Universidad Michoacana De San Nicolás De Hidalgo

FACULTAD DE INGENIERÍA ELÉCTRICA

IMPLEMENTACION Y DISEÑO DE UN SISTE MA DE ALMACENAMIENTO MASIVO BASADO EN MICROCONTROLADOR Y MEMORIA FLASH SD CON ENFOQUE A SISTEMA MINIMO

TESIS

Que para obtener el grado de:

INGENIERO ELECTRICISTA

Presenta:

GUSTAVO GARIBAY GARIBAY

Asesor de tesis:

ING. FELIX JIMÉNEZ PEREZ

MORELIA, MICHOACÁN, FEBRERO 2008

Agradecimientos

Dedicatoria

Contenido

Capítulo 1

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 6

Capítulo 7

Capítulo 8

Apéndices

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

Este trabajo presenta el diseño e implementación de un dispositivo de almacenamiento masivo basado en microcontrolador y orientado a sistemas mínimos, utilizando tarjetas de memoria Flash.

1.1 Antecedentes

En la actualidad los medios de almacenamiento son cada vez más necesarios y se requieren almacenar grandes cantidades de información ya sean imágenes, videos, sonidos, documentos o simplemente datos. Como ejemplo podríamos tomar lo teléfonos celulares, que además de funcionar como teléfono ahora ya contienen mas y mas funciones integradas como cámaras digitales, reproductores de música y video, procesadores de texto, entre otras. Todas estas funciones requieren que el dispositivo cuente con suficiente memoria.

Con el avance de la tecnología, los medios de almacenamiento son diseñados con mayores capacidades de memoria y de menor tamaño físico. Algunos dispositivos como organizadores, celulares u otros sistemas mínimos no son diseñados para utilizar memoria adicional a la ya integrada, pero en ocasiones la memoria integrada no nos es suficiente para nuestras necesidades.

1.2 Objetivo

Los objetivos principales son presentar una introducción a los sistemas de archivos y servir como referencia para trabajos posteriores.

El sistema a desarrollar se basa en un microcontrolador de la familia PIC18 el objetivo es implementar un dispositivo para que pueda realizar operaciones de lectura y escritura sobre tarjetas flash tipo SD (Secure Digital) de diferentes capacidades y además se pretende utilizar este trabajo en sistemas mínimos, por lo tanto el dispositivo deberá ser pequeño y de bajo consumo además de un fácil manejo.

El dispositivo debe ser de uso sencillo y transparente para el usuario que solo se encargara de dar las instrucciones ya sea para realizar una lectura o almacenar información.

Una de las principales características es que este sistema debe ser compatible con otros sistemas operativos como Windows, además es conveniente que la memoria tenga algunas ventajas, por ejemplo: memoria no volátil para que la información no se borre o modifique cuando sea retirada la energía.

1.3 Organización de la tesis

En el capítulo 1 se da una breve introducción a este trabajo, se mencionan las necesidades de almacenar información con los avances de la tecnología, y se describe en breve el objetivo del sistema que se pretende desarrollar.

En el capítulo 2 se presentan los tipos de memoria como medio de almacenamiento donde se comparan las características de la tarjeta SD con otros tipos de memoria o incluso con diferentes dispositivos de memoria flash.

En el capítulo 3 se analizan a fondo las tarjetas SD, se describen los registros y componentes que conforman la memoria, así como especificaciones físicas y la forma en que se organiza la memoria.

En el capítulo 4 se da una pequeña introducción al sistema de archivos FAT16 , se analiza como son organizados los componentes en este formato y se explica que es el Master Boot Record, la tabla FAT y el directorio raíz.

En el capítulo 5 muestra el microcontrolador de la familia PIC18, se analizan los registros de control y estado, así como los subsistemas disponibles y los utilizados para nuestro sistema.

El capítulo 6 aborda el sistema desarrollado, se describen las partes que los conforman como es la fuente, el zócalo para la tarjeta SD, el cristal, etc. se describe también el programa desarrollado.

El capítulo 7 muestra la implementación del sistema y su funcionamiento, dando una breve descripción para el usuario.

Por ultimo se establecen las conclusiones del trabajo se muestran las desventajas y las mejoras a futuro.

Capítulo 2

Tipos de memorias

Memoria se refiere a componentes de una computadora, dispositivos o medios de grabación que retienen datos durante algún intervalo determinado.

En la actualidad la memoria suele referirse a una forma de almacenamiento de datos e instrucciones. En la memoria se realizan dos operaciones básicas: lectura y escritura. A continuación se dan algunos conceptos básicos de memoria.

Unidades de memoria

BIT: puede tener valor de 1 o 0, es decir sistema binario.

BYTE: son 8 bits.

KILOBYTE (KB) = 1024 bytes

MEGABYTE (MB) = 1024 KB

GIGABYTE (GB) = 1024 MB

Tabla 2.1 Diferentes Tipos de Memorias

Tipo	**Categoría**	**Método de Borrado**	**Volátil**	**Aplicación Típica**
RAM	Lectura/Escritura	Eléctrico	Si	Cache/ Memoria principal
EPROM	Lectura principalmente	Luz UV	No	Prototipos
EEPROM	Lectura principalmente	Eléctrico	No	Prototipos y configuración
FLASH	Lectura/Escritura	Eléctrico	No	Dispositivos portátiles

2.1 RAM

Este tipo de memoria se le denomina memoria de acceso aleatorio, es de tipo volátil, esto es, que sólo almacena información mientras esté conectada la energía eléctrica y pierde su contenido al desconectarla. Se dicen de acceso aleatorio porque los diferentes acceso son independientes entre si. En estas memorias se accede a cada celda mediante un cableado interno, es decir, cada byte tiene un camino para entrar y salir, a diferencia de otros tipos de almacenamiento, en las que hay una cabeza lecto-grabadora que tiene que ubicarse en la posición deseada antes de leer el dato deseado.

Las memorias RAM se dividen en estáticas y dinámicas:

Estática – también conocida como SRAM, es un tipo de memoria que es más rápida y más confiable que la DRAM. El término se deriva del hecho de que no necesita ser restaurada, mantiene su contenido inalterado mientras esté alimentada.

Dinámica – un tipo de memoria física usado en la mayoría de las computadoras personales. El término dinámico indica que la memoria debe ser restaurada constantemente (re-energizada) o perderá su contenido. Se refiere a veces como DRAM para distinguirla de la RAM estática (SRAM) que es más rápida y menos volátil que la DRAM. Aunque la SRAM requiere mas potencia y es mas costosa.

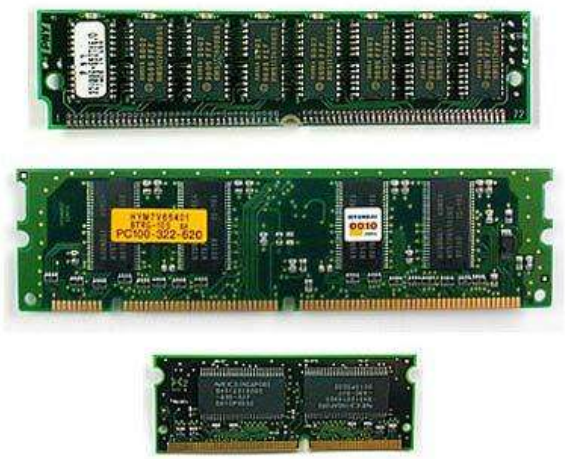


Figura 2.1 Tarjetas de memoria tipo RAM

2.2 EPROM

Erasable Programmable Read-Only Memory (memoria de solo lectura, borrable programable). Es un chip de memoria inventado por el ingeniero Dov Frohman. Este tipo de memoria retiene los datos cuando la fuente de energía se apaga. En otras palabras, es "no volátil". Está formada por *transistores de puerta flotante*. Cada uno de ellos viene de fábrica sin carga, por lo que es leído como un 1 (por eso una EPROM sin grabar se lee como **0xFF** en todas sus celdas). Se programan mediante un dispositivo electrónico que proporciona voltajes superiores a los normalmente utilizados en los circuitos electrónicos. Las celdas que reciben carga se leen entonces como un 0. Una vez programada, una EPROM se puede borrar solamente mediante exposición a una fuerte luz ultravioleta debido a que los fotones de la luz excitan a los electrones de las celdas provocando que se descarguen. Las EPROMs se reconocen fácilmente por una ventana transparente en la parte alta del encapsulado, a través de la cual se puede ver el chip de silicio y que admite la luz ultravioleta durante el borrado.

Figura 2.2 Memoria tipo EPROM

2.3 EEPROM

De las siglas *electrically-erasable programmable read-only memory*. Es un tipo de memoria ROM que puede ser programada, borrada y reprogramada de nuevo eléctricamente, a diferencia de la EPROM que es borrada mediante rayos ultravioletas. Aunque una EEPROM puede ser leída un número ilimitado de veces, sólo puede ser borrada y reprogramada entre 100,000 y 1,000, 000 de veces.

Las memorias EEPROM son memorias no volátiles y eléctricamente borrables a nivel de bytes. La posibilidad de programar y borrar las memorias a nivel de bytes supone una gran flexibilidad, pero también una celda de memoria más compleja. Debido a que la celda elemental de este tipo de memorias es más complicada que sus equivalentes en EPROM o PROM (y por ello bastante más cara), este tipo de memoria no dispone en el mercado de una variedad tan amplia.

Estos dispositivos suelen comunicarse mediante protocolos como I^2C, SPI y Microwire. En otras ocasiones se integra dentro de chips como microcontroladores y DSPs para lograr una mayor rapidez.

Figura 2.3 Memoria tipo EEPROM

2.4 Flash

La memoria flash es similar a la EEPROM, es decir que se puede programar y borrar eléctricamente, son de *alta densidad* (gran capacidad de almacenamiento), significa que se puede empaquetar en una pequeña superficie del chip, gran cantidad de celdas, lo que implica que cuanto mayor sea la densidad, más información se puede almacenar en un chip de tamaño determinado. Sin embargo ésta reúne algunas de las propiedades de las memorias anteriormente vistas, y es de fabricación sencilla, lo que permite fabricar memorias de capacidad equivalente a las EPROM a menor costo que las EEPROM.

La memoria Flash es ideal para docenas de aplicaciones portátiles. Actualmente, los usos de Memoria Flash se están incrementando rápidamente, ya sean cámaras digitales, asistentes digitales portátiles, reproductores de música digital o teléfonos celulares, todos necesitan una forma fácil y confiable de almacenar y transportar información.

Por esto la memoria Flash se ha convertido en poco tiempo en una de las más populares tecnologías de almacenamiento de datos. Es más flexible que un diskette. Es mucho más rápida que un disco duro, y a diferencia de la memoria RAM, las memorias flash son de tipo no volátil, la información almacenada no se pierde cuando se desconecta de la energía, una característica muy valorada para la multitud de usos en los que se emplea este tipo de memoria.

Figura 2.4 Diferentes modelos de memoria tipo FLASH

2.4.1 Diferencia entre memorias EEPROM y FLASH

La principal diferencia de las memorias flash y las EEPROM reside en su velocidad: Son más rápidas en términos de programac ión y borrado, aunque también necesitan de una tensión de grabado del orden de 12 voltios.

Por otra parte estas memorias son bastante más baratas que las EEPROM, debido a que utilizan una tecnología más sencilla y se fabrican con grandes capacidades de almacenamiento.

2.4.2 Tipos de memoria Flash

La memoria flash siempre ha estado muy vinculada con el avance del resto de las tecnologías a las que presta sus servicios como routers, modems, BIOS de los PCs, wireless, etc. Fujio Masuoka en 1984 inventó este tipo de memoria como evolución de las EEPROM existentes por aquel entonces.

Entre los años 1994 y 1998, se desarrollaron los principales tipos de memoria que conocemos hoy, como la SmartMedia o la CompactFlash. La tecnología pronto planteó aplicaciones en otros campos. En 1998, la compañía Rio comercializó el primer *Walkman* sin piezas móviles aprovechando el modo de funcionamiento de SmartMedia. Era el sueño de todo deportista que sufría los saltos de un diskman en el bolsillo.

CompactFlash (CF)

Este fue el primer tipo de tarjetas flash que se hizo popular al inicio. Creado por Sandisk, son pequeñas y ligeras con gran capacidad de almacenamiento. Mide 43x36 milímetros y pesa entre 15 y 20 gramos según el tipo, I o II. El segundo es más fino aunque también funciona en las ranuras del tipo I. La emplean numerosas marcas, como Nikon, Samsung o Minolta. Las hay desde 8 Megabytes hasta 12 Gigabytes aunque las más habituales son de 512 Megabytes ó 1 Gigabytes.

Figura 2.5 Tarjeta de memoria FLASH tipo CompactFlash

Smart Media Card (SMC)

Es mucho muy pequeña y apenas pesa dos gramos. La emplean Fuji, Samsung y Olympus. Está quedando relegada, en parte debido a que no supera los 256 Megabytes de capacidad.

Figura 2.6 Tarjetas de memoria FLASH tipo SMC

Memory Stick

Creada por Sony, se emplea en casi todos sus dispositivos, desde teléfonos móviles a consolas de videojuegos portátiles, pero apenas ha cuajado en la competencia. Hay una versión más reciente bautizada como Memory Stick Pro que, según Sony, podría alcanzar los 32 Gigabytes de capacidad con una velocidad de transferencia de 20 Megabytes por segundo.

Figura 2.7 Tarjetas de memoria FLASH tipo Memory Stick

Secure Digital (SD)

Se emplea con frecuencia para ofrecer datos en los que se quiere limitar la copia (como programas informáticos o música). Pesa unos dos gramos, mide 32x24 milímetros y es empleada por Kodak, Casio, Hewlett Packard, Nikon, Canon, Minolta, Panasonic y Toshiba, entre otras. Su capacidad de almacenamiento puede alcanzar los 2 Gigabytes. Actualmente tiene el 40% del mercado de memorias flash.

Figura 2.8 Tarjetas de memoria FLASH tipo SD

Multimedia Card (MMC)

Nacida en 1997 de la mano de Siemens y Sandisk, es similar a Secure Digital. Muchos aparatos con puerto para SD la pueden utilizar, aunque no pasa lo mismo a la inversa. Mide 24x32 milímetros y almacena hasta 2Gb.

Figura 2.9 Tarjetas de memoria FLASH tipo MMC

USB

Es un pequeño dispositivo de almacenamiento que utiliza memoria flash para guardar la información sin necesidad de baterías. Estas memorias son resistentes a los rasguños y al polvo que han afectado a las formas previas de almacenamiento portátil, como los CD y los disquetes. Estas memorias se han convertido en el sistema de almacenamiento y transporte personal de datos más utilizado, desplazando en este uso a los tradicionales disquetes, y a los CDs. Se pueden encontrar en el mercado fácilmente memorias de 1, 2, 4, 8 GB o más por un precio moderado. Su gran popularidad le ha supuesto infinidad de denominaciones populares relacionadas con su pequeño tamaño y las diversas formas de presentación .

Figura 2.10 Memoria FLASH tipo USB

Algunas especificaciones para considerar al elegir un tipo de memoria FLASH incluyen tamaño físico, capacidades, opciones de interfaz, velocidad de transferencia de datos, voltaje de suministro y costo.

Capítulo 3

La memoria SD

En agosto de 1999, Panasonic, SanDisk y Toshiba anunciaron un acuerdo para desarrollar una nueva generación de tarjetas de memoria llamada SD Card. Esta tarjeta nueva fue diseñada para competir con el formato Memory Stick de Sony.

Para crear una tarjeta SD, Toshiba añadió hardware cifrado a la ya existente tarjeta MMC, para aliviar las preocupaciones de la industria de la música. Las tarjetas SD han sustituido a las SmartMedia como formato de tarjeta de memoria dominante en cámaras digitales compactas.

La tarjeta SD es una memoria flash, que esta diseñada para obtener seguridad, capacidad, funcionamiento y requerimientos de los nuevos dispositivos electrónicos de audio y video. Permite almacenar información desde los 16 MB hasta los 2 GB . Además del empaquetado original existen otros dos formas de tarjeta SD: miniSD y microSD. La tarjeta SD tiene gran disponibilidad en el mercado y es de bajo costo .

La tarjeta SD esta basada en una interfaz avanzada de 9 pines (Reloj, Comando, 4 líneas de Datos y 3 de alimentación) diseñada para operar en un rango de bajo voltaje. La tarjeta SD utiliza dos modos de comunicación SD bus y SPI.

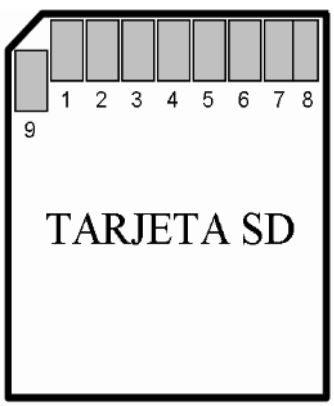


Figura 3.1 Contactos en la tarjeta SD

3.1 Especificaciones físicas

Peso máximo 2.0 gr.

Dimensiones: 32mm x 24mm x 2.1mm

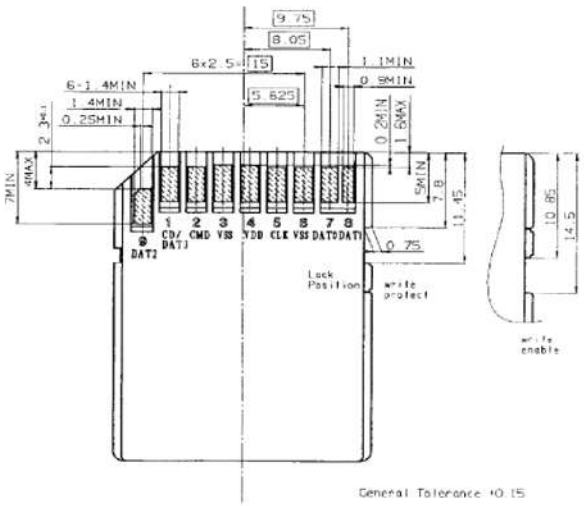


Figura 3.2 Dimensiones físicas de la tarjeta SD

Dependiendo del protocolo de comunicación utilizado, las terminales están asignadas de la siguiente forma:

Tabla 3.1 Asignación de pines en modo SD

PIN	NOMBRE	TIPO	DESCRIPCION SD
1	DT/DAT3	I/O	Detector de tarjeta/ Línea de datos [Bit 3]
2	CMD	I/O	Comando/Respuesta
3	VSS1	S	Tierra
4	VDD	S	Voltaje de alimentación
5	CLK	I	Reloj
6	VSS2	S	Tierra
7	Dat0	I/O	Línea de datos [Bit 0]
8	Dat1	I/O	Línea de datos [Bit 1]
9	Dat2	I/O	Línea de datos [Bit 2]

S = Alimentación; I = Entrada; O = Salida.

Tabla 3.2 Asignación de pines en modo SPI

PIN	NOMBRE	TIPO	DESCRIPCION SPI
1	CS	I	Chip Select
2	DataIn	I	Comandos y datos de entrada
3	VSS1	S	Tierra
4	VDD	S	Voltaje de alimentación
5	CLK	I	Reloj
6	VSS2	S	Tierra
7	DataOut	O	Estado y datos de salida
8	RSV	I	Reservado
9	RSV	I	Reservado

S = Alimentación; I = Entrada; O = Salida.

Además la tarjeta SD cuenta con un interruptor mecánico para protección de escritura, una marca en la tarjeta indica si esta en posición de protección o no.

Algunos de los modelos disponibles en el mercado así como sus capacidades se muestran en la siguiente tabla:

Tabla3.3 Algunos modelos de tarjetas SD

Modelo No.	Capacidad
SDSDB-16	16 MB
SDSDB-32	32 MB
SDSDJ-64	64 MB
SDSDJ-128	128 MB
SDSDJ-256	256 MB
SDSDJ-512	512 MB
SDSDJ-1024	1024 MB

3.2 Características de la tarjeta SD

Almacenamiento de más de 1Gb.

Protocolo SD compatible.

Soporta modo SPI.

Rango de voltaje:

Comandos básicos (CMD0, CMD15, CMD55, CMD41): 2.0-3.6V.

Otros comandos y acceso a la memoria: 2.7-3.6V.

Reloj variable 0-25Mhz.

Transferencia de datos arriba de 12.5Mb/sec (usando 4 líneas de datos paralelos).

Corrección de errores de campo de memoria.

Mecanismo de protección de derechos de copia.

Seguro mecánico de protección de escritura.

Detección de tarjeta

Comandos de aplicación especificas.

3.3 Organización de la memoria

En general cada tarjeta SD esta dividida en 2 áreas separadas como sigue:

Área para el usuario – se usa para almacenar datos seguros o no seguros y puede ser accesada por el usuario con comandos de lectura y escritura.

Área protegida segura – usada por aplicaciones de protección de derechos de copia para asegurar datos relacionados con el fabricante. El tamaño de esta área esta definida por el fabricante, aproximadamente el uno por ciento del tamaño total de la tarjeta. El mecanismo de protección de escritura no afecta esta área.

