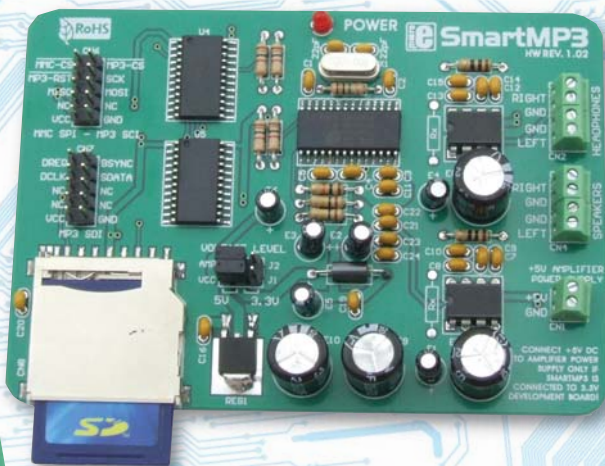


# OK. Nu hebt u een ... MP3-speler



SmartMP3 module en BIGAVR2 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 Atmega128 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.

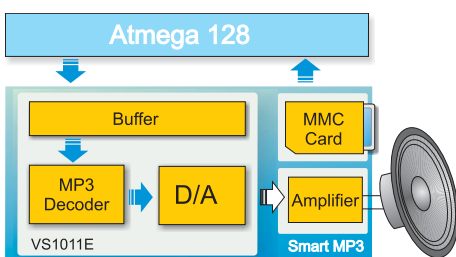
**Stap 3:** Lezen van een deel van het bestand.

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

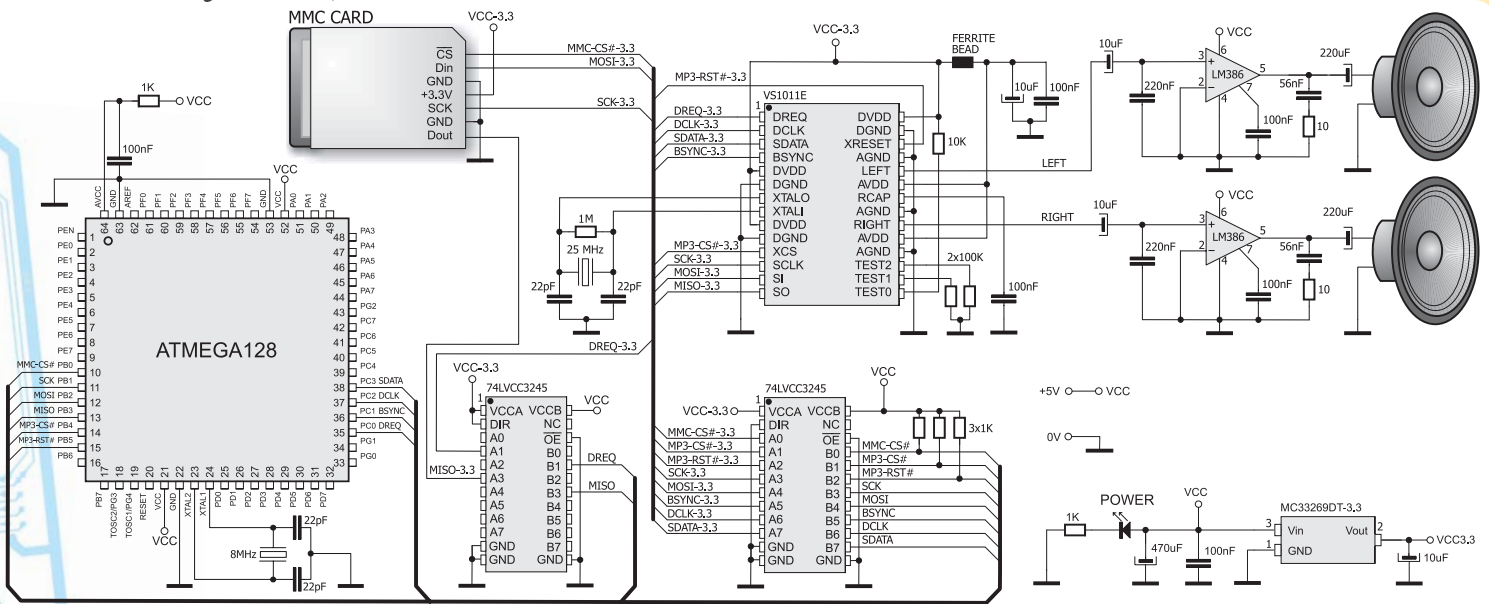
**Stap 5:** Naar stap 3 springen als het einde van het bestand nog niet bereikt is.

### 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 Atmega128.



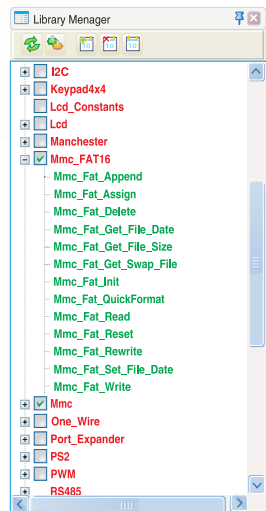
Schema 1. Aansluiten van de Smart MP3-module op een Atmega128

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/hogetonn 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 *mikroBASIC PRO for AVR*-compilers.



