Acceso a Tarjetas de Memoria de Estado Sólido

Realizado por:

Jordi Ortiz Verdu
Luis Gigante Beneito
Pau Cucart Galbis
Asmae el Moudden
Daniel Abad Linares

Laboratorio de Sistemas Electrónicos Digitales

Universidad Politécnica de Valencia

EPSA

Martes, 1 de Abril de 2008.

INDICE

Pág.

1. Introducción de las memorias SD/MMC

1.1.- Tarjetas de memoria

Una tarjeta de memoria es un sistema de almacenamiento informático para dispositivos portátiles como cámaras digitales, reproductores de MP3 o impresoras. También las actuales videoconsolas hacen uso de tarjetas de memoria específicas para cada una.

1.2.- La tarjeta de memoria SD

SD significa Secure Digital, es un formato de tarjeta de memoria flash[1]. Se utiliza en dispositivos portátiles tales como cámaras fotográficas digitales, la Wii, ordenadores PDA, entre otros.

Estas tarjetas tienen unas dimensiones de 32 mm x 24 mm x 2'1 mm.

Fig.1-tarjeta SD.

Las tarjetas Secure Digital son utilizadas como soportes de almacenamiento por algunos dispositivos portátiles como:

- Cámaras digitales, para almacenar imágenes.
- La consola Wii, la cual dispone de un lector de este tipo de tarjetas, que ofrece la posibilidad de guardar datos relativos a los juegos.
- Videocámaras, para almacenar tanto imágenes instantáneas como videos.
- PDA, para almacenar todo tipo de datos.
- Teléfonos móviles, para almacenar imágenes, archivos de sonido y otros archivos multimedia.
- Palm.

Existen dos tipos: unos que funcionan a velocidades normales, y otros de alta velocidad que tienen tasas de transferencia de datos más altas. Ya que algunas cámaras fotográficas digitales requieren tarjetas de alta velocidad para poder grabar vídeo con fluidez o para capturar múltiples fotografías en una sucesión rápida.

Los dispositivos con ranuras SD pueden utilizar tarjetas MMC, que son más finas, pero las tarjetas SD no se pueden introducir en las ranuras MMC. Sus variantes MicroSD y MiniSD se pueden utilizar, también directamente, en ranuras SD mediante un adaptador.

1 La memoria flash es una forma evolucionada de la memoria EEPROM que permite que múltiples posiciones de memoria sean escritas o borradas en una misma operación de programación mediante impulsos eléctricos, frente a las anteriores que sólo permite escribir o borrar una única celda cada vez. Por ello, flash permite funcionar a velocidades muy superiores cuando los sistemas emplean lectura y escritura en diferentes puntos de esta memoria al mismo tiempo.

1.3.- Introducción al mercado.

En 2005, las capacidades típicas de una tarjeta SD eran de 128, 256 y 512 megabytes, y 1, 2 y 4 gigabytes. En 2006, se alcanzaron los 8 GB, y en 2007, los 16 GB. El 22 de agosto de 2007 Toshiba anunció que para 2008 empezará a vender memorias de 32 GB.

Las tarjetas SD han sustituido a las SmartMedia como formato de tarjeta de memoria dominante en las cámaras digitales compactas. En 2001, SmartMedia había logrado un uso de cerca del 50%, pero en 2005, SD/MMC había alcanzado más de un 40% de cuota de mercado, y el porcentaje de SmartMedia cayó en picado. La gran mayoría de los principales fabricantes de cámaras fotográficas digitales usa SD en sus líneas de productos, como Canon, Nikon, Kodak y Konica Minolta. Sin embargo, tres fabricantes importantes se han adherido a sus propios formatos propietarios en sus cámaras fotográficas: Olympus y Fuji, que usan tarjetas XD; y Sony con su Memory Stick. Además, el SD no ha conquistado el mercado de Digital SLR, donde Compact Flash sigue siendo el formato más popular.

Secure Digital, viene del origen de la tarjeta. Para crear una tarjeta SD, Toshiba añadió hardware de cifrado a la ya existente tarjeta MMC, para aliviar las preocupaciones de la industria de la música, que giraban en torno a que las tarjetas MMC permitirían el pirateo fácil de la música. En teoría, este cifrado permitiría cumplir fácilmente los esquemas DRM (en español, Gestión de derechos digitales) sobre la música digital, pero esta funcionalidad se utiliza poco.

1.4.- Apertura de estándares.

SD está cubierto por numerosas patentes y marcas registradas, y sólo se puede licenciar a través de la Secure Digital Card Association (Asociación de la Tarjeta Secure Digital). El acuerdo de licencia actual de esta organización no permite controladores de código abierto para lectores de tarjetas SD, un hecho que genera consternación en las comunidades de código abierto y software libre. Generalmente, se desarrolla una capa de código abierto para un controlador SD de código cerrado disponible en una plataforma particular, pero esto está lejos de ser lo ideal.

1.5.- Diferentes tipos de tarjetas MMC/SD

La tarjeta SD no es el único estándar de tarjetas de memoria flash ratificado por la Secure Digital Card Association. Existen otros formatos de dicha asociación, como son el MiniSD y el MicroSD.

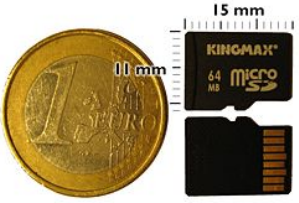


Fig.2-tarjeta MicroSD.

MiniSD es un formato de tarjeta de memoria flash. La tarjeta MiniSD fue adoptada como una extensión de tamaño ultra-pequeño para el estándar de tarjeta SD.
MicroSD o Transflash corresponden a un formato de tarjeta de memoria flash más pequeña que la MiniSD, desarrollada por SanDisk. Es casi cuatro veces más pequeña que la MiniSD, que era hasta ahora el formato más pequeño de tarjetas SD, y es alrededor de un décimo del volumen de una SD Card. Sus tasas de Transferencia no son muy altas, sin embargo, empresas como SanDisk han trabajado en ello, llegando a versiones que soportan velocidades de lectura de hasta 10Mbps.

2. conexión física de la memoria MMC/SD

2.1.- Pines de la SD

Esta tarjeta de memoria tiene tres modos de conexión, pero en este caso el estudio se centrará en el bus SPI por su funcionalidad, la minimización de pines y porque el microcontrolador tiene incorporado un módulo para SPI.

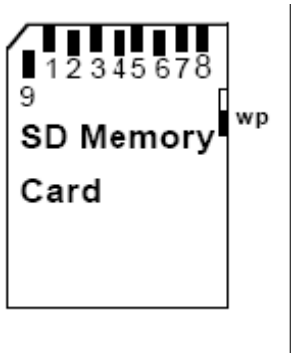


Fig.3- Pines de la Tarjeta SD

Las tarjetas SD tienen 9 pines de conexión distribuidos como muestra la figura anterior, mas uno de bloqueo manual de los cuales para la comunicación en bus SPI solo necesitamos 4 de datos más 3 de alimentación(2 GND i VDD).

DI (pin 2): Entrada de datos.
DO (pin 7): Salida de datos
CLK (pin 5): Señal de reloj
CS (pin 1): Chip select, activa a nivel bajo.
VDD (pin 4): Alimentación a 3.3 V.
GND (pin 3 - 6): Alimentación 0V.

Pin	Name	Type	SD Description
1	CS/DAT32	I/O3	Chip Select/Data Line [Bit 3]
2	CMD/DI	I/O	Command/Data In (SPI)
3	VSS1	S	Supply voltage ground
4	VDD	S	Supply voltage
5	CLK	I	Clock
6	VSS2	S	Supply voltage ground
7	DAT0	I/O	Data Line [Bit 0]
8	DAT1	I/O	Data Line [Bit 1]
9	DAT2	I/O	Data Line [Bit 2]

