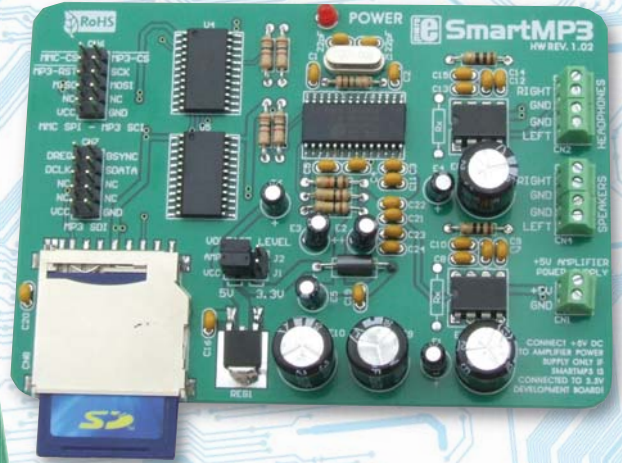


OK. Nu hebt u een ... MP3-speler



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 audiomededelingen 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/analogue-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.

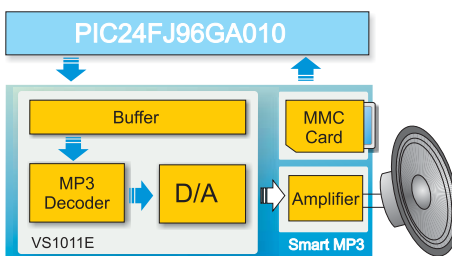
Stap 3: Lezen van een deel van het bestand.

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

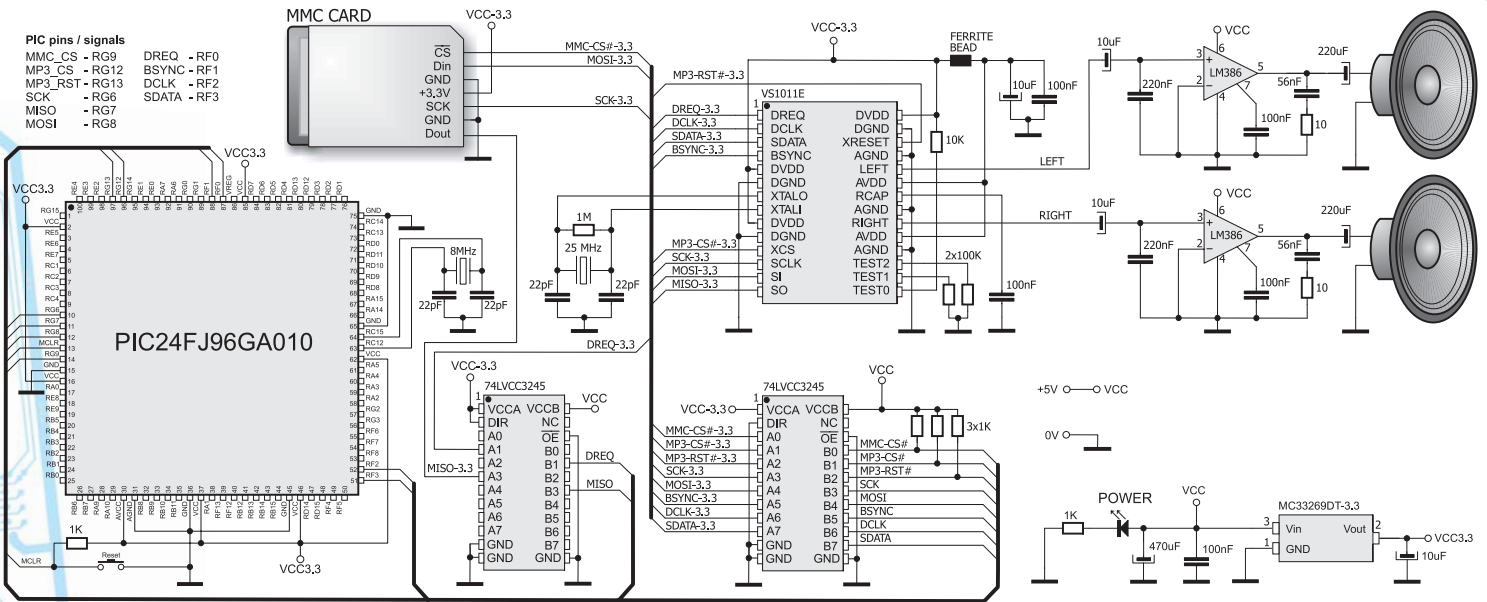
Stap 5: Naar stap 3 springen als het einde van het bestand nog niet bereikt is.

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.



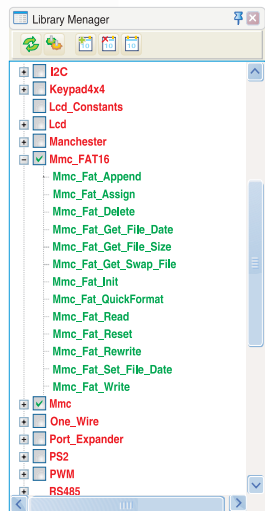
Schema 1. Aansluiten van de Smart MP3-module op een PIC24FJ96GA010

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 *mikroC 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 *mikroC for dsPIC*-functies:

Spi_Init_Advanced() Initialiseren van de microcontroller SPI-module.

