

OK. Vama je potreban ... MP3 plejer



Milan Rajčić
MikroElektronika - Sektor za razvoj softvera

SmartMP3 modul povezan sa
razvojnim sistemom EasyPIC5

Upotreba MP3 formata je dovela do revolucije u kompresiji digitalnog zvuka omogućujući da audio fajlovi budu i po nekoliko puta manji. Ako želite da u vaš projekat uključite zvučne poruke ili muziku, videćete da je to veoma lako. Potrebna vam je bilo koja standardna MMC ili SD memorijska kartica, nešto hardvera i malo vremena...

Za razvoj softvera i njegovo testiranje korišćeni su razvojni sistem *EasyPIC5* i modul *Smart MP3*. Pre početka rada, potrebno je formatirati MMC karticu i na nju snimiti fajl *sound1.mp3* (kartica se formatira u FAT16, tj. FAT formatu).

Kvalitet reprodukcije zvuka u MP3 formatu zavisi od brzine smplovanja i bitrejt-a. Kao i audio CD, većina MP3 fajlova je smplovana frekvencijom 44.1 kHz. Bitrejt MP3 fajla nam označava koliko je verno kodiran originalni zvuk. Za reprodukciju govora je dovoljno 64 kbit/s dok se za muziku koristi bitrejt 128 kbit/s ili viši. U našem primeru smo koristili muzički fajl sa bitrejtom 128 kbit/s.

Hardver

Zvuk u ovom fajlu je kodiran u MP3 formatu tako da nam je za njegovu reprodukciju potreban MP3 dekodir. U našem primeru, za ovu svrhu se koristi čip VS1011E. Ovo kolo dekoduje MP3 zapisi i vrši digitalno-analognu konverziju signala tako da se nakon obrade dobija signal koji se preko malog audio pojačavača može dovesti na zvučnike.

Zbog činjenice da MMC/SD kartice koriste sektore sa 512 bajtova, za kontrolu rada MP3 dekodera je potreban mikrokontroler sa više od 512 bajtova RAM-a. Mi smo izabrali PIC18F4520 koji ima 1536 bajtova RAM memorije.

Softver

Rad programa koji upravlja ovim uređajem može se podeliti u pet koraka:

- Korak 1:** Inicijalizacija SPI modula mikrokontrolera.
- Korak 2:** Inicijalizacija Mmc_FAT16 biblioteke u kompajleru čime se omogućava

čitanje MP3 fajlova sa MMC ili SD kartica.

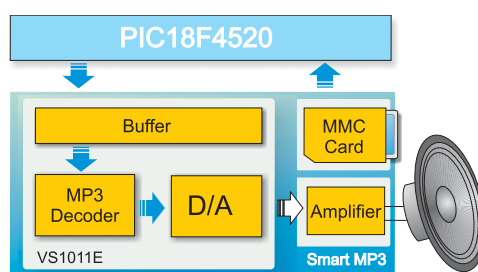
Korak 3: Čitanje dela fajla.

Korak 4: Slanje podataka u bafer MP3 dekodera.

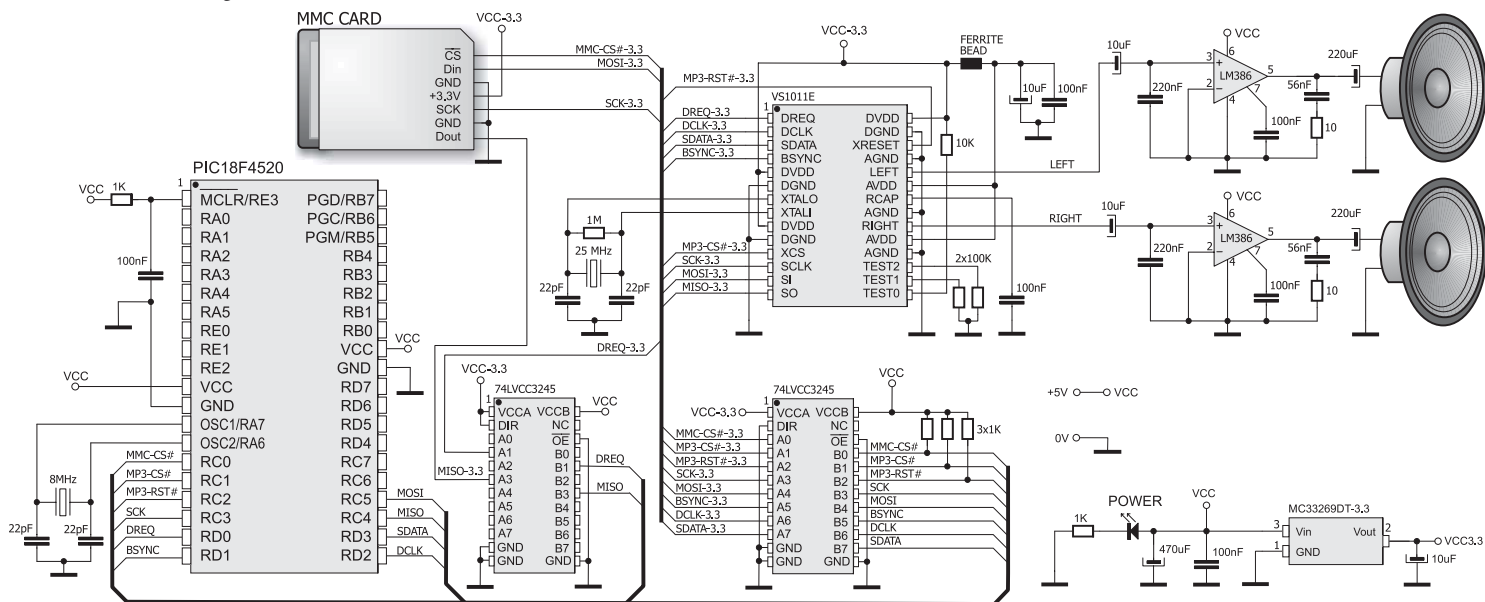
Korak 5: Ako nije završena reprodukcija kompletnog fajla, vraćamo se na korak 3.

