Temporizadores y contadores en tiempo real: El módulo Timer0 del PIC.

Creado: 19/08/2004
Actualizado: 18/01/2005

By [-Ali-] #pic (irc hispano)

1. Introducción

A menudo al utilizar un microcontrolador nos encontramos con la necesidad de contar o generar eventos cada cierto tiempo. Para ayudar en este tipo de tareas, es habitual que los microcontroladores dispongan de circuitos internos para ello. Este circuito, es comúnmente denominado Timer/Counter (Temporizador/Contador) aunque también por motivos históricos es habitual encontrarlo con el nombre de RTCC (Real Time Clock Counter).

Nos vamos a centrar en el Timer/Counter de 8 bits habitual en los microcontroladores PIC 16F84, denominado en la hoja de especificaciones como **módulo Timer0** (en otros modelos es posible encontrar adicionales módulos de 8 ó 16 bits cuyo el funcionamiento básico es el mismo).

Antes de explicar el funcionamiento y uso del Timer0, vamos de definir los siguientes conceptos para evitar confusiones:

- *Frecuencia de oscilación* (F_{osc}): Frecuencia de trabajo externa del PIC (un cristal de cuarzo, un resonador, etc.).
- *Frecuencia interna de instrucciones* (F_{int}): Frecuencia del reloj interno de instrucciones generada a partir de la frecuencia de oscilación externa. Para los microcontroladores PIC está frecuencia no coincide con F_{osc}, ya que para mantener la compatibilidad con los diseños originales es necesario dividirla por cuatro:

$$F_{\text{int}} = \frac{F_{osc}}{4}$$

2. Funcionamiento del Timer0

El Timer0 es un dispositivo puede funcionar conceptualmente de dos formas: como contador de pulsos externos o como temporizador para calcular intervalos de tiempo.

En el caso que dicha señal provenga del reloj interno de instrucciones (F_{int}), el Timer0 se utiliza para generar interrupciones periódicas a través de una cuenta programada. Ya que conocemos la frecuencia de funcionamiento y en base a un valor cargado en el contador del timer (lo veremos mas adelante) podemos temporizar eventos.

En el caso que dicha señal sea de una fuente externa al microcontrolador (F_{ext}), es especialmente útil para contar el número de pulsos que dicha señal genera en el tiempo ya que cada pulso de dicha señal incrementa el TMR0.

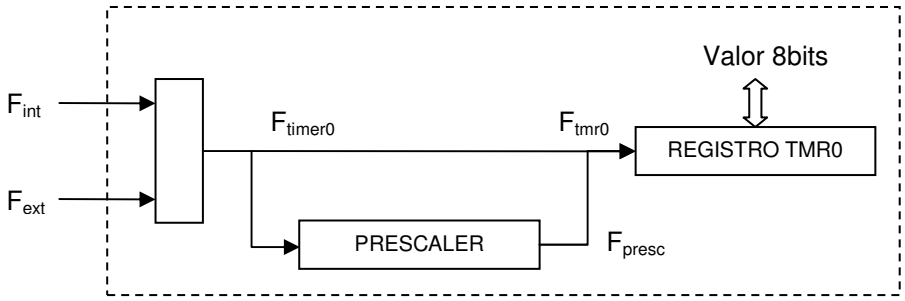
Ejemplo: Queremos medir el intervalo de tiempo entre los pulsos del cardiograma de un paciente.

Usando el timer como temporizador, podemos contar –por ejemplo- periodos de 1 ms, acumulándolos en un registro auxiliar. Cada vez que ocurra un pulso del electrocardiograma, podemos consultar el registro auxiliar y ver cuantos incrementos han sucedido desde el último

pulso y sabiendo que el temporizador cuenta periodos de 1ms, calcular el tiempo transcurrido. Una vez conocido el tiempo, iniciamos la cuenta de los incrementos desde 0 para el siguiente pulso.

2.1. Estructura general del Timer0

El esquema del Timer0 corresponde a la siguiente figura:



Esquema del módulo Timer0

Donde destacamos

- La entrada de reloj F_{timer0} (patilla RA4/T0CKI o el reloj interno de instrucciones).
- Un circuito divisor de frecuencias programable o **prescaler**.
- Un registro contador TMR0 de 8 bits.
- Varios registros de control del Timer0: OPTION_REG e INTCON.

