Cika Electrónica

Nota de Aplicación: CAN-064
Título: **Reproducción de audio en microcontroladores: Rabbit 4000**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	09/02/07	

En el comentario técnico CTC-046 vimos la forma de aprovechar los generadores de PWM para reproducir audio. En esta nota de aplicación veremos un caso práctico con Rabbit 4000.

Análisis

El módulo RCM4000 dispone de un clock de 59MHz. Intentamos reproducir una señal de audio con un ancho de banda cercano al del oído, con una resolución de 8-bits.

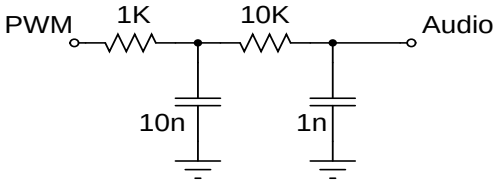
El generador de PWM de Rabbit funciona a 10-bits, y toma reloj del clock de periféricos, por lo que la máxima frecuencia de trabajo es de $\frac{59\,MHz}{2\times1024}=28800\,Hz$. Para bajar los requerimientos de filtrado, habilitaremos el modo SPREAD, lo que elevará la frecuencia de la señal de salida a 115200Hz. Nuestra frecuencia de muestreo será 28800 Hz y esto nos permite un ancho de banda de audio de poco más de 14KHz.

Para disponer de una señal de audio que reproducir, leeremos un archivo convenientemente convertido de una tarjeta MMC o SD. Dada la característica de la fuente de audio, deberemos reproducirlo a medida que lo leemos de la tarjeta. Para esto, la velocidad de lectura desde la SD deberá ser superior a la de reproducción de audio, es decir, mayor a 30KB/s. El formato será PCM 8-bits sin signo.

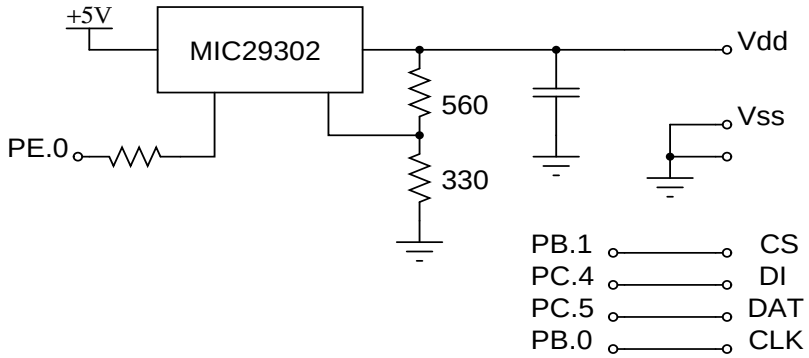
Debido a que no podemos pausar, utilizaremos un sistema double-buffered: disponemos de dos buffers, mientras reproducimos uno podemos ir llenando el otro, y conmutamos luego.

Hardware

Colocamos un simple filtro pasabajos RC a la salida del generador de PWM utilizado. La salida del filtro se conecta a un amplificador de audio, teniendo cuidado de agregar un capacitor de desacople si el amplificador no lo tiene, dado que nuestra salida está entre 0 y 3,3V. Los valores utilizados son comerciales y sitúan el filtro alrededor de los 16KHz



El hardware de la tarjeta MMC/SD es el siguiente:



Software

El software de lectura de MMC/SD es prácticamente idéntico al desarrollado en CAN-054 y CAN-055, con menores modificaciones para adecuarlo al hardware de Rabbit 4000, como ser diferentes ports y fundamentalmente diferentes pines de I/O, y a su mayor velocidad de operación. Se agregaron modificaciones en la rutina de inicialización para garantizar que los pines queden en estado bajo al momento del reset, de modo que la tarjeta no quede parcialmente alimentada por los pines de I/O y responda satisfactoriamente al reset inicial. No desarrollaremos esto aquí, el lector puede observarlo en el listado que acompaña a esta nota.