Testiranje

Testiranje rada uređaja je bolje početi sa manjim bitrejt faktorom pa ga zatim postepeno povećavati. Naime, bafer MP3 dekodera je veličine 2048 bajtova. Ako taj bafer popunimo delom MP3 fajla bitrejt 128 kbit/s, on će sadržati dva puta više odbiraka zvuka nego kada bismo u njega upisali deo fajla bitrejt 256 kbit/s. Samim tim, zvuk koji je u baferu će se reprodukovati duplo duže kod fajla sa manjim bitrejtom. Ako suviše povećamo bitrejt fajla može se desiti da se zvuk iz bafera reprodukuje pre nego što mikrokontroler stigne da pročita sledeći deo fajla sa kartice i upiše ga u bafer, zbog čega će zvuk biti isprekidan. Ako se to desi, možemo smanjiti bitrejt MP3 fajla



Slika 1. Blok dijagram *Smart MP3* modula sa mikrokontrolerom PIC18F4520



Šema 1. Povezivanje modula Smart MP3 sa mikrokontrolerom PIC18F4520

la koristeći neki od audio editora ili upotrebiti kristal frekvencije više od 8MHz (pogledajte šemu 1).

U svakom slučaju, o ovome ne morate previše brinuti jer je naš program testiran na nekoliko familija mikrokontrolera sa različitim vrednostima kristala i bio je u stanju da reprodukuje MP3 fajlove srednjeg i visokog kvaliteta.

Sa druge strane, mali bitrejt znači da bafer dekodera punimo zvukom dužeg trajanja. Može se desiti da dekokder ne reprodukuje zvuk iz bafera pre nego što mi pokušamo da ga ponovo napunimo. Da bismo to izbegli, pri slanju podataka uvek proveravamo da li je dekokder spreman za prijem novih podataka. Stoga u funkcijama za slanje podataka čekamo da data request signal dekodera (DREQ) bude na logičkoj jedinici (1).

Proširenja programa

Kada isprobamo ovaj primer možemo ga proširiti. DREQ signal se može periodično proveravati. Može se dodati rutina za kontrolu jačine zvuka, za kontrolu ugrađenog Bass/Treble enhancer-a itd. MMC biblioteka nam daje mogućnost da odaberemo fajl sa nekim drugim imenom. Na taj način možemo stvoriti bazu MP3 poruka, zvukova ili pesama koje ćemo koristiti u našoj aplikaciji i poslati odgovarajući MP3 fajl dekokderu u zavisnosti od situacije.

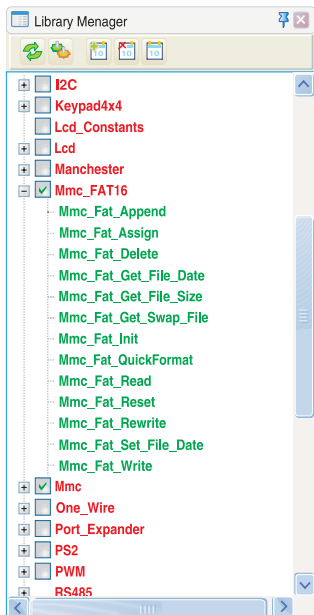
Na donjoj slici je prikazana lista funkcija koje se nalaze u biblioteci *Mmc_FAT16*. Ova biblioteka je sastavni deo kompajlera *mikroPASCAL for PIC* compiler.

