Facultad de Ingeniería – UNMdP- 2007

TÍTULO DEL PROYECTO:

DATALOGGER BASADOS EN MICROCONTROLADOR Y TARJETA FLASH SD

AUTOR: EDGARDO RODOLFO HIRSCH

DIRECTORES: ING. RIVERA, RAÚL
ING. GERMÍN, WALTER

ÍNDICE

CAPÍTULO 1: Introducción y Diseño del Sistema

1.1.Resumen.....5

1.2. Introducción.....6

1.2.1 Microcontroladores. La familia de los PIC.....6

1.2.2. Características de los PIC.....7

1.2.3. La memoria en los PIC.....9

1.2.4. Memorias FLASH.....9

1.2.5. Tarjetas de memoria FLASH.....10

1.2.6. Tipos de tarjetas de memoria FLASH.....11

1.3. Elección del hardware y herramientas.....12

1.3.1. El PIC.....12

1.3.2. La Tarjeta FLASH.....13

1.3.3. El sistema de archivos.....13

1.3.4. Las herramientas de desarrollo software.....14

1.4. Diseño del Sistema.....14

1.4.1. Hardware.....14

1.4.2. El Software.....18

CAPÍTULO 2: Descripción de los Elementos del Sistema

2.1. El PIC18F4550.....21

2.1.1. Características Generales.....21

2.1.2. Memoria FLASH.....23

2.2. Sistema de Archivos FAT16.....23

2.2.1. Características del Sistema FAT.....23

2.2.2. El BPB (Bios Parametric Block).....25

2.2.3. La Tabla FAT.....26

2.2.4. La Zona de Datos.....27

2.2.4.1. Estructuras de directorios FAT....28

2.3. Tarjeta SD (Secure Digital) y MMC (MultiMediaCard)....31
2.3.1. Introducción....31
2.3.2. Características y Funcionamiento de la Tarjeta SD/MMC....33
2.3.3. Registros internos de una tarjeta SD/MMC....36
2.3.4. Modo MMC....38
2.3.5. Modo SPI....38
2.3.5.1. Protocolo SPI....39
2.3.5.2. Comandos de la tarjeta en modo SPI....41
2.3.5.3. Respuestas de la tarjeta en modo SPI....43
2.3.5.4. Transacción de datos en la tarjeta en modo SPI....44
2.3.5.5. Uso de los registros de la tarjeta en modo SPI....46
2.3.5.6. Funcionamiento de la tarjeta en modo SPI....48
2.3.6. El MBR....50
2.3.6.1. Código Maestro de Carga (MBC)....51
2.3.6.2. Tabla Maestra de Particiones (MPT)....51
2.3.6.3. Marcador del Sector de Carga....53

2.4. Herramientas de Desarrollo....53
2.4.1. El código C y el compilador PIC-C....53
2.4.2. Formato Intel HEX....54
2.4.3. MPLAB® IDE....55

CAPÍTULO 3: Desarrollo del Código

3.1. Modulo principal....56
3.1.1. Desarrollo....56
3.1.2. Tarjeta MMC/SD....57
3.1.3. Reloj de Tiempo Real....59
3.1.4. Sistema de Archivos FAT16....59
3.2. Distribución de la Memoria....61

CAPÍTULO 4: Guía de Usuario del Sistema

4.1. Funcionamiento General.....64

4.2. Desarrollo de Aplicaciones.....64
4.2.1. Recursos utilizados por el software.....65
4.2.1.1. Manejo de las librerías del sistema.....66
4.2.1.2. Finalización del programa.....66
4.2.1.3. Dependencia del código con el compilador.....66

CAPÍTULO 5: Conclusiones

5.1. Conclusiones.....67
5.2. Aplicaciones.....67
5.3 Bibliografía.....68

ANEXO

Contenido del CD.....69

CAPÍTULO 1:

INTRODUCCIÓN Y DISEÑO DEL SISTEMA

1.1. Resumen

En este proyecto se desarrolló una plataforma portable y autónoma de hardware y software, que dota a un sistema basado en un microcontrolador PIC, de la capacidad de manejar tarjetas de memoria Flash SD (y debido a su compatibilidad es extensivo también a las tarjetas MMC).

El usuario del sistema será aquel que desarrolle una aplicación especifica de adquisición de datos para almacenarlos en la tarjeta SD en forma de archivo de texto (con extensión *.txt). Por lo tanto, su contenido, podrá ser visualizado por casi todos los ordenadores, ya que la mayoría funcionan bajo el sistema operativo de Microsoft.

El tipo de microcontrolador escogido limitó en gran medida las aplicaciones. Además, su arquitectura también influyó en la forma en la que se desarrolló el software para él. Por tanto se ha de entender que el sistema que se desarrolló será muy específico para los microcontroladores PIC. Igualmente ocurre para el tipo de tarjeta de memoria Flash escogida.

Los siguientes elementos fueron utilizados para estructurar y darle funcionabilidad al sistema de adquisición.

- El microcontrolador (Pic18F4550)
- Sistema de Archivos (FAT16)
- Almacenamiento de datos (Tarjeta SD)
- Hardware (interfaz SPI)
- Herramientas de Desarrollo (El código C y el compilador PIC-C)

En todo momento, se trató de explicar de forma teórica las fases del desarrollo, de forma que el usuario comprenda el funcionamiento y los principales mecanismos del sistema, lo cual facilitará en gran medida el proceso de portabilidad a sistemas basados en otro tipo de microcontroladores, o para otro tipo de tarjetas.

Como "sistema de propósito general", tal y como se ha diseñado, ofrece las mayores posibilidades. El usuario puede utilizar un mismo sistema para realizar diversas tareas.

1.2. Introducción

En los últimos años los microcontroladores han evolucionado mucho, son cada vez más rápidos y poseen mayores capacidades de memoria. Poseen más periféricos, tales como conversores A/D, temporizadores, puertos serie, puertos USB, etc...En principio, su uso está pensado para ejecutar programas monolíticos que permanecen siempre almacenados en la memoria, y que desarrollan una función muy concreta, constituyendo un sistema de propósito específico. Además la propia aplicación dicta el tipo de microcontrolador a utilizar, los periféricos de este, la capacidad, la velocidad, etc...

Cuanto mayor sea la capacidad y la potencia del microcontrolador, mayores y más complejos pueden ser los programas que ejecute, con lo cual cobra más sentido para ellos la idea de un "sistema de propósito general". En torno a esta idea se desarrolló este trabajo.

A continuación se detallan brevemente los antecedentes y herramientas de los dispositivos que se usaron en este proyecto.

1.2.1. Microcontroladores. La familia de los PIC.

Un microcontrolador consiste en un circuito integrado que incluye en su interior las tres unidades funcionales de un computador: la unidad central de proceso o CPU, la memoria y las unidades de Entrada/Salida. Además, también suelen disponer de otros periféricos tales como: puertos serie, comparadores, convertidores analógico-digitales, etc... Aunque sus prestaciones son limitadas, además de dicha integración, su característica principal es su alto nivel de especialización, existiendo una amplia variedad de tipos y modelos de modo que se pueda encontrar el modelo adecuado para cada aplicación.

Las aplicaciones de los microcontroladores son cada vez más amplias y se pueden encontrar tanto en el sector industrial como en el sector de consumo.

De entre todos estos se escogió trabajar con un microcontrolador PIC de la empresa Microchip. Las principales características por las que se ha optado por el uso de un PIC son las siguientes:

-la amplia gama de modelos existente.

-las herramientas de desarrollo disponibles.

-el uso extendido de la familia de los PIC en la docencia.

-la gran cantidad de información disponible.

-la capacidad de escribir en su memoria mediante instrucciones en los modelos con memoria Flash.

-unidades funcionales integradas (temporizadores, USART,I2C,SPI, unidades de comparación/captura/PWM,convertidores A/D,USB, etc)

2ª. Se aplica la técnica de segmentación ("pipe-line") en la ejecución de las instrucciones.

La segmentación permite al procesador ejecutar cada instrucción en un ciclo de instrucción equivalente a cuatro ciclos de reloj. En cada ciclo se realiza la búsqueda de una instrucción y la ejecución de la anterior

La segmentación permite al procesador realizar al mismo tiempo la ejecución de una instrucción y la búsqueda del código de la siguiente.

3ª. El formato de todas las instrucciones tiene la misma longitud.

Salvo algunas excepciones, todas las instrucciones de los microcontroladores de la gama baja tienen una longitud de 12 bits. Las de la gama media tienen 14 bits y 16 bits las de los PIC de la gama alta. Esta característica es muy ventajosa en la optimización de la memoria de instrucciones y facilita enormemente la construcción de ensambladores y compiladores.

4ª. Procesador RISC (Computador de Juego de Instrucciones Reducido)

Dependiendo de la gama del procesador (baja, media o alta) disponen de más o menos número de instrucciones. Los modelos de la gama baja disponen de un repertorio de 33 instrucciones, 35 los de la gama media y unas 75 los de la alta. Según el modelo puede tener además un repertorio extendido de instrucciones.

5ª. Todas las instrucciones son ortogonales

Cualquier instrucción puede manejar cualquier elemento de la arquitectura como fuente o como destino.

6ª. Arquitectura basada en un banco de registros.

Esto significa que todos los objetos del sistema (puertos de E/S, temporizadores, posiciones de memoria, etc.) están implementados físicamente como registros.

7ª. Diversidad de modelos de microcontroladores con prestaciones y recursos diferentes.

La gran variedad de modelos de microcontroladores PIC permite que el usuario pueda seleccionar el más conveniente para su proyecto.

8ª. Herramientas de soporte potentes y económicas

Existen numerosas herramientas para el desarrollo de software para los PIC. La propia empresa Microchip y otras que ponen a disposición de los usuarios numerosas herramientas para desarrollar hardware y software. Son muy abundantes los programadores, los simuladores software, los emuladores en tiempo real, ensambladores, Compiladores C, Intérpretes y Compiladores BASIC, etc.

1.2.3. La memoria en los PIC.

Al estar construidos con arquitectura Harvard, poseen dos bloques de memoria distintos, uno para la memoria de datos y otro para la de programa. Cada bloque posee su propio bus, de tal forma que el acceso a cada uno puede producirse durante el mismo ciclo del oscilador. Estas dos memorias son independientes entre ellas teniendo tamaño y longitudes de palabra distintas.

Memoria de programa.

Existen modelos de PIC con distintos tipos de memoria de programa dependiendo de las necesidades, OTP, EPROM, FLASH, etc. En nuestro caso, nos centraremos en los PIC con memoria FLASH, que son memorias que pueden borrarse y programarse eléctricamente un gran número de veces y a gran velocidad, además de poder hacerlo sin necesidad de tensiones especiales. En esta memoria es donde se almacena el programa a ejecutar por el PIC.

Memoria de datos.

La memoria de datos puede dividirse en; Registros de Funciones Especiales (SFR) y la RAM de propósito general. Los SFRs sirven para gobernar los distintos periféricos del PIC. La memoria de datos se divide en "bancos" y la cantidad de estos depende de la memoria disponible para el modelo de PIC en uso.

1.2.4. Memorias FLASH.

La memoria Flash es una forma evolucionada de la memoria EEPROM que permite que múltiples posiciones de memoria sean escritas o borradas en una misma operación de programación mediante impulsos eléctricos, frente a las anteriores que sólo permite escribir o borrar una única celda cada vez. Por ello, la tecnología flash permite funcionar a velocidades muy superiores cuando los sistemas emplean lectura y escritura en diferentes puntos de esta memoria al mismo tiempo, lo cual las hace muy adecuadas para sistemas de almacenamiento masivos y sistemas donde se precise alta velocidad de borrado y escritura en la memoria.

Las memorias flash son de tipo no volátil, esto es, la información que almacena no se pierde cuando se desconecta de la corriente, una característica muy valorada para la multitud de usos en los que se emplea este tipo de memoria.

Los principales usos de este tipo de memorias pueden ser:

-Memoria ROM para dispositivos electrónicos, tales como microcontroladores, BIOS para PCs, etc...

-Memoria de almacenamiento para pequeños electrodomésticos, tales como reproductores portátiles, cámaras fotográficas, etc...

-Tarjetas o cartuchos de almacenamiento masivo, como pueden ser las memorias USB y que sustituyen a discos duros o disquetes.

En el desarrollo de este proyecto nós encontramos con dos tipos de memoria Flash:

-La incluida en la memoria de programa del PIC. Al disponer de memoria Flash nos permite cargar programas en su memoria un gran número de veces, así como hacerlo de una forma eléctrica, sencilla y rápida

-La incluida en la tarjeta de memoria MMC/SD, la cual supone un sistema de almacenamiento masivo.

1.2.5. Tarjetas de Memoria Flash.

De entre todas las aplicaciones de la memoria Flash, son las tarjetas de memoria, las que mayor evolución han sufrido en los últimos tiempos, debido a la gran cantidad de aplicaciones en las que se pueden ver en la electrónica de consumo. Y serán las que mayor evolución sufrirán en el futuro, ya que se piensa que tratarán de sustituir a los discos duros, por la cantidad de ventajas que suponen frente a estos. Estas ventajas se pueden resumir en:

-El gran ahorro de consumo, al no contener partes mecánicas que mover.
-El ahorro de tamaño.
-Resistencia a golpes.
-La velocidad de acceso.

Aún presentan ciertas desventajas frente a los discos duros:
-El elevado precio frente a los discos duros.
-Las capacidades de memoria que aún manejan.
-El número limitado de veces que pueden ser escritas.

Las capacidades de almacenamiento de estas tarjetas que integran memorias flash comenzaron en 8 MB pero actualmente se pueden encontrar en el mercado tarjetas bastante asequibles con capacidad de 2 GB. A fines de 2007, ya hay fabricantes que anuncian ordenadores portátiles con tarjetas de memoria flash de 32GB. La velocidad de transferencia, al igual que la capacidad de almacenamiento, se ha ido incrementando progresivamente. La nueva generación de tarjetas permitirá velocidades de hasta 20 MB/s.

Como se ha dicho, todos los tipos de memoria flash sólo permiten un número limitado de escrituras y borrados, generalmente entre cien mil y un millón, dependiendo de la celda, de la precisión del proceso de fabricación y del voltaje necesario para su borrado. Este tipo de memoria está fabricado con puertas lógicas NOR y NAND para almacenar los 0's ó 1's correspondientes. Actualmente hay una gran división entre los fabricantes de un tipo u otro, especialmente a la hora de elegir un sistema de archivos para estas memorias. Sin embargo se está comenzando a desarrollar memorias basadas ORNAND en que reúne lo mejor de cada una de las tecnologías anteriores.

Los sistemas de archivos para estas memorias están en pleno desarrollo, aunque ya en funcionamiento como por ejemplo JFFS originalmente para NOR, evolucionado a JFFS2 para soportar además la tecnología NAND. O YAFFS, ya en su segunda versión, para NAND. Sin embargo, en la práctica, el sistema de archivos más empleado es el FAT de Microsoft, por compatibilidad, sobre todo, con la mayoría de (Computadoras Personales) PCs, ya que es la principal plataforma utilizada para transmitir o leer datos desde la tarjeta.

Otra característica de reciente aparición ha sido la resistencia térmica de algunos encapsulados de tarjetas de memoria orientadas a las cámaras digitales de gama alta. Esto permite funcionar en condiciones extremas de temperatura ya que el rango de temperaturas soportado abarca desde los -25 °C hasta los 85 °C.

1.2.6. Tipos de tarjetas de Memoria Flash.

Existe una gran variedad de tarjetas y a su vez, una gran cantidad de variantes de cada tipo. Podemos resumir las principales en la lista siguiente:

- **Compact Flash (CF)**: estas tarjetas son de las más usadas hoy en día. Al poder ser fabricadas de acuerdo a los estándares de una asociación hay mucha variedad de marcas; las más reconocidas son Sandisk y Kingston. Hay dos tipos de tarjetas, la Type I y Type II que tienen más capacidad, son mucho más rápidas, pero son más gruesas que las TypeI. Este es el tipo de tarjeta más barata entre todas.
- **Memory Stick:** Originalmente desarrolladas por Sony, disponen de varios tipos a su vez, como pueden ser las "Estándar", las PRO, las DUO o las PRO

DUO. Son de las más caras en relación capacidad/precio y son de las que se ha anunciado una mayor evolución. Se esperan modelos de hasta 32 GB de capacidad y hasta 160 MB/s de transferencia.

