Cika Electrónica

Nota de Aplicación: CAN-040
Título: **Ejemplo de aplicación de PIC10F206**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	14/09/05	

Presentamos un sencillo ejemplo de aplicación de PIC10F206, un integrante de la nueva línea PIC10F de Microchip. Estos dispositivos son micros de 6 patitas en encapsulado SOT-23, de muy bajo costo, tendientes a eliminar la utilización de varios chips de lógica o proveer funciones adicionales sin por ello incrementar el espacio ocupado. En este caso, reemplazamos una típica aplicación de 555 y/o algunas compuertas lógicas con sus capacitores y resistencias asociadas para la generación de ondas cuadradas.

Breve descripción del PIC10F206

El PIC10F206 es sumamente simple, se trata de un microcontrolador con 512 words de flash y 24 bytes de RAM, sin interrupciones, con un timer de 8-bits (TMR0), un comparador, tres entradas/salidas, y una entrada; todo en seis pines. Posee además un oscilador interno de 4MHz al 1%, lo que le permite operar a 1MIPS pico. El core es de 12-bits, y las instrucciones son similares al 12C508, es decir, incluye TRIS y OPTION. Todos los I/O configurados como salidas son capaces de proveer 20mA, y si se los configura como entradas, pueden tener un pull-up interno. El pin que es sólo entrada no tiene pull-up interno. Por supuesto incluye un WDT con su propio oscilador, y a pesar de consumir sólo alrededor de 1/2 A, puede ponerse en modo sleep, saliendo de éste al detectar movimiento en un pin (wake on change).

Otra característica interesante es que puede ser programado en PICSTART Plus o ICD2, o con programadores económicos como el Baseline Flash Programmer o el mismo Flash Starter Kit, mediante un adaptador.

Descripción del ejemplo propuesto

Como ejemplo de aplicación, les proponemos realizar un simple ahuyentador de artrópodos rastreros, particularmente hemípteros y arácnidos, aunque tal vez funcione con algún que otro roedor y/o enloquezca al molesto perro del vecino, dependiendo del grado de amplificación que el lector esté dispuesto a realizar.

Observando las especificaciones de los ahuyenta cucarachas, arañas, roedores, y etc. que publican alguna, notamos que generan una señal de entre 20y 50 KHz, la cual modulan en frecuencia y/o aplican de forma pulsante para que el pobre animalito no se acostumbre y decida irse a hacer su nido a otro lugar, donde ese molesto ruido no perturbe su sistema nervioso[1]. Nuestro ejemplo, entonces, consiste en generar trenes de pulsos de diversas frecuencias distribuidas entre ese rango, de longitud cercana al segundo, separados por pausas de alrededor de medio segundo. Cada tren de pulsos incrementa la frecuencia respecto al tren anterior, y alcanzada la frecuencia máxima reinicia desde la frecuencia mínima.

Hardware

Aprovechamos las tres salidas del PIC10F206, y utilizaremos una de ellas como salida single-ended, destinada a ser aplicada a un amplificador o equivalente, siendo puesta a cero en los silencios; y las otras dos como salida diferencial, destinada a excitar directamente un buzzer piezoeléctrico o por qué no un puente H. Pensando en el buzzer piezoeléctrico, pondremos ambas salidas a cero para no dejar cargado el buzzer y evitar el molesto "click" que produce. La aplicabilidad o no de determinado buzzer para esta tarea, corre por cuenta del lector.

1 Lamentablemente no conseguimos ningún entomólogo que atienda nuestros llamados (Gil Grissom se excusó por cuestiones de tiempo) y no podemos asegurar que realmente funcione, pero al menos sí podemos garantizar que como generador de señales similares a lo que la mayoría de los fabricantes de estos aparatitos dicen generar, resulta más compacto que su equivalente con componentes discretos y circuitos integrados.

Software

Generar una señal de más de 40KHz con un clock de 1MHz no es tarea fácil, afortunadamente no tenemos nada más importante que hacer y podemos dedicar toda la CPU a este fin. Para lograr una buena granularidad, es decir, tener una cantidad interesante de frecuencias diferentes, desarrollamos un esquema de saltos que aprovecha la posibilidad de sumar al PC: salteando la cantidad de NOPs en el loop de generación de pulsos controlamos la frecuencia generada:

```
movf freq,W
addwf PCL,f                    ; saltea nop's de acuerdo a freq
nop
nop
nop
```