Fig.4- Tabla de los pines de conexión de las SD

2.2.- Zócalos para la tarjeta

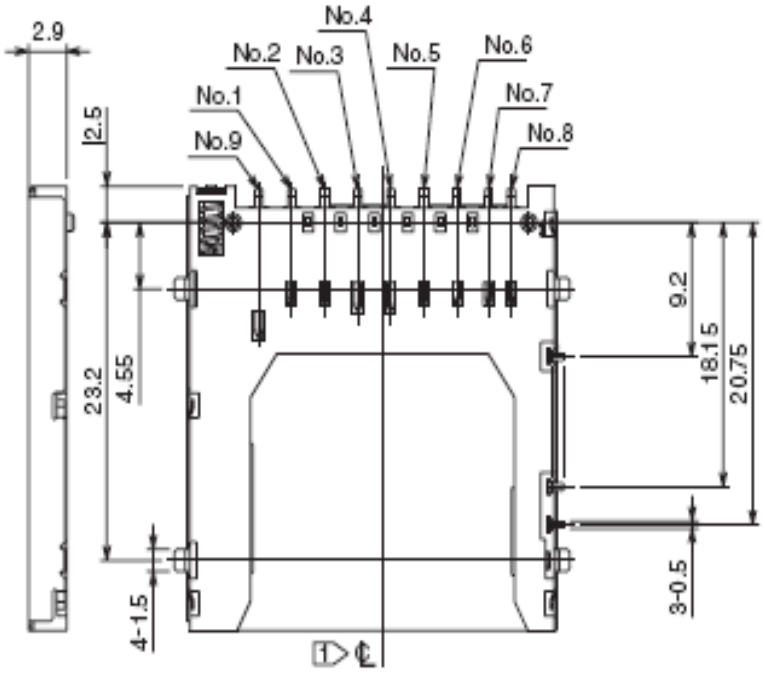


Fig.5- Vista frontal

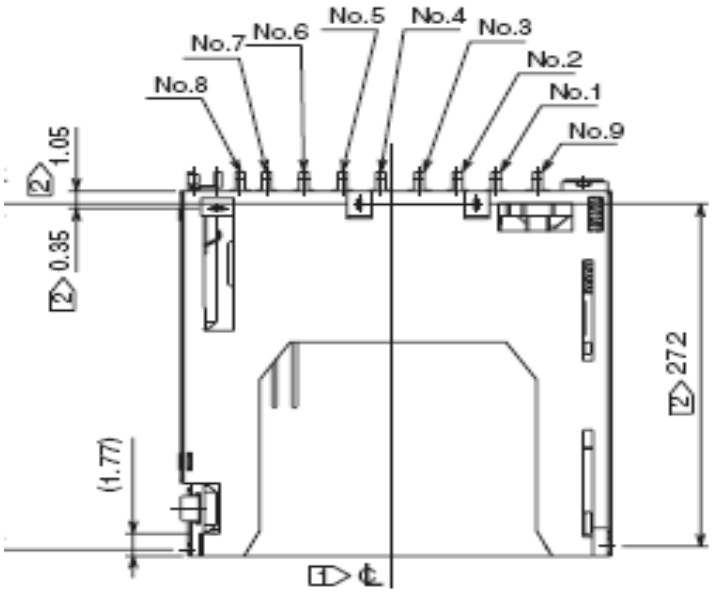


Fig.6- Vista trasera

2.3.- Esquema de Conexión física del conector al micro DSPIC4013

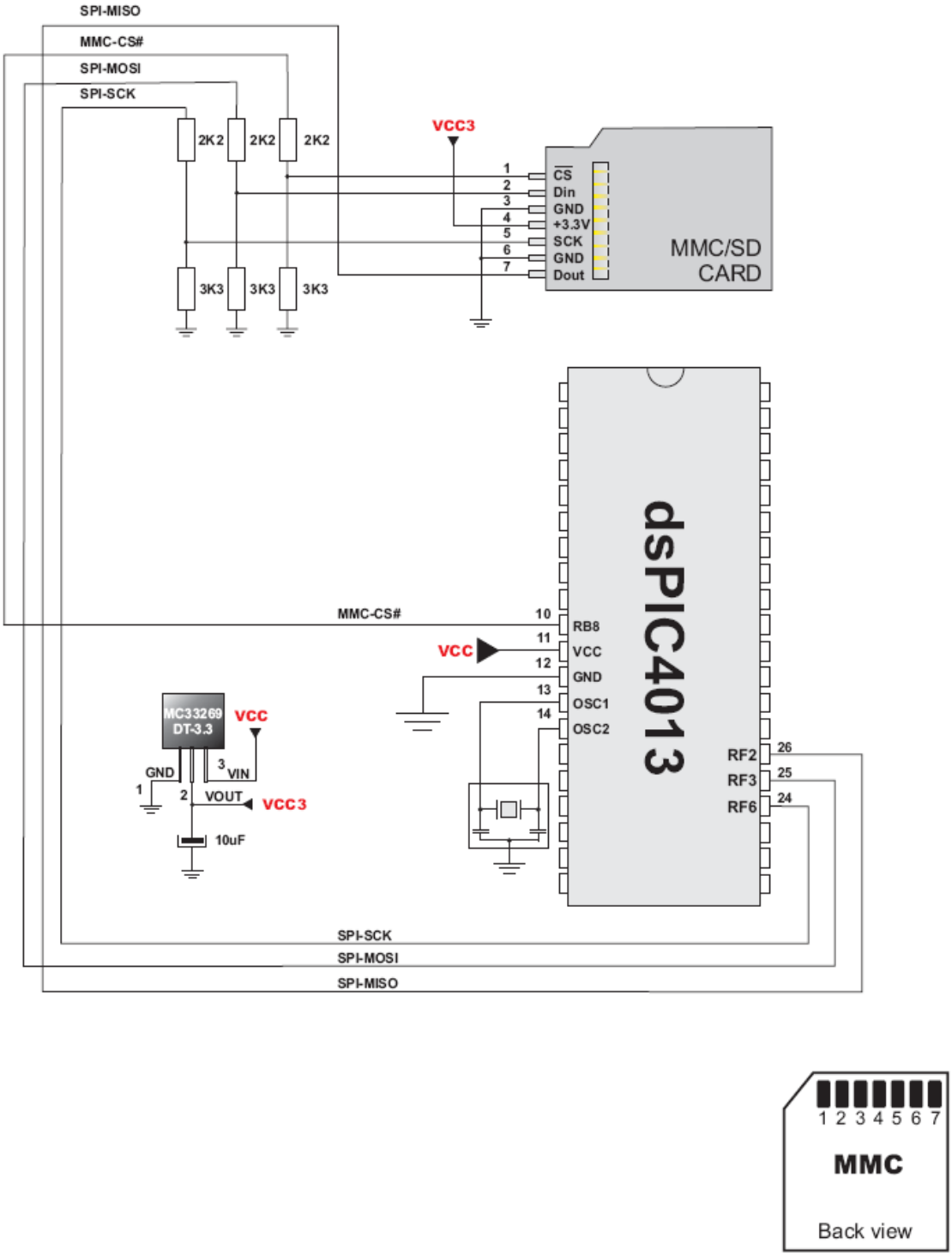
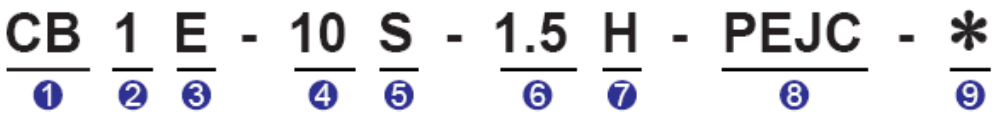


Fig.7- Esquema de conexión de la tarjeta con el microcontrolador

2.4.- Códigos de los conectores de tarjetas SD



1. Nombre de serie.
2. Número de serie
3. Tipo de expulsador: -C, E, F, G con mecanismo de expulsión.
   -A, D sin mecanismo de expulsión.
4. Número de contactos: 10
5. Tipo de conector: S: Receptor
6. Brea de contacto.
7. Superficie de soporte
8. Códigos del mecanismo de expulsión: -PEJC: Card push insert/ Push eject
   -EJL: Left button eject
   -RJL: Right button eject
9. Sufijo

3. El bus SPI

3.1.- Introducción

El SPI, o Serial Peripheral Interface (Interfaz de Periféricos en Serie) es un bus de comunicaciones entre dos dispositivos, uniéndolos de forma síncrona y mediante un enlace serie que funciona en modo full dúplex.

Se considera un estándar de facto de comunicaciones, ya que no está aceptado por un organismo de regulación internacional como ISO o IEEE, debido posiblemente a su evidente simplicidad. Esto generará diversas aplicaciones que resultarán en ocasiones incompatibles entre sí.

Desarrollado por Motorola, SPI distribuye los dispositivos entre maestros y esclavos, siendo el primero el que inicia la comunicación con los esclavos. Se conoce también como bus serie de cuatro cables, para diferenciarlos con los de tres, dos y un cable.

3.2.- Símbolos lógicos

Las especificaciones de esta tecnología describen cuatro señales lógicas que rigen el comportamiento del bus:

1. SLCK: Señal de reloj, comandada por el maestro.
2. MOSI: Máster Output, Slave Input. Transmisión del maestro al esclavo.
3. MISO: Máster Input, Slave Output. Transmisión del esclavo al maestro.
4. SS: Slave Select. El maestro activa el dispositivo esclavo correspondiente mediante esta salida activa por nivel bajo.

3.3.- Tipos de conexión

El Serial Peripheral Interface puede operar con un único dispositivo en modo maestro y, necesariamente, un esclavo que dependa de él, aunque puede manejar simultáneamente más de un dispositivo conectado a él; es decir, varios esclavos.

En el caso de un sólo esclavo, la comunicación es bien sencilla, ya que los dispositivos estarán conectados sin interrupción y transmitiéndose los datos entre ellos sin esperar más que a la señal de reloj que habilite este proceso. Por lo tanto, la señal de SS siempre está activa (nivel bajo) hasta que no haya más datos que transmitir y se finalice la conexión

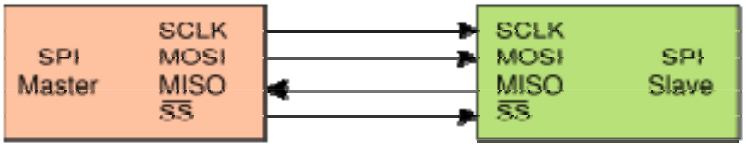


Fig.8- Bus SPI simple, con un maestro y un esclavo

Sin embargo, cuando se tiene más de un dispositivo esclavo conectado al maestro, primeramente nos encontramos con la problemática de qué dispositivo se comunica en cada momento con el maestro. Para solventar este inconveniente, se han ideado dos sistemas de conexión que permiten realizar este tipo de circuitos de forma fácil.

En primer lugar, la solución más empleada en este caso es la llamada configuración de esclavos independientes, consistente en asignar a cada dispositivo esclavo una salida diferente de Slave Select desde el maestro, cortocircuitando todas las entradas y salidas MISO y MOSI de los dispositivos esclavos respectivamente, debiendo estás ser tri-estado. Esto significa que deben poder convertirse en conexiones de alta impedancia cuando no se les requiera, es decir, cuando el maestro no las tenga en consideración para realizar una transmisión, se simulará una desconexión física del cable.

