Paolo Rognoni (Paolino)

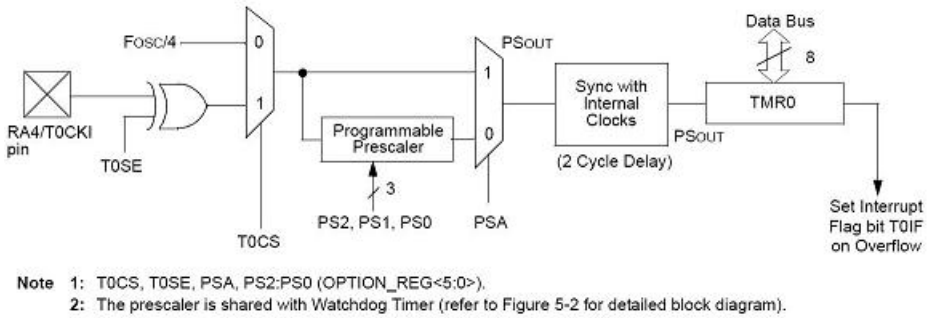
LO HAI MAI REALIZZATO CON UN PIC?
UN APPROCCIO AI TIMER DEI PICMICRO

18 March 2010

Sebbene i timer siano dispositivi molto semplici, in relazione a tutte le periferiche disponibili oggi sui microcontrollori, emerge come, per i principianti, rappresenti una seria difficoltà impostare correttamente i registri associati ai timer per sfruttarne appieno le caratteristiche. In questo capitolo de "**LO HAI MAI REALIZZATO CON UN PIC?**" vediamo come sia possibile avvicinarsi ai timer dei PICMicro, come al solito con un esempio quanto più possibile diretto. Nel seguito verrà illustrato come impostare il **TIMER 0** di un **PIC16F84A**, prendendo spunto da una richiesta pervenuta nel forum di Electroportal.

PIC16F84A e TIMER 0

I PICMicro della famiglia mid-range dispongono di uno o più timer; con i PIC18F, PIC24F, dsPIC e PIC32 il numero dei timer aumenta in modo considerevole. Rimanendo nella famiglia dei PICMicro a 8 bit (PIC12F, 16F e 18F), le strutture dei timer sono a 8 o a 16 bit. Alcuni possono essere adibiti a funzioni specifiche (come ad esempio avviene per TMR2, adibito a base dei tempi per il segnale PWM) ma tutti possono essere comunque utilizzati come timer veri e propri. Per semplicità, vediamo lo schema a blocchi del timer 0 del PIC16F84A. Nonostante questo PIC non sia tra quelli "di piccola taglia" con ampie potenzialità, si è distinto nel tempo per diversi utilizzi cui è stato applicato.



Schema a blocchi di timer 0

Timer 0 presenta un registro a 8 bit in grado di incrementarsi in modo continuo dal valore 0 al valore 255 (0xFF in notazione esadecimale); inoltre, non appena avviene la

transizione da 0xFF a 0x00, **il timer può scatenare un interrupt** del quale tenere conto durante lo svolgimento delle funzionalità. La sorgente che fornisce i TICK al timer perché si incrementi, può essere scelta tra l'oscillatore interno oppure quello esterno e, in questo caso particolare, l'incremento del registro TMR0 può avvenire sul fronte di salita o su quello di discesa del segnale esterno. Il segnale viene fatto transitare attraverso un prescaler, ossia un divisore che è programmabile; i valori slezionabili vanno da 1:2 fino a 1:256. Questo significa che, se considerassimo ad esempio il prescaler impostato a 1:4, ogni 100 impulsi in entrata al prescaler, ce ne saranno 25 in uscita da quest'ultimo. Nel caso in cui non si desideri l'utilizzo di un prescaler, è sufficiente assegnare questa risorsa (il prescaler) ad un altro timer, il Watch Dog Timer.

