Nota de Aplicación: CAN-055
Título: **Lectura de datos externos de tarjetas MMC**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	02/09/06	
1	23/03/07	modificación menor con efectos mayores (casting en cálculo de sector desde cluster)

En la CAN-054 vimos la forma de acceder a estas interesantes y económicas tarjetas flash mediante una interfaz SPI. En esta oportunidad, mostraremos una forma fácil de leer información de éstas, que puede haber sido grabada en cualquier otro sistema si están formateadas utilizando el sistema de archivos FAT16.

Información de las tarjetas MMC y de FAT16

Tanto SD como MMC, FAT y FAT16, son standards de las respectivas organizaciones que definen su funcionamiento y lo documentan. El developer o fabricante de productos puede adquirir dicha documentación directamente de estas organizaciones. En este caso en particular, utilizamos información disponible de forma gratuita en Internet, la cual puede no ser del todo correcta. Sin embargo, el código de la presente nota de aplicación ha funcionado correctamente con las tarjetas MMC que hemos probado.

Debido a que dicha información no nos pertenece, no podemos publicarla junto a esta nota ni tampoco distribuirla, se recomienda al lector conseguir y leer la información pertinente para así poder comprender esta nota de aplicación.

Hardware

Utilizamos el mismo hardware descripto en CAN-054

Software

El código de bajo nivel puede obtenerse de CAN-054. Aquí presentaremos una forma simple y rápida para poder buscar un archivo en el directorio principal y poder leerlo sector por sector. Dada la simpleza de la operación, centrada en mostrar el procedimiento más que en optimizar el funcionamiento, sólo es posible abrir un archivo por vez. Seguramente, el lector encontrará muy fácil el agregar arrays e indirecciones para soportar varios archivos simultáneos, si así lo desea.

Comenzamos nuestra tarea con una vieja rutina desarrollada hace ya muchos años, en un rápido hack cuya misión era leer un directorio de un floppy de 5,25", FAT12. Dado que la máquina empleada en ese entonces estaba basada en Z-80, no ha sido nada difícil portarla. Dicha rutina, se encarga de hacer la búsqueda en el directorio de un nombre, soportando el "formato reducido" *nombre.extensión* cuando el nombre es menor a 8 caracteres. La razón por la que la utilizamos, es que debido a que la operación se realiza de forma repetitiva, nos pareció que aprovechar una vieja rutina assembler que resolvía el problema era más eficiente que escribir una función nueva en C.

```
#asm
FIND::
      LD IY,buffer
FLOP: BIT 3,(IY+11)           ; Descarta directorios y nombres largos
      JR NZ,FLOP1
      BIT 2,(IY+11)
      JR NZ,FLOP1
      LD A,(IY)               ; fin
      AND A
      JR Z,NONE
      CP (HL)                 ; compara 1er caracter
      JR NZ,FLOP1
      CALL GO1                ; match, sigue
      JR NC,FLOP1             ; falsa alarma, siguiente
```

```
      PUSH HL                 ; el nombre es igual
      PUSH IY
      LD L,C
      LD H,B
      LD BC,7
      ADD IY,BC               ; compara extensión
      LD B,3
      CALL GO2
      POP IY
      POP HL
      JR NC,FLOP1             ; falsa alarma, siguiente
      LD HL,IY                ; lo encontré !
      RET
FLOP1: LD DE,32               ; siguiente entrada
      ADD IY,DE
      EX DE,HL
      LD HL,IY
      LD BC,buffer+512        ; fin de sector ?
      AND A
      SBC HL,BC
      EX DE,HL
      JR C,FLOP
NONE: LD HL,-1                ; no encontrado
      RET
;
GO1:  LD B,7                  ; 7 caracteres más
GO2:  PUSH IY
      PUSH HL
GLOP: INC HL
      INC IY
      LD A,(IY)               ; compara
      CP (HL)
      JR NZ,DIF1              ; diferencia, chequeo extra
      DJNZ GLOP
      INC HL                  ; listo !
IGU:  LD C,L
      LD B,H
      POP HL
      POP IY
      SCF
      RET
DIF1: CP 0x20                 ; detecta puntos en el nombre: k1.txt -> k1      .txt
      JR NZ,DIF               ; pidiendo k1.txt es válido encontrar 'k1      ' en el dir
      LD A,'.'
      CP (IY-1)
      JR Z,DIF
      CP (HL)
      JR Z,IGU
DIF:  POP HL
      POP IY
      AND A
      RET
#endasm
```

