

# 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 kodo 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.

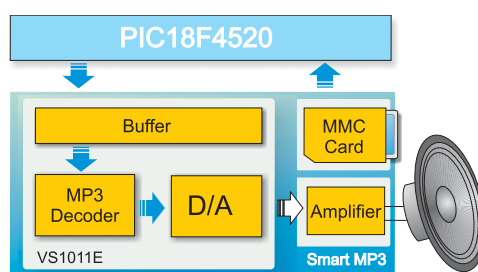
**Korak 3:** Čitanje dela fajla.

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

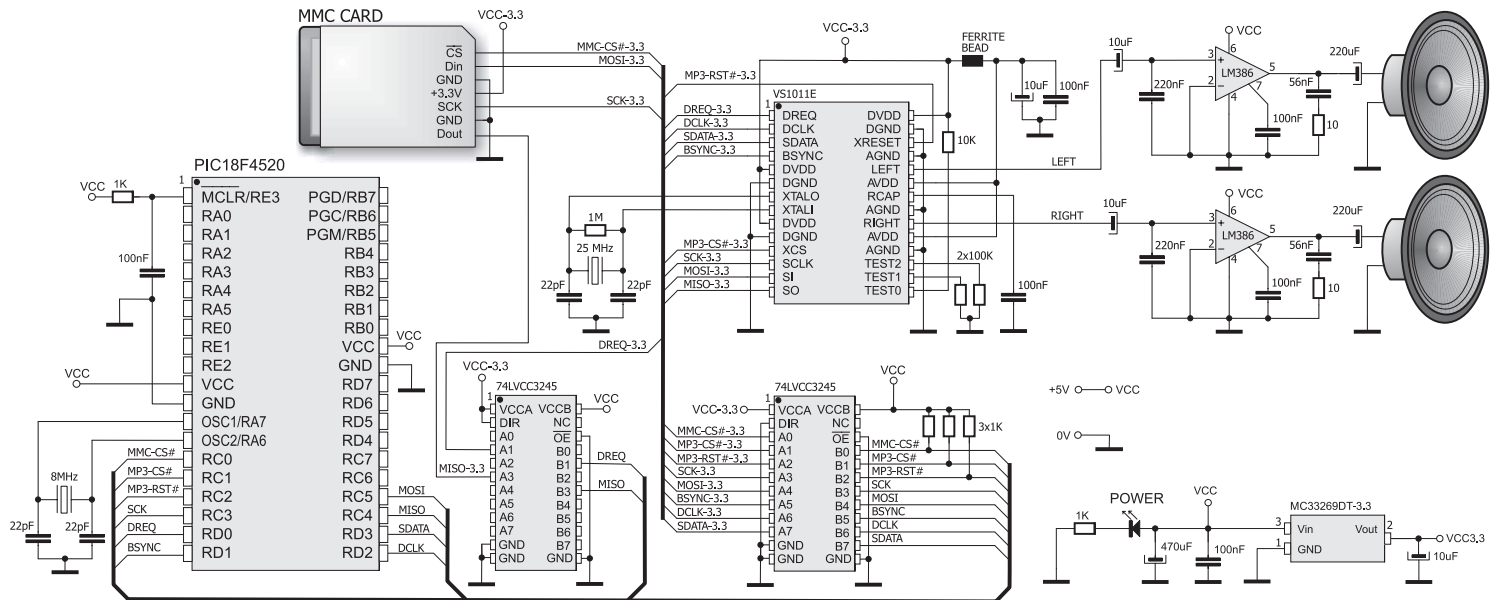
**Korak 5:** Ako nije završena reprodukcija kompletnog fajla, vraćamo se na korak 3.

## 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 isprekidano. 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

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

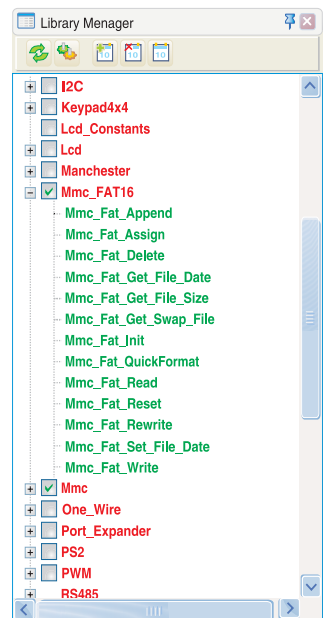
Kada isprobamo ovaj primer možemo ga proširiti. DREQ signal se može periodično proveravati. Može se dodati rutina za kontrolu jačine zvuka, za kontrolu ugrađenog Bass/Treble enhancer-a itd. MMC biblioteka nam daje mogućnost da odaberemo fajl sa nekim drugim imenom. Na taj način možemo stvoriti bazu MP3 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 *mikroC PRO for PIC* compiler.

