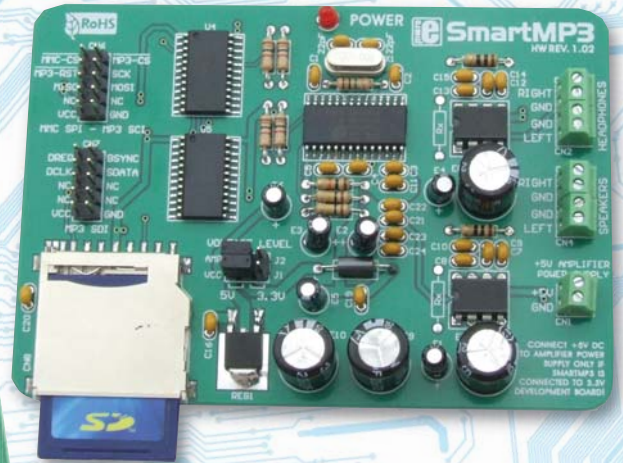


Maintenant il vous faut un ...

Bon. Lecteur MP3



Module SmartMP3 et système de développement LV24-33A

par Milan Rajic
MikroElektronika - Software Department

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 PIC24FJ96GA010, qui possède 1536 octets de RAM.

Logiciel

Le programme du microcontrôleur comporte cinq étapes :

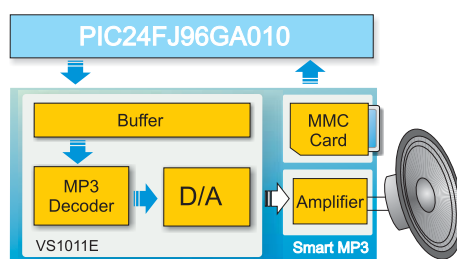


Figure 1. Synoptique du module Smart MP3 connecté à un PIC24FJ96GA010

- 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

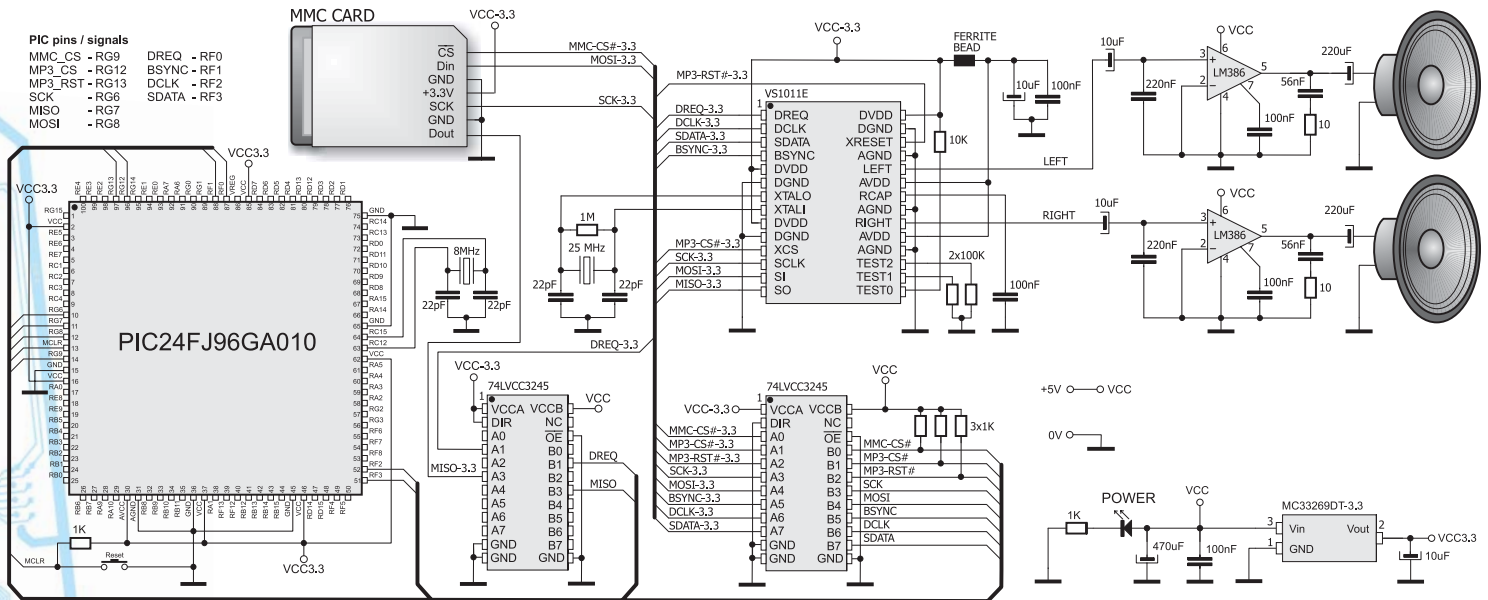


Schéma 1. Connexion du module Smart MP3 à un PIC24FJ96GA010

est faible. Si nous utilisons un taux de transfert trop élevé, il se peut que le contenu du tampon soit décodé avant que le microcontrôleur a pu lire et charger la suite du fichier depuis 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.

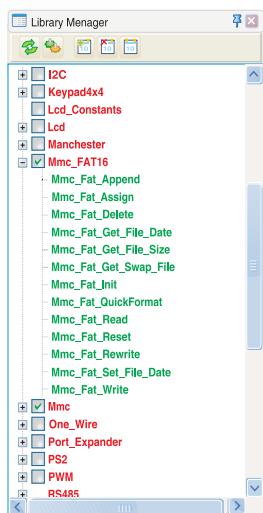
Ne vous inquiétez pas, notre programme a été testé sur plusieurs familles de microcontrôleurs avec des quartz différents et il est capable de décoder les fichiers MP3 de moyenne et bonne qualité.

