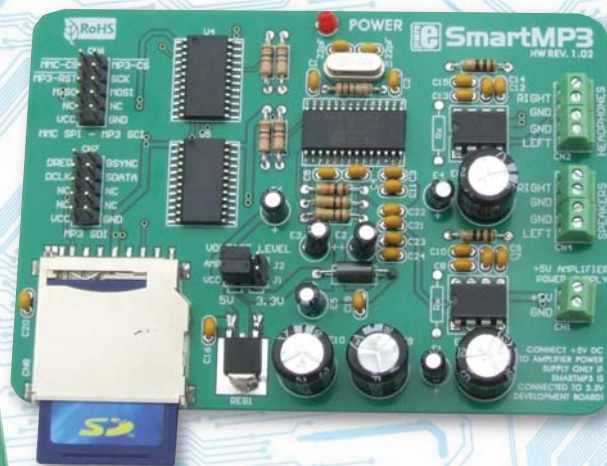


# OK. Nu hebt u een ... MP3-speler



SmartMP3 module en LV24-33A Development System

Door: Milan Rajic  
MikroElektronika – Software Department

Het MP3-formaat heeft met zijn veel kleinere audiobestanden een revolutie in de digitale geluidcompressietechnologie teweeg gebracht. Als u wilt dat audiomedelingen of muziek deel van uw project gaan uitmaken, dan kunt u dat nu makkelijk doen. Het enige wat u nodig hebt, is een standaard MMC- of SD-geheugenkaartje, een paar chips en wat tijd...

U begint met formatteren van het MMC-kaartje en slaat daarop het bestand sound1.mp3 op (het kaartje moet worden geformatteerd in FAT16, dat wil zeggen het bestandstype FAT). De geluidskwaliteit in MP-3-formaat is afhankelijk van bemonsteringsfrequentie en bitrate. Net als bij een audio-CD worden MP-3-bestanden bemonsterd met een frequentie van 44,1 kHz. De bitrate van het MP3-bestand is, in vergelijking met het oorspronkelijke ongecomprimeerde geluid, een maatstaf voor de kwaliteit van het gecompriëerde audiosignaal, dat wil zeggen voor de getrouwheid ervan. Voor het reproduceren van spraak is een bitrate van 64 kbit/s voldoende, maar voor het reproduceren van muziek moet dat 128 kbit/s zijn. In dit voorbeeld wordt voor een muziekbestand een bitrate van 128 kbit/s gebruikt.

### Hardware

Het in dit bestand opgeslagen geluid is gecodeerd in MP3-formaat, zodat u voor het decoderen ervan een MP3-decoder nodig hebt. In dit voorbeeld gebeurt dat met de chip VS1011E. Deze chip decodeert MP3-bestanden, voert een digitaal/analooog-conversie uit op het signaal en

produceert een signaal dat via een kleine laagfrequentversterker aan luidsprekers kan worden toegevoerd.

Gezien het feit dat MMC/SD-kaartjes met 512 bytes grote sectoren werken, is voor het MP3-decodeerproces een microcontroller met 512 byte RAM of meer nodig. Hier is gekozen voor de PIC24FJ96GA010 met 1.536 byte RAM.

### Software

Het programma dat de werking van dit apparaat bestuurt bestaat, uit vijf stappen:

**Stap 1:** Initialiseren van de SPI-module van de microcontroller.

**Stap 2:** Initialiseren van de Mmc\_FAT16-bibliotheek van de compiler, zodat MP3-bestanden vanaf MMC- of SD-kaartjes te kunnen worden gelezen.

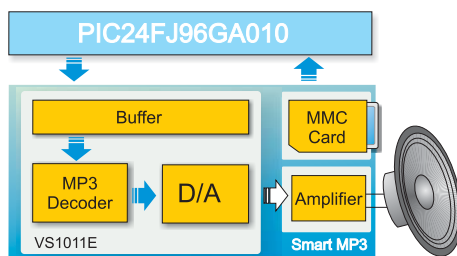
**Stap 3:** Lezen van een deel van het bestand.