2.2. Entrada de reloj del modulo Timer0

Como anteriormente se indicó, la señal de reloj base de entrada al módulo Timer0 se puede obtener de dos maneras: con un oscilador externo a través de la patilla RA4/T0CKI -F_{ext}-, o bien con el oscilador interno de instrucciones -F_{int}- (recordemos que es un cuarto de la frecuencia de oscilación del microcontrolador). También podemos indicar si el disparo se realizará por flanco ascendente o flanco descendente de la señal.

La configuración la podemos realizar con los siguientes bits de control:

- T0SC (Timer0 Clock Select) en el registro OPTION_REG[5]: Indica el origen del reloj del contador: oscilador interno o señal externa
- T0SE (Timer0 Set Edge) en el registro OPTION_REG[4]: Cuando se selecciona señal externa, indica el flanco activo que se usará.

2.3. El prescaler

El prescaler es un circuito que permite modificar la frecuencia del reloj de entrada del Timer0, dividiendo esta y generando una nueva señal de menor frecuencia a la salida que será

la señal de reloj de entrada al registro TMR0. El prescaler es una facilidad para cuando la señal de entrada es demasiado rápida para nuestros propósitos y necesitamos ralentizarla.

Nota: El prescaler también se puede utilizar con el registro WDT (Watch Dog Timer) del PIC, pero en este caso recibe el nombre de postescaler ya que se usa a la salida del WDT, no pudiendo estar asignado a la vez al Timer0 o al WDT.

El prescaler del registro TMR0 es configurado a través de 4 bits del registro OPTION_REG, y permite dividir la frecuencia de una señal por 1, 2, 4, 8, 16, 32, 64, 128 o 256. En caso de utilizar un divisor por 1, la señal de salida es la de entrada sin ningún cambio.

Por ejemplo, si usamos como oscilador externo del PIC un cristal de 4Mhz, entonces el reloj interno de instrucciones funciona a F_{int} = 4Mhz /4 = 1 Mhz. Si el Timer0 usa la señal del reloj interno y la pasamos por el prescaler configurado para una división por 4, la señal a la salida del prescaler será F_{presc} = 250 Khz.

La configuración se realiza con los siguientes bits de control:

- PSA (Post Scaler Assignament) en el registro OPTION_REG[3]: Indica si el postscaler es asignado al WDT (1) o al Timer0 (0).
- PS2:0 en el registro OPTION_REG[2:0]: Indican el valor del divisor a utilizar en el postscaler (consultar datasheet para los valores).

No debemos confundir la frecuencia de trabajo **base** del Timer0, con la frecuencia de trabajo del registro TMR0: la primera es la frecuencia de entrada utilizada por el modulo, mientras que la segunda es dicha frecuencia dividida por el prescaler y que es el reloj del registro TMR0.

Nota de Microchip:

Al alimentar el PIC o despues de un overflow por WDT, el postscaler esta asignado al WDT. Si se asigna el postscaler al Timer0, es posible que ocurra un reset por el WDT (incluso aun deshabilitado). Por ello recomiendo usar clrwdt antes de reasignar el postscaler:

```
clrwdt                  ; borra postscaler y wdt
molw    b'11110001'     ; reloj externo disparado por flanco de bajada
movwf   OPTION_REG      ; prescaler 1:4 asignado al Timer0
```

2.4. El registro TMR0

El registro TMR0 es el corazón del módulo Timer0. Es un registro contador de 8 bits (podemos contar hasta 2^8=256 valores, entre 0 y 255) que **incrementa automáticamente su contenido** en cada oscilación de su señal de reloj $F_{tmr0.}$

Cuando el valor del registro TMR0 pasa de 255 a 0 (0xFF a 0x00), se produce una interrupción por desbordamiento u "overflow". El PIC indica esta situación activando el flag de interrupción por desbordamiento T0IF (Timer0 Interrupt Flag) del registro INTCON. Para ejecutar la rutina de interrupción además deben estar habilitadas tanto las interrupciones

globales como la especifica para el Timer0:

1. Interrupciones globales activadas: bit GIE (Global Interrupt Enable) de INTCON[7] activo.
2. Mascara de interrupción de Timer0 activada: bit T0IE (Timer0 Interrupt Enable) de INTCON[5] activo
3. Ocurrió un desbordamiento en el Timer0: bit T0IF (Timer0 Interupt Enable) de INTCON[2] activo.