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 *mikroPASCAL for dsPIC*.



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

* fonctions de Mmc_FAT16 utilisées dans le programme

Autres fonctions de *mikroPASCAL for dsPIC* 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 = 2048;
var filename: string[13];
    i, file_size: dword;
    data_buffer: array[32] of byte;
    BufferLarge: array[BUFFER_SIZE] of byte;
procedure SW_SPI_Write(data_: byte); begin
    PORT1.1 := 1;
    PORT2.0 := 0; PORT3 := data_0; PORT2.1 := 1;
    PORT2.2 := 0; PORT3 := data_1; PORT2.1 := 1;
    PORT1.1 := 0;
    PORT2.2 := 0; PORT3 := data_2; PORT2.1 := 1;
    PORT2.0 := 0; PORT3 := data_3; PORT2.1 := 1;
    PORT2.2 := 0; PORT3 := data_4; PORT2.1 := 1;
    PORT2.0 := 0; PORT3 := data_5; PORT2.1 := 1;
    PORT2.2 := 0; PORT3 := data_6; PORT2.1 := 1;
    PORT2.0 := 0; PORT3 := data_7; PORT2.1 := 1;
    PORT2.2 := 0;
end;
// Writes one word to MP3 SCI
procedure MP3_SCI_Write(address: byte; data_in: word); begin
    PORTG.12 := 0;
    Spi2_Write(0x02);
    Spi2_Write(address);
    Spi2_Write(Hi(data_in));
    Spi2_Write(Lo(data_in));
    PORTG.12 := 1;
    while (PORTF.0 = 0) do nop;
datasheet, Serial Protocol for SCI
end;
// Reads words_count words from MP3 SCI
procedure MP3_SCI_Read(start_address, words_count: byte; data_buffer: ^byte);
var i: byte;
begin
    PORTG.12 := 0;
    Spi2_Write(0x03);
    Spi2_Write(start_address);
    for i := 1 to (2*words_count) do
        begin
            data_buffer[i] := Spi2_Read(0);
            Inc(data_buffer);
        end;
    PORTG.12 := 1;
    while (PORTF.0 = 0) do nop;
datasheet, Serial Protocol for SCI
end;
procedure MP3_SDI_Write(data_: byte); begin
    while (PORTF.0 = 0) do nop;
datasheet, Serial Protocol for SCI
    SW_SPI_Write(data_);
end;
procedure MP3_SDI_Write_32(data_: ^byte);
var i: byte;
begin
    while (PORTF.0 = 0) do nop;
datasheet, Serial Protocol for SCI
    for i := 1 to 32 do begin
        SW_SPI_Write(data_[i]);
        Inc(data_);
    end;
end;
procedure Set_Clock(clock_khz: word; doubler: byte); begin // Set clock
    clock_khz := clock_khz / 2;
    // calculate value
    if (doubler > 0) then clock_khz := clock_khz or 0x8000;
    MP3_SCI_Write(0x03, clock_khz);
    // Write value to LOCKF register
end;
procedure Soft_Reset(); begin
    MP3_SCI_Write(0x00, 0x204); // Set SM_RESET bit and SM_BITORD bit (bitorder is LSB first)
    Delay_us(2); // Required, see MP3 codec datasheet -> Software Reset
    while (PORTF.0 = 0) do nop;
datasheet, Serial Protocol for SCI
    for i := 1 to 2048 do MP3_SDI_Write(0); // feed 2048 zeros to the MP3 SDI bus
end;
procedure Init(); begin
    ADPCFG := 0xFFFF; // set all AN pins to digital
    PORT2.2 := 0; TRISF.2 := 0; // Clear SW SPI SCK, configure pin as output
    PORT3.3 := 0; TRISF.3 := 0; // Clear SW SPI SDA, configure pin as output
    PORTG.12 := 1; TRISG.12 := 0; // Deselect MP3_CS, configure pin as output
    TRISF.13 := 1; TRISG.13 := 0; // Set MP3_RST pin, configure pin as output
    TRISF.1 := 0; TRISG.1 := 0; // Configure DREQ as input
    PORTF.1 := 0; TRISF.1 := 0; // Clear BSYNC, configure pin as output
end;
begin
    // main function
    filename := 'sound1.mp3'; // Set File name
    Init();
    Spi2_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);
    Soft_Reset(); Set_Clock(25000, 0); // SW Reset, set clock to 25MHz
    if (Mmc_Fat_Init(PORTG, 9) = 0) then
        if (Mmc_Fat_Assign(filename, 0) <> 0) then
            begin
                Mmc_Fat_Reset(file_size); // Call Reset before file reading.
                // procedure returns size of the file
                // send file blocks to MP3 SDI
                while (file_size > BUFFER_SIZE) do begin
                    for i := 0 to BUFFER_SIZE - 1 do Mmc_Fat_Read(BufferLarge[i]);
                    for i := 0 to BUFFER_SIZE / 32 - 1 do MP3_SDI_Write_32(@BufferLarge + i * 32);
                    file_size := file_size - BUFFER_SIZE;
                end;
                // send the rest of the file to MP3 SDI
                for i := 0 to file_size - 1 do Mmc_Fat_Read(BufferLarge[i]);
                for i := 0 to file_size - 1 do MP3_SDI_Write(BufferLarge[i]);
            end;
end;

```



GO TO

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