MANEJO DE UNA MEMORIA SD/MMC CON UN PIC16F87x.

How to use a SD/MMC memory with PIC16F87x.

RESUMEN
En este artículo se muestra cómo manejar una memoria SD/MMC con un microcontrolador PIC, para ello se presenta a nivel hardware el proceso de lectura y escritura, además de la respectiva simulación y programación en el software Proteus y CCS respectivamente.

PALABRAS CLAVES: SD/MMC, SPI, Debug,Token, CCS, Proteus

CARLOS ALBERTO HENAO
Tecnólogo Eléctrico
Estudiante Ingeniería Eléctrica
Universidad Tecnológica de Pereira
caramelo@utp.edu.co

EDISON DUQUE CARDONA
Profesor Universidad Tecnológica de Pereira

ABSTRACT
This article shows how to handle an SD / MMC with a PIC microcontroller, for this is presented at the hardware level the process of reading and writing, as well as the corresponding simulation and programming in the Proteus software and CCS respectively.

KEYWORDS: SD/MMC, SPI, Debug, Token, CCS, Proteus.

1. INTRODUCCIÓN

En muchas aplicaciones de sistemas electrónicos, y en general en cualquier sistema de instrumentación, es necesario almacenar grandes cantidades de datos en donde las memorias seriales EEPROM suelen ser insuficientes, por lo tanto, el uso de las memorias tipo flash (SD/MMC) nos brinda una gran ventaja, otorgándonos gran capacidad de almacenamiento y una gran disponibilidad en mercado a muy bajo costo.

Estás memorias poseen dos protocolos de comunicación, la estándar denominada BUS SD que utiliza 4 líneas paralelas para la comunicación y el tipo serial SPI (*Serial Peripheral Interface*). Este último protocolo es fácilmente encontrado en los microcontroladores de la familia 16f87xx, por tal razón, se decidió trabajar con dicho protocolo para realizar la comunicación entre ambos dispositivos.

2. ALGUNAS CARACTERÍSTICAS DE LAS MEMORIAS SD/MMC

Las tarjetas SD poseen 9 pines, de los cuales uno es de reloj (CLK), otro es para los comandos, cuatro son de datos y los tres restantes son de alimentación como se muestra en la **tabla 1**, para la MMC son 7 pines la única diferencia respecto a la SD es que posee sólo dos pines para datos. El rango de voltaje de alimentación permitido es de 2.7 a 3.6 V [7].

Internamente, la tarjeta posee chips de memoria flash como medio de almacenamiento. Además posee un controlador inteligente que maneja los diferentes protocolos de comunicación, algoritmos de seguridad para la protección contra copia no autorizada de información almacenada, algoritmos de corrección de errores de código, diagnósticos y control de potencia [7].

PIN No	Nombre	Descripción
1	CS	Selección del chip
2	DI	Entrada de datos
3	Vss	Tierra
4	Vcc	Fuente de alimentación
5	CLK	Reloj (SPI)
6	Vss	Tierra
7	Do	Salida de datos
8	D1	Salida de datos
9	D2	Salida de datos

Tabla 1 Pines de la memoria SD/MMC

2.1 MODO DE COMUNICACIÓN BUS SD

Este modo permite que se utilicen los terminales de datos (D0 - D3) en forma bidireccional, lo cual da mayor ancho de banda durante las transmisiones. Por otro lado, los comandos se transmiten por la línea CMD en forma serial, y por último la respuesta de la tarjeta al comando se transmite por la línea CMD.

PIN_No	Nombre	Descripción
1	CD/DAT3	Detención de la tarjeta
2	CMD	Comando/Respuesta
3	VSS	GND
4	VDD	Alimentación
5	CLK	Reloj
6	VSS	GND
7	DAT0	Línea de datos
8	DAT1	Línea de datos
9	DAT2	Línea de datos

Tabla 2. Pines en el modo BUS SD

Fecha de Recepción: Enero 26 de 2010
Fecha de Aceptación: Marzo 25 de 2010

Al inicio de la transferencia solo se envían datos por D0, ya después se puede ampliar el ancho de banda de los datos hasta D3. Como lo muestra la tabla 2 [7].