El proceso en general es muy similar al que se sigue con cualquier otra interrupción: si se dan las tres condiciones el PIC comienza a ejecutar la rutina de servicio de interrupción, deshabilitando estas (GIE=0). La rutina de interrupción debería comprobar el flag T0IF para determinar el origen de la interrupción y, si ocurrió una interrupción por desbordamiento, borrar el flan T0IF (debemos hacerlo nosotros pues no es automático), para evitar que el PIC vuelva a la rutina de interrupción cuando las interrupciones sean de nuevo habilitadas.

A la salida de la rutina de interrupción con la instrucción "retfie" el PIC habilita estas de nuevo (GIE=1).

El registro TMR0 continúa incrementándose **siempre** según dicta su reloj, por lo que si se ha producido un desbordamiento, el registro seguirá contando desde 0. Si interesa que la cuenta comience desde un valor predeterminado, **nosotros debemos precargar** el TMR0 con dicho valor una vez detectado el desbordamiento. Esto se realiza normalmente dentro de la rutina de interrupción.

Ejemplo:

```
TIMER0_IRQ  . . .
            bcf     INTCON, T0IF  ; borra la interrupción actual
            . . .
            movwf   TMR0          ; recarga el Timer0 con un valor
            . . .
            otro codigo de la interrupción
            . . .
            retfie
```

Observaciones:

1. Si el microcontrolador está dormido (mediante una instrucción SLEEP) y se utiliza como señal de reloj del Timer0 el reloj interno de instrucciones, el Timer0 estará desactivado y no se incrementará el contador TMR0 ya que el reloj interno se haya detenido. Por tanto jamás se producirá ninguna interrupción por desbordamiento que permita salir del estado SLEEP.

2. Si la fuente de la señal de reloj del modulo Timer0 es externa, si se puede producir interrupción por desbordamiento, ya que aunque el Timer0 este desactivado, el contador no depende activamente del reloj interno (parado) del microcontrolador sino que se incrementa en cada pulso de la señal externa. En este caso si se puede salir del estado SLEEP a través de la interrupción.

2.5. Sincronización del Timer0

Esta modificación del TMR0 podría plantear algunos problemas si se modifica al mismo tiempo que se incrementa. El PIC resuelve esto internamente, mediante un mecanismo de sincronización de la señal de reloj del TMR0 cuando se escribe un dato en el Timer0:

Se usa un registro de desplazamiento de 2 bis a la salida del reloj del TMR0 de forma que produce un retraso de 2 * Fosc/4 en la señal que incrementa el contador (2 µs con un cristal de 4 Mhz; 1 µ con 8 Mhz…). Esto se traduce en que cuando se escribe un dato en el TMR0, se conecta al reloj interno introduciendo un retraso de 2 cuentas, en los cuales no ocurre nada en el TMR0 y ese tiempo es aprovechado para escribir el dato escrito.

Estas dos cuentas de retraso nos permiten escribir y leer sin interferencias con la cuenta actual del contador.

Una vez asignado el prescaler al Timer0, una instrucción que escriba el Timer0, provocará el borrado del prescaler y sincronizará el reloj con el Timer0.

3. Cálculo de temporizaciones

Supongamos que el módulo Timer0 se va a alimentar con una señal de frecuencia F_{timer0} (puede ser de una fuente externa o el oscilador interno del microcontrolador) la cual pasa por el prescaler generando a su salida una nueva señal de frecuencia:

$$F_{presc} = \frac{F_{timer0}}{prescaler}$$

Esta es la frecuencia del reloj que llega al registro TMR0, el cual parte de un valor inicial V_{tmr0} que se incrementa cada ciclo del reloj. Cuando haya pasado V_{tmr0} * T_{presc} segundos, el TMR0 se pasará de 255 a 0 generando una interrupción por desbordamiento (overflow).

Pero esto no es totalmente correcto: el TMR0 cuenta lo que **falta para llegar al desbordamiento**, no el valor almacenado en él; es decir, si inicialmente el TMR0 vale 8 se cuenta desde 8 hasta el máximo valor del registro, 255 y luego pasará a 0:

8, 9, 10, 11, 12, … , 253, 254, 255, 0

desbordamiento

Es decir se cuentan realmente 256 - V_{tmr0} valores. En general, para un contador de n bits tenemos: 2^n - V_{tmr0} valores. El paso por cero entonces es (2^n - V_{tmr0}) * T_{presc} seg. siendo el tiempo hasta el desbordamiento o time-out del contador es:

$$Time - out_{tmr0} = (2^n - V_{tmr0}) * \frac{prescaler}{T_{timer0}}$$

Y la frecuencia de paso por cero:

$$F_{tmr0} = \frac{1}{Time - out_{tmr0}} = \frac{F_{timer0}}{(2^n - V_{tmr0}) * prescaler}$$