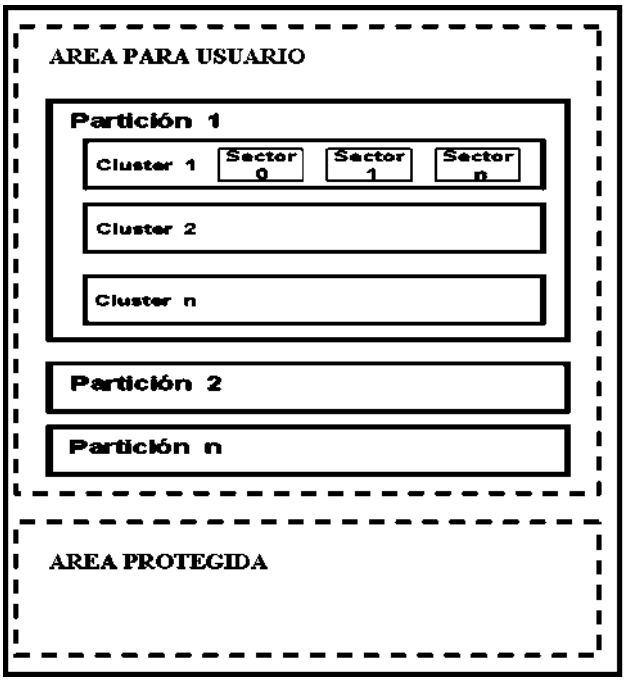


Figura 3.3 Arreglo de particiones de la memoria

Físicamente, la unidad mínima de transferencia en una memoria es el ***sector*** o ***bloque***. Un sector consta de 512 bytes. Sin embargo los sistemas operativos, no operan con sectores, sino con ***clusters***. Un cluster es una agrupación de sectores, y es la unidad más pequeña de almacenamiento en una memoria. Por lo tanto independientemente del número de bytes que ocupe un archivo, siempre ocupara un número entero de clusters en la memoria. Dependiendo del tipo de partición se tendrán uno o más sectores de 512 bytes en un cluster. El tamaño del cluster queda generalmente determinado por el sistema de archivos (FAT, NTFS) y el tamaño de la memoria.

Al leer o escribir algún archivo en la memoria se debe especificar correctamente la dirección del primer bloque del cluster.

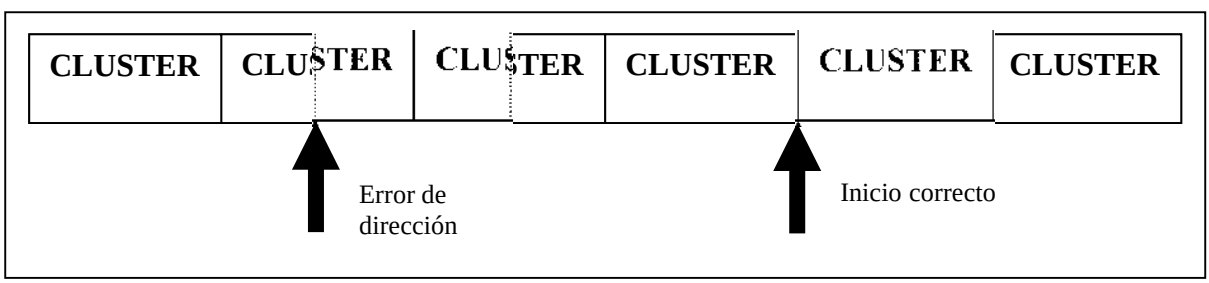


Figura 3.4 Ejemplo de Formato de bloques

3.4 Registros

Los registros son espacios de memoria que contienen información sobre la configuración de la tarjeta. Hay 7 registros dentro de la interfaz de la tarjeta. Los registros OCR, CID, CSD y SCR. El registro RCA contiene la dirección relativa de comunicación para la sesión en uso. Los registros Status y SD Status contienen el protocolo de comunicación relacionado con el estado de la tarjeta.

3.4.1 Registro CID (*Card Identification*)

El registro CID es de 16 bytes y contiene un número único de identificación de la tarjeta como se muestra en la tabla 3.4. Es programado durante la fabricación de la tarjeta y no puede ser cambiado por el usuario. **Este registro contiene la capacidad de la memoria.**

Tabla 3.4 Registro CID

Nombre	**Tipo**	**Tamaño (bits)**	**No. De bit**	**Valor* CID**
Identificación de fabricante (MID)	Binario	8	[127:120]	0x03
Nombre del producto(PNM)	ASCII	40	[103:64]	SD128, SD064, SD032, SD016
Revisión del producto (PRV)	BCD	8	[63:56]	(30)
Numero de serie (PSN)	Binario	32	[55:24]	Numero serial del producto
Reservado		4	[23:20]	
Código de fecha de fabricación	BCD	12	[19:8]	Ejemplo: Apr 2001 = 0x014
CRC	Binario	7	[7:1]	CRC7
No usado, siempre es 1		1	[0:0]	

*Valores tomados como ejemplo para tarjetas marca SandDisk

La interfase de la tarjeta SD permite la fácil integración a cualquier diseño, independientemente del microprocesador usado. Para compatibilidad con controladores existentes, la memoria SD ofrece, además de la interfase SD, un protocolo de comunicación alterno, el cual esta basado en el estándar SPI.

Además del chip de memoria flash de almacenamiento masivo, la tarjeta SD incluye un controlador inteligente que maneja protocolos de interfaz, algoritmos de seguridad para protección de derechos de autor, almacenaje de datos y recuperación, así como algoritmos de Código de Corrección de Error (ECC), defecto de manejo y diagnóstico, manejo de energía y control de reloj.

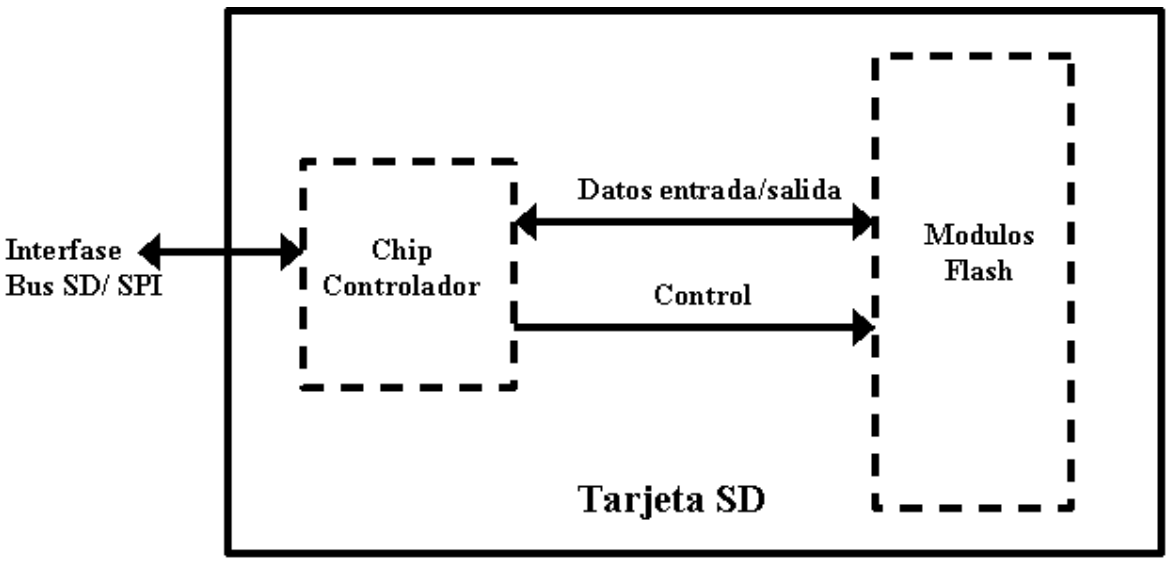


Figura 3.5 Diagrama de bloques de la tarjeta SD

La tarjeta SD contiene un sub-sistema inteligente de alto nivel. Este sub-sistema (microprocesador) provee muchas capacidades no encontradas en otros tipos de tarjetas de memoria.

3.5 Descripción del protocolo SD card

La comunicación sobre el bus SD esta basado en comandos y datos “bit -stream”, los cuales son iniciados por un bit de inicio y terminado por un bit de paro:

Comando – Un comando es una señal que comienza una operación. Un comando es mandado por el controlador a la tarjeta y es transferido en serie por la línea CMD.

Respuesta – una respuesta es una señal que es mandada por la tarjeta hacia el controlador como una respuesta a un comando recibido y también es transferida por la línea CMD.

Dato – Un dato puede ser transferido de la tarjeta al controlador o viceversa. Los datos son transferidos a través de las líneas de datos.

La transferencia de datos de y hacia la tarjeta SD es en bloques. Los bloques de datos son siempre seguidos de bits CRC (Control de Redundancia Cíclica) que es un mecanismo de detección de errores en sistemas digitales. La transferencia de datos puede configurarse por el controlador para usar una simple línea o múltiples líneas de datos (siempre y cuando la tarjeta tenga esa característica).

3.6 Definición del protocolo SPI

Mientras que el canal SD esta basado en comandos y datos ”bit -stream”, el canal SPI esta orientado a bytes. Cada comando o bloque de datos esta hecho en multiplos de 8 bits.

Al igual que en el protocolo SD card, la comunicación en el protocolo SPI se basa en señales de comando, respuesta y bloques de datos. Toda comunicación entre controlador (maestro) y tarjeta es dirigida por el controlador. El controlador inicia cada transacción en el bus con una señal en bajo en la línea CS.

El comportamiento de respuesta del modo bus SPI difiere del modo bus SD en las siguientes formas:

La tarjeta seleccionada siempre responde al comando.

Se usa una estructura de respuesta de 8 o 16 bits.

Cuando una tarjeta encuentra un problema de dato, responderá con una respuesta de error (que reemplaza al bloque de dato esperado) en lugar de un tiempo fuera como en el modo bus SD.

Además del comando de respuesta, cada bloque de datos enviados a la tarjeta durante operaciones de escritura tiene como respuesta un dato especial.

3.6.1 Formato De Comando

Todos los comandos son de 6 bytes y se transmite primero el bit mas significativo (MSB).

Tabla 3.5 Formato de comandos

Byte 1			**Byte 2 – 5**	**Byte 6**	
7	6	5 0	31 0	7	0
0	1	Comando	Argumento	CRC	1

Los comandos necesarios para inicializar la tarjeta y para funciones de lectura y escritura se muestran en la tabla 3.6.

Tabla 3.6 Comandos utilizados

Comando	**Descripción**	**Argumento**
CMD0	Reinicia la tarjeta y entra en modo de espera	No son necesarios
CMD1	Activa el proceso de inicialización de la tarjeta	No son necesarios
CMD10	Lee el registro CID que es la identificación de la tarjeta	No son necesarios
CMD16	Selecciona la longitud definida para un bloque	Longitud en bytes que necesitamos para un bloque
CMD17	Lee un solo bloque del tamaño seleccionado con el comando anterior	Dirección del bloque que se quiera leer. ARG = bloque*longitud de bloque
CMD24	Escribe un solo bloque del tamaño seleccionado con el comando 16	Dirección del bloque que se quiera escribir. ARG = bloque*longitud de bloque

La longitud de un bloque para lectura y escritura, así como su dirección de inicio debe ser seleccionada de tal forma que no sea mayor que la longitud de bloque seleccionada o no comience la operación a mitad del bloque.

3.6.2 Formato de respuesta R1

La tarjeta siempre responderá a cualquier comando con 1 byte, los comandos mencionados tienen el formato de respuesta R1. El bit mas significativo siempre será 0 los demás bits son indicaciones de error cuando son 1.

Bits

7	Siempre 0.
6	Error de parámetro.
5	Error de dirección.
4	Error de secuencia de borrado.
3	Error de CRC.
2	Comando Ilegal.
1	Error en la secuencia de borrado.
0	La tarjeta esta en modo de espera y proceso de inicialización.

Capítulo 4

Sistema de archivos FAT16

Durante años, el software de componentes ha evolucionado para soportar dispositivos con nuevas y mayores capacidades. Windows y otros sistemas operativos soportan variedad de sistemas de archivos de forma que PCs y otros sistemas pueden acceder sin problemas al mismo dispositivo. Uno de los sistemas de archivos mas usados es el sistema de archivos FAT (*File Allocation Table*). El termino FAT se refiere a la familia de sistema de archivos, así como a la tabla de asignación de archivos que contiene cada sistema FAT. FAT16 y FAT32 son los 2 sistemas FAT más comunes.

El sistema FAT32 tiene una implementación mas complicada que FAT16. El sistema FAT16 es comúnmente encontrada en dispositivos de memoria portátiles y puede ser leída en Windows sin necesidad de software adicional. El sistema FAT16 permite funcionalidad con cualquier dispositivo de memoria actualmente (hasta 2 GB) en el mercado y la complejidad de su interfaz es mínima, aunque para mantener su simplicidad, sacrifica otras características como seguridad.

Todos los dispositivos formateados con FAT16 tienen los siguientes componentes:

- Región reservada, la cual contiene el sector de arranque (Boot Sector).
  El sector de arranque comienza en el valor LBA (*Logical Block Addressing* o Dirección Lógica de Bloques) guardado en el MBR (Master Boot Record) de la tabla de partición. En dispositivos que no tienen MBR, el sector de arranque es el primer sector.
- Región FAT.
  Después de la región reservada hay dos copias idénticas de la FAT. Una tabla FAT16 tiene entradas de 16 bits para cada cluster de datos. La segunda tabla es un respaldo para reparar algún posible daño en la primera copia.

- Región del directorio raíz (Root Directory).

  Los sectores de este directorio comienzan después de las FATs. El direct orio raíz puede almacenar 512 entradas de 32 bytes cada una.
- Región de archivos y directorio de datos, aloja archivos y subdirectorios .

  Los sectores siguientes del director io raíz están agrupados en clusters. Un cluster de datos puede consistir de uno o más sectores. Como las dos primeras entradas de la tabla FAT están destinadas para el nombre del volumen, el primer cluster de datos es el cluster 2.

El formato básico de un sistema de archivos FAT se puede visualizar en la siguiente figura:

Figura 4.1 Formato FAT en un dispositivo de memoria

4.1 Master Boot Record

Un dispositivo de almacenamiento puede estar formateado con un Master Boot Record y hasta cuatro particiones primarias. El MBR esta situado siempre al principio de la memoria, en el primer sector (sector cero).

Tabla 4.1 Formato de particiones y el MBR

MASTER BOOT RECORD
PRIMERA PARTICION PRIMARIA
SEGUNDA PARTICION PRIMARIA (OPCIONAL)
TERCERA PARTICION PRIMARIA (OPCIONAL)
CUARTA PARTICION PRIMARIA (OPCIONAL)

Algunos dispositivos no tienen el sector de MBR ya que no es necesario gastar 512 bytes en un sector para el MBR en dispositivos que requieren solo una partición. Estos dispositivos inician con el Boot Sector de la partición. Sin embargo para una máxima compatibilidad debería incluir un sector para el Master Boot Reco rd. **Una tarjeta SD esta formateada de fábrica con MBR.**

En el MBR contiene tres áreas: el código ejecutable, la tabla de particiones y marca ejecutable.

Tabla 4.2 Primer sector físico de la memoria

512 bytes	446 bytes	Código ejecutable
	64 bytes	Tabla de particiones
	2 bytes	Marca ejecutable (55h AAh)

Código ejecutable. Al iniciar el sistema operativo busca el código en la tabla de particiones por una partición activa, y al encontrar una inicia el sistema con un código de arranque alojado en el primer sector de la partición.

Tabla de particiones. El MBR tiene espacio para 4 entradas de 16 bytes que especifican los sectores que pertenecen a cada partición y el inicio de esta. La tabla 4.3 muestra el contenido de una entrada.

Marca ejecutable. Indica el final del sector del MBR y siempre es 55h AAh

Tabla 4.3 Entrada de la tabla de particiones en el Master Boot Record

Posición hexadecimal	**Descripción**	**Tamaño**
0x00	Estado actual de la partición	1 byte
0x01	Principio de la partición – Cabeza/Cilindro/Sector	3 bytes
0x04	Tipo de partición	1 byte
0x05	Fin de la partición - Cabeza/Cilindro/Sector	3 bytes
0x08	Numero de sectores entre el MBR y la partición	4 bytes
0x0C	Numero de sectores en la partición	4 bytes

Estado de la partición: si el bit 7 esta activo (“1” logico) es una partición activa, los otros 7 bits deben ser ceros, es decir, un valor de 0x80 es una partición activa y 0x00 inactiva, cualquier otro valor en este campo no tiene significado.

El tipo de partición define el formato que tiene u na partición, por ejemplo:
00 - Partición vacía
01 - Partición FAT12
04 - Partición FAT16
Entre otros.

4.2 Boot Record de la FAT16

La primera región en una partición FAT16 es la región reservada, la cual consiste de un solo sector llamado Boot Record. Los primeros 62 bytes en el Boot Record contiene el bloque de parámetros BIOS (BPB), un área reservada para código de arranque y una marca ejecutable.

Bloque de Parámetros BIOS. Primeros 62 bytes en el Boot Record. La información en el BPB permite al sistema localizar el Directorio Raíz, número de sectores por cluster, numero de sectores por FAT y demás información sobre el formato de la tarjeta.

Código de arranque. Como se explico en la sección anterior, el código ejecutable del MBR salta a una partición activa en busca del código de arranque en el Boot Record. El código de arranque carga el sistema operativo.

Marca ejecutable. Indica el final del sector del Boot Record.

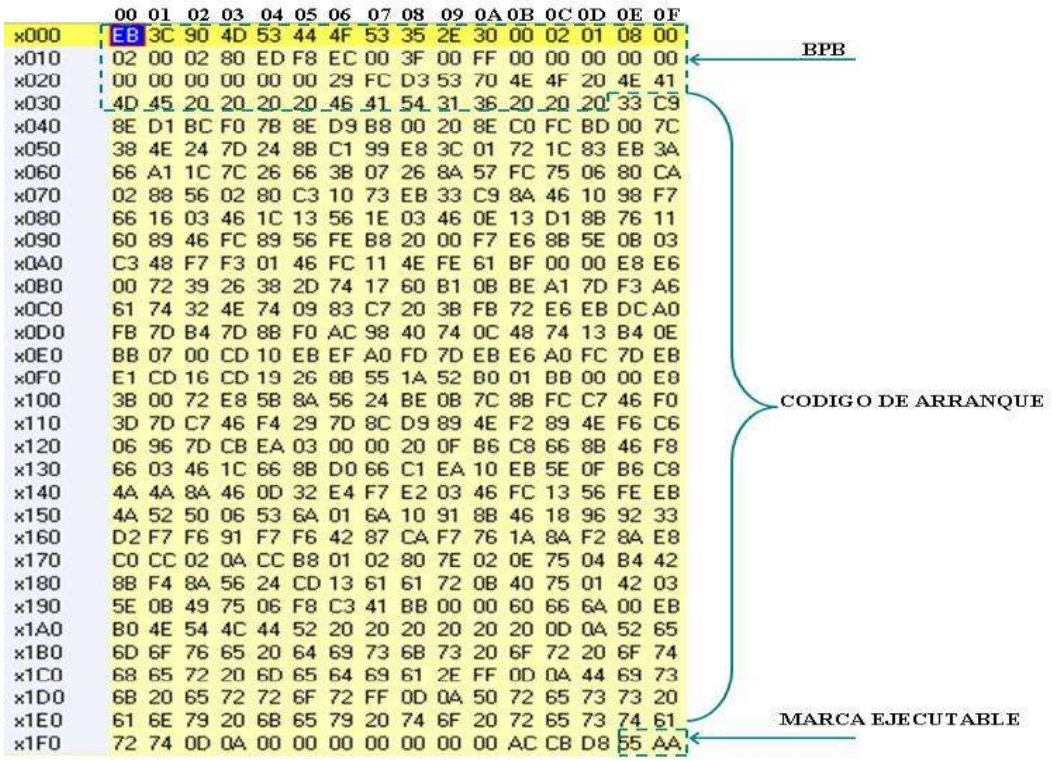


Figura 4.2 Ejemplo del primer sector (Boot Record)

Tabla 4.4 Descripcion del Boot Record

Posición	**Descripción**	**Tamaño (Bytes)**	
00h	Instrucción para salto a código de arranque	3	Si el byte 0 contiene un EBh o E9h es una partición ejecutable
03h	Nombre del sistema operativo que formateo la tarjeta	8	Ejemplo de sistema operativo MSDOS 5.0
0Bh	Numero de bytes por sector	2	512 bytes por sector es lo mas común para mayor compatibilidad
0Dh	Numero de sectores por cluster	1	Se permiten valores de 1, 2, 4, 8, 16, 32, 64 y 128
0Eh	Numero de sectores reservados	2	8 sectores reservados ubicados después del Boot Record
10h	Numero de copias de la FAT	1	2, la tabla original y una copia de respaldo
11h	Máximo numero de entradas en el directorio raíz	2	512 entradas de 32 bytes cada una
13h	Numero de sectores en una partición menor de 32 MB	2	60800 sectores de 512 bytes en una partición de 32 MB
15h	Descripción del dispositivo	1	
16h	Sectores por FAT	2	Numero de sectores en una tabla FAT
18h	Sectores por track	2	No usado en LBA
1Ah	Numero de cabezas	2	No usado en LBA
1Ch	Numero de sectores ocultos en la partición	4	Numero de sectores ocultos que preceden a la partición
20h	Numero de sectores en la partición	4	Numero total de bytes en una partición mayor de 32 MB
24h	Especificación del sistema operativo	2	
26h	Firma de arranque extendida	1	29h
27h	Numero de serie de la partición	4	Creado usando la fecha y hora en que se dio formato
2Bh	Nombre de volumen de la partición	1	Texto que identifica el volumen.
36h	Tipo del sistema de archivos (FAT16)	8	FAT16 o FAT
3Eh	Códigos de arranque	448	
1FEh	Maraca ejecutable	2	55h AAh

4.3 Tabla FAT

Es una lista de todos los archivos y su posición física en el medio de almacenamiento. Esta lista o tabla esta formada con elementos que corresponden a los clusters en una memoria. Es decir, el elemento situado en la posición 40 de la tabla corresponde al cluster número 40 de la memoria. Cada elemento de la tabla tiene uno de los tres valores siguientes:

- 0x0000 para indicar que se trata de un cluster libre.
- 0xFFFF para indicar que se trata del último cluster de un archivo; es decir, que los sectores de ese cluster almacenan la parte final del archivo.
- Cualquier otro número se interpreta como el siguiente cluster del archivo.

