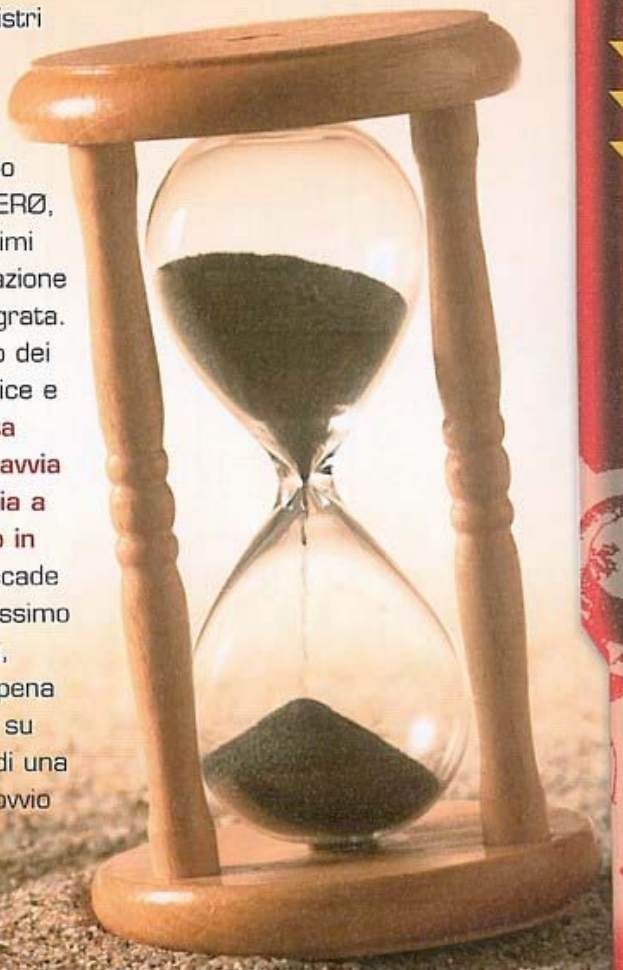
DIAMO TEMPO AL TEMPO

In questo Workshop osserviamo il funzionamento del timer integrato numero Ø e sperimentiamo come utilizzarlo per generare interrupt utili ogni qual volta si vogliano ottenere eventi periodici.

Nel fascicolo precedente abbiamo parlato di dispositivi integrati ai microcontrollori citando, in particolare, il **TIMERØ** (o 'timer numero zero'). Ma cos'è un timer e a cosa serve sul piano pratico? **Un timer** (letteralmente 'temporizzatore') **è un dispositivo digitale composto da un contatore binario a n-bit che incrementa automaticamente il suo stato sulla base di un segnale digitale di clock ricevuto in ingresso.** I timer sono, di conseguenza, paragonabili a variabili numeriche in grado di autoincrementarsi a ogni impulso del clock di controllo, ed esattamente come le normali variabili possono essere inizializzati attribuendo loro il valore numerico da cui si desidera che inizino il conteggio.

I TIMER DEL 16F628 ›››

Passiamo ora a osservare più nello specifico i timer del microcontrollore 16F628, che abbiamo usato nel corso di questi Workshop. Il PIC in questione è dotato di **tre differenti temporizzatori** chiamati rispettivamente **TIMERØ, TIMER1 e TIMER2**; tre dispositivi che differiscono sotto il profilo **dell'architettura**, delle **funzionalità** e delle **modalità di utilizzo** (ad esempio il TIMERØ e il TIMER2 hanno architetture basate su un registro a **8 bit**, mentre il TIMER1 lavora a **16 bit** con l'aiuto di due sottoregistri a 8 bit, ma questa non è la sola differenza).
In questo Workshop, in particolare, ci dedicheremo a un primo studio del TIMERØ, utile a poter condurre i primi esperimenti di programmazione con questa periferica integrata.
In linea di principio l'utilizzo dei timer è abbastanza semplice e facile da capire: **si imposta un valore di partenza, si avvia il conteggio e il timer inizia a incrementare il suo stato in modo ciclico.** Ma cosa accade quando si raggiunge il massimo valore rappresentabile? Sì, perché come abbiamo appena detto, i timer sono basati su un registro che è dotato di una quantità finita di bit ed è ovvio che tale dispositivo sia in grado di rappresentare **solo un range limitato di numeri** (0-255 con 8 bit e 0-65535 con 16 bit). **Al superamento del valore numerico più alto il timer genera un cosiddetto 'overflow'** (letteralmente, in inglese, 'straripamento'), che può essere notificato e intercettato dall'unità



