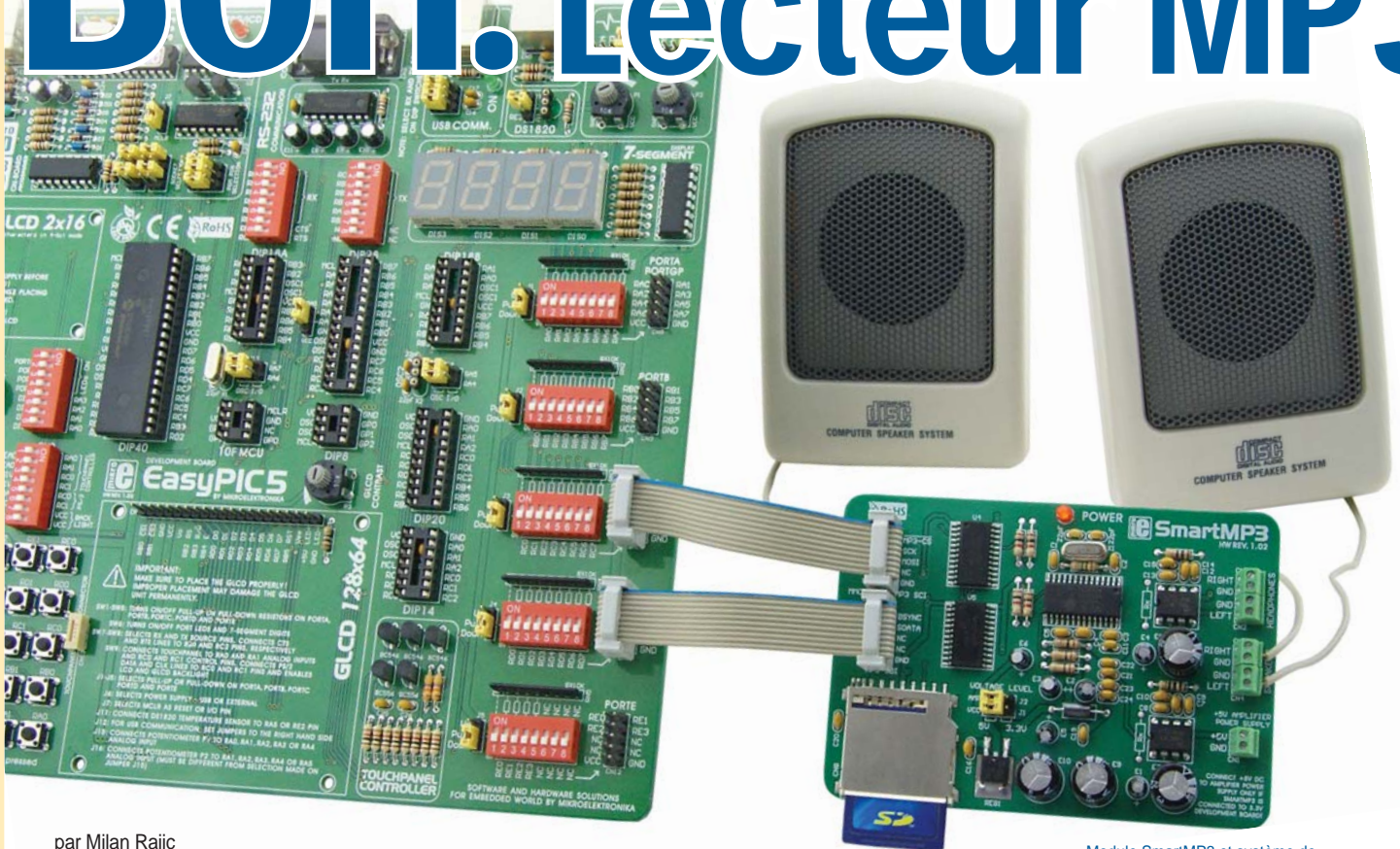


# Maintenant il vous faut un ...

# Bon. Lecteur MP3



par Milan Rajic  
MikroElektronika - Software Department

Module SmartMP3 et système de développement EasyPIC5

Le format MP3 a révolutionné la compression des signaux audio grâce à ses taux de compression élevés. Si vous voulez intégrer des messages audio ou musicaux dans votre projet, vous allez pouvoir le faire très vite, il vous faudra juste une carte mémoire MMC ou SD standard, quelques puces et un peu de temps pour y parvenir...