Demonstratieprogramma voor de werking van de Smart MP3-module.

```
#include <built_in.h>
#include <spi_const.h>
#define MP3_CS PORTG.F12 // Smart MP3 board connections
#define MP3_RST PORTG.F13
#define DREQ PORTF.F0
#define BSYNC PORTF.F1
#define DCLK PORTF.F3
#define SDATA PORTF.F3
#define MP3_CS_Direction TRISG.F12
#define MP3_RST_Direction TRISG.F13
#define DREQ_Direction TRISF.F0
#define BSYNC_Direction TRISF.F1
#define DCLK_Direction TRISF.F2
#define SDATA_Direction TRISF.F3
char filename[14] = "sound1.mp3"; // global variables
unsigned long i, file_size;
char data_buffer[32];
const BUFFER_SIZE = 2048;
char BufferLarge[BUFFER_SIZE];
void SW_SPI_Write(char data){ // Writes one byte to MP3 SDI
    BSYNC = 1; // Set BSYNC before sending the first bit
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1; // bitorder is LSB first
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    BSYNC = 0; // Clear BSYNC after sending the second bit
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
}
void MP3_SCI_Write(char address, unsigned int data_in){ // Writes one word to MP3 SCI
    MP3_CS = 0; // select MP3 SCI
    Spi2_Write(0x02); Spi2_Write(address); // send WRITE command, send address
    Spi2_Write(Hib(data_in)); Spi2_Write(Lob(data_in)); // Send High byte, send Low byte
    MP3_CS = 1; // deselect MP3 SCI
    while (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
}
// Reads words_count words from MP3 SCI
void MP3_SCI_Read(char start_address, char words_count, unsigned int *data_buffer){
    unsigned int temp;
    MP3_CS = 0; // select MP3 SCI
    Spi2_Write(0x03); Spi2_Write(start_address); // send READ command, send address
    while (words_count--){ // read words_count words byte per byte
        temp = Spi2_Read(0);
        temp <<= 8;
        temp += Spi2_Read(0);
        *(data_buffer++) = temp;
    }
    MP3_CS = 1; // deselect MP3 SCI
    while (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
}
void MP3_SDI_Write(char data){ // Write one byte to MP3 SDI
    while (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
    SW_SPI_Write(data);
}
void MP3_SDI_Write_32(char *data){ // Write 32 bytes to MP3 SDI
    char i;
    while (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
    for (i=0; i<32; i++) SW_SPI_Write(data[i]);
}
void Set_Clock(unsigned int clock_khz, char doubler){ // Set clock
    clock_khz = 2; // calculate value
    if (doubler) clock_khz = 0x8000;
    MP3_SCI_Write(0x03, clock_khz); // Write value to CLOCKF register
}
void Soft_Reset(){ // 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 (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
    for (i=0; i<2048; i++) MP3_SDI_Write(0); // feed 2048 zeros to the MP3 SDI bus
}
void Init(){
    ADPCFG = 0xFF; // set all AN pins to digital
    DCLK = 0; DCLK_Direction = 0; // Clear SW SPI SCK, configure pin as output
    SDATA = 0; SDATA_Direction = 0; // Clear SW SPI SDA, configure pin as output
    MP3_CS = 1; MP3_CS_Direction = 0; // Deselect MP3_CS, configure pin as output
    MP3_RST = 1; MP3_RST_Direction = 0; // Set MP3_RST pin, configure pin as output
    DREQ_Direction = 1; // Configure DREQ as input
    BSYNC = 0; BSYNC_Direction = 0; // Clear BSYNC, configure pin as output
}
void main(){
    // main function
    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_IDLE);
    Soft_Reset(); Set_Clock(25000, 0); // SW Reset, set clock to 25MHz
    if (Mmc_Fat_Init(&PORTG, 9)) // Mmc_Fat_Assign(&filename, 0){
        Mmc_Fat_Reset(&file_size); // Call Reset before file reading
        while (file_size > BUFFER_SIZE){ // send file blocks to MP3 SDI
            for (i=0; i<BUFFER_SIZE; i++) Mmc_Fat_Read(BufferLarge + i);
            for (i=0; i<BUFFER_SIZE; i++) MP3_SDI_Write_32(BufferLarge + i*32);
            file_size -= BUFFER_SIZE;
        }
        // send the rest of the file to MP3 SDI
        for (i=0; i<file_size; i++) Mmc_Fat_Read(BufferLarge + i);
        for (i=0; i<file_size; i++) MP3_SDI_Write(BufferLarge[i]);
    }
}
```



GO TO

De voor dit voorbeeld geschreven code voor dsPIC® microcontrollers in C, Basic en Pascal alsmede de programma's geschreven voor PIC® en AVR® microcontrollers vindt u op onze website: www.mikroe.com/en/article