Si por cualquier motivo se corrompe la tabla FAT posiblemente se perderá gran parte de los datos de la memoria, ya que el sistema operativo no sabrá dónde continúa un archivo y donde termina. De igual forma cuando algún archivo es modificado en cuanto a su tamaño, es necesario actualizar la tabla FAT. Es tal la importancia de la FAT, que normalmente se guarda una copia de la FAT original para poder recuperar los datos en caso de que se corrompa una de las copias.

4.3.1 Cadena FAT (*Chain FAT*)

La serie de clusters usados por un archivo la llamamos cadena FAT. Si un archivo en una partición FAT16 requiere solo de un cluster, la entrada en la tabla FAT para ese cluster contiene un valor de FFF8h a FFFFh. Estos valores, llamados marcadores de fin de cadena, indican que ese cluster es el último cluster usado por un archivo.

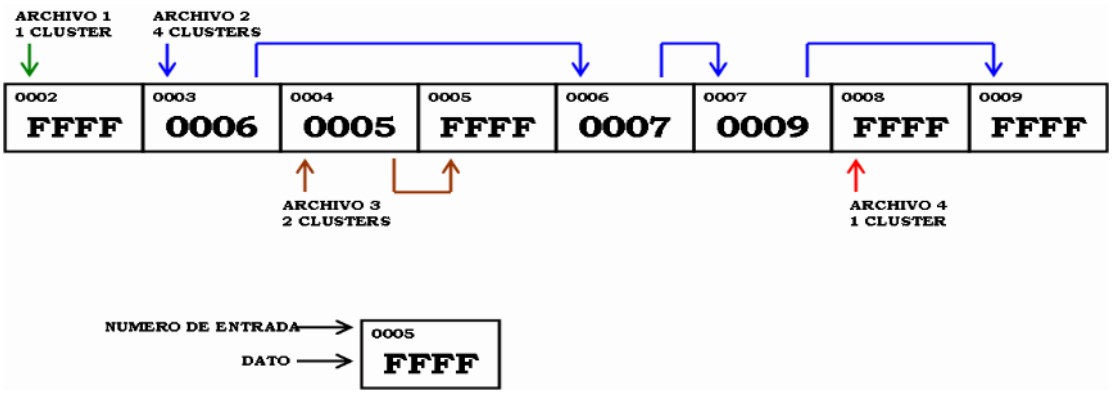


Figura 4.3 Ejemplo de cadena FAT

De forma similar si un archivo requiere dos o más clusters, la entrada d e la tabla FAT para cada cluster contiene el número del siguiente cluster del archivo a excepción del último cluster que contiene un marcador de fin de ca dena. Por ejemplo: un archivo “A” que comienza en el cluster 4 y continúa en el 7, 8, 10 y termina en el cluster 15. La tabla de directorio tendrá un 4 para el cluster de inicio del archivo “A”. después en la tabla FAT en la posición 3 tendrá la información 7, la posición 7 tendrá un 8, así continua cada posición en la cadena hasta la ultima posición que ocupa el archivo en este caso la posición 15 contiene un Fin De Cadena “FFh FFh”. Los valores contenidos en cada posición están en formato hexadecimal y el fin de cadena se muestra <eof> (end of file)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
x000	<eof>	<eof>	0003	0005	0007	0006	0009	0008	000A	000B	000F	000C	000D	000E	0010	<eof>
x020	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F	0020
x040	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D	002E	002F	0030
x060	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F	0040
x080	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F	0050
x0A0	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F	0060
x0C0	0061	0062	0063	0064	0065	0066	0067	0068	0069	006A	006B	006C	006D	006E	006F	0070
x0E0	0071	0072	0073	0074	0075	0076	0077	0078	0079	007A	007B	007C	007D	007E	007F	0080
x100	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F	0090
x120	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F	00A0
x140	00A1	00A2	00A3	00A4	00A5	00A6	00A7	00A8	00A9	00AA	00AB	00AC	00AD	00AE	00AF	00B0
x160	00B1	00B2	00B3	00B4	00B5	00B6	00B7	00B8	00B9	00BC	<eof>	<eof>	00BD	00BE	00BF	00C0
x180	<eof>	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
x1A0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
x1C0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
x1E0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
x000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Figura 4.4 Ejemplo del primer bloque de la tabla FAT

La tabla FAT tiene 256 entradas por cada sector , debido a que es FAT16 cada entrada de la tabla es de 2 bytes. El número de sectores que utiliza una tabla FAT se puede leer en el Boot Record en la posición 16h (Sectores por FAT).

4.4 Directorio Raíz

El directorio raíz es una región de la memoria SD donde se encuentran registrados todos los archivos guardados en la memoria. El directorio raíz muestra in formación del archivo como el nombre, la extensión, donde inician los datos del archivo y su tamaño.

Para saber la ubicación del directorio raíz se utiliza la siguiente ecuación: inicio de la partición + numero de sectores reservados + numero de sectores por FAT * numero de FATs. La información de estos parámetros se puede saber directamente del BR. La tabla del directorio contiene todas las direcciones de los archivos en el orden en que se van anexando. Cada entrada es de 32 bytes y su estructura es como sigue.

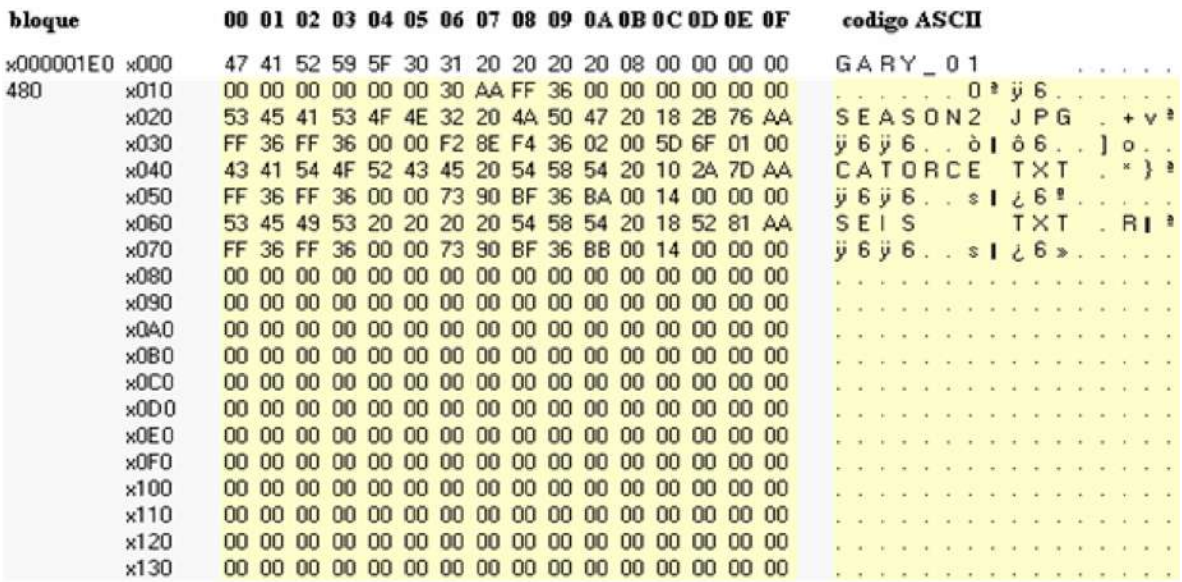
Tabla 4.5 Entrada de archivo en el Directorio Raíz

Posición relativa	**Descripción**	**Tamaño**
0x00	Nombre del archivo	8 bytes
0x08	Extensión del archivo	3 bytes
0x0B	Atributo	1 byte
0x16	Hora	2 bytes
0x18	Fecha	2 bytes
0x1A	Cluster de inicio	2 bytes
0x1C	Tamaño del archivo	4 bytes

Las posiciones de 0x09 – 0x0A son bytes reservados.

Si el primer byte de una entrada del Directorio Raíz contiene un valor E5 significa que el archivo correspondiente a esa entrada fue borrado y esa entrada puede ser reescrita con información de otro archivo.

Dentro del primer bloque del directorio raíz, la primera entrada de 32 bytes contiene el volumen de la memoria, únicamente en el primer bloque.



bloque		00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	codigo ASCII
x000001E0	x000	47	41	52	59	5F	30	31	20	20	20	20	08	00	00	00	00	GARY_01
480	x010	00	00	00	00	00	00	30	AA	FF	36	00	00	00	00	00	00	0ª ÿ6
	x020	53	45	41	53	4F	4E	32	20	4A	50	47	20	18	2B	76	AA	SEASON2 JPG +vª
	x030	FF	36	FF	36	00	00	F2	8E	F4	36	02	00	5D	6F	01	00	ÿ6ÿ6 ò ô6]o
	x040	43	41	54	4F	52	43	45	20	54	58	54	20	10	2A	7D	AA	CATORCE TXT *}ª
	x050	FF	36	FF	36	00	00	73	90	BF	36	BA	00	14	00	00	00	ÿ6ÿ6 s ¿6º
	x060	53	45	49	53	20	20	20	20	54	58	54	20	18	52	81	AA	SEIS TXT R ª
	x070	FF	36	FF	36	00	00	73	90	BF	36	BB	00	14	00	00	00	ÿ6ÿ6 s ¿6»
	x080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x0C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x0F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
	x130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Figura 4.5 Ejemplo de inicio del directorio raíz

La imagen muestra como están ordenados los archivos en el primer bloque del directorio raíz. Se observa el volumen de la memoria y tres archivos guardados en la memoria. A la izquierda se ve el número de bloque en la memoria que corresponde al primer bloque del directorio raíz y el número de byte del bloque de 512 bytes. Se muestra el contenido de cada byte en hexadecimal y en código ASCII en la parte derecha de la imagen.

Tabla 4.6 Distribución de un archivo en el directorio raíz

Descripción	Posición en el bloque	Contenido en hexadecimal	Valor
Nombre	20h – 27h	53 45 41 53 4F 4E 32 20	SEASON2
Extensión	28h – 2Ah	4A 50 47	JPG
Atributo	2Bh	20	
Hora	36h – 37h	F2 8E	
Fecha	38h – 39h	F4 36	
Cluster de inicio	3Ah – 3Bh	00 02	Cluster 2
Tamaño del archivo	3Ch – 3Fh	00 01 6F 5D	94045 Bytes

Ejemplo del primer archivo mostrado en la figura 4.5

En el caso del cluster de inicio y tamaño de archivo los bytes se toman de derecha a izquierda , es decir el byte menos significativo para el valor, es el primer byte del contenido en hexadecimal.

Capítulo 5

El microcontrolador

Un microcontrolador es un circuito integrado que incluye en su interior las tres unidades funcionales de una computadora: CPU, memoria y Unidades de E/ S, es decir, se trata de una computadora completa en un solo circuito.

Un microcontrolador típico tiene un generador de reloj integrado y una pequeña cantidad de memoria RAM y ROM, significando que para hacerlo funcionar, todo lo que se necesita son unas pocas instrucciones en programas de control y un cristal de sincronización.

El microcontrolador utilizado es de la familia PIC18. PIC (Controlador de Interfaz Periférico) son una familia de microcontroladores fabricados por Microchip Technology Inc.

Para transferir el código de una computadora al microcontrolador se usa un dispositivo llamado programador. Microchip proporciona un entorno de desarrollo freeware llamado MPLAB que incluye un simulador software y un ensamblador. Microchip también vende compiladores para los PICs de gama alta ("C18" para la serie F18).

5.1 PIC18F458

La elección de este tipo de microcontrolador se debe, además de su buena relación calidad/precio, a su capacidad de memoria RAM para transferencia de datos ya que cuenta con un puerto serie (SPI) necesario para la comunicación con la tarjeta SD y un puerto USART para comunicación serie.

La familia PIC18 cuenta con el soporte de los sistemas de desarrollo de altas prestaciones de Microchip, entre ellos el entorno de desarrollo integrado MPLAB; el compilador C18 y el depurador MPLAB ICD 2.

A continuación se da una breve descripción de los módulos utilizados del microcontrolador.

5.1.1 MSSP (Puerto Serial Síncrono Maestro)

Este modulo es una interfaz serial útil para comunicación con otros dispositivos controladores o periféricos. El modulo MSSP puede operar en alguno de estos dos modos:

Interfaz periférica serie (SPI)

Circuito inter-integrado (I^2C)

-Modo maestro

-Modo esclavo

El modo SPI funciona con 8 bits de datos para ser transmitidos y recibidos simultáneamente. Para completar la comunicación, normalmente se usan 3 pines.

Serial data out (SDO) – RC5/SDO	Datos de salida
Serial data in (SDI) – RC4/SDI	Datos de entrada
Serial clock (SCK) – RC3/SCK	Reloj

Adicionalmente, se usa un 4to pin cuando se opera en modo esclavo. A este pin se le denomina Slave select.

5.1.2 Registros de operación del puerto MSSP

Dependiendo del modo en el que este operando el modulo MSSP, ya sea modo SPI o modo I^2C, el uso de los registros y su configuración individual difieren significativamente.

El modulo MSSP cuenta con 4 registros para operación de puerto serie en modo SPI:

MSSP registro de control 1 (SSPCON1)

MSSP registro de estado (SSPSTAT)

Buffer de recepción/transmisión (SSPBUF)

MSSP registro de intercambio (SSPSR) – no accesible directamente

SSPCON1 y SSPSTAT son los registros de control y estado en modo de operación SPI. El registro SSPCON1 permite lectura y escritura. Los 6 bits menos significativos del registro SSPSTAT son de solo lectura y los 2 bits más significativos son de lectura y escritura. SSPBUF es el registro buffer al cual se escribe o desde el cual se lee.

En la recepción, los registros SSPSR y SSPBUF juntos crean un receptor de doble buffer. Cuando SSPSR recibe un byte completo, este es transferido al SSPBUF y la bandera SSPIF se pone en 1.

Descripción del registro SSPCON1: Registro de Control MSSP

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	**SSPOV**	**SSPEN**	**CKP**	**SSPM3**	**SSPM2**	**SSPM1**	**SSPM0**
bit 7							bit 0

R = Bit de Lectura
W = Bit de Escritura
0 = Bit en cero

bit 7 **WCOL:** Bit de detección de colisión de escritura (solo en transmisión)

1 = Se escribe en el registro SSPBUF mientras aun esta transmitiendo.

0 = No hay colisión

bit 6 **SSPOV**: Bit indicador de sobre flujo en recepción

1 = Un byte nuevo es recibido mientras el registro SSPBUF aun tiene datos. El sobreflujo solo ocurre en modo esclavo

0 = No hay sobre flujo

bit 5 **SSPEN**: Bit de puerto serie síncrono habilitado

1 = Habilita el puerto serie y configura SCK, SDO, SDI como pines de puerto serie.

0 = Deshabilita el puerto serie y configura los pines como de entrada y salida

bit 4 **CKP**: Bit de selección de polaridad del reloj

1 = Estado de espera del reloj es en nivel alto

0 = Estado de espera del reloj es en nivel bajo

bit 3-0 **SSPM3:SSPM0**: Bits de selección de modo del puerto serie síncrono

0101 = SPI modo esclavo, reloj = pin SCK, pin SS control deshabilitado

0100 = SPI modo esclavo, reloj = pin SCK, pin SS control habilitado

0011 = SPI modo maestro, reloj = TMR2 salida/2

0010 = SPI modo maestro, reloj = Fosc/64

0001 = SPI modo maestro, reloj = Fosc/16

0000 = SPI modo maestro, reloj = Fosc/4

Descripción del registro SSPSTAT: Registro de Estado MSSP

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	**CKE**	**D/A**	**P**	**S**	**R/W**	**UA**	**BF**
bit 7							bit 0

R = Bit de Lectura
W = Bit de Escritura
x = Bit desconocido
0 = Bit en cero

bit 7 **SMP:** Bit de muestreo

SPI modo esclavo:

SMP debe ser cero cuando SPI es usado en modo esc lavo

bit 6 **CKE**: Bit de selección de transmisión para reloj SPI

1 = La transmisión ocurre en la transición de activo a espera del estado del reloj

0 = La transmisión ocurre en la transición de espera a activo del estado del reloj

bit 5 **D/A**: Bit de Datos/Dirección

Usado solo en modo I^2C

bit 4 **P**: Bit de paro

Usado solo en modo I^2C. Este bit se pone en cero cuando le modulo MSSP esta deshabilitado

bit 3 **S**: Bit de arranque

Usado solo en modo I^2C

bit 2 **R/W**: Bit de información de Lectura/Escritur a

Usado solo en modo I^2C

bit 1 **UA**: Bit de actualización de dirección

Usado solo en modo I^2C

bit 0 **BF:** Bit de estado de buffer lleno (modo de recepción solamente)

1 = Recepción completa, SSPBUF esta lleno

0 = Recepción no completa, SSPBUF esta vac io

5.2 Conexión típica

La figura muestra una conexión entre 2 microcontroladores utilizando el puerto MSSP. El controlador maestro (procesador 1) inicia la transferencia de datos mandando la señal de reloj SCK. Ambos procesadores deben estar programados con la misma polaridad del reloj (CKP), así los controladores mandaran y recibirán datos al mismo tiempo.

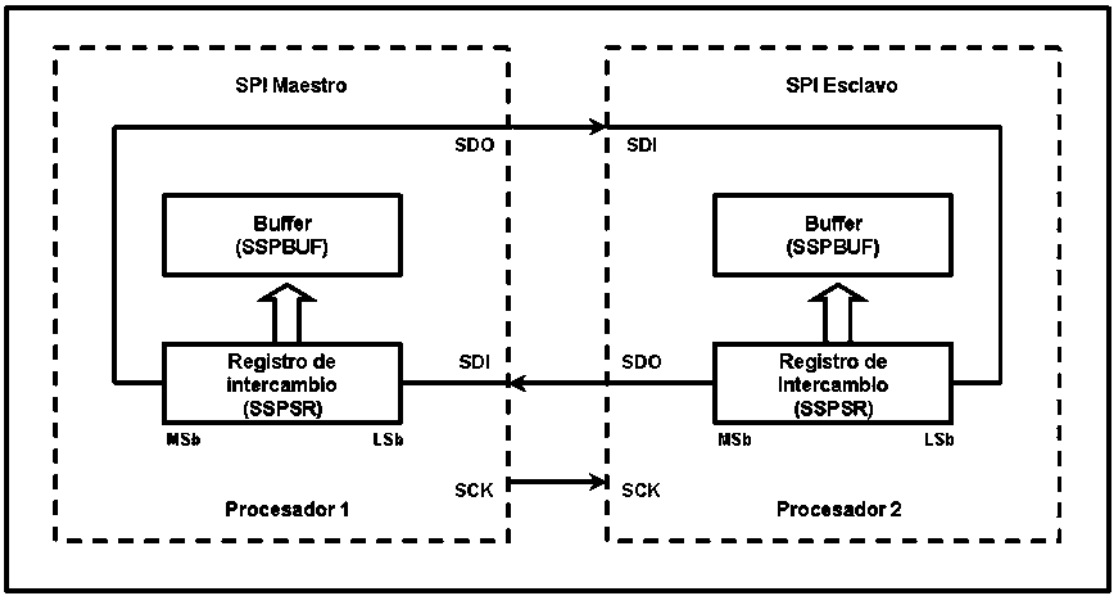


Figura 5.1 Conexión entre dos microcontroladores

5.3 Modo maestro

El microcontrolador maestro puede iniciar la transferencia de datos en cualquier momento porque el controla el reloj (SCK). El maestro determina cuando el esclavo debe enviar datos según el protocolo utilizado. En modo maestro, los datos son transmitidos/recibidos tan pronto como se escriba en el registro SSPBUF. Si el SPI solo va a recibir, la salida SDO puede ser deshabilitada (programándola como entrada).

La polaridad del reloj es seleccionada programando apropiadamente el bit CKP (SSPCON1 <4>) y el bit CKE (SSPSTAT <6>) define si la transmisión ocurre en la transición de alto-bajo o de bajo-alto del reloj. Esto dará formas de onda para la

comunicación SPI como se muestra en la figura. En el modo maestro la velocidad del reloj del SPI se puede programar para las siguientes velocidades:

Fosc/4 (o TCY)

Fosc/16 (o 4*TCY)

Fosc/64 (o 16*TCY)

Timer2 output/2

Esto permite una velocidad de transferencia máxima (a 40 MHz) de 10.00 Mbps.

