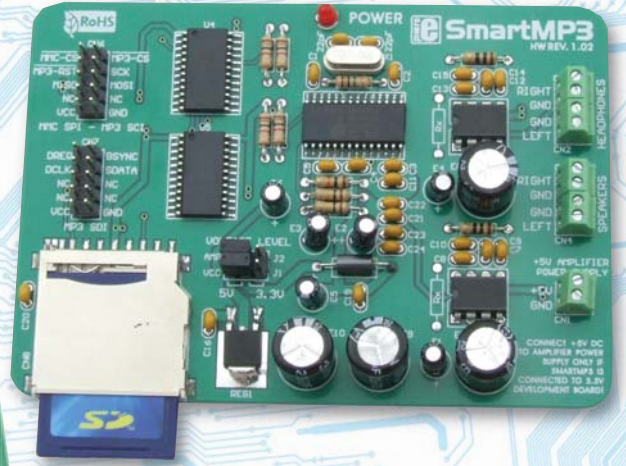


OK. Ahora necesita un ... Reproductor MP3



modulo Smart MP3 y el sistema de desarrollo LV24-33A

Por Milan Rajic
MikroElektronika – Departamento de Software

La adopción del formato MP3, provocó una revolución en la tecnología de compresión del audio digital, permitiendo que los ficheros de audio llegasen a ser mucho más pequeños. Si deseamos que mensajes de audio o de música formen parte de nuestros proyectos, podemos conseguirlo fácilmente. Tan solo necesitamos una memoria estándar MMC o SD, unos pocos componentes y un poco de tiempo...

Antes de comenzar, es necesario formatear la tarjeta MMC y salvar el fichero sound1.mp3 en ella (la tarjeta debe estar formateada en FAT16, es decir, formato FAT). La calidad del sonido codificado en formato MP3 depende de la velocidad de muestreo y la velocidad de datos. Al igual que en un CD de audio, la mayoría de los ficheros MP3 son muestreados a 44,1 kHz. La velocidad de datos del fichero MP3 indica la calidad del audio comprimido comparado con el audio original no comprimido, es decir, su fidelidad. Una velocidad de datos de 64 kbit/s es suficiente para la voz hablada, mientras que para la reproducción de música, esta velocidad debe ser, como mínimo, de 128 kbit/s. En este ejemplo se ha usado un fichero de música con una velocidad de datos de 128 kbit/s.

El Circuito

El sonido contenido en este fichero está codificado en formato MP3, por lo que se necesita un decodificador MP3 para su descodificación. En nuestro ejemplo hemos usado el circuito integrado VS1011E para este propósito. Este circuito integrado descodifica grabaciones MP3 y realiza la conversión Analógico/Digital de la señal para producir una señal que puede ser llevada a un altavoz, a través de un pequeño

amplificador de audio. Considerando que las tarjetas MMC/SD usan secciones de 512 bytes de tamaño, se necesita un microcontrolador con 512 byte de memoria RAM o más para poder controlar el proceso de descodificación MP3. Hemos elegido el microcontrolador PIC24FJ96GA010 con 1536 byte de memoria RAM.

El Programa

El programa que controla las operaciones de este dispositivo está dividido en cinco fases o pasos:

Paso 1: Inicialización del modulo SPI del microcontrolador.

Paso 2: Inicialización de la librería *Mmc_FAT16* del compilador, que permite que los ficheros MP3 puedan ser leídos desde las tarjetas MMC o SD.

Paso 3: Lectura de una parte del fichero.

Paso 4: Envío de los datos al "buffer" del decodificador MP3.

Paso 5: Si no se ha alcanzado aún el final del fichero, volver al paso 3.

Pruebas

Se recomienda comenzar las pruebas del dispositivo con una velocidad de datos baja e incrementarla de forma gradual. El "buffer" del decodificador MP3 tiene un tamaño de 2048 bytes. Si el buffer se carga con parte del fichero MP3 a una velocidad de 128 kbit/s, contendrá dos veces el número de muestras de sonido que cuando ha sido cargado con una parte de un fichero con una velocidad de datos de 256 kbit/s. De acuerdo con esto, si la velocidad de datos del fichero es inferior, tardará el doble en codificar el contenido del buffer. Si sobrepasamos la velocidad de datos del fichero

