en formato LaTeXy PDF.
- **PDFs**: PDF's con las hojas de especificaciones de los principales componentes usados en este proyecto.
  - **Decodificador**: Especificaciones del decodificador de 'MP3' usado.
  - **Digital**: Especificaciones de todos los integrados digitales usados.
  - **Disco duro**: Características técnicas del disco duro.
  - **LCD**: Todo lo referente al display LCD y su interfaz.
  - **Memoria**: Especificaciones de los chips de memoria empleados.

  - **Microprocesador**: Datos técnicos, notas de aplicación, todo lo necesario para programar correctamente el microprocesador.
  - **Reguladores de tensión**: Especificaciones de los reguladores de tensión usados.
  - **USB**: Información acerca de los integrados que podrían usarse para añadir una interfaz USB al reproductor.

- **Presentación**: Ficheros en formato Power Point con la presentación del proyecto
- **Utilidades**: Programa para actualizar el firmware del dispositivo desde un PC.

Bibliografía

[1] Technical Committee T13 AT Attachment. *Information technology - AT Attachment Interface for Disk Drives.*
`http://www.t13.org/`.

[2] G.H. Bennett. *PCM y transmisiín digital.* Ediciones Técnicas Rede, 1988.

[3] Microsoft Corporation. *Microsoft Extensible Firmware Initiative FAT32 File System Specificatio.*

[4] Peter Flynn. *Formatting information - A beginner's introduction to typesetting with LATEX.*

[5] Juan F. Marcelo and Eva Martín. *MP3.* Anaya Multimedia, 2000.

[6] I. Angulo Martínez, José María Angulo Usategui, and Susana Romero Yesa. *MICRO-CONTROLADORES PIC 16F87X.* McGraw-Hill, 2000.

[7] VLSI Solution Oy. *VS1001k - MPEG AUDIO CODEC.*
`http://www.vlsi.fi/download/download.shtml`.

[8] VLSI Solution Oy. *VS10XX - APPLICATION NOTES.*
`http://www.vlsi.fi/download/download.shtml`.