Cika Electrónica

Nota de Aplicación: CAN-063
Título: **Reproducción de audio en microcontroladores: Ramtron VRS51L3074**
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 los Ramtron VRS51L3074.

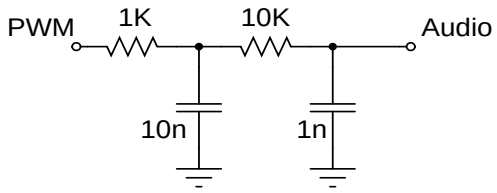
Análisis

El VRS51L3074 dispone de un clock interno de 40MHz. Intentamos reproducir una señal de audio con un ancho de banda cercano al del oído, con una resolución de 8-bits.

Para disponer de 8-bits de resolución, necesitamos de una frecuencia de operación de $\frac{40\,MHz}{256} = 156{,}25\,KHz$. Para bajar la frecuencia de procesamiento y los requerimientos de filtrado, utilizaremos una frecuencia de muestreo cuatro veces inferior: 39062,5 Hz, esto nos permite un ancho de banda de audio de poco más de 19KHz

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



Para disponer de una señal de audio que reproducir, utilizaremos el MCP3204, con el hardware descripto en la CAN-061.

Software

Loop infinito

A fin de simplificar el software y mostrar el concepto, realizaremos primero la lectura del ADC y actualización del ancho de pulso dentro de un loop infinito en el programa principal. Más adelante veremos la utilización de interrupciones.

La inicialización del ADC es la misma que se describe en CAN-061, por lo que no requiere comentarios aquí.

La inicialización del generador de PWM aprovecha código provisto por el fabricante, y es la siguiente:

```
void initPWM(void)
{
     PERIPHEN2 |= (1<<1);              // Habilita PWM SFRs (PWMSFREN)
     DEVIOMAP |= (1<<6);               // Alternate mapping (usamos P5.5)
     PWMCFG = 0x20;                    // borra canales
     PWMCLKCFG = 0x00;                 // no clock prescaler
     PWMLDPOL = (1<<5);                // polaridad invertida, ancho pulso positivo = PWMMID
                                       // PWM5 END = 0x0FF (8-bit)
     PWMCFG = 0x5D;                    // PWM5 END MSB
```

```
     PWMDATA = 0x00;                    // Max Count MSB = 0x00
     PWMCFG = 0x4D;                     // PWM5 END LSB
     PWMDATA = 0xFF;                    // Max Count = 0xFF
                                         // PWM5 MID = nada
     PWMCFG = 0x55;                     // PWM5 MID MSB
     PWMDATA = 0x00;                    // PWM MID MSB = 0x00
     PWMCFG = 0x45;                     // PWM5 MID LSB
     PWMDATA = 0x01;                    // PWM MID LSB = 0x01

     PWMCFG &= 0x1F;                    // Resetea PWMWAIT para actualizar los generadores
     PWMEN |= (1<<5);                   // habilita PWM5 (PWM5EN)
}
```

A continuación, el loop principal, donde leemos el valor obtenido del ADC y actualizamos el ciclo de trabajo. La frecuencia de muestreo viene dada por la velocidad de lectura del ADC (~50Ksps), y dado que no está sincronizada con el PWM se perderán algunas actualizaciones. Como los generadores del VRS51L3074 son buffered, no generará inconvenientes a esas frecuencias elevadas.

```
#define START  (1<<2)
#define SINGLE (1<<1)
#define CHANNEL 3

void main ()
{
unsigned int adcdata;

    initAD();
    initPWM();

    while (1){
        // Lee el ADC
        SPIRXTX2 = 0x00;
        SPIRXTX1 = (CHANNEL<<6);
        SPIRXTX0 = START | SINGLE | (CHANNEL>>2);

        while(!(SPISTATUS & 0x02));


        // Obtiene muestra (12-bits)
        adcdata= (SPIRXTX1 << 8);
        adcdata|= SPIRXTX0;
        adcdata&= 0x0FFF;                    // 12-bits
        adcdata>>=4;                         // 8-bits
        if(adcdata==0)                       // PWM no acepta el valor 0
          adcdata++;
        // PWM5 MID = adcdata
        PWMCFG = 0x55;                     // PWM5 MID MSB
        PWMDATA = (adcdata>>8);            // PWM MID MSB = low(adcdata)
        PWMCFG = 0x45;                     // PWM5 MID LSB
        PWMDATA = adcdata&0x00FF;          // PWM MID LSB = high(adcdata) (en este caso 0)
        PWMCFG &= 0x1F;                    // Resetea PWMWAIT para actualizar el generador
    }
}
```

Interrupciones

Una vez mostrado el concepto, desarrollaremos un esquema que nos permita que nuestro generador funcione de forma autónoma, por interrupciones, sincronizado con PWM. Debido a que los generadores de PWM no generan interrupciones excepto cuando funcionan como timer, utilizaremos un segundo generador (disponemos de ocho) para las interrupciones. A cada interrupción, iniciamos la conversión del ADC. éste, al terminar, generará una interrupción (en realidad será la recepción de una transacción de 24-bits en la interfaz SPI lo que generará la interrupción), momento en el cual obtenemos la muestra del AD y actualizamos el ancho de pulso. Entonces, el PWM timer interrumpe cada aproximadamente 25,6us, disparamos el AD; el módulo SPI interrumpirá unos 20us después y actualizamos el generador PWM. Para poder visualizar la secuencia de operaciones, nos tomamos la libertad de manipular algunos bits del port P2.

