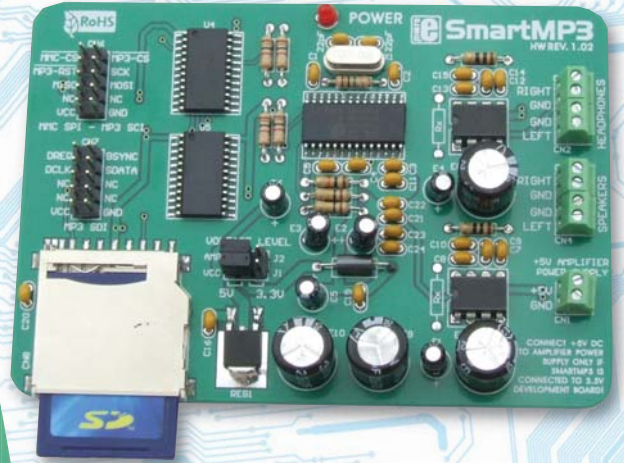


OK. Ahora necesita un ... Reproductor MP3



modulo *Smart MP3* y el sistema de desarrollo *BIGAVR2*

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 Atmega128 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

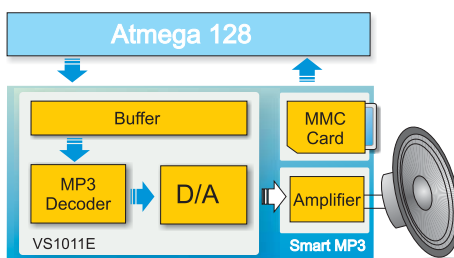
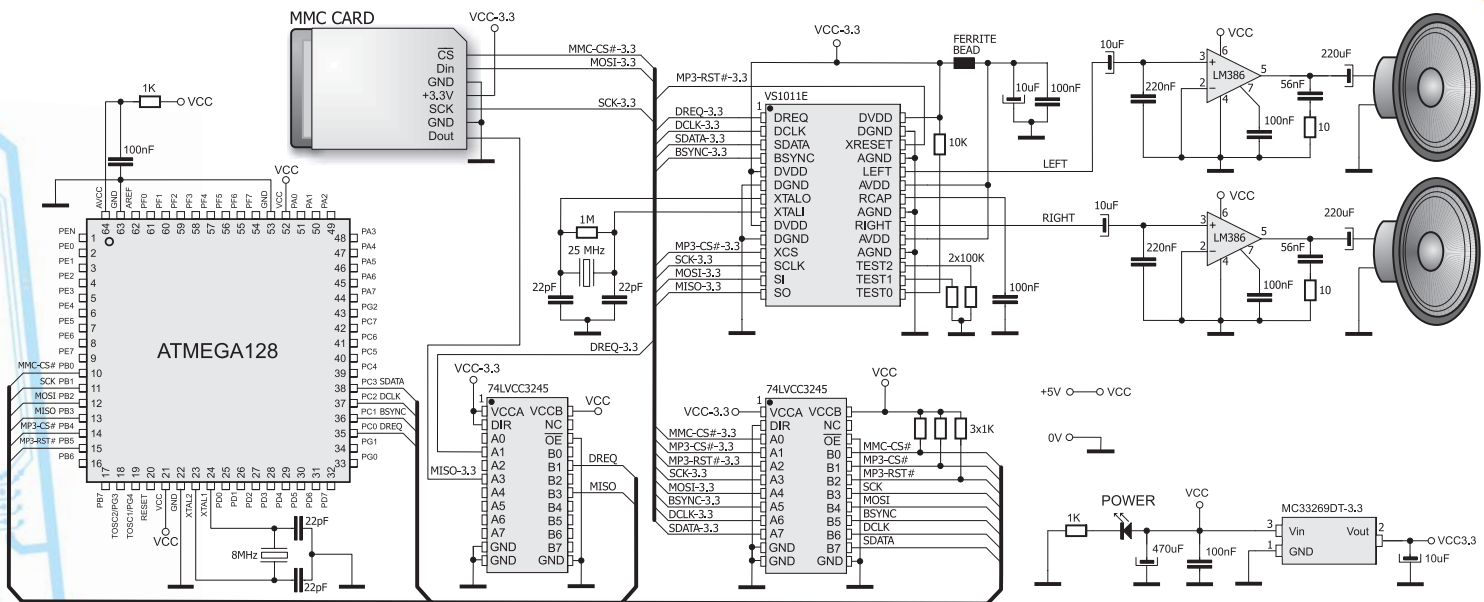


Figura 1. Diagrama de bloques del módulo *Smart MP3* conectado a un *Atmega128*.



Esquema Eléctrico 1. Conexión de un módulo Smart MP3 al Atmega128

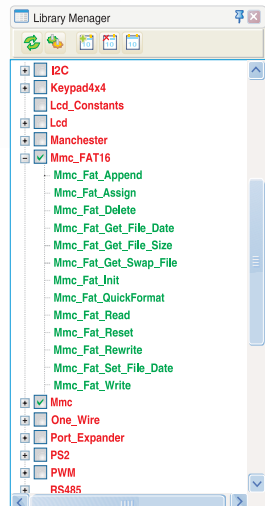
puede suceder que el contenido del buffer sea codificado antes que el microcontrolador pueda gestionar la lectura de la siguiente parte del fichero desde la tarjeta de memoria y escribirla en el buffer, lo que provocaría que el sonido se oyese de modo discontinuo. Si sucede esto, podemos reducir la velocidad 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 *mikroBASIC PRO for AVR*.



