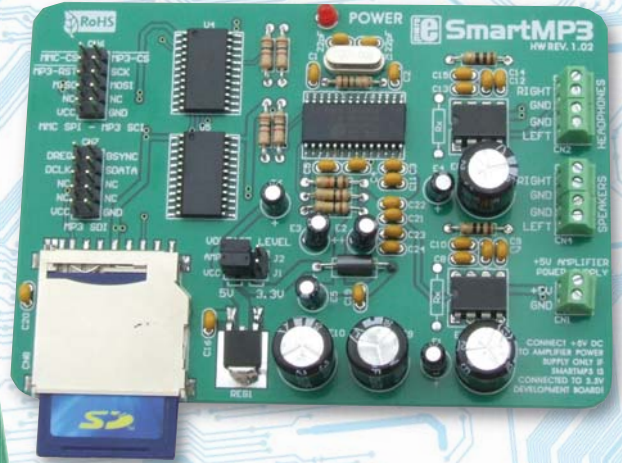


OK. Brauchen Sie einen... MP3 player



Das SmartMP3-Modul und LV24-33A
- Entwicklungssystem

Von Milan Rajic
MikroElektronika Software-Entwicklung

Die Einführung des MP3-Formats verursachte eine Revolution in der Kompressionstechnologie von Audio-Daten. Die so erzeugten Dateien waren bei guter Qualität viel kleiner als bisher. Und wenn man heute MP3-Dateien in Form von Nachrichten oder Musik in eigene Projekte integrieren will, dann ist dies mittlerweile einfach: Man benötigt lediglich eine MMC- oder SD-Speicherkarte, ein paar Chips und etwas Zeit...

Bevor man loslegt, sollte man die Speicherkarte formatieren (FAT16) und anschließend die Datei „save the sound1.mp3“ auf die Karte kopieren.

Die Tonqualität einer MP3-Datei ergibt sich aus der Abtastrate (sampling rate) und der Bitrate. Wie bei der gewöhnlichen Audio-CD sind die Signale von MP3-Dateien meistens mit 44,1 kHz abgetastet. In Sachen Bitrate und Qualität (im Vergleich zum unkomprimierten Signal) gilt folgende Daumenregel: 64 kbit/s reicht für gute Sprachwiedergabe aus, während für Musik 128 kbit/s (und mehr) besser geeignet ist. Die Beispieldatei weist eine Bitrate von 128 kbit/s auf.

Hardware

Die Audiodaten der Beispieldatei sind im MP3-Format codiert, sodass man für die Wiedergabe einen MP3-Decoder benötigt. In unserem Beispiel wird für diesen Zweck der Chip VS1011E eingesetzt. Dieser Chip decodiert nicht nur die MP3-Daten, sondern erledigt auch gleich noch die notwendige Digital/Analog-Umsetzung, sodass an seinem Ausgang lediglich noch ein kleiner Audioverstärker angeschlossen werden

muss, um die Audio-Signale via Lautsprecher hörbar zu machen.

Da FAT16-formatierte Speicherkarten eine Sektorengroße von 512 Byte aufweisen, benötigt man für elegantes Auslesen der Daten und die Übertragung an den MP3-Decoder einen Mikrocontroller mit mindestens 512 Byte RAM. Der eingesetzte Controller PIC24FJ96GA010 bringt hierfür mehr als ausreichende 1.536 Byte RAM mit.

Software

Das Programm erledigt die Audioverarbeitung in fünf Schritten:

- Schritt 1:** Initialisierung des SPI-Moduls des Mikrocontrollers.
- Schritt 2:** Initialisierung der Compiler-Library „Mmc_FAT16“, mit Hilfe derer MP3-Dateien von Medien wie MMC- oder SD-Karten gelesen werden können.
- Schritt 3:** Lesen eines Teils der Datei.
- Schritt 4:** Übertragen von Daten in den Puffer des MP3-Decoders.
- Schritt 5:** Falls noch nicht das Ende der Datei erreicht ist, Sprung zu Schritt 3.

