

# OK. Vama je potreban ... MP3 plejer



SmartMP3 modul sa razvojnim sistemom BIGAVR2

Milan Rajić  
MikroElektronika - Sektor za razvoj softvera

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 *BIGAVR2* 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 semplovanja i bitrejt-a. Kao i audio CD, većina MP3 fajlova je semplovana 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 dekodera. U našem primeru, za ovu svrhu se koristi čip VS1011E. Ovo kolo dekoduje MP3 zapis 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 Atmega128 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.

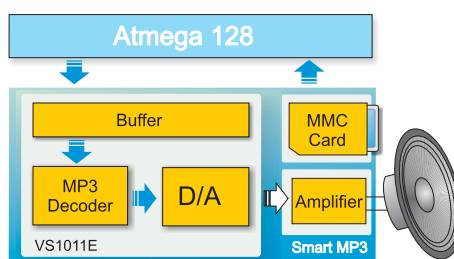
**Korak 3:** Čitanje dela fajla.

**Korak 4:** Slanje podataka u bafer MP3 dekodera.

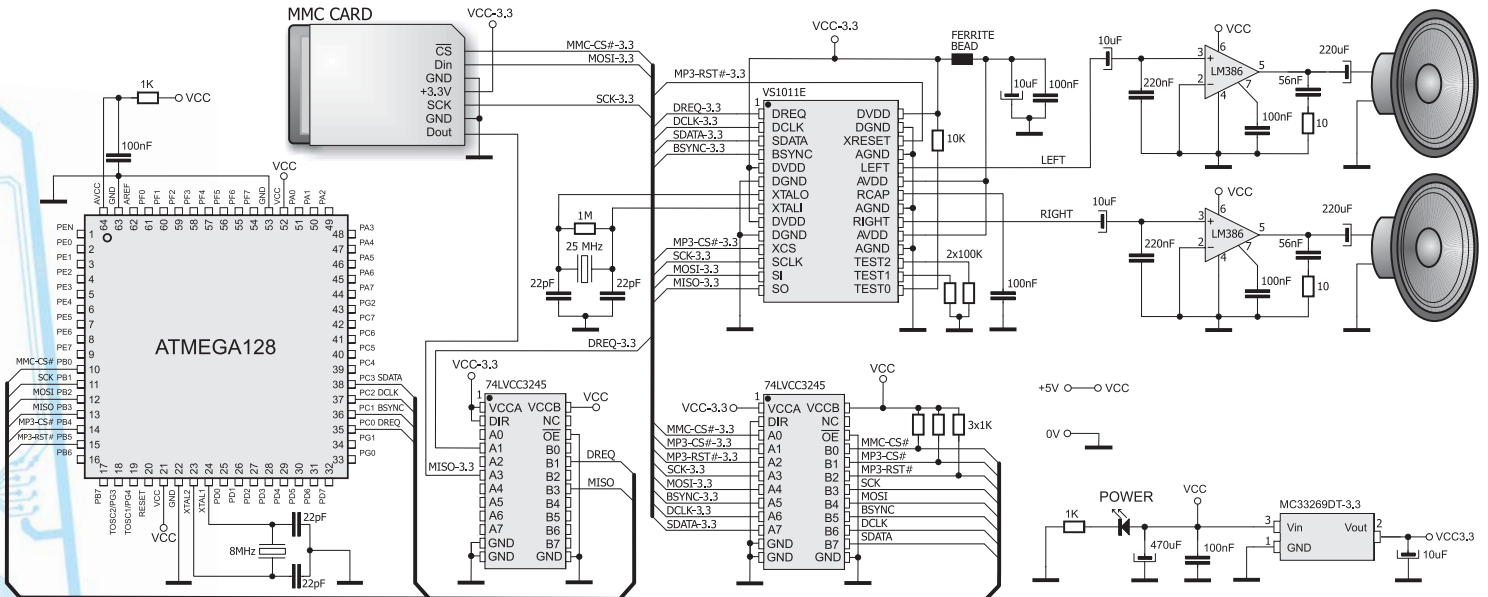
**Korak 5:** Ako nije završena reprodukcija kompletnog fajla, vraćamo se na korak 3.

## 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 Atmega128



Šema 1. Povezivanje modula Smart MP3 sa mikrokontrolerom Atmega128

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

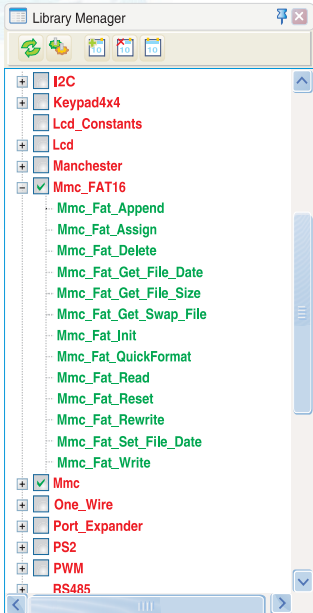
U svakom slučaju, o ovome ne morate previše brinuti jer je naš program testiran na nekoliko familija mikrokontrolera sa različitim vrednostima kristala i bio je u stanju da reprodukuje MP3 fajlove srednjeg i visokog kvaliteta.