Il setup del timer 0 avviene con il registro OPTION_REG; i bit di tale registro assumono in seguente significato:

- **T0CS** (bit 5): se posto a 0 seleziona l'oscillatore di sistema (con frequenza Fosc/4) come sorgente per il timer; se posto a 1 i TICK del timer giungono dal pin RA4/T0CKI.
- **T0SE** (bit 4): questo bit va considerato solamente se il bit T0CS è posto a 1, ossia nel caso in cui i TICK del timer giungano dal pin RA4/T0CKI. Se T0SE è posto a 0, l'incremento del timer avviene sui fronti di salita del segnale applicato a RA4/T0CKI, se posto a 1, l'incremento avviene sul fronte di discesa.
- **PSA** (bit 3): se viene posto a 0, il blocco prescaler viene assegnato al timer 0, con la possibilità di selezionare un valore tra otto disponibili; se viene posto a 1, il blocco prescaler viene assegnato al Watch Dog Timer e timer 0 viene automaticamente connesso ai blocchi successivi senza alcuna divisione di frequenza.
- **PS2**, **PS1**, **PS0** (bit 2, 1, 0): questi sono i bit che selezionano il valore del prescaler, secondo dei valori tabulati. Nella figura che segue è riportata la tabella con i valori del prescaler ed il corrispondente valore di questi tre bit.

I bit 6 e 7 non intervengono sulla impostazione del timer 0.

OPTION REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **RBPU**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Il registro OPTION_REG

Il timer e l'interrupt

La gestione più interessante dei timer è quella che avviene mediante l'uso degli interrupt. Per capire meglio come funziona il timer ci si riferisce ad un esempio piuttosto pratico, quello che porterà a realizzare un timer di un secondo, il più preciso possibile. Dovensodi basare sul clock di sistema, tanto più preciso è il quarzo tanto maggiore sarà la precisione della temporizzazione. Eventuali derive termiche inevitabilmente porteranno il timer ad accelerare o rallentare. Se al PIC viene connesso un quarzo con **Fosc = 4MHz**, si ha che **Fosc/4 = 1 MHz**. Suponendo di impostare il prescaler ad un valore unitario (1:1), si ha che il registro del timer si incrementa con periodo pari a 1us e dopo 256 us il registro è saturato. Si potrebero contare un milione di microsecondi per aver realizzato il timer. Ma forse il numero di microsecondi da contare è troppo elevato. Allora si può pensare di impostare in modo oppurtuno il prescaler, ad esempio con un valore pari a 1:128; ciò comporta che il timer si incrementi con cadenza pari a 1MHz / 128 = 7812,5 Hz che corrisponde a 128 us. Quindi, se si pensasse di riempire il registro TMR0 partendo da 0 e arrivando a 255, si otterrebbe un ritardo di tempo pari a 32.768ms, dato dal prodotto di 128us

per 256. Questo numero non è comodo da utilizzare per contare il tempo che ci si è prefissi, non è un sottomultiplo intero di 1 secondo. Un altro escamotage è quello di far contare il PIC da un valore di TMR0 diverso da 0. Se il conteggio partisse dal valore 119, per arrivare a saturare il registro TMR0 è necessario contare 137 TICK il che significa che sono trascorsi 17.5 ms, ottenuti come prodotto di 128us per 137. Se nella routine di interrupt vengono contati 57 intervalli, ciò che si ottiene è un periodo di tempo complessivo pari a 999.52 ms, ottenuto dal prodotto di 17.5 ms per 57. Il valore ottenuto è molto prossimo al valore di un secondo; ora, se si considera che il codice dovrà eseguire le oprazioni di context switch e che l'interrupt ha un tempo di latenza pari a due o tre cicli macchina, sembra che l'obiettivo sia stato raggiunto.

Non resta che abilitare l'interrupt; per far ciò si impiegano i bit **T0IE** (**T**imer **0** **I**nterrupt **E**nable) e **GIE** (**G**eneral **I**nterrupt **E**nable), entrambi nel **registro INTCON**.

INTCON REGISTER (ADDRESS 0Bh, 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
0 = Disables the EE Write Complete interrupt