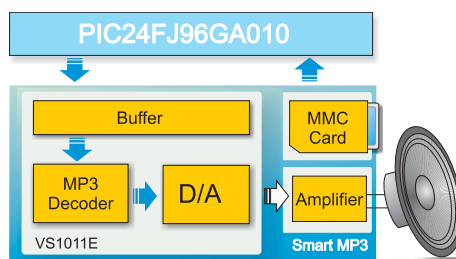
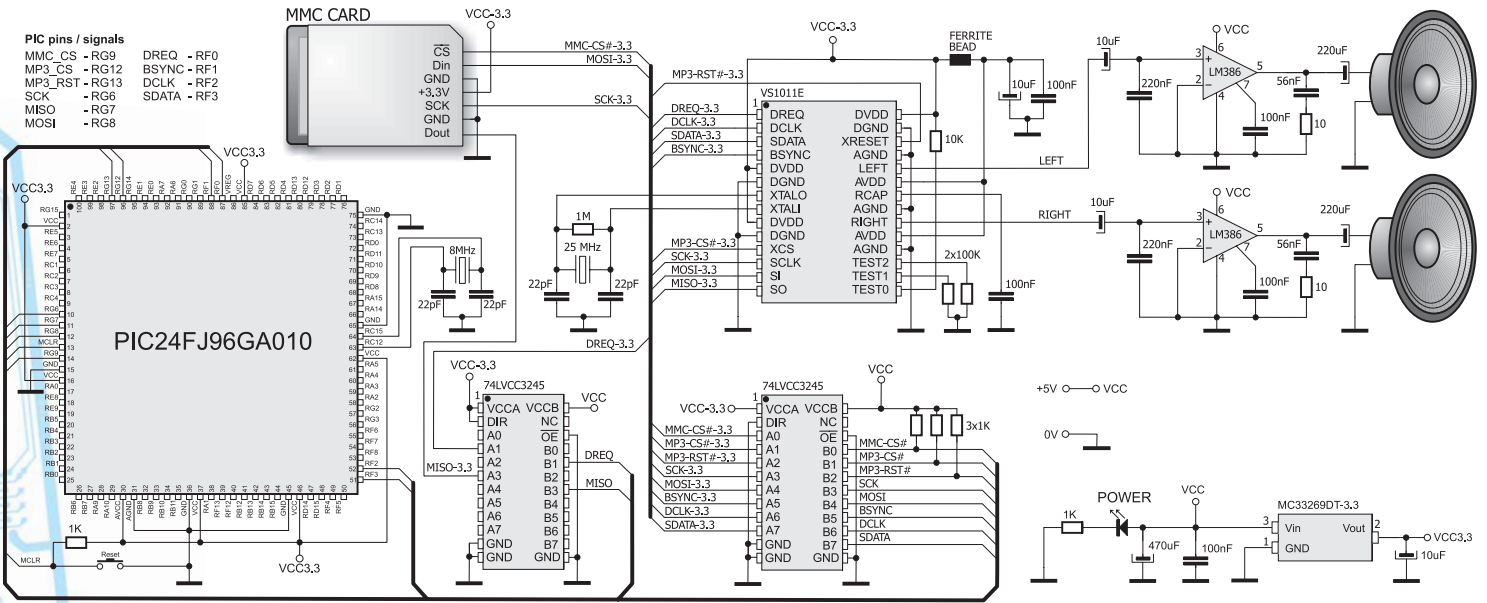


Figura 1. Diagrama de bloques del módulo Smart MP3 conectado a un PIC24FJ96GA010.



Esquema Eléctrico 1. Conexión de un módulo Smart MP3 al PIC24FJ96GA010

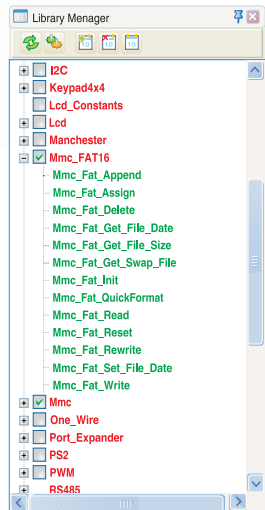
puede suceder que el contenido del buffer sea codificado antes que el microcontrolador pueda gestionar la lectura de la siguiente parte del fichero desde la tarjeta de memoria y escribirla en el buffer, lo que provocaría que el sonido se oyese de modo discontinuo. Si sucede esto, podemos reducir la velocidad de datos del fichero MP3 o usar un cristal de cuarzo de 8 MHz o superior. Ver el esquema eléctrico 1.

En cualquier caso, no tendremos que preocuparnos de esto ya que nuestro programa ha sido verificado sobre varias familias de microcontroladores con diferentes valores de cristal y fue capaz de decodificar ficheros MP3 de calidad media y alta. Por otro lado, una velocidad de datos baja significa que el buffer del codificador es rellenado con datos de sonido de mayor duración. Podría suceder que el decodificador no decodifique el contenido del buffer antes de que se intente su recarga. Para evitar esto, es necesario que estemos seguros que el decodificador está listo para recibir nuevos datos antes de que éstos sean enviados. En otras palabras, es necesario esperar hasta que la señal de petición de datos del decodificador (DREQ) esté a nivel lógico uno (1).

Mejoras

Este ejemplo puede también ser ampliado una vez que ha sido verificado. La señal DREQ puede ser comprobada de forma periódica. En el programa también se puede incorporar una rutina para el control de volumen o para mejorar el control interno de Bajos/Agudos etc. La librería MMC nos permite seleccionar un fichero con un nombre diferente. También es posible crear un conjunto de mensajes, sonidos o canciones MP3 que pueden ser usados en otras aplicaciones y enviar los ficheros MP3 adecuados para el codificador, dependiendo de las necesidades.