Sa druge strane, mali bitrejt znači da bafer dekodera punimo zvukom dužeg trajanja. Može se desiti da dekodera 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 dekodera 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 dekodera 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 PRO for AVR* compiler.



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 PRO for AVR* korišćene u programu:

Spi\_Init\_Advanced() Inicijalizuj SPI modul mikrokontrolera

### Primer 1: Program za demonstraciju rada modula Smart MP3

```

program MP3_Simple_Test;
// Smart MP3 board connections
var Mmc_Chip_Select : sbit at PORTB.0; var Mmc_Chip_Select_Direction : sbit at DDRB.0;
var MP3_CS : sbit at PORTB.4; var MP3_CS_Direction : sbit at DDRB.4;
var MP3_RST : sbit at PORTB.5; var MP3_RST_Direction : sbit at DDRB.5;
var DREQ : sbit at PINC.0; var DREQ_Direction : sbit at DDRC.0;
var BSYN : sbit at PORTC.1; var BSYN_Direction : sbit at DDRC.1;
var DCLK : sbit at PORTC.2; var DCLK_Direction : sbit at DDRC.2;
var SDATA : sbit at PORTC.3; var SDATA_Direction : sbit at DDRC.3;
const BUFFER_SIZE = 512; // global variables
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); begin // Writes one byte to MP3 SDI
    BSYN := 1; // Set BSYN before sending the first bit
    DCLK := 0; SDATA := data_0; DCLK := 1; // bitorder is LSB first
    DCLK := 0; SDATA := data_1; DCLK := 1;
    BSYN := 0; // Clear BSYN after sending the second bit
    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;
procedure MP3_SCI_Write(address : byte; data_in : word); begin // Writes one word to MP3 SCI
    MP3_CS := 0; // select MP3 SCI
    SPI1_Write(0x02); SPI1_Write(address); // Read command, send address
    SPI1_Write(Hi(data_in)); SPI1_Write(Lo(data_in)); // Send high byte, send low byte
    MP3_CS := 1; // deselect MP3 SCI
    while (DREQ = 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
    MP3_CS := 0; // select MP3 SCI
    SPI1_Write(0x03); // Read command
    SPI1_Write(start_address);
    for i := 1 to (2*words_count) do begin // read words_count words byte per byte
        data_buffer[i] := SPI1_Read(0); // read and store a byte
        Inc(data_buffer); // point to next byte
    end;
    MP3_CS := 1; // deselect MP3 SCI
    while (DREQ = 0) do nop; // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
end;
procedure MP3_SDI_Write(data : ^byte); begin // Write one byte to MP3 SDI
    while (DREQ = 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 (DREQ = 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
        Inc(data);
    end;
end;
procedure Set_Clock(clock_khz : word; doubler : byte); begin // Set clock
    clock_khz := clock_khz / 2; // calculate value
    if (doubler > 0) then clock_khz := clock_khz * 0x8000; // Write value to CLOCKF register
    MP3_SCI_Write(0x03, clock_khz);
end;
procedure Soft_Reset; begin // 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) 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
    DCLK_Direction := 1; DCLK := 0; // Clear SW SPI_SCK, configure pin as output
    SDATA_Direction := 1; SDATA := 0; // Clear SW SPI_SDA, configure pin as output
    MP3_CS_Direction := 1; MP3_CS := 1; // Deselect MP3_CS, configure pin as output
    MP3_RST_Direction := 1; MP3_RST := 1; // Set MP3_RST pin, configure pin as output
    DREQ_Direction := 0; // Configure DREQ as input
    BSYN_Direction := 1; BSYN := 0; // Clear BSYN, configure pin as output
end;
begin
    // main function
    filename := 'sound1.mp3'; Init(); // Set File name, call Init
    Spi_Init_Advanced(SPI_MASTER, SPI_FCY_DIV128, SPI_CLK_LO_LOADING);
    Spi_Read_Prv := @SPI1_Read;
    Soft_Reset(); Set_Clock(25000.0); // SW Reset, set clock to 25MHz
    if (Mmc_Fat_Init() = 0) then begin
        SPI1_Init_Advanced(SPI_MASTER, SPI_FCY_DIV2, SPI_CLK_LO_LOADING); // reinic spi at higher speed
        if (Mmc_Fat_Assign(filename, 0) <> 0) then begin
            Mmc_Fat_Reset(file_size); // Call Reset before file reading
            while (file_size > BUFFER_SIZE) do begin // send file blocks to MP3 SDI
                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);
                file_size := file_size - BUFFER_SIZE;
            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;
end;

```

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

Napisano u kompajleru mikroPASCAL PRO for AVR