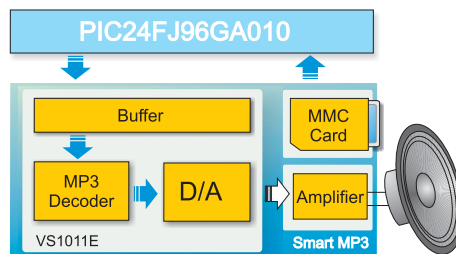


Bild 1. Blockschaltung des an einen PIC24FJ96GA010 angeschlossenen SmartMP3-Moduls

Test

Es ist empfehlenswert, zu Anfang mit einer niedrigen Bitrate zu starten und diese sukzessive zu erhöhen. Der Puffer des MP3-Decoders fasst 2048 Byte. Wenn der Puffer mit MP3-Daten mit der Bitrate von 128 kbit/s geladen wird, enthält er etwa doppelt so viele Samples, wie wenn mit einer Bitrate von 256 kbit/s codiert worden wäre. Folglich dauert es auch etwa doppelt so lange, den

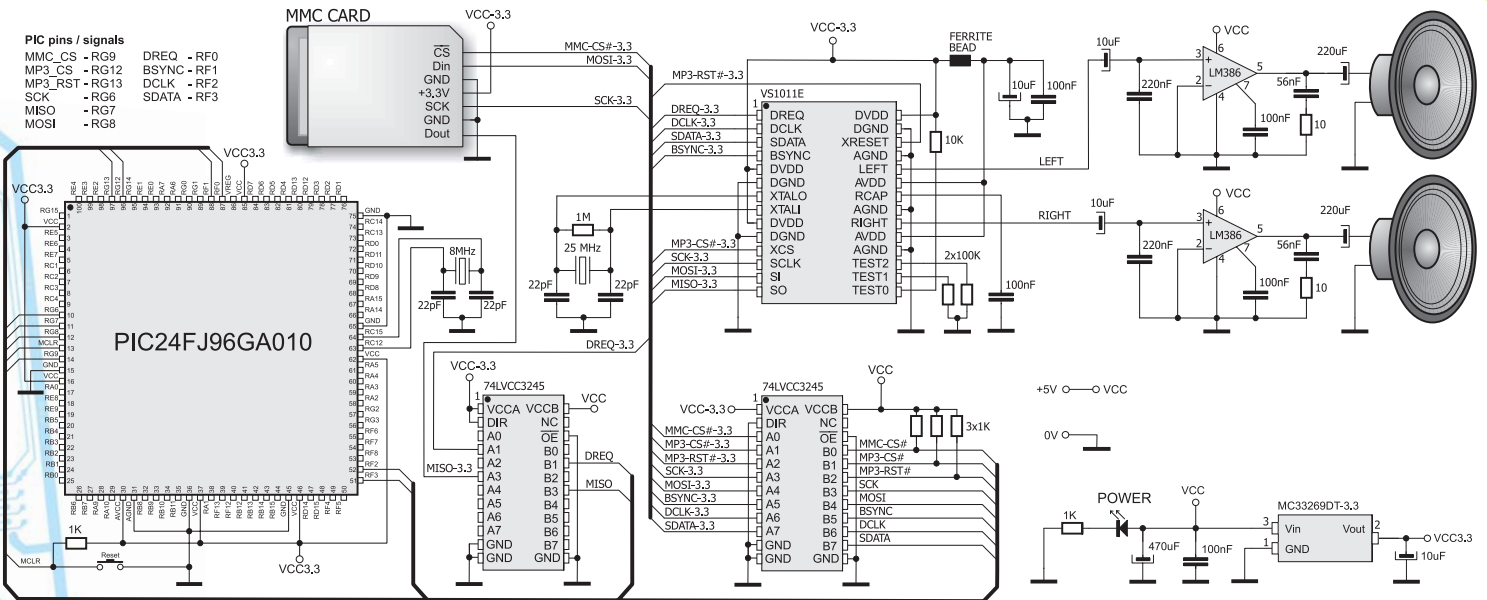


Bild 2. MP3-Lösung aus SmartMP3-Modul, PIC24FJ96GA010und Audioverstärker.

