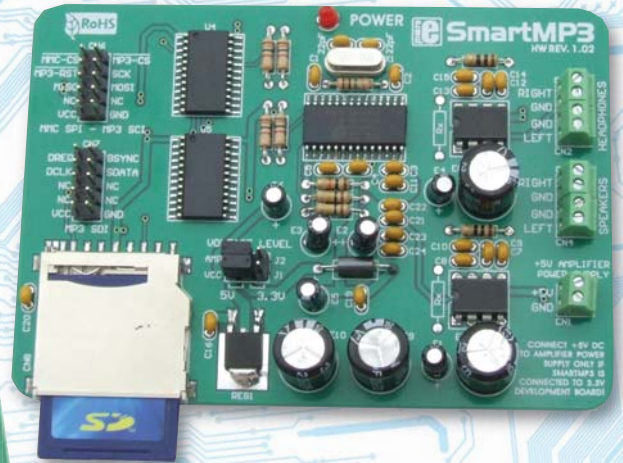


# OK. Vama je potreban ... MP3 plejer



SmartMP3 modul povezan sa razvojnim sistemom LV24-33A

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 LV24-33A 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

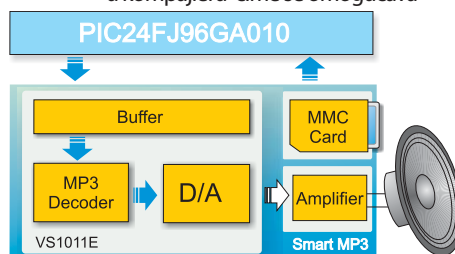
Zvuk u ovom fajlu je kodiran u MP3 formatu tako da nam je za njegovu reprodukciju potreban MP3 dekodir. U našem primeru, za ovu svrhu se koristi čip VS1011E. Ovo kolo dekoduje MP3 zapis i vrši digitalno-analognu konverziju signala tako da se nakon obrade dobija signal koji se preko malog audio pojačavača 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 PIC24FJ96GA010 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



Slika 1. Blok dijagram Smart MP3 modula sa mikrokontrolerom PIC24FJ96GA010

čitanje MP3 fajlova sa MMC ili SD kartica.

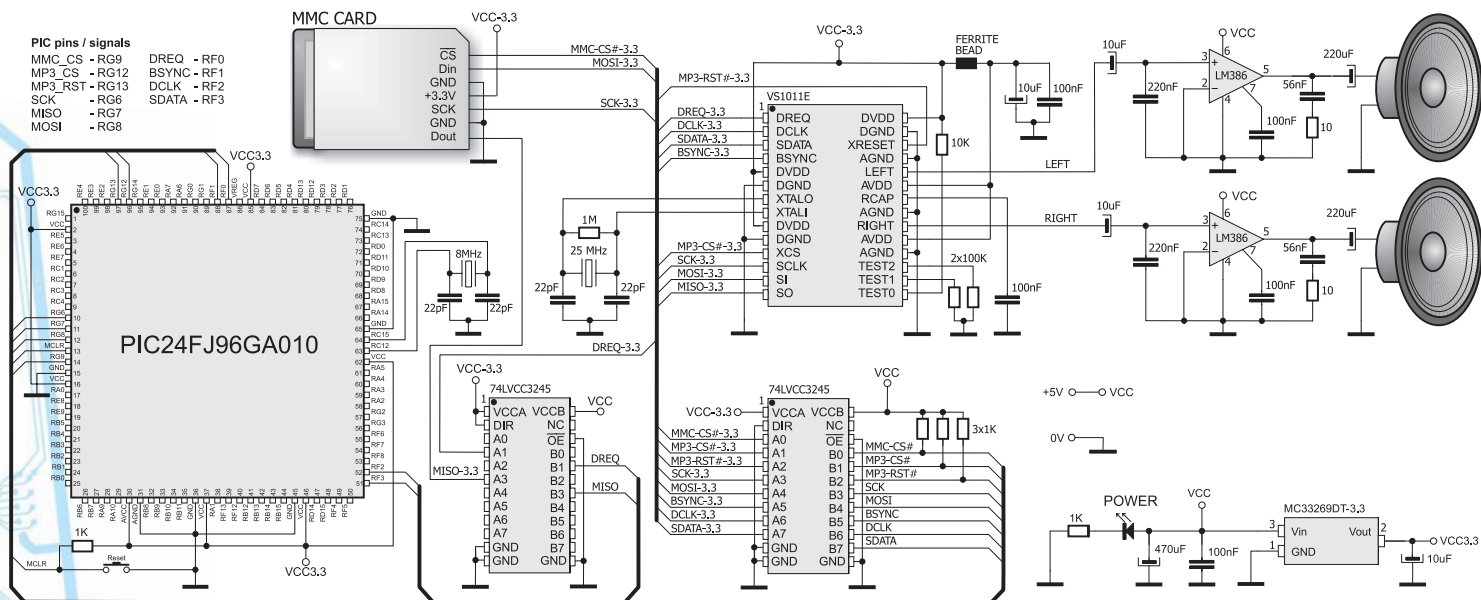
**Korak 3:** Čitanje dela fajla.