A continuación, la rutina que realiza una operación similar a montar el file system. Leemos el sector 0 en la tarjeta y luego el primer sector de la primera partición, imprimiendpo la informaión relevante, como hiciéramos en CAN-054. esta vez, además, inicializamos algunas variables que nos van servir más adelante, las cuales guardarán la información del sistema de archivos.

```
int MMCFAT_mount(unsigned int part)
{
long addr;

       if(part>3)
       return(-1);
       if(MMC_RdSector(0L, buffer)==MMC_TRANSFER_OK){
              printf("File system descriptor for first partition: ");
              switch(buffer[0x1be+0x10*part+4]){
              case 4:
              printf("DOS 16-bit FAT <32MB\n");
           break;
              case 6:
              printf("DOS 16-bit FAT >32MB\n");
```

```
            break;
                    default:
               printf("Unsupported\n");
            return(-1);
          }
           addr=(*((long *)&buffer[0x1be+0x10*part+8]));
                       printf("Address of first sector in first partition: %08lX\n\n",addr);
                       if(MMC_RdSector(addr, buffer)==MMC_TRANSFER_OK){
               buffer[0xb]=0;
            printf("OEM name: %s\n",&buffer[3]);
               buffer[0x3e]=0;
            printf("Volume label, file system type: %s\n",&buffer[0x2b]);
            sectors_cluster=buffer[0x0D];
            fat_offset=addr+*((int *)&buffer[0x0e]);
            root_dir_offset=fat_offset+(buffer[0x10]*(*((int *)&buffer[0x16])));
            data_area_offset=root_dir_offset+((long)*((int *)&buffer[0x11])*32)/512;
            printf("Sectors per cluster: %d\n",sectors_cluster);
            printf("Root directory entries: %d\n",*((int *)&buffer[0x11]));
            printf("FAT first sector: %ld\n",fat_offset);
            printf("Root directory first sector: %ld\n",root_dir_offset);
            printf("Data area first sector: %ld\n",data_area_offset);
            return(1);
          }
        }
   return(-1);
}

static unsigned long offset_in_file,file_lenght;
static unsigned int sector_in_cluster;
static unsigned int first_file_cluster,cur_file_cluster,cur_file_offset;
```

Para abrir un archivo, necesitamos buscarlo en el directorio raíz, y si existe, inicializaremos nuestras variables con los datos del mismo. En una implementación más completa, utilizaríamos una estructura conteniendo todos los elementos requeridos, de la cual crearíamos un array para poder manejar varios archivos a la vez. En nuestro caso, decidimos mantener las cosas lo más simple posible, y manejar variables sueltas nos pareció más claro.

```
void MMCFAT_rewind(void)
{
      cur_file_cluster=first_file_cluster;
      cur_file_offset=0;
      sector_in_cluster=0;
      offset_in_file=0;
}

int MMCFAT_open(char *name)
{
int entry;
long sector;

       sector=root_dir_offset;
       do {                                          // para cada sector del directorio
              if(MMC_RdSector(sector++, buffer)==MMC_TRANSFER_OK){        // lo leo
                     if((entry=FIND(name))!=-1){                          // busco el nombre
                            if(!(*(unsigned char *)(entry+11)&0x10)){    // es archivo ?
                                   file_lenght=*((long *)(entry+28));    // copio
                                   printf("%ld bytes\n",file_lenght);    // datos
                                   first_file_cluster=*(int *)(entry+26);
                                   MMCFAT_rewind();              // inicializo variables
                                   return(first_file_cluster);
                            }
                     }
              }
       } while(sector < data_area_offset);
       return(-1);
}
```