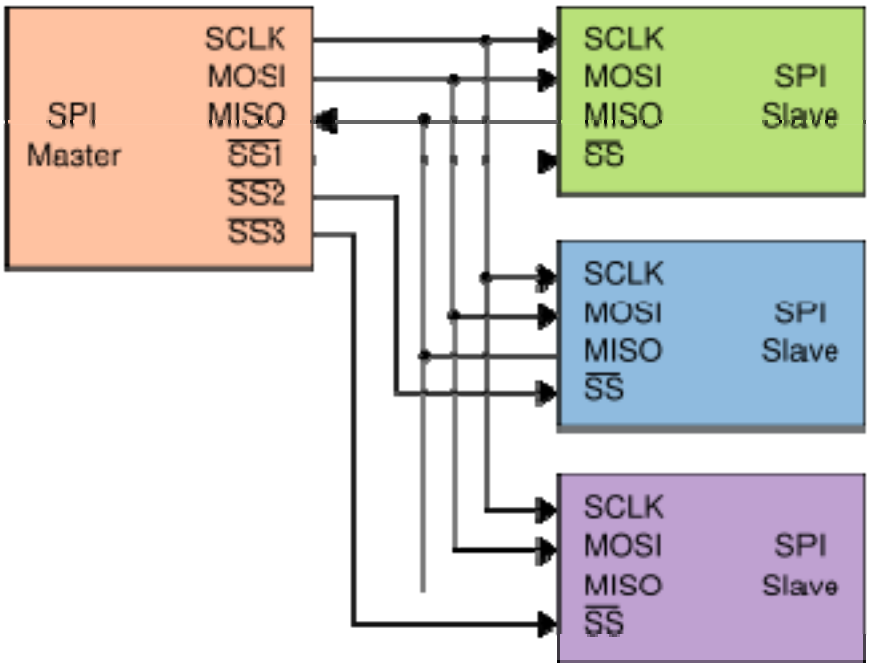


Fig.9- Bus SPI con configuración independiente de esclavos

El dispositivo maestro es el que se encargará de distribuir la conectividad y seleccionar en cada momento el dispositivo esclavo con el que se quiere conectar para realizar una transmisión. Obviamente, la señal de reloj es compartida.

Alternativamente existe la disposición con esclavos en cadena. Este sistema está disponible en ciertos productos, y consiste en ir conectando las salidas de datos del esclavo anterior a la entrada de datos del siguiente, y así sucesivamente hasta que la salida MISO del último esclavo se conecta directamente a la entrada MISO del maestro. De este modo, los datos van trasladándose entre dispositivos esclavos en cada ciclo de reloj hasta alcanzar el maestro.

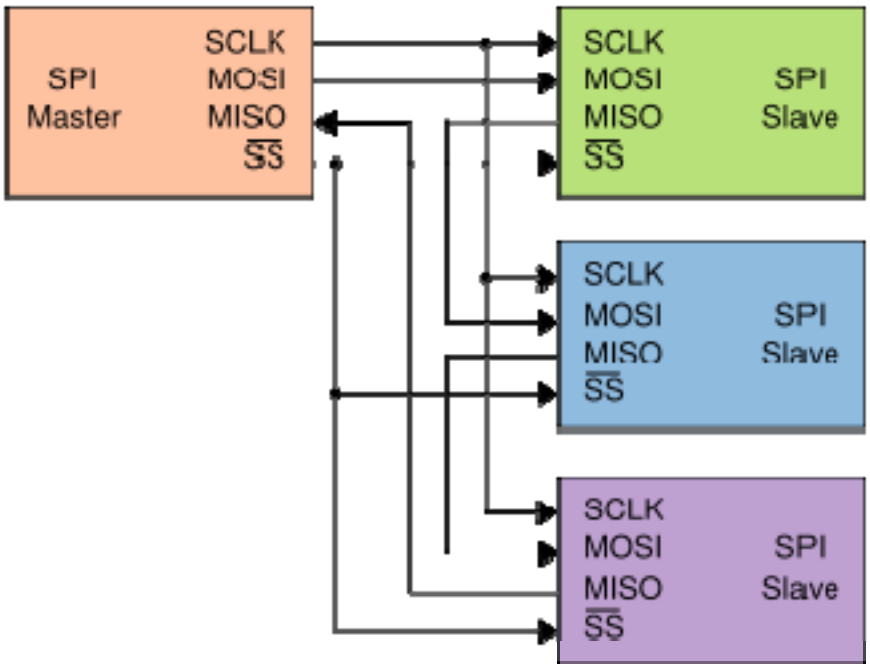


Fig.10-Bus SPI con configuración de esclavos en cadena

Como se puede observar en la figura, la entrada de Slave Select está compartida por todos los dispositivos, activándose todos simultáneamente. Esto no genera problemas de transmisión debido a que tan sólo el primero y el último de los dispositivos esclavos se comunican directamente con el maestro.

3.4.- Transmisión

Pero para empezar una comunicación, primeramente el maestro configura el reloj a una frecuencia menor o igual al de la máxima frecuencia soportada por el dispositivo esclavo. En el caso de las tarjetas SD, se trata de 25 MHz para las tarjetas normales y de 50 MHz para tarjetas de alta velocidad.

Después sitúa la línea de selección de esclavo en nivel bajo para el dispositivo con el que se quiere conectar y puede empezar la transmisión de datos.

Esta transmisión en cada ciclo de reloj de SPI se compone típicamente de dos sucesos. El maestro envía un bit por la línea MOSI y el esclavo la lee, mientras que paralelamente el esclavo envía otro bit por la línea MISO para que sea leída por el maestro.

Generalmente, y poniendo como situación la más simple de un único dispositivo maestro y otro esclavo para una transmisión, la típica configuración de hardware para intercambiar datos son dos registros de desplazamiento de un tamaño de palabra dado, por ejemplo 8 bits, que forman una especie de buffer circular.

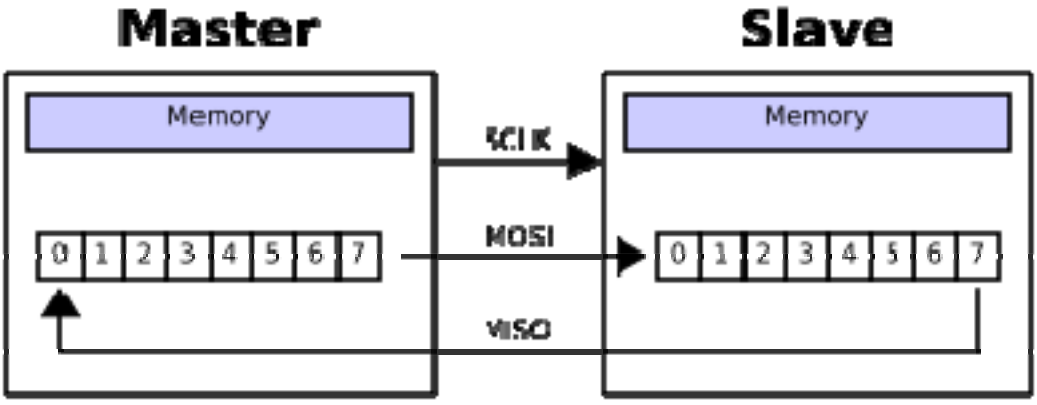


Fig.11- Esquema de transmisión. Se observa el buffer en círculo.

Los datos son transmitidos normalmente iniciando el envío del bit más significativo, mientras que el bit recibido llega para ocupar la posición menos significativa en el registro. El proceso se repite tantas veces sea necesaria como para la transmisión de los dos registros se complete y se realiza alguna operación con ellos, por ejemplo, copiarlos a la memoria del dispositivo. Si hay más datos para enviar, los registros se recargan con nuevos datos y el proceso se repite. Finalmente, cuando la transmisión finaliza, el maestro detiene la señal de reloj y se desactiva el esclavo.

Como se puede observar y como se ha dicho anteriormente, se trata de una comunicación full-dúplex, pues se envían y se reciben datos simultáneamente.

Una de las consecuencias de que el SPI no esté regularizado por ningún organismo internacional provoca que haya dispositivos con diferentes tamaño de palabra en los registros, aunque habitualmente son de 8 bits (1 byte). Por ejemplo, en muchos controladores para pantallas táctiles o codificadores de audio se emplean tamaños de palabra de 16 bits, y varios conversores digitales-analógicos y analógicos-digitales usan tamaños de palabra de 12 bits.

3.5.- Ventajas e inconvenientes

Como en todos los casos, toda tecnología tiene sus pros y sus contras:

Ventajas

1. Comunicación full-dúplex
2. Mejor rendimiento que otros protocolos como I^2C o SMBus
3. Flexibilidad
4. Simple de implementar
5. Menor consumo de energía respecto a I^2C y SMBus.
6. Buses compartidos

Inconvenientes

7. No tiene control de flujo
8. Sin fiabilidad
9. Envío a corta distancia

3.6- Aplicaciones

El sistema SPI debido a su simplicidad ha tenido una gran penetración en sistemas “todo en uno”, como en microcontroladores como el PIC que se usa en la asignatura, ya que implementan un controlador capaz de hacerlo funcionar en modo maestro o esclavo.

Además, su capacidad full-dúplex la hacen muy eficiente en sistemas de alto rendimiento como audio digital, procesado digital de señal o canales de telecomunicaciones.

En cuanto a periféricos. SPI se usa en una buena variedad de éstos: sensores, dispositivos de control, comunicaciones como Ethernet, USB o CAN; memorias flash y EEPROM, pantallas LCD y las protagonistas del presente trabajo, tarjetas de memoria removible MMC y SD.

4. Software de la comunicación para la tarjeta SD/MMC

4.1.- Introducción

Desde el punto de vista técnico, estas memorias pueden trabajar mediante dos protocolos serie distintos: el protocolo MultiMediaCard propiamente dicho (SD), y el protocolo SPI.

El primero de los protocolos es el más potente ya que permite más operaciones que el segundo, pero por otro lado, el segundo es más fácil de implementar si se dispone de una interfaz SPI y es suficiente para la mayoría de aplicaciones. De hecho el protocolo SPI se puede considerar como una versión reducida del protocolo MultiMediaCard. Los siguientes apartados muestran como acceder las tarjetas SD/MMC mediante el protocolo SPI y también como trabajar con ficheros almacenados en la tarjeta formateada en FAT/FAT16. Esto es lo que realmente será útil para leer y escribir datos también desde el ordenador.

Nota: Las siguientes librerías solo pueden escribir y leer ficheros guardados en la carpeta principal.

Nota: Antes de escribir, estar seguro de no sobrescribir sectores reservados para el formato FAT, que podrían dejar la tarjeta ilegible en PCs u otros dispositivos. Programas para organizar mapeado de memoria, como el Winhex, pueden ser de gran ayuda.

