Cika Electrónica

Nota de Aplicación: CAN-069
Título: **Software timers con Ramtron VRS51L3074**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	21/02/07	

Si bien el VRS51L3074 dispone de tres timers y sus ocho canales PWM pueden configurarse cada uno para funcionar como timers, en algunos casos la cantidad de eventos simultáneos, de salidas a controlar, o el tipo de aplicación misma, requieren de un sistema de software timers sincronizados para controlar eventos en tiempos más relajados, del orden de las decenas de milisegundos hasta las decenas de segundos.

Para estos casos, presentamos un sencillo y eficiente esquema de software timers de 8-bits controlados por uno de los hardware timers del micro.

La idea es muy simple: el Timer2 interrumpe cada 10ms; su rutina de interrupción se encarga de llevar la cuenta para contar segundos y decrementar los timers en uso. A fin de dejar libre la memoria interna para asuntos que requieran mayor velocidad, utilizamos la RAM on-chip mapeada en el área de memoria de datos externa:

```
#define NUM_CSTIMERS 5
#define NUM_SECTIMERS 5

static unsigned char c1s;
__xdata unsigned char CS_TIMERS[NUM_CSTIMERS],SEC_TIMERS[NUM_SECTIMERS];

void tmr2Interrupt(void) interrupt 8 __naked
{
        __asm
        anl _T2CON,#~(1<<7)                 ; ACK
        mov _TH2,#0x9E                      ; recarga cuenta
        mov _TL2,#0x58
        push acc                            ; salva registros
        push dpl
        push dph
        push ar2
        push psw
        mov psw,#0x00                       ; bank 0
        mov dpl,#_CS_TIMERS                 ; apunta a timers cortos (10ms)
        mov dph,#(_CS_TIMERS >> 8)
        mov R2,#NUM_CSTIMERS                ; cantidad de timers
l20:    movx a,@dptr                        ; lee valor
        jz l21                              ; ignora si es cero
        dec a                               ; decrementa
        movx @dptr,a                        ; actualiza
l21:    inc dptr                            ; siguiente
        djnz R2,l20                         ; loop

        dec _c1s                            ; cuenta 0,01s
        mov a,_c1s
        jnz l0                              ; llegó a 0 ?
        mov _c1s,#100                       ; reinicia en 100, para detectar 1s
        ; llama a rutina de RTC si es necesario
        mov dpl,#_SEC_TIMERS                ; apunta a timers de 1s
        mov dph,#(_SEC_TIMERS >> 8)
        mov R2,#NUM_SECTIMERS               ; cantidad de timers
l30:    movx a,@dptr                        ; lee
        jz l31                              ; ignora si es cero
        dec a                               ; decrementa
        movx @dptr,a                        ; actualiza
l31:    inc dptr                            ; siguiente
        djnz R2,l30                         ; loop
l0:     pop psw                             ; recupera registros
        pop ar2
```

```
        pop dph
        pop dpl
        pop acc
        reti                                ; devuelve control
        __endasm;
}
```

Agregamos además una simple rutina de inicialización para configurar el Timer2 y poner todos los software timers a cero:

```
void init_softtimers(void)
{
register unsigned char r;
__xdata unsigned char *p;

  c1s=100;                          // 100 x 10ms = 1s
  r=NUM_CSTIMERS;
  p=CS_TIMERS;
  while(r--){                       // pone a cero CS_TIMERS
    *(p++)=0;
  }
  r=NUM_SECTIMERS;
  p=SEC_TIMERS;
  while(r--){                       // pone a cero SEC_TIMERS
    *(p++)=0;
  }
  PERIPHEN1 |= (1<<2);               // habilita Timer 2
  TH2 = 0x9E;                        // 10ms @40MHz (cuenta hacia arriba)
  TL2 = 0x58;
  T2CLKCFG = 0x04;                   // /16 prescaler
  T2CON = 0x04;                      // Inicia timer, cuenta hacia arriba
  INTSRC2 &= ~(1<<0);                // Timer2 module interrupt
  INTEN2 |= (1<<0);                  // Interrupt 8 enable
}
```

Finalmente, un sencillo programa de ejemplo para observar cómo pueden utilizarse los software timers:

```
#include "softtimers.h"

main()
{
        init_softtimers();
        GENINTEN = 0x03; //Enable global interrupts
        ...

        CS_TIMERS[3]=28;              // 28x10=280ms
        while(CS_TIMERS[3]){
                // hace algo mientras espera (o no)
        }
        ...

        SEC_TIMERS[4]=5;              // ~5 seg
        while(SEC_TIMERS[4]){
                // etc
        }
}
```

Dado que inicializamos el timer cuando queremos, y éste es decrementado por la rutina de interrupciones, que corre libre, la precisión que tenemos es de una cuenta en defecto, es decir, si seteo un timer en un valor justo en el instante en el que la rutina de interrupciones se ejecutó, tendré la demora especificada; si lo hago justo en el instante anterior a que la rutina de interrupciones se ejecute, tendré una cuenta menos. Entonces, el tiempo de demora de esta implementación es de $\left(x-\frac{1}{2}\right)\pm\frac{1}{2}$. Por ejemplo: 28 en CS_TIMERS (cuentas de 10ms) es en realidad $275\pm5\,ms$ y 5 en SEC_TIMERS es $4{,}5\pm0{,}5\,seg$.

Si se desea mayor precisión se puede emplear una combinación de un timer corto y uno largo: el timer corto puesto en 1 detecta el instante de cuenta, y a continuación se inicia el timer largo; aunque en realidad, recomendamos utilizar los hardware timers u otro esquema si la precisión de esta implementación no es satisfactoria.