- **MultiMediaCard (MMC):** Desarrolladas por una asociación, lo que implica que existan muchos fabricantes. Son de las más flexibles y se utilizan en bastantes dispositivos. Como ventajas tienen fundamentalmente su bajo precio con relación a las otras. Es la que se utilizara en este proyecto. Una información detallada puede verse en su correspondiente capítulo.
- **Secure Digital (SD)**: Son tarjetas basadas en el formato MMC pero se crearon para ofrecer determinadas ventajas frente a éstas, como puede ser una mayor velocidad de transferencia de datos, protección contra escritura, etc... Ofrecen una gran compatibilidad con las anteriores, de modo que puedan usarse indistintamente en muchos dispositivos. Son más caras que la MMC. A su vez presentan una amplia gama de variantes y subtipos.
- **Smart Media**: Fue una de las primeras tarjetas que se vieron en el mercado, por tanto no son demasiado avanzadas y están tendiendo a desaparecer
- **XD-Photo Card**: Es un tipo de tarjeta propietario de Olympus, con lo cual sólo es fabricado por Olympus y pocas marcas más. Se utilizan principalmente en cámaras de fotos de estas marcas y la ventaja es que las cámaras de esas marcas tienen hasta el firmware optimizado para el crecimiento de las mismas.

Como se ha comentado anteriormente, la tarjeta escogida para el proyecto es la SD. En el siguiente capítulo se detallarán más características de esta tarjeta y se expondrá el porqué se adoptaron para el proyecto.

1.3. Elección del hardware y herramientas.

A continuación se detallan brevemente los dispositivos y herramientas que se usaron en el proyecto. Las características técnicas de dichos dispositivos pueden verse en los siguientes capítulos.

1.3.1. El microcontrolador PIC.

Ya se ha comentado el por qué del uso de un PIC. Ahora debemos seleccionar el modelo adecuado. En la página de microchip pueden verse todos los modelos existentes (www.microchip.com). Se ofrecen en diversos tipos de encapsulado, y de ellos, el que mejor se adaptó a nuestro proceso de desarrollo, es el PDIP de 40 pines debido a que puede ser

fácilmente extraído de un zócalo y se puede usar con un programador muy fácilmente. Así, de entre los disponibles con encapsulado PDIP, se ha escogido el que ofrezca mayores capacidades de memoria. Por tanto, de los modelos que ofrece microchip, se opto por elegir el PIC184550, que dispone de 32.768 Bytes de memoria de código (Flash), 2.048 Bytes de memoria RAM de datos y memoria EEPROM de datos 256 Bytes.

1.3.2. La Tarjeta Flash.

Como se explicó en el apartado anterior, se ha optado por trabajar con tarjetas SD por diversas razones. En primer lugar, las tarjetas SD representan un estándar, por tanto existen muchos fabricantes, lo que implica que sean desarrolladas para gran cantidad de usos en general. Esto también conlleva a su bajo precio con respecto a otro tipo de tarjetas. Además, poseen un modo de comunicación alternativo al creado en el estándar MMC, el modo SPI. Esto la hace especialmente compatible con la gran mayoría de controladores del mercado, ya que la mayoría de estos, disponen de un módulo SPI para la comunicación serie.

Se ha adquirido más concretamente una tarjeta SD de 128 MB de capacidad.

1.3.3. El sistema de archivos

Ya se ha comentado que no existe un sistema de archivos suficientemente desarrollado para tarjetas de memoria Flash. La gran mayoría de dispositivos electrónicos con memorias Flash utilizan el sistema de archivos FAT, de Microsoft, más concretamente en la versión FAT16. Esto es debido a que normalmente, cualquiera de esos dispositivos necesita de comunicación con un ordenador personal, y al día de hoy la gran mayoría de ordenadores funcionan con sistemas operativos de Microsoft. Por ejemplo, una cámara de fotos digital, necesita comunicarse con un ordenador para descargar las fotos, y el ordenador necesita conocer el sistema de archivos de la tarjeta, para poder trabajar con esos archivos de imagen.

Esto hace que el sistema FAT sea el más utilizado en dispositivos de almacenamiento portátil. Microsoft, ha desarrollado al menos tres variantes de FAT, FAT12, de uso en disquetes, FAT16 desarrollado para los primeros discos duros, y FAT32, creado con la introducción de discos duros de gran capacidad. El hecho de hablar de discos duros corresponde con que el sistema de archivos fue desarrollado para los sistemas operativos correspondientes para ordenador, que siempre han funcionado con discos duros. Sin embargo, el sistema de archivos es aplicable a cualquier sistema de almacenamiento masivo.

De entre todos ellos, se ha optado por el uso de FAT16, que es el sistema más adecuado para tarjetas de este tipo de hasta aproximadamente 4GB. FAT16 ofrece la principal ventaja de la rapidez frente a FAT32, pero ofrece la desventaja de que cuanto mayor sea la

partición, mayor sea el tamaño del clúster, con el consecuente desperdicio de memoria que ello conlleva. Una explicación más detallada de estos términos y del sistema FAT16 puede verse en el Capítulo 4.

1.3.4. Las herramientas de desarrollo software

Otra elección fundamental a la hora de crear un programa es la de elegir las herramientas software necesarias para el desarrollo.

La primera elección consiste en el lenguaje utilizado para el código fuente. Se ha optado por el uso del lenguaje C por la cantidad de ventajas que ofrece, y entre las que se pueden destacar dos fundamentalmente:

-Ofrece un alto nivel de abstracción del dispositivo a programar.

-Facilita en gran medida la portabilidad tanto a otros modelos de PIC, como a otros modelos de microcontrolador.

El hecho de programar en código C, implica el uso de un compilador que se encargue de traducir el lenguaje C a código máquina, que es el que entiende el microcontrolador. Se ha seleccionado trabajar con el compilador PIC-C (CCS C Compiler v.4).

La empresa Microchip, fabricante de los PIC, además ofrece un entorno de Programación y Desarrollo llamado Mplab. Consiste en un conjunto de aplicaciones gráficas para gestionar la creación y depuración de programas de manera sencilla pero muy potente.

1.4. Diseño del sistema.

Como ya se ha comentado anteriormente, el objetivo del proyecto es el de dotar a un sistema basado en un microcontroladores PIC de la capacidad de manejar archivos en una tarjeta Flash SD. Característica relevante a la hora de guardar los datos adquiridos.

1.4.1. Hardware.

Aunque la parte principal del proyecto consiste en el desarrollo del código, se han de definir determinadas características de hardware comunes a los sistemas para los que está diseñado. Estas características determinan la conexión del microcontrolador con la tarjeta SD.

La interfase hardware que se debe definir consiste en la que realiza la conexión del PIC con la tarjeta MMC/SD. Para ello se utilizó el módulo MSSP ("Master Synchronous Serial Port" o "Puerto Serie Síncrono Maestro"), que consiste en una interfase serie

implementada en el PIC para la comunicación con otros dispositivos. Este módulo permite la operación en dos modos diferentes:

-Modo SPI: Serial Peripherical Interface

-Modo I^2C: Inter-Integrated Circuit

Como se comentó anteriormente, la tarjeta MMC/SD ofrece un modo de funcionamiento SPI, de modo que se ha de configurar el módulo MSSP en modo SPI para la comunicación con la tarjeta.

Cada módulo dispone de sus propios pines en el PIC para su conexión, por tanto el siguiente esquema hardware será común en todos los sistemas integrados en los que se ejecute el software. Esto implica que los modelos de PIC en los que se vaya a ejecutar el software deban disponer de dicho módulo.

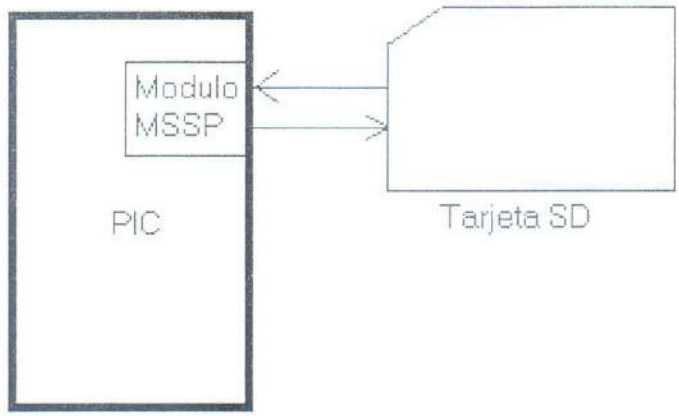

Figura 1.4. Conexión del PIC con la tarjeta y la PC

Diagrama esquemático:

La distribución topológica fue realizada mediante la utilización del programa LiveWire. La siguiente figura muestra el diagrama circuital utilizado y la distribución topológica.

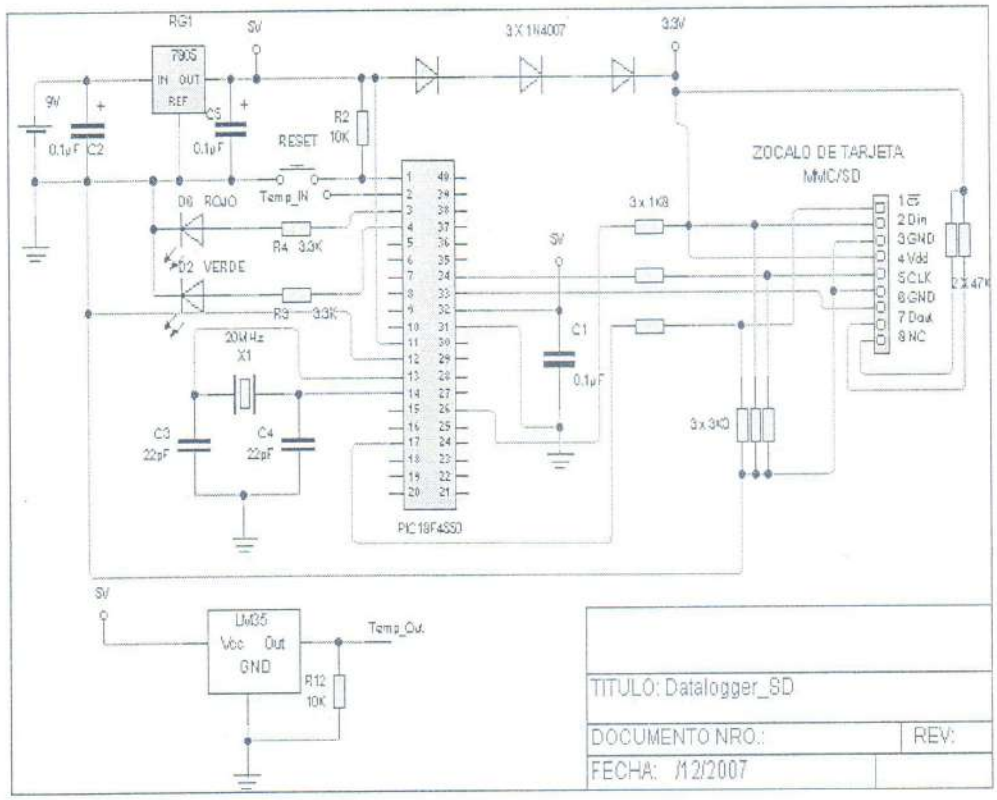

Figura 1.5. Diagrama circuital

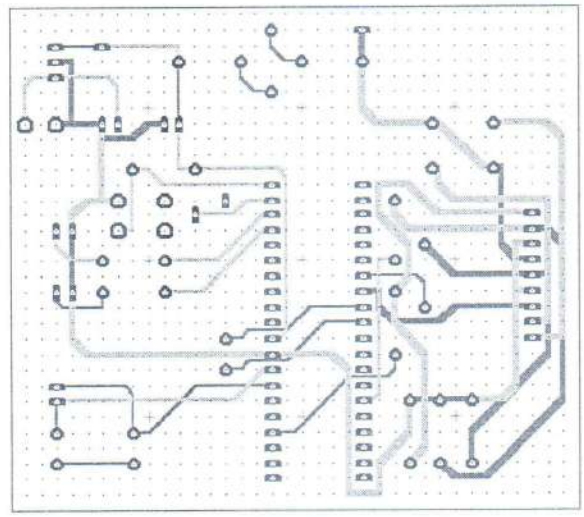

Figura 1.6. Diagrama plaqueta

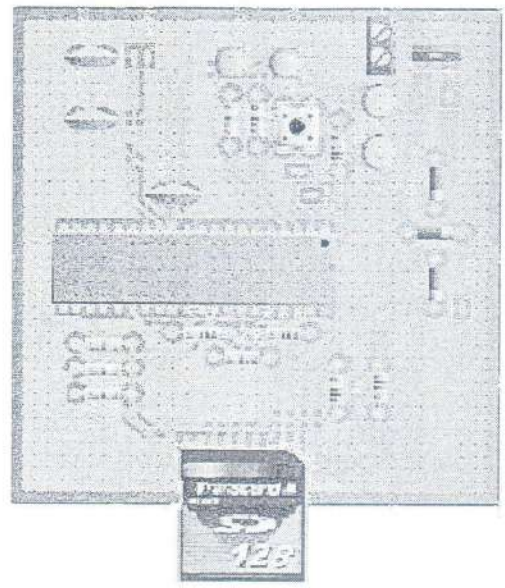

Figura 1.7. Diagrama de simulación topológica

El mismo consta de un microcontrolador PIC18F4550 con un oscilador de 20Mhz (luego se explicara la razón de esta frecuencia).Se colocan resistencias (divisor resistivo) de 1K8 y 3K3 Ω para "ajustar" los niveles de tensión que manejan las tarjetas SD, Micro SD y MMC (2.7 a 3.3 v) a los niveles de la lógica TTL de la arquitectura del PIC.

En la salida de la fuente regulada (78L05) de 5 V se coloca filtros de línea (capacitores), el motivo es que si se analiza en un osciloscopio, el PIC introduce un "ruido" de 50mVpp de 20Mhz debido al cristal, esta componente indeseada se introduce en la alimentación general ya que las masas están compartidas pudiendo llegar a afectar a las líneas del módulo SPI y en consecuencia a la memoria en los procesos de lectura y escritura.

Un uno lógico para la tarjeta será el límite alto de alimentación. La señal de reloj la entrega el micro por lo tanto debe ajustarse para que oscile entre 0 y 3.3v mediante un divisor de tensión, lo mismo se deberá hacer en la entrada de datos a la memoria, pero no es necesario hacerlo para la señal de salida de datos de la Micro SD ya que el PIC tiene en la entrada asociado un seguidor del tipo Smith Trigger, por lo tanto una tensión de 3.3v para el pic será un nivel alto.

Hay que tener en cuenta que a la hora de hacer el prototipo, si este se lo realiza en una protoboard, surgirán limitaciones, debido a las capacidades parásitas e inductancias indeseadas que se generan y deforman las señales a esta frecuencia, modificando los pulsos "rectangulares" del bus de comunicaciones.

Los 3.3V de alimentación de la tarjeta se obtuvieron a partir de los 5V del regulador 78L05 por medio de un arreglo de diodos 1N4007.

1.4.2. El Software.

Principalmente se deseó desarrollar un software que permita a sistemas basados en microcontroladores PIC de alta gama adquirir una señal externa y mediante la capacidad de crear archivos, almacenarla en una tarjeta de memoria MMC/SD.

El modulo principal del código (main.c) será el encargado de comunicar la tarjeta con el hardware, es decir, deberá implementar las funciones de más bajo nivel destinadas a manejar la tarjeta MMC/SD y el formato de archivos en FAT16.

La figura siguiente muestra los principales componentes del modulo principal.

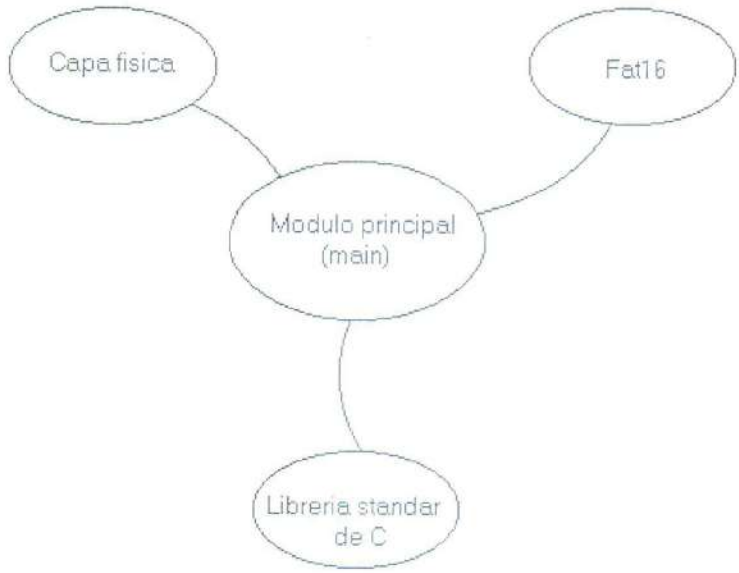

Figura 1.8. Componentes principales contenidos main.c

Una información mas detallada del funcionamiento del modulo principal así como de sus componentes puede verse en el capítulo 3.

Librerías de desarrollo.

Se ofrece al usuario la posibilidad de adaptar el sistema a sus necesidades (seteo de parámetros de adquisición, manejo de archivos, etc) y se pone a disposición una serie de librerías de C para el manejo de archivos (Ver figura 1.8) de modo que se facilite al programador esta tarea, sin requerir un conocimiento en detalle del sistema FAT16 o el funcionamiento de la tarjeta MMC/SD.

