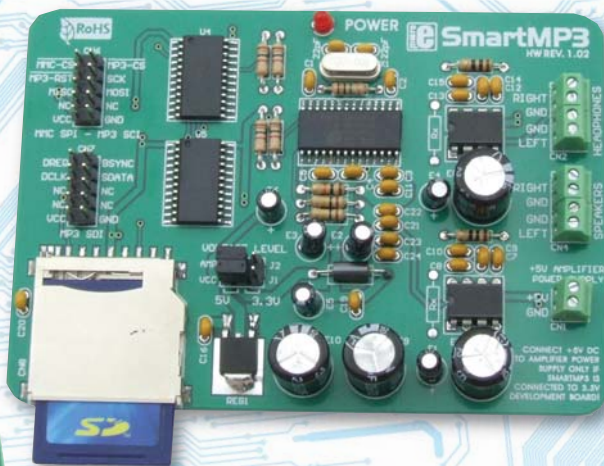


OK. Now you need an ... MP3 player



SmartMP3 module connected to LV24-33A Development System

By Milan Rajic
MikroElektronika - Software Department

The use of MP3 format caused a revolution in digital sound compression technology by enabling audio files to be several times smaller. If you want audio messages or music to be part of your project then you can easily make it true. You just need any standard MMC or SD memory card, a few chips and a little time...

Before we start, it is necessary to format MMC card and save the sound1.mp3 file on it (the card should be formatted in FAT16, i.e. FAT format).

The quality of sound coded in MP3 format depends on sampling rate and bitrate. Similar to an audio CD, most MP3 files are sampled at the frequency of 44.1 kHz. The MP3 file's bitrate indicates the quality of compressed audio comparing to the original uncompressed one, i.e. its fidelity. A bitrate of 64 kbit/s is sufficient for speech reproduction, while it has to be 128 kbit/s or more for music reproduction. In this example a music file with a bitrate of 128 kbit/s is used.

Hardware

The sound contained in this file is coded in the MP3 format so that an MP3 decoder is needed for its decoding. In our example, the VS1011E chip is used for this purpose. This chip decodes MP3 record and performs digital-to-analog conversion of the signal in order to produce

a signal that can be brought to audio speakers over a small audio amplifier. Considering that MMC/SD cards use sections of 512 bytes in size, a microcontroller with 512 byte RAM or more is needed for the purpose of controlling the operation of MP3. We have chosen the PIC24FJ96GA010 with 1536 byte RAM.

Software

The program controlling the operation of this device can be broken up into five steps:

- Step 1:** Initialization of the SPI module of the microcontroller.
- Step 2:** Initialization of the compiler's Mmc_FAT16 library, which enables MP3 files to be read from MMC or SD cards.
- Step 3:** Reading a part of file.
- Step 4:** Sending data to the buffer of MP3 decoder.
- Step 5:** If the end of the file is not reached, jump to step 3.

Testing

It is recommended to start testing device operation with lower bitrate and increase it gradually. The buffer of MP3 decoder is 2048 bytes in size. If the buffer is loaded with a part of MP3 file with 128 kbit/s bitrate, it will contain twice the sound samples than when it is loaded with a part of file with 256 kbit/s bitrate. Accordingly, if the bitrate of the file is lower it will take twice as long to encode the buffer content. If we over increase the bi-

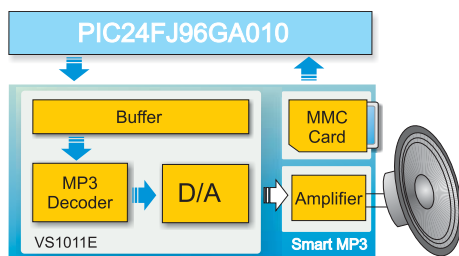
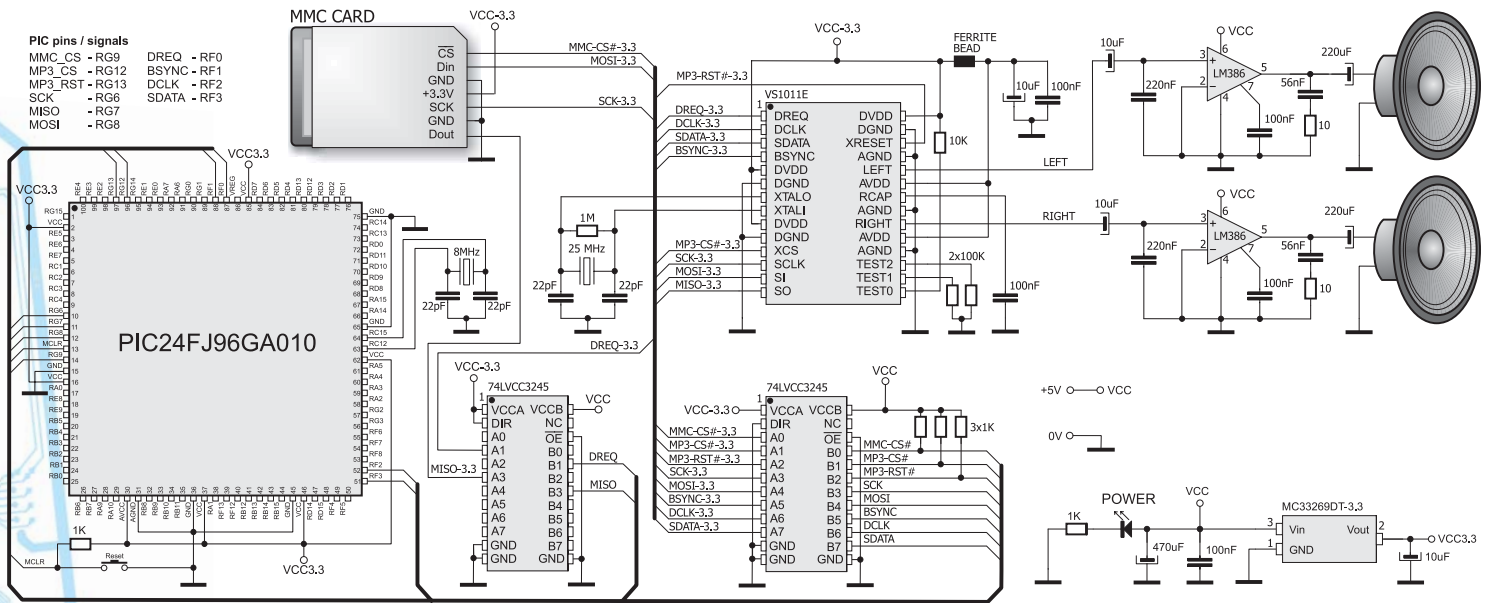