Más abajo está la lista de las funciones listas para usar, contenidas en la librería *Mmc_FAT16*. Esta librería está integrada en el compilador *mikroPASCAL for dsPIC*.



Mmc_Fat_Append()	Escribe al final del fichero
Mmc_Fat_Assign()	Asigna el fichero para operaciones con la FAT
Mmc_Fat_Delete()	Borra fichero
Mmc_Fat_Get_File_Date()	Obtiene fecha y hora del fichero
Mmc_Fat_Get_File_Size()	Obtiene tamaño del fichero
Mmc_Fat_Get_Swap_File()	Crea un fichero de intercambio
Mmc_Fat_Init()	Inicializa la tarjeta para operaciones FAT
Mmc_Fat_QuickFormat()	
Mmc_Fat_Read()	Lee datos desde el fichero
Mmc_Fat_Reset()	Abre el fichero para lectura
Mmc_Fat_Rewrite()	Abre el fichero para escritura
Mmc_Fat_Set_File_Date()	Establece fecha y hora del fichero
Mmc_Fat_Write()	Escribe datos en el fichero

* Funciones Mmc_FAT16 usadas en el programa

Otras funciones del compilador *mikroPASCAL for dsPIC* usadas en el programa:

Spi_Init_Advanced() Inicializa el módulo SPI del microcontrolador

Programa para demostrar el funcionamiento del módulo Smart MP3

```

program MP3_Simple_Test;
const BUFFER_SIZE = 2048; // global variables
var filename: string[13];
    i, file_size: dword;
    data_buffer: array[32] of byte;
    BufferLarge: array[BUFFER_SIZE] of byte;
procedure SW_SPI_Write(data_: byte); begin //Writes one byte to MP3 SDI
    PORTF.1 := 1; // Set BSYNC before sending the first bit
    PORTF.2 := 0; PORTF.3 := data_0; PORTF.2 := 1; // bitorder is LSB first
    PORTF.2 := 0; PORTF.3 := data_1; PORTF.2 := 1;
    PORTF.1 := 0; // Clear BSYNC after sending the second bit
    PORTF.2 := 0; PORTF.3 := data_2; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_3; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_4; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_5; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_6; PORTF.2 := 1;
    PORTF.2 := 0; PORTF.3 := data_7; PORTF.2 := 1;
    PORTF.2 := 0;
end;
// Writes one word to MP3 SCI
procedure MP3_SCI_Write(address: byte; data_in: word); begin
    PORTG.12 := 0; // select MP3 SCI
    Spi2_Write(0x02); // Write command
    Spi2_Write(address);
    Spi2_Write(Hi(data_in));
    Spi2_Write(Lo(data_in));
    PORTG.12 := 1; // deselect MP3 SCI
    while (PORTF.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
    PORTG.12 := 0; // select MP3 SCI
    Spi2_Write(0x03); // Read command
    for i := 1 to 2*(words_count) do
        begin
            data_buffer[i] := Spi2_Read(0); // read and store a byte
            Inc(data_buffer); // point to next byte
        end;
    PORTG.12 := 1; // deselect MP3 SCI
    while (PORTF.0 = 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 (PORTF.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 (PORTF.0 = 0) do nop; // wait until DREQ becomes 1, see MP3 codec
    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); begin // Set clock
    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 CLOCKSFR register
end;
procedure Soft_Reset(); begin // Software Reset
    MP3_SCI_Write(0x00, 0x204); // Set SM_RESET bit and SM_BITORD bit(bitorder is LSB first)
    Delay_us(2); // Required, see MP3 codec datasheet -> Software Reset
    while (PORTF.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
    ADPCFG := 0x0FFF; // set all AN pins to digital
    PORTF.2 := 0; TRISF.2 := 0; // Clear SW SPI SCK, configure pin as output
    PORTF.3 := 0; TRISF.3 := 0; // Clear SW SPI SDA, configure pin as output
    PORTG.12 := 1; TRISG.12 := 0; // Deselect MP3_CS, configure pin as output
    PORTG.13 := 1; TRISG.13 := 0; // Set MP3_RST pin, configure pin as output
    TRISF.1 := 1; PORTF.1 := 0; // Configure DREQ as input
    PORTF.1 := 0; TRISF.1 := 0; // Clear BSYNC, configure pin as output
end;
begin
    // main function
    filename := 'sound1.mp3'; // Set File name
    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);
    Soft_Reset(); Set_Clock(25000, 0); // SW Reset, set clock to 25MHz
    if (Mmc_Fat_Init(PORTG, 9) = 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
                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 file_size - 1 do MP3_SDI_Write(BufferLarge[i]);
                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;

```



GO TO

Tanto el código para este ejemplo, que ha sido escrito en C, Basic y Pascal para microcontroladores dsPIC® C, como los programas escritos para microcontroladores PIC® y AVR®, pueden ser localizados en nuestra página web: www.mikroe.com/en/article/