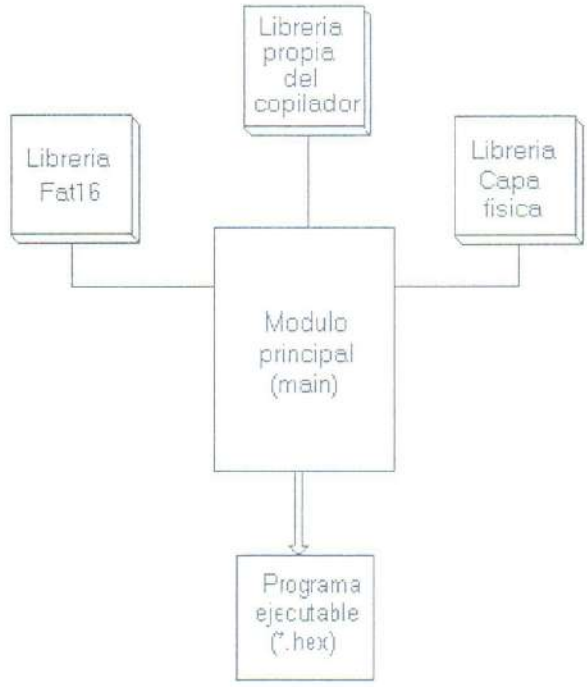

Figura 1.8. Manejo de las librerías

Características generales e integración del software.

Ya se puede disponer de un sistema funcional que permita al usuario realizar aplicaciones. Además, el programador deberá conocer ciertos aspectos generales a la hora de realizar los programas, como son la cantidad de memoria disponible para el programa, los periféricos que puede utilizar, o el manejo de las librerías. Por ello, el siguiente paso en la fase de diseño, consistirá en definir la estructura del software, la integración de los dos elementos anteriores y ciertos aspectos fundamentales de su funcionamiento.

Como se ha comentado, el modulo principal "main.c" es la parte del código en la que se encuentran las funciones básicas del sistema. Como cualquier código, ocupa una determinada cantidad de memoria y además tiene el control de determinados periféricos. La

cantidad de memoria ocupada por el código y la cantidad de memoria disponibles para los futuros programas de aplicación, sólo la podremos conocer tras el proceso de elaboración del mismo.

El último paso consistirá en desarrollar las librerías, de modo que puedan se utilizadas en el proceso de compilación. Así solo será añadido al modulo principal, el código de las funciones que sean usadas para la aplicación dada.

CAPÍTULO 2:

DESCRIPCIÓN DE LOS ELEMENTOS DEL SISTEMA

2.1. El PIC18F4550.

2.1.1. Características generales.

El PIC18F4550 es un microcontrolador de 8 bits que pertenece a la gama alta o avanzada (Enhanced, según Microchip) de microcontroladores PIC.

Posee una gran cantidad de memoria y puede trabajar a altas frecuencias. Además, tiene la capacidad de auto programar su memoria de código, mediante un determinado software, conocido comúnmente como cargador, o "bootloader" en inglés.

Estas tres características lo hacen especialmente adecuado para este proyecto, en el que se desea desarrollar un sistema de propósito general, que permita ejecutar distintos programas en el microcontrolador.

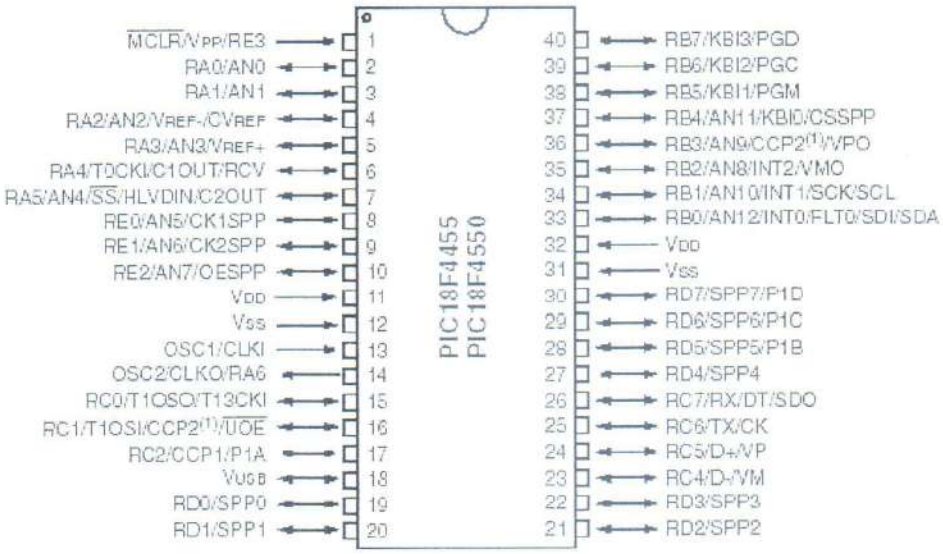

Figura 2.1. PinOut del PIC18F4550 en encapsulado PDIP.

A continuación se detallan las características más relevantes del microcontrolador. Para una información mas detallada, se ha de consultar el DataSheet del dispositivo que puede descargarse de la página del fabricante (www.microchip.com).

Recursos principales.

CARACTERISTICAS	PIC18F4550
Frecuencia de Operación	Hasta 48MHz
Memoria de Programa (bytes)	32.768
Memoria RAM de Datos (bytes)	2.048
Memoria EEPROM Datos (bytes)	256
Interrupciones	20
Líneas de E/S	35
Temporizadores	4
Módulos de Comparación/Captura/PWM (CCP)	1
Módulos de Comparación/Captura/PWM mejorado (ECCP)	1
Canales de Comunicación Serie	MSSP,EUSART
Canal USB	1
Puerto Paralelo de Transmisión de Datos (SPP)	1
Canales de Conversión A/D de 10 bits	13 Canales
Comparadores analógicos	2
Juego de instrucciones	75 (83 ext.)
Encapsulados	PDIP 40 pines QFN 40 pines TQFP 40 pines

Tabla 2.1. Principales recursos del PIC18F4550.

Características de la CPU.

-CPU de 8 bits.

-Capacidad de ejecutar hasta 10 millones de instrucciones por segundo (10 MIPS)

-Arquitectura RISC optimizada para compiladores de C. 75 instrucciones comunes a todos los PIC18, a las que se han añadido 8 nuevas instrucciones destinadas a optimizar el código.

-Multiplicador hardware de 8x8 bits que permite realizar multiplicaciones en una sola instrucción.

-Rango de tensión de funcionamiento: 2.0V-5.5V.

-Frecuencia máxima del cristal externo: 48Mhz.

-Multiplicador (PLL) x4 de la frecuencia de cristal externo.

Características de los periféricos.

-Manejo de alta corriente en los pines: 25mA.

-Dos comparadores.

-Dos salidas PWM

-Tres timers (uno de 8 bits, dos de 16 bits)

-Conversor analógico digital de 13 canales y 10 bits por canal.

-Módulo MSSP ("Master Synchronous Serial Port") para comunicación SPI™ e I^2C™.

-Módulo EUSART ("Enhanced Universal Synchronous Receiver Transmitter) con soporte para los estándar RS-232 y RS-485.

2.1.2. Memoria FLASH autoprogramable.

Existen ciertas consideraciones especiales a la hora de escribir o leer de la memoria FLASH, como por ejemplo que solo se puede escribir en bloques de 32 bytes de una sola vez, es decir no se puede escribir en ella un solo byte. Sin embargo, la lectura se realiza byte a byte. Todo el proceso de lectura, escritura y borrado de la memoria FLASH se encuentra detallado en el DataSheet del microcontrolador.

2.2. Sistema de archivos FAT16.

El sistema de archivos FAT (File Allocation Table o Tabla de Asignación de Ficheros) tiene su origen a finales de los 70's y principios de los 80's y era el sistema de archivos desarrollado para los sistemas operativos MS-DOS de Microsoft. Fue desarrollado inicialmente para ser usado en disquetes de menos de 500KB de capacidad. Con el tiempo, ha ido siendo revisado y mejorado para soportar mayores capacidades. Actualmente existen tres tipos de sistemas de archivos FAT: FAT12, FAT16 y FAT32. La diferencia básica entre estos tres tipos y la razón de sus nombres es el tamaño en bits de las entradas de la estructura FAT del disco. Hay 12 bits en una entrada FAT12, 16 en una entrada FAT16 y 32 en una entrada FAT32.

2.2.1. Características del sistema FAT

En informática, la información se almacena en forma de archivos. Los sistemas de archivos definen el modo en que esos archivos se almacenan en los sistemas de almacenamiento de modo que el acceso a ellos sea lo más eficiente posible, sin olvidar la seguridad etc... Un mismo disco o sistema de almacenamiento puede tener diversas particiones con distintos sistemas de archivos.

La mínima unidad de trabajo en sistemas de almacenamiento es el byte. Sin embargo, al ser un valor muy pequeño, los sistemas trabajan con una agrupación de éstos denominada sector. Para la mayoría de discos, un sector está formado por 512 bytes.

En el sistema FAT se trabaja con un concepto denominado clúster, que consiste en la mínima unidad de almacenamiento de archivos. El número de sectores que conforman un clúster en una determinada partición FAT es una potencia de 2 (1, 2, 4, 8, etc.). De este modo, en una partición FAT, la unidad mínima para almacenar archivos es el clúster, que en un disco de sector de 512 bytes puede ser de 512, 1024, 2048... bytes. En el proceso de formateo del disco, se asigna el tamaño de clúster mas adecuado para el tamaño de la partición. Esto quiere decir que los archivos se almacenan en porciones que constituyen los clústeres. Por ejemplo, si un archivo tiene un tamaño de 1 byte y se quiere almacenar en una partición FAT cuyo tamaño de clúster es de 2048 bytes, el archivo se colocará en un cluster libre, ocupando 1 byte y dejando los restantes 2047 sin que puedan ser utilizados por otro archivo. De este modo, un archivo siempre ocupa en disco, como mínimo, el tamaño de un clúster al ser almacenado en una partición FAT. Este es el principal inconveniente de los sistemas FAT que manejan gran cantidad de archivos de tamaño pequeño, la cantidad de espacio desaprovechado. Los archivos de tamaño superior a un clúster se almacenan en varios clústers, sin embargo, estos no tiene por qué ser consecutivos dentro del disco, sino que se irán almacenando a medida que el sistema operativo vaya encontrado clústeres vacíos. Esto constituye el segundo inconveniente fundamental de los sistemas FAT y se denomina fragmentación del disco, de modo que los archivos se encuentran fragmentados por todo el disco. Todos los clústeres están numerados en un partición FAT (0, 1, 2, 3, 4....), quedando el 0 y el 1 reservados, de modo que no existen físicamente en la zona de datos de la partición.

Una partición FAT está formada por tres estructuras fundamentales:

-La tabla FAT

-El BPB

-La zona de datos, donde se almacenan los archivos.

La tabla FAT es una estructura en la que todos los elementos tienen el mismo tamaño. Dicho tamaño es de 12 bits en FAT12, 16 bits en FAT16 y 32 bits en FAT32. Cada elemento corresponde con un clúster, de modo que el primer elemento corresponde al clúster 0 y así sucesivamente. El valor que almacena cada uno de los elementos de la tabla es el correspondiente al clúster siguiente ocupado por un archivo. Por ejemplo, si un archivo está almacenado ocupando el clúster 3 y 5, en la entrada de la tabla FAT correspondiente al clúster 3 (la cuarta en orden creciente) estará almacenado el valor "5" y la entrada correspondiente al clúster 5 contendrá un valor correspondiente a final de fichero (por ejemplo, 0xFFFF en FAT16). De este modo, sabiendo que el archivo comienza en el clúster 3, lecturas sucesivas de la tabla FAT determinan todos los clústers ocupados por el archivo.

El BPB (BIOS Parameter Block), como su nombre indica, guarda todos los parámetros relativos a la partición FAT, como pueden ser el número de sectores que componen un clúster, el tamaño en bytes de un sector, el tamaño en sectores de la partición y muchos mas. Todo ello puede observarse en la tabla 2.2. Son valores necesarios para que el sistema operativo o el BIOS o el software correspondiente puedan leer correctamente de la partición FAT.

La zona de datos de la partición es la zona destinada a almacenar los archivos y queda dividida en los distintos clústeres que componen la partición.

2.2.2. El BPB (BIOS Parameter Block)

El BPB es una estructura que guarda información acerca de la partición FAT correspondiente. Está situado a partir del primer sector de la partición FAT (llamado "sector 0"). El número de sectores ocupados por el BPB se denominan "sectores reservados". Normalmente, en un disco de sector de tamaño 512 bytes, el BPB cabe en el primer sector de la partición. A continuación se muestra una tabla con los campos del BPB más importantes y útiles para este proyecto:

Nombre	Offset (byte)	Tamaño (bytes)	Descripción
BPB_BytsPerSec	11	2	Número de bytes que componen un sector.
BPB_SecPerClus	13	1	Número de sectores que componen un clúster.
BPB_RsvdSecCnt	14	2	Número de "sectores reservados" a partir del primer sector de la partición. En FAT12 y FAT16 es igual a 1.
BPB_NumFATs	16	1	Número de tablas FAT presentes. Normalmente suelen ser 2, sirviendo la segunda como copia de seguridad.
BPB_RootEntCnt	17	2	Número de entradas del directorio raíz. Normalmente 512, que dicta el máximo número de archivos y directorios que se pueden almacenar en el directorio raíz.
BPB_TotSec16	19	2	Número de sectores ocupados por la partición. Sólo valido para FAT16 y FAT12. Para FAT32 existe otro campo llamado BPB_TotSec32 de tamaño 4 bytes.

BPB_FATSz16	22	2	Número de sectores ocupados por cada una de las tablas FAT. Solamente válidos para FAT12 y FAT16. Para FAT32 existe el campo BPB_FATSz32.

Tabla 2.2. Estructura del BPB

Todo sistema que deba leer de un disco con una partición FAT debe tener en cuenta esta serie de parámetros en primer lugar.

2.2.3. La tabla FAT

La tabla FAT es una estructura que se sitúa en el primer sector a continuación de los sectores reservados de la partición FAT (donde está el BPB). En FAT16 se sitúa en el segundo sector (a continuación del "Sector 0"). Como se ha comentado, el tamaño de cada elemento de la tabla es fijo y depende de la versión FAT. Para FAT12 es de 12bits, para FAT16 es de 16 bits y para FAT32 es de 32 bits. Cada elemento corresponde a un clúster de la partición y el valor que guarda cada elemento se corresponde con el número de clúster con el que enlaza el anterior.

De aquí se puede extraer otra conclusión acerca de los sistemas FAT. De entre ellos, el que nos interesa para el proyecto es el FAT16. Como el tamaño de cada elemento de la tabla es de 16bits, el número máximo de clústeres que se pueden direccionar es de 2^{16}=65536, que viene a ser el número máximo de archivos que puede almacenar una partición FAT16, ya que cada archivo ocupará como mínimo un clúster. Otra conclusión que se puede extraer de esto, es que los sistemas FAT12 y FAT16 no son muy adecuados para particiones de gran tamaño. Por ejemplo, sea un disco de 4GB, al convertirlo a bytes tenemos 4.292.967.296 bytes. Para FAT16 el número máximo posible clústeres es de 65536, por tanto al dividir el tamaño total del disco por este valor, obtenemos que el tamaño mínimo del clúster es de 65536 bytes. De este modo, la cantidad de espacio desperdiciada es mayor al almacenar archivos de poco tamaño. Para ello surgió FAT32 que puede direccionar hasta 2^{32} = 4294967296 clústers. El inconveniente de esto es que la tabla FAT es mucho mayor, lo que hace que la búsqueda en la tabla se haga mucho más lenta por el sistema. Es por ello que la opción mas acertada para nuestro proyecto, en el que usaremos tarjeta de memoria SD, es el uso de FAT16, que ofrece la mejor relación de velocidad y aprovechamiento del espacio.

Cada elemento de la tabla puede almacenar tres tipos de valores, uno para indicar que el clúster correspondiente está vacío; otro para indicar que el cluster está ocupado por un archivo, pero es el último clúster ocupado por dicho archivo; y otro para indicar que el clúster

está ocupado por un archivo, y el valor que guarda corresponde con el siguiente clúster ocupado por el archivo.

Valor	Descripción
0 (0x00h)	El clúster correspondiente está vacío (no utilizado por ningún archivo)
65535 (0xFFFFh)	El clúster está ocupado por un determinado archivo y es el último ocupado por dicho archivo
XXXXX	Siendo XXXXX cualquier valor entre 1 y 65534. El clúster está ocupado por un archivo. El valor XXXXX indica el siguiente clúster ocupado por el archivo, por tanto, para poder seguir leyendo dicho archivo

Tabla 2.3. Posibles valores de cada elemento de una tabla FAT16

2.2.4. La zona de datos

Es la zona de la partición donde se almacenan los archivos. Todos los archivos están contenidos en "directorios". Un directorio es un tipo de archivo especial que almacena la información correspondiente a cada uno de los archivos que contiene, tal como el nombre de dicho archivo, el tamaño, si es un archivo normal u otro directorio, etc.. De este modo, el software correspondiente tendrá la información necesaria para leer un archivo determinado. Existe además un directorio especial denominado directorio "raíz" que es el directorio principal de la partición en el que están contenidos todos los demás. Para FAT16 el número máximo de archivos (siendo éstos archivos o directorios) que puede contener el directorio raíz es de 512 (Ver BPB_Root_EntCnt en tabla 2.2).