<b>Mmc_Fat_Append()</b>	Schrijven aan het einde van het bestand
<b>Mmc_Fat_Assign()*</b>	Bestand toekennen voor FAT-bewerkingen
<b>Mmc_Fat_Delete()</b>	Bestand verwijderen
<b>Mmc_Fat_Get_File_Date()</b>	Datum en tijd ophalen
<b>Mmc_Fat_Get_File_Size()</b>	Bestandsgrootte ophalen
<b>Mmc_Fat_Get_Swap_File()</b>	Wisselbestand aanmaken
<b>Mmc_Fat_Init()*</b>	Kaartje voor FAT-bewerkingen initialiseren
<b>Mmc_Fat_QuickFormat()</b>	
<b>Mmc_Fat_Read()*</b>	Data lezen uit bestand
<b>Mmc_Fat_Reset()*</b>	Bestand openen om te lezen
<b>Mmc_Fat_Rewrite()</b>	Bestand openen om te schrijven
<b>Mmc_Fat_Set_File_Date()</b>	Datum en tijd van bestand instellen
<b>Mmc_Fat_Write()</b>	Data naar bestand wegschrijven

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

Andere in dit programma gebruikte *mikroBASIC PRO for AVR*-functies:

<b>Spi_Init_Advanced()</b>	Initialiseren van de microcontroller SPI-module.
----------------------------	--

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

```

program MP3_Simple_Test
'Smart MP3 board connections
dim Mmc_Chip_Select as sbit at PORTB.B0 dim Mmc_Chip_Select_Direction as sbit at DDRB.B0
dim MP3_CS as sbit at PORTB.B4 dim MP3_CS_Direction as sbit at DDRB.B4
dim MP3_RST as sbit at PORTB.B5 dim MP3_RST_Direction as sbit at DDRB.B5
dim DREQ as sbit at PINC.B0 dim DREQ_Direction as sbit at DDRC.B0
dim BSYN as sbit at PORTC.B1 dim BSYN_Direction as sbit at DDRC.B1
dim DCLK as sbit at PORTC.B2 dim DCLK_Direction as sbit at DDRC.B2
dim SDATA as sbit at PORTC.B3 dim SDATA_Direction as sbit at DDRC.B3
const BUFFER_SIZE = 512
dim filename as string[13]
i, file_size as longword
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
BSYN = 1
DCLK = 0 SDATA = data_0 DCLK = 1
DCLK = 0 SDATA = data_1 DCLK = 1
BSYN = 0
DCLK = 0 SDATA = data_2 DCLK = 1
DCLK = 0 SDATA = data_3 DCLK = 1
DCLK = 0 SDATA = data_4 DCLK = 1
DCLK = 0 SDATA = data_5 DCLK = 1
DCLK = 0 SDATA = data_6 DCLK = 1
DCLK = 0 SDATA = data_7 DCLK = 1
end sub
sub procedure MP3_SCI_Write(dim address as byte, dim data_in as word) 'Writes one word to MP3 SCI
MP3_CS = 0
SPI_Write(0x02) SPI_Write(address) 'Write command, send address
SPI_Write(Hi(data_in)) 'high byte
SPI_Write(Lo(data_in)) 'low byte
MP3_CS = 1
while (DREQ = 0) nop wend 'deselect MP3 SCI
end sub
Protocol for SCI
sub procedure MP3_SCI_Read(dim start_address, words_count as byte, dim data_buffer as ^byte)
dim i as byte
MP3_CS = 0
SPI_Write(0x03) SPI_Write(start_address) 'Read command, send address
for i = 1 to (2*words_count)
data_buffer[i] = SPI_Read(0) 'read and store a byte
Inc(data_buffer) 'point to next byte
next i
MP3_CS = 1
while (DREQ = 0) nop wend 'deselect MP3 SCI
end sub
Protocol for SCI
sub procedure MP3_SDI_Write(dim data_ as byte) 'Write one byte to MP3 SDI
while (DREQ = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet, Serial
end sub
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 (DREQ = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet, Serial
for i = 1 to 32
SW_SPI_Write(data_[i]) 'Write byte pointed by 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
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 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) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet, Serial
end sub
Protocol for SCI
for i = 1 to 2048 MP3_SDI_Write(0) next i 'feed 2048 zeros to the MP3 SDI bus
end sub
sub procedure Init()
DCLK_Direction = 1 DCLK = 0
SDATA_Direction = 1 SDATA = 0
MP3_CS_Direction = 1 MP3_CS = 1
DREQ_Direction = 1 MP3_RST = 1
BSYN_Direction = 1 BSYN = 0
end sub
main:
filename = "sound1.mp3"
Init()
SPI_Init_Advanced(SPI_MASTER, SPI_FCY_DIV128, SPI_CLK_LO_LEADING)
Spi_Rd_Ptr = @SPI_Read
Soft_Reset() Set_Clock(25000.0)
if (Mmc_Fat_Init()) = 0 then
'reinitialize spi at higher speed
SPI_Init_Advanced(SPI_MASTER, SPI_FCY_DIV2, SPI_CLK_LO_LEADING)
if (Mmc_Fat_Assign(filename, 0) < 0) then
Mmc_Fat_Reset(file_size)
while (file_size > BUFFER_SIZE)
while (file_size - BUFFER_SIZE) MP3_SDI_Write(BufferLarge[i]) next i
for i = 0 to BUFFER_SIZE - 1 Mmc_Fat_Read(BufferLarge[i]) next i
file_size = file_size - BUFFER_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
end.
    
```



GO TO

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