2.2 MODO DE COMUNICACIÓN SPI

Para la comunicación con la tarjeta utilizando este modo, se necesitan sólo cuatro líneas de comunicación, **DATA IN**, **DATA OUT**, **CS** y **CLK**. La memoria recibe los datos y los comandos por DATA IN y envía datos por DATA OUT [4]. Para habilitar la tarjeta se debe poner en nivel bajo el Terminal CS. La señal de CLK que se envía desde el PIC16F87x es la que establece la velocidad de la comunicación.

Como los datos se envían solo por un terminal se tiene menor utilización del potencial de la memoria (en el modo SD los datos se envían por cuatro terminales). La aplicación de cada uno de estos pines se muestra en la tabla 3 [4].

PIN_No	Nombre	Descripción
1	CS	Activación de tarjeta
2	DATA IN	Comandos de datos desde el host
3	VSS	GND
4	VDD	Alimentación
5	CLK	Reloj
6	VSS	GND
7	DATAOUT	Datos hacia el Host
8	RSV	Reservado
9	RSV	Reservado

Tabla 3. Pines en el modo SPI

2.3 FORMATO DE COMANDOS EN MODO SPI

La secuencia de comandos para la tarjeta en modo SPI consiste de 6 Bytes tal como se ilustra en la tabla 4 [4].

Byte 1			Byte 2 - 5	Byte 6	
7	6	5 0	31 0	7	0
0	1	Comand	Comand	CRC	1

Tabla 4. Secuencia de comandos SPI

1. El primer byte es el comando (escritura, lectura etc.).

2. Del segundo byte al quinto son datos adicionales, por ejemplo dirección.

3. El sexto byte es un byte de verificación.

2.4 COMANDOS POR COMUNICACIÓN SPI

La memoria contiene varios comandos que funcionan por comunicación SPI, por ejemplo.

1. Reset de la memoria comando 0 (CMD0)
2. Inicialización de la memoria comando 1 (CMD1)
3. Configuración del bloque de 512 bytes comando 16 (CMD16)
4. Escritura de un bloque de 512 bytes (CMD24)
5. Lectura de un bloque de 512 bytes comando 17 (CMD17)

Una vez enviado el respectivo comando, la memoria responde por medio de un registro llamado R1 (Figura 1) indicando si hubo un error o si todo está bien [4].

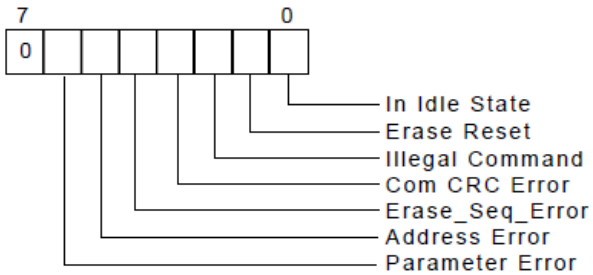

Figura 1. Registro de errores enviado por la memoria

Según la figura 1 la memoria debe responder con 0x00 para todos los comando antes mencionados, pero para el comando 0 (CMD0) la memoria responde con 0x01 debido a su estado inicial.

3. SIMULACIÓN EN PROTEUS

Todas la pruebas se hicieron en el software Proteus 7.2 ya que esta edición tiene entre sus librerías la memoria SD/MMC, además cuenta con un *Debug* SPI. La programación se hizo en el compilador de lenguaje C para microcontrolador PIC CCS ya que esta herramienta contiene entre sus funciones el manejo del protocolo SPI para la familia de micros PIC16F87x.

3.1 INICIALIZACIÓN DE LA MEMORIA EN MODO SPI

Para que la memoria pueda escribir o leer un sector determinado primero se debe inicializar en modo SPI, es un paso fundamental en el manejo de la memoria, además para que la ella se inicialice de forma correcta se debe enviar el comando CMD0, CMD1 y CMD16 sin activar el pin C2 (0 V) [6].

Para inicializar la memoria el PIC debe enviar los seis Bytes del comando CMD0, lo seis Bytes de comando CMD1 y los seis Bytes del comando CMD16.