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

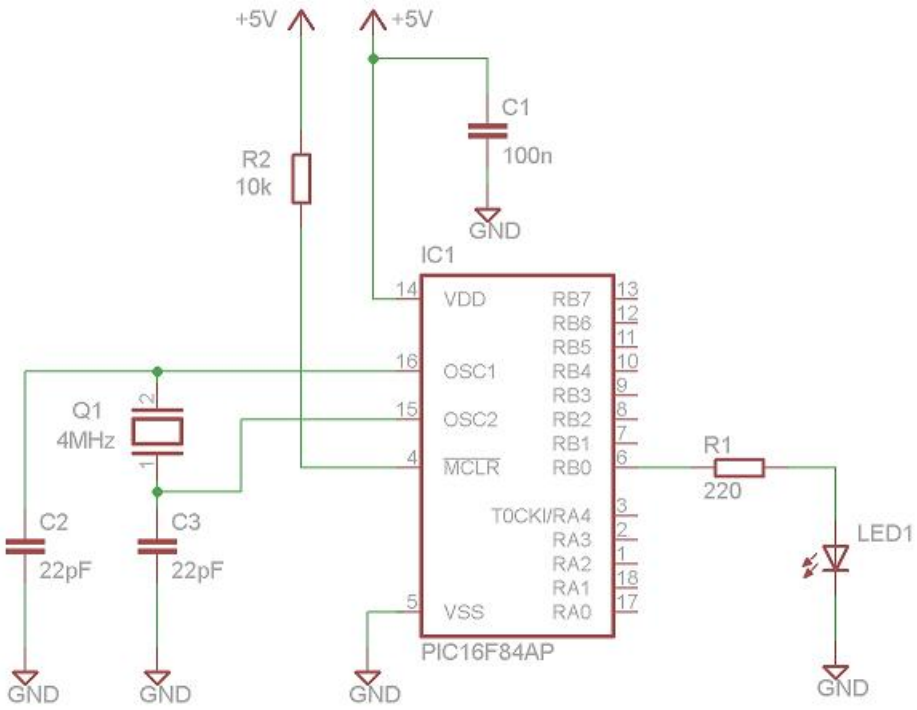
bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Il registro INTCON

Schema elettrico

Lo schema elettrico, facile da realizzare, è riportato nel seguito



Schema elettrico

Il firmware

L'esempio che realizza il timer di un secondo è stato scritto in **assembly** utilizzando l'ambiente di sviluppo integrato MPLAB. Il codice sorgente è stato commentato ampiamente, per modo di capire, contestualmente a quanto scritto in questo articolo, come si comporta il microcontrollore.

```
;
; Esempio di funzionamento del TIMER 0
; su PICMicro PIC16F84A
;


   processor   PIC16f84a
  #include "p16F84A.inc"
  radix      dec
  __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
    errorlevel   -302

; Definizione delle variabili
; Quando questa variabile assume valore 0, è trascorso un secondo
OneSec      equ 0x20
```

```
; Variabile utilizzata per le operazioni di Context Switch sul registro W
W_TEMP      equ 0x21
; Variabile utilizzata per le operazioni di Context Switch sul registro STATUS
STATUS_TEMP equ 0x22

    ORG     0x00
    goto    start

; Interrupt vector
        ORG     0x04

; ISR - Interrupt Service Routine
INTR
        ; Context save, come da datasheet
  movwf         W_TEMP                ; Copy W to TEMP register,
  swapf         STATUS, W         ; Swap status to be saved into W
  movwf         STATUS_TEMP         ; Save status to STATUS_TEMP register

        ; chiamata alla ISR del TIMER 0
  call        T0ISR

        ;Context restore, come da datasheet:
  swapf         STATUS_TEMP,W ; Swap nibbles in STATUS_TEMP register and place result i
  movwf         STATUS                ; Move W into STATUS register (sets bank to orig
  swapf         W_TEMP, F         ; Swap nibbles in W_TEMP and place result in W_TEMP
  swapf         W_TEMP, W         ; Swap nibbles in W_TEMP and place result into W

  retfie

; Routine T0ISR - Risposta all'interrupt di TMR0
T0ISR
  bcf               INTCON,T0IF        ; Pulizia dell'interrupt flag T0IF di timer 0
  bcf     STATUS,RP0
  movlw        1                       ; Decremento di una unità della
  subwf        OneSec               ; variabile OneSec.
  bnz               quit                        ; Se OneSec è diversa da zero, salta a
  call        SetRB0                      ; altrimenti invoca la funzione SetRB0
quit
  movlw        d'119'               ; Si ripristina il valore del registro
  movwf        TMR0                      ; del timer e si ritorna.
  return

SetRB0
```