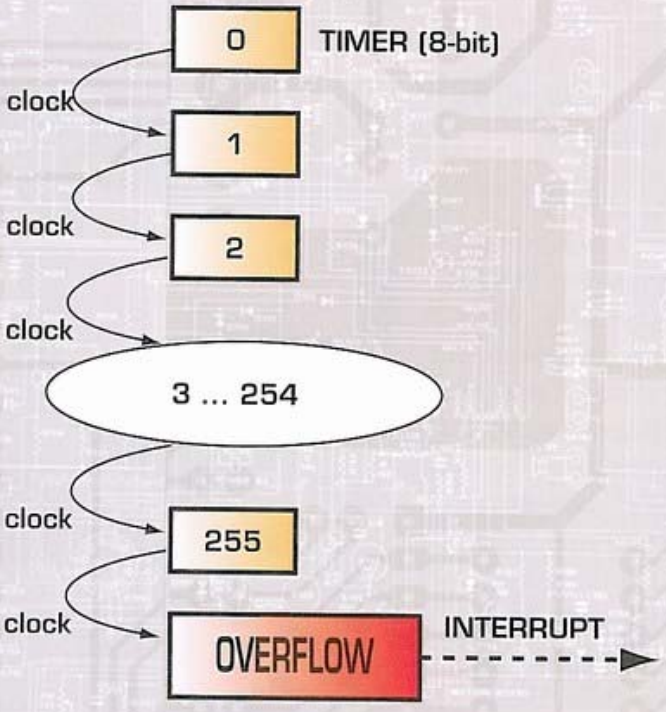
di elaborazione del PIC sotto forma di interrupt, con il conseguente avvio di una apposita funzione di risposta. Ecco che entra in gioco la possibilità di eseguire procedure (descritte nell'implementazione della funzione di risposta agli interrupt) a cadenze di tempo regolari. Sfruttando la generazione degli interrupt da parte dei timer, infatti, non sarà più il programma principale a doversi far carico di temporizzare le operazioni (come abbiamo sperimentato nei primi esempi con l'uso della funzione 'Delay_us'), ma sarà l'hardware stesso a richiamare periodicamente l'esecuzione della nostra funzione di risposta all'interruzione. Una descrizione riassuntiva e schematica del funzionamento di un generico timer a 8 bit è contenuta all'interno del box che trovi in questa pagina.

SCEGLIERE LA CORRETTA TEMPORIZZAZIONE ›››

Dopo ciò che abbiamo visto nei paragrafi precedenti è più che evidente che la frequenza con cui vengono generate le richieste di interrupt è legata al numero di 'step' che separano l'avvio del contatore dal raggiungimento dello stato di overflow. Maggiore sarà il valore di inizializzazione del timer, quindi, e minori saranno gli impulsi di clock (e quindi il tempo) necessari a fargli superare la sua soglia di overflow. Se così fosse sarebbe logico aspettarsi che il massimo periodo di temporizzazione sia calcolabile all'incirca attraverso una funzione analoga a:

$$T_{MAX} = (1 / f_{LAVORO}) \times 2^{n\text{-bit}}$$

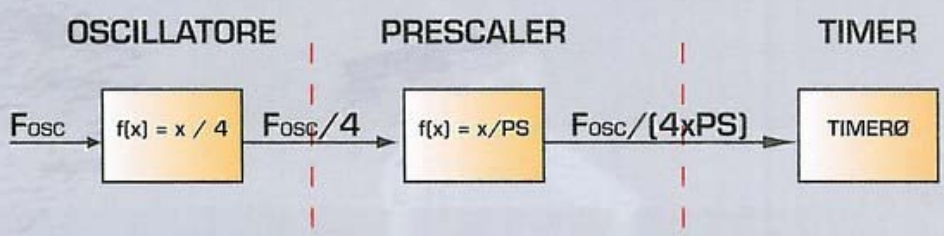
Questo perché, fissata la frequenza di clock di ingresso al timer, il ritardo massimo si ottiene quando il suo contatore deve scandire tutto il range di