Avant de commencer il est nécessaire de formater la carte MMC et d'y charger le fichier sonore 1.mp3 (la carte doit être formaté en FAT16 ou FAT). La qualité du son codé en MP3 dépend de la fréquence d'échantillonnage et du taux de transfert. A l'instar des CD audio, beaucoup de fichiers MP3 sont échantillonnés à 44,1 kHz. Le taux de transfert d'un fichier MP3 indique la qualité de l'audio comprimé comparée à celle de l'originale non compressée, autrement dit, sa fidélité. Un taux de transfert de 64 kbit/s est suffisant pour reproduire la voix, tandis que 128 kbit/s ou davantage est nécessaire pour reproduire correctement de la musique. Le présent exemple utilise un fichier musique à 128 kbit/s.

## Matériel

Le contenu du fichier sonore est codé en format MP3 et un décodeur MP3 est nécessaire pour son décodage. Notre exemple utilise la puce VS1011E comme décodeur MP3. Cette puce décode un flux MP3 et fait la conver-

sion numérique/analogique pour que le son puisse être restitué par des haut-parleurs connectés à un (petit) amplificateur audio. Vu que les cartes MMC/SD utilisent des sections de 512 octets, un microcontrôleur possédant aux moins 512 octets de mémoire RAM est nécessaire pour contrôler le décodage MP3. Nous avons choisi le PIC18F4520, qui possède 1536 octets de RAM.

## Logiciel

Le programme du microcontrôleur comporte cinq étapes :

- Etape 1 :** L'initialisation du module SPI du microcontrôleur.
- Etape 2 :** L'initialisation de la librairie Mmc\_FAT16 permettant la lecture des fichiers MP3 à partir des cartes MMC ou SD.
- Etape 3 :** La lecture d'une partie du fichier.
- Etape 4 :** L'envoi des données à la mémoire tampon du décodeur MP3.
- Etape 5 :** Si la fin du fichier n'est pas atteinte, repasser à l'étape 3.

## Test

Il est recommandé de lancer le test de fonctionnement du montage avec un faible taux de transfert et de l'augmenter petit à petit. Le tampon du décodeur MP3 a une taille de 2048 octets. Si le tampon est rempli avec une partie d'un fichier MP3 à 128 kbit/s, il contiendra deux fois plus d'échantillons audio que s'il est rempli avec un fichier MP3 à 256 kbit/s. En conséquence, il faut plus de temps pour décodifier le contenu du tampon si le taux de transfert du fichier est faible. Si nous utilisons un taux de