El programa “mikroC para dsPIC30/33 y PIC24” es una buena herramienta que proporciona las librerías necesarias para el uso de la memoria MMC/SD usando el bus SPI. Todas las funciones que aparecen en este manual están disponibles en este compilador. Se puede descargar en la página de siguiente:
http://www.mikroe.com/en/compilers/mikroc/pic/download.htm.

Antes que todo, para entender las siguientes funciones hay que definir lo siguiente:

Siempre que se diga de activar el pin de “chip enabled” lo que hay que hacer es poner a “0” el bit 8 del registro PORTB, que “interactúa con el exterior del chip”, esto lo que hará es que pase de nivel alto (5v de tensión) a nivel bajo (0 voltios de tensión).
La función es: SET_BIT (del puerto PORTX, el pin N).
En este caso, la patilla de la MMC de CS está conectada a PORTB.8.

Siempre que se hable de enviar un dato por SPI, lo que hay que hacer es llamar a la función ENVIA_SPI (variable “enviar_por_spi” es byte).
Esto lo que hará, a más bajo nivel será, “jugando” con las patillas 2, 3 y 6 del puerto PORTF, enviar el contenido de la variable “enviar_por_spi” por el puerto serie.

4.2.- Comandos para inicializar y manejar el bus SPI

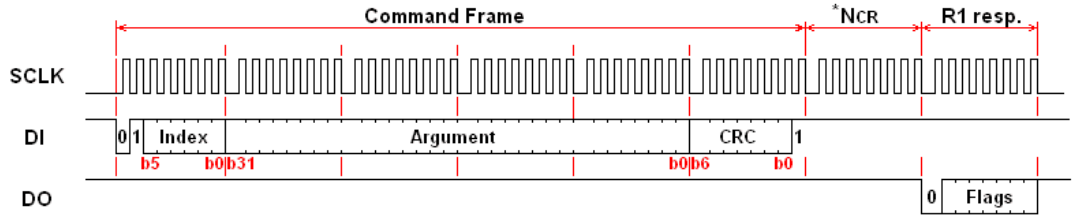


Fig.12- Datagrama de comunicación PIC-MMC mediante SPI

4.2.1.- Comandos MODO SPI

Los comandos son bloques de bytes con un formato fijo:

Byte 1			Byte 2-5	Byte 6	
Num bit:	7 6	5-0	31-0	7	0
Valor bit:	0 1	Command	Command Argument	CRC	1

Fig.13- Estructura de un comando

La tarjeta identifica los comandos porque el primer byte de estos (byte 1) siempre comienza por 01, el resto de bits del primer byte contiene el número de comando codificado en binario natural.
Así el primer byte correspondiente al comando 0 (CMD0) seria: 01000000, o el primer byte correspondiente al comando 39 (CMD39) sería: 01100111. Los siguientes 4 bytes (bytes 2–5) contienen los argumentos del comando. En los comandos que no requieren argumentos estos valen 0. El último byte (byte 6) es el byte de CRC para la verificación de errores y en realidad en el protocolo SPI no se utiliza, a no ser que en el registro de configuración se especifique que se desea utilizar CRC. Para usarlo, antes se debe de activar la verificación de CRC mediante ACMD41. Mediante el comando CRC_ON_OFF (CMD59) este se deshabilita.

Toda la comunicación entre la tarjeta y el controlador se realiza según el orden del esquema, de izquierda a derecha, es decir que primero se transmite el bit de más peso (bit 7) del byte 1, y por último el bit de menos peso (bit 0) del byte 6.
Es una transferencia More Significant Bit First (MSB First).

4.2.2.- Respuestas MODO SPI

Las respuestas de la tarjeta son bloques formados por 1 ó 2 bytes (R1 y R2 respectivamente), dependiendo del tipo de respuesta que se trate. El tipo de respuesta está asociado a cada comando.

Bit	
0	Idle State
1	Erase Reset
2	Illegal Command
3	Com CRC Error
4	Erase_Seq_Error
5	Address Error
6	Parameter Error
7	0

Fig.14- Estructura del Byte de la respuesta R1.

Bit	
0	0
1	WP Erase Skip
2	Error
3	CC Error
4	Card ECC Failed
5	WP Violation
6	Erase Parameter
7	Out of Range

Fig.15- Estructura del 1er byte de la respuesta R2.
(El 2º byte de R2 es el mismo que R1)

4.2.3.- Bloques de datos MODO SPI

Los bloques de datos comienzan siempre con el byte 0xFE, a éste le siguen los bytes de datos y por último los 2 bytes de CRC. El número de bytes de datos depende de la longitud de bloque definida mediante el comando 16, y ésta puede ir de 1 hasta 512 bytes (por defecto 512). Por tanto, sumando a los bytes de datos el byte de inicio y los dos bytes de CRC, la longitud total del bloque de datos puede variar entre 4 y 515 bytes. Como por defecto en el protocolo de acceso SPI no se consideran los bytes de CRC, estos pueden tomar cualquier valor.

4.2.4.- Inicialización del bus SPI

A nivel del micro, lo primero que se debe hacer es inicializar el BUS SPI. Para ello hay que llamar a la función “Spi_Init_Advanced” con los parámetros siguientes:

- SPI Máster
- Modo a 8 bits
- Temporizador secundario = 1
- Temporizador primario = 64
- Slave Select deshabilitado
- “data sampled in the middle of data output time”
- En espera, la señal “clock” se mantiene a nivel alto
- La salida de datos serie cambia en la transición del estado de reloj active al estado en reposo.

Ejemplo de la función:

```
Spi_Init_Advanced (
        _SPI_MASTER,
        _SPI_8_BIT,
        _SPI_PRESCALE_SEC_1,
        _SPI_PRESCALE_PRI_64,
        _SPI_SS_DISABLE,
        _SPI_DATA_SAMPLE_MIDDLE,
        _SPI_CLK_IDLE_HIGH,
        _SPI_ACTIVE_2_IDLE);
```

Nota: Una vez inicializada la tarjeta, se puede reiniciar el SPI a mayor velocidad, reduciendo el temporizador primario.

4.2.5.- Reset de la tarjeta para arrancar en MODO SPI

Por defecto, al arrancar la tarjeta, ésta se encuentra en modo MultiMediaCard. Para que entre en modo SPI, hay que enviarle el comando 0 mientras se mantiene activa la señal ¡CS (¡CS=0) (lógica negativa) pero antes de todo, para poder iniciar la comunicación por el bus hay que enviar como mínimo 74 ciclos de reloj a la tarjeta. Así, para hacer el reset de la tarjeta y prepararla para trabajar en modo SPI, hay que seguir la siguiente secuencia:

- Dar como mínimo 74 ciclos de reloj, es decir enviar unos 10 bytes a través de la SPI.
- Activar la señal ¡CS (¡CS=0).
- Enviar el comando 0 con el CRC bien calculado, ya que todavía no está en modo SPI, por lo que sí se considera el CRC. De hecho la secuencia del comando 0 siempre es la misma: 0x40,0x00,0x00,0x00,0x00,0x95
- Esperar el byte de respuesta que ha de ser 00000001 (tarjeta en modo idle).

4.2.6.- Activar la inicialización de la tarjeta MODO SPI

Una vez reseteada y en modo SPI, hay que hacer la inicialización de la tarjeta, para ello hay que enviar el comando 1. La secuencia general es esta:

- Activar el pin ¡CS (¡CS=0).
- Enviar el comando 1: 0x41, 0x00, 0x00, 0x00, 0x00, 0xXX. Como la tarjeta ya está en modo SPI el CRC puede tomar cualquier valor.
- Esperar el byte de respuesta que ha de ser 00000000 (tarjeta lista).

La siguiente función de MicroC: Mmc_Init(PORTB,8) hace las funciones de resetear e inicializar el MODO SPI:

Prototipo	unsigned Mmc_Init(unsigned *port, unsigned CSpin);
Devuelve	0 – si la tarjeta MMC/SD se detectó e inicializó satisfactoriamente. 1 – si hubo un error.
Parámetros de entrada	Port - registro del Puerto utilizado para CS (chip select port). CSpin - valor del pin asociado a la función CS (chip select pin). Solo le damos el pin asociado a la función CS, ya que las del SPI ya vienen por defecto y no se pueden modificar.

4.3.- Comandos para leer y escribir en sectores de la SD

4.3.1.-Escritura de un bloque en la tarjeta MODO SPI

Una vez inicializada la tarjeta, para escribir un bloque en ésta, hay que enviar el comando 24 con la dirección de inicio a partir de la cual se desean guardar los datos.
Si todo va bien la tarjeta enviará TRES respuestas R1 repetidas informando al controlador que ya puede enviar el bloque de datos, que ha de tener una longitud de 512 bytes (en la escritura solo se permiten 512 bytes) más el byte de inicio de bloque de datos y los dos bytes de CRC. La secuencia a seguir es:

- Activar el PIN ¡CS (¡CS=0).
- Enviar el comando COM24: 0x58, 0xXX,0xXX,0xXX,0xXX,0xYY. Los 4 bytes XX corresponden a la dirección a partir de la cual se quieren guardar los datos. 0xYY corresponde al byte de CRC y como la tarjeta esta en modo SPI pueden tomar cualquier valor ya que no se consideran.
- Si todo va bien la tarjeta responde con el byte de respuesta R1 tres veces consecutivas.
- Enviar a la tarjeta el bloque de datos que consiste en:
- 1 byte de inicio de bloque de datos 0xFE
- 512 bytes con los datos a guardar.
- 2 bytes de CRC
- Mientras la tarjeta esta ocupada guardando el valor, irá enviando bytes indicando que está ocupada, y cuando finalice la escritura enviará un byte de confirmación.

La siguiente función de MicroC: Mmc_Write_Sector (sector, databuffer),
escribirá en el sector indicado “sector” la información almacenada en “databuffer”, un vector de 512 bytes.