Despejando podemos obtener el valor inicial de TMR0 que genera una interrupción por desbordamiento para un tiempo y prescaler dados:

$$V_{tmr0} = \frac{2^n - (Time - out_{tmr0} * F_{timer0})}{prescaler}$$

4. Ejemplos prácticos

Ejemplo 1: Calcular el mayor retraso o time-out que es posible obtener con el Timer0 usando como señal de reloj la interna de un PIC que tiene un cristal de 4 Mhz.

El mayor retraso se puede obtener con el mayor divisor del prescaler (÷256), un valor 0 para el TMR0 (contará desde 0 a 255 antes del desbordamiento, es decir, 256 incrementos). Como F_{osc}= 4 Mhz el retraso máximo es:

$$\text{Time-out}_{tmr0} = (256 * 256) * (4/ 4\ \text{Mhz}) = 65{,}566\ \text{ms}$$

Ejemplo 2: Elegir V_{tmr0} para generar un retraso de 1.5 ms usando un cristal del 10 Mhz.

$$V_{tmr0} = 2^n - ((\text{Retraso}_{tmr0} * F_{int}) / \text{prescaler})$$
$$V_{tmr0} = 256 - ((1.5\ \text{ms} * (10\ \text{Mhz} / 4)) / \text{prescaler})$$
$$V_{tmr0} = 256 - (3750 / \text{prescaler})$$

Sólo resta dar valores al prescaler hasta obtener un valor adecuado, por ejemplo:

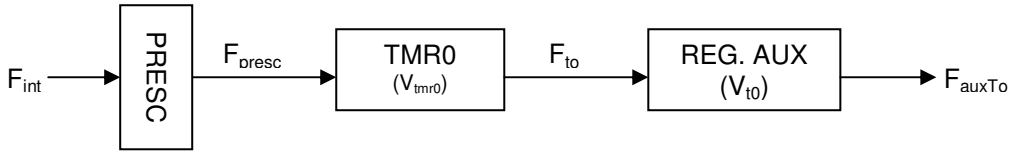
si prescaler = 256, V_{tmr0} = 242 (redondeado) Retraso = 400 ns * 256 * (256-242) = 1,4336 ms
si prescaler = 128, V_{tmr0} = 227 (redondeado) Retraso = 400 ns * 128 * (256-227) = 1,4848 ms
si prescaler = 64, V_{tmr0} = 197 (redondeado) Retraso = 400 ns * 64 * (256-197) = 1,5104 ms
si prescaler = 32, V_{tmr0} = 139 (redondeado) Retraso = 400 ns * 32 * (256-139) = 1,4976 ms
etc.

Podemos elegir prescaler = 64 y V_{tmr0} = 197, dándonos 1.5 ms + 0,0104 ms de error.

Ejemplo 3: Generar un retraso de 1 segundo si tenemos un cristal de 4 Mhz.

Como vimos en el ejemplo 1, con esta frecuencia de oscilación el mayor retraso que podemos generar es de 65,566 ms, lo cual dista mucho de obtener 1 segundo (1000 ms).

Esto se podría arreglar si tuviéramos un contador de mayor número de bits. La solución está en extender el Timer0 con un registro controlado por software, de forma que el comportamiento total sea como si tuviéramos un Timer0 de 16 bits. Dicho registro contará el número de interrupciones por desbordamiento que genera el Timer0 de forma que pase por 0 cuando haya pasado 1 segundo.



Pasos:

1. Escoger un valor para el prescaler:

   Un posible criterio es calcular todas las frecuencias de prescaler que podemos obtener con cada divisor posible. Las frecuencias sin decimales son interesantes al poder encontrar múltiplos de ellas a la salida del TMR0 con mayor facilidad. En general, la elección del valor de prescaler es empírica: depende del problema y la experiencia.

   Elegimos **1:32** que nos da permite una $F_{presc} = F_{int} / 32 = (4Mhz / 4) / 32 = 31250$ Hz

2. Calcular un valor para el TMR0:

   Un criterio adecuado seria aquel valor que genere una frecuencia de paso por 0 del TMR0 tal que esta frecuencia a la entrada del registro auxiliar sea un múltiplo de su valor (sin olvidar que este tiene 256 valores como máximo al ser un registro de 8 bits). De nuevo la elección de un valor que genere una frecuencia múltiplo de un número entre 0 y 255 es más cómodo y preciso para los cálculos al no aparecer decimales. Sin embargo, esto no se podrá conseguir siempre.