L'INTERRUPT DI OVERFLOW DEI TIMER ›››

Uno dei tipici utilizzi dei timer nella programmazione dei PIC consiste nello sfruttare gli interrupt di overflow generati dal temporizzatore per richiamare periodicamente la specifica funzione di risposta. Aiutandoci con lo schema a lato, che mostra la sequenza di funzionamento di un timer a 8 bit, vediamo in che modo si procede generalmente per mettere in atto questo approccio di programmazione. Il primo passo consiste nell'inizializzare il contatore, esattamente come se fosse una variabile (nello schema il valore di partenza del timer è stato impostato a '0', ma tale valore dipende dal tempo che vogliamo far intercorrere tra l'avvio del processo di conta e la generazione dell'interruzione). Successivamente, il timer deve essere abilitato, dopo di che il dispositivo inizierà immediatamente a incrementare in modo automatico il suo valore in risposta agli impulsi di clock. Tale processo di 'conta' proseguirà fino al superamento del valore limite del timer (in questo caso pari a $2^8-1 = 255$), che genererà l'evento di overflow e il corrispondente interrupt. Il vantaggio di questo modello di programmazione, rispetto a quanto abbiamo sperimentato in precedenza utilizzando la funzione 'Delay_us', consiste nella possibilità di scorporare il processo di temporizzazione dal vero e proprio ciclo di vita del programma principale, che può continuare a elaborare informazioni senza dover tenere conto degli eventi periodici del sistema (almeno nella maggior parte dei casi più semplici).

numeri descrivibili da n bit. Nella realtà, tuttavia, **la frequenza del clock del timer non corrisponde a quella di lavoro del PIC, ma può essere configurata agendo su una serie di dispositivi integrati o addirittura impostata utilizzando un clock esterno al microcontrollore.** Per iniziare a familiarizzare con questi concetti, analizziamo un esempio concreto, ossia il **TIMERØ del 16F628** (fai attenzione perché, a seconda del modello di PIC e del timer interno che si sta utilizzando, lo schema logico di funzionamento può differire! È sempre necessario consultare i datasheet). Del TIMERØ, sappiamo dai paragrafi precedenti che è un **timer con architettura a 8 bit,** che **può essere inizializzato come un normale registro** e che **è in grado di generare richieste di interrupt.** Ciò che non abbiamo ancora visto, però, è **come viene temporizzato** e **come può essere configurato.** Una caratteristica importante del TIMERØ è, come prima cosa, quella di **permettere la selezione della sorgente di clock** che si vuole impiegare. Possiamo, in particolare, decidere se utilizzare un **segnale esterno** prelevato tramite il pin **RA4/TOCK** oppure **sfruttare l'oscillatore in uso dal PIC,** legando di conseguenza la frequenza di incremento a quella di temporizzazione del microcontrollore. Proprio in questo secondo caso entra in gioco un ulteriore dispositivo: il prescaler. Il **prescaler** non è altro che è un sistema elettronico programmabile che **opera come 'divisore di frequenza'.** La sua funzione è molto semplice: **riceve in ingresso un'onda quadra a una certa frequenza e ne emette una seconda che mantiene un rapporto di frequenza costante con la prima.** Per dare un esempio numerico, se a un prescaler con rapporto di 1:8 viene posto in ingresso un clock con frequenza di 40 kHz, otterremo una conseguente onda di uscita pari a 1/8 di quella di ingresso, ossia di 5 kHz. La struttura logica della sequenza di divisione della frequenza in ingresso al TIMERØ è mostrata nello schema in alto. Come puoi notare, prima del prescaler è inserito un blocco di riduzione con fattore di 1:4 (nel prescaler entra, cioè, un clock con frequenza pari a 1/4 della frequenza dell'oscillatore del PIC, rappresentata con il simbolo F_{OSC}). Questo è un vincolo fisico implicito nel PIC, legato al fatto che per eseguire le istruzioni di basso livello l'architettura dei PIC richiede 4 cicli di clock. La frequenza massima di lavoro dei timer è, di conseguenza, pari a F_{OSC} (con prescaler che non applica divisione, ossia settato con fattore di 1:1).



Il clock prodotto dall'oscillatore in uso dal PIC arriva al TIMERØ attraverso due passaggi consecutivi: una prima divisione di frequenza di fattore 4 e una seconda, configurabile impostando il rate di divisione del prescaler dedicato.

I REGISTRI INTCON E OPTION ›››

Nei paragrafi precedenti e nel fascicolo scorso abbiamo parlato di interrupt, di sorgenti di clock e di prescaler, ma non sappiamo ancora in che modo è possibile configurare tutti questi elementi. **Il TIMERØ lega il suo funzionamento a due particolari registri del PIC 16F628: il registro INTCON** (vedi box a pagina 4) **e il registro OPTION** (box a pagina 5). Essi sono due registri di configurazione accessibili via software sia in lettura sia in scrittura, attraverso i quali il programmatore può attivare, disattivare e parametrizzare le varie funzionalità del timer (ma anche di altri dispositivi). In particolare, **INTCON è un registro legato principalmente all'attivazione degli interrupt del microcontrollore** e consente di **abilitare o disabilitare alcune delle interruzioni messe a disposizione dal PIC** (overflow del TIMERØ, interrupt esterni ecc.), ma anche di **verificare quali periferiche**

hanno generato una chiamata di interruzione (attraverso una serie di appositi bit). Il registro OPTION, al contrario, contiene principalmente i bit di configurazione del TIMERØ e, in particolare, permette di impostare il fattore di divisione di frequenza del prescaler (per conoscere nel dettaglio il contenuto dei due registri in questione ti rimandiamo al box sottostante e a quello di pagina 5).