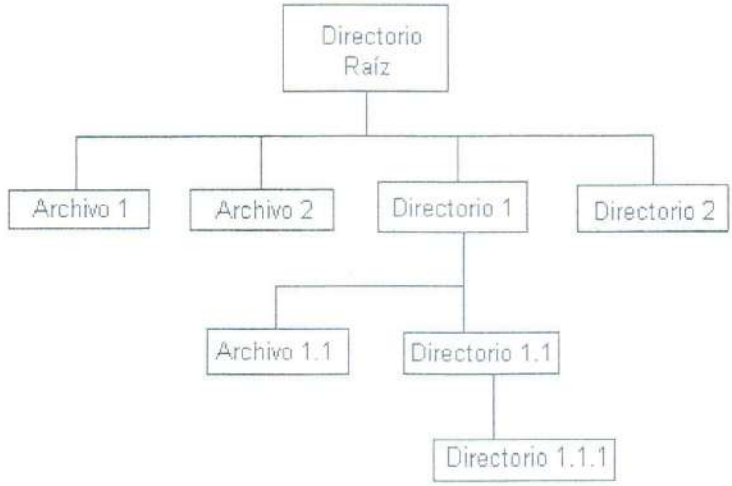

Figura 2.2. Jerarquía de directorios en una partición FAT

2.2.4.1. Estructuras de directorios FAT

Un directorio no es más que un archivo compuesto de una lista lineal de estructuras de 32 bytes. El único directorio especial que está siempre presente en una partición FAT, es el directorio raíz. En FAT16, el directorio raíz está colocado en una posición fija inmediatamente después de la última tabla FAT y tiene un tamaño fijo de 16384 bytes (512 entradas de 32 bytes). Su tratamiento es algo diferente al resto de archivos o directorios de la partición, ya que no se puede borrar, no tiene nombre, sus bytes se encuentran de forma consecutiva, ya que no se almacena en la zona dividida por clústeres, no tiene fecha ni hora, y tiene un tamaño máximo. Inmediatamente después del directorio raíz se sitúa la zona de datos destinada a almacenar todos los archivos y directorios de la partición, y ahí comienza la zona divida en clústeres, a cada cual le corresponde una entrada de la tabla FAT. Así el primer clúster usable es el "2" (ya que tanto el 0 como el 1 se encuentran reservados).

Como se ha explicado, un directorio es un archivo que contiene información necesaria para leer cada uno de los archivos o subdirectorios que contiene. La información de cada uno de ellos es una estructura de 32 bytes, cuyos campos pueden verse en la siguiente tabla:

Nombre	Offset (byte)	Tamaño (bytes)	Descripción
DIR_Name	0	11	Nombre corto del archivo o directorio
DIR_Attr	11	1	Atributos del archivo o directorio: 0x01 Sólo lectura

			0x02 Oculto 0x04 Archivo de sistema 0x08 ID_Volume 0x10 Es un directorio 0x20 Es un archivo
DIR_NTRes	12	1	Para uso excluivo de Windows NT. Poner a 0, al crear el archivo
DIR_CrtTimeTenth	13	1	Décimas de segundo de cuando el archivo fue creado
DIR_CrtTime	14	2	Hora a la que el archivo fue creado
DIR_CrtDate	16	2	Fecha en que el archivo fue creado
DIR_LstAccDate	18	2	Fecha del último acceso al archivo (lectura o escritura)
DIR_FstClusHI	20	2	Palabra alta del número del primer clúster ocupado por el archivo. Vale 0x0000 en FAT12 y FAT16, ya que el número máximo de clústeres en estos casos cabe en 2 bytes
DIR_WrtTime	22	2	Hora de la última escritura en el archivo
DIR_WrtDate	24	2	Fecha de la última escritura en el archivo
DIR_FstClusLO	26	2	Palabra baja del primer clúster ocupado por el archivo
DIR_FileSize	28	4	Tamaño del archivo en bytes

Tabla 2.4. Estructuras de 32 bytes de entradas de directorios

Nombres en FAT

Como se puede observar en la tabla, para el nombre de un archivo están reservados 11 bytes, de los cuales 8 se utilizan para el nombre en sí y 3 para la extensión del archivo. En caso de tratarse de un directorio, los 11 bytes se pueden utilizar para el nombre. Existe una mejora para esta limitación que permite el uso de nombres largos de hasta 256 caracteres, sin embargo, por la limitación de memoria que tenemos no será implementado en nuestro sistema, con lo cual obviaremos su explicación. Aún así, todo archivo con nombre largo tiene un nombre corto de 11 caracteres con el que poder ser accedido.

El primer byte de cada una de las entradas (DIR_name[0]) sirve para determinar si la entrada está libre (no hay archivo), si hay algún archivo ocupándola o si ya no existen mas archivos dentro del directorio actual. Los valores posibles para ello, son:

- Si DIR_Name[0]==0xE5 ->La entrada del directorio está libre, no hay archivo, o ha sido borrado el que la ocupaba

- La entrada punto-punto se utiliza para conocer el comienzo del directorio que contiene al directorio actual

Formatos de hora y fecha

Existe un formato específico para la hora y la fecha utilizada en los distintos campos de la estructura. Tanto DIR_WrtTime como DIR_WrtDate son campos que todo sistema debe modificar al tratar con archivos. El resto son opcionales, y solo son utilizados en algunos sistemas.

Formato de la fecha:
Bits 0-4: Día de la semana, rango válido de 1-31
Bits 5-8: Mes del año, rango válido 1-12
Bits 9-15: Año, rango válido 0-127, correspondiendo el 0 con 1980

Formato de la hora:
Bits 0-4: Cuenta de "2 segundos", rango válido 0-29 (0-58 segundos)
Bits 5-8: Minutos, rango válido 0-59
Bits 11-15 Horas, rango válido 0-23

2.3. TARJETA MMC/SD

2.3.1. Introducción

Se conoce por MultiMediaCard a un estándar de comunicación y almacenamiento de datos avanzado. Su definición engloba a todo un sistema compuesto por controladores, sistemas de almacenamiento (tarjetas), y un bus de comunicación entre ambos, lo cual en conjunto se denomina "Sistema MultiMediaCard". Esas tarjetas de almacenamiento pueden ser de solo lectura, de lectura/escritura o de entrada/salida. De entre todas ellas, el objeto de nuestro estudio se centra en un tipo de tarjetas de memoria flash de lectura/escritura denominada "MMC/CD". Es importante distinguir entre "Sistema MultiMediaCard" y "Tarjeta MMC".

La tarjeta MMC/SD consiste en una solución de bajo coste para almacenamiento de datos. Está diseñada como medio de almacenamiento para una amplia área de aplicaciones, tales como telefonía móvil, cámaras fotográficas, reproductores de MP3, grabadoras digitales, etc... Se caracterizan por un bajo consumo así como una alta transferencia de datos. La comunicación de la MMC/SD está basada en un avanzado bus. El protocolo de comunicación

está definido como una parte del estándar MMC y es denominado como el "Modo MMC". Sin embargo y para asegurar compatibilidad con la mayoría de controladores existentes, la tarjeta MMC/SD ofrece un modo de comunicación alternativo basado en el estándar SPI.

Actualmente la tarjeta MMC está tendiendo a desaparecer y a ser renovada por las tarjetas SD (Secure Digital) que están basadas en las primeras pero ofrecen características más avanzadas tales como una mayor velocidad de transferencia de datos, protección contra escritura, mayores capacidades, etc... Sin embargo todas estas versiones están basadas en el estándar MMC, incluidas las SD (que tienen su propio estándar pero basado en el MMC), y cada una de ellas posee unas determinadas características avanzadas. Por ello hay que diferenciar entre el "estándar MultiMediaCard" y la "tarjeta MMC".

En la siguiente figura se ve la evolución de algunos tipos de tarjetas existentes a partir de las MMC originales.

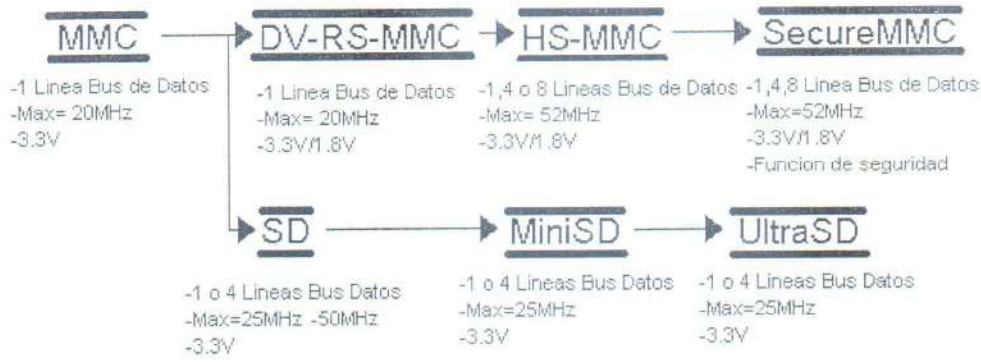

Figura 2.3. Evolución de distintos tipos de tarjetas a partir de la tarjeta MMC

Cabe destacar que cada versión guarda total compatibilidad con la anterior de modo que se pueda garantizar el uso de tarjetas mas avanzadas en sistemas que fueron desarrollados para tarjetas mas antiguas.

Dicho esto, se debe puntualizar que a continuación se hará una breve descripción de la tarjeta MMC/SD convencional y de su funcionamiento, la cual es usada en nuestro diseño, y no una descripción total del estándar MMC que se encuentra en constante evolución y mejora y engloba a todas las versiones existentes de tarjetas (de tamaños, patillajes y desempeño distinto), además de contener una detallada explicación del bus de comunicación MMC. Por ello hay que tener muy en cuenta que lo que se pretende detallar a continuación es la parte del estándar MMC necesario para poder realizar una comunicación con una tarjeta flash MMC/SD convencional.

2.3.2. Características y funcionamiento de la tarjeta MMC/SD.

Las principales características de una tarjeta MMC convencional son:

-Rango de voltaje de operación: 2.7-3.6V

-Rango de frecuencias de reloj: 0-20MHz para MMC y 0-25MHz para SD

-Tamaños: Tamaño normal: 24mm x 32mm x 1.4 mm) y Tamaño reducido (24mm x 18mm x 1.4mm)

-Soporte para dos modos de funcionamiento: MMC (MultiMediaCard) y SPI

-Consumo de corriente: Depende del modo en que se encuentre.

Sleep 250 [μA]

Read 65 [mA]

Write 75 [mA]

Modo MultiMediaCard*	Modo SPI
Bus serie de tres señales(reloj, comando y datos)	Bus serie de tres señales(reloj, dataIn y dataOut)
Capacidad de direccionar hasta 64K tarjetas mediante el protocolo	Selección de la tarjeta mediante las señal hardware CS
Se pueden apilar hasta 30 tarjetas en un solo bus físico	Para apilar tarjetas se requiere una señal CS por tarjeta
Fácil identificación de la tarjeta en el bus	No disponible la identificación
Protección de error en la transferencia de datos	Opcional. Hay disponible un modo de transferencia sin protección de error.
Soporta comandos de escritura/lectura secuencial y de simples o múltiples bloques	Soporta comandos de escritura/lectura de simples/multiples bloques

Tabla 2.5. Diferencias entre los modos de funcionamiento

MMC y SPI de una tarjeta MMC convencional

PIN	Nombre	Tipo	Descripción
1	CD/DAT3/RSV	I/O	Detección de tarjeta/ Línea de datos
2	CMD	I/O/PP/OD	Comando/Respuesta
3	VSS1	S	GND
4	VDD	S	V+
5	CLK	I	Clock
6	VSS2	S	GND

7	DAT0	I/O/PP	Línea de datos
8	DAT1	I/O	Línea de datos
9	DAT2	I/O	Línea de datos

Tabla 2.6. Pines en modo MMC

PIN	Nombre	Tipo	Descripción
1	CS	I	Chip Select (activo en bajo)
2	DATAIN	I	Comandos y Datos desde el HOST
3	VSS1	S	GND
4	Vcc	S	V+
5	CLK	I	Clock
6	VSS2	S	GND
7	DATAOUT	O/PP	Datos hacia el HOST
8	RSV	I	Reservado
9	RSV	I	Reservado

Tabla 2.7. Patillaje en modo SPI

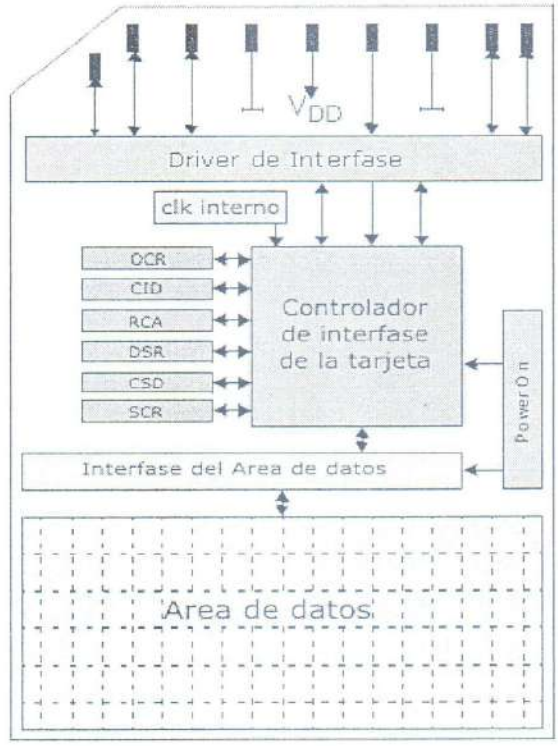

Figura 2.4. Diagrama de bloques de una tarjeta SD

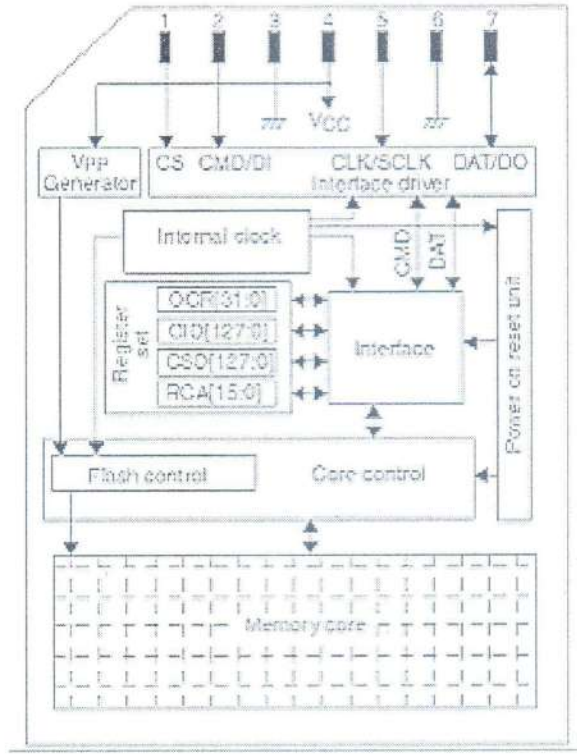

Figura 2.4. Diagrama de bloques de una tarjeta MMC

Por defecto la tarjeta trabaja en el modo MMC. Es el modo en el que se obtiene mejor rendimiento pero implica el uso de controladores que soporten dicho protocolo. El modo SPI está soportado por la mayoría de microcontroladores del mercado por lo que se ofrece como alternativa y será el modo en el que trabaje nuestro sistema. Independientemente del modo de operación en un sistema donde se comunique un controlador con una o varias tarjetas MMC/SD se pueden distinguir tres tipos de estructuras:

-Comando: Es una estructura mediante la cual el controlador indica a la tarjetas o tarjetas que se va a realizar una operación (entendiéndose por operación cualquier evento que se vaya a realizar en la tarjeta, sea leer datos, escribir datos, leer de sus registros internos, escribir en ellos, etc...)

-Respuesta: La respuesta es una estructura que la tarjeta o tarjetas seleccionadas envían al controlador en respuesta a un comando previamente recibido.

-Datos: Los datos son estructuras que pueden ser transmitidos del controlador a la tarjeta y viceversa, cuando una determinada operación lo requiera.