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()	Creación de 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 *mikroBASIC PRO for AVR* 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
'Smart MP3 board connections
dim Mmc_Chip_Select as sbit at PORTB.B0 dim Mmc_Chip_Select_Direction as sbit at DDRB.B0
dim MP3_CS as sbit at PORTB.B4 dim MP3_CS_Direction as sbit at DDRB.B4
dim MP3_RST as sbit at PORTB.B5 dim MP3_RST_Direction as sbit at DDRB.B5
dim DREQ as sbit at PINC.B0 dim DREQ_Direction as sbit at DDRC.B0
dim BSYNC as sbit at PORTC.B1 dim BSYNC_Direction as sbit at DDRC.B1
dim DCLK as sbit at PORTC.B2 dim DCLK_Direction as sbit at DDRC.B2
dim SDATA as sbit at PORTC.B3 dim SDATA_Direction as sbit at DDRC.B3
const BUFFER_SIZE = 512
dim filename as string[13]
i, file_size as longword
data_buffer_32 as byte[32]
BufferLarge as byte[BUFFER_SIZE]
sub procedure SW_SPI_Write(dim data as byte) 'Writes one byte to MP3 SDI
BSYNC = 1 'Set BSYNC before sending the first bit
DCLK = 0 SDATA = data_0 DCLK = 1 'bitorder is LSB first
DCLK = 0 SDATA = data_1 DCLK = 1
BSYNC = 0 'Clear BSYNC 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
end sub
sub procedure MP3_SCI_Write(dim address as byte, dim data_in as word) 'Writes one word to MP3 SCI
MP3_CS = 0 'select MP3 SCI
SPI_Write(0x02) SPI_Write(address) 'Write command, send address
SPI_Write(Hi(data_in)) 'high byte
SPI_Write(Lo(data_in)) 'low byte
MP3_CS = 1 'deselect MP3 SCI
while (DREQ = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet, Serial
end sub
'Protocol for SCI
sub procedure MP3_SCI_Read(dim start_address, words_count as byte, dim data_buffer as ^byte)
dim i as byte
MP3_CS = 0
SPI_Write(0x03) SPI_Write(start_address) 'Read command, send address
for i = 1 to (2*words_count) '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
next i
MP3_CS = 1 'deselect MP3 SCI
while (DREQ = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet, Serial
end sub
'Protocol for SCI
sub procedure MP3_SDI_Write(dim data as ^byte) 'Write one byte to MP3 SDI
while (DREQ = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet, Serial
end sub
'Protocol for SCI
SW_SPI_Write(data_)
end sub
sub procedure MP3_SDI_Write_32(dim data as ^byte) 'Write 32 bytes to MP3 SDI
dim i as byte
while (DREQ = 0) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet, Serial
for i = 1 to 32
SW_SPI_Write(data_[i]) 'Write byte pointed by data
next i
end sub
sub procedure Set_Clock(dim clock_khz as word, dim doubler as byte) 'Set clock
clock_khz = clock_khz / 2 'calculate value
if (doubler > 0) then clock_khz = clock_khz or 0x8000 end if
MP3_RST_Write(0x03, clock_khz) 'Write value to CLOCK register
end sub
sub procedure Soft_Reset() 'Software Reset
MP3_RST_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) nop wend 'wait until DREQ becomes 1, see MP3 codec datasheet, Serial
end sub
'Protocol for SCI
for i = 1 to 2048 MP3_SDI_Write(0) next i 'feed 2048 zeros to the MP3 SDI bus
end sub
sub procedure Init()
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
BSYNC_Direction = 1 BSYNC = 0 'Clear BSYNC, configure pin as output
end sub
main:
' main function
filename = "sound1.mp3" 'Set File name
Init()
SPI_Init_Advanced(SPI_MASTER, SPI_FCY_DIV128, SPI_CLK_LO_LEADING)
Spi_Rd_Ptr = @SPI_Read
Soft_Reset() Set_Clock(25000.0) 'SW Reset, set clock to 25MHz
if (Mmc_Fat_Init()) = 0 then
'reinitialize spi at higher speed
SPI_Init_Advanced(SPI_MASTER, SPI_FCY_DIV2, SPI_CLK_LO_LEADING)
if (Mmc_Fat_Assign(filename, 0) < 0) then
Mmc_Fat_Reset(filename) 'Call Reset before file reading
while (file_size > BUFFER_SIZE) 'send file blocks to MP3 SDI
for i = 0 to BUFFER_SIZE - 1 Mmc_Fat_Read(BufferLarge[i]) next i
for i = 0 to BUFFER_SIZE/32 - 1 MP3_SDI_Write_32(@BufferLarge[i*32]) next i
file_size = file_size - BUFFER_SIZE
wend
'send the rest of the file to MP3 SDI
for i = 0 to file_size - 1 Mmc_Fat_Read(BufferLarge[i]) next i
for i = 0 to file_size - 1 MP3_SDI_Write(BufferLarge[i]) next i
end if
end if
end.

```



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