Nel prossimo Workshop vedremo come accedere a questi due registri utilizzando l'ambiente di sviluppo mikroC, in modo da configurare il TIMER per generare un semplice evento periodico.

IL REGISTRO INTCON

Il registro INTCON (utilizzabile in mikroC utilizzando l'omonimo simbolo) è uno speciale registro del PIC 16F628 accessibile sia in lettura sia in scrittura, utilizzato per attivare e disattivare le principali fonti di interrupt del microcontrollore. È composto da 8 bit, ognuno dei quali con un particolare nome e adibito a una specifica funzione. Nello schema seguente puoi vedere il significato attribuito a ognuno degli 8 bit del registro.

INTCON

GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

bit 7 ← bit 0

Bit 7 **GIE:** (*Global Interrupt Enable*) se '1' abilita l'utilizzo degli interrupt, se '0' li disabilita.

Bit 6 **PEIE:** (*Peripheral Interrupt Enable*) se '1' abilita gli interrupt periferici, se '0' li disabilita.

Bit 5 **TOIE:** (*TMR0 Overflow Interrupt Enable*) se '1' abilita la generazione dell'interrupt di overflow del TIMERØ, se '0' la disabilita.

Bit 4 **INTE:** (*RB0/INT External Interrupt Enable*) se '1' abilita il pin RB0/INT a generare una chiamata di interrupt sulla base del segnale logico in ingresso. Se '0' disabilitato.

Bit 3 **RBIE:** (*RB Port Change Interrupt Enable*) se '1' abilita la generazione di interrupt ogniqualvolta cambi lo stato di almeno uno dei bit RB4, RB5, RB6, RB7. Se '0' disattivato.

Bit 2 **TOIF:** (*Timer 0 Overflow Interrupt Flag*) viene portato a '1' quando si verifica l'overflow del TIMERØ. Va riportato a 0 'manualmente'.

Bit 1 **INTF:** (*External Interrupt Flag*) viene portato a '1' quando viene generato un interrupt attraverso il pin RB4/INT. Va riportato a 0 'manualmente'.

Bit 0 **RBIF:** (*RB Port Change Flag*) viene portato a '1' quando viene generato un interrupt dovuto al cambiamento di almeno uno dei bit RB4/RB7. Va riportato a 0 'manualmente'.

IL REGISTRO OPTION

Il registro **OPTION** è un registro di 8 bit adibito alla configurazione di alcune particolari funzioni interne del **16F628**. In particolare, permette la configurazione del prescaler del **TIMERØ** e del Watchdog Timer, consente di configurare l'interrupt esterno RB4/INT e di attivare le resistenze di pull-up sulla porta B.

Bit 7 **RBPU:** (*PortB Pullup Enable*) Se '0' abilita le resistenze di pullup interne, se '1' le disabilita.

Bit 6 **INTEDG:** (*Interrupt Edge Selection*) Se '1' RB4/INT produce un interrupt nel momento in cui il suo stato logico passa da '0' a '1', se '0' nel passaggio da '1' a '0'.

Bit 5 **TOCS:** (*TIMER0 Clock Source Select*) Se '1', TIMERØ utilizza come clock il segnale entrante dal pin RA4/TOCKI, se '0' l'oscillatore interno.

Bit 4 **TOSE:** (*TIMER0 Source Edge Select*) Se '1', il timer incrementa il suo valore nel momento in cui il segnale di clock passa da '0' a '1', se '0' nel passaggio da '1' a '0'.

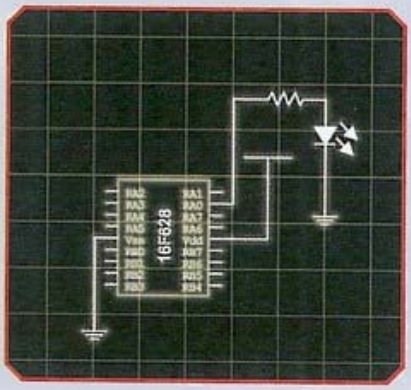
Bit 3 **PSA:** (*Prescaler Assignment*) Se '1' assegna il prescaler al Watchdog Timer, se '0' al TIMERØ.