Figure 1. Block diagram of Smart MP3 module connected to a PIC24FJ96GA010



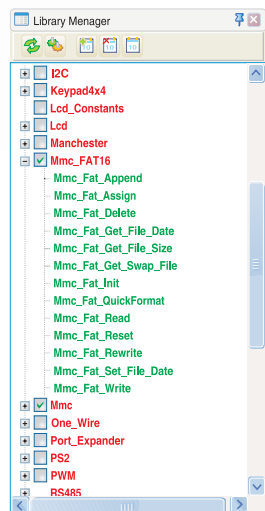
Schematic 1. Connecting the Smart MP3 module to a PIC24FJ96GA010

trate of the file it may happen that buffer content is encoded before the microcontroller manages to read the next part of the file from the card and write it in the buffer, which will cause the sound to be discontinuous. If this happens, we can reduce the MP3 file's bitrate or use a quartz-crystal of frequency higher than 8MHz. Refer to Schematic 1. Anyway, you don't have to worry about this as our program has been tested on several microcontroller families with different crystal values and it is able to decode MP3 files of average and high quality. On the other hand, a low bitrate means that buffer decoder is filled with sound of longer duration. It may happen that decoder doesn't decode the buffer content before we try to reload it. In order to avoid this, it is necessary to make sure that decoder is ready to receive a new data before it has been sent. In other words, it is necessary to wait until decoder's data request signal (DREQ) is set to logic one (1).

Enhancements

This example may also be extended after being tested. The DREQ signal can be periodically tested. A routine for volume control or built-in Bass/Treble enhancer control etc. may be included in the program as well. The MMC library enables us to select a file with different name. In this way it is possible to create a set of MP3 messages, sounds or songs to be used in further/other applications and send appropriate MP3 file to the decoder depending on the needs.

Below is a list of ready to use functions contained in the *Mmc_FAT16 Library*. This library is integrated in *mikroBASIC* for *dsPIC* compiler.



- Mmc_Fat_Append()** Write at the end of the file
- Mmc_Fat_Assign()*** Assign file for FAT operations
- Mmc_Fat_Delete()** Delete file
- Mmc_Fat_Get_File_Date()** Get file date and time
- Mmc_Fat_Get_File_Size()** Get file size
- Mmc_Fat_Get_Swap_File()** Create a swap file
- Mmc_Fat_Init()*** Init card for FAT operations
- Mmc_Fat_QuickFormat()**
- Mmc_Fat_Read()*** Read data from file
- Mmc_Fat_Reset()*** Open file for reading
- Mmc_Fat_Rewrite()** Open file for writing
- Mmc_Fat_Set_File_Date()** Set file date and time
- Mmc_Fat_Write()** Write data to file

*** Mmc_FAT16 functions used in program**

Other *mikroBASIC* for *dsPIC* functions used in program:

- Spi_Init_Advanced()** Initialize microcontroller SPI module

Example 1: Program to demonstrate operation of Smart MP3 module

```

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_32(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)
    clock_khz = clock_khz / 2
    calculate value
    if (doubler > 0) then clock_khz = clock_khz * 0x8000 end if
    MP3_SDI_Write(0x03, clock_khz)
    'Write value to CLOCKF register
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 = 0x7FFF
    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)
IDF)
    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)
            'SW Reset, set clock to 25MHz
            'Call Reset before file reading,
            'procedure returns size of the file
            '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

Code for this example written for dsPIC® microcontrollers in C, Basic and Pascal as well as the programs written for PIC® and AVR® microcontrollers can be found on our web site: www.mikroe.com/en/article/