**Stap 4:** Verzenden van data naar de buffer van de MP3-decoder.

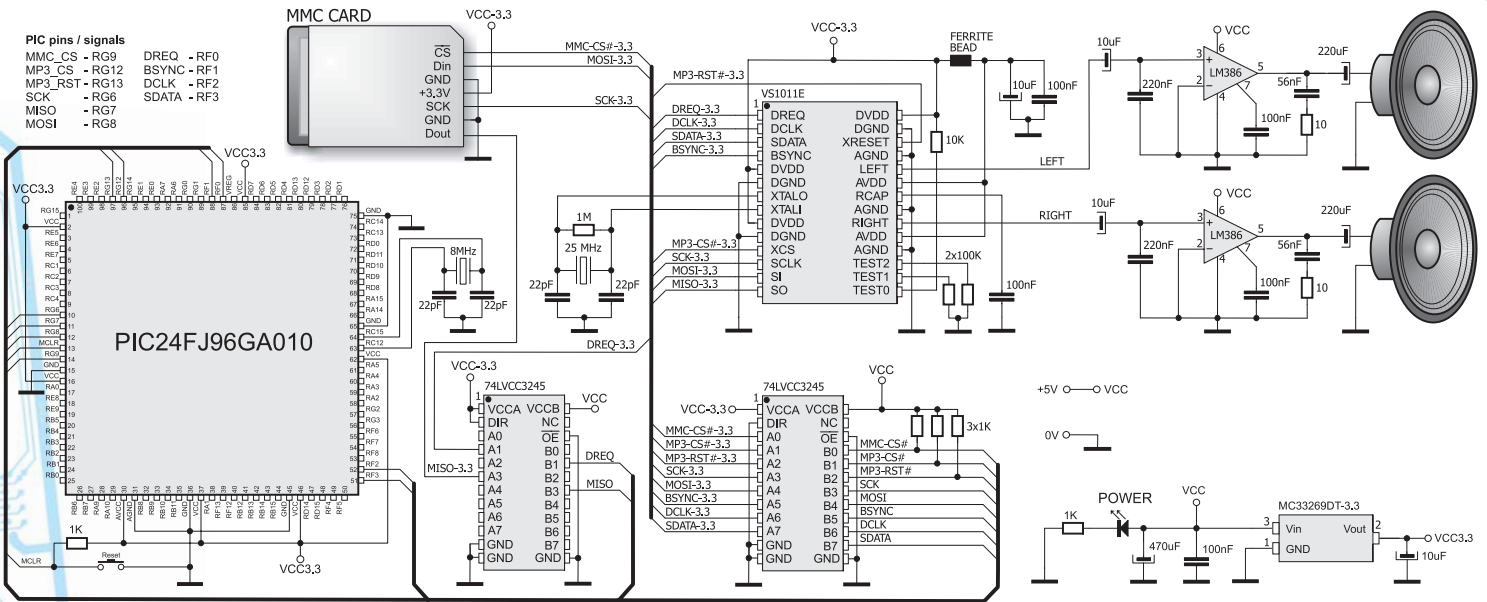
**Stap 5:** Naar stap 3 springen als het einde van het bestand nog niet bereikt is.

### Testen

Het verdient aanbeveling de werking van het apparaat eerst te testen met een lagere bitrate en die geleidelijk op te voeren. De buffer van de MP3-decoder is 2.048 bytes groot. Wordt de buffer met een deel van een MP3-bestand geladen met een snelheid van 128 kbit/s, dan zal de buffer de helft van het aantal geluidsmonsters bevatten dan wanneer de buffer met een deel van een bestand wordt geladen met een snelheid van 256 kbit/s. Bij een lagere bitrate van het bestand zal het decoderen van de bufferinhoud dus langer duren. Wordt de bitrate van het bestand te hoog gekozen, dan kan het gebeuren dat de buf-



Figuur 1. Blokschema Smart MP3-module aangesloten op een PIC van het type PIC24FJ96GA010.



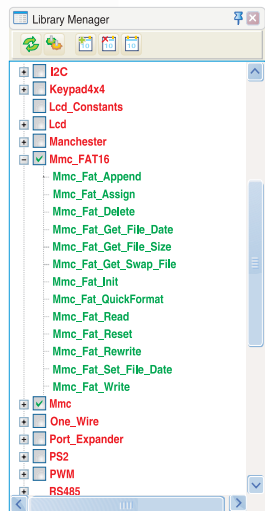
Schema 1. Aansluiten van de Smart MP3-module op een PIC24FJ96GA010

ferinhoud al gecodeerd is voordat de microcontroller er in slaagt het volgende deel van het bestand vanaf het kaartje te lezen en naar de buffer weg te schrijven, waardoor het geluid haperend gaat klinken. Als dat gebeurt kunt u de bitrate van het MP3-bestand verlagen of een kwartskristal met een frequentie van 8 MHz of meer gebruiken. Zie ook schema 1.

Hoe dan ook, u hoeft zich hierover niet te bekommeren omdat het hier beschreven programma werd getest met diverse microcontrollerfamilies met verschillende kristalfrequenties en het MP3-bestanden van gemiddelde en hoge kwaliteit kan decoderen. Een lage bitrate daarentegen betekent dat de bufferdecoder wordt gevuld met geluid van langere duur. Het kan gebeuren dat de decoder er niet in slaagt de bufferinhoud te decoderen voordat getracht wordt de buffer opnieuw te laden. Om dat te voorkomen moet het zeker zijn dat de decoder de nieuwe data kan ontvangen voordat die verzonden worden. Met andere woorden er moet worden gewacht tot het ontvangstbevestigingsignaal (DREQ) van de decoder op logisch één (1) wordt gezet.