Veremos primero la inicialización de los generadores de PWM, el que utilizamos como generador y el que utilizamos como timer:

```
void initPWM(void)
{
```

```
    PERIPHEN2 |= (1<<1);              // Habilita PWM SFRs (PWMSFREN)
    DEVIOMAP |= (1<<6);              // Alternate mapping (usamos P5.5)
    PWMCFG = 0x20;                   // borra canales
    PWMCLKCFG = 0x00;                // no clock prescaler
    PWMLDPOL = (1<<5);               // polaridad invertida, ancho pulso positivo = PWMMID
                                     // PWM5 END = 0x00FF (8-bit)
    PWMCFG = 0x5D;                   // PWM5 END MSB
    PWMDATA = 0x00;                  // Max Count MSB = 0x00
    PWMCFG = 0x4D;                   // PWM5 END LSB
    PWMDATA = 0xFF;                  // Max Count = 0xFF
                                     // PWM5 MID = nada
    PWMCFG = 0x55;                   // PWM5 MID MSB
    PWMDATA = 0x00;                  // PWM MID MSB = 0x00
    PWMCFG = 0x45;                   // PWM5 MID LSB
    PWMDATA = 0x01;                  // PWM MID LSB = 0x01
                                     // PWM4 MID = 0x03FF (10bit)
    PWMCFG = 0x54;                   // PWM4 MID MSB
    PWMDATA = 0x03;                  // Max Count MSB = 0x03
    PWMCFG = 0x44;                   // PWM4 MID LSB
    PWMDATA = 0xFF;                  // Max Count = 0xFF

                                     // PWM4-7 timer interrupt
    INTSRC2 &= ~(1<<5);               // PWM7:4 Timer module interrupt
    INTEN2 |= (1<<5);                 // habilita PWM7:4 timer module interrupt

    PWMTMREN = (1<<4);               // Usa PWM4 como timer
    PWMEN = (1<<5)|(1<<4);           // Habilita PWM5 y PWM4

    PWMCFG &= 0x1F;                  // Resetea PWMWAIT para actualizar los generadores
}
```

A continuación, la inicialización de la interfaz SPI para generar interrupciones. Además de seleccionar qué interrupciones genera el periférico en SPICONFIG, debemos asignar el módulo de interrupción al periférico (SPI) en vez del pin, y luego habilitar la interrupción generada por el módulo, lo que hacemos al final de la rutina

```
void initAD(void)
{
char readflag = 0x00;

       PERIPHEN1 |= 0xC0;                  // Habilita interfaz SPI

       while(!(SPISTATUS &= 0x08));        // espera inactividad

       SPICTRL = 0x85;                     // SPICLK = /32 (1.25MHz)
                                           // CS0 Activo
                                           // SPI Modo 0 Phase = 0, POL = 0
                                           // SPI Master

       SPICONFIG = 0x42;                   // SPI Chip select automático
                                           // Clear SPIUNDEFC Flag
                                           // SPILOAD = 0 -> Manual CS3
                                           // SPI RXAV Interrupt habiltada

       SPISTATUS = 0x00;                   // MSB primero

       SPISIZE = 0x17;                     // Transaction Size = 24-bits

       readflag = SPIRXTX0;                // Lee SPI RX buffer para resetear RXAVF
        INTSRC1 &= ~(1<<2);                // SPI RXAV module interrupt
        INTEN1 |= (1<<2);                  // habilita interrupt
}
```

Luego, veremos las correspondientes rutinas de interrupciones. Por comodidad hemos utilizado C. En otras notas de aplicación más exigentes, emplearemos assembler o una combinación de ambos.

Las operaciones sobre el port de I/O P2 permiten visualizar los instantes y duración de la interrupción mediante un osciloscopio.

```
#define START  (1<<2)
#define SINGLE (1<<1)
#define CHANNEL 3
```

```
void PWMTMRInterrupt(void) interrupt 13
{
        // como usamos sólo PWM4, no revisamos quién interrumpe

P2|=0x1F;
        PWMTMRF &= ~(1<<4);                         // resetea PWM Timer 4 OV Flag
                                                    // SPI
       SPIRXTX2 = 0x00;
       SPIRXTX1 = (CHANNEL<<6);
       SPIRXTX0 = START | SINGLE | (CHANNEL>>2);
P2&=~0x1F;
}

void ADInterrupt(void) interrupt 2
{
unsigned int adcdata;

P2|=0x1F;
                                                    // Lee SPI y escribe PWMMID
       adcdata= (SPIRXTX1 << 8);
       adcdata|= SPIRXTX0;
       adcdata&= 0x0FFF;                            // 12-bits
       adcdata>>=4;                                 // 8-bits
       if(adcdata==0)                               // PWM no acepta 0
         adcdata++;
                                                    // PWM5 MID = adcdata
       PWMCFG = 0x05;
       PWMDATA = (unsigned char)adcdata;
P2&=~0x1F;
}
```

Finalmente, el programa principal, que no hace absolutamente nada una vez inicializado el hardware.

```
void main ()
{
    initAD();
    initPWM();
        P2&=~0x1F;                                  // Outputs = 0
        P2PINCFG&=~0x1F;                            // 11100000 P2.0,1,2,3,4 = output
    GENINTEN = 0x03;                                // global interrupt

    while (1){
    }
}
```

La ocupación de CPU es mínima gracias a las capacidades del hardware del VRS51L3074.