El modo escogido dicta como se ha de realizar la transacción de esas estructuras, y el formato de ellas. Así, por ejemplo, en el modo MMC, mediante la línea CMD se envían comandos a la tarjeta, ésta responde mediante la línea CMD también y en caso de ser una

operación donde se trasmiten datos, estos son enviados en los dos sentidos por la línea DAT. En el modo SPI cada estructura enviada por el controlador a la tarjeta (sean datos o comandos) se realiza a través de la línea "Data In" y cada estructura enviada de la tarjeta al controlador (sean datos o respuestas) se realiza a través de la línea "Data Out". La comunicación en el modo MMC es orientada a bits mientras que en el modo SPI es orientada a bytes, de modo que en el primero, las estructuras se entienden como un flujo continuo de bits, mientras en el segundo cada estructura se ha de entender como un flujo continuo de bytes de 8 bits.

El formato y tamaño de los comandos y los datos son iguales en ambos modos de funcionamiento, sin embargo el comportamiento de la respuesta difiere del modo SPI al modo MMC en varios aspectos:

-En modo SPI, la tarjeta siempre responde al controlador tras recibir un comando a diferencia del modo MMC.

-En el modo SPI existen varias estructuras de respuesta diferentes a las usadas en el modo MMC

-En modo SPI cuando la tarjeta detecta un problema en la recepción de un bloque de datos responde con una respuesta especifica de error, (además de la respuesta al comando) sin embargo en el modo MMC un error es indicado mediante un "time-out".

2.3.3. Registros internos de una tarjeta MMC/SD

Las tarjetas MMC convencionales poseen 5 registros de propósito general, que pueden ser accedidos mediante unos determinados comandos.

-OCR ("Operation Conditions Register"): Es un registro de 32 bits. Almacena los valores de tensión de operación que soporta la tarjeta. Posee un bit de estado que es seteado tras un *power-up*. Es obligado que todas las tarjetas posean este registro para cumplir el estándar MMC.

-CID ("Card Identification register"): Es un registro de 16 bits. Contiene la información necesaria para poder identificar una tarjeta. Cada tarjeta posee un único número que la identifica del resto. Es un registro obligado por el estándar.

Cave destacar que la estructura de este registro en una tarjeta SD es diferente a la estructura en una tarjeta MMC.

Nombre	Tipo	Tamaño	N° de Bit	Comentario	CID Value
Manufacturer ID (MID)	Binario	8	[127:120]	El ID del fabricante es asignado y controlado por la asociación SD Card.	0x03
OEM/Application ID (OID)	ASCII	16	[119:104]	El OID es asignado por las 3C.[1]	SD ASCII Code 0x53, 0x44
Product Name (PNM)	ASCII	40	[103:64]	5 caracteres ASCII	SD128, SD064
Product Revision[2] (PRV)	BCD	8	[63:56]	Dos dígitos BCD	3.0
Serial Number (PSN)	Binario	32	[55:24]	Entero de 32 Bits	Numero de serie
Reserved		4	[23:20]		
Manufacture Date Code (MDT)	BCD	12	[19:8]	Fecha de fabricación –yym (desde el 2000)	Apr 2001 = 0x014
CRC7 checksum[3] (CRC)	Binario	7	[7:1]	Calculado	CRC7
Not used, always '1'		1	[0:0]		

[1] 3C = Las tres compañías fundadoras de la Asociación SD: Toshiba, SanDisk, y MEI.

[2] La revisión de producto esta compuesta de dos dígitos BCD "n.m" (numero de revisión). Por ejemplo los números para una revisión 6.2 serian: 0110 0010.

[3] El CRC se calcula con la siguiente formula:
CRC: G(x)=x7+3+1
M(x)=(MID-MSB)*x119+...+(CIN-LSB)*x0
CRC[6...0]=Remainder[(M(x)*x7)/G(x)]

-CSD ("Card-Specific Data"): Es un registro de 128 bits. Proporciona información de cómo acceder a los contenidos de la tarjeta. Define el formato de los datos, el tipo de corrección de errores, el tiempo máximo de acceso, la velocidad de transferencia de datos, si el registro DSR puede ser usado, etc... Es un registro que debe ser implementado obligatoriamente en toda tarjeta.

-RCA (Relative Card Address): Registro de 16-bits. Es un registro escrito por el controlador en la inicialización, en el cual almacena la dirección relativa que la tarjeta ocupa en el sistema. Su valor por defecto es de 0x0001. Es un registro de implementación obligatoria.

-DSR (Driver Stage Register): Registro de 16 bits. Es un registro opcional que puede ser usado para mejorar el funcionamiento del bus mediante condiciones de operación extendidas (dependiendo del tamaño del bus, de la transferencia de datos o el número de tarjetas). La información sobre el uso del registro DSR es almacenada en el registro CSD. El valor por defecto del registro DSR es 0x404.

2.3.4. Modo MMC

En el modo MMC la línea CMD se usa para transmitir tanto los comandos como las respuestas. La línea DAT se usa para la transferencia de bloques de datos entre el controlador y la tarjeta. Los comandos son enviados desde el controlador a la tarjeta y la respuesta es enviada desde la tarjeta hasta el controlador. Los datos son enviados desde la tarjeta al controlador en las operaciones de lectura y viceversa en las operaciones de escritura.

Figura 2.5. Operación de lectura de la tarjeta en modo MMC

Figura 2.6. Operación de escritura en la tarjeta en modo MMC

2.3.5. Modo SPI

El modo SPI es un modo de operación de la tarjeta alternativo, desarrollado para obtener mayor compatibilidad con los controladores existentes en el mercado. Es el escogido para nuestro sistema debido a que la mayoría de microcontroladores PIC poseen un módulo de comunicación SPI. Por defecto, tras un *power-up* la tarjeta MMC se sitúa en el modo de operación MMC, por tanto será nuestro sistema el que deberá inicializar la tarjeta en modo SPI.

Mientras en el modo MMC las líneas CMD y DAT eran bidireccionales, las líneas DataIn y DataOut del modo SPI son unidireccionales. Por tanto los comandos y los datos

desde el dispositivo controlador hacia la tarjeta son enviados a través de la línea DataIn y las respuestas y los datos desde la tarjeta hacia el dispositivo controlador son enviados a través de la línea DataOut.

Figura 2.7. Operación de lectura de la tarjeta en modo SPI

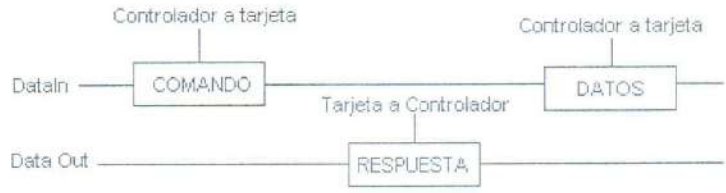

Figura 2.8. Operación de escritura en la tarjeta en modo SPI

2.3.5.1. Protocolo SPI

SPI (Serial Peripherical Interface) es un protocolo de comunicación serie mediante el cual se transmiten paquetes de 8 bits entre un dispositivo maestro y un esclavo. Es muy importante tener en cuenta que el protocolo SPI únicamente define la interfase o bus de comunicación y no establece nada acerca de la estructura de la información intercambiada entre maestro y esclavo, salvo que debe estar formada por paquetes de 8 bits (1 byte) por lo que es más correcto denominarlo "interfaz SPI" en lugar de "protocolo SPI".

La comunicación es full-dúplex, de modo que cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo. El bus de comunicación SPI está compuesto por 4 líneas, SCLK, MOSI, MISO, y /CS o /SS que son unidireccionales, de manera que la información por cada línea solo se transmite en una dirección.

-SCLK (Serial Clock): es la línea que transmite la señal de reloj. Ésta es generada por el maestro y sincroniza la transferencia de datos, por lo que también se dice que la comunicación SPI es una comunicación síncrona.

-MOSI (Master Out Slave In): transporta los datos desde el maestro al esclavo

-MISO (Master In Slave Out): transporta los datos desde el esclavo al maestro

-/CS (Chip Slect) ó /SS (Slave Select): Es la línea mediante la cual el maestro selecciona o activa al esclavo para comunicarse con el. Al ser señales negadas la activación se realiza manteniendo un nivel lógico '0'. Al existir varios esclavos, cada uno tendrá su línea /CS y el maestro deberá activar sólo aquel con el que quiera comunicarse.

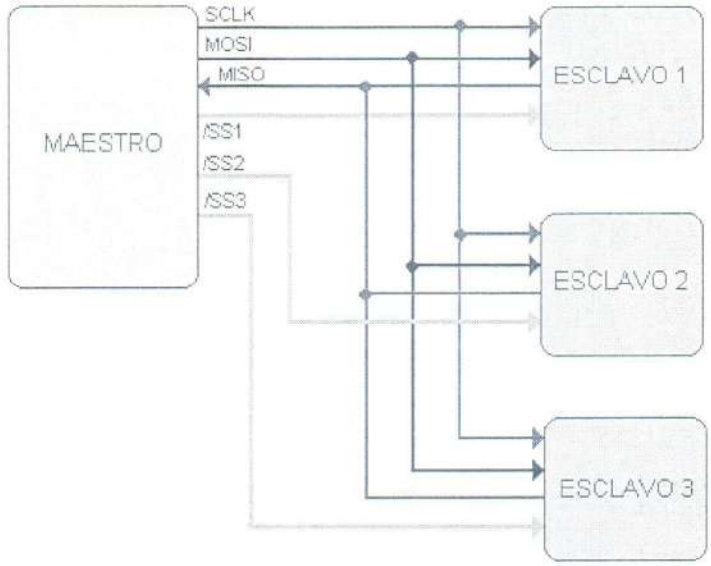

Figura 2.9. Ejemplo de conexión del bus SPI con varios esclavos

Una comunicación se establece cuando el maestro selecciona a un esclavo activando la señal /SS que le corresponde. La comunicación entre ambos se realiza a través de un simple registro de desplazamiento existente en cada uno de ellos. Cuando el maestro selecciona al esclavo para comenzar una transmisión, tanto maestro como esclavo colocan en su registro de desplazamiento el byte que deben transmitir, en caso de que deban transmitir alguno, ya que la operación solicitada por el maestro puede ser para recibir únicamente del esclavo, para enviar únicamente o para recibir a la vez que envía. De todas formas, en una transmisión siempre se envía un byte del esclavo al maestro y viceversa, ya que un registro no puede estar vacío (el hecho de contener el byte '00000000' no quiere decir que esté vacío). Por ello bastará con que el maestro no procese el dato recibido por el esclavo si no lo esperaba y que envía un byte fantasma (byte que no genere ninguna acción en el esclavo) si solamente necesitaba recibir un paquete (byte) del esclavo.

La señal de reloj permanece en estado de reposo hasta que el maestro tenga preparado el byte a transmitir, momento en el que genera 8 ciclos de reloj de manera que en cada ciclo

se trasmita un bit a través de los registro de desplazamiento (se transmite un bit del maestro al esclavo a la vez que se transmite un bit del esclavo al maestro). Tras los 8 ciclos de reloj se habrá transmitido un byte del maestro al esclavo y viceversa.

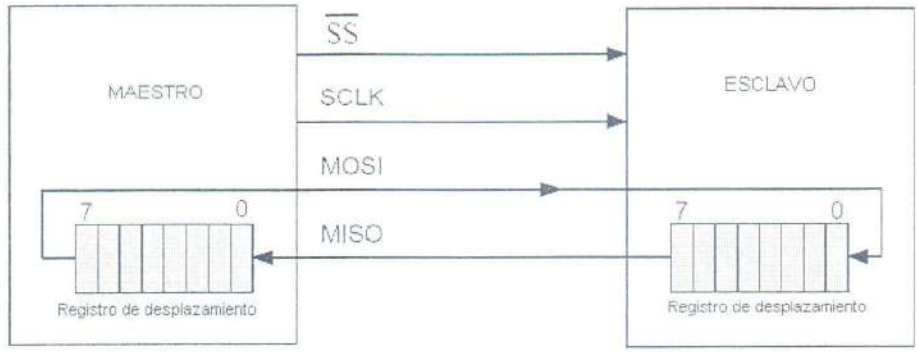

Figura 2.10. Comunicación maestro-esclavo en un bus SPI

Para comunicar dos dispositivos mediante una interfaz SPI se deben tener en cuenta las características del módulo SPI en cada uno de los dispositivos de manera que por ejemplo se debe conocer el tiempo que emplea el esclavo en procesar el byte recibido para que el maestro pueda iniciar otro intercambio de bytes. Es por ello por lo que en el estándar SPI sólo se define la interfaz o capa física.

2.3.5.2. Comandos de la tarjeta en modo SPI

Todos los comandos tienen un tamaño de 48 bits (6 bytes) y tiene la misma estructura independientemente del modo de funcionamiento de la tarjeta. En la siguiente figura se muestra el formato de un comando:

Posición de los bits	Número de bits	Valor	Descripción
47	1	'0'	Bit de inicio
46	1	'1'	Bit de transmisión
[45:40]	6	X	Índice del comando
[39:8]	32	X	Argumento
[7:1]	7	X	CRC7
0	1	'1'	Bit de final

Tabla 2.8. Formato de un comando

Una tarjeta MMC/SD convencional soporta hasta 60 comandos distintos. Estos a su vez se dividen en clases según la operación que realicen. No todas las clases son soportadas en el modo SPI a diferencia del modo MMC. Es por ello que a continuación solo se detallarán los comandos más importantes para el modo SPI y aquellos que pueden ser útiles para nuestro trabajo. Como ya se ha dicho, en modo SPI, la tarjeta siempre responde a un comando. Además en este modo existen tres tipos de respuestas a diferencia del modo MMC en que solo hay un tipo. En el siguiente apartado se explicará el formato de las respuestas.

Índice del comando y nombre abreviado	Respuesta	Argumento	Descripción
CMD0 GO_IDLE_STATE	R1	-	Resetea la tarjeta
CMD1 SEND_OP_COND	R1	-	Activa el proceso de inicialización de la tarjeta
CMD9 SEND_CSD	R1	-	Pide a la tarjeta el contenido del registro CSD
CMD10 SEND_CID	R1	-	Pide a la tarjeta el contenido del registro CID
CMD12 STOP_TRANSMISSION	R1b	-	Fuerza a la tarjeta a parar la transmisión en una operación de lectura múltiple de bloques
CMD13 SEND_STATUS	R2	-	Pide a la tarjeta el contenido de su registro de estado
CMD16 SET_BLOCKLEN	R1	[31:0] tamaño del bloque	Selecciona un tamaño para bloque (en bytes) para todos los siguientes comandos
CMD17 READ_SINGLE_BLOCK	R1	[31:0] dirección	Lee un bloque del tamaño indicado por SET_BLOCKLEN a partir de la dirección indicada
CMD18 READ_MULTIPLE_BLOCK	R1	[31:0] dirección	Lee bloques de datos continuamente desde la dirección indicada hasta que reciba el comando CMD12
CMD24 WRITE_BLOCK	R1	[31:0] dirección	Escribe un bloque de datos del tamaño seleccionado por SET_BLOCKLEN
CMD25 WRITE_MULTIPLE_BLOCK	R1	[31:0] dirección	Escribe bloques continuamente hasta recibir el final de transmisión
CMD27 PROGRAM_CSD	R1	-	Programación de los bits programables del registro CSD
CMD58 READ_OCR	R3	-	Lee el registro OCR de la tarjeta
CMD59 CRC_ON_OFF	R1	[0:0]	Activa o desactiva la opción CRC. '1':activada, '0': Desactivada

Tabla 2.9. Principales comandos usados en el modo SPI

*Formato R1b

La respuesta es idéntica a la R1 con la adición de la señal opcional de "ocupado" (busy) de la tarjeta. Esta señal puede consistir en un número cualquiera de bytes. El valor de '0' indica que la tarjeta está ocupada mientras que cualquier otro valor indica que la tarjeta está preparada para recibir el siguiente comando.

Formato R2

El tamaño de la respuesta R2 es de 16 bits y es enviada como respuesta al comando CMD13. El primer byte consiste en la respuesta R1 y los siguientes 8 bits indican otro tipo de errores tales como errores generales de la tarjeta, protección contra escritura, etc...

Formato R3

El tamaño de la respuesta R3 es de 40 bits y es enviada como respuesta al comando CMD58. El primer byte consiste en la respuesta R1 y los 4 bytes siguientes contienen el registro OCR.

Cualquier bit indicador de error es puesto a '0' automáticamente con la lectura de las distintas respuestas por parte del controlador (sean parte de R1, R2 ó R3).

2.3.5.4. Transacción de datos en la tarjeta en modo SPI

Todos los comandos tanto de escritura como de lectura tienen una transferencia de datos asociada a ellos. Todos los bytes que componen una estructura de datos son enviados con el MSB (bit mas significativo) primero. Nos referimos a estructura y no a bloque porque en una transacción de datos existirá tanto el bloque de datos en sí como el CRC así como otra estructura de "comienzo de bloque". Esta estructura tiene el tamaño de un byte y tiene el siguiente formato:

Tipo de operación	Valor del byte de comienzo
Lectura simple de bloques, escritura simple de bloques y lectura múltiple de bloques	11111110
Escritura múltiple de bloques cuando los datos van a ser transmitidos	11111100
Escritura múltiple de bloques cuando se requiere la finalización de la transmisión. Es más correcto denominar al byte de comienzo como byte de parada de la transmisión	11111101

Tabla 2.10. Tipos de comienzo de bloque

- Si DIR_Name[0]==0x00 ->La entrada del directorio está libre y es la última entrada del directorio. Con lo cual el software correspondiente no debe seguir buscando más entradas dentro del directorio
- Si DIR_Name[0]=0xXX ->Distinto a los anteriores, la entrada está ocupada por un archivo cuyo primer carácter del nombre corresponde con el valor 0xXX.