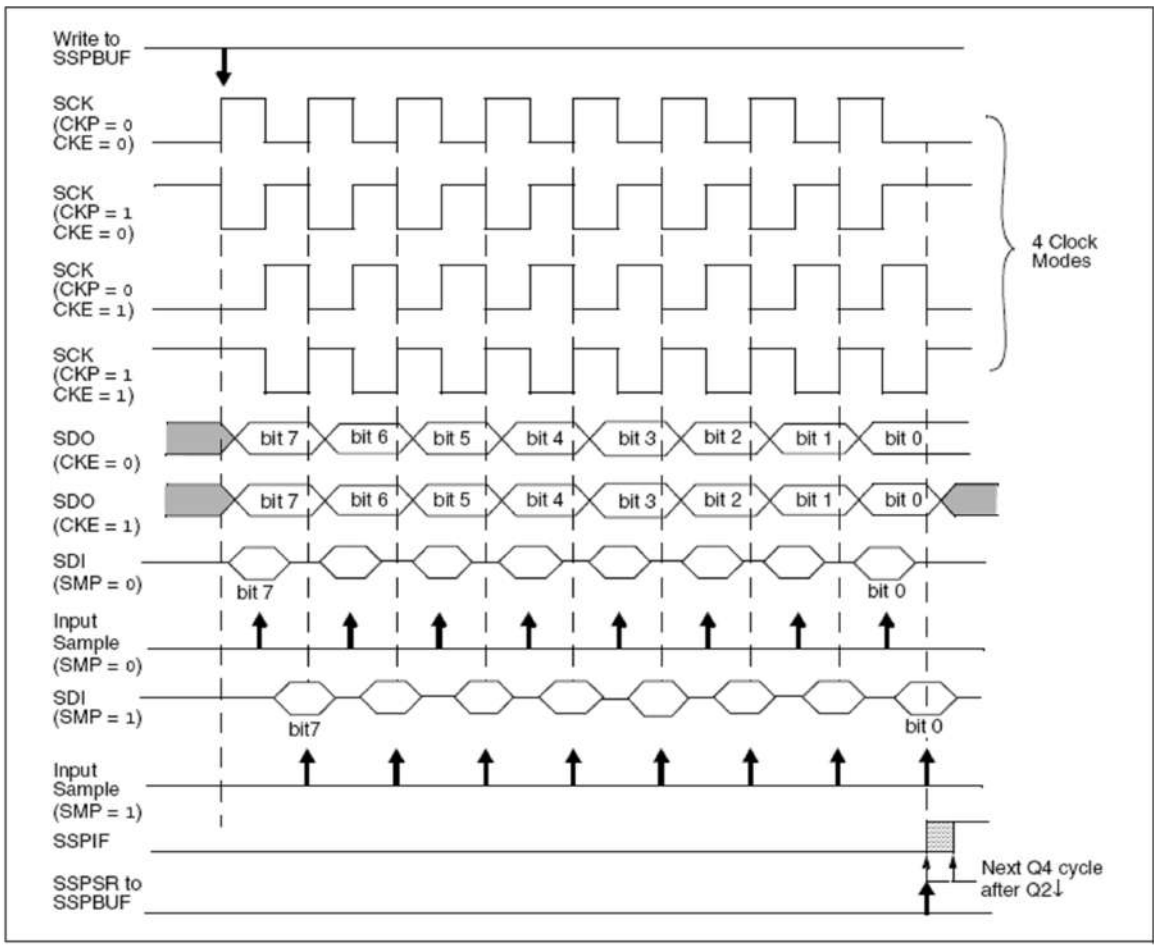


Figura 5.2 Formas de onda en modo SPI

5.4 Puerto serie USART

El modulo Universal de Transmisión Recepción Síncrona Asíncrona *(Universal Synchronous Asynchronous Receiver Transmitter).* Este modulo de comunicacion serie de E/S del microcontrolador. El puerto serie USART puede ser configurado como sistema asíncrono que se puede comunicar con dispositivos periféricos como computadoras personales, o puede ser configurado como sistema síncrono que puede comunicar con dispositivos periféricos como circuitos integrados A/D o D/A, memorias seriales, etc.

El modulo USART cuenta con 3 registros para su configuración.
TXSTA es el registro de control y estado de transmisión.
RCSTA es el registro de control y estado de recepción.
SPBRG controla la velocidad de transmisión en modo asíncrono.

Descripción del registro TXSTA: Registro de Control y Estado de Transmisión

R/W	R/W	R/W	R/W	U	R/W	R	R/W
CSRC	**TX9**	**TXEN**	**SYNC**	**--**	**BRGH**	**TRMT**	**TX9D**
Bit 7							Bit 0

R = Bit de Lectura

W = Bit de Escritura

U = no se utiliza = ‘0’

bit 7 **CSRC:** Bit de Selección del Reloj

Modo Asíncrono:

No importa.

Modo Síncrono:

1 = Maestro (pulsos de reloj interno generado por BRG)

0 = Esclavo (pulsos de reloj generados externamente)

bit 6 **TX9**: Permiso de Transmisión de 9 bits

1 = Habilita

0 = Deshabilita

bit 5 **TXEN**: Permiso de Transmisión

1 = Habilitado

0 = Deshabilitado

Note: SREN/CREN maneja TXEN en modo SYNC

bit 4 **SYNC:** Selector de modo USART

1 = Síncrono

0 = Asíncrono

bit 3 **No implementado:** Se lee como '0'

bit 2 **BRGH**: Selector de Alto Rango de Baudios

Modo Asíncrono:

1 = Alta velocidad

0 = Baja Velocidad

Modo Síncrono:

No se utiliza en este modo

bit 1 **TRMT**: Bit de Estado de Transmisión

1 = TSR vacío

0 = TSR lleno

bit 0 **TX9D:** 9^o Dato de transmisión, puede ser bit de paridad

Descripción del registro RCSTA: Registro de Control y Estado de Recepción

R/W	R/W	R/W	R/W	R/W	R	R	R-x
SPEN	**RX9**	**SREN**	**CREN**	**SDDEN**	**FERR**	**OERR**	**RX9D**
Bit 7							Bit 0

R = Bit de Lectura

W = Bit de Escritura

x = Bit desconocido

bit 7 **SPEN:** Bit de encendido del Puerto Serie

1 = Puerto serie encendido

0 = Puerto serie apagado

bit 6 **RX9**: Habilita Recepción de 9 bits

1 = Selecciona recepción de 9 bit

0 = Selecciona recepción de 8 bits

bit 5 **SREN**: Permiso de Recepción Sencilla

Modo Asíncrono:

No importa

Modo Síncrono Maestro:

1 = Permite la recepción sencilla

0 = No permite la recepción sencilla

Modo Síncrono Esclavo:

No importa

bit 4 **CREN**: Recepción Continua

Modo Asíncrono:

1 = Permite la recepción continua

0 = No permite la recepción continua

Modo Síncrono:

1 = Permite la recepción continua solo si CREN = 0

0 = No permite la recepción continua

bit 3 **ADDEN**: Detección de Dirección

Modo Asíncrono de 9 bits (RX9 = 1):

1 = Permite la detección de dirección, permite la interrupción y la carga del buffer de recepción cuando RSR<8>

0 = No permite la detección de dirección, todos los bytes son recibidos, el noveno bit puede ser usado como paridad

bit 2 **FERR**: Error de Encuadre

1 = Error de encuadre

0 = No hay error de encuadre

bit 1 **OERR**: Error Sobreescritura

1 = Error sobrante (puede ser limpiado poniendo en cero CREN)

0 = No hay error por sobreescritura

bit 0 **RX9D:** 9^o bit de Recepción de Datos

Capítulo 6

El sistema desarrollado

En este Capítulo se describe el desarrollo del hardware y software del sistema. Para la construcción del hardware se tomo como base el diseño de un circuito depurador construido durante la clase de procesamiento digital de señales al cual se le conecto una base o zócalo para tarjeta SD, tanto el microcontrolador como la tarjeta SD funcionan en un rango bajo de voltaje a lo que se podría decir que el sistema es de consumo mínimo. El sistema esta diseñado para utilizar tarjetas de 16 MB hasta 2GB de memoria, extraíbles y compatibles en otros S.O.

El sistema es capaz de almacenar archivos en la memoria, así como obtener una lista o directorio de los archivos contenidos en esta. Utiliza una interfaz del tipo serie RS -232 para la conexión con un dispositivo o sistema mínimo. Este dispositivo se encargara de dar instrucciones al sistema para almacenar o leer los datos ya almacenados en la memoria.

La figura 6.1 muestra la conexión entre un dispositivo o sistema mínimo y el sistema desarrollado en este proyecto.

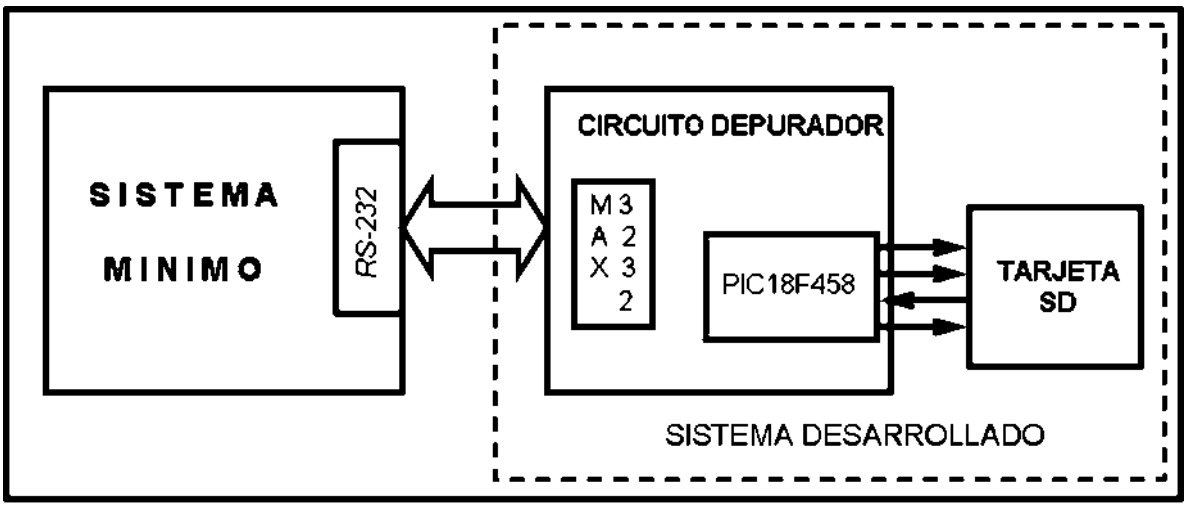


Figura 6.1 Interconexión del sistema

Para la implementación del sistema se utilizo un zócalo para tarjetas SD, el cual permite acceder físicamente y de manera simple a los pines de la misma. El Mcc se programo utilizando lenguaje "C" y se utilizo el software MPLAB IDE de Microchip Technology Incorporated para realizar la depuración del software.

6.1 Hardware implementado

6.1.1 Fuente de alimentación

La tarjeta SD se alimenta en un rango de voltaje de 2.7 a 3.6 V. Para la alimentación del Mcc de la grafica Voltaje-Frecuencia del manual del PIC18F458 observamos a que frecuencia puede operar el microcontrolador siendo alimentado al mismo voltaje que la tarjeta SD. Entonces para obtener un voltaje regulado en este rango utilizamos un regulador de voltaje LM317 y con la ecuación:

$$Vout = 1.25\left(1 + \frac{R1}{R2}\right) + Iajuste * R1$$

Usando los valores de resistencia de R1 = 470 ohms y R2 = 270 ohms, que son valores comerciales, tendremos un voltaje de 3.4V aproximadamente.

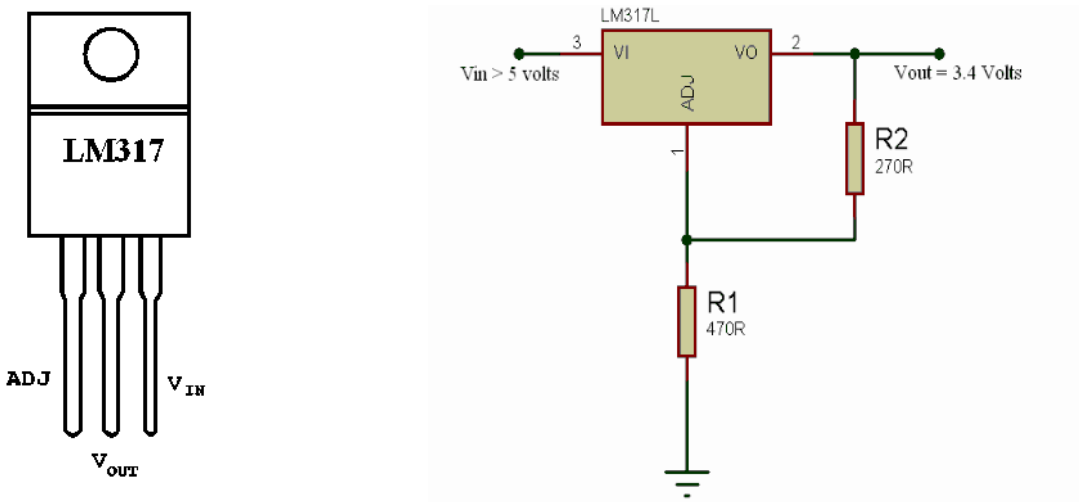


Figura 6.2 Regulador de voltaje

La frecuencia de operación del microcontrolador se determina de acuerdo con la figura 6. 3 donde a un voltaje de 3.4 volts se obtiene una frecuencia máxima de operación de aproximadamente 25 MHz.

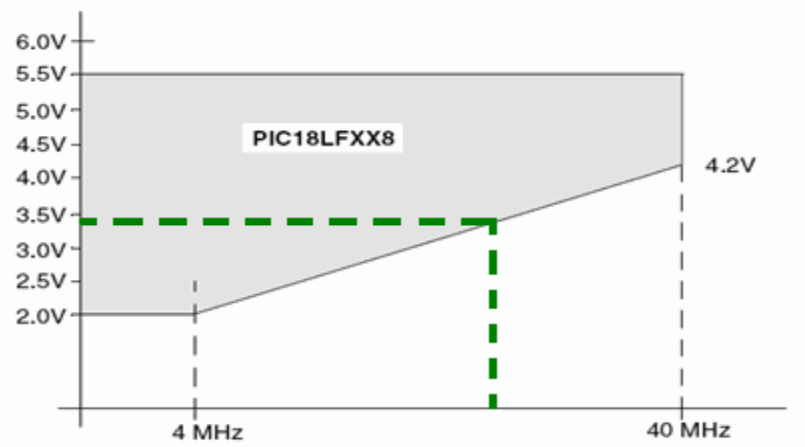


Figura 6.3 Grafica de operación Frecuencia vs Voltaje

6.1.2 Cristal

El cristal es utilizado para generar la señal de reloj y velocidad de operación de los circuitos osciladores convirtiendo las vibraciones mecánicas en voltajes eléctricos a una frecuencia. El cristal es capaz de producir frecuencias precisas y estables. La mayoría de los microcontroladores requieren de un circuito que le indique a que velocidad debe operar. El microcontrolador PIC18F458 requiere de un oscilador de cristal como generador de pulsos de reloj. El pin 13 y el pin 14 del Mcc son utilizados para conectar el cristal oscilador. La figura 6.4 muestra como es conectado el cristal con el Mcc. El cristal utilizado es de una frecuencia de 10MHz.

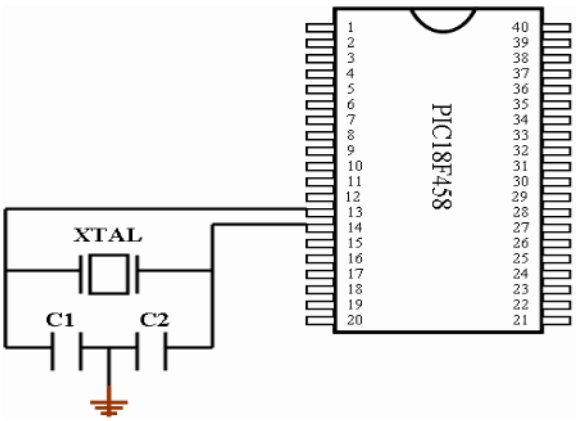
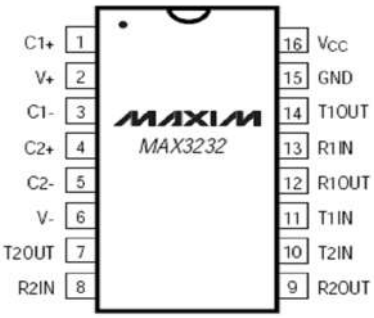


Figura 6.4 Conexión del circuito del cristal oscilador al microcontrolador

6.1.3 Max3232

Es un circuito de interfaz de entrada/salida convertidor de señales RS-232 a TTL y viceversa. Este circuito se conecta en las terminales del microcontrolador para adecuar los niveles de voltaje. El circuito MAX3232 fue elegido debido a que funciona con rangos de voltaje entre 3.3 y 5 Volts.



Esta interfaz es utilizada para comunicarse entre la memoria y una PC u otro sistema ya que es el método de acceder al uso de la memoria.

6.1.4 Zócalo

Para conectar la tarjeta con el microcontrolador, es necesario un zócalo de conexión para memorias SD que permite insertar y retirar la tarjeta fácilmente, el zócalo es fabricado a la medida de las tarjetas SD y también acepta tarjetas MMC y cuenta con pines que conecta las terminales de la tarjeta con el circuito. El zócalo utilizado en este proyecto se retiro de un lector de memorias dañado. La figura 6.6 muestra el circuito de conexión del zócalo con el Mcc. Las resistencias R1, R2 y R3 se les llama resistencias *Pull-up*, protegen las líneas de comunicación contra voltajes flotantes cuando no esta insertada una tarjeta.

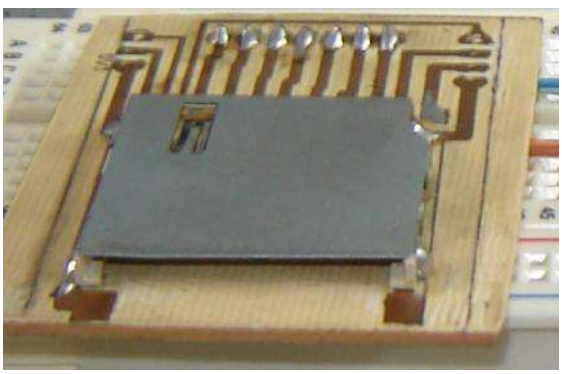


Imagen 6.5 Zócalo de tarjeta SD

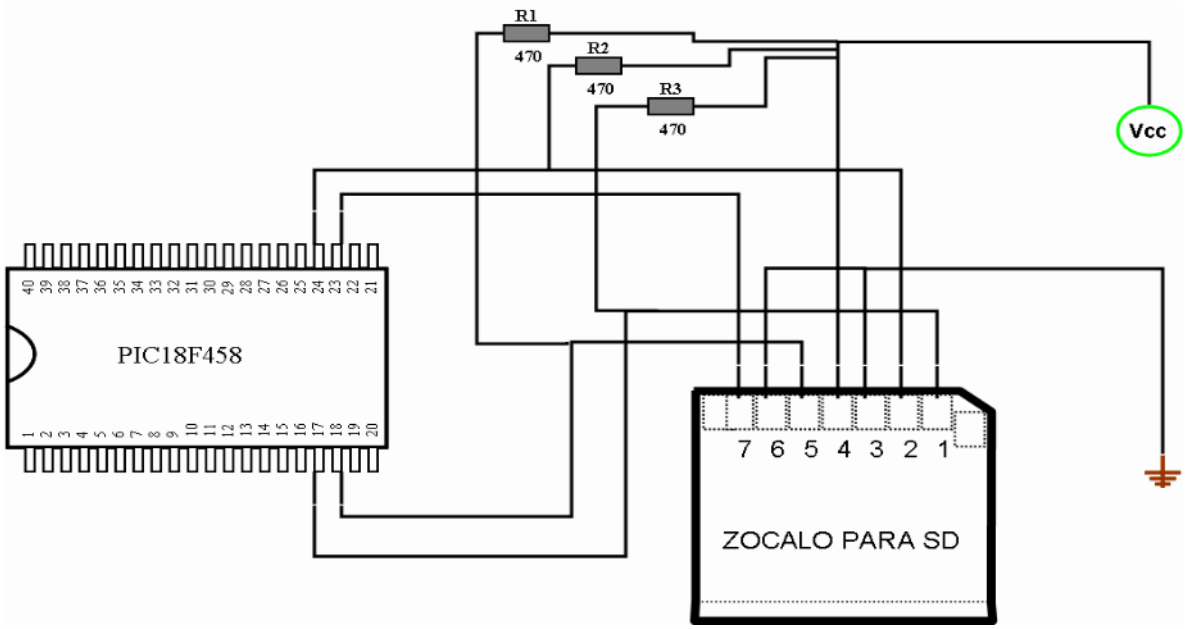


Figura 6.6 Diagrama esquemático de conexión del microcontrolador a la tarjeta SD

6.2 Inicialización

Debido a que el microcontrolador elegido no solo posee el protocolo SPI, es necesario configurar la tarjeta SD en modo SPI para completar la comunicac ión con el microcontrolador.

La tarjeta SD siempre inicia su Bus en modo SD, y cambia a modo SPI si la señal CS es puesta a cero durante la recepción del comando reset (CMD0). Si la tarjeta reconoce que se necesita el modo SD, no responderá al comando y p ermanecerá en el modo Bus SD. Si el modo SPI es requerido, la tarjeta cambiara a modo SPI y responderá con la respuesta R1 del modo de operación SPI.

CMD0 es un comando estático y siempre genera un CRC de 7 bits 4A (hex). La siguiente secuencia hexadecima l puede usarse para mandar el comando 0 en toda situación para cambiar el bus al modo SPI 40 00 00 00 00 95 (hexadecimal).

6.3 Inicializando los puertos

6.3.1 Puerto USART

El puerto USART es usado para comunicar al microcontrolador con otro sistema o dispositivo periférico.

El registro TXSTA es el registro de control y estado de transmisión. El bit 6 configura que la transmisión sea de 8 bits, el bit 5 habilita la transmisión y los bits 4 y 2 definen el modo asíncrono en alta velocidad.

Registro TXSTA

0	0	1	0	0	1	0	0

bit7 bit0

El registro SPBRG controla la velocidad de transmisión y recepción, el valor se obtiene de las tablas del manual del Microcontrolador según la velocidad que utilizamos. En nuestro caso SPBRG = 0x40, Fosc=10Mhz, en modo asíncrono y BRGH=1 (valores seleccionados en el registro TXSTA) por lo tanto la velocidad es de 9.62 KBAUD.