Al iniciar la memoria se establece en modo SD. Para entrar en modo SPI se debe enviar el comando CMD0 con el pin CS a 0V, si la memoria reconoce la petición de cambio de protocolo responde con la respuesta R1 (0x01) como se ilustra en la figura 2.

Figura 2. Respuesta al cambio de protocolo

En la figura 2 se puede ver como el microcontrolador manda la secuencia 0x40, 0x00, 0x00, 0x00, 0x00, 0x95 (CMD0) y como la memoria responde acertadamente al comando CMD0 con un 0x01 [5].

Para inicializar la memoria se debe mandar por el microcontrolador el comando CMD1 con el pin C2 a 0 V la estructura del comando tiene la siguiente forma: (0x41, 0x00, 0x00, 0x00, 0x00, 0xFF), una vez recibido los datos la memoria responde acertadamente con 0x00. Como se muestra en la figura 3.

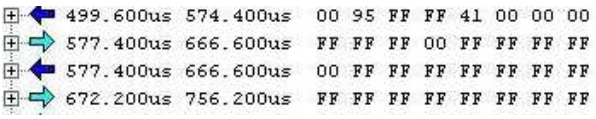

Figura 3. Respuesta a la inicialización

En la figura 3 se puede ver como el microcontrolador manda la secuencia 0x41, 0x00, 0x00, 0x00, 0x00, 0xFF (CMD1) y como la memoria responde acertadamente al comando CMD1 con un 0x00.

Para configurar el bloque de escritura y lectura (CMD16) el microcontrolador debe enviar los argumentos del comando (6 Bytes) indicando en los 4 bytes adicionales (2-5 Bytes) el tamaño del bloque, para las pruebas se configuro el bloque a 512 Bytes, entonces, la cadena a enviar por el microcontrolador es (0x50, 0x00, 0x00, 0x02, 0x00, 0xFF), una vez recibido los datos la memoria responde afirmativamente con un 0x00 como se ilustra en la figura 4.

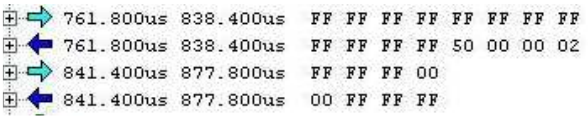

Figura 4. Respuesta a la configuración RD/WR

En la figura 4 se puede ver como el microcontrolador manda la secuencia 0x50, 0x00, 0x00, 0x02, 0x00, 0xFF (CMD16) y como la memoria responde acertadamente al comando CMD16 con un 0x00.

De la figura 2, 3 y 4 se puede observar como la memoria siempre responde con 0xFF y es debido a que la memoria no debe quedar inactiva, el microcontrolador siempre debe enviar por el bus SPI 0xFF para mantener en alto la memoria. Para realizar las pruebas se utilizó el circuito que se observa en la figura 5. El divisor de tensión con resistencias reduce el voltaje de 5V a cerca de 3V para que la memoria no sufra daños.

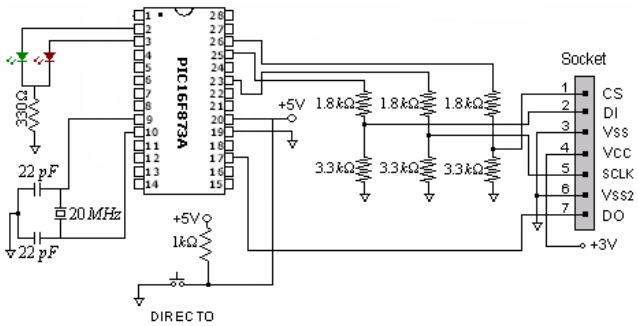

Figura 5. Conexión entre el PIC y la memoria

3.2 ESCRITURA Y LECTURA DE UN BLOQUE DE 512 BYTES

Las operaciones de lectura y escritura se realizan enviando el comando correspondiente junto a la dirección del primer byte del bloque con el largo indicado anteriormente (Comando CMD16). El largo del bloque puede ser desde 1 hasta 512 Bytes, y no esta permitido realizar operaciones en dos sectores a la vez, o sea que si el largo de bloque fijado en CMD16 es 512 Bytes, la dirección para realizar lectura o escritura debe ser la del byte inicial del sector [4].