**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



Šema 1. Povezivanje modula Smart MP3 sa mikrokontrolerom PIC24FJ96GA010

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

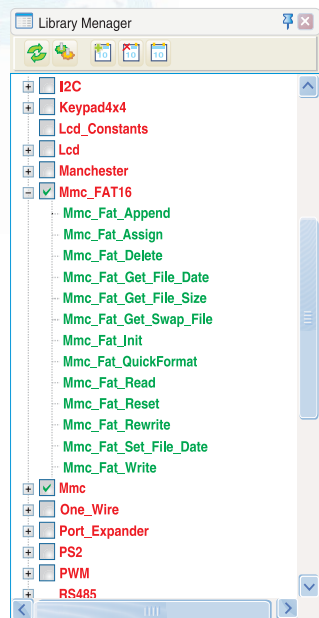
U svakom slučaju, o ovome ne morate previše brinuti jer je naš program testiran na nekoliko familija mikrokontrolera sa različitim vrednostima kristala i bio je u stanju da reprodukuje MP3 fajlove srednjeg i visokog kvaliteta.

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 dsPIC* 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()	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 dsPIC* korišćene u programu:

Spi\_Init\_Advanced() Inicijalizuj SPI modul mikrokontrolera

Primer 1: Program za demonstraciju rada modula Smart MP3

```
#include <built_in.h>
#include <spi_const.h>
#define MP3_CS PORTG.F12 // Smart MP3 board connections
#define MP3_RST PORTG.F13
#define DREQ PORTF.F0
#define BSYNC PORTF.F1
#define DCLK PORTF.F2
#define SDATA PORTF.F3
#define MP3_CS_Direction TRISG.F12
#define MP3_RST_Direction TRISG.F13
#define DREQ_Direction TRISF.F0
#define BSYNC_Direction TRISF.F1
#define DCLK_Direction TRISF.F2
#define SDATA_Direction TRISF.F3
char filename[14] = "sound1.mp3"; // global variables
unsigned long i, file_size;
char data_buffer[32];
const BUFFER_SIZE = 2048;
char BufferLarge[BUFFER_SIZE];
void SW_SPI_Write(char data) { // Writes one byte to MP3 SDI
    BSYNC = 1; // Set BSYNC before sending the first bit
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1; // bitorder is LSB first
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    BSYNC = 0; // Clear BSYNC after sending the second bit
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
    DCLK = 0; SDATA = data; DCLK = 1; data >>= 1;
}
void MP3_SCI_Write(char address, unsigned int data_in) { // Writes one word to MP3 SCI
    MP3_CS = 0; // select MP3 SCI
    Spi2_Write(0x02); Spi2_Write(address); // send WRITE command, send address
    Spi2_Write(Hib(data_in)); Spi2_Write(Lob(data_in)); // Send High byte, send Low byte
    MP3_CS = 1; // deselect MP3 SCI
    while (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
}
// Reads words_count words from MP3 SCI
void MP3_SCI_Read(char start_address, char words_count, unsigned int *data_buffer) {
    unsigned int temp;
    MP3_CS = 0; // select MP3 SCI
    Spi2_Write(0x03); Spi2_Write(start_address); // send READ command, send address
    while (words_count--) { // read words_count words byte per byte
        temp = Spi2_Read(0);
        temp <<= 8;
        *(data_buffer++) = temp;
    }
    MP3_CS = 1; // deselect MP3 SCI
    while (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
}
void MP3_SDI_Write(char data) { // Write one byte to MP3 SDI
    while (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
    SW_SPI_Write(data);
}
void MP3_SDI_Write_32(char *data) { // Write 32 bytes to MP3 SDI
    char i;
    while (DREQ == 0); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
    for (i=0; i<32; i++) SW_SPI_Write(data[i]);
}
void Set_Clock(unsigned int clock_khz, char doubler) { // Set clock
    clock_khz = 2; // calculate value
    if (doubler) clock_khz = 0x8000;
    MP3_SCI_Write(0x03, clock_khz); // Write value to CLOCKF register
}
void Soft_Reset() { // 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); // wait until DREQ becomes 1, see MP3 codec datasheet, Serial Protocol for SCI
    for (i=0; i<2048; i++) MP3_SDI_Write(0); // feed 2048 zeros to the MP3 SDI bus
}
void Init() {
    ADPCFG = 0xFFFF; // set all AN pins to digital
    DCLK = 0; DCLK_Direction = 0; // Clear SW SPI SCK, configure pin as output
    SDATA = 0; SDATA_Direction = 0; // Clear SW SPI SDA, configure pin as output
    MP3_CS = 1; MP3_CS_Direction = 0; // Deselect MP3_CS, configure pin as output
    MP3_RST = 1; MP3_RST_Direction = 0; // Set MP3_RST pin, configure pin as output
    DREQ_Direction = 1; // Configure DREQ as input
    BSYNC = 0; BSYNC_Direction = 0; // Clear BSYNC, configure pin as output
}
void main() { // main function
    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_IDL);
    Soft_Reset(); Set_Clock(25000, 0); // SW Reset, set clock to 25MHz
    if (Mmc_Fat_Init(&PORTG, 9)) { // Mmc_Fat_Assign(&filename, 0) {
        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);
            for (i=0; i<BUFFER_SIZE/32; i++) MP3_SDI_Write_32(BufferLarge + i*32);
            file_size -= BUFFER_SIZE;
        }
        // send the rest of the file to MP3 SDI
        for (i=0; i<file_size; i++) Mmc_Fat_Read(BufferLarge + i);
        for (i=0; i<file_size; i++) MP3_SDI_Write(BufferLarge[i]);
    }
}
```

Napisano u kompajleru mikroC PRO for dsPIC