El registro RCSTA es el registro de control y estado de recepción. El bit 7 habilita el puerto serie, el bit 6 configura que la recepción sea de 8 bits, y el bit 4 en modo asíncrono habilita recepción continua.

Registro RCSTA

1	0	0	1	0	0	0	0

bit7 bit0

6.3.2 Puerto SPI

El microcontrolador se comunica con la tarjeta SD mediante el puerto SPI, este puerto es fácil de utilizar y su conexión es simple. Muchos microcontroladores incluyen hardware de soporte para SPI que simplifica la programación.

Todas las señales SPI son unidireccionales así que el controlador no necesita tener pines de puerto bi-direccional.

El registro SSPSTAT es el registro de estado, se configura para que la transmisión ocurra en la transición de alto a bajo y el modo idle sea uno logico.

Registro SSPSTAT

0	1	0	0	0	0	0	0

bit7 bit0

El registro SSPCON1 es el registro de control 1, se configura el bit 5 para habilitar el modo SPI, el bit 4 configura el estado de espera en bajo. En los últimos 4 bits de este registro se configura el modo maestro y la velocidad del reloj Fosc/64.

Registro SSPCON1

0	0	1	0	0	0	1	0

bit7 bit0

Se configuran también los pines para transmisión de datos:

RC2 Pin CS salida

RC3 Pin SCK salida

RC4 Pin SDI entrada

RC5 Pin SDO salida

6.4 Transferencia de bloques

6.4.1 Lectura de datos

En el modo SPI, la tarjeta SD soporta operaciones de lectura de un solo bloque o múltiples bloques. En la recepción de un comando de lectura valido, la tarjeta responderá con una señal de respuesta seguido de un bloque de datos de una longitud definida.

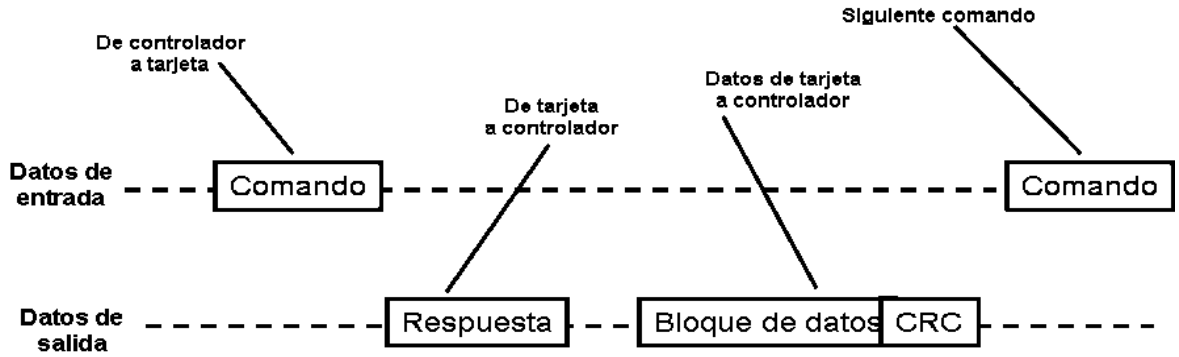


Figura 6.7 Operación de lectura de un bloque

Para la lectura de un bloque se utiliza el comando CMD17, especificando la dirección del bloque a leer. La dirección es el primer byte del bloque que se desea leer, el cual se calcula con el número del bloque por la longitud de bloque (512 bytes). Si se envió correctamente el comando, la tarjeta SD responderá con una señal seguido del bloque de datos y terminara con dos bytes CRC. El bloque de datos se almacena en un buffer previamente reservado dentro del controlador.

Nota: El inicio de la dirección puede ser cualquier byte en el rang o de dirección valido de la tarjeta. Sin embargo, cada bloque debe estar contenido dentro de un solo sector físico de la tarjeta.

6.4.2 Escritura de datos

En el modo SPI, la tarjeta SD soporta también operaciones de escritura de un bloque o múltiples bloques. En la recepción de un comando de escritura valido, la tarjeta responderá con una señal y esperara por un bloque de datos enviado por el microcontrolador.

Para escribir un bloque a la tarjeta se utiliza el comando CMD24 con la dirección del bloque, si todo va bien la tarjeta SD responderá con una señal (token de inicio de escritura) informando al controlador que ya puede enviar el bloque de datos, que ha de tener una longitud de 512 bytes, si se define un bloque diferente de 512 bytes se causara un error de escritura. La SD responderá si los datos fueron aceptados.

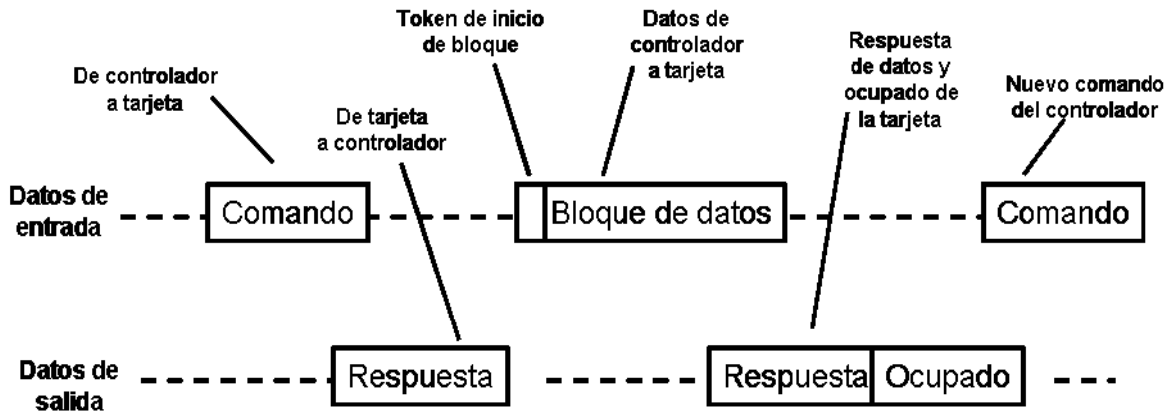


Figura 6.8 Operación de escritura de un bloque

Hay algunas restricciones que el controlador SPI debe seguir:

Después de la última transacción, se requiere que el controlador envíe a la tarjeta 8 ciclos de reloj para completar la operación antes de que se apague el reloj. Durante este periodo de 8 ciclos, el estado de CS es irrelevante.

6.5 Estructuras de la tarjeta y de archivo

Una estructura es una colección de variables. Permiten organizar y manejar datos de mejor manera.

La estructura dat_SD contiene las variables que identifican la organización de la memoria SD como bytes por sector, ubicación, cantidad de sectores de los componentes o regiones de la memoria, así como capacidad de memoria.

Tabla 6.1 Variables utilizadas en la estructura de la tarjeta

Nombre de la variable	**Tipo de variable**	**Descripción**
Memoria	Arreglo	Contiene el tamaño de memoria de la tarjeta SD
Bytes_Sec	Entero	Contiene la cantidad de bytes por cada sector en la memoria
RootDir	Entero	Ubicación del primer bloque del directorio raíz
Sec_Directorio	Entero	Guarda la cantidad de sectores en el directorio raíz
FATs	Char	Numero de copias de la tabla FAT
FAT1, FAT2	Entero	Bloque de inicio de la tabla FAT
SecPorFAT	Entero	Contiene la cantidad de sectores por tabla FAT
Sec_res	Entero	Sectores reservados entre el Boot Record y la tabla FAT
MaxFilesRoot	Entero	Máximo numero de entradas en el directorio raíz
DataIni	Entero	Numero de bloque donde inicia la región de datos en la memoria

La estructura dat_File contienen variables con información de un archivo como son nombre de archivo, extensión, tamaño y el cluster de inicio del archivo.

Tabla 6.2 Variables utilizadas en la estructura de archivo

Nombre de la variable	**Tipo de variable**	**Descripción**
nombre	Arreglo	Contiene el nombre del archivo
extension	Arreglo	Contiene la extensión del archivo
inicio	Entero	Cluster inicial del archivo
size	Long	Tamaño del archivo en bytes

6.6 Funciones implementadas

ReadWord

Esta función lee un Word en formato "*little endian*" (adoptado por Intel para leer datos de mas de un byte) y regresa el valor de 2 bytes de un bloque capturado previamente y almacenado en el buffer. El contenido del primer byte será la parte menos significativa del valor y el segundo byte la parte más significativa. La función recibe la dirección del primer byte.

```
unsigned int ReadWord(unsigned int Direccion)
{
unsigned int valor;
      valor = buf[Direccion+1];
      valor = (valor << 8) | buf[Direccion];
      return valor;
}
```

SPI_SEND

SPI_SEND envía un byte al puerto SPI, todos los datos transmitidos o recibidos son capturados en el buffer del microcontrolador (SSPBUF). El bit 0 (BF) del registro SSPSTAT tiene un "1 lógico" cuando el dato esta completamente capturado en el buffer.

```
void SPI_SEND(unsigned char Dato)
{
      SSPBUF = Dato;
      while(!SSPSTATbits.BF);
}
```

SPI_READ

Funciona igual que SPI_SEND solo que se usa en una respuesta de la tarjeta hacia el Mcc y además regresa el contenido del registro SSPBUF.

```
unsigned char SPI_READ(unsigned char Dato)
{
      SSPBUF = Dato;
      while(!SSPSTATbits.BF);
      return SSPBUF;
}
```

SD_response

Esta función espera una respuesta de la tarjeta, esto lo realiza enviando un 0xFF. La función SD_response inicia un contador hasta que se obtenga esa respuesta y regresa un OK, pero si el contador termina o se recibe una respuesta no esperada, regresa un ERROR. Por ejemplo cuando se envía el comando de escritura de bloque con todos sus argumentos, utilizando esta función esperamos que la tarjeta SD responda con un 0x00 que indica que el comando fue recibido.

```
unsigned char SD_response(unsigned char response)
{
unsigned char count = 0xFF;

    while(SPI_READ(0xFF) != response &&  --count > 0);
    if(count == 0)
        return ERROR;
    else
        return OK;
}
```

send_command

Envía a la tarjeta SD un comando con su argumento y CRC, recordando que un comando esta compuesto por 6 bytes, la función send_command solo recibe el comando (1 byte), argumentos (4 bytes) y el CRC (1 byte), la función se encarga de enviar cada byte a la tarjeta utilizando la función SPI_SEND.

```
void send_command(unsigned char comando, unsigned char arg1, unsigned char arg2,
                  unsigned char arg3, unsigned char arg4, unsigned char CRC)
{
    SPI_SEND(comando);
    SPI_SEND(arg1);
    SPI_SEND(arg2);
    SPI_SEND(arg3);
    SPI_SEND(arg4);
    SPI_SEND(CRC);
}
```

Write_block

Guarda el contenido del buffer en un bloque de la tarjeta de memoria. La función recibe el número del bloque al cual se va a escribir. Envía el comando de escritura de bloque (CMD24) y como argumento envía la dirección del bloque, utilizando la función **send_command**, la dirección del bloque es el número del bloque multiplicado por la longitud de bloque (512). Espera por una respuesta que indica que el comando fue recibido y un token de inicio de transmisión (FEh). Envía cada uno de los 512 bytes del buffer. Espera una respuesta 0x05 que indica que los datos fueron aceptados y termina con ocho ciclos de reloj.

```
unsigned char Write_block(unsigned long bloque)
{
unsigned int j;
    bloque = (bloque<<9);              //  "<< 9" es igual a multiplicar por 512
    CS = 0;
    respuesta = OK;

    send_command(WRITE_BLOCK, (bloque>>24)&0xFF, (bloque>>16)&0xFF,
                 (bloque>>8)&0xFF, bloque&0xFF, 0xFF);
    if(SD_response(0x00) == OK)
    {
        // SPI_SEND(0xFF);                    // ciclos de espera
        SPI_SEND(0xFE);                       // token de inicio de escritura
        for(j=0; j<512; j++)
        {
            SPI_SEND(buf[j]);
        }
        SPI_SEND(0xFF);                       //CRC
        SPI_SEND(0xFF);

        if((SPI_READ(0xFF) & 0x0F) == 0x05)       //revisa que los datos fueron
                                                  //aceptados
        {
            while(SPI_READ(0xFF) != 0xFF);
        }
    }
    SPI_SEND(0xFF);
    CS = 1;
    return OK;
}
```

Read_block

Lee un bloque de la tarjeta de memoria y lo almacena en el buffer de 512 bytes. La función recibe el número del bloque que se quiere leer. Envía el comando de lectura de bloque (CMD17) como argumento se envía la dirección del bloque. Espera una señal de la tarjeta que indica que el comando fue recibido. Espera una señal de inicio d e transmisión llamada token (FEh). Recibe la lectura de bloque y se almacena en un arreglo. Envía ocho ciclos de reloj para finalizar.

```
unsigned char Read_block(unsigned long bloque)
{
unsigned int j;
unsigned long address = (bloque<<9);          //  "<< 9" es igual a multiplicar por 512


    CS = 0;
    send_command(0x51,(address>>24)&0xFF, (address>>16)&0xFF,
                              (address>>8)&0xFF, address&0xFF, 0xFF);
    if(SD_response(0x00) == OK)
    {
        if(SD_response(0xFE) == OK)
        {
            for(j=0; j<512; j++)
            {
                buf[j] = SPI_READ(0xFF);
            }
            SPI_READ(0xFF);
            SPI_READ(0xFF);
        }
    }
    SPI_SEND(0xFF);                 // ciclos de reloj para finalizar
    CS = 1;
    return OK;
}
```

read_CID

Existe un comando para leer el registro CID de la tarjeta SD. Recordar que este registro es programado de fábrica, este registro es de identificación de la tarjeta. Para leer el registro se sigue el mismo procedimiento de lectura de cualquier otro bloque:

-Envía el comando de lectura del registro CID (CMD10), no se requiere enviar la di rección del bloque por lo tanto como argumento se escriben ceros.

-Espera por la señal de recepción de comando y el token de inicio de transmisión (FEh).

-Recibe la lectura del bloque y la almacena en el buffer.

-Envía ciclos de reloj para finalizar.

```
void read_CID(void)
{
unsigned char i;

    CS = 0;
    send_command(READ_CID, 0, 0, 0, 0, 0xFF);
    if(SD_response(0x00) == OK)
    {
        if(SD_response(0xFE) == OK)
        {
            for(i=0; i<18; i++)                    // el registro CID contiene 18 bytes
            {
                buf[i] = SPI_READ(0xFF);
            }
        }
    }
    SPI_SEND(0xFF);                                //ciclos de reloj para finalizar
    CS = 1;
}
```

Read_SD

Al ejecutar el programa, después de inicializar los puertos del Mcc y la tarjeta SD, se llama a la función Read_SD que almacenara la información necesari a en la estructura dat_SD, esta función obtiene los parámetros mínimos para operar correctamente el software de la memoria

-Lee el registro CID.

-Almacena en la variable *memoria* de la estructura dat_SD los bytes 3 a 7 del buffer de la lectura del registro CID

-Lee el bloque 0 (Boot Record).

-Almacena los bytes del Boot Record correspondientes para cada variable de la estructura dat_SD para las variables de 2 bytes se utiliza la función ReadWord

-*RootDir* se calcula con la siguiente formula Sec_res + SecPorFAT * FATs

```
void Read_SD(void)
{
	read_CID();
	registro.memoria[0] = buf[3];
	registro.memoria[1] = buf[4];
	registro.memoria[2] = buf[5];		// Obtiene el tamaño de tarjeta SD
	registro.memoria[3] = buf[6];
	registro.memoria[4] = buf[7];

	Read_block(0);

		// Sectore reservados, antes de la FAT
	registro.Sec_res = ReadWord(0x0E);

		// Bytes por sector
	registro.Bytes_Sec = ReadWord(0x0B);

		// numero de sectores por FAT
	registro.SecPorFAT = ReadWord(0x16);

	registro.FATs = buf[0x10];			// Numero de FATs

		// direccion del Directorio Raiz
	registro.RootDir = registro.Sec_res + registro.SecPorFAT * registro.FATs;

		// Direccion de la FAT 1 y Fat 2
	registro.FAT1 = registro.Sec_res;
	registro.FAT2 = registro.FAT1 + registro.SecPorFAT;

		// maximo numero de entradas en el directorio raiz
	registro.MaxFilesRoot = ReadWord(0x11);

		// Sectores para el Directorio Raiz
	registro.Sec_Directorio = (registro.MaxFilesRoot * 32) / registro.Bytes_Sec;

		// Direccion de inicio de los datos
	registro.DataIni = registro.Sec_Directorio + registro.RootDir;
}
```

Read_File

Esta función se encarga de formar la estructura de un archivo, obteniendo datos de una entrada del directorio raíz (32 bytes). Recibe el número de entrada que corresponde al archivo que se quiere leer, para esto el bloque del directorio raíz ya ha sido leído con la función Read_block.

Las variables de la estructura de archivo se guardan con los bytes correspondientes en la entrada.

```
void Read_File(unsigned int Archivo)
{
    file.nombre[0] = buf[Archivo];
    file.nombre[1] = buf[Archivo + 1];
    file.nombre[2] = buf[Archivo + 2];
    file.nombre[3] = buf[Archivo + 3];
    file.nombre[4] = buf[Archivo + 4];
    file.nombre[5] = buf[Archivo + 5];
    file.nombre[6] = buf[Archivo + 6];
    file.nombre[7] = buf[Archivo + 7];

    file.extension[0] = buf[Archivo + 8];
    file.extension[1] = buf[Archivo + 9];
    file.extension[2] = buf[Archivo + 10];

    file.inicio = buf[Archivo + 27];
    file.inicio = file.inicio <<8 | buf[Archivo + 26];

    file.size = buf[Archivo + 31];
    file.size = file.size << 8 | buf[Archivo + 30];
    file.size = file.size << 8 | buf[Archivo + 29];
    file.size = file.size << 8 | buf[Archivo + 28];

}
```

guardaEn_Directorio

Cuando se requiere almacenar un archivo, es necesario modificar el directorio raíz con los datos de la estructura de archivo, de la siguiente forma:

-Lee un bloque del directorio raíz.

-Localiza la primer entrada de archivo disponible (32 bytes), esto es, que el primer byte de la entrada sea 0x00 (entrada vacía) o 0xE5 (archivo borrado). Si se lee el primer bloque del

directorio raíz, busca la entrada disponible a partir de la segunda entrada ya que la primera entrada corresponde al volumen de la tarjeta SD. En caso de no encontrar una entrada disponible, cambia al siguiente bloque del directorio raíz.

-Escribe los valores de las variables de la estructura de archivo en las posiciones correspondientes dentro de la entrada disponible.

Tabla 6.3 Posición de variables en una entrada del directorio raíz

Variable	Posición dentro de la entrada disponible
nombre	0 a 7
extensión	8 a 10
inicio	27 y 26
size	28 a 31

Guarda el bloque en la memoria con la función *Write_block*.

En caso de que no se encuentra una entrada disponible en este bloque se busca en el siguiente y en cada uno de los sectores del directorio raíz.

6.7 Comandos

6.7.1 Comando D (Dir)

Este comando muestra el nombre de los archivos incluyendo su extensión contenidos en la memoria, como información adicional muestra el tamaño de cada archivo.

Leer primer bloque del directorio raíz

Leer archivos validos

Obtener Nombre, Extensión y Tamaño

Leer el siguiente bloque hasta el fin del directorio raíz.

6.7.2 Comando S (Save)

Este comando guarda un archivo en la memoria, co mo entrada se debe dar el nombre incluyendo su extensión. El sistema sigue el siguiente proceso para guardar un archivo.

Obtener Nombre, Extensión.

Modificar la tabla FAT con la cadena correspondiente al archivo .

Modificar y guardar en la primera entrada disponib le del directorio raíz.

Guardar datos del archivo en la región de datos de la memoria

Capítulo 7

Ejemplo de aplicación

En este capítulo se presenta una breve descripción de operación del sistema, así como un ejemplo de donde podría ser aplicado.

Como se menciona en el capítulo anterior el sistema esta basado en un circuito depurador mostrado en la figura 7.1. Es aquí donde se encuentra el microcontrolador, la conexión RS232, el regulador de voltaje, cristal y entrada para alimentación. Se monta el circuito en un protoboard para conectarlo con el zócalo para tarjeta SD.

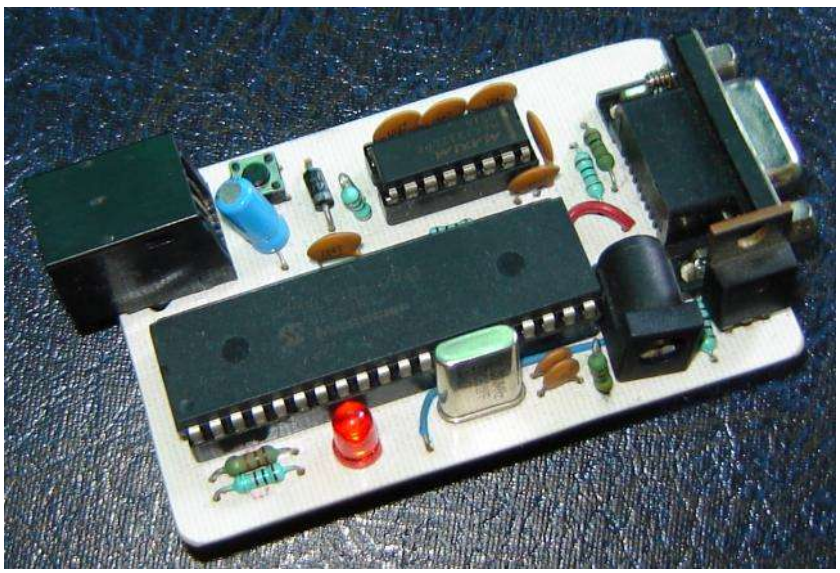


Figura 7.1 Circuito depurador original.