Prototipo	unsigned Mmc_Write_Sector(unsigned long sector, char *dbuff);
Devuelve	0 – si la escritura se realizó con éxito. 1 – si hubo un error en el envio del comando. 2 – si hubo un error al escribir (dato rechazado).
Parámetros de entrada	Sector: dirección del sector de la MMC/SD a escribir. dbuff: data a escribir (buffer de 512 bytes).
Ejemplo	unsigned int error; unsigned long sectorNo = 510; (- numero del sector char dataBuffer[512]; ... error = Mmc_Write_Sector(sectorNo, dataBuffer);

4.3.2.-Lectura de un bloque en la tarjeta MODO SPI

Para leer un bloque de la tarjeta hay que enviar a esta el comando 17 con la dirección de inicio de lectura en los bytes de argumento. La dirección puede tomar cualquier valor comprendido dentro del rango de direcciones válidas de la tarjeta pero todo el bloque leído debe estar dentro de un mismo sector físico. A continuación la tarjeta envía un byte de respuesta R1 seguido del bloque de datos, que comienza por 0xFE, continua con los bytes de datos y finaliza con los 2 bytes de CRC que no se usan. El número de bytes de datos depende del tamaño de bloque que se haya programado mediante el comando 16, y en la lectura puede ir de 1 a 512. La secuencia a seguir es la siguiente:

- Activar el PIN ¡CS (¡CS=0).
- Enviar el comando 17 0x51, 0xXX, 0xXX, 0xXX, 0xXX, 0xYY. Los 4 bytes XX corresponden a la dirección a partir de la cual se quieren leer los datos. 0xYY corresponde al byte de CRC y como la tarjeta esta en modo SPI puede tomar cualquier valor ya que no se considera.
- Si todo va bien, la tarjeta responde con un byte de respuesta R1, seguido del bloque de datos con la información solicitada y que el controlador tendrá que ir capturando. Esta tiene la misma estructura que los bloques de datos utilizados en la escritura:
    - 1 byte de inicio de bloque de datos 0xFE
    - n bytes con los datos a guardar.
    - 2 bytes de CRC.
    - Si se produce un error durante la comunicación, la tarjeta no transmitirá ningún dato y en lugar de estos enviará un byte de respuesta indicador de error.

El comando *stop transmisión* (CMD12), detiene la operación de transferencia de datos.

La siguiente función de MicroC: Mmc_Read_Sector(sector, databuffer)
leerá el sector indicado (“sector”) y guardara la información almacenada en “databuffer”, un vector de 512 bytes.

Prototipo	unsigned Mmc_Read_Sector(unsigned long sector, char * databuffer);
Devuelve	0 – si la lectura se hizo correctamente. 1 – si ocurrió algún error.
Parámetros	Sector: sector de la MMC/SD a leer. databuffer: buffer de 512 bytes(como mínimo) para almacenar los datos.

Ejemplo	// read sector 510 of the MMC/SD card
	unsigned int error;
	unsigned long sectorNo = 510;
	char dataBuffer[512];
	...
	error = Mmc_Read_Sector(sectorNo, dataBuffer);

4.4. Comandos para trabajar con el formato FAT16

4.4.1. Iniciar tarjeta de memoria en fat16

Una vez la memoria iniciada en SPI correctamente, además de escribir y leer en sectores “sueltos”, se pueden manejar ficheros.

Primeramente se tiene que llamar a la función Mmc_Fat_Init que hará lo siguiente:
Iniciar la memoria MMC/SD, leer el sector de arranque del FAT16 y extraer la información necesaria.

Nota: la tarjeta MMC/SD debe estar formateada en FAT16.

Antes de usar esta función se debe de ejecutar el “Mmc_ Init “.

Prototipo	unsigned Mmc_Fat_Init(unsigned *port, unsigned pin)
Devuelve	0 – si la MMC/SD se detecto e inicializo correctamente. 1 – si no se encontró el sector de arranque del FAT16 255 - si la MMC/SD no se pudo detectar
Parámetros	Port - registro del Puerto utilizado para CS (chip select port) Cspin - valor del pin asociado a la función CS (chip select pin)

Ejemplo	// Initialize the SPI module
	// Initialize MMC/SD card and MMC_FAT16 library globals
	Mmc_Fat_Init(&PORTB, 8);
	// Reinitialize the SPI module at higher speed (change primary prescaler).

4.4.2.-Formatear tarjeta de memoria

Después de inicializar la memoria, se puede formatearla llamando a la función: Mmc_Fat_QuickFormat.

Si la MMC ya esta formateada correctamente en FAT16 no será necesario este paso.

Prototipo	unsigned Mmc_Fat_QuickFormat(unsigned *port, unsigned pin, char * mmc_fat_label)
Devuelve	**0 –** si la MMC/SD se detecto, formateo e inicializo correctamente. **1 –** si el formato FAT16 resulto INsatisfactorio **255 –** si la MMC/SD no se pudo detectar
Parámetros	**Port** - registro del Puerto utilizado para CS (chip select port) **Cspin** - valor del pin asociado a la función CS (chip select pin) **Mmc_fat_etiqt:** volume label (11 caracteres). Si se usan menos de 11 caracteres, rellenar con espacios.

Ejemplo	// Initialize the SPI module
	// Format and initialize MMC/SD card and MMC_FAT16 library globals
	Mmc_Fat_QuickFormat(&PORTB, 8, "MMC512MB");
	// Reinitialize the SPI module at higher speed (change primary prescaler).

4.4.3.-Asignar un nuevo fichero

Seguidamente, una vez la tarjeta esta formateada se puede, por ejemplo, crear un fichero, para ello se llama a la siguiente función. Después de llamar a esta función, todas las siguientes operaciones (lectura, escritura, reset, y borrado) se aplicaran sobre el fichero inicializado.

Antes de usar esta función se debe de ejecutar el “Mmc_Fat_Init “.

PROTOTIPO	unsigned Mmc_Fat_Assign(char *filename, char file_cre_attr);
DEVUELVE	1 – si el nuevo fichero se ha creado 0 – si el fichero No existe pero no se ha creado
PARAMETROS	filename: nombre del archivo sobre el que se realizaran las posteriores operaciones. El nombre debe tener el formato DOS 8.3 (nombre.extension) . file_cre_attr: atributos del fichero. cada bit corresponde a un atributo: Bit Mask Descripción 0 0x01 Solo Lectura 1 0x02 Oculto 2 0x04 Del Sistema 3 0x08 Nombre del volumen 4 0x10 Subdirectorio 5 0x20 Bit de Archivo 6 0x40 reservado 7 0x80 reservado

EJEMPLO	// create file with archive attribut if it does not already exist
	//Mmc_Fat_Assign(“archivo1 .txt” , atributos)
	
	Mmc_Fat_Assign("MIKRO007.TXT",0xA0);

4.4.4.-Reset FAT16

La siguiente función inicia el uso del fichero en modo lectura. La función devuelve a la variable “tamaño” el tamaño del fichero e inicia el puntero de dirección al primer byte de este.
Antes de usar esta función se debe de ejecutar el “Mmc_Fat_Assign “.

PROTOTIPO	void Mmc_Fat_Reset(unsigned long *size);
PARAMETROS	Tamaño: variable por referencia donde se guarda el tamaño de del archivo una vez abierto.
EJEMPLO	unsigned long size; ... Mmc_Fat_Reset(size);

4.4.5.-Leer texto en fichero FAT16

Lee un byte del archivo abierto en modo lectura. Tras la función , el puntero se configurará para entregar, en la siguiente lectura, el siguiente carácter del archivo.
Antes de usar esta función se debe de ejecutar el “Mmc_Fat_Reset “.

PROTOTIPO	void Mmc_Fat_Read(unsigned char *bufferdata);
PARAMETROS	• Bufferdata: parámetro pasado por referencia, donde se devuelve el byte leído.
EJEMPLO	char character; ... Mmc_Fat_Read(&character);

4.4.6.- Reescribir texto en fichero FAT16

Abre el archivo asignado actualmente en modo de escritura. Si el archivo no está vacío, su contenido será borrado.
Antes de usar esta función se debe de ejecutar el “Mmc_Fat_Assign “.

PROTOTIPO	void Mmc_Fat_Rewrite();

EJEMPLO	// open file for writing
	Mmc_Fat_Rewrite();

4.4.7.- Añadir texto en fichero FAT16

Abre el archivo asignado actualmente en modo “añadir carácter”. Tras esta función, el puntero se coloca después del último byte del archivo, por lo que cualquier operación de escritura posterior archivo empezará desde allí.
Antes de usar esta función se debe de ejecutar el “Mmc_Fat_Assign “.

PROTOTIPO	void Mmc_Fat_Append();

EJEMPLO	// open file for appending
	Mmc_Fat_Append();

4.4.8.- Borrar fichero FAT16

Esta función elimina el fichero abierto en el momento, de la memoria de la SD/MMC
Antes de usar esta función se debe de ejecutar el “Mmc_Fat_Assign “.

PROTOTIPO	void Mmc_Fat_Delete();

EJEMPLO	// delete current file
	Mmc_Fat_Delete();

4.4.9.- Escribir texto en fichero FAT16

Esta función escribe en el fichero n caracteres, donde n es el número de bytes a escribir, enviado a la función a través de la variable “data_lenght”, y “fdata” el vector de caracteres a escribir
Antes de usar esta función debe de estar en modo escritura, para ello ejecutar el “Mmc_Fat_Append “o bien “Mmc_Fat_Rewrite “.