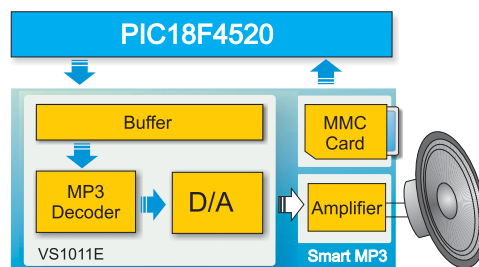


Figure 1. Synoptique du module SmartMP3 connecté à un PIC 18F4520

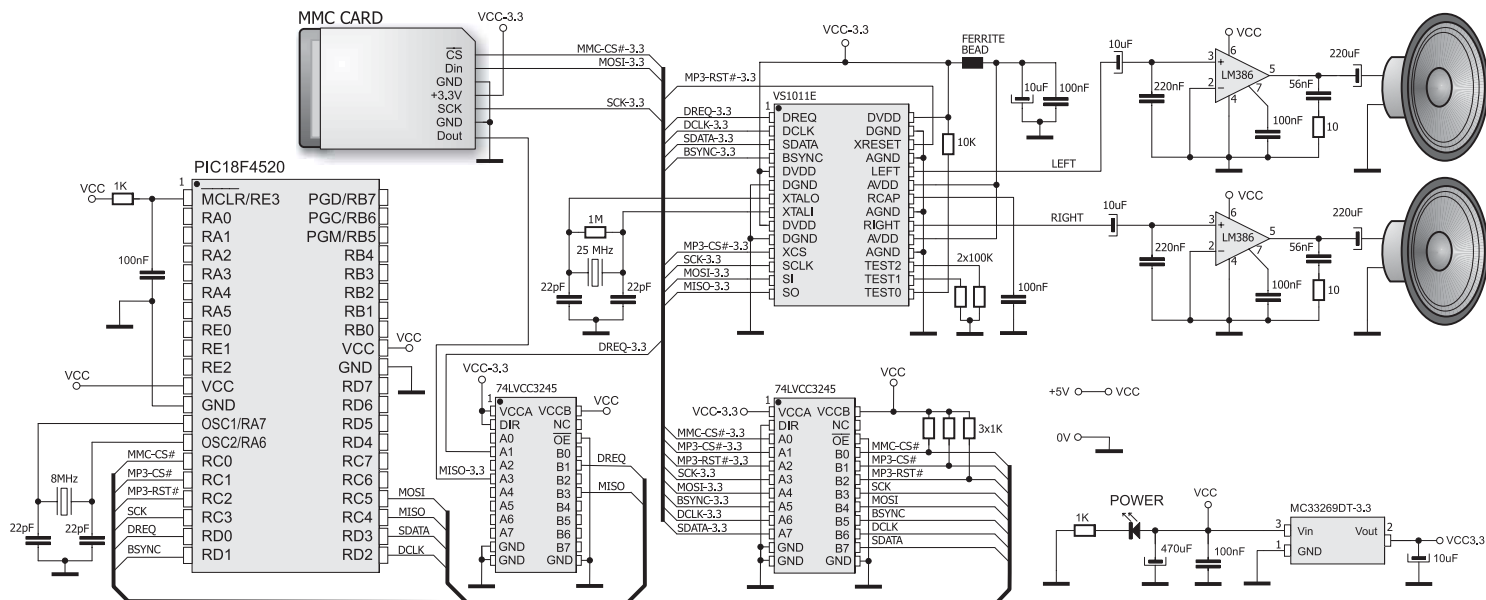
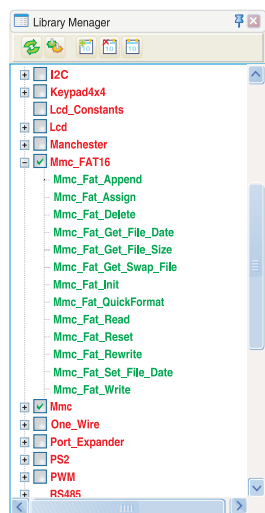


Schéma 1. Connexion du module Smart MP3 à un PIC18F4520

transfert trop élevé, il se peut que le contenu du tampon soit décodé avant que le microcontrôleur a pu lire et charger la suite de fichier de la carte dans le tampon, ce qui provoquerait des discontinuités dans le son. Si cela arrive, nous pouvons réduire le taux de transfert du fichier MP3 ou utiliser un quartz de 8 MHz ou plus. Voir schéma 1. Ne vous inquiétez pas, notre programme a été testé sur plusieurs familles de microcontrôleurs avec des quartz différents et il est capable de décoder les fichiers MP3 de moyenne et bonne qualité. D'autre part, un faible taux de transfert signifie que le tampon du décodeur peut contenir un son plus long et il se peut que le décodeur n'a pas décodé le contenu du tampon avant que nous essayons de le recharger. Pour éviter ceci, il faut s'assurer que le décodeur est prêt à recevoir de nouvelles données avant de les envoyer. Autrement dit, il faudra attendre que le signal data request (DREQ) du décodeur passe à un (1).