   GUIA: La frecuencia de salida del TMR0 es como máximo la F_{presc} (cargando el TMR0 para que interrumpa después de contar un valor).

   Si queremos generar una frecuencia de interrupción del Timer0 de 125 Hz:

   $$F_{t0} = F_{presc} / V_{tmr0} \Rightarrow V_{tmr0} = F_{presc} / F_{t0} = 31250 \text{ Hz} / 125 \text{ Hz} = 250$$

   Es decir, para que cuente nuestros 250 pasos hay que recargarlo con un valor 256-250 = **6**.

   > Este resultado se podría haber obtenido igualmente aplicando sucesivamente la formula calculada teóricamente:
   >
   > $$F_{to} = (4 \text{ Mhz} / 4) / ((256-6)*32)) = 125 \text{ Hz}$$

   En resumen, si cargamos TMR0 con un 6 con prescaler 1:32, el TMR0 pasa por cero 125 veces por segundo generando una interrupción por desbordamiento cada vez.

3. Repetir el proceso para el registro de extensión:

   El proceso es exactamente igual a los cálculos del Timer0, solo que ahora no tenemos ningún prescaler y al frecuencia de reloj del registro auxiliar es la salida del TMR0, es decir, F_{to} = 125 Hz. De esta forma tenemos que calcular un valor del registro de extensión tal que cada 125 pasos por cero del Timer0, este pase por 0 transcurrido 1 segundo.

   $T_{to} * V_{tmr0} = T_{auxTo}$ => $V_{aux} / F_{to} = T_{auxTo}$ => $V_{aux} = T_{auxTo} * F_{to}$ = 1 sg * 125 Hz = 125

   Es decir, el registro cuenta 256-125 = 131 valores. Con una frecuencia de reloj de 125 Hz, nos da una frecuencia de paso por 0 de: 131* (1/125) = 1048 ms = 1,048 seg

En definitiva, **buscamos un valor de frecuencia a la salida del prescaler (F_{presc}) que dividido por el valor del Timer0 (V_{tmr0}) dé una frecuencia de paso por 0 del Timer0 que sea múltiplo de la que queremos generar con el registro auxiliar.** De esta forma tendremos la máxima exactitud al no tener decimales en los cálculos (pero no siempre será posible esto).

Ejemplo de código:

```
T0AUX       equ   0x0d
OPTIONV     equ   b'10000100'   ; reloj interno, flanco subida y prescaler 1:32
T0VALUE     equ   d'6'          ; valor del timer0
T0AUXVAL    equ   d'125'        ; valor del contador virtual

        org   0
        goto  INICIO

        org   4
        goto  IRQ

;****** RUTINA DE INTERRUPCION ******
IRQ
        ; comprobamos desbordamiento de contador virtual
        incf  T0AUX         ; incrementamos reg auxiliar de tiempo
        btfss STATUS, Z     ; contador virtual a 0?
         goto RECARGA       ; si no es 0, recarga el tmr0.

        ; poner aqui el codigo de la rutina

        ; recargamos contador virtual
        movlw T0AUXVAL      ; recarga registro auxiliar
        movwf T0AUX

RECARGA
        ; recargamos Timer0 y borramos la interrupcion
        movlw T0VALUE
        movwf TMR0          ; recargamos el timer0
        bcf   INTCON, T0IF  ; indicamos interrupcion ya servida
        retfie              ; fin de interrupcion

;****** PROGRAMA PRINCIPAL ******
INICIO
        clrwdt              ; recomendacion de microchip antes de reasignar el prescaler
```

```
        bsf     STATUS, RP0  ; cambiamos al banco 1
        movlw   OPTIONV
        movwf   OPTION_REG  ; configuramos timer0
        bcf     STATUS, RP0  ; cambiamos al banco 0

        movlw   T0VALUE      ; cargamos el timer0
        movwf   TMR0
        movlw   T0AUXVAL     ; cargamos contador virtual
        movwf   T0AUX
        bsf     INTCON, T0IE ; habilitamos interrupcion por Timer0
        bsf     INTCON, GIE  ; habilitamos interrupcion general

MAIN    goto    $            ; bucle principal
        end
```