Nota de Aplicación: CAN-054
Título: **Utilización de tarjetas MMC en bajo nivel**
Autor: Sergio R. Caprile, Senior Engineer, colaboración IT&T (distr. Cika en Mendoza)

Revisiones	Fecha	Comentarios
0	02/09/06	

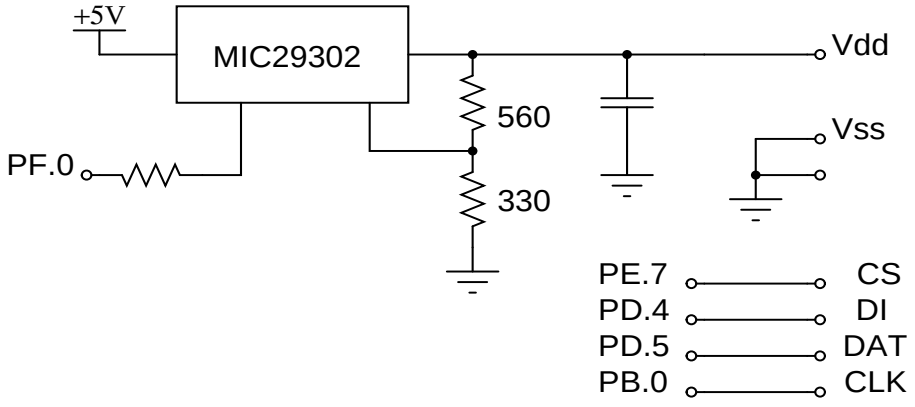
Una opción interesante y económica a la hora de almacenar grandes cantidades de datos, son las tarjetas flash. Entre ellas, las MMC y SD presentan la particularidad de poder ser controladas mediante una interfaz SPI, característica que explotaremos en esta oportunidad.

Información de las tarjetas MMC

Tanto SD como MMC 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

Una característica muy importante del hardware, es que la tarjeta debe recibir alimentación con señales inactivas y estables, y éstas deben operar en un relativamente corto tiempo luego de recibida la alimentación. Es decir, no es posible tenerla conectada directamente al módulo, compilar el proyecto, y pretender que la tarjeta responda. Por esta razón, hemos incorporado un control de la alimentación de la tarjeta, de modo de poder mantenerla sin alimentación hasta tanto hayamos inicializado el hardware del micro. Por comodidad y disponibilidad, empleamos un regulador LDO con pin de control; cada developer resolverá el problema como lo encuentre más conveniente.



Software

El código que presentamos es un código simple destinado a facilitar la comprensión y permitir la utilización de estas tarjetas con módulos Rabbit. Por tal motivo, los ciclos de espera se realizan directamente, sin provisión para ejecutar otras tareas. Queda a cuenta del lector realizar las modificaciones pertinentes para adecuarlos a sus necesidades particulares, en su aplicación.

En esta sección, a fin de concentrarnos en la operación, omitiremos las definiciones de los valores particulares de constantes como los comandos, y algunas variables globales. La totalidad del listado se encuentra en el archivo que acompaña a esta nota, la cual se centra en la operación.

Lo primero que debemos hacer al dar alimentación a la tarjeta, es esperar un cierto tiempo enviando pulsos de clock, y luego habilitarla y comenzar su secuencia de reset e inicialización en modo SPI. Esto lo haremos mediante las tres funciones que detallamos a continuación:

```
unsigned char MMC_init (unsigned int timeout)
{
unsigned char response;
int t;

        BitWrPortI ( PEDR, &PEDRShadow, 1, 7 );      // disable CS
        t=MMC_TMOUT_RST;                             // espera:
        while(t--)                                   // envía algunos pulsos de clock
                SPIWrite(DUMMYBUF,1);                // sin data
        BitWrPortI ( PEDR, &PEDRShadow, 0, 7 );      // enable CS

        MMC_SndCmd(MMC_GO_IDLE_STATE,0L,MMC_CMD0CRC); // comando de reset
        while((response=MMC_GetResponse(MMC_TMOUT_NAC))==MMC_R1_IDLING){
                if(!timeout--)
                        break;                       // sale por timeout
                BitWrPortI ( PEDR, &PEDRShadow, 1, 7 ); // disable CS
                SPIWrite(DUMMYBUF,1);                // clocks extra
                BitWrPortI ( PEDR, &PEDRShadow, 0, 7 ); // enable CS
                MMC_SndCmd(MMC_SEND_OP_COND,0L,DUMMY); // envía inicialización
        }
        BitWrPortI ( PEDR, &PEDRShadow, 1, 7 );      // disable CS
        SPIWrite(DUMMYBUF,1);                        // extra clocks
        return (response);
}


void MMC_SndCmd (unsigned char cmd, unsigned long data, unsigned char crc)
{
unsigned char *a,buf[6];
int i;

        a=(unsigned char *) &data;                   // en el buffer, LSB primero
        buf[0]=cmd|0x40;                             // transmitter bit, start bit es 0
        i=4;
        do {
                buf[i]=(*(a++));                     // transmitimos MSB primero
        } while(--i);
        buf[5]=((crc<<1)|0x01);                      // end bit
        SPIWrite(buf,6);                             // Send
}


char MMC_GetResponse(int timeout)
{
unsigned char response;

        while(timeout--){
        SPIRead(&response,1);
                if(response!=DUMMY)
                        break;
        }
        return(response);
}
```

Si la inicialización fue exitosa, obtendremos MMC_R1_OK, caso contrario recibiremos un mensaje de error de la tarjeta o abortaremos por timeout.