Primer 1: Program za demonstraciju rada modula Smart MP3

```

program MP3_Simple_Test;
const BUFFER_SIZE = 512;
var filename: string[13];
    i, file_size: dword;
    data_buffer_32: array[32] of byte;
    BufferLarge: array[BUFFER_SIZE] of byte;
procedure SW_SPI_Write(data_: byte); // Writes one byte to MP3 SDI
begin
    PORTD.1 := 1; // Set BSYNC before sending the first bit
    PORTD.2 := 0; PORTD.3 := data_0; PORTD.2 := 1; // Send data_LSB, data_0
    PORTD.2 := 0; PORTD.3 := data_1; PORTD.2 := 1; // Send data_1
    PORTD.1 := 0; // Clear BSYNC after sending the second bit
    PORTD.2 := 0; PORTD.3 := data_2; PORTD.2 := 1; // Send data_2
    PORTD.2 := 0; PORTD.3 := data_3; PORTD.2 := 1; // Send data_3
    PORTD.2 := 0; PORTD.3 := data_4; PORTD.2 := 1; // Send data_4
    PORTD.2 := 0; PORTD.3 := data_5; PORTD.2 := 1; // Send data_5
    PORTD.2 := 0; PORTD.3 := data_6; PORTD.2 := 1; // Send data_6
    PORTD.2 := 0; PORTD.3 := data_7; PORTD.2 := 1; // Send data_7
    PORTD.2 := 0;
end;
procedure MP3_SCI_Write(address: byte; data_in: word); // Writes one word to MP3 SCI
begin
    PORTC.1 := 0; // select MP3 SCI
    SPI_write(0x02); SPI_write(address);
    SPI_write(Hi(data_in)); SPI_write(Lo(data_in));
    PORTC.1 := 1; // deselect MP3 SCI
    while (PORTD.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
    PORTC.1 := 0; // select MP3 SCI
    SPI_write(0x03); // send READ command
    SPI_write(start_address);
    for i := 1 to (2*words_count) do begin // read words_count words byte per byte
        data_buffer[i] := Spi_Read(0); // read and store a byte
        Inc(data_buffer); // point to next byte
    end;
    PORTC.1 := 1; // deselect MP3 SCI
    while (PORTD.0 = 0) do nop; // wait until DREQ becomes 1, see MP3 codec datasheet, Serial
    Protocol for SCI
end;
procedure MP3_SDI_Write(data_: byte); // Write one byte to MP3 SDI
begin
    while (PORTD.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 (PORTD.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); // Set clock
begin
    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 CLOCKS register
end;
procedure Soft_Reset(); begin // Software Reset
    MP3_SCI_Write(0x00, 0x0204); // Set SW_RESET bit and SM_BITORDR bit (bitorder is LSB first)
    Delay_us(2); // Required, see MP3 codec datasheet -> Software Reset
    while (PORTD.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
    ADCON1 := ADCON1 or 0x0F; // Configure AN pins as digital
    CMCON := CMCON or 7; // Disable comparators
    TRISD.2 := 0; TRISD.3 := 0; // Set SW_SPI pin directions to output
    PORTD.2 := 0; PORTD.3 := 0; // Clear SW_SPI SCK and SDO
    TRISC.1 := 0; PORTC.1 := 1; // Configure MP3_CS as output and deselect MP3_CS
    TRISC.2 := 0; PORTC.2 := 1; // Configure MP3_RST as output and set MP3_RST pin
    TRISD.0 := 1; // Configure DREQ as input
    TRISD.1 := 0; PORTD.1 := 0; // Configure BSYNC as output and clear BSYNC
end;
begin // main function
    filename := 'sound1.mp3'; // Set File name
    Init();
    SPI_init_advanced(MASTER_OSC_DIV64, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH);
    Soft_Reset(); Set_Clock(25000, 0); // SW_Reset, set clock to 25MHz, do not use clock doubler
    if (Mmc_Fat_Init(PORTC, 0) = 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]));
            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;

```



Mmc_Fat_Append()	Dodaj podatke na kraj fajla
Mmc_Fat_Assign()*	Pripremi fajl za FAT operacije
Mmc_Fat_Delete()	Obriši fajl
Mmc_Fat_Get_File_Date()	Vрати datum i vreme fajla
Mmc_Fat_Get_File_Size()	Vрати veličinu fajla
Mmc_Fat_Get_Swap_File()	Napravi swap fajl
Mmc_Fat_Init()*	Inicijalizuj karticu za FAT operacije
Mmc_Fat_QuickFormat()	Formatiraj karticu
Mmc_Fat_Read()*	Pročitaj podatke iz fajla
Mmc_Fat_Reset()*	Pripremi fajl za čitanje
Mmc_Fat_Rewrite()	Pripremi fajl za pisanje
Mmc_Fat_Set_File_Date()	Setuj datum i vreme fajla
Mmc_Fat_Write()	Upiši podatke u fajl

Ostale funkcije kompajlera *mikroPASCAL for PIC* korišćene u programu:

Spi_Init_Advanced() Inicijalizuj SPI modul mikrokontrolera

Pored proširene verzije ovog programa koji je napisan za PIC® mikrokontrolere, sa našeg sajta možete preuzeti i verzije pisane za dsPIC® i AVR® mikrokontrolere www.mikroe.com/en/article/