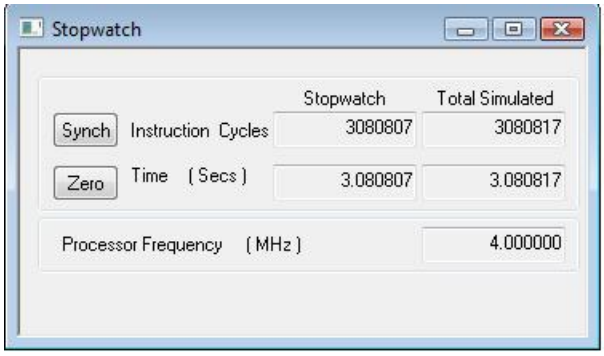
```
  bcf     STATUS,RP0
  movfw         PORTB   ; Lettura della porta PORTB
  xorlw         1           ; esecuzione dell'operazione XOR
  andlw         0x01            ; soltanto con il bit RB0 (toggle del bit della porta).
  movwf         PORTB   ; Scrittura sulla porta PORTB
  movlw         d'57'             ; Si ripristina il contenuto della variabile
  movwf         OneSec  ; OneSec per il conteggio e si ritorna.
  return

start
  bsf     STATUS,5
  movlw   0x00
  movwf   TRISB
  bcf     STATUS,5
  clrf        PORTB
  call        SetupTMR0 ; Chiamata alla routine di setup di TIMER 0

inizio
  goto        inizio        ; ciclo infinito, in attesa dell'interrupt...

; Routine di setup di TIMER 0
; Il timer 0 viene impostato con prescaler 1:128 (registro OPTION_REG)
; Inoltre si abilita la risposta all'interrupt (registro INTCON)
SetupTMR0
  bsf               STATUS,RP0

  movlw         b'10000110'  ; TIMER 0 con prescaler 1:128
  movwf         OPTION_REG ; e sorgente di TICK interna (Fosc/4)

  movlw   b'10100000'         ; abilitazione dell'interrupt, agendo sui bit
  movwf         INTCON        ; GIE e T0IE del registro INTCON

  bcf               STATUS,RP0

  movlw         d'119'        ; TMR0 non viene fatto contare da 0, bensì
  movwf         TMR0                ; da un valore diverso (in questo caso 119)

  movlw         d'57'                ; la variabile OneSec viene inizializzata
  movwf         OneSec        ; al valore 57
  return

; Fine del programma
```

```
END
```

Il codice va compilato all'interno di MPLAB, avendo cura di realizzare un ***progetto***; questo aspetto è molto importante in quanto consente di simulare il firmware con MPSIM.