PROTOTIPO	void Mmc_Fat_Write(char *fdata, unsigned data_len);
PARAMETROS	fdata: data a escribir data_len: numero de bytes a escribir..
EJEMPLO	char file_contents[42]; ... Mmc_Fat_Write(file_contents, 42); // write data to the assigned file

4.4.10.- Fecha en fichero FAT16

Las siguientes funciones, escriben y leen, el campo de fecha de modificación del fichero en uso.
Antes de usar esta función se debe de ejecutar el “Mmc_Fat_Assign “

Mmc_Fat_Set_File_Date

EJEMPLO	Mmc_Fat_Set_File_Date(2008,3,27,17,41,0);

Mmc_Fat_Get_File_Date

EJEMPLO	unsigned year; char month, day, hours, mins; ... Mmc_Fat_Get_File_Date(&year, &month, &day, &hours, &mins);

4.4.11.- Tamaño del fichero FAT16

La siguiente función: Mmc_Fat_Get_File_Size, devuelve el tamaño del archivo en uso en bytes.
Antes de usar esta función se debe de ejecutar el “Mmc_Fat_Assign “

PROTOTIPO	unsigned long Mmc_Fat_Get_File_Size();
EJEMPLO	unsigned long my_file_size; ... my_file_size = Mmc_Fat_Get_File_Size();

4.5.- Lista de comandos SPI MMC SD

Estos son algunos de los principales comandos:

Comando:	**Argumentos:**	**Respuesta:**	**Descripción:**
CMD0	No	R1	Resetea la tarjeta
CMD1	No	R1	Inicializa la tarjeta
CMD9	No	R1	Pide a la tarjeta su información CSD
CMD10	No	R1	Pide a la tarjeta su identificación CID
CMD13	No	R2	Consulta el estado de la tarjeta
CMD16	[31..0] Longitud del bloque.	R1	Establece la longitud (en bytes) del bloque para los datos en las operaciones de lectura y escritura.
CMD17	[31..0] Dirección de datos.	R1	Lee un bloque del tamaño indicado por el comando 16.
CMD24	[31..0] Dirección de datos	R1 R1 R1	Escribe un bloque del tamaño indicado por el comando 16.

Fig.16- Lista de comandos principales.

CMD3 SEND_RELATIVE-ADDR
CMD5 IO_SEND_OP_COND
CMD6 SWITCH_FUNC Añadido en la parte 1 v 1.10
CMD7 SELECT/DESELECT_CARD
CMD12 STOP_TRANSMISSION
CMD15 GO_INACTIVE_STATE
CMD18 READ_MULTIPLE_BLOCK Leer varios bloques.
CMD25 WRITE_MULTIPLE_BLOCK Escribir varios bloques.
CMD27 PROGRAM_CSD
CMD28 SET_WRITE_PROT
CMD29 CLR_WRITE_PROT
CMD30 SEND_WRITE_PROT
CMD32 ERASE_WR_BLK_START
CMD33 ERASE_WR_BLK_END
CMD38 ERASE
CMD42 LOCK_UNLOCK
CMD52 IO_RW_DIRECT
CMD53 IO_RW_EXTENDED El modo bloqueo es opcional
CMD55 APP_CMD
CMD56 GEN_CMD
CMD58 READ_OCR
CMD59 CREC_ON_OFF
ACMD6 SET_BUS_WIDTH
ACMD13 SD_STATUS
ACMD22 SEND_NUM_WR_BLOCKS
ACMD23 SET_WR_BLK_ERASE_COUNT
ACMD41 SD_APP_OP_COND
ACMD42 SET_CLR_CARD_DETECT
ACMD51 SEND_SCR SCR incluye sólo información SDMEM

Fig.17- Lista de comandos principales.

4.6.- Anexo: Ejemplo de código completo

```
#include <spi_const.h>

const char SWAP_FILE_MSG[] = "Swap file at: ";

char
 fat_txt[20] = "FAT16 not found",
 file_contents[50] = "XX MMC/SD FAT16 library by Anton Rieckertn";

char
 filename[14] = "MIKRO00xTXT";          // File names
unsigned
 loop, loop2;
unsigned short
 caracter;
unsigned long
 i, size;
char Buffer[512];

//I-I-I--------- Writes string to USART
void I_Write_Str(char *ostr) {
  unsigned i;

  i = 0;
  while (ostr[i]) {
    Uart1_Write_Char(ostr[i++]);
  }
}//~

//M-M-M--------- Creates a new file and writes some data to it
void M_Create_New_File() {
  filename[7] = 'A';
  Mmc_Fat_Assign(&filename, 0xA0);        // Will not find file and then create file
  Mmc_Fat_Rewrite();                  // To clear file and start with new data
  for(loop = 1; loop <= 99; loop++) {   //  We want 5 files on the MMC card
    Uart1_Write_Char('.');
    file_contents[0] = loop / 10 + 48;
    file_contents[1] = loop % 10 + 48;
    Mmc_Fat_Write(file_contents, 42);   // write data to the assigned file
  }
}//~

//M-M-M--------- Creates many new files and writes data to them
void M_Create_Multiple_Files() {
  for(loop2 = 'B'; loop2 <= 'Z'; loop2++) {
    Uart1_Write_Char(loop2);            // signal the progress
    filename[7] = loop2;                // set filename
    Mmc_Fat_Assign(&filename, 0xA0);        // find existing file or create a new one
    Mmc_Fat_Rewrite();                  // To clear file and start with new data
    for(loop = 1; loop <= 44; loop++) {
      file_contents[0] = loop / 10 + 48;
      file_contents[1] = loop % 10 + 48;
      Mmc_Fat_Write(file_contents, 42);  // write data to the assigned file
    }
  }
}//~

//M-M-M--------- Opens an existing file and rewrites it
void M_Open_File_Rewrite() {
```

```
  filename[7] = 'C';
  Mmc_Fat_Assign(&filename, 0);
  Mmc_Fat_Rewrite();
  for(loop = 1; loop <= 55; loop++) {
    file_contents[0] = loop / 10 + 64;
    file_contents[1] = loop % 10 + 64;
    Mmc_Fat_Write(file_contents, 42);    // write data to the assigned file
  }
}//~

//M-M-M--------- Opens an existing file and appends data to it
//              (and alters the date/time stamp)
void M_Open_File_Append() {
     filename[7] = 'B';
     Mmc_Fat_Assign(&filename, 0);
     Mmc_Fat_Set_File_Date(2005,6,21,10,35,0);
     Mmc_Fat_Append();                          // Prepare file for append
     Mmc_Fat_Write(" for mikroElektronika 2005n", 27);   // Write data to the assigned file
}//~

//M-M-M--------- Opens an existing file, reads data from it and puts it to USART
void M_Open_File_Read() {
  filename[7] = 'B';
  Mmc_Fat_Assign(&filename, 0);
  Mmc_Fat_Reset(&size);               // To read file, procedure returns size of file
  for (i = 1; i <= size; i++) {
    Mmc_Fat_Read(&caracter);
    Uart1_Write_Char(caracter);        // Write data to USART
  }
}//~

//M-M-M--------- Deletes a file. If the file doesn't exist, it will first be created
//              and then deleted.
void M_Delete_File() {
  filename[7] = 'F';
  Mmc_Fat_Assign(filename, 0);
  Mmc_Fat_Delete();
}//~

//M-M-M--------- Tests whether file exists, and if so sends its creation date
//              and file size via USART
void M_Test_File_Exist(char fLetter) {
  unsigned long fsize;
  unsigned int year;
  unsigned short month, day, hour, minute;
  unsigned char outstr[12];

  filename[7] = fLetter;
  if (Mmc_Fat_Assign(filename, 0)) {
    //--- file has been found - get its date
    Mmc_Fat_Get_File_Date(&year, &month, &day, &hour, &minute);
    WordToStr(year, outstr);
    I_Write_Str(outstr);
    ByteToStr(month, outstr);
    I_Write_Str(outstr);
    WordToStr(day, outstr);
    I_Write_Str(outstr);
    WordToStr(hour, outstr);
    I_Write_Str(outstr);
    WordToStr(minute, outstr);
```

```
    I_Write_Str(outstr);
    //--- get file size
    fsize = Mmc_Fat_Get_File_Size();
    LongToStr((signed long)fsize, outstr);
    I_Write_Str(outstr);
  }
  else {
    //--- file was not found - signal it
    Uart1_Write_Char(0x55);
    Delay_ms(1000);
    Uart1_Write_Char(0x55);
  }
}//~

//--------------- Tries to create a swap file, whose size will be at least 100
//              sectors (see Help for details)
void M_Create_Swap_File() {
  unsigned int i;

  for(i=0; i<512; i++)
    Buffer[i] = i;

  size = Mmc_Fat_Get_Swap_File(5000, "mikroE.txt", 0x20);   // see help on this function for
details

  if (size) {
    LongToStr((signed long)size, fat_txt);
    I_Write_Str(fat_txt);

    for(i=0; i<5000; i++) {
      Mmc_Write_Sector(size++, Buffer);
      Uart1_Write_Char('.');
    }
  }
}//~

//--------------- Main. Uncomment the function(s) to test the desired operation(s)
void main() {
    //--- prepare PORTD for signalling
    PORTD = 0;
    TRISD = 0;
    ADPCFG = 0xFFFF;
    //--- set up USART for the file read
    Spi1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1,
_SPI_PRESCALE_PRI_64,
                       _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_HIGH,
_SPI_ACTIVE_2_IDLE);

    Uart_Init(19200);
    U1MODEbits.ALTIO = 1;              // clear the way for SPI
    Delay_ms(200);                  // wait for the UART module to stabilize
    //--- init the FAT library
    if (!Mmc_Fat_Init(&PORTB,8)) {
       // reinitialize spi at higher speed
       Spi1_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_1,
_SPI_PRESCALE_PRI_4,
                          _SPI_SS_DISABLE, _SPI_DATA_SAMPLE_MIDDLE,
_SPI_CLK_IDLE_HIGH, _SPI_ACTIVE_2_IDLE);
       //--- Test start
       PORTD = 0x0005;
```

```
        //--- Test routines. Uncomment them one-by-one to test certain features
        M_Create_New_File();
        M_Create_Multiple_Files();

        M_Open_File_Rewrite();
        M_Open_File_Append();
        M_Delete_File();
        M_Create_Swap_File();

        M_Open_File_Read();
        M_Test_File_Exist('F');        // this file will not exist here
        M_Test_File_Exist('B');        // this file will exist here

    }
     else {
       l_Write_Str(fat_txt);
     }
    //--- Test termination
    PORTD = 0xFFFF;
}//~!
```

5.- Otras memorias

5.1.- Conectores Compact Flash

Esta tecnología fue creada por SanDisk. Hay dos modelos, uno normal y el denominada Fast, por su mayor velocidad de transferencia, que alcanza hasta 16MB/s y capacidad de hasta 137 GB.