Existen varias limitaciones para el uso de nombres de archivos:

- EL primer carácter no puede ser "." (0x20)
- No se permiten caracteres con valores menores de 0x20
- Los caracteres x22, 0x2A, 0x2B, 0x2C, 0x2E, 0x2F, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x5B, 0x5C, 0x5D y 0x7C no están permitidos

Dentro de un mismo directorio no se permiten crear más de un archivo con el mismo nombre.

Creación de un directorio. Los Directorios "." y ".."

Cuando se crea un directorio en FAT, se ha de crear tal y como un archivo cualquiera con la salvedad de que se ha de setear el atributo de directorio (setear el bit correspondiente en el byte DIR_Attr), y su tamaño se ha de poner a "0" (DIR_FileSize). Se reserva un primer clúster para el directorio y se pone dicho valor en DIR_FstClusLO. A continuación se deben crear dos entradas especiales en dicho directorio, el "punto" y el "punto-punto" ("." y ".."), que irán en las primeras entradas de 32 bytes en la región de datos del cluster que se ha reservado para la creación del directorio.

La primera entrada debe tener el nombre (DIR_Name):

". "

Y la segunda:

".. "

-EL campo, DIR_FileSize en ambas entradas debe estar a 0, y la fecha y hora deben coincidir con la fecha y hora de la creación del directorio.

-El campo DIR_FstClusLO, de la entrada "." debe coincidir con el campo DIR_FstClusLO del directorio recién creado.

-El campo DIR_FstClusLO, de la entrada ".." debe coincidir con el campo DIR_FstClusLO del directorio que contiene al directorio recién creado. En el caso del que el directorio recién creado se encuentre en el directorio raíz basta con poner, este campo valdrá 0. El directorio raíz será el único que no contenga las entradas "." Y ".."

Así la función de estas entradas:

- La entrada punto, se utiliza para conocer el comienzo del directorio actual

Mediante el comando SET_BLOCKLEN se puede seleccionar que el tamaño de un bloque de datos sea desde 1 byte hasta 512. Como la estructura de comienzo de bloque tiene un tamaño de 1 byte y el CRC tiene un tamaño de 2 bytes, las estructuras de datos pueden tener un tamaño desde 4 bytes hasta 515 y su formato es el siguiente:

-Primer byte: byte de comienzo de bloque.

-Bytes 2-513 (dependiendo del tamaño del bloque seleccionado): datos de usuario

-Últimos dos bytes: CRC.

En operaciones de escritura, además la tarjeta responde a continuación con una estructura para confirmar si el dato ha sido escrito o no. Dicha estructura tiene el siguiente formato:

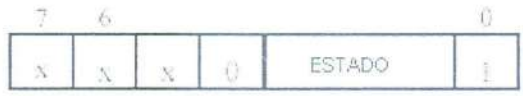

Figura 2.12. Estructura de respuesta de la tarjeta en operaciones de escritura

El significado de los bits "estado" es el siguiente:

-'010': El dato a sido aceptado

-'101': El dato ha sido rechazado debido a un error de CRC

-'110': El dato ha sido rechazado por un error en la escritura en la tarjeta

En caso de recibirse un error de este tipo durante una operación de escritura múltiple de bloques, el controlador debe detener la transmisión mediante el envío del comando CMD12. En caso de obtenerse por respuesta '110', el controlador puede conocer la naturaleza exacta del error mediante el envío del comando CMD13 (SEND_STATUS) y recogiendo su respuesta R2.

En operaciones de lectura si se produce un error, además de la respuesta esperada (R1 indicando el error) la tarjeta en vez de enviar una estructura de datos enviará una estructura de error de 1 byte con el siguiente formato:

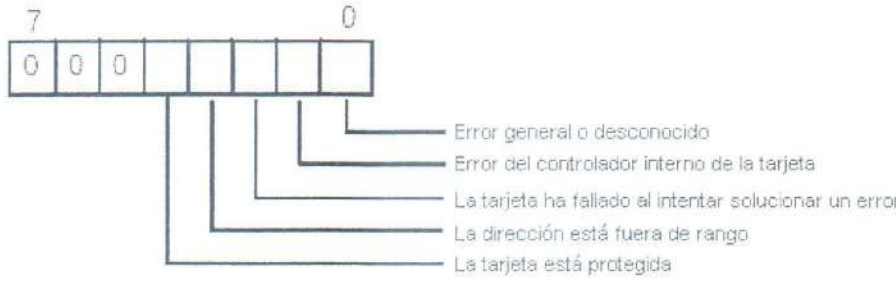

Figura 2.13. Estructura de datos al producirse un error en operaciones de lectura

Como vemos, un mismo error puede ser indicado en varios lugares. Por ejemplo, si se produce un error de CRC, éste puede ser indicado tanto en R1, como en el lugar donde la tarjeta debería responder con la estructura de datos.

2.3.5.5. Uso de los registros de la tarjeta en modo SPI

En el modo SPI tan solo son operativos los registros OCR, CSD y CID y no todos sus campos son útiles para este modo.

OCR (Operation condition register).

Registro de solo lectura de 32 bits que contiene los valores de tensión de operación soportados por la tarjeta.

Bit Nº	Voltaje Vdd
0-7	Reservado
8	2.0-2.1
9	2.1-2.2
10	2.2-2.3
11	2.3-2.4
12	2.4-2.5
13	2.5-2.6
14	2.6-2.7
15	2.7-2.8
16	2.8-2.9
17	2.9-3.0
18	3.0-3.1
19	3.1-3.2
20	3.2-3.3
21	3.3-3.4
22	3.4-3.5
23	3.5-3.6
24-30	Reservado
31	Bit indicador de "power-up" (ocupado)

Tabla 2.11. Formato del registro OCR

Así, por ejemplo, para la tarjeta MMC/SD convencional, el valor de OCR debe ser el indicado en la figura siguiente:

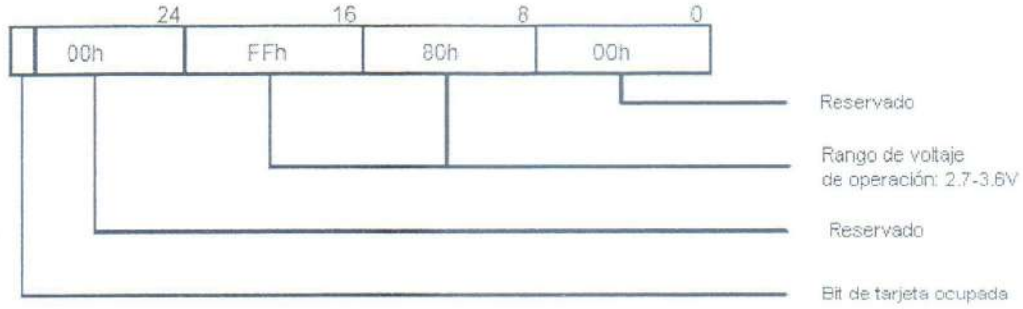

Figura 2.14. Valor de OCR en una tarjeta MMC convencional

CID (Card Identification Register).

Registro de solo lectura de 16 bits que contiene números de identificación de la tarjeta.

CSD (Card Specific Data)

Registro de lectura/escritura de 128 bits con información acerca del funcionamiento general del modelo de tarjeta.

A continuación se muestra una tabla con una ligera mención de los distintos campos del registro CSD y solamente se indican sus posibles valores para los campos más relevantes (para más información sobre los campos ver manual de la tarjeta). Los distintos valores de la columna tipo son: R (lectura), W (escritura) o E (borrable).

Bit Nº	Tipo	Descripción	Posibles Valores
[127:126]	R	Versión de la estructura de CSD	
[125:122]	R	Versión de la especificación MMC	
[121:120]	R	Reservado	
[119:112]	R	Tiempo de acceso 1 (TAAC)	
[111:104]	R	Tiempo de acceso 2 (NSAC)	
[103:96]	R	Máxima transferencia de datos	
[95:84]	R	Clases de comandos de la tarjeta	
[83:80]	R	Tamaño máximo del bloque de lectura	
79	R	Lectura parcial de bloques	'1': Permitida '0':No permitida
78	R	Escritura traspasando un bloque fisico de la tarjeta	'1': Permitido '0': No permitido
77	R	Lectura traspasando un bloque fisico de la tarjeta	'1': Permitido '0': No permitido
76	R	Registro DSR implementado	'1': Sí '0': No
[75:74]	R	Reservado	
[73:62]	R	Tamaño de la tarjeta	
[61-59]	R	Mínima corriente en la lectura	

[58:56]	R	Máxima corriente en la lectura	
[55:53]	R	Mínima corriente en la escritura	
[52:50]	R	Máxima corriente en la escritura	
[49:47]	R	Multiplicador para hallar el tamaño de la tarjeta	
[46:42]	R	Tamaño de un sector borrable	
[41:37]	R	Tamaño de un grupo a borrable	
[36:32]	R	Tamaño de un grupo protegido contra escritura	
31	R		
[30:29]	R	ECC predeterminado por el fabricante	
[28:26]	R	Facto de velocidad de lectura-escritura	
[25:22]	R	Máximo tamaño del bloque de escritura	
21	.R	Escritura parcial de bloques físicos.	'1': Permitido '0': No permitido
[20:16]	R/W	Reservado	
15	R/W	Grupo de formatos de archivo	
14	R/W	Copia (Para tarjetas OTP)	
13	R/W	Protección permanente contra escritura	
12	R/W/E	Protección temporal contra escritura	
[11:10]	R/W	Formato de archivos de la tarjeta	
[9:8]	R/W/E	Código ECC (Error Correction Code)	
[7:1]	R/W/E	CRC	
0	-	No usado Siempre vale '1'	

Tabla 2.12. Formato del registro CSD

2.3.5.6. Funcionamiento de la tarjeta MMC/SD en modo SPI

Inicialización.

El procedimiento para activar la tarjeta en modo SPI es muy sencillo, basta con que el controlador mantenga activa la línea /CS de la tarjeta (la ponga a valor lógico '0'), mientras manda el comando "CMD0", comando que resetea la tarjeta. Este procedimiento sólo puede realizarse una sola vez tras un power-up, es decir, una vez aplicada la tensión de funcionamiento a la tarjeta se debe enviar el comando CMD0 para resetearla, poniendo la línea /CS a nivel bajo si se desea el modo SPI o dejándola a nivel alto si el modo a usar es el MMC. Repetir este procedimiento no modifica el modo de funcionamiento a no ser que se retire la tensión de alimentación y se vuelva a aplicar.

Las características eléctricas de la tarjeta indican que tras la aplicación de la tensión de la alimentación el tiempo a esperar para que la tarjeta pueda responder al bus debe ser el mayor entre 1ms o 74 ciclos de reloj. Tras una aplicación de la tensión (y pasado el tiempo

necesario para que la tarjeta se inicialice) o tras un reseteo la tarjeta entra en un estado de reposo en el que sólo responderá a la aplicación de el CMD1, comando que sirve para sincronizar todo el bus. Una vez la tarjeta responda correctamente a este comando, se puede decir que se ha terminado el proceso de inicialización y se pueden comenzar a realizar operaciones en el bus.

Resumiendo, para poder realizar la correcta inicialización se deben seguir los siguientes pasos en el siguiente orden:

- Aplicar tensión de operación a la tarjeta
- Esperar 1ms o 74 ciclos de reloj (el máximo de los dos)
- Poner a '0' la señal /CS y enviar el comando CMD0 a la tarjeta. Esta debe responder con el valor 0x01 que corresponde con una respuesta en la que no se han producido errores y la tarjeta se encuentra en el estado de reposo. En caso contrario se ha producido un error.
- Enviar el comando CMD1. La tarjeta debe responder con el valor 0x00, señal de que no se ha producido ningún error y la tarjeta ya no se encuentra en el estado de reposo y está preparada para recibir comandos.

Control del reloj

El estándar establece ciertas restricciones de reloj que debe mantener el controlador para el correcto funcionamiento de la tarjeta. Se pueden resumir en:

- El reloj debe correr al menos durante 8 ciclos de reloj después de una transacción del bus para que la tarjeta termine su operación
- Tras enviar un comando a la tarjeta, ésta tardará 8 ciclos de reloj antes de procesarlo y enviar la respuesta al comando
- La frecuencia del bus se puede cambiar en cualquier momento, sólo teniendo en cuenta el valor máximo de funcionamiento

Protección de errores

Cualquier bloque enviado por el bus, sean comandos, respuestas o datos cuenta con bits CRC (código de protección de errores).

El modo SPI ofrece un modo no protegido, de manera que esta opción puede estar activada o desactivada (ver CMD59). Al inicializar la tarjeta en modo SPI, se activa el modo no protegido de manera que la tarjeta no tiene en cuenta los bits CRC. Sin embargo, si se quiere usar el modo no protegido, se deben enviar los bits CRC en la parte del

comando que correspondan, con la única salvedad de que pueden tener cualquier valor, ya que la tarjeta no los tendrá en cuenta.

La tarjeta sólo activa el modo no protegido de forma automática al entrar en el modo SPI, de modo que cualquier operación anterior deberá llevar su correspondiente CRC. Como el único comando que se debe enviar para iniciar la tarjeta en modo SPI es el CMD0, éste debe llevar su correspondiente CRC. Sin embargo, como es un comando que no lleva argumentos, no habrá que calcular CRC cada vez que se quiera usar y su valor será constante. Dicho valor ya viene calculado y el comando CMD0 quedará de la siguiente forma:

Posición de los bits	Valor	Descripción
47	'0'	Bit de inicio
46	'1'	Bit de transmisión
[45:40]	'000000'	Índice del comando CMD0 : 0
[39:8]	'00...00'	Argumento: 0
[7:1]	'1001010'	CRC7. Valor calculado: 74 (decimal)
0	'1'	Bit de final

Tabla 2.13. Formato del comando CMD0

Como en el modo SPI toda la información se envía byte a byte, y cada comando está formado por 6 bytes, el comando CMD0 consistiría en la sucesión de los siguientes bytes:

0x40,0x00,0x00,0x00,0x00,0x95

2.3.6. El MBR

MBR: Master Boot Record ("Sector De Arranque Maestro" o "Registro Principal de Arranque"). El sector de arranque maestro es el primer sector absoluto de todo disco de almacenamiento que sea arrancable (sea un disco duro, o como en nuestro caso nos interesa, una tarjeta flash). Los diskettes al no poder contener particiones son una excepción y su primer sector, denominado sector de carga del volumen, tiene una estructura ligeramente distinta. Sus 512 bytes contienen tres bloques con información sobre la arquitectura física y lógica del disco: el Código Maestro de Carga o MBC (Master Boot Code) la Tabla Maestra de Particiones o MPT (Master Partition Table) y el Marcador del Sector de Carga ("Boot Record Signature").

NOTA: Se ha de tener en cuenta que el MBR se creó para trabajar en PCs y con discos duros de modo que la BIOS pueda leer de él y cargar correctamente un sistema operativo. En nuestro sistema no contaremos con una BIOS, sin embargo nuestra tarjeta flash viene con su correspondiente MBR y por tanto se necesita conocer su estructura para poder conocer exactamente cómo usar la partición de dicha tarjeta.

Offset	Naturaleza	Tamaño
+00h	Código ejecutable (MBC)	446 bytes
+1BEh	1ª entrada de tabla de particiones	16 bytes
+1CEh	2ª entrada de tabla de particiones	16 bytes
+1DEh	3ª entrada de tabla de particiones	16 bytes
+1EEh	4ª entrada de tabla de particiones	16 bytes
+1FEh	Marcador ejecutable (AA55h)	2 bytes

Tabla 2.14. Estructura Del MBR

2.3.6.1. Código Maestro De Carga (MBC)

Si el disco es "bootable" (arrancable), los primeros 446 bytes del MBR (sector de arranque), están ocupados por un pequeño trozo de código denominado código maestro de carga MBC ("Master Boot Code") o cargador inicial (bootstrap loader), que es cargado por la BIOS en el caso de un pc para comenzar el proceso de carga. El bootstrap loader repasa la tabla maestra de particiones (ver a continuación) buscando una partición activa. En caso de encontrarla, busca su sector inicial, carga su código en memoria, y le transfiere el control. Dicho código es ya capaz de cargar y ejecutar cualquier otro programa situado en cualquier partición del disco. Que a su vez inicializará directamente el SO, o tal vez una utilidad conocida como gestor de arranque, que permite elegir entre distintas alternativas.
Como vemos, es un proceso en cadena: el bootstrap loader es cargado en memoria por un programa situado en la BIOS, y a su vez es capaz de continuar la carga del Sistema Operativo.