Para soportar la lectura de la tarjeta a alta velocidad, desarrollamos una versión restringida de *FAT_read()*, la que llamamos *FAT_fastread()*, que nos permite eliminar el overhead de lectura por bytes, dado que leeremos bloques múltiplos de 512 bytes correctamente alineados. De esta forma, pudimos llegar a velocidades de lectura desde la SD mayores a 50KB/s[1]. Si bien esto no es crítico, dado que el sistema funciona con la rutina genérica, el optimizar la lectura nos permite disponer de mayor capacidad de procesamiento para otras tareas. Tampoco desarrollaremos esto aquí, el lector puede observarlo en el listado que acompaña a esta nota

El código que toma cada muestra del buffer, revisa si se llega al final, conmuta de buffer, señaliza el pedido de llenado, y escribe en el registro de PWM, está desarrollado en assembler, y es el siguiente:

```
#define BUF_SIZE 6144
unsigned char buf1[BUF_SIZE],buf2[BUF_SIZE],*ptr;
unsigned int bytes,bytes1,bytes2;
unsigned char which,load,run;

#asm root
PWM_handler::
        push af
        ld a,(run)                  ; si no está funcionando, mantiene nivel de '0'
        and a
        jr nz,ok
        ld a, 0x80
        ioi ld (PWM2R),a
        jr chau
ok:
        push hl
        ld hl,(ptr)                 ; puntero
        ld a,(hl)                   ; toma muestra
        ioi ld (PWM2R),a            ; escribe en PWM (8-bits más significativos)
        inc hl                      ; incrementa puntero
        ld (ptr),hl                 ; guarda
        ld hl,(bytes)               ; byte count
        dec hl                      ; -1
        ld (bytes),hl               ; actualiza
        ld a,h                      ; 0 ?
        or l
        jr nz,done                  ; no, saltea
        ld a,1                      ; sí, recargar este buffer
        ld (load),a
        ld a,(which)                ; y conmutar al otro
```

1 90KB/s con una tarjeta SD y el divisor de SPI en valores muy bajos

```
        xor 0xFF
        ld (which),a
        jr nz,use2
        ld hl,buf1                  ; actualiza puntero
        ld (ptr),hl
        ld hl,(bytes1)              ; y byte count para el nuevo buffer
        jr use
use2:
        ld hl,buf2                  ; ídem
        ld (ptr),hl
        ld hl,(bytes2)
use:
        ld (bytes),hl
        ld a,h                      ; bytes = 0? (nuevo buffer vacío)
        or l
        jr nz,done
        ld (run),a                  ; sí, fin del tema
done:
        pop hl
chau:
        pop af
        ipres
        ret
#endasm
```

Las interrupciones son generadas por el PWM, y el llenado del buffer se realiza desde el programa principal, como puede observarse en el listado a continuación. Este programa reproduce tres temas, FORD1.PCM[2] FORD2.PCM y FORD3.PCM de la tarjeta SD, y luego termina. Debido a que utilizamos PC.4 como salida, se perderá la comunicación con Dynamic C.

```
int main( void )
{
unsigned char rc;
int i,k;
char filename[13];


        if(MMCFAT_init()==-1)
        goto oops;
        if(MMCFAT_mount(0)==-1)
        goto oops;

        run=0;
        pwm_init(28800L);                           // sample rate
        SetVectIntern(0x17,PWM_handler);            // vector de interrupción
        pwm_set(2,513,PWM_SPREAD | PWM_INTNORMAL | PWM_USEPORTC | PWM_INTPRI3);

        for(k=1;k<4;k++){                           // cada tema
                sprintf(filename,"FORD%d.PCM",k);   // genera nombre
                puts(filename);
                if(MMCFAT_open(filename)!=-1) {      // abre archivo
                        which=load=1;               // comienza cargando ambos buffers
                        do {
                                if(load){           // hay que cargar ?
                                        load=0;
                                        if(which){     // recarga buffer 1 (reproduce 2)
                                                i=MMCFAT_fastread(buf1,BUF_SIZE);
                                                bytes1=i;
                                        }
                                        if((!which)||(!run)){  // recarga buffer 2 (reproduce 1)
                                                i=MMCFAT_fastread(buf2,BUF_SIZE);
                                                bytes2=i;
                                        }
                                        if(!run){      // comienza reproduciendo buffer 1
                                                which=0;
                                                bytes=bytes1;
                                                ptr=buf1;
                                                run=1;
                                        }
                                }
                        } while (i>0); // repite ciclo hasta que termina el archivo
```

2 Robben Ford

```
                        if(which)                       // indica buffer vacío
                                bytes1=0;
                        else
                                bytes2=0;
                }
                while(run);             // espera a que termine de reproducir el tema
        }                               // y sigue con el otro

oops:   BitWrPortI ( PEDR, &PEDRShadow, 0, 0 );      // no PWR
}
```