Sin embargo, el ultimo estándar CF 3.0, que soporta tasas de hasta 66MB/s.

Su uso se limita actualmente a las cámaras DSLR profesionales.

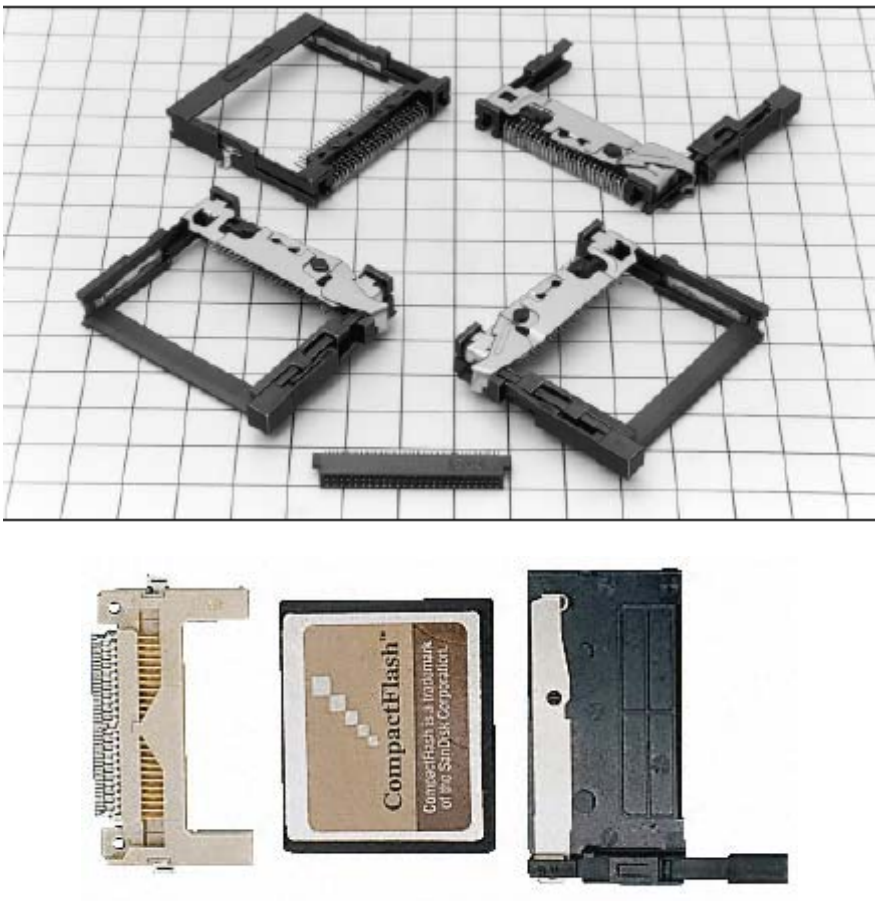


Fig.18- Conectores CF

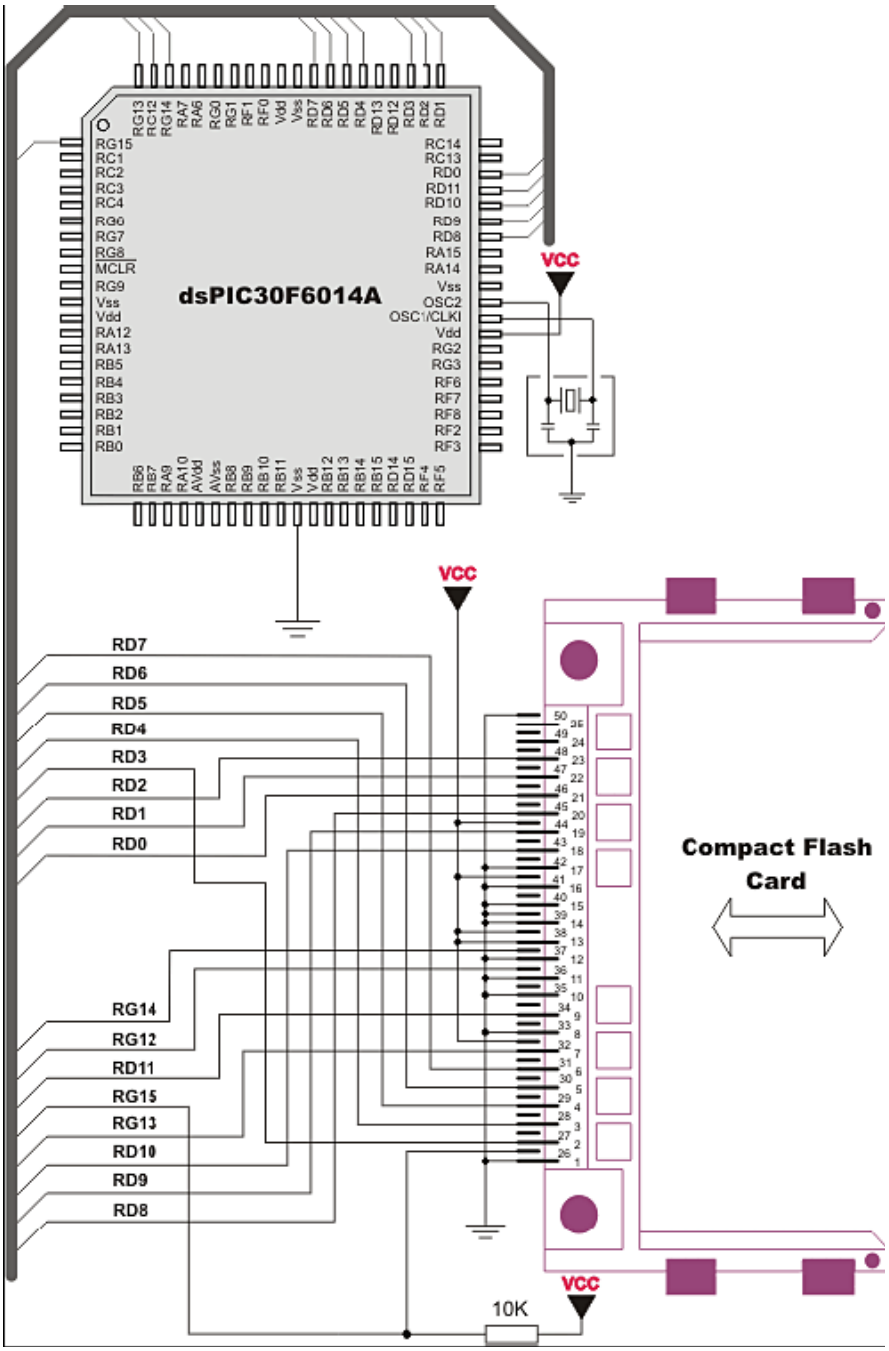


Fig.19- Esquema de conexión de tarjetas Compact Flash

5.2.-Conectores Memory Stick (Sony)

Se trata de un formato propietario de Sony, habitualmente más caro que los competentes, y empleados únicamente en dispositivos de la compañía japonesa, como teléfonos móviles Sony-Ericsson, consolas portátiles PSP y cámaras digitales de Sony.

Su capacidad alcanza los 32 GB en su version Pro, que es la última evolución de esta tecnología de tarjetas de almacenamiento no volatil.

Sus tasas de transferencia quedan en unos escasos 10 MB/s en la mayoría de los casos.

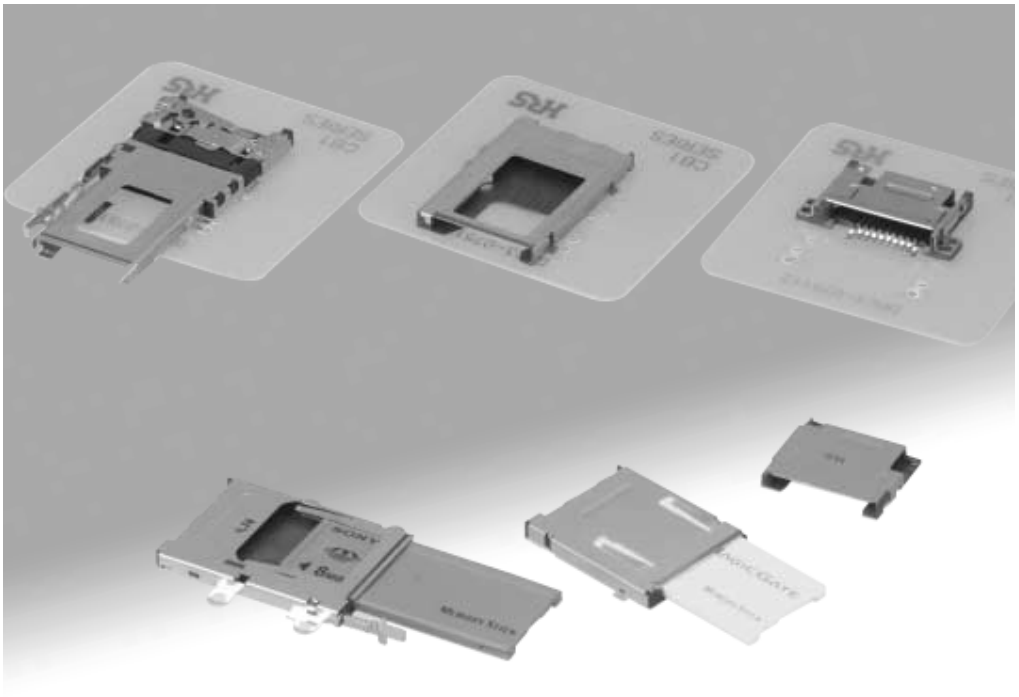
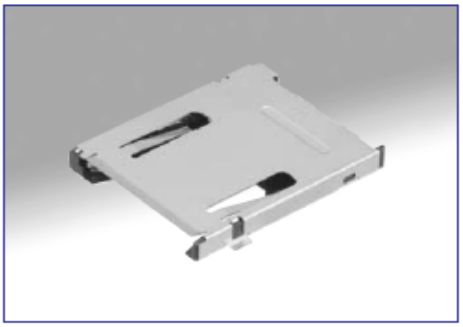
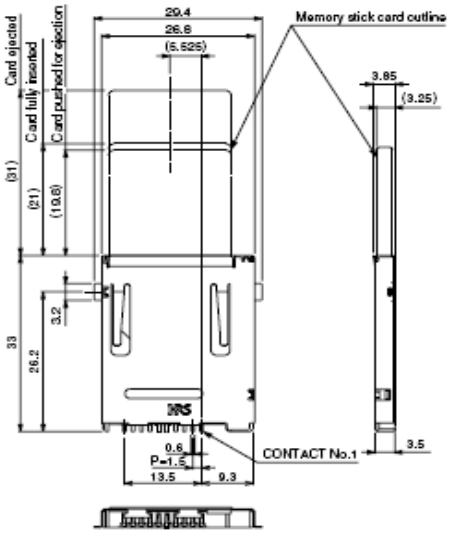


Fig.20- Tipos de conectores memory stick.