2.3.6.2. Tabla Maestra De Particiones (MPT)

A continuación del MBC, se sitúa la tabla maestra de particiones MPT ("Master Partition Table"). Está constituida por cuatro trozos de 16 bytes (4 entradas) que contienen información sobre las particiones definidas en la unidad.

Nota: las particiones a que nos referimos son las denominadas particiones primarias o volúmenes. Hay que recordar que solo hay sitio para cuatro, por lo que un disco duro solo

puede contener cuatro particiones primarias. Cualquier otra que pueda establecerse, se denomina partición secundaria o volumen lógico y debe estar contenida en alguna de las primarias.

Los desplazamientos de cada una de las 4 entradas son respectivamente 1BEh (446); 1CEh (462); 1DEh (478) y 1EEh (494). Cada entrada contiene la siguiente información: (T = tamaño del campo en bytes):

<table>
<tr><td>T</td><td>Descripción</td></tr>
<tr><td>1</td><td>Estado de la partición (00h = Inactiva; 80h = Activa)</td></tr>
<tr><td>1</td><td>Principio de la partición (Cabeza).</td></tr>
<tr><td>2</td><td>Principio de la partición (Cilindro/Sector en forma codificada).
La se justifica porque en el momento de su diseño, la conservación de espacio era una preocupación fundamental en el mundo informático y porque existen dos datos empaquetados en una palabra de 16 bits, que a su vez están en forma invertida; "Back-words" o "Little endian". De los 16 bits disponibles, 6 se reservan para el sector y 10 para el cilindro. Lo que conduce a un máximo de 64 = 2^6 sectores, y 1024 = 2^{10} cilindros. El esquema de distribución de bits es el siguiente:
<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">bits 7-0 del cilindro</td><td colspan="2">bits 9-8 cilindro</td><td colspan="6">bits 5-0 del sector</td></tr>
</table>
Estos valores, junto a las 256 = 2^8 posibilidades par las cabezas señaladas en el byte anterior, originarían posteriormente numerosos inconvenientes a la hora de manejar discos muy grandes.</td></tr>
<tr><td>1</td><td>Tipo de partición ID. Contiene detalles sobre el formato de la partición. Por ejemplo, si es FAT-12, FAT-16 o FAT-32; si será accedida utilizando direccionamiento CHS tradicional; ECHS, o LBA. Algunos de sus posibles valores se muestran a continuación
<table>
<tr><td>ID</td><td>Descripción</td></tr>
<tr><td>01h</td><td>Partición primaria FAT12 o volumen lógico (volumen de menos de 32,680 sectores)</td></tr>
<tr><td>04h</td><td>Partición primaria FAT16 o volumen lógico (32,680–65,535 sectores o 16 MB–33 MB)</td></tr>
<tr><td>05h</td><td>Partición DOS Extendida</td></tr>
<tr><td>06h</td><td>Partición FAT16 BIGDOS o volumen lógico (33 MB–4 GB)</td></tr>
<tr><td>07h</td><td>Partición NTFS o volumen lógico ("Installable File System")</td></tr>
<tr><td>0Bh</td><td>Partición FAT32 o volumen lógico (hasta 2048 GB)</td></tr>
<tr><td>0Ch</td><td>Partición FAT32 o volumen lógico usando extensiones de la INT 13h BIOS</td></tr>
<tr><td>0Eh</td><td>Partición FAT16 BIGDOS o volumen lógico usando extensiones de la INT</td></tr>
</table></td></tr>
</table>

			13h BIOS
		0Fh	Partición DOS extendida usando extensiones de la INT 13h BIOS
		12h	Partición EISA
		42h	Dynamic disk volume
		86h	Legacy FT FAT16 disk
		87h	Legacy FT NTFS disk
		8Bh	Legacy FT volume formateado con FAT32
		8Ch	Legacy FT formateado con FAT32 usando extensiones de la INT 13h BIOS
1	Final de la partición (cabeza).		
2	Final de la partición (Cilindro/Sector en forma codificada).		
4	Número de sectores entre el MBR y el primer sector de la partición.		
4	Número de sectores en la partición.		

Tabla 2.15. Estructura de una entrada de la tabla de particiones

Cabe destacar que los parámetros de más de un byte están almacenados en forma de palabras invertidas ("Back-words") o "*Little endian*", y como es usual, los desplazamientos empiezan por cero desde el comienzo del sector. El primer byte tiene desplazamiento cero. El último 511 (1FFh).

2.3.6.3. Marcador del Sector de Carga

Los dos últimos bytes del sector de arranque (MBR) contienen dos caracteres (55h, AAh). que son denominados marcador del sector de carga ("Boot record signature"), que se utilizan para corroborar la integridad del MBR, ya que siempre deben contener este valor.

2.4. Herramientas de desarrollo.

2.4.1. El código C y el compilador PIC-C

Como se ha comentado en el primer capítulo se ha optado por escoger el lenguaje C para el desarrollo del código por diversos motivos. Entre ellos está principalmente la facilidad para portar el código a otros modelos de microcontrolador o a otras plataformas así como todas las ventajas que ofrece el código C frente al ensamblador.

De entre los compiladores disponibles hemos optado por el compilador PIC-C (CCS C Compiler v.4).

A los largo del siguiente capítulo, se detalla el proceso de realización del software en el cual se pueden observar distintas peculiaridades del compilador; para una mayor información se ha de consultar su correspondiente manual.

2.4.2. Formato Intel HEX.

Los programas ejecutables en el sistema deben cumplir el formato Intel HEX. Normalmente por defecto, cualquier compilador generará este tipo de archivo al compilar un programa. Se trata de un formato normalizado mediante el cual un programa ejecutable se almacena en un fichero como un conjunto de datos ASCII para su posterior transferencia. Dicho fichero se divide en una serie de registros cada uno de los cuales puede contener hasta 256 bytes de código utilizable. Cada registro se organiza en una serie de campos cuyo formato se muestra a continuación.

Marca de registro	Tamaño de registro	Dirección de Inicio	Tipo de registro	Bytes de datos	Byte Checksum
:	##	aaaa	tt	dd....dd	cc

Tabla 2.16. Formato de cada registro

Donde:

- : Es el carácter de inicio de registro.
- ## Es un valor ASCII hexadecimal de dos dígitos que indica el tamaño del registro. Es decir, el número de bytes de datos que lo compone (máx. 256). Si el registro es del tipo 01, este campo debe valer 00.
- aaaa Cuatro dígitos hexadecimales expresados en ASCII que corresponden con la dirección inicial de memoria donde deben almacenarse, de forma consecutiva, los bytes de datos. Si el registro es del tipo 01, este campo debe valer 0000.
- tt Es un valor ASCII hexadecimal de dos dígitos que representan el tipo de registro.
  - 00 - Registro de datos
  - 01 - Último registro (fin de fichero)
- dd...dd Son dos caracteres ASCII por byte de datos. Habrá tantos bytes por registro como indique el campo ##. Si el tipo de registro es 01 (último registro) no habrá ningún byte de datos.
- cc Son dos caracteres ASCII hexadecimal que representan el byte del checksum. Este byte se obtiene de forma tal que, al sumar todos los bytes de los distintos campos de un registro comenzando desde ## y finalizando con el propio cc, se obtenga siempre el valor 00.

2.4.3. MPLAB® IDE.

El MPLAB® IDE ("Integrated Development Environment" o "Entorno de Desarrollo Integrado") es un software desarrollado por Microchip para realizar aplicaciones para la familia microcontroladores de Microchip. Es un software totalmente gratis y puede descargarse de la página web de Microchip (www.Microchip.com).

Este software permite desarrollar el código, tanto en ensamblador como en lenguaje C, ya que puede integrarse con todos los compiladores desarrollados para los microcontroladores de Microchip. Pero además posee otras muchas funciones útiles como son un avanzado simulador de programas, la programación de dispositivos, etc...

CAPÍTULO 3:

DESARROLLO DEL CÓDIGO

Este capítulo se detalla el proceso de desarrollo del código fuente de forma general. En ningún caso será una explicación línea a línea de éste, ya que esto puede verse en el propio código fuente. Se trata de explicar la forma en la que se ha desarrollado, detallando las principales técnicas que se han utilizado para conseguir los objetivos marcados. Para una correcta comprensión se recomienda la lectura de este capítulo junto a la del código fuente.

3.1. Modulo principal

Está compuesto de todas las funciones para manejar el hardware al más bajo nivel. Es decir, contiene las funciones básicas para el manejo de la tarjeta MMC/SD.

En realidad, es un programa que siempre permanece cargado en el PIC ocupando su correspondiente memoria de código y su correspondiente memoria de datos.

Se ejecuta al iniciar el microcontrolador o al resetearlo. Se encarga de inicializar todo el sistema, sean variables, periféricos del pic o la propia tarjeta MMC/SD, además de gestionar las funciones de bajo nivel .

El módulo principal lleva el nombre de **main.c** y se incluyen distintas macros con parámetros para el uso del resto de módulos, como pueden ser la velocidad del cristal exterior, y otras de uso general. El uso de estas macros facilita la modificación del programa, ya que si por ejemplo se cambia el cristal exterior bastará con indicarlo en su correspondiente macro y así quedará modificado para el resto del programa.

3.1.1. Desarrollo.

Dentro de este módulo como en todo programa en c está la función **main** ( ), y es lo primero que se ejecuta al iniciar el sistema. Se encarga de inicializar todo el sistema, llamando a las funciones correspondientes de inicialización de cada uno de los demás módulos e inicializando los principales registros usados en el PIC.

3.1.2 Tarjeta MMC/SD.

Al módulo con el código correspondiente a la gestión de la tarjeta SD se le ha dado el nombre de **Capa_Fisica.c** y a su archivo de cabecera **Capa_Fisica.h** Contiene las funciones básicas de comunicación con la tarjeta MMC/SD.

Como se explica en el capítulo relativo a la tarjeta MMC/SD, la comunicación en modo SPI es orientada a bytes y consiste en enviar comandos a la tarjeta y esperar una respuesta por parte de ella. Por ello se han utilizado dos funciones principales: **spi_put ( )** y **spi_get ( )**. Con la función SPI_put ( ) se envía un byte desde el PIC a la tarjeta a la vez que se recibe otro.

La función spi_get ( ) recibe los parámetros y el código de un determinado comando y los envía a la tarjeta. Como un comando esta formado por 48 bits, realizará 6 llamadas a SPI para enviarlo. Según el comando enviado, la tarjeta responderá con un tipo de respuesta R1, R2 ó R3. Sin embargo el primer byte de los tres tipos es siempre igual y coincide con R1. Por ello, la misma función spi_get ( ) recoge esa respuesta (R1) , que es el valor que devuelve.

Cuando el PIC desea recibir un byte únicamente, debe enviar un "dummy byte", es decir un byte que no debe ser interpretado por la tarjeta. Todo comando comienza con un '0' en su bit más significativo, de modo que si la tarjeta está a la espera de recibir un comando, en el momento que reciba un '0' sabrá que ha comenzado la transmisión de un comando. Por ello, si el controlador (el PIC) quiere recibir únicamente, bastará con que envíe un byte con todos sus bits a '1'. Así, el "dummy byte" es 0xFF, byte que debe enviar el PIC cada vez que quiera recibir bytes sin provocar otra reacción en la tarjeta. Una vez implementadas estas funciones se puede establecer una mínima comunicación con la tarjeta y continuar con el resto del código, de modo que se puedan enviar comandos y recibir respuestas.

Con esto, se puede realizar una lectura del registro CSD que nos da información acerca de cómo trabajar con la tarjeta. Mediante una lectura del registro se obtuvo ciertos datos a tener muy en cuenta:

- Con la tarjeta SD con la que estamos trabajado no se pueden realizar operaciones de escritura parciales de un bloque físico pero si de lectura. Un bloque físico de la tarjeta es de 512 bytes, por tanto eso quiere decir que en una operación de lectura simple de la tarjeta se pueden leer desde 1 hasta 512 bytes pero en una operación simple de escritura solamente se pueden escribir 512 bytes.

- No se puede leer ni escribir atravesando el límite impuesto por un bloque físico de la tarjeta. Es decir, si se quieren leer por ejemplo 50 bytes, no pueden ser 20 bytes de un bloque y 30 de otro.

Es por ello que se ha creado un buffer de datos para la tarjeta de 512 bytes.

```
unsigned char buf[sector_size];// sector_size = 512
```

Este búfer se utilizará tanto en operaciones de escritura como en operaciones de lectura. Además se ha utilizado una función de escritura, **WriteSector ( )** y otra de lectura, **ReadSector ( )** . La función de escritura escribe un sector completo en la tarjeta según lo almacenado en el búfer, mientras que la función de lectura puede leer desde 1 hasta 512 bytes (según lo indicado por el usuario) consecutivos pero siempre dentro del mismo sector físico de la tarjeta. Estas limitaciones hacen el proceso de escritura parcial algo lento, ya que por ejemplo, si se quiere modificar un único byte, primero se debe leer el sector completo donde se sitúa dicho byte y almacenarlo en el búfer, a continuación se ha de modificar el byte dentro del búfer, y por último se debe escribir el sector completo en la tarjeta.

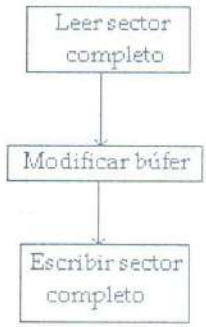

Figura 3.4. Proceso de escritura parcial de un bloque.

Adicionalmente, y en la relación a la forma en que se almacenan los datos en el formato Fat16 se ha utilizado una función: **opendread( )** que sirve para leer valores determinados dentro de la estructura Fat16 de la tarjeta. En este sistema de archivos los valores de un campo determinado se almacenan en forma de palabras invertidas, ya sean de 1 byte (char), 2 bytes (entero) o 4 bytes (long) según el rango de dicho valor. Por ejemplo, el campo "Bytes Por Sector" pude tener los valores de 512, 1024, 2048 ó 4096, por tanto necesitará de al menos un entero (2 bytes) para almacenarse. Sin embargo ese valor viene en forma invertida de modo que el byte menos significativo se encuentre en la dirección superior al byte más significativo. Así la función **opendread** puede leer un valor almacenado en un char, un int o un long (según se indique en sus parámetros).

Mediante la función **sendCommand ( )**, se realiza la inicialización de la interfaz SPI de la tarjeta.. El registro de control para el módulo SPI es el SSPCON1, mediante el cual se configura la activación del modo SPI, el funcionamiento maestro o esclavo, la velocidad del bus y otros parámetros que no serán importantes en nuestro diseño. Configuramos el bus con la velocidad más alta posible, Fosc/4, siendo Fosc la velocidad del cristal exterior del PIC. Así la velocidad de funcionamiento del bus SPI es de 20Mhz/4 = 5Mhz.

3.1.3. Reloj de tiempo real.

La frecuencia de reloj utilizada es de 20.MHz. La arquitectura interna del pic determina que el timer se incrementa con una frecuencia de Fosc/4 lo cual nos permite temporizar un tiempo mínimo de 1/20.000.000/4=0.0000002 segundos, que es el tiempo que tarda en incrementarse el timer. Hemos determinado que para nuestro sistema, un tiempo base aceptable sea el de 10ms, por tanto, para temporizar 10 ms el temporizador deberá generar una interrupción cada:

$$\frac{10ms}{0.0002ms} = 50000cuentas$$

Así, si un timer se desborda provocando una interrupción cada 50000 cuentas, sabremos que esa interrupción se provocará cada 10ms. Se puede establecer una fórmula de carácter general de modo que el cambio del cristal externo no afecte al funcionamiento del RTC. De este modo el número de incrementos que debe realizar el timer antes de desbordarse será:

$$\frac{PIC\_CLK * Tiempo\_Base}{4}$$

donde tiempo base indica el tiempo mínimo a temporizar en segundos y PIC_CLK, la frecuencia de reloj externo en hertzios.

Una interrupción del timer se provoca por el paso de su valor máximo a 0, por tanto para contar 10 ms el timer debe cargarse con el valor:

(Valor máximo + 1) – (Nº de cuentas a realizar)

Como se necesitan 50000 cuentas se ha optado por utilizar el timer en modo 16 bits lo que permite un máximo 65535 cuentas y se ha desactivado el prescaler

3.1.4. Sistema de archivos FAT16.

El módulo con el código correspondiente al manejo del sistema de archivos FAT16 lleva el nombre de **FAT16.c** y su archivo de cabecera lleva el nombre de **FAT16.h**. Se debe

proveer al sistema de todas las funciones necesarias para el manejo de archivos y directorios, lo cual lo convierte en el módulo más importante y complejo.