Para realizar escritura de un único bloque debemos enviar el comando CMD24 (0x58, 0x00, 0x00, 0x02, 0x00, 0xFF) indicando la dirección del bloque en el argumento de la función. La memoria al reconocer el comando envía la respuesta R1, donde puede indicar si hay algún error. Si todo es correcto el PIC debe enviar un *token* (0xFE) y luego los 512 datos del bloque más 2 bytes de CRC. Luego de enviados estos datos debemos quedar a la espera de una respuesta de la memoria indicando si los datos se han recibido correctamente o ha ocurrido un error, además de condición de desocupado. Si no hubo un error la memoria responde con 0x05 tal como se ilustra en la figura 6 y en la figura 7 [4].

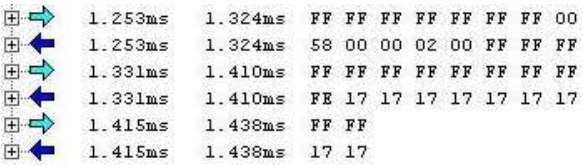

Figura 6. Configuración para escritura

En la figura 6 se observa como el microcontrolador envía la cadena de 6 Bytes (0x58, 0x00, 0x00, 0x02, 0x00, 0xFF), es importante notar que el argumento indica que se va a escribir en el segundo bloque de la memoria, también se puede visualizar como la memoria responde acertadamente (0x00) a la petición de escritura, después el microcontrolador envía el *token* 0xFE y los 512 Bytes.

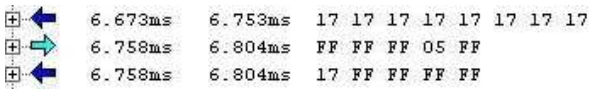

Figura 7. Respuesta a la configuración de escritura

En la figura 7 se puede observar que una vez escritos los 512 datos el microcontrolador envía dos 0xFF y después la memoria responde acertadamente con 0x05

Para realizar una lectura debemos enviar el comando CMD17 (0x51, 0x00, 0x00, 0x02, 0x00, 0xFF) indicando en el argumento la dirección del bloque. Luego se espera la respuesta R1 desde la memoria, si todo es correcto se pasa a recibir el *token* (0xFE) y luego los datos, la cantidad es establecida por el largo del bloque.(CMD16). Lo anterior se muestra en la figura 8 [4].

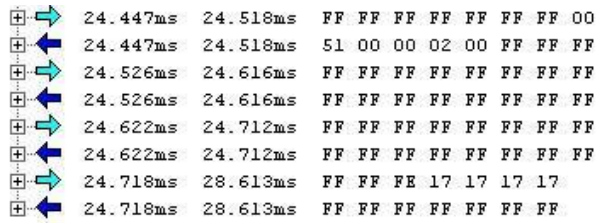

Figura 8. Configuración del modo lectura

En la figura 8 se puede observar como se envía el comando CMD17 y como la memoria responde acertadamente (0x00), después de algunos pulsos de reloj la memoria envía el *token* (0xFE) indicando que va a enviar los datos y después se envían los datos desde la memoria hacia el microcontrolador.

4. PROGRAMACIÓN EN EL CCS

El programa del microcontrolador fue escrito en lenguaje C, utilizando el compilador CCS, debido a que contiene funciones para el manejo del protocolo SPI y además facilita mucho las labores del programador. Las funciones son la siguientes [3].

SETUP_SPI(Mode)

Esta función inicializa el SPI; el *Mode* puede ser:

- SPI_MASTER, SPI_SLAVE
- SPI_L_TO_H, SPI_H_TO_L
- SPI_CLK_DIV_4, SPI_CLK_DIV_16,
- SPI_CLK_DIV_64, SPI_CLK_T2
- SPI_SS_DISABLED

SPI_WRITE(Valor) : Esta función escribe el valor por el buffer del SPI.

SPI_READ(): Esta función devuelve un valor leído por el SPI. Cuando SPI_READ() recibe un dato, se temporiza, y la función devuelve el valor leído. Si no hay datos dispuestos, SPI_READ() permanece a la espera.