Una vez inicializada la tarjeta, ya podemos leer y escribir en ella. Para realizar cualquier operación, enviaremos un comando utilizando la función MMC:SndCmd que describiéramos en el punto anterior. A continuación detallaremos las funciones que utilizamos para leer y escribir un bloque de datos en la tarjeta. A fin de simplificar, utilizaremos bloques de 512 bytes coincidentes con los sectores por defecto en las mismas.

Lectura

Para leer, enviamos el comando de lectura y a continuación comenzaremos a recibir los datos de la tarjeta. Si la operación fue exitosa, obtendremos MMC_TRANSFER_OK, y los datos en el buffer; caso contrario recibiremos un mensaje de error o un data token de la tarjeta, a lo cual abortaremos.

```
unsigned char MMC_rdblk(unsigned char *buf,unsigned int len,)
{
unsigned char response,CRCbuf[2];

       if ((response=MMC_GetResponse(MMC_TMOUT_NCR))==MMC_R1_OK) {
              if ((response=MMC_GetResponse(MMC_TMOUT_NAC))==MMC_START_BLOCK_SREAD) {
                     SPIRead(buf,len);       // get block
                     SPIRead(CRCbuf,2);            // get CRC (ignored)
                     response=MMC_TRANSFER_OK;
              }
       }
   return(response);
}

unsigned char MMC_RdSector(unsigned long sector,unsigned char *buf)
{
unsigned char response;
int i;
unsigned long address;

       address=sector << MMC_SECTORL2;
       BitWrPortI ( PEDR, &PEDRShadow, 0, 7 );      // enable CS
       MMC_SndCmd(MMC_READ_SINGLE_BLOCK,address,DUMMY);
   response=MMC_rdblk(buf,MMC_SECTORSZ);
       BitWrPortI ( PEDR, &PEDRShadow, 1, 7 );      // disable CS
       SPIWrite(DUMMYBUF,1);
       return(response);
}
```

Escritura

La operación de escritura es un poco más complicada. Primero enviamos el comando de escritura, luego esperamos la respuesta de la tarjeta; a continuación enviamos un start token, seguido de los datos. Luego esperamos una respuesta positiva de la tarjeta, y deberemos luego esperar a que termine la operación de escritura. Mientras dure la misma, la tarjeta mantendrá su pin de comunicaciones en estado bajo.
Si el comando es aceptado, recibiremos MMC_R1_OK, si los datos son aceptados, recibiremos MMC_ACCEPTED, a lo cual esperaremos a que la tarjeta termine de escribir, consultando periódicamente el estado del pin de salida.

```
unsigned char MMC_bsywait(int timeout)
{
unsigned char response;

       while(timeout--){
              SPIRead(&response,1);
              if(response!=MMC_BUSY)
              break;
  }
  return(response);
}

unsigned char MMC_WrSector (unsigned long sector, unsigned char *buf)
{

unsigned char response;
int i;
const static unsigned char wrcmd[]={DUMMY,MMC_START_BLOCK_SWRITE};
unsigned long address;

       address=sector << MMC_SECTORL2;
       BitWrPortI ( PEDR, &PEDRShadow, 0, 7 );      // enable CS
       MMC_SndCmd(MMC_WRITE_BLOCK,address,DUMMY);
```

```
        if((response=MMC_GetResponse(MMC_TMOUT_NCR))==MMC_R1_OK) {
      SPIWrite(wrcmd,2);                                          // send start token
               SPIWrite(buf,MMC_SECTORSZ);
               SPIWrite(DUMMYBUF,2);                  // send dummy CRC bytes
      response=MMC_GetResponse(MMC_TMOUT_NCR)&MMC_DRMASK;
               if(response==MMC_ACCEPTED)
        if((response=MMC_bsywait(MMC_TMOUT_BSY))==DUMMY)
                              response=MMC_TRANSFER_OK;
        }
        BitWrPortI ( PEDR, &PEDRShadow, 1, 7 );      // disable CS
        SPIWrite(DUMMYBUF,1);
        return(response);
}
```