### Verbeteringen

Dit voorbeeld kan, nadat het is getest, nog worden uitgebreid. Het DREQ-sig-naal kan periodiek worden gecontroleerd. Ook kan in het programma een routine voor volumeregeling of lage/hogetonen regeling enzovoort worden opgenomen. Het gebruik van een MMC-bibliotheek stelt u in staat bestanden met een andere naam te selecteren. Op die manier kunt u voor gebruik in andere toepassingen een repertoire van MP3-berichten, geluiden of songs aanleggen en, al naar gelang de behoefte, de juiste MP3-bestanden naar de decoder verzenden. Het onderstaande is een lijst van gebruiksklare in de *Mmc\_FAT Library* opgenomen functies. Deze bibliotheek is geïntegreerd in *mikroPASCAL for dsPIC*-compilers.



- Mmc\_Fat\_Append()** Schrijven aan het einde van het bestand
- Mmc\_Fat\_Assign()\*** Bestand toekennen voor FAT-bewerkingen
- Mmc\_Fat\_Delete()** Bestand verwijderen
- Mmc\_Fat\_Get\_File\_Date()** Datum en tijd ophalen
- Mmc\_Fat\_Get\_File\_Size()** Bestandsgrootte ophalen
- Mmc\_Fat\_Get\_Swap\_File()** Wisselbestand aanmaken
- Mmc\_Fat\_Init()\*** Kaartje voor FAT-bewerkingen initialiseren
- Mmc\_Fat\_QuickFormat()**
- Mmc\_Fat\_Read()\*** Data lezen uit bestand
- Mmc\_Fat\_Reset()\*** Bestand openen om te lezen
- Mmc\_Fat\_Rewrite()** Bestand openen om te schrijven
- Mmc\_Fat\_Set\_File\_Date()** Datum en tijd van bestand instellen
- Mmc\_Fat\_Write()** Data naar bestand wegschrijven

\* In het programma gebruikte Mmc\_FAT16-functies.

Andere in dit programma gebruikte *mikroPASCAL for dsPIC*-functies:

- Spi\_Init\_Advanced()** Initialiseren van de microcontroller SPI-module.

### Demonstratieprogramma voor de werking van de Smart MP3-module.

```

program MP3_Simple_Test;
const BUFFER_SIZE = 2048; // global variables
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 //Writes one byte to MP3 SDI
    PORTF.1 := 1; // Set BSYNC before sending the first bit
    PORTF.2 := 0; PORTF.3 := data_0; PORTF.2 := 1; // bitorder is LSB first
    PORTF.2 := 0; PORTF.3 := data_1; PORTF.2 := 1;
    PORTF.1 := 0; // Clear BSYNC after sending the second bit
    PORTF.2 := 0; PORTF.3 := data_2; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_3; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_4; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_5; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_6; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_7; PORTF.2 := 1;
    PORTF.2 := 0;
end;
// Writes one word to MP3 SCI
procedure MP3_SCI_Write(address : byte; data_in : word); begin
    PORTG.12 := 0; // select MP3 SCI
    Spi2_Write(0x02); // Write command
    Spi2_Write(address);
    Spi2_Write(Hi(data_in));
    Spi2_Write(Lo(data_in));
    PORTG.12 := 1; // deselect MP3 SCI
    while (PORTF.0 = 0) do nop; // wait until DREQ becomes 1, see MP3 codec
    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; // select MP3 SCI
    Spi2_Write(0x03); // Read command
    Spi2_Write(start_address);
    for i := 1 to (2*words_count) do // read words_count words byte per byte
        begin
            data_buffer[i] := Spi2_Read(0); // read and store a byte
            Inc(data_buffer); // point to next byte
        end;
    PORTG.12 := 1; // deselect MP3 SCI
    while (PORTF.0 = 0) do nop; // wait until DREQ becomes 1, see MP3 codec
    datasheet, Serial Protocol for SCI
end;
procedure MP3_SDI_Write(data_ : byte); begin //Write one byte to MP3 SDI
    while (PORTF.0 = 0) do nop; // wait until DREQ becomes 1, see MP3 codec
    datasheet, Serial Protocol for SCI
    SW_SPI_Write(data_);
end;
procedure MP3_SDI_Write_32(data_ : ^byte); //Write 32 bytes to MP3 SDI
var i : byte;
begin
    while (PORTF.0 = 0) do nop; // wait until DREQ becomes 1, see MP3 codec
    datasheet, Serial Protocol for SCI
    for i := 1 to 32 do begin
        SW_SPI_Write(data_[i]); //Write byte pointed by 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 CLOCKSFR register
end;
procedure Soft_Reset(); begin // Software Reset
    MP3_SCI_Write(0x00, 0x0204); // 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; // wait until DREQ becomes 1, see MP3 codec
    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 := 0x6FFF; // set all AN pins to digital
    PORTE.2 := 0; TRISF.2 := 0; // Clear SW SPI SCK, configure pin as output
    PORTE.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
    PORTG.13 := 1; TRISG.13 := 0; // Set MP3_RST pin, configure pin as output
    TRISF.1 := 1; PORTF.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
    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

De voor dit voorbeeld geschreven code voor dsPIC \* microcontrollers in C, Basic en Pascal alsmede de programma's geschreven voor PIC \* microcontrollers vindt u op onze website: [www.mikroe.com/en/article](http://www.mikroe.com/en/article)