Primer 1: Program za demonstraciju rada modula Smart MP3

```

char filename[14] = "sound1.mp3"; // Set File name
unsigned long i, file_size;
const BUFFER_SIZE = 512;
char data_buffer_32[32], BufferLarge[BUFFER_SIZE];
char Mmc_Chip_Select_at RC0_bit;
sbit Mmc_Chip_Select_Direction at TRISC0_bit;
// Writes one byte to MP3 SDI
void SW_SPI_Write(unsigned data_) {
RD1_bit = 1; // Set BSYNC before sending the first bit
RD2_bit = 0; RD3_bit = data_; RD2_bit = 1; data_ >>= 1; // Send data_LSB, data_0
RD2_bit = 0; RD3_bit = data_; RD2_bit = 1; // Send data_1
RD1_bit = 0; // Clear BSYNC after sending the second bit
RD2_bit = 0; RD3_bit = data_; RD2_bit = 1; data_ >>= 1; // Send data_2
RD2_bit = 0; RD3_bit = data_; RD2_bit = 1; data_ >>= 1; // Send data_3
RD2_bit = 0; RD3_bit = data_; RD2_bit = 1; data_ >>= 1; // Send data_4
RD2_bit = 0; RD3_bit = data_; RD2_bit = 1; data_ >>= 1; // Send data_5
RD2_bit = 0; RD3_bit = data_; RD2_bit = 1; data_ >>= 1; // Send data_6
RD2_bit = 0; RD3_bit = data_; RD2_bit = 1; data_ >>= 1; // Send data_7
RD2_bit = 0;
}
// Writes one word to MP3 SCI
void MP3_SCI_Write(char address, unsigned int data_in) {
RC1_bit = 0; // select MP3 SCI
SPI1_Write(0x02); // send WRITE command
SPI1_Write(address); // Send High byte
SPI1_Write(data_in >> 8); // Send Low byte
SPI1_Write(data_in); // deselect MP3 SCI
RC1_bit = 1;
Delay_us(5); // Required, see VS1001k datasheet chapter 5.4.1
}
// Reads words_count words from MP3 SCI
void MP3_SCI_Read(char start_address, char words_count, unsigned int *data_buffer) {
unsigned int temp;
RC1_bit = 0; // select MP3 SCI
SPI1_Write(0x03); // send READ command
SPI1_Write(start_address);
while (words_count--) {
temp = SPI1_Read(0);
temp <<= 8;
temp += SPI1_Read(0);
*(data_buffer++) = temp;
}
RC1_bit = 1;
Delay_us(5); // Required, see VS1001k datasheet chapter 5.4.1
}
// Write one byte to MP3 SDI
void MP3_SDI_Write(char data_) {
while (RD0_bit == 0); // wait until DREQ becomes 1
SW_SPI_Write(data_);
}
// Write 32 bytes to MP3 SDI
void MP3_SDI_Write_32(char *data_) {
char i;
while (RD0_bit == 0); // wait until DREQ becomes 1
for (i=0; i<32; i++) SW_SPI_Write(data_[i]);
}
// Set clock
void Set_Clock(unsigned int clock_khz, char doubler) {
clock_khz /= 2; // calculate value
if (doubler) clock_khz |= 0x8000; // Write value to CLOCKF register
MP3_SCI_Write(0x03, clock_khz);
}
void Init() {
ADCON1 = 0x0F; // set all AN pins to digital
RD2_bit = 0; RD3_bit = 0; // Clear SW SPI SCK and SDO
TRISD2_bit = 0; TRISD3_bit = 0; // Set SW SPI pin directions
RC1_bit = 1; // Deselect MP3_CS
TRISC1_bit = 0; // Configure MP3_CS as output
RC2_bit = 1; // Set MP3_RST pin
TRISC2_bit = 0; // Configure MP3_RST as output
TRISD0_bit = 1; // Configure DREQ as input
RD1_bit = 0; // Clear BSYNC
TRISD1_bit = 0; // Configure BSYNC as output
}
// Software Reset
void Soft_Reset() {
MP3_SCI_Write(0x00, 0x0204); // Write to MODE register: set SM_RESET bit and SM_BITORD bit
Delay_us(2); // Required, see VS1001k datasheet chapter 7.4
while (RD0_bit == 0); // wait until DREQ becomes 1
for (i=0; i<2048; i++) MP3_SDI_Write(0); // feed 2048 zeros to the MP3 SDI bus;
}
void main() {
// main function
Init();
SPI1_Init_Advanced(MASTER_OSC_DIV64, DATA_SAMPLE_MIDDLE, CLK_IDLE_LOW, LOW_2_HIGH);
Spi_Rd_Pt = SPI1_Read;
Set_Clock(25000, 0); // Set clock to 25MHz, do not use clock doubler
Soft_Reset(); // SW Reset
if (Mmc_Fat_Init() == 0) {
if (Mmc_Fat_Assign(&filename, 0)) { // Assign file "sound1.mp3"
Mmc_Fat_Reset(&file_size); // Call Reset before file reading
while (file_size > BUFFER_SIZE) { // Send file blocks to MP3 SDI
for (i=0; i<BUFFER_SIZE; i++)
Mmc_Fat_Read(BufferLarge + i); // Read file block
for (i=0; i<BUFFER_SIZE/32; i++)
MP3_SDI_Write_32(BufferLarge + i*32); // Send file block to mp3 decoder
file_size -= BUFFER_SIZE; // Decrease file size
}
for (i=0; i<file_size; i++)
Mmc_Fat_Read(BufferLarge + i); // Send the rest of the file
for (i=0; i<file_size; i++)
MP3_SDI_Write(BufferLarge[i]);
}
}
}
    
```



Mmc_Fat_Append()	Dodaj podatke na kraj fajla
Mmc_Fat_Assign()*	Pripremi fajl za FAT operacije
Mmc_Fat_Delete()	Obrisi fajl
Mmc_Fat_Get_File_Date()	Vrati datum i vreme fajla
Mmc_Fat_Get_File_Size()	Vrati 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

\* Mmc\_FAT16 funkcije korišćene u programu

Ostale funkcije kompajlera *mikroC PRO 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/](http://www.mikroe.com/en/article/)