La figura 7.2 muestra el circuito impreso al cual solo se le modifico un poco para adaptar el regulador de voltaje LM317

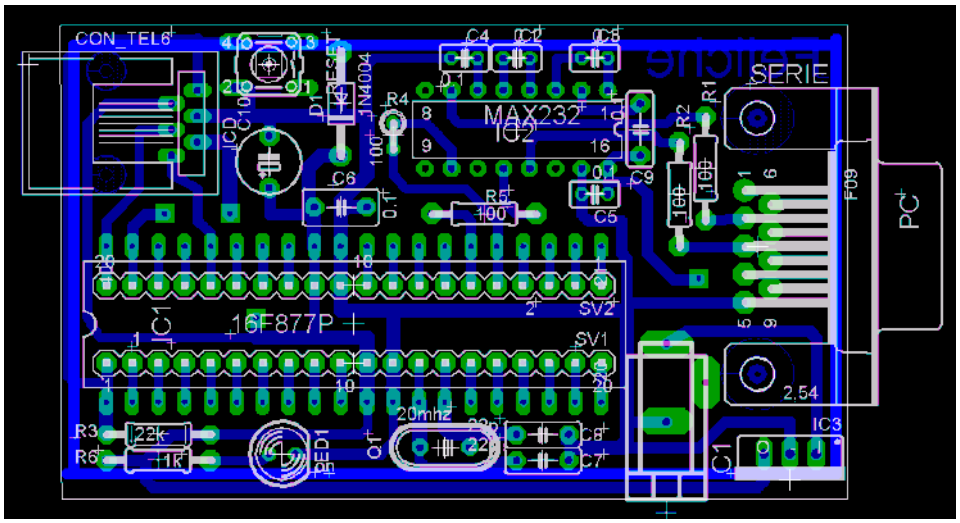


Figura 7.2 Diseño de circuito impreso para el circuito depurador

7.1 Operación del sistema

El sistema se conecta mediante la conexión RS232 a otro sistema o dispositivo el cual necesita almacenar información en la tarjeta SD.

7.1.1 Almacenar archivos

Para almacenar archivos el sistema cuenta con el comando Save (Guardar) el cual se indica con una letra S. Al recibir una 'S' el sistema esperara un espacio seguido d el nombre del archivo y la extensión del mismo, y después de otro espacio recib e los datos del archivo, es decir el contenido.

La figura 7.3 muestra una pant alla de la Terminal de Windows a modo de ejemplo se almacenan 3 archivos de texto, después del nombre y extensión del archivo se muestra el contenido que tendrá el archivo. Par a que el sistema identifique el final del contenido de un archivo se utiliza como identif icador de fin de archivo CTRL+Z (→).

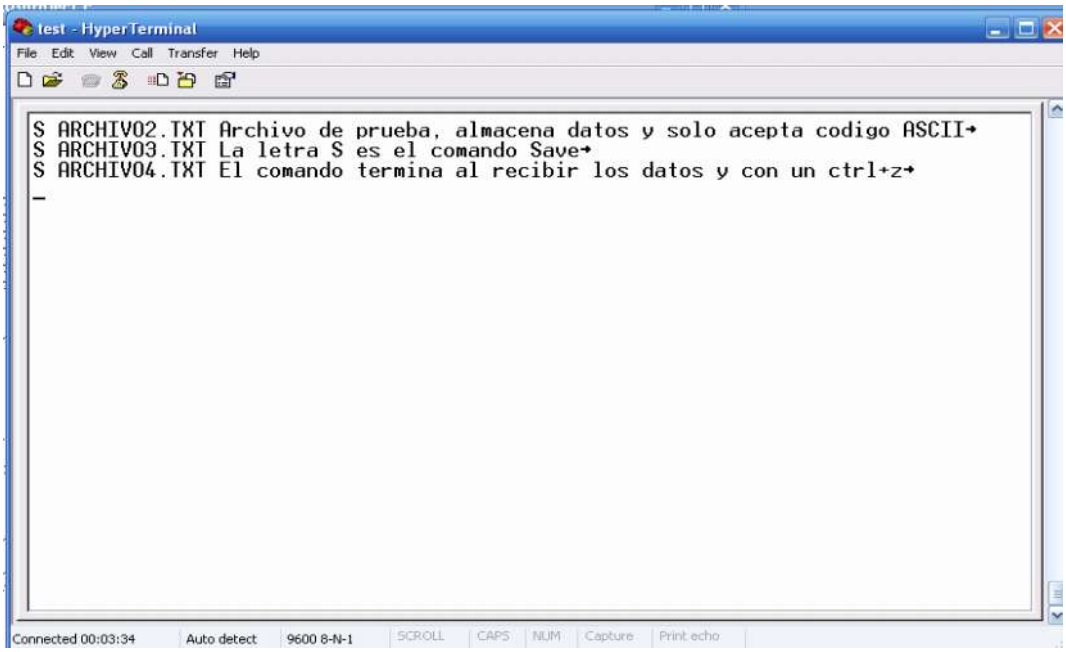


Figura 7.3 Ejemplo de secuencia para almacenar archivos

Como se puede ver en la figura 7.4 los mismos archivos que se guardan mediante este sistema en la tarjeta SD pueden ser leídos y procesados en Windows utilizando un lector de memorias SD, manteniendo así la compatibilidad con otros sistemas.

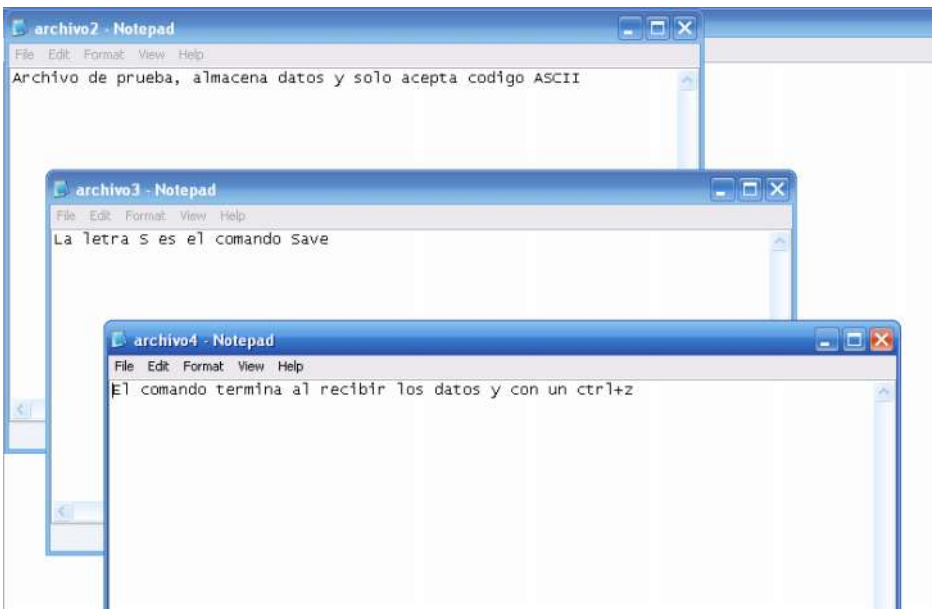


Figura 7.4 Archivos de ejemplo vistos en Windows

7.1.2 Directorio de archivos

Otra comando que tiene el sistema es el comando Dir o D irectorio que se indica con una ‘D’ y muestra una lista de los archivos almacenados en la tarjeta . Cuando el sistema recibe una ‘D’ recorre el directorio raíz de la tarjeta y hace una lista de todos los archivos almacenados en ella. En la lista se muestra el nombre del archivo, su exten sión y el tamaño en bytes en valor hexadecimal como se muestra en la figura 7.5.

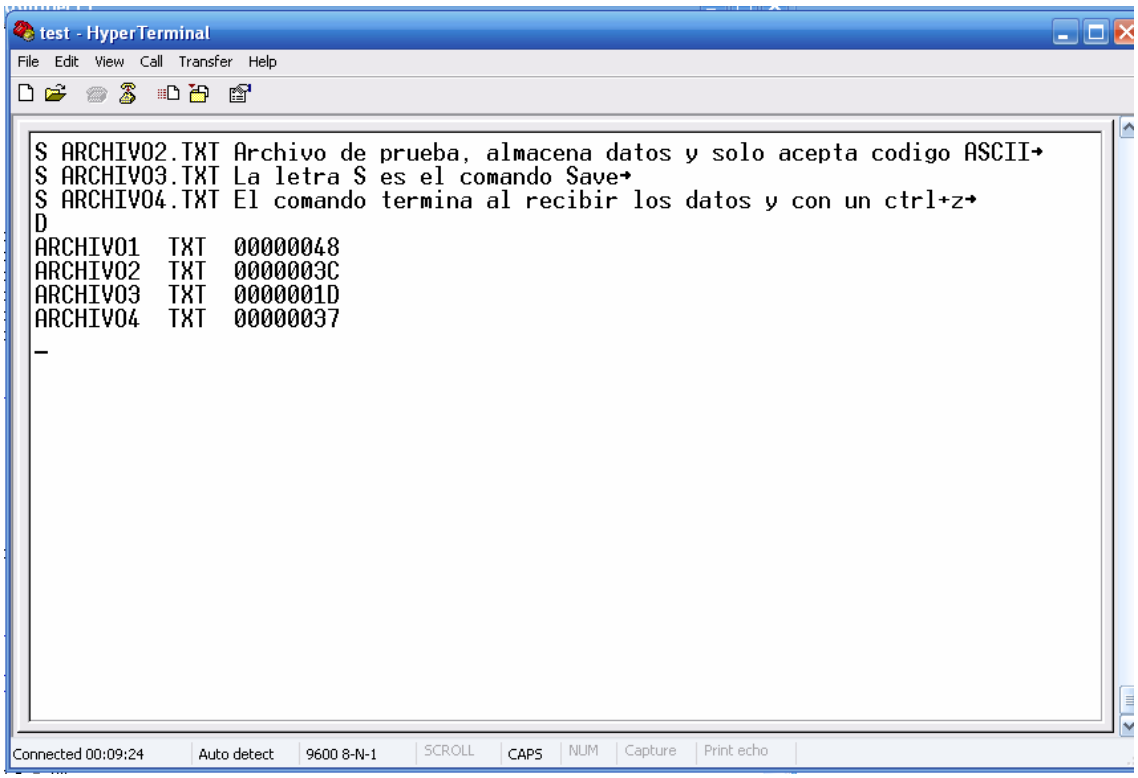


Figura 7.5 Ejemplo de comando D (Directorio de archivos)

Capítulo 8

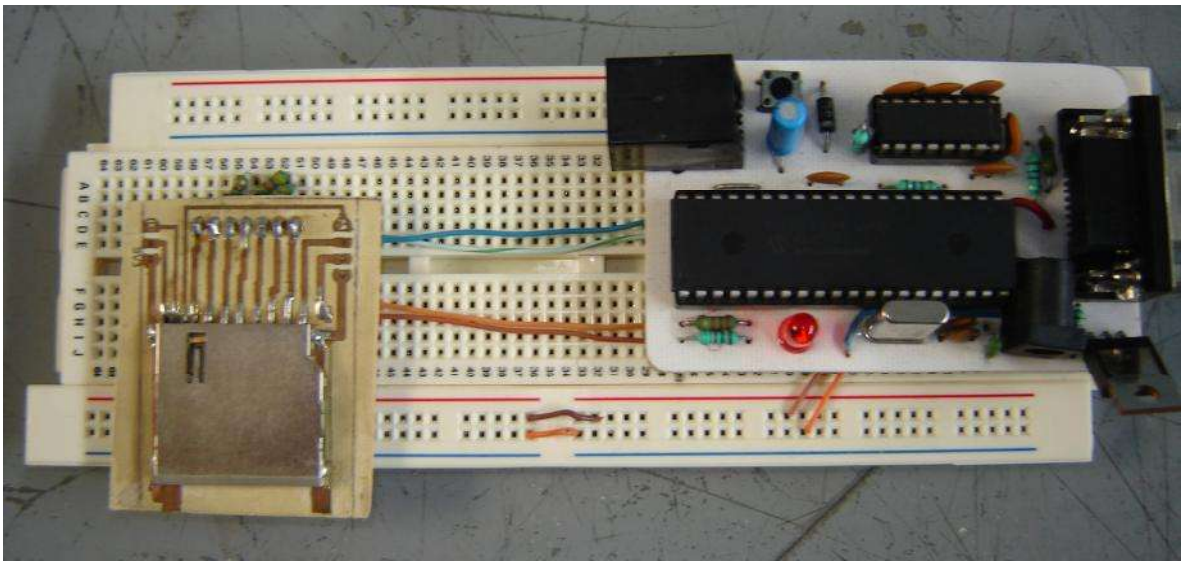
Conclusiones

Los diferentes tipos de memoria ofrecen variedad de formas, capacidades y me nor costo lo que ha convertido a la memoria flash en el tipo de memoria más común y usada en dispositivos móviles. Para este proyecto el tipo de memoria que mejor se adapta es la tarjeta SD por ser memoria de tipo no volátil, es decir, que la información n o se borra al desconectarle la energía. Además que dispone del protocolo de comunicación SPI que reduce el numero de pines para accederla y simplifica a su vez el hardware necesario para su control.

Una de la características de este sistema es que el for mato de archivos FAT mantiene compatibilidad con sistemas operativos y la información almacenada en nuestro sistema puede ser leída sin problemas en una PC y sin necesidad de software adicional ya que utiliza el sistema de archivos FAT16 el cual es uno de los sistemas de archivos mas usados y es comúnmente encontrado en las tarjetas flash. A diferencia del sistema FAT32, FAT16 tiene una implementación más sencilla. FAT32 al tener mayor tamaño de cluster se desperdicia mucho espacio si se requiere almacenar un archivo pequeño.

La interfase SPI es la forma más fácil para transferir datos entre la tarjeta de memoria y el microprocesador, ya que en caso contrario se requeriría implementar un Bus SD para la transferencia de los archivos e información por eso s e eligió el microcontrolador PIC18F458 el cual puede utilizar el protocolo SPI y tiene los puertos necesarios para completar la comunicación con la tarjeta y un sistema mínimo, así como la capacidad de memoria RAM necesaria para implementar las diferentes funciones.

El sistema desarrollado permite realizar operaciones de lectura y escritura sobre tarjetas flash SD de diferentes capacidades. Este sistema funciona con una fuente de 5 volts de corriente continua.



Características obtenidas

- El sistema es un prototipo.
- El número de archivos que se pueden almacenar depende del tamaño de la tarjeta y de la cantidad de bloques por cluster que contenga.
- Al almacenar archivos tanto el nombre como la extensión debe n de introducirse en mayúsculas para conservar la compatibilidad con DOS y Windows de lo contrario dichos archivos no serán compatibles.
- El tamaño de archivo máximo es de 512 bytes que es el tamaño de un bloque, esto simplifica el diseño del software.
- Al almacenar un archivo en la SD el tamaño se calcula automáticamente al introducir los datos o contenido del archivo.
- Longitud y extensión del nombre del archivo de acuerdo a DOS (8.3)

Mejoras a futuro

- Incrementar el limite del tamaño de los archivos, es decir que el archivo sea limitado solo por la capacidad de memoria de la tarjeta.
- No tener restricciones a utilizar mayúsculas o minúsculas en el nombre del archivo, así como su longitud.
- Ver el contenido del archivo utilizando el sistema, es decir agregarle un comando de lectura de archivos.
- Introducir mas comando como el de borrado de archivos.

Apéndice A

Especificaciones del PIC18FXX8

PIC18FXX8
Data Sheet

28/40-Pin High-Performance,
Enhanced Flash Microcontrollers
with CAN Module

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona, Gresham, Oregon and Mountain View, California. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

MICROCHIP

PIC18FXX8

28/40-Pin High-Performance, Enhanced Flash Microcontrollers with CAN

High-Performance RISC CPU:

- Linear program memory addressing up to 2 Mbytes
- Linear data memory addressing to 4 Kbytes
- Up to 10 MIPS operation
- DC – 40 MHz clock input
- 4 MHz-10 MHz oscillator/clock input with PLL active
- 16-bit wide instructions, 8-bit wide data path
- Priority levels for interrupts
- 8 x 8 Single-Cycle Hardware Multiplier

Peripheral Features:

- High current sink/source 25 mA/25 mA
- Three external interrupt pins
- Timer0 module: 8-bit/16-bit timer/counter with 8-bit programmable prescaler
- Timer1 module: 16-bit timer/counter
- Timer2 module: 8-bit timer/counter with 8-bit period register (time base for PWM)
- Timer3 module: 16-bit timer/counter
- Secondary oscillator clock option – Timer1/Timer3
- Capture/Compare/PWM (CCP) modules; CCP pins can be configured as:
  - Capture input: 16-bit, max resolution 6.25 ns
  - Compare: 16-bit, max resolution 100 ns (TCY)
  - PWM output: PWM resolution is 1 to 10-bit Max. PWM freq. @:8-bit resolution = 156 kHz 10-bit resolution = 39 kHz
- Enhanced CCP module which has all the features of the standard CCP module, but also has the following features for advanced motor control:
  - 1, 2 or 4 PWM outputs
  - Selectable PWM polarity
  - Programmable PWM dead time
- Master Synchronous Serial Port (MSSP) with two modes of operation:
  - 3-wire SPI™ (Supports all 4 SPI modes)
  - I^2C™ Master and Slave mode
- Addressable USART module:
  - Supports interrupt-on-address bit

Advanced Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter module (A/D) with:
  - Conversion available during Sleep
  - Up to 8 channels available
- Analog Comparator module:
  - Programmable input and output multiplexing
- Comparator Voltage Reference module
- Programmable Low-Voltage Detection (LVD) module:
  - Supports interrupt-on-Low-Voltage Detection
- Programmable Brown-out Reset (BOR)

CAN bus Module Features:

- Complies with ISO CAN Conformance Test
- Message bit rates up to 1 Mbps
- Conforms to CAN 2.0B Active Spec with:
  - 29-bit Identifier Fields
  - 8-byte message length
  - 3 Transmit Message Buffers with prioritization
  - 2 Receive Message Buffers
  - 6 full, 29-bit Acceptance Filters
  - Prioritization of Acceptance Filters
  - Multiple Receive Buffers for High Priority Messages to prevent loss due to overflow
  - Advanced Error Management Features

Special Microcontroller Features:

- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator
- Programmable code protection
- Power-saving Sleep mode
- Selectable oscillator options, including:
  - 4x Phase Lock Loop (PLL) of primary oscillator
  - Secondary Oscillator (32 kHz) clock input
- In-Circuit Serial Programming™ (ICSP™) via two pins

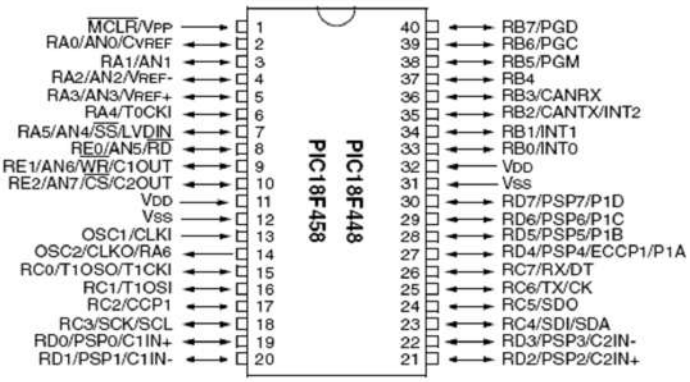
Flash Technology:

- Low-power, high-speed Enhanced Flash technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Industrial and Extended temperature ranges

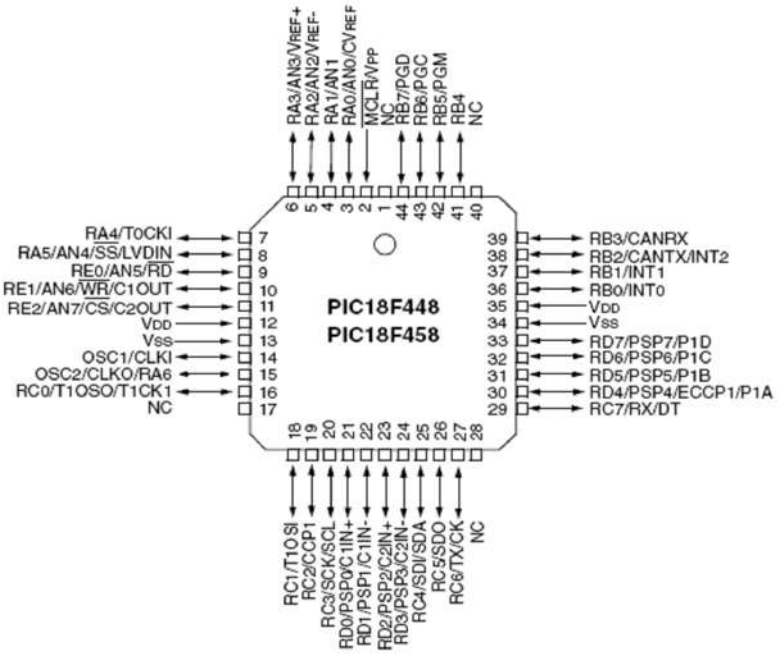
Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	Comparators	CCP/ ECCP (PWM)	MSSP		USART	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI™	Master I^2C™		
PIC18F248	16K	8192	768	256	22	5	—	1/0	Y	Y	Y	1/3
PIC18F258	32K	16384	1536	256	22	5	—	1/0	Y	Y	Y	1/3
PIC18F448	16K	8192	768	256	33	8	2	1/1	Y	Y	Y	1/3
PIC18F458	32K	16384	1536	256	33	8	2	1/1	Y	Y	Y	1/3

Pin Diagrams

PDIP



PLCC



27.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings(†)

Ambient temperature under bias ..... -40°C to +125°C
Storage temperature ..... -65°C to +150°C
Voltage on any pin with respect to V_{SS} (except V_{DD}, $\overline{MCLR}$ and RA4) ..... -0.3V to (V_{DD} + 0.3V)
Voltage on V_{DD} with respect to V_{SS} ..... -0.3V to +7.5V
Voltage on $\overline{MCLR}$ with respect to V_{SS} **(Note 2)** ..... 0V to +13.25V
Voltage on RA4 with respect to V_{SS} ..... 0V to +8.5V
Total power dissipation **(Note 1)** 1.0W
Maximum current out of V_{SS} pin ..... 300 mA
Maximum current into V_{DD} pin ..... 250 mA
Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{DD}$) ..... ±20 mA
Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{DD}$) ..... ±20 mA
Maximum output current sunk by any I/O pin ..... 25 mA
Maximum output current sourced by any I/O pin ..... 25 mA
Maximum current sunk by all ports (combined) ..... 200 mA
Maximum current sourced by all ports (combined) ..... 200 mA

Note 1: Power dissipation is calculated as follows:
$Pdis = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD} - V_{OH}) \times I_{OH}\} + \sum(V_{OL} \times I_{OL})$

2: Voltage spikes below V_{SS} at the $\overline{MCLR}/V_{PP}$ pin, inducing currents greater than 80 mA, may cause latch-up. Thus, a series resistor of 50-100Ω should be used when applying a "low" level to the $\overline{MCLR}/V_{PP}$ pin rather than pulling this pin directly to V_{SS}.