### Extensions

Cet exemple peut être étendu après l'avoir testé. Le signal DREQ peut être testé périodiquement. Une routine pour contrôler le volume ou un filtrage de graves/aigus etc. peut être rajouté au programme. La librairie MMC vous permet de sélectionner un fichier portant un nom différent. Ainsi est-il possible de créer plusieurs messages ou sons MP3, utilisables dans d'autres applications, en envoyant le bon fichier MP3 au décodeur. Voici une liste des fonctions contenues dans la librairie *Mmc\_FAT16*. Cette librairie est intégrée dans le compilateur *mikroBASIC for PIC*.



<b>Mmc_Fat_Append()</b>	Ecriture à la fin d'un fichier
<b>Mmc_Fat_Assign()*</b>	Affecter le fichier pour opérations FAT
<b>Mmc_Fat_Delete()</b>	Effacer un fichier
<b>Mmc_Fat_Get_File_Date()</b>	Lire la date et l'heure d'un fichier
<b>Mmc_Fat_Get_File_Size()</b>	Lire la taille d'un fichier
<b>Mmc_Fat_Get_Swap_File()</b>	Créer un fichier swap
<b>Mmc_Fat_Init()*</b>	Initialiser la carte pour opérations FAT
<b>Mmc_Fat_QuickFormat()</b>	
<b>Mmc_Fat_Read()*</b>	Lire des données depuis un fichier
<b>Mmc_Fat_Reset()*</b>	Ouvrir un fichier pour lecture
<b>Mmc_Fat_Rewrite()</b>	Ouvrir un fichier pour écriture
<b>Mmc_Fat_Set_File_Date()</b>	Ecrire la date et l'heure d'un fichier
<b>Mmc_Fat_Write()</b>	Ecrire des données dans un fichier

**\* fonctions de Mmc\_FAT16 utilisées dans le programme**

Autres fonctions de *mikroBASIC for PIC* utilisées dans le programme:

**Spi\_Init\_Advanced()** Initialiser le module SPI du microcontrôleur

### Programme montrant le fonctionnement du module Smart MP3

```

program MP3_Simple_Test
const BUFFER_SIZE = 512
dim filename as string[13]
i, file_size as dword
data_buffer_32 as byte[32]
BufferLarge as byte[BUFFER_SIZE]
sub procedure SW_SPI_Write(dim data as byte) 'Writes one byte to MP3 SDI
PORTD.1 = 1 'Set BSYNC before sending the first bit
PORTD.2 = 0 PORTD.3 = data_0 PORTD.2 = 1 'Send data_LSB, data_0
PORTD.2 = 0 PORTD.3 = data_1 PORTD.2 = 1 'Send data_1
PORTD.1 = 0 'Clear BSYNC after sending the second bit
PORTD.2 = 0 PORTD.3 = data_2 PORTD.2 = 1 'Send data_2
PORTD.2 = 0 PORTD.3 = data_3 PORTD.2 = 1 'Send data_3
PORTD.2 = 0 PORTD.3 = data_4 PORTD.2 = 1 'Send data_4
PORTD.2 = 0 PORTD.3 = data_5 PORTD.2 = 1 'Send data_5
PORTD.2 = 0 PORTD.3 = data_6 PORTD.2 = 1 'Send data_6
PORTD.2 = 0 PORTD.3 = data_7 PORTD.2 = 1 'Send data_7
PORTD.2 = 0
end sub
'Writes one word to MP3 SCI
sub procedure MP3_SCI_Write(dim address as byte, dim data_in as word)
PORTC.1 = 0 'select MP3 SCI
SPI_write(0x02) 'send WRITE command
SPI_write(address)
SPI_write(data_in >> 8) 'Send High byte
SPI_write(data_in) 'Send Low byte
PORTC.1 = 1 'deselect MP3 SCI
while (PORTD.0 = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet,
Serial Protocol for SCI
end sub
'Reads words_count words from MP3 SCI
sub procedure MP3_SCI_Read(dim start_address, words_count as byte, dim data_buffer as
^byte)
dim i as byte
PORTC.1 = 0 'select MP3 SCI
SPI_write(0x03) 'send READ command
SPI_write(start_address)
for i = 1 to (2*words_count)
data_buffer^ = SPI_read(0) 'read words_count words byte per byte
' read and store a byte
point to next byte
next i
PORTC.1 = 1 'deselect MP3 SCI
while (PORTD.0 = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet,
Serial Protocol for SCI
end sub
sub procedure MP3_SDI_Write(dim data as ^byte) 'Write one byte to MP3 SDI
while (PORTD.0 = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet,
Serial Protocol for SCI
SW_SPI_Write(data_)
end sub
sub procedure MP3_SDI_Write_32(dim data as ^byte) 'Write 32 bytes to MP3 SDI
dim i as byte
while (PORTD.0 = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet,
Serial Protocol for SCI
for i = 1 to 32
SW_SPI_Write(data_^) 'Write byte pointed by data
Inc(data_)
next i
end sub
sub procedure Set_Clock(dim clock_khz as word, dim doubler as byte) 'Set clock
clock_khz = clock_khz / 2 'calculate value
if (doubler > 0) then clock_khz = clock_khz or 0x8000 end if
MP3_SCI_Write(0x03, clock_khz_) 'Write value to CLOCKS register
end sub
sub procedure Init()
ADCON1 = ADCON1 or 0x0F 'Configure AN pins as digital
CMCON = CMCON or 7 'Disable comparators
TRISD.2 = 0 TRISD.3 = 0 'Set SW SPI pin directions to output
PORTD.2 = 0 PORTD.3 = 0 'Clear SW SPI_SCK and SDO
TRISC.1 = 0 PORTC.1 = 1 'Configure MP3_CS as output and deselect MP3_CS
TRISC.2 = 0 PORTC.2 = 1 'Configure MP3_RST as output and set MP3_RST pin
TRISD.1 = 1 'Configure DREQ as input
TRISD.0 = 0 PORTD.1 = 0 'Configure BSYNC as output and clear BSYNC
end sub
sub procedure Soft_Reset()
MP3_SCI_Write(0x00, 0x204) 'Software Reset
' Set SM_RESET bit and SM_BITORD bit(bitorder is LSB first)
Delay_us(2) 'Required, see MP3 codec datasheet -> Software Reset
while (PORTD.0 = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet,
Serial Protocol for SCI
for i = 1 to 2048 MP3_SDI_Write(0) next i 'feed 2048 zeros to the MP3 SDI bus
end sub
main:
filename = "sound1.mp3" 'Set File name
Init()
SPI_init_advanced(MASTER_OSC_DIV64, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_High)
Soft_Reset() Set_Clock(25000, 0) 'SW Reset, set clock to 25MHz, do not use clock doubler
if (Mmc_Fat_Init(PORTC, 0) = 0) then
if (Mmc_Fat_Assign(filename, 0) < 0) then Assign file "sound1.mp3"
Mmc_Fat_Reset(file_size) 'Call Reset before file reading
while (file_size > BUFFER_SIZE) 'Send file blocks to MP3 SDI
for i = 0 to BUFFER_SIZE - 1
Mmc_Fat_Read(BufferLarge[i])
next i
for i = 0 to BUFFER_SIZE/32 - 1 'Send file block to mp3 decoder
MP3_SDI_Write_32@BufferLarge + i*32)
next i
file_size = file_size - BUFFER_SIZE 'Decrease file size
wend
' send the rest of the file to MP3 SDI
for i = 0 to file_size - 1 Mmc_Fat_Read(BufferLarge[i]) next i
for i = 0 to file_size - 1 MP3_SDI_Write(BufferLarge[i]) next i
end if
end if

```



GO TO

Les codes source de cet exemple en C, BASIC et PASCAL pour microcontrôleurs PIC®, ainsi que tous les programmes écrits pour les microcontrôleurs dsPIC® et AVR® sont disponibles sur notre site Internet : [www.mikroe.com/en/article/](http://www.mikroe.com/en/article/)