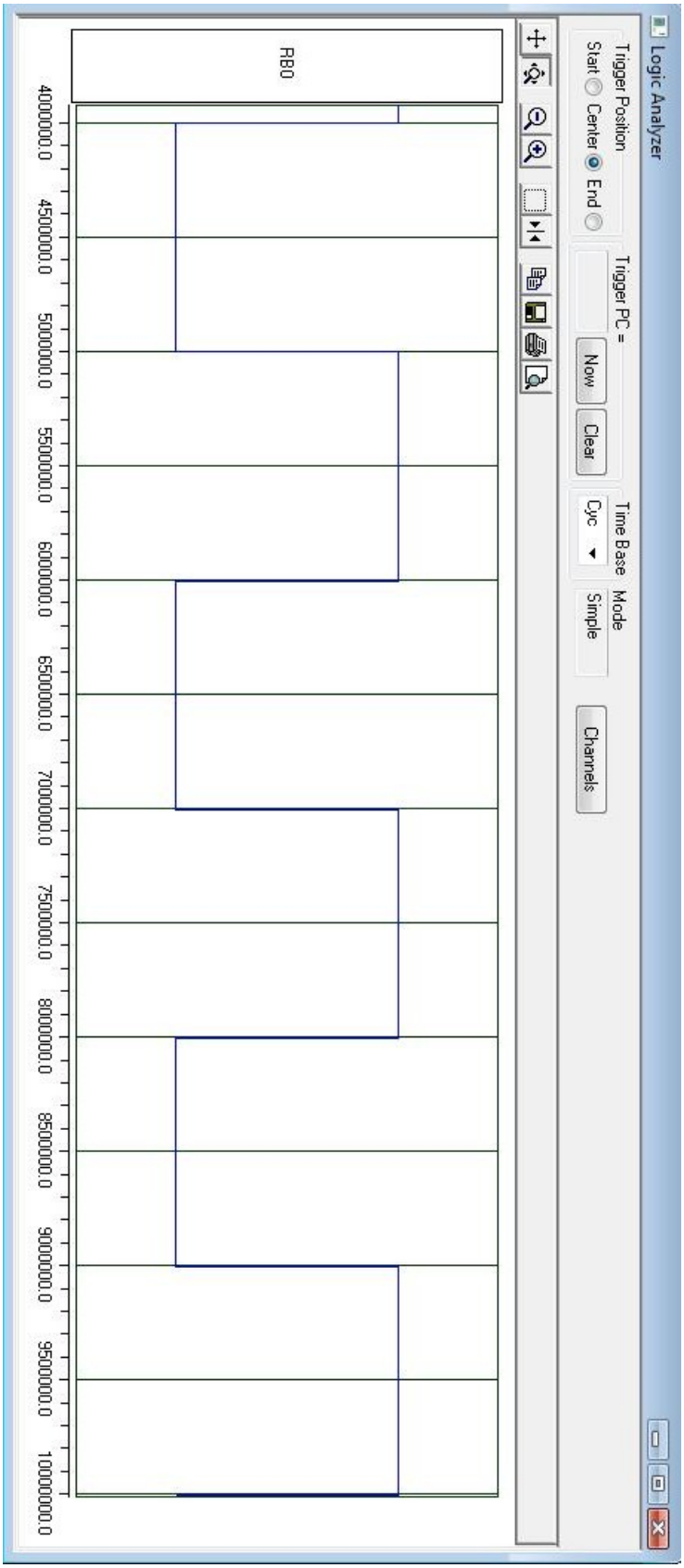
La simulazione

L'ambiente di sviluppo MPLAB contiene al suo interno MPSIM, attivabile dal menu **Debugger -> Select Tool -> MPLAB SIM**. Con questo strumento è possibile simulare il comportamento del firmware. In particolare due sono gli strumenti che in questo caso vengono in aiuto: uno è STOP WATCH, un vero e proprio cronometro che, se fissati opportunamente i break point lungo il codice sorgente, permette di misurare con estrema precisione la durata di routine.



Dati rilevati da StopWatch

Il secondo strumento utilizzato è il logic analyzer, un vero e proprio analizzatore di stati logici per quanto concerne la simulazione. Mediante il logic analyzer si è potuto constatare l'andamento di RB0 e verificare che il firmware andasse ad eseguire in modo corretto le istruzioni, come previsto.


Logic Analyzer
Trigger Position
Start
Center
End
Trigger PC =
Now
Clear
Time Base
Cyc
Mode
Simple
Channels
RB0
4000000.0
4500000.0
5000000.0
5500000.0
6000000.0
6500000.0
7000000.0
7500000.0
8000000.0
8500000.0
9000000.0
9500000.0
10000000.0

Comportamento simulato del pin RB0

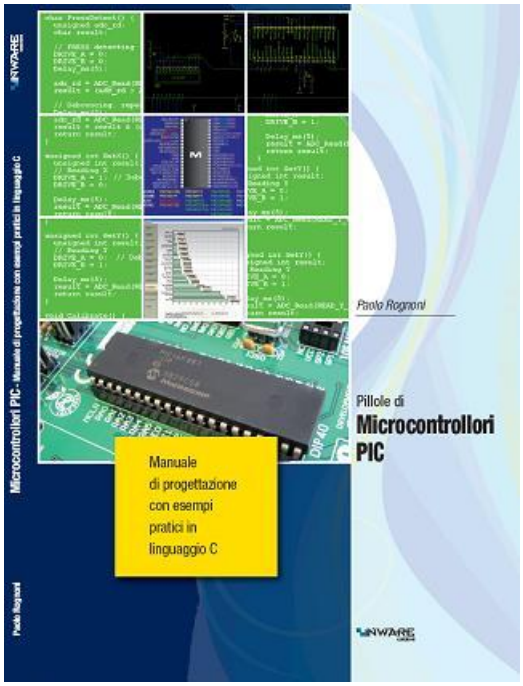
Come si può notare, l'andamento di RB0 è come quello stabilito dai calcoli effettuati in precedenza. Volendo migliorare gli aspetti di misura del tempo sarebbe utile compensare il comportamento del timer affinché le derive dovute agli errori di conteggio restino non trascurabili.

E con altri PIC...?

Se, aprendo il cassetto dei componenti elettronici si scoprisse che il (glorioso) PIC16F84A non c'è ma al suo posto ci sono altri PICMicro a 8 bit, non è il caso di disperare. L'esempio proposto, adattando opportunamente il codice assembly scritto, è facilmente eseguibile da altri PICMicro, come il PIC16F628A, PIC16F819, PIC16F684A, PIC16F876A, ecc.

Riferimenti

- **Ambiente di sviluppo MPLAB**: http://www.microchip.com/mplab
- **Datasheet PIC16F84A**: http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf
- **Collana "LO HAI MAI REALIZZATO CON UN PIC?"**:
  - **Il contamarce**
  - **Una sorpresa musicale per Babbo Natale**
  - **Una tecnica antirimbalzo**
  - **Il dado elettronico**
- **Pillole di microcontrollori PIC**:



Pillole di Microcontrollori PIC.JPG

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Paolino:picmicrotimer"