SPI_READ(Valor): Además de hacer lo de la función SPI_READ(), también, carga el buffer con *Valor*.

4.1 RUTINA DE RESET (CMD0) EN CCS

La rutina que se presenta a continuación maneja el comando CMD0 se puede observar en la rutina como es la configuración de protocolo SPI, envío del comando (CMD0) y respuesta de la memoria ante el comando (CMD0). La rutina es la siguiente:

```
int init_mmc()
  {
   SETUP_SPI(SPI_MASTER | SPI_H_TO_L |
SPI_CLK_DIV_16 | SPI_XMIT_L_TO_H);
   OUTPUT_HIGH(PIN_C2);
   for(i=0; i<20; i++)
    {
 // espero a que se inicialice la memoria que van desde 8
a 10 pulsos del reloj
      SPI_WRITE(0xFF);
    }
   OUTPUT_LOW(PIN_C2);          // Activo la tarjeta
   for(i=0; i<20; i++)
    {

 // espero a que se inicialice la memoria que van desde 8
a 10 pulsos del reloj

      SPI_WRITE(0xFF);
    }
// Comando CMDO datos a enviar

   SPI_WRITE(0x40);
   SPI_WRITE(0x00);
   SPI_WRITE(0x00);
   SPI_WRITE(0x00);
   SPI_WRITE(0x00);
   SPI_WRITE(0x95);
if(mmc_respuesta(0x01)==1)
    {
// Error en memoria por que se supero el tiempo
      return 1;
    }
// Retorna 0 si todo fue exitoso
   return 0
  }
```

La función mmc_respuesta solo espera a que la memoria responda acertadamente o equivocadamente, la rutina es la siguiente:

```
int mmc_respuesta(unsigned char respuesta)
  {
   int ayer;
   unsigned long espera = 0xFFFF;
   while(SPI_READ(0xFF)!= respuesta && espera> 0)
    {
```

```
    espera= espera - 1;
   }
  if(espera==0)
   {
   return 1;              // se ha superado el tiempo
   }
  else
   {
   return 0;        // Retorna 0 si respuesta es la correcta
   }
 }
```

5. CONCLUSIONES

1. La memoria Flash (MMC/SD) es un dispositivo que se puede aplicar en gran diversidad de diseños electrónicos debido a su gran capacidad de almacenamiento, su fácil programación y su accesibilidad debido a su bajo precio
2. El compilador CCS es una herramienta poderosa que permite programar este tipo de dispositivos utilizando funciones relativamente sencillas.
3. La programación de microcontroladores en lenguaje ensamblador optimiza el rendimiento del micro y reduce el espacio de memoria ocupado, pero en tareas complejas como operaciones matemáticas y comunicación con elementos externos, la programación en lenguaje C facilita mucho el proceso.

5. BIBLIOGRAFÍA

[1] José María Angulo Usategui, Diseño Práctico de Aplicaciones de Microcontroladores PIC16F877, Mcgraw-hill, Segunda Edición, España 2000.

[2] Jose María Angulo Usategui, Diseño Práctico de Aplicaciones Primera Parte PIC16F84 Lenguaje Pbasic y Ensamblador, Tercera Edición, Mcgraw-Hill, España 2003.

[3] Andrés Cánova López, Manual de Usuario del Compilador PCW de CCS, Microchip, España 2000.

[4] Hoja de datos memoria SD SanDisk Securite Digital Card.

[5] Hoja de datos memoria MMC Samsung Electronics MultimeMediaCard Specification.

[6] Héctor Fabio Restrepo Espinosa, Diseño de un Prototipo Holter Basado en Plataforma DSP, Universidad Tecnológica de Pereira, Pereira 2006.

[7] Ivan Pfarher, Germán Reula, Pablo Tovorosky, Miguel Zanin, Módulo portátil de adquisición de señales electrocardiográficas y registro en memorias SD, Universidad Tecnológica Nacional.

[8] Foro de microntroladores TODOPIC. www.todopic.com.ar/foros

[9] Compilador C-CCS y Simulador Proteus para microcontroladores PIC. Eduardo García Breijo. Ed. Alfaomega Marcombo. Primera Edición, 2008.