Note:	Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

FIGURE 27-1: PIC18FXX8 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)

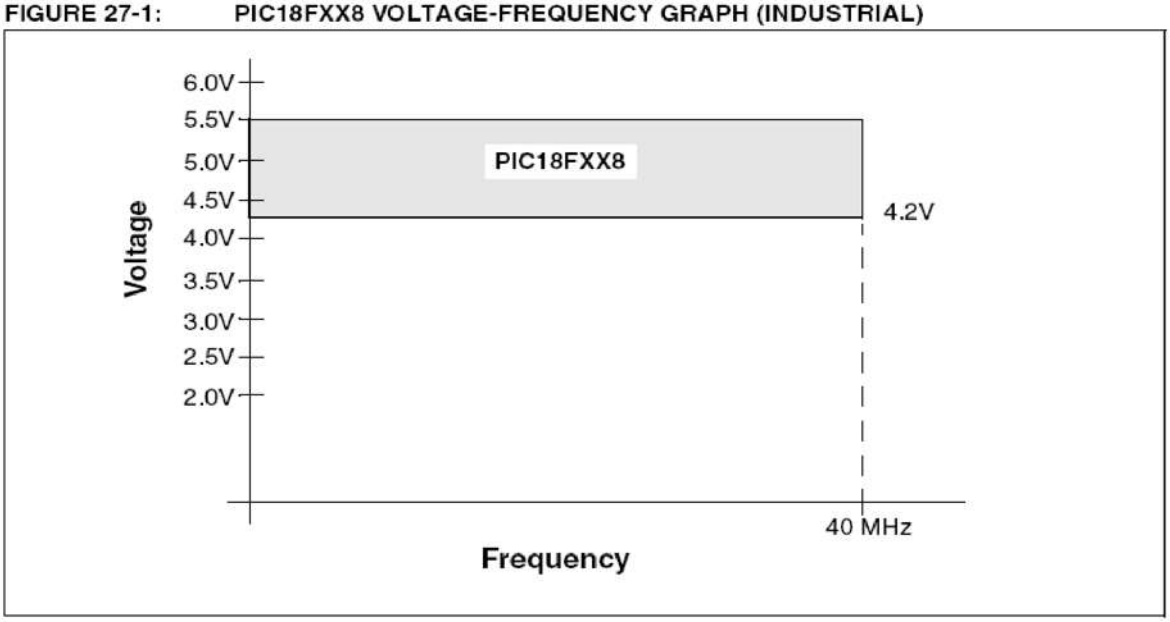


FIGURE 27-2: PIC18FXX8 VOLTAGE-FREQUENCY GRAPH (EXTENDED)

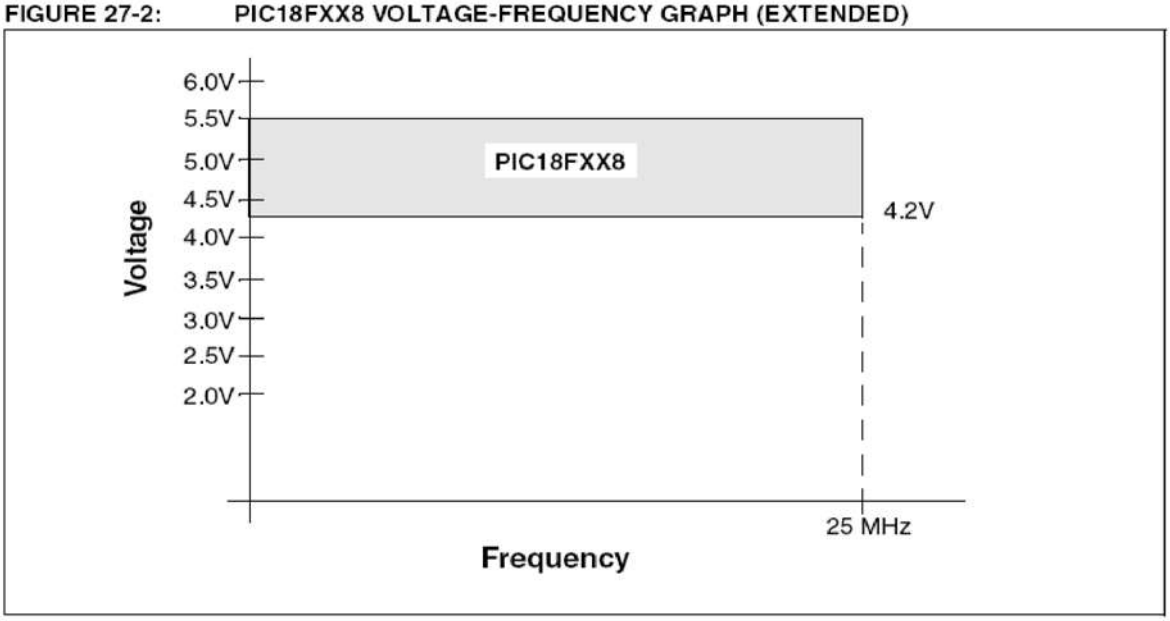
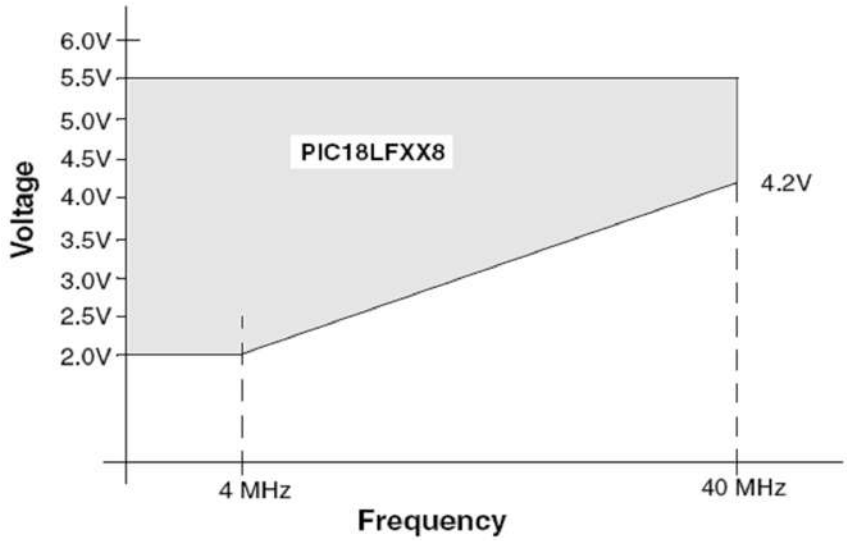


FIGURE 27-3: PIC18LFXX8 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)



FMAX = (16.36 MHz/V) (VDDAPPMIN – 2.0V) + 4 MHz, if VDDAPPMIN ≤ 4.2V = 40 MHz, if VDDAPPMIN > 4.2V

Note: VDDAPPMIN is the minimum voltage of the PICmicro® device in the application.

27.1 DC Characteristics

PIC18LFXX8 (Industrial)			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
PIC18FXX8 (Industrial, Extended)			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic/ Device**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
	VDD	**Supply Voltage**					
D001		PIC18LFXX8	2.0	—	5.5	V	HS, XT, RC and LP Oscillator modes
D001		PIC18FXX8	4.2	—	5.5	V	
D002	VDR	**RAM Data Retention Voltage(1)**	1.5	—	—	V	
D003	VPOR	**VDD Start Voltage** to ensure internal Power-on Reset signal	—	—	0.7	V	See section on Power-on Reset for details
D004	SVDD	**VDD Rise Rate** to ensure internal Power-on Reset signal	0.05	—	—	V/ms	See section on Power-on Reset for details
	VBOR	**Brown-out Reset Voltage**					
		PIC18LFXX8					
D005		BORV1:BORV0 = 11	1.96	—	2.16	V	
		BORV1:BORV0 = 10	2.64	—	2.92	V	
		BORV1:BORV0 = 01	4.07	—	4.59	V	
		BORV1:BORV0 = 00	4.36	—	4.92	V	
		PIC18FXX8					
D005		BORV1:BORV0 = 1x	N.A.	—	N.A.	V	Not in operating voltage range of device
		BORV1:BORV0 = 01	4.07	—	4.59	V	
		BORV1:BORV0 = 00	4.36	—	4.92	V	

Legend: Rows are shaded for improved readability.

Note 1: This is the limit to which VDD can be lowered in Sleep mode, or during a device Reset, without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors, such as I/O pin loading and switching rate, oscillator type, internal code execution pattern and temperature, also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
OSC1 = external square wave, from rail-to-rail; all I/O pins tri-stated, pulled to VDD
$\overline{MCLR}$ = VDD; WDT enabled/disabled as specified.

3: The power-down current in Sleep mode does not depend on the oscillator type. Power-down current is measured with the part in Sleep mode, with all I/O pins in high-impedance state and tied to VDD and VSS and all features that add delta current disabled (such as WDT, Timer1 Oscillator, BOR, ...).

4: For RC oscillator configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2 REXT (mA) with REXT in kOhm.

5: The LVD and BOR modules share a large portion of circuitry. The ΔIBOR and ΔILVD currents are not additive. Once one of these modules is enabled, the other may also be enabled without further penalty.

27.1 DC Characteristics (Continued)

PIC18LFXX8 (Industrial)			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
PIC18FXX8 (Industrial, Extended)			**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic/ Device**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
	IDD	**Supply Current**(2,3,4)					
D010		PIC18LFXX8					XT oscillator configuration
			—	.7	2	mA	VDD = 2.0V, +25°C, FOSC = 4 MHz
			—	.7	2	mA	VDD = 2.0V, -40°C to +85°C, FOSC = 4 MHz
			—	1.7	4	mA	VDD = 4.2V, -40°C to +85°C, FOSC = 4 MHz
							RC oscillator configuration
			—	1	2.5	mA	VDD = 2.0V, +25°C, FOSC = 4 MHz
			—	1	2.5	mA	VDD = 2.0V, -40°C to +85°C, FOSC = 4 MHz
			—	2.5	5	mA	VDD = 4.2V, -40°C to +85°C, FOSC = 4 MHz
							RCIO oscillator configuration
			—	.7	2.5	mA	VDD = 2.0V, +25°C, FOSC = 4 MHz
			—	.7	2.5	mA	VDD = 2.0V, -40°C to +85°C, FOSC = 4 MHz
			—	1.8	4	mA	VDD = 4.2V, -40°C to +85°C, FOSC = 4 MHz
D010		PIC18FXX8					XT oscillator configuration
			—	1.7	4	mA	VDD = 4.2V, +25°C, FOSC = 4 MHz
			—	1.7	4	mA	VDD = 4.2V, -40°C to +85°C, FOSC = 4 MHz
			—	1.7	4	mA	VDD = 4.2V, -40°C to +125°C, FOSC = 4 MHz
							RC oscillator configuration
			—	2.5	5	mA	VDD = 4.2V, +25°C, FOSC = 4 MHz
			—	2.5	5	mA	VDD = 4.2V, -40°C to +85°C, FOSC = 4 MHz
			—	2.5	6	mA	VDD = 4.2V, -40°C to +125°C, FOSC = 4 MHz
							RCIO oscillator configuration
			—	1.8	4	mA	VDD = 4.2V, +25°C, FOSC = 4 MHz
			—	1.8	5	mA	VDD = 4.2V, -40°C to +85°C, FOSC = 4 MHz
			—	1.8	5	mA	VDD = 4.2V, -40°C to +125°C, FOSC = 4 MHz
D010A		PIC18LFXX8					LP oscillator, FOSC = 32 kHz, WDT disabled
			—	18	40	µA	VDD = 2.0V, -40°C to +85°C
D010A		PIC18FXX8					LP oscillator, FOSC = 32 kHz, WDT disabled
			—	60	150	µA	VDD = 4.2V, -40°C to +85°C
			—	60	180	µA	VDD = 4.2V, -40°C to +125°C

Legend: Rows are shaded for improved readability.

Note 1: This is the limit to which VDD can be lowered in Sleep mode, or during a device Reset, without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors, such as I/O pin loading and switching rate, oscillator type, internal code execution pattern and temperature, also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
OSC1 = external square wave, from rail-to-rail; all I/O pins tri-stated, pulled to VDD
$\overline{MCLR}$ = VDD; WDT enabled/disabled as specified.

3: The power-down current in Sleep mode does not depend on the oscillator type. Power-down current is measured with the part in Sleep mode, with all I/O pins in high-impedance state and tied to VDD and VSS and all features that add delta current disabled (such as WDT, Timer1 Oscillator, BOR, ...).

4: For RC oscillator configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2 REXT (mA) with REXT in kOhm.

5: The LVD and BOR modules share a large portion of circuitry. The ΔIBOR and ΔILVD currents are not additive. Once one of these modules is enabled, the other may also be enabled without further penalty.

27.1 DC Characteristics (Continued)

PIC18LFXX8 (Industrial)			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
PIC18FXX8 (Industrial, Extended)			**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic/ Device**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
	IDD	**Supply Current(2,3,4)**					
D010C		PIC18LFXX8	—	21	28	mA	EC, ECIO oscillator configurations VDD = 4.2V, -40°C to +85°C
D010C		PIC18FXX8	—	21	30	mA	EC, ECIO oscillator configurations VDD = 4.2V, -40°C to +125°C, FOSC = 25 MHz
D013		PIC18LFXX8	— — —	1.3 18 28	3 28 40	mA mA mA	HS oscillator configurations FOSC = 6 MHz, VDD = 2.0V FOSC = 25 MHz, VDD = 5.5V HS + PLL osc configuration FOSC = 10 MHz, VDD = 5.5V
D013		PIC18FXX8	— —	18 28	28 40	mA mA	HS oscillator configurations FOSC = 25 MHz, VDD = 5.5V HS + PLL osc configuration FOSC = 10 MHz, VDD = 5.5V
D014		PIC18LFXX8	—	32	65	µA	Timer1 oscillator configuration FOSC = 32 kHz, VDD = 2.0V
D014		PIC18FXX8	— —	62 62	250 310	µA µA	Timer1 oscillator configuration FOSC = 32 kHz, VDD = 4.2V, -40°C to +85°C FOSC = 32 kHz, VDD = 4.2V, -40°C to +125°C
	IPD	**Power-Down Current(3)**					
D020		PIC18LFXX8	— —	0.3 2	4 10	µA µA	VDD = 2.0V, -40°C to +85°C VDD = 4.2V, -40°C to +85°C
D020		PIC18FXX8	—	2	10	µA	VDD = 4.2V, -40°C to +85°C
D021B			—	6	40	µA	VDD = 4.2V, -40°C to +125°C

Legend: Rows are shaded for improved readability.

Note 1: This is the limit to which VDD can be lowered in Sleep mode, or during a device Reset, without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors, such as I/O pin loading and switching rate, oscillator type, internal code execution pattern and temperature, also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
$\overline{\text{OSC1}}$ = external square wave, from rail-to-rail; all I/O pins tri-stated, pulled to VDD
$\overline{\text{MCLR}}$ = VDD; WDT enabled/disabled as specified.

3: The power-down current in Sleep mode does not depend on the oscillator type. Power-down current is measured with the part in Sleep mode, with all I/O pins in high-impedance state and tied to VDD and VSS and all features that add delta current disabled (such as WDT, Timer1 Oscillator, BOR, ...).

4: For RC oscillator configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2 REXT (mA) with REXT in kOhm.

5: The LVD and BOR modules share a large portion of circuitry. The ΔIBOR and ΔILVD currents are not additive. Once one of these modules is enabled, the other may also be enabled without further penalty.

27.1 DC Characteristics (Continued)

PIC18LFXX8 (Industrial)			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
PIC18FXX8 (Industrial, Extended)			**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic/ Device**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
	ΔIWDT	**Module Differential Current**					
D022		**Watchdog Timer**	—	0.75	1.5	µA	VDD = 2.5V, +25°C
		PIC18LFXX8	—	0.8	8	µA	VDD = 2.0V, -40°C to +85°C
			—	7	25	µA	VDD = 4.2V, -40°C to +85°C
D022		**Watchdog Timer**	—	7	25	µA	VDD = 4.2V, +25°C
		PIC18FXX8	—	7	25	µA	VDD = 4.2V, -40°C to +85°C
			—	7	45	µA	VDD = 4.2V, -40°C to +125°C
D022A	ΔIBOR	**Brown-out Reset(5)**	—	38	50	µA	VDD = 2.0V, +25°C
		PIC18LFXX8	—	42	55	µA	VDD = 2.0V, -40°C to +85°C
			—	49	65	µA	VDD = 4.2V, -40°C to +85°C
D022A		**Brown-out Reset(5)**	—	46	65	µA	VDD = 4.2V, +25°C
		PIC18FXX8	—	49	65	µA	VDD = 4.2V, -40°C to +85°C
			—	50	75	µA	VDD = 4.2V, -40°C to +125°C
D022B	ΔILVD	**Low-Voltage Detect(5)**	—	36	50	µA	VDD = 2.0V, +25°C
		PIC18LFXX8	—	40	55	µA	VDD = 2.0V, -40°C to +85°C
			—	47	65	µA	VDD = 4.2V, -40°C to +85°C
D022B		**Low-Voltage Detect(5)**	—	44	65	µA	VDD = 4.2V, +25°C
		PIC18FXX8	—	47	65	µA	VDD = 4.2V, -40°C to +85°C
			—	47	75	µA	VDD = 4.2V, -40°C to +125°C
D025	ΔITMR1	**Timer1 Oscillator**	—	6.2	40	µA	VDD = 2.0V, +25°C
		PIC18LFXX8	—	6.2	45	µA	VDD = 2.0V, -40°C to +85°C
			—	7.5	55	µA	VDD = 4.2V, -40°C to +85°C
D025		**Timer1 Oscillator**	—	7.5	55	µA	VDD = 4.2V, +25°C
		PIC18FXX8	—	7.5	55	µA	VDD = 4.2V, -40°C to +85°C
			—	7.5	65	µA	VDD = 4.2V, -40°C to +125°C

Legend: Rows are shaded for improved readability.

Note 1: This is the limit to which VDD can be lowered in Sleep mode, or during a device Reset, without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors, such as I/O pin loading and switching rate, oscillator type, internal code execution pattern and temperature, also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
OSC1 = external square wave, from rail-to-rail; all I/O pins tri-stated, pulled to VDD
$\overline{MCLR}$ = VDD; WDT enabled/disabled as specified.

3: The power-down current in Sleep mode does not depend on the oscillator type. Power-down current is measured with the part in Sleep mode, with all I/O pins in high-impedance state and tied to VDD and VSS and all features that add delta current disabled (such as WDT, Timer1 Oscillator, BOR, ...).

4: For RC oscillator configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2 REXT (mA) with REXT in kOhm.

5: The LVD and BOR modules share a large portion of circuitry. The ΔIBOR and ΔILVD currents are not additive. Once one of these modules is enabled, the other may also be enabled without further penalty.

Apéndice B

Especificaciones de la tarjeta SD SanDisk

2. Product Specifications

For all the following specifications, values are defined at ambient temperature and nominal supply voltage unless otherwise stated.