El loop de programa así desarollado, genera semiciclo positivo y semiciclo negativo en un solo loop, ejecutándose en 13 ciclos de máquina más 2 veces la cantidad de NOPs insertados. La duración de un ciclo es:

$$T = \frac{4}{clockrate} \cdot ((2 * NOPs) + 13)$$

De donde se deduce que la cantidad de NOPs a insertar para obtener un período (*T*) o frecuencia dados es:

$$NOPs = \frac{\frac{clockrate \cdot T}{4} - 13}{2} = \frac{\frac{clockrate}{4 \cdot frecuencia} - 13}{2}$$

La cantidad máxima de NOPs es 24 (ponemos 24 NOPs en el código). La cantidad de NOPs a evitar para cada semiciclo, a una frecuencia dada es entonces:

$$freq = 24 - \left(\frac{\frac{clockrate}{4 \cdot frecuencia} - 13}{2} \right)$$

A fin de computar la frecuencia y duración del tren de pulsos, calculamos la cantidad de ciclos de clock necesarios para generar las frecuencias mínima y máxima, y la cantidad de ciclos de esa frecuencia que entran en la duración del tren de pulsos. Luego, iremos decrementando la duración del ciclo e incrementando la cantidad de ciclos para ir de uno a otro extremo. Si bien esto es erróneo porque no hay una relación lineal, dudamos mucho que las cucarachas decidan quedarse por este motivo.

El listado del programa es el siguiente:

```
              list    p=10F206           ; list directive to define processor
        #include      <p10F206.inc>

              RADIX   dec

#define uno      B'00000110'
#define cero B'00000001'


; Longitud del tren de pulsos en décimas de segundo
#define pulseLenght 10
; Longitud de la pausa en mitades de segundo
#define noni   1
; Frecuencias límite en Hz
#define fmin 20000
#define fmax 50000

clockrate equ 4000000                              ; frecuencia del oscilador interno
delayconst1    equ     (clockrate/(8*256*8))       ; 1/2 segundo
delayconst2    equ     (noni)

freqconstm     equ 24-(((clockrate/(4*fmin))-13)/2)  ; iteraciones por ciclo
freqconstM     equ 24-(((clockrate/(4*fmax))-13)/2)
cyclesm        equ     (fmin*pulseLenght)/2560       ; iteraciones de 256 ciclos para 0,1 sec
cyclesM        equ (fmax*pulseLenght)/2560
cyclesK        equ (cyclesM-cyclesm)/(freqconstM-freqconstm)
```

```
vars    UDATA
delay   RES 3
freq    RES 1
cycles  RES 3

master  CODE    0
start
        movwf OSCCAL                ; Calibra el oscilador interno
         clrf GPIO
         movlw B'01000001'
         movwf CMCON0               ; Selecciona GP2
         movlw B'11001111'
        option                      ; GP2
         movlw  B'00001000'         ; GP0,GP1,GP2 = output
        tris GPIO
        clrf cycles+2               ; 256 iteraciones de 1 ciclo
main
        movlw freqconstm
        movwf freq
        movlw cyclesm
        movwf cycles
        movwf cycles+1
cycloop movlw uno                               ; 1
         movwf GPIO
        movf freq,W
        addwf PCL,f                             ; 5 + nop's
        fill (nop),24                                   ; 24 nop's
        movlw cero
         movwf GPIO
        movf freq,W
        addwf PCL,f                             ; 10 + 2*nop's
        fill (nop),24                                   ; 24 nop's
        decfsz cycles+2,f                   ; realiza 256 ciclos
        goto cycloop                        ; 13 + 2*freq
        decfsz cycles+1,f                   ; cycles' veces para generar un tren de pulsos
        goto cycloop
        clrf GPIO                           ; elimina tensión en salidas durante la pausa
        call Sleepdelay                     ; no hace nda por 'noni' mitades de segundo
        incf freq,f                         ; Incrementa la frecuencia
        movlw freqconstM+1
        subwf freq,w
        btfsc STATUS,Z              ; Si excedemos fmax
        goto main                   ; reinicia el ciclo
        movlw cyclesK               ; sino, incrementa la cantidad de ciclos a realizar
        addwf cycles,f
        goto cycloop


Sleepdelay                                      ; Demora de 'noni' mitades de segundo
        movlw delayconst2
        goto l0
HSdelay movlw 1                                 ; Demora de medio segundo
l0      movwf delay+2
        clrf delay
l2      movlw delayconst1
        movwf delay+1
l1      goto l1a
l1a     goto l1b
l1b     nop
        decfsz delay,f
        goto l1
        decfsz delay+1,f
        goto l1
        decfsz delay+2,f
        goto l2
        retlw 0

        end
```