De esta forma, obtuvimos el primer cluster correspondiente al archivo. Para poder leer todos los datos del mismo, deberemos, cluster por cluster, ir leyendo sus correspondientes sectores, e ir buscando en la FAT en qué cluster continúa. Para ello, desarrollaremos primero unas rutinas para encontrar un sector en base al

número de cluster, luego para encontrar el cluster siguiente dentro de la FAT, y finalmente las rutinas de desplazamiento dentro del archivo y de lectura del mismo.

```
#define MMCFAT_cluster2sector(i)     ((unsigned long)sectors_cluster*(i-2)+data_area_offset)

int MMCFAT_nextcluster(int cluster)
{
long FATsector;
int FATentry;

        FATsector=fat_offset+(((long)cluster)>>8);                  // 2*cluster/512, <<1 >>9
        FATentry=(cluster<<1)&0x01FF;                               // (2*cluster)%512
        if(MMC_RdSector(FATsector, buffer)!=MMC_TRANSFER_OK)
                return(-1);
        return(buffer[FATentry]+(buffer[FATentry+1]<<8));
}

int MMCFAT_seek(long pos)
{
unsigned int clusters,curcluster,newcluster,sectors,bytes;
unsigned char *bptr;

        if(pos<0)
                MMCFAT_rewind();
        if(pos==-1L)
        pos=0;
        sectors=(int)(pos >> 9);                                    // /bytes per sector
        bytes=(int)(pos&0x1FF);                                     // %bytes per sector
        clusters=sectors/sectors_cluster;
        sectors%=sectors_cluster;
        bytes+=cur_file_offset;
        sectors+=sector_in_cluster;
        if(bytes>=512){
                bytes-=512;
                sectors++;
        }
        if(sectors>=sectors_cluster){
                sectors-=sectors_cluster;
                clusters++;
        }
        newcluster=curcluster=cur_file_cluster;
        while(clusters--){
                if((newcluster=MMCFAT_nextcluster(curcluster))==-1){
                        if(pos<0L){
                                offset_in_file=file_lenght;
                                break;
                        }
                        else
                                return(-1);
                }
                curcluster=newcluster;
        }
        if(MMC_RdSector(MMCFAT_cluster2sector(curcluster)+sectors, buffer)!=MMC_TRANSFER_OK)
                return(-1);
        cur_file_cluster=curcluster;
        sector_in_cluster=sectors;
        cur_file_offset=bytes;
        if(offset_in_file!=file_lenght)
        offset_in_file+=pos;
        return(1);
}

int MMCFAT_read(unsigned char *buf,int bytes)
{
int count;
unsigned char *bptr;

        if(offset_in_file==0L){
                if(MMCFAT_seek(0)==-1)
                return(-1);
        }
        count=bytes;
        bptr=buffer+cur_file_offset;
        while(bytes&&(offset_in_file<file_lenght)){
```

```
                if(cur_file_offset>=512){
                        cur_file_offset=0;
                        bptr=buffer;
                        if(++sector_in_cluster==sectors_cluster){
                                sector_in_cluster=0;
                                if((cur_file_cluster=MMCFAT_nextcluster(cur_file_cluster))==-1)
                                        break;
                        }
                        if(MMC_RdSector(MMCFAT_cluster2sector(cur_file_cluster)+
                          sector_in_cluster,buffer)!=MMC_TRANSFER_OK)
                                break;
                }
                *(buf++)=*(bptr++);
                cur_file_offset++;
                offset_in_file++;
                bytes--;
        }
        return(count-bytes);
}
```

Utilización

En base a estas simples rutinas, ya podemos realizar la lectura de cualquier archivo ubicado en el directorio raíz, recordando que solamente soportaremos FAT16 y nombres en 8.3. el ejemplo siguiente muestra en pantalla el contenido de LICENSE.TXT, luego se posiciona en la posición 10700 del mismo, y muestra 2Kbytes a partir de allí.

```
int main( void )
{
unsigned char rc;
int i;
char aux[2049];
long fsize;

        if(MMCFAT_init()==-1)
                goto oops;
        if(MMCFAT_mount(0)==-1)
                goto oops;
        if(MMCFAT_open("LICENSE.TXT")!=-1) {
                fsize=0;
                do {
                        i=MMCFAT_read(aux,2048);
                        aux[i]=0;
                        puts(aux);
                        fsize+=i;
                } while (i==2048);
        }
        if(MMCFAT_seek(-1L)!=-1){
                if(MMCFAT_seek(10700L)!=-1){
                        i=MMCFAT_read(aux,2048);
                        aux[i]=0;
                        puts(aux);
                }
        }

oops:   BitWrPortI ( PFDR, &PFDRShadow, 0, 0 );      // no PWR
        printf("%ld bytes\n",fsize);
}
```