2.1. System Environmental Specifications

Table 2-1. System Environmental Specifications

Temperature	Operating: Non-Operating:	25° C to 85° C -40° C to 85° C
Humidity	Operating: Non-Operating:	25% to 95%, non condensing 25% to 95%, non-condensing
ESD Protection	Contact Pads:	± 4kV, Human body model according to ANSI EOS/ESD S5.1-1998
	Non Contact Pad Area:	± 8kV (coupling plane discharge) ± 15kV (air discharge) Human body model per IEC61000-4-2

2.2. Reliability and Durability

Table 2-2. Reliability and Durability Specifications

Durability	10,000 mating cycles
Bending	10N
Torque	0.15N-m or ±2.5 deg
Drop Test	1.5m free fall
UV Light Exposure	UV: 254nm, 15Ws/cm² according to ISO 7816-1
Visual Inspection/Shape and Form	No warpage, no mold skin, complete form, no cavities, surface smoothness ≤ 0.1 mm/cm² within contour, no cracks, no pollution (oil, dust, etc.)
Minimum Moving Force of WP Switch	40 gf (ensures that the WP switch will not slide while it is inserted in the connector)
WP Switch Cycles	Minimum 1,000 Cycles @ slide force 0.4N to 5N

2.3. Typical Card Power Requirements

Table 2-3. Card Power Requirements

VDD (ripple: max, 60 mV peak to peak)	2.7 V 3.6 V

(Ta = 25°C @3 V)

	Value	Measurement	Notes
Sleep	250	uA	Max
Read	65	mA	Max
Write	75	mA	Max

2.4. System Performance

Table 2-4. System Performance

	Typical	Maximum
Block Read Access Time Binary Products MLC Products	 1.5msec 10msec	 100msec 100msec
Block Write Access Time Binary Products MLC Products	 24msec 40msec	 250msec 250msec
CMD1 to Ready (after power up)	50msec	500msec
Sleep to Ready	1msec	2msec

NOTES: All values quoted are under the following conditions:
1) Voltage range: 2.7 V to 3.6 V.
2) Temperature range: -25° C to 85° C.
3) Are independent of the SD Card clock frequency.

2.5. System Reliability and Maintenance

Table 2-5. System Reliability and Maintenance Specifications

MTBF	> 1,000,000 hours
Preventive Maintenance	None
Data Reliability	< 1 non-recoverable error in 10^{14} bits read
Endurance	100,000 write/erase cycles (typical)

2.6. Physical Specifications

Refer to Table 2-6 and to Figures 2-1 through 2-3 for SD Card physical specifications and dimensions.

Table 2-6. Physical Specifications

Weight	2.0 g. maximum
Length:	32mm ± 0.1mm
Width:	24mm ± 0.1mm
Thickness:	2.1mm ± 0.15mm (in substrate area only, 2.25mm maximum)

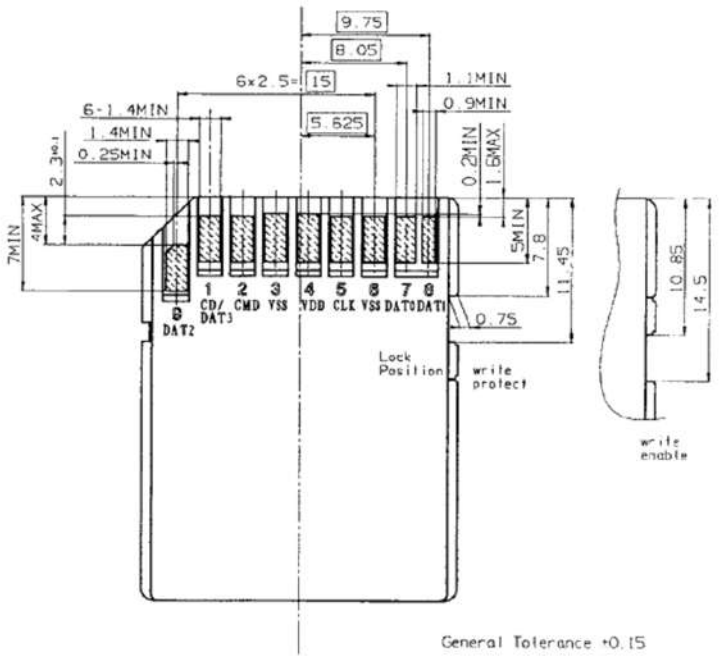


Figure 2-1. SD Card Dimensions

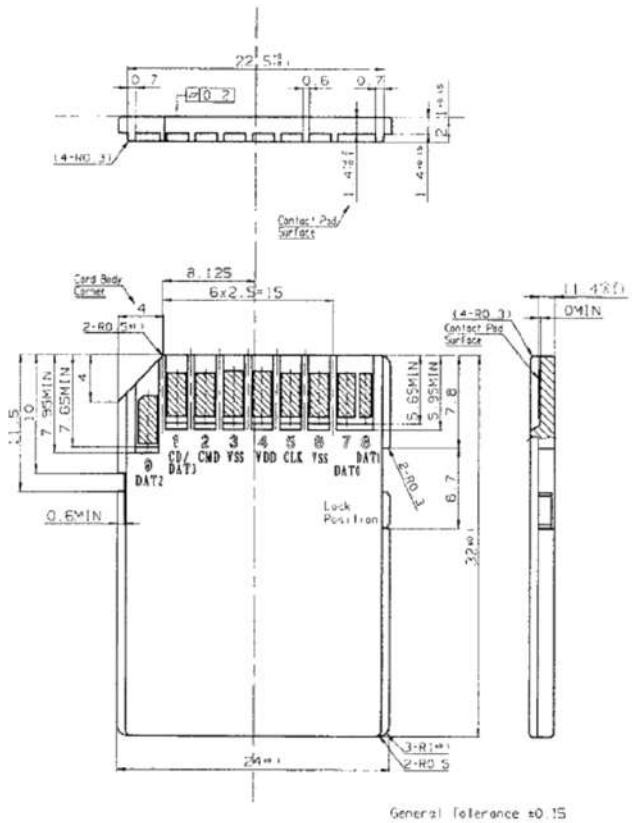


Figure 2-2. SD Card Dimensions

Apéndice C

Código del sistema

```
#include <p18cxxx.h>
#define CS PORTCbits.RC2
#define OK 0
#define ERROR 1
#define BORRA 0x40 + 38
#define READ_CSD 0x49
#define READ_CID 0x4A
#define START_BLOCK 0x40 + 32
#define END_BLOCK       0x40 + 33
#define WRITE_BLOCK   0x40 + 24

            // Forma la estructura de la tarjeta
struct dat_SD{
      unsigned char memoria[5];
      unsigned int Bytes_Sec;
      unsigned int RootDir;
      unsigned int Sec_Directorio;
      unsigned char FATs;
      unsigned int FAT1;
      unsigned int FAT2;
      unsigned int SecPorFAT;
      unsigned int Sec_res;
      unsigned int MaxFilesRoot;
      unsigned int DataIni;
} registro;
```

```
        // Forma la estructura de archivo
struct dat_File{
    unsigned char nombre[8];
    unsigned char extension[3];
    unsigned int inicio;
    unsigned long size;
} file;

#pragma udata bank1 = 0x300
unsigned char buf[512];

unsigned int ReadWord(unsigned int Direccion)
{
unsigned int valor;
    valor = buf[Direccion+1];
    valor = (valor << 8) | buf[Direccion];
    return valor;
}

void SPI_SEND(unsigned char Dato)
{
    SSPBUF = Dato;
    while(!SSPSTATbits.BF);
}

unsigned char SPI_READ(unsigned char Dato)
{
    SSPBUF = Dato;
    while(!SSPSTATbits.BF);
    return SSPBUF;
}
```

```
unsigned char SD_response(unsigned char response)
{
unsigned char count = 0xFF;

      while(SPI_READ(0xFF) != response &&  --count > 0);
      if(count == 0)
            return ERROR;
      else
            return OK;
}

void Init_SPI(void)
{
            TRISCbits.TRISC2=0;           // Pin CS salida
            TRISCbits.TRISC3=0;           // Pin SCK salida
            TRISCbits.TRISC4=1;           // Pin SDI entrada
            TRISCbits.TRISC5=0;           // Pin SDO salida

            SSPSTAT = 0x40;               // Dato en flanco de bajada---se configura para
                                          // que la transmicion acurra en la transicion de
                                          // alto a bajo
            SSPCON1 = 0x22;               // Se configura SSPEN para habilitar modo
                                          // SPI, los ultimos 4 bits de este registro se
                                          // configura el modo maestro y la velocidad
                                          // Se configura tambien para que en el estado
                                          // de espera este en bajo
}

void Init_USART(void)
{
      TXSTA = 0x24;
```

```
    SPBRG = 0x40;
    RCSTA = 0x90;
}

void send_command(unsigned char comando, unsigned char arg1, unsigned char arg2,
                  unsigned char arg3, unsigned char arg4, unsigned char CRC)
{
    SPI_SEND(comando);
    SPI_SEND(arg1);
    SPI_SEND(arg2);
    SPI_SEND(arg3);
    SPI_SEND(arg4);
    SPI_SEND(CRC);
}

unsigned char getchar(void)
{
    while(PIR1bits.RCIF == 0);          // Recibe caracter
    return  RCREG;
}

void putchar(unsigned char dato)
{
    while(TXSTAbits.TRMT == 0);         // Envia caracter
    TXREG = dato;
}

unsigned char Write_block(unsigned long bloque)
{
unsigned char respuesta;
unsigned int j;
```

```
bloque = (bloque<<9);
CS = 0;
respuesta = OK;

send_command(WRITE_BLOCK, (bloque>>24)&0xFF, (bloque>>16)&0xFF,
                    (bloque>>8)&0xFF, bloque&0xFF, 0xFF);
if(SD_response(0x00) == OK)
{
    // SPI_SEND(0xFF);                      // Ciclos de espera
    SPI_SEND(0xFE);                         // Token de inicio de escritura
    for(j=0; j<512; j++)
    {
        SPI_SEND(buf[j]);
    }
    SPI_SEND(0xFF);                         // CRC
    SPI_SEND(0xFF);

    if((SPI_READ(0xFF) & 0x0F) == 0x05)     // Revisa que los datos fueron
                                            // aceptados
    {
        while(SPI_READ(0xFF) != 0xFF);
    }
    else
    {
        respuesta = 2;
    }
}
else
{
    respuesta = 1;
```

```
    }

    SPI_SEND(0xFF);
    CS = 1;
    return respuesta;
}

unsigned char Read_block(unsigned long bloque)
{
unsigned char respuesta;
unsigned int j;
unsigned long address = (bloque<<9);

    CS = 0;
    respuesta = OK;

    send_command(0x51,(addre ss>>24)&0xFF, (address>>16)&0xFF,
                        (address>>8)&0xFF, address&0xFF, 0xFF);
    if(SD_response(0x00) == OK)
    {
        if(SD_response(0xFE) == OK)
        {
            for(j=0; j<512; j++)
            {
                buf[j] = SPI_READ(0xFF);
            }
            SPI_READ(0xFF);
            SPI_READ(0xFF);
        }
        else
        {
```

```
            respuesta = 1;
        }
    }
    else
    {
        respuesta = 2;
    }

    SPI_SEND(0xFF);
    CS = 1;
    return respuesta;
}


void read_CID(void)
{
unsigned char i;

    CS = 0;
    send_command(READ_CID, 0, 0, 0, 0, 0xFF);
    if(SD_response(0x00) == OK)
    {
        if(SD_response(0xFE) == OK)
        {
            for(i=0; i<18; i++)
            {
                buf[i] = SPI_READ(0xFF);
            }
        }
    }
    SPI_SEND(0xFF);
```

```
    CS = 1;
}


void Read_SD(void)
{
    read_CID();
    registro.memoria[0] = buf[3];
    registro.memoria[1] = buf[4];
    registro.memoria[2] = buf[5];                    // Calcula el tamaño de tarjeta SD
    registro.memoria[3] = buf[6];
    registro.memoria[4] = buf[7];

    Read_block(0);

        // Sectore reservados, antes de la FAT
    registro.Sec_res = ReadWord(0x0E);

        // Bytes por sector
    registro.Bytes_Sec = ReadWord(0x0B);

        // Numero de sectores por FAT
    registro.SecPorFAT = ReadWord(0x16);

    registro.FATs = buf[0x10];                       // Numero de FATs

        // Direccion del Directorio Raiz
    registro.RootDir = registro.Sec_res + registro.SecPorFAT * registro.FATs;

        // Direccion de la FAT 1 y Fat 2
    registro.FAT1 = registro.Sec_res;
```

```
    registro.FAT2 = registro.FAT1 + registro.SecPorFAT;

        // Maximo numero de entradas en el directorio raiz
    registro.MaxFilesRoot = ReadWord(0x11);

        // Sectores para el Directorio Raiz
    registro.Sec_Directorio = (registro.MaxFilesRoot * 32) / registro.Bytes_Sec;

        // Direccion de inicio de los datos
    registro.DataIni = registro.Sec_Directorio + registro.RootDir;

}

void Read_File(unsigned int Archivo)
{
        // "Archivo" es el numero de entrada en un bloque del Directorio  Raiz
    file.nombre[0] = buf[Archivo];
    file.nombre[1] = buf[Archivo + 1];
    file.nombre[2] = buf[Archivo + 2];
    file.nombre[3] = buf[Archivo + 3];
    file.nombre[4] = buf[Archivo + 4];
    file.nombre[5] = buf[Archivo + 5];
    file.nombre[6] = buf[Archivo + 6];
    file.nombre[7] = buf[Archivo + 7];

    file.extension[0] = buf[Archivo + 8];
    file.extension[1] = buf[Archivo + 9];
    file.extension[2] = buf[Archivo + 10];

        // CLUSTER inicial del archivo, faltaria sumarle "DataIni"
    file.inicio = buf[Archivo + 27];
```

```
        file.inicio = file.inicio <<8 | buf[Archivo + 26];

        file.size = buf[Archivo + 31];
        file.size = file.size << 8 | buf[Archivo + 30];
        file.size = file.size << 8 | buf[Archivo + 29];
        file.size = file.size << 8 | buf[Archivo + 28];

}

unsigned int Dir(void)
{
unsigned int bloque;
unsigned int archivo;
unsigned char i;
unsigned char resultado, n;

        bloque = 0;
        do
        {
                archivo = 0;
                Read_block(registro.RootDir + bloque);

                // Es el nombre del volumen
                if((bloque == 0) && (archivo == 0))
                {
                        archivo = archivo + 32;
                }
                do
                {
                        // Cero es entrada vacia y E5 es archivo borrado
                        if((buf[archivo] != 0x00) && (buf[archivo] != 0xE5))
```

```
{
    Read_File(archivo);
    i = 0;
    do
    {
        while(TXSTAbits.TRMT == 0);
        TXREG = file.nombre[i];             // Escribe el nombre del Archivo
        i++;
    }while(i < 8);

    putchar(0x20);                          // Da un espacio
    putchar(0x20);

    i = 0;
    do
    {
        while(TXSTAbits.TRMT == 0);
        TXREG = file.extension[i];          // Escribe la extension del
        i++;
    }while(i < 3);

    putchar(0x20);
    putchar(0x20);

    n = 28;
    for(i=0; i<8; i++)                      // Tamaño en hexadecimal
    {
        resultado = (file.size >> n) & 0x0F;    // Obtener cada nibble
        if(resultado < 10)
        {
            resultado = resultado + 0x30;       // Convertir a ascii
```

```
                        }
                        else
                        {
                              resultado = resultado + 0x37;          // En caso de letras
                        }

                        putchar(resultado);
                        n = n - 4;
                  }

                  putchar('\n');
                  putchar('\r');

            }
            // Un archivo en el directorio raíz utiliza 32 bytes
            archivo = archivo + 32;

      // 200 = longitud de un bloque
      }while(archivo < 0x0200);
      bloque++;

}while(bloque < registro.Sec_Directorio);
}

void guardaEn_Directorio(void)
{
unsigned int bloque;
unsigned int entrada;

      bloque = 0;
```

```
do
{
    entrada = 0;
    do
    {
        Read_block(registro.RootDir + bloque);
        if((bloque == 0) && (entrada == 0))
        {
            entrada = entrada + 32;
        }
            // 0x00 es vacio y 0xE5 es archivo borrado
        if(buf[entrada] == 0x00 || buf[entrada] == 0xE5)
        {
            buf[entrada] = file.nombre[0];
            buf[entrada+1] = file.nombre[1];
            buf[entrada+2] = file.nombre[2];
            buf[entrada+3] = file.nombre[3];
            buf[entrada+4] = file.nombre[4];
            buf[entrada+5] = file.nombre[5];
            buf[entrada+6] = file.nombre[6];
            buf[entrada+7] = file.nombre[7];

            buf[entrada+8] = file.extension[0];
            buf[entrada+9] = file.extension[1];
            buf[entrada+10] = file.extension[2];

                // Atributos (los mismo de otros archivos)
            buf[entrada+11] = 0x20;

            buf[entrada+12] = 0x18;
            buf[entrada+13] = 0x43;
```

```
                    buf[entrada+14] = 0x70;
                    buf[entrada+15] = 0xBE;
                    buf[entrada+16] = 0x92;
                    buf[entrada+17] = 0x37;
                    buf[entrada+18] = 0x92;
                    buf[entrada+19] = 0x37;

                        // TIME y DATE (tambien copiados de otro archivo)
                    buf[entrada+22] = 0xF9;
                    buf[entrada+23] = 0x90;
                    buf[entrada+24] = 0xC7;
                    buf[entrada+25] = 0x36;

                        // CLUSTER inicial del archivo, faltaria sumarle "DataIni"
                    buf[entrada + 27] = file.inicio>>8;          // mas significativo
                    buf[entrada + 26] = file.inicio & 0x00FF;    // menos significativo

                    buf[entrada+28] = (file.size & 0xFF);
                    buf[entrada+29] = (file.size>>8) & 0xFF;
                    buf[entrada+30] = (file.size>>16) & 0xFF;
                    buf[entrada+31] = (file.size>>24) & 0xFF;

                    Write_block(registro.RootDir + bloque);
                    return;             }
                entrada = entrada + 32;
            }while(entrada < 512);
            bloque++;
        }while(bloque < registro.Sec_Directorio);
}
void guardaEn_FAT(void)
{
```

```
unsigned int bloque;
unsigned int Contenido;
unsigned int i;

    bloque = 0;
    do
    {
        Read_block(registro.FAT1 + bloque);
        i = 0;
        do
        {
            Contenido = ReadWord(i);
            i++;
            i++;
        }while((Contenido != 0x0000) && (i < 512));
        if(Contenido == 0x0000)
        {
            i = i-2;
            file.inicio = ((bloque << 8) + (i >> 1));          //(X<<8) = (X*256)
            buf[i] = 0xFF;                                     //(X>>1) = (X/2)
            buf[i+1] = 0xFF;
            Write_block(registro.FAT1 + bloque);
            break;
        }
        bloque++;
    }while(bloque < registro.SecPorFAT);
}

void Save(void)
{
unsigned int disponible;
```

```
unsigned char caracter;
unsigned int i;

        // elementos que no se ocupan llevan "espacio"
    for(i=0; i<8; i++)
        file.nombre[i] = ' ';
    for(i=0; i<3; i++)
        file.extension[i] = ' ';

    i = 0;
    while(1)
    {
        caracter = getchar();
        putchar(caracter);
        if((caracter == '\r') || (caracter == '.'))
            break;
        if(i<8)
            file.nombre[i] = caracter;
        i++;
    }

    if(caracter == '.')
    {
        i = 0;
        while(1)
        {
            caracter = getchar();
            putchar(caracter);
            if(caracter == '\r' || (caracter == ' '))
                break;
            if(i<3)
```

```
                file.extension[i] = caracter;
            i++;
        }
        if(caracter == ' ')
        {
            guardaEn_FAT();
            for(i=0; i<512; i++)                    // Limpia buffer con ceros
            {
                buf[i] = 0;
            }
            i = 0;
            while(1)
            {
                caracter = getchar();
                if(caracter == '\r')
                    putchar('\n');
                putchar(caracter);
                if(caracter == 26)              //CTRL+Z indicador de fin de archivo
                {
                    break;
                }
                buf[i] = caracter;
                i++;
            }
            file.size = i;
            Write_block(file.inicio + registro.DataIni  - 2);
            guardaEn_Directorio();
        }
    }
}
```

```
void main(void)
{
unsigned char i, respuesta;
unsigned int a_ver;
unsigned char comando;
unsigned long diskFree;

    Init_SPI();
    Init_USART();

    CS = 1;
    for(i=0;i<10;i++)
        SPI_SEND(0xFF);
    CS = 0;
    send_command(0x40,0,0,0,0,0x95);            // Reset y modo IDLE
    if((SD_response(0x01)) == OK)               // La tarjeta respondió
    {
        i=0;
        while((i < 255) && (SD_response(0x00) == ERROR))
        {
            send_command(0x41,0,0,0,0,0xFF);    // Activa el proceso de
                                                // inicialización de la tarjeta
            i++;
        }
    }
    CS = 1;
    SPI_SEND(0xFF);
    CS = 0;
    do
    {
        send_command(0x50,0,0,2,0,0xFF);        // Bloques de 512 bytes
```

```
    }while(SD_response(0x00) == ERROR);
    CS = 1;
    SPI_SEND(0xFF);

    Read_SD();
    while(1)
    {
        comando = getchar();
        putchar(comando);
        if(comando == 'D')
        {
            putchar('\n');
            putchar('\r');
            Dir();
        }
        if(comando == 'S')
        {
            comando = getchar();
            putchar(comando);
            if(comando == ' ')
            {
                Save();
                putchar('\n');
                putchar('\r');
            }
        }
    }
}
```

Referencias

[Microchip 2006]

Microchip Technology Inc., PIC18FXX8 Data Sheet, 2006

[Secure Digital Card 2003]
SanDisk Secure Digital Card, Product Manual, 2003

[Evans 2004]
Application Note Fat16 Interface for MSP430, Dept. of Electrical and Computer Engineering, Michigan State University, 2004

[Axelson 2006]
Jan Axelson, USB Mass Storage, Designing and Programming Devices and Embedded Hosts, 2006

[Semiconductor 2006]
National Semiconductor Corporation, LM317 Terminal Adjustable Regulator, 2006

[Maxim 1999]
Maxim Integrated Products, MAX3232, 1999