La primera acción a realizar dentro del sistema FAT16 es conocer donde se sitúa la partición FAT dentro del disco y a continuación se deben conocer todos los parámetros de la FAT situados en el BPB (BIOS Parameter Block). Como se detalló en el capítulo correspondiente al FAT16 esos parámetros dependen del tamaño del disco y de cómo haya sido formateado así como del MBR. Es decir, el tamaño de un clúster puede ser distinto de un disco a otro así como el sector de comienzo de la partición FAT. Para guardar toda esa información se ha usado una estructura y una variable de dicha estructura, que debe ser rellena al inicializar el módulo fat16 mediante la función **FATLibInit( )**;

Es necesario conocer todos estos parámetros antes de nada ya que se debe conocer en qué dirección de la tarjeta MMC/SD comienza la partición FAT, en que dirección comienza la tabla FAT, en que dirección se encuentran los datos, etc... Como ya se ha comentado, en el módulo MMC/SD se utilizo una función llamada **opendread( )** mediante la cual podemos obtener todos esos valores de la tarjeta. El primer parámetro a conocer es el de comienzo de la partición FAT. Para ello se debe leer del MBR. El MBR se encuentra en el primer sector físico de la tarjeta. Debemos obviar que las tarjetas utilizadas en este sistema contienen una única partición FAT16, es por ello que el sistema no tratará de averiguar el tipo de partición de la tarjeta ni si existen más particiones dentro de esta. Observando las tablas 2.14 y 2.15 correspondientes al MBR se extrae que la información sobre la primera partición se sitúa en el MBR en el offset 0x1Beh, y dentro de esta, el campo con información sobre el comienzo de la partición se sitúa en el offset +8. Este campo tiene el tamaño de 4 bytes, aunque como hemos obviado que sólo se tendrá una partición, el número será lo suficientemente pequeño como para caber en un entero (2 bytes).

Donde el '0' indica el número de sector de la tarjeta (se comienza la numeración por '0') y 0x1C6 el offset a leer dentro de dicho sector. Se ha dado el nombre de *"dataStarts"* porque es el primer sector de una partición FAT donde comienzan los datos (BPB) y el campo *"rootDirectory"* indica el sector de comienzo de la tabla FAT (no de la partición). Una vez conocido el sector donde se sitúa el BPB, se pueden leer el resto de parámetros y se pueden calcular otros importantes tales como el espacio libre, etc...y la función **FindEmptyClusters( )** busca todos los clústeres vacíos en la tabla FAT.

La unidad mínima para almacenar un archivo en un sistema FAT es el clúster y un archivo puede almacenarse en clústeres no consecutivos. Cada entrada de la tabla FAT corresponde a un clúster y el valor de dicha entrada corresponde con el clúster siguiente al anterior. Por ejemplo, si un archivo está formado por 2 clústeres, el 5 y el 8, y el clúster de

comienzo del archivo es el 5, si se lee en la entrada de la FAT número 5, el valor de dicha entrada será 8, y una lectura de la entrada número 8 dará el valor 0xFFFF que indica que no existe un clúster siguiente para dicho archivo. La función **getNextFAT( )**, buscar el cluster siguiente de un archivo aceptando como parámetro el número de cluster actual. Mediante la función **getFirstCluster( )** se obtiene el primer clúster libre de la tabla (entrada de la tabla FAT con valor 0x00), valor útil a la hora de crear archivos o directorios o modificarlos. Mediante la función **WriteSector( )**, se escribe un valor en una determinada entrada de la tabla FAT. Mediante la función **createNewCluster ( )**, se inicializa un clúster, es decir se ponen todos sus bytes con valor 0x00h.

A partir de estas funciones se han desarrollado todas las funciones necesarias para el manejo de archivos de la partición FAT16. En el código fuente puede verse la implementación de todas esas funciones.

3.2. Distribución de la memoria

Una vez desarrollado el modulo principal con todas las funciones necesarias para su correcto funcionamiento, se puede conocer la memoria ocupada por este y qué cantidad de memoria queda libre para las futuras aplicaciones de usuario. Así, al compilar todo el código main.c podemos observar que la cantidad de memoria ocupada es la siguiente:

```
ROM used  13014 (40%)
          13014 (40%) including unused fragments

          4 Average locations per line
          12 Average locations per statement

RAM Used  739 (36%) at main() level
          1815 (89%) worst case

Stack Used  6  worst case (out of 31 total available)
```

Figura 3.9. Memoria ocupada por el modulo principal

Cada modelo de compilador obtendrá un código distinto al obtenido por el resto de compiladores en términos de tamaño, optimización, rapidez, etc etc..., lo cual no es objeto de este estudio

Se decidió colocar el código a partir de la dirección 0x00 tanto en la memoria de código como en la memoria de datos, y dejar la zona restante para los programas de usuario.

Sin embargo nos encontramos con una limitación fundamental impuesta por la estructura de la memoria de datos del PIC y por el propio funcionamiento del compilador. Esa limitación queda impuesta por el llamado "Access Bank" que implementan los PIC.

"Access Bank"

Consiste en una zona de memoria de 256 bytes formada por los registros de funciones especiales (SFRs) y por la zona baja del primer banco de memoria RAM. La memoria de datos de los PIC se divide en bancos de memoria de 256 bytes. La del PIC18F4550 está formada por 8 bancos, por tanto dispone de una memoria de 8*256=2048bytes. De ellos 160 bytes están reservados para los SFR quedando para el usuario 1888 bytes. Para direccionar toda esa zona se necesitan al menos 12 bits (2^{11}=2048). Para ello existe el registro BSR cuyos últimos 4 bits indican que banco es el seleccionado. Así para referirse a una dirección bastará con indicar la dirección dentro del banco seleccionado e indicar en los últimos 4 bits de BSR el banco seleccionado.

Para agilizar este proceso los PIC implementan el llamado "Access Bank", creando una zona de memoria que se puede acceder mediante 8 bits. Sin el uso del Access Bank, la lectura de una dirección implica en primer lugar una lectura de BSR para saber a que banco corresponde la dirección introducida a continuación. Obviamente, todo este proceso es realizado por el compilador. Sin embargo, el escribir y leer continuamente de BSR puede desembocar en la creación de un código bastante lento e ineficiente. Un ejemplo de ello son los SFR (registros de funciones especiales), que deben ser leídos y escritos muchas veces en el desarrollo de un programa. Es por ello que toda la zona de los SFR se ha incluido en el Access Bank. Además existe otra zona de los registros de propósito general (GPR – la zona de memoria de datos de usuario) también incluida en el Access Bank. Esta última zona es usada por el compilador principalmente para situar variables temporales, por ejemplo para ir almacenando el valor de retorno de una función de mas de 1 byte. El Access Bank consiste en una forma de direccionar la memoria de datos sin utilizar el BSR de modo que se puede conseguir una mayor rapidez en el funcionamiento del programa. Las instrucciones que soportan dicho direccionamiento tienen un bit bandera llamado 'a' en el datasheet del PIC, mediante el que se indica si se usará el Access Bank o no. Si se va a usar, no se leerá el registro BSR y la dirección introducida corresponderá a la dirección dentro del Access Bank. Si dicho bit indica que no se va a usar, para determinar la dirección se debe leer el registro BSR.

El tamaño del Access Bank es de 256 bytes, ya que si BSR no es utilizado, una dirección se indica con 8 bits. (2^8=256). No existe forma de seleccionar la localización de esa zona que siempre es fija y consiste en los primeros 128 bytes del banco 0 y en los últimos 128 bytes del banco 15 (donde se encuentran los SFR).

Como ya se ha comentado, por motivos de eficiencia, el compilador usa el llamado Access Bank. Existen dos modelos de memoria generados por el compilador pero en ambos casos se usa dicha zona. Para los SFR no hay ningún problema ya que el modulo principal podrá acceder a ellos. Sin embargo, sí existe dicho problema para la zona baja del banco 0.

Como el compilador siempre usa el Acces Bank deberá de disponer siempre de parte de la zona baja del Banco 0.

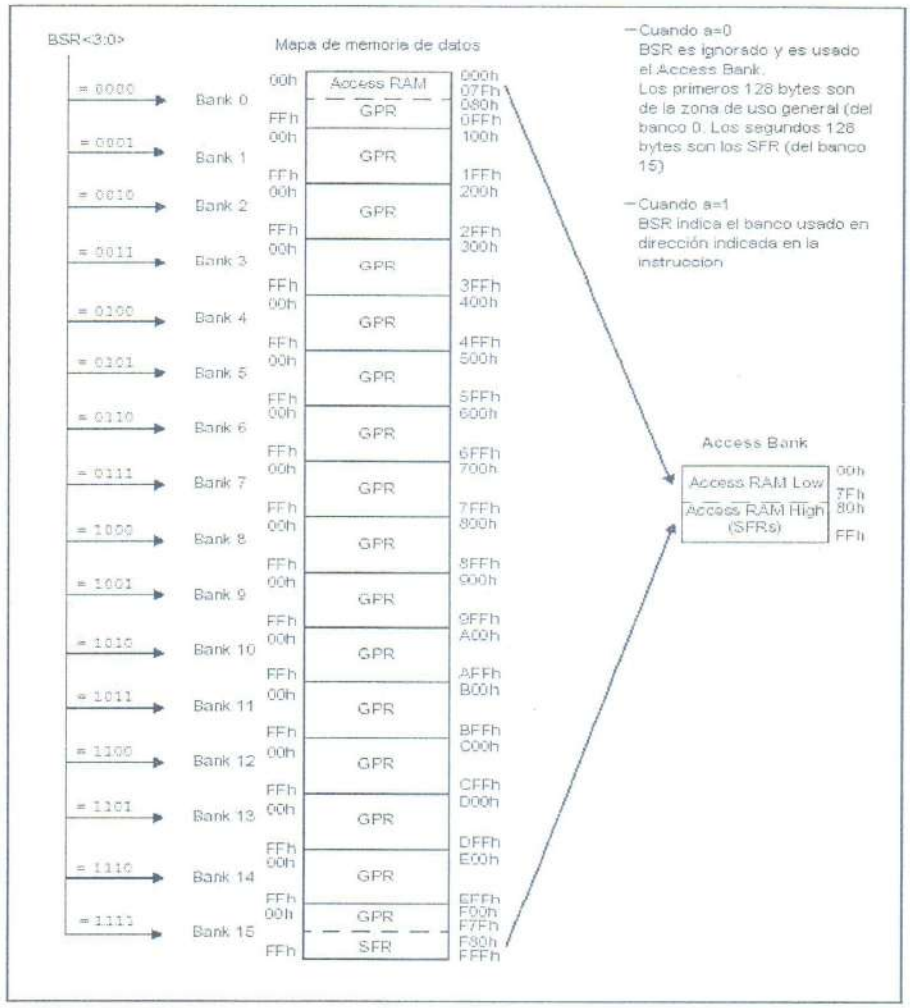

Figura 3.10. Mapa de memoria de datos del PIC

Un inconveniente que se puede presentar en este proceso es que el compilador produce errores en ciertas ocasiones cuando se pretenden utilizar zonas parciales de un banco físico de memoria de datos. Por tanto, para evitar estos errores, a la hora de distribuir la memoria de datos se debe tener en cuenta que cada zona reservada no ocupe un banco parcialmente. La capacidad de manejar la memoria por parte del compilador permite resolver estos errores, pero ello implica un conocimiento muy exhaustivo del compilador por parte del programador, lo cual no es aconsejable. Por ello se ha optado por la distribución de la memoria por defecto.

CAPÍTULO 4:

GUÍA DE USUARIO DEL SISTEMA

4.1. Funcionamiento General.

El usuario del sistema será aquel que desarrolle una aplicación especifica de adquisición de datos para almacenarlo en la tarjeta MMC/SD en forma de archivo con extensión *.txt. Es por ello que debe conocer ciertas peculiaridades y características de funcionamiento para conocer qué posibilidades y qué limitaciones tiene.

Se trató de conseguir el mayor nivel de abstracción posible para el usuario, de manera que se libere a éste de tener que conocer en detalle el funcionamiento de la tarjeta MMC/SD, o del sistema de archivos FAT16, por ejemplo, y tenga a su disposición una serie de funciones estándar para el manejo de archivos, como puede ser la de crear un directorio introduciendo su nombre, por ejemplo. No obstante, el capítulo 3 aborda toda la cuestión del desarrollo del software con todas sus peculiaridades y las principales técnicas utilizadas, de modo que el usuario comprenda los aspectos fundamentales de funcionamiento y pueda desarrollar sistemas personalizados, cambiando parámetros básicos de funcionamiento del modulo principal, o añadiendo o quitando funciones.

4.2. Desarrollo de aplicaciones.

El desarrollador de aplicaciones debe conocer algunas peculiaridades con el fin de poder realizar programas que se ejecuten de forma adecuada, sin afectar al funcionamiento del modulo principal. Para ello, los aspectos fundamentales que debe conocer son:

- La memoria disponible para los programas de usuario, conociendo la cantidad de memoria, tanto de código cómo de datos, tiene disponible.
- Qué periféricos son utilizados por el modulo principal y no pueden ser utilizados por los programas de usuario.
- Los pines de E/S básicos utilizados para la comunicación con la tarjeta MMC/SD, que no estarán disponibles para uso general.

Será responsabilidad del programador realizar los programas de la manera correcta, de acuerdo con estas limitaciones

4.2.1. Recursos utilizados por el software.

Memoria.

La distribución de la memoria, así como las técnicas utilizadas para la comunicación de los programas con modulo principal pueden verse en el capítulo 3.

Periféricos.

El modulo principal, además de memoria utiliza determinados periféricos del microcontrolador, de modo que éstos no estén disponibles para los programas de usuario. El programador debe conocer éstos periféricos y que registros tienen asociados, incluidos los de control de sus interrupciones, de modo que el código no afecte a la configuración de estos periféricos.

- **Módulo MSSP.**
  Es utilizado para la comunicación con la tarjeta MMC funcionando en el modo SPI.
- **Timer 0.**
  El timer 0 sirve de base de tiempo para el reloj de tiempo real del sistema, necesario para dotarlo de hora y fecha al trabajar con ficheros.

Pines de E/S.

Todo microcontrolador en el que se desee ejecutar el sistema debe dedicar una serie de pines a la comunicación con la tarjeta MMC/SD. De este modo, estos pines no estarán disponibles para uso general de un programa de usuario. La siguiente figura muestra los pines de E/S utilizados por la tarjeta SD

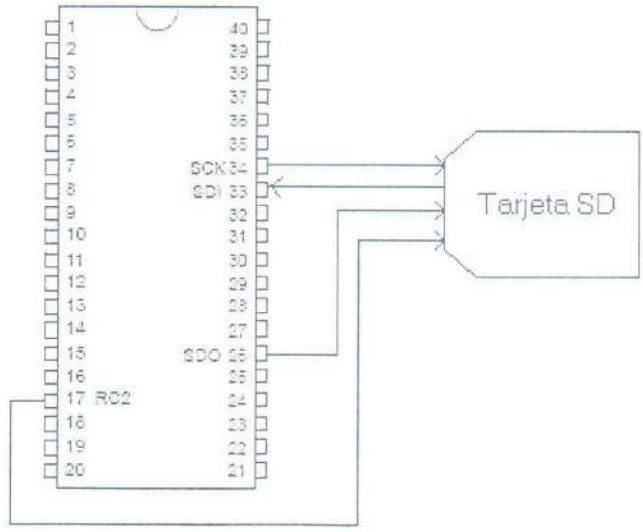

Figura 4.3. Conexión básica del sistema.

4.2.1.1. Manejo de las librerías del sistema.

El usuario tiene a su disposición todas las librerías que forman parte del sistema: "string.h" , "FAT16.h", "capa_fisica.h" Tanto el código fuente, como los archivos de cabecera.

Éstas librerías contiene todas las llamadas a las funciones implementadas en el modulo principal del sistema. Contiene llamadas a funciones de bajo nivel, como puede ser el envío de un comando a la tarjeta MMC/SD, la lectura de un sector de ésta, etc...

4.2.1.2. Finalización del programa.

Las aplicaciones que se ejecuten en el sistema pueden tener un funcionamiento; permanente, hasta que se produzca un reset del sistema, tras realizar una tarea determinada o se cumpla una determinada condición. En el ejemplo implementado la finalización tiene lugar al tomarse 20 muestras (cada 10 seg) de temperatura guardando su valor en un archivo. (Usando el C.I. LM35 conectado a AN0)

4.2.1.3. Dependencia del código con el compilador.

La elección de un compilador u otro, determino en gran medida la sintaxis del código. Un ejemplo de esto puede verse en el manejo de los registros del PIC, ya que cada compilador puede utilizar nombres distintos para ellos. A su vez, en el código han sido utilizadas algunas funciones que proporcionaba el compilador, como pueden ser las relativas al tratamiento de cadenas de texto. Sin embargo, todas las funciones utilizadas son estándar y las proporciona cualquier compilador, no habiéndose utilizado ninguna específica del compilador PIC-C (CCS C).

En algunas situaciones muy concretas, y cuando la situación lo requería, se ha utilizado sintaxis de ensamblador. No obstante, en esta memoria, en todo momento se han tratado de explicar los procesos y las técnicas por encima de la sintaxis del código. De este modo, y gracias a la portabilidad que ofrece el código C, el usuario puede adaptar fácilmente el código de las librerías al compilador que use.