Utilización

En base a estas simples rutinas, ya podemos realizar operaciones más complejas. Lo que vamos a hacer, para comprobar al menos que la lectura funcione correctamente, es imprimir en pantalla algunos datos de la tarjeta, el directorio principal, y el contenido de un sector de la misma. Para esto, solamente soportaremos FAT16 y nombres en 8.3. La estructura de este sistema de archivos puede obtenerse de su creador, o en algunos sitios públicos de Internet.

```
int main( void )
{
unsigned char rc;
int i;
long addr;
char aux[20];

        for(i=0;i<MMC_SECTORSZ;i++)
                buffer[i]=0;

        MMC_hardwareinit();                                 // inicializa pines del Rabbit
        BitWrPortI ( PFDR, &PFDRShadow, 1, 0 );             // alimenta la tarjeta
        if((rc=MMC_init(1000))!=MMC_R1_OK){                 // inicializa la tarjeta
                printf ("init: %02X\n", rc);                // muestra error
                exit(1);                                    // adios
        }
        if((rc=MMC_RdSector(0, buffer))==MMC_TRANSFER_OK){  // lee sector 0 y muestra
                printf("File system descriptor for first partition: ");
                switch(buffer[0x1be+4]){
                case 0:
                        printf("Empty");
                        exit(1);
                case 4:
                        printf("DOS 16-bit FAT <32MB\n");
                        break;
                case 6:
                        printf("DOS 16-bit FAT >32MB\n");
                        break;
                default:
                        printf("Unsupported\n");
                        exit(1);
                }
                addr=(*((long *)&buffer[0x1be+8]));
                printf("Address of first sector in first partition: %08lX\n\n",addr);
                if((rc=MMC_RdSector(addr, buffer))==MMC_TRANSFER_OK){ // lee primer sector en
                                                                      // la primera partición
                        buffer[0xb]=0;
                        printf("OEM name: %s\n",&buffer[3]);
                        buffer[0x3e]=0;
                        printf("Volume label, file system type: %s\n",&buffer[0x2b]);
                        addr+=((*((int *)&buffer[0x0e]))+
                                buffer[0x10]*(*((int *)&buffer[0x16])));
                        printf("Sectors per cluster: %d\n",buffer[0x0D]);
                        printf("Root directory entries: %d\n",*((int *)&buffer[0x11]));
                        printf("Root directory sector: %ld\n",addr);
                        if(printsector(addr)!=MMC_TRANSFER_OK)       // lee directorio raíz
                                exit(1);                             // y muestra en hexa
                        printf("\n\nDirectory:\n\n");
                        i=0;
```

```
                        while((buffer[i]!=0)&&(i<512)){             // muestra entradas
                                if((buffer[i]!=0xE5)&&(!(buffer[i+11]&0x0C))){
                                        strncpy(aux,&buffer[i],8);
                                        aux[8]=0;
                                        printf("%s\t",aux);
                                        strncpy(aux,&buffer[i+8],3);
                                        aux[3]=0;
                                        printf("%s\t",aux);
                                        if(buffer[i+11]&0x10)
                                                printf("<DIR>\t");
                                        else
                                                printf("%ld\t",*((long *)&buffer[i+28]));
                                        printf("%02d/%02d/%02d\t",(int)(buffer[i+24]&0x1F),
                                                (*((int *)&buffer[i+24])&0x01E0)>>5,
                                                (int)((buffer[i+25]&0xFE)>>1)-20);
                                        printf("%02d:%02d:%02d\n",(int)(buffer[i+23]>>3),
                                                (*((int *)&buffer[i+22])&0x07E0)>>5,
                                                (int)(buffer[i+25]&0x1F));
                                }
                                i+=32;
                        }
                }
                else printf("result: %02X\n",rc);     // no pudo leer sector, muestra error
        }
        else printf("result: %02X\n",rc);            // no pudo leer sector, muestra error

        BitWrPortI ( PFDR, &PFDRShadow, 0, 0 );      // saca alimentación
}

int printsector(long sector)
{
unsigned char rc;
int i;

        if((rc=MMC_RdSector(sector, buffer))==MMC_TRANSFER_OK){     // lee sector
                for(i=0;i<MMC_SECTORSZ;i++){
                        if((i&0xFF0)==i)                            // da formato
                                printf("\n%04X\t",i);
                        printf("%02X ",buffer[i]);                  // muestra datos
                }
        }
        else printf("result: %02X\n",rc);                          // muestra error
        return(rc);
}

void MMC_hardwareinit(void)
{
        SPIinit();
        BitWrPortI ( PEDDR, &PEDDRShadow, 1, 7 );    // CS = output
        BitWrPortI ( PEFR, &PEFRShadow, 0, 7 );      // I/O pin
        BitWrPortI ( PFDDR, &PFDDRShadow, 1, 0 );    // PWR = output
        BitWrPortI ( PFFR, &PFFRShadow, 0, 0 );      // I/O pin
        BitWrPortI ( PEDR, &PEDRShadow, 1, 7 );      // disable CS
        BitWrPortI ( PFDR, &PFDRShadow, 0, 0 );      // no PWR
};
```