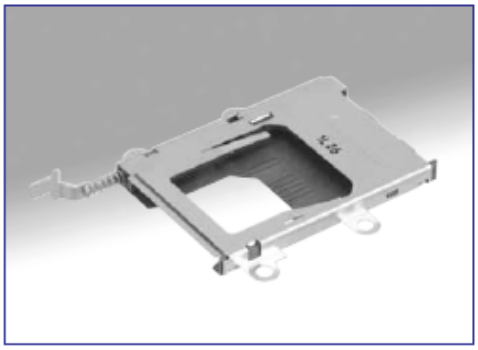
5.3.- Low profile, push insert push eject



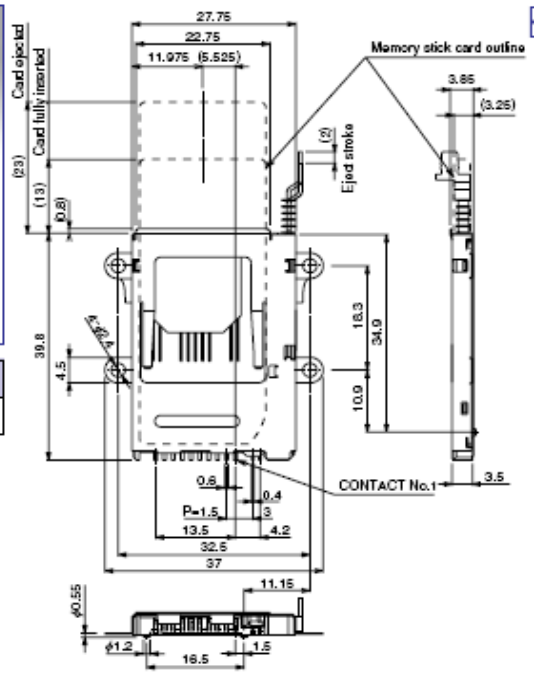
Part No.	CL No.
CB1G-10S-1.5H-PEJC2	CL689-0037-5



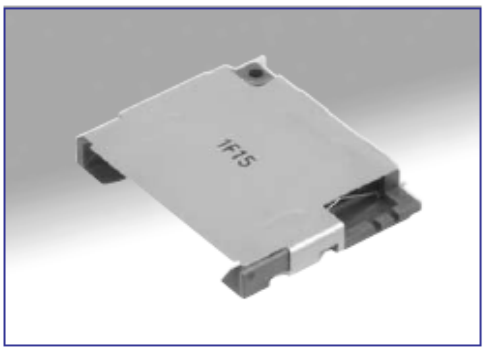
5.4.- Button touch eject

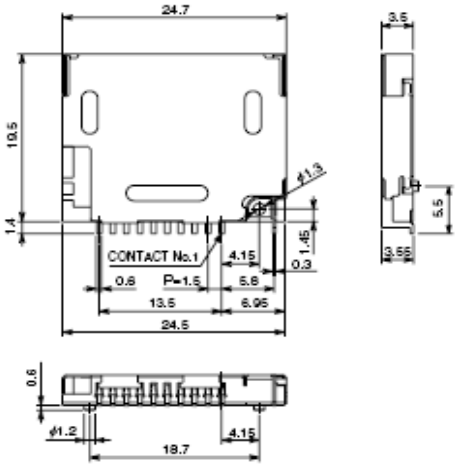


Part No.	CL No.
CB1F-10S-1.5H-TEJL-PA	CL689-0028-4



5.5.- Without card ejection

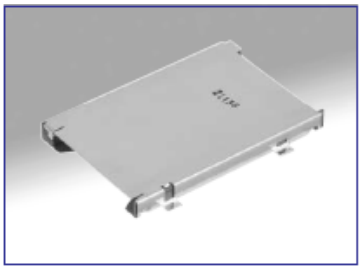




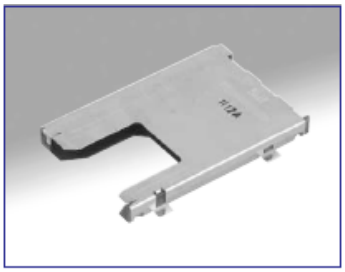
Part No.	CL No.
CB1D-10S-1.5H	CL689-0021-5

5.6.- Push insert-push eject

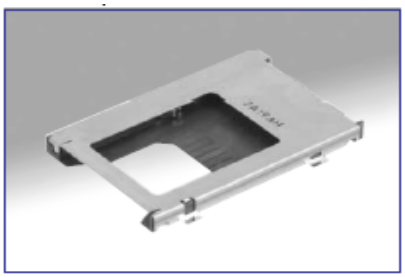
5.6.1. Normal type 5.6.2. With “U” cut-out. 5.6.3. With square window.



Part No.	CL No.
CB1EB-10S-1.5H-PEJC2	CL689-0026-9



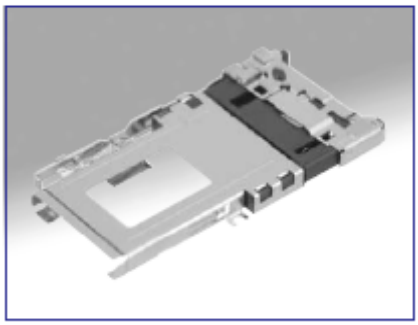
Part No.	CL No.
CB1EBG-10S-1.5H-PEJC2	CL689-0034-7



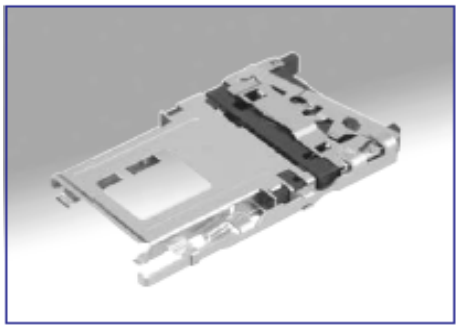
Part No.	CL No.
CB1EBH-10S-1.5H-PEJC2	CL689-0035-0

5.7. Left and right ejection

5.7.1.LEFT EJECTION 5.7.2. RIGHT EJECTION.



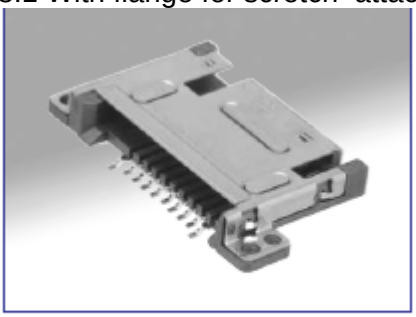
Part No.	CL No.
CB1C-10S-1.5H-EJL(56)	CL689-0006-1-56



Part No.	CL No.
CB1C-10S-1.5H-EJR(59)	CL689-0007-4-59

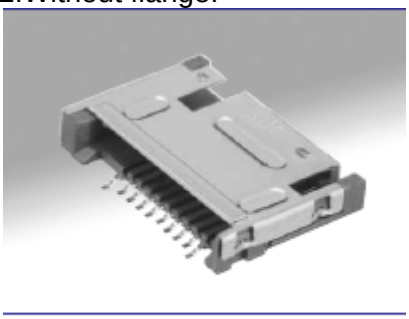
5.8. With flange for scretch attachment and without flange

5.8.1 With flange for scretch attachment



Part No.	CL No.
CB1A-10S-1.5H(57)	CL689-0001-8-57

5.8.2.Without flange.



Part No.	CL No.
CB1AA-10S-1.5H(57)	CL689-0002-0-57

6.- BIBLIOGRAFIA

http://es.wikipedia.org/wiki/Tarjeta_de_memoria
http://es.wikipedia.org/wiki/Secure_Digital
http://es.wikipedia.org/wiki/MicroSD
http://es.wikipedia.org/wiki/MiniSD
http://es.wikipedia.org/wiki/Memoria_flash

http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
http://en.wikipedia.org/wiki/I2C
http://en.wikipedia.org/wiki/Secure_Digital_card
http://en.wikipedia.org/wiki/Full_duplex
http://en.wikipedia.org/wiki/System_Management_Bus
http://en.wikipedia.org/wiki/MultiMediaCard
http://en.wikipedia.org/wiki/Secure_Digital_card

http://elm-chan.org/docs/mmc/mmc_e.html
http://homepages.mty.itesm.mx/al778081/

http://www.forosdeelectronica.com/about13868-0-asc-10.html
http://www.tolaemon.com/otros/mmcard.htm
Manual de mikroC compilator dsPIC 30/33. www.mikroe.com

www.hirose-connectors.com