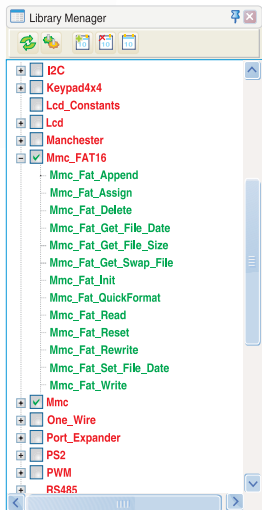
Puffer-Inhalt zu decodieren und auszugeben. Wenn man zu hohe Bitraten verwendet, kann es passieren, dass der MP3-Decoder schon zu früh fertig ist, und der Mikrocontroller mit dem Lesen und Übergeben der Daten nicht nachkommt. In der Folge würde die Tonausgabe gestört klingen. Als Abhilfe kann man entweder mit niedrigeren Bitraten codiertes MP3-Material verwenden oder aber die Taktrate des Mikrocontrollers über die angegebenen 8 MHz (siehe Bild 2) erhöhen.

Man muss sich allerdings keine Sorgen machen: Die Software wurde schon mit einigen Mikrocontroller-Familien mit unterschiedlichen Quarz-Frequenzen getestet und kommt nicht nur mit MP3-Dateien mittlerer, sondern auch höherer Qualität zurecht. Da allerdings bei niedrigen Bitraten mehr Musik im Puffer steckt, kann es passieren, dass der Puffer-Inhalt erst decodiert wird, wenn man ihn ein weiteres Mal füllt. Um dies zu vermeiden, muss die Software überprüfen, ob der Decoder schon für die Aufnahme neuer Daten bereit ist, bevor sie abgeschickt werden. Mit anderen Worten: Der Mikrocontroller muss warten, bis das DREQ-Signal (Data Request) „High“ (logisch „1“) wird.

Erweiterungen

Nach ausgiebigem Testen kann man das Beispiel auch erweitern: Das DREQ-Signal kann periodisch abgefragt werden. Denkbar ist die Erweiterung um eine Routine für Lautstärkeeinstellung und/oder eine eingebaute Bass/Höhen-Anhebung. Die MMC-Library erlaubt auch die Auswahl einer Datei mit anderem Namen. Man kann also beliebige Audio-Nachrichten, Klänge oder Songs erzeugen und an den MP3-Decoder übergeben.

Nachfolgend eine Liste mit den Funktionen, die schon in der Library „Mmc_FAT16“ enthalten sind. Die Library gehört beim Compiler *mikroBASIC for dsPIC* zum Lieferumfang.



Mmc_Fat_Append()	Schreiben ab dem Ende der Datei
Mmc_Fat_Assign()*	Dateizuweisung für FAT-Operationen
Mmc_Fat_Delete()	Datei löschen
Mmc_Fat_Get_File_Date()	Datum und Uhrzeit der Datei lesen
Mmc_Fat_Get_File_Size()	Dateigröße lesen
Mmc_Fat_Get_Swap_File()	Swap-Datei erzeugen
Mmc_Fat_Init()*	Karte für FAT-Operationen initialisieren
Mmc_Fat_QuickFormat()	
Mmc_Fat_Read()*	Daten der Datei lesen
Mmc_Fat_Reset()*	Datei zum Lesen öffnen
Mmc_Fat_Rewrite()	Datei zum Schreiben öffnen
Mmc_Fat_Set_File_Date()	Datum und Uhrzeit der Datei schreiben
Mmc_Fat_Write()	Daten in Datei schreiben

* Mmc_FAT16-Funktionen des Beispiel-Programms

Andere Funktionen von *mikroBASIC for dsPIC* des Beispiel-Programms:

Spi_Init_Advanced() Initialisiere das SPI-Modul des Mikrocontrollers

Beispiel 1: Demonstrationsprogramm für das SmartMP3-Modul.

```

program MP3_Simple_Test

symbol MP3_CS = PORTG.12 symbol MP3_CS_Direction = TRISG.12
symbol MP3_RST = PORTG.13 symbol MP3_RST_Direction = TRISG.13
symbol DREQ = PORTF.0 symbol DREQ_Direction = TRISF.0
symbol BSYNCK = PORTF.1 symbol BSYNCK_Direction = TRISF.1
symbol DCLK = PORTF.2 symbol DCLK_Direction = TRISF.2
symbol SDATA = PORTF.3 symbol SDATA_Direction = TRISF.3
const BUFFER_SIZE = 2048
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)
    BSYNCK = 1
    DCLK = 0 SDATA = data_0 DCLK = 1
    DCLK = 0 SDATA = data_1 DCLK = 1
    BSYNCK = 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
    DCLK = 0
end sub
'Writes one word to MP3 SCI
sub procedure MP3_SCI_Write(dim address as byte, dim data_in as word)
    MP3_CS = 0
    SPI_Write(0x02) SPI_Write(address)
    SPI_Write(Hi(data_in)) SPI_Write(Lo(data_in))
    MP3_CS = 1
    while (DREQ = 0) nop wend
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
    MP3_CS = 0
    SPI_Write(0x03) SPI_Write(start_address)
    for i = 1 to (2*words_count)
        data_buffer[i] = SPI_Read(0)
        Inc(data_buffer)
    next i
    MP3_CS = 1
    while (DREQ = 0) nop wend
datasheet, Serial Protocol for SCI
end sub
sub procedure MP3_SDI_Write(dim data_ as byte)
    while (DREQ = 0) nop wend
datasheet, Serial Protocol for SCI
    SW_SPI_Write(data_)
end sub
sub procedure MP3_SDI_Write_32(dim data_ as ^byte)
    dim i as byte
    while (DREQ = 0) nop wend
datasheet, Serial Protocol for SCI
    for i = 1 to 32
        SW_SPI_Write(data_[i])
    next i
end sub
sub procedure Set_Clock(dim clock_khz_ as word, dim doubler as byte)
    calculate value
    clock_khz = clock_khz / 2
    if (doubler > 0) then clock_khz = clock_khz * 0x8000 end if
    MP3_SDI_Write(0x03, clock_khz)
end sub
sub procedure Soft_Reset()
    MP3_SDI_Write(0x00, 0x0204)
    Delay_us(2)
    while (DREQ = 0) nop wend
datasheet, Serial Protocol for SCI
    for i = 1 to 2048 MP3_SDI_Write(0) next i
end sub
sub procedure Init()
    ADPCFG = 0x0FFF
    DCLK = 0 DCLK_Direction = 0
    SDATA = 0 SDATA_Direction = 0
    MP3_CS = 1 MP3_CS_Direction = 0
    MP3_RST = 1 MP3_RST_Direction = 0
    DREQ_Direction = 1
    BSYNCK = 0 BSYNCK_Direction = 0
end sub
main:
    filename = "sound1.mp3"
    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)
    if (Mmc_Fat_Init(PORTG.9) = 0) then
        if (Mmc_Fat_Assign(filename, 0) <> 0) then
            Mmc_Fat_Reset(filename)
            ' send file blocks to MP3 SDI
            while (file_size > BUFFER_SIZE)
                for i = 0 to BUFFER_SIZE - 1 Mmc_Fat_Read(BufferLarge[i]) next i
                for i = 0 to BUFFER_SIZE/32 - 1 MP3_SDI_Write_32(@BufferLarge + i*32) 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

Download der Source-Codes für dsPIC®, PIC®- und AVR®-Mikrocontroller in den Sprachen C, Basic und Pascal von unserer Webseite: www.mikroe.com/en/article/