Bit 2-0 **PS:** (*Prescaler Rate*) Permette di configurare il fattore di divisione di frequenza applicato dal prescaler. Per il TIMERØ valgono le seguenti configurazioni:

Bit 2-0 del reg. OPTION	Fattore di divisione
000	1:2
001	1:4
010	1:8
011	1:16
100	1:32
101	1:64
110	1:128
111	1:256

SCEGLIERE LA GIUSTA TEMPORIZZAZIONE ›››

Vediamo ora di stabilire la configurazione del timer facendo ricorso a un semplice esempio pratico. Immaginiamo di collegare un LED in serie a un resistore al pin RA0 e di **voler produrre un'intermittenza con una frequenza di circa 2 Hz.** Ora prendiamo come riferimento un intervallo di tempo di 1 secondo: in questo intervallo **possiamo identificare una sequenza di quattro eventi che danno origine all'intermittenza** (accendi LED, spegni LED, accendi LED, spegni LED), **distanziati l'uno dall'altro di 0,25 secondi** (ossia 250 millisecondi). Vediamo come procedere. Innanzitutto partiamo definendo la **frequenza di base del PIC:** come negli esperimenti precedenti, anche in questo caso utilizzeremo l'oscillatore interno a 4 MHz. **Il prescaler, di conseguenza, riceverà in ingresso un segnale pari a un quarto della frequenza dell'oscillatore, ossia pari a 1 MHz** (1.000.000 di colpi di clock al secondo, equivalenti a un microsecondo). Dobbiamo ora decidere **come impostare il prescaler**, facendo riferimento ai possibili fattori mostrati nella tabella del box di pagina 5. Se lo impostassimo a un rapporto di riduzione di un **1:128** otterremmo **un incremento del timer ogni 128 microsecondi,** che è anche l'incremento più lento ottenibile sfruttando la divisione di frequenza del prescaler. Questo periodo, però, è troppo breve per poter generare un interrupt ogni 250 msec. Per averne



Lo schema elettrico del circuito che realizzeremo nel prossimo Workshop. I due LED si accenderanno a intermittenza, uno comandato tramite interrupt e l'altro dalla funzione main.

la prova è sufficiente moltiplicare 128 µsec per il numero massimo di step ottenibili con il TIMER0, ossia 256. Infatti:

128 µsec/step x 256 step = 32.768 µsec = 32,768 msec

Possiamo generare un interrupt, quindi, ogni 32,768 msec: troppo poco per le nostre esigenze. Tuttavia, se utilizzassimo **una variabile di conta** che viene incrementata a ogni chiamata di interrupt **potremmo 'contare' il numero di occorrenze degli interrupt,** riuscendo così a 'estendere' il range di funzionamento del timer. Così, ad esempio, se inizializzassimo la variabile di conta a 0, dopo il primo interrupt sarebbero trascorsi poco più di 32,768 msec, dopo il secondo 65,536 msec e così via fino a giungere al settimo, che si verifica a circa 230 ms, e all'ottavo (poco più di 260 millisecondi). Non abbiamo, quindi, una temporizzazione 'esatta', in quanto il periodo da noi cercato cade tra la settima e la ottava interruzione (non sono stati considerati, inoltre, i ritardi legati alle chiamate della funzione di risposta all'interrupt e alle operazioni di incremento della variabile contatore). Proviamo a 'raffinare' ulteriormente il processo **raddoppiando il fattore di riduzione del prescaler,** ossia portandolo a **1:64**, anziché 1:128. Con questo fattore otterremo che ogni ciclo di conta completo del TIMER0 avrà durata pari a:

64 µsec/step x 256 step = 16.384 µsec = 16,384 msec

Riproponendo il ragionamento precedente, in queste condizioni saremo vicini alla temporizzazione desiderata al verificarsi del **15° interrupt** con un tempo che si aggira attorno ai **245 msec** (molto più prossimo ai 250 ms desiderati rispetto al precedente). Per il nostro esperimento possiamo ritenere più che adatti questi parametri che possiamo riassumere con i dati mostrati nella tabella in basso. Nel prossimo fascicolo scriveremo il codice che implementa l'esempio appena descritto, vedendo i passi necessari per configurare il TIMER0.

Prescaler TIMER0	1:64
Valore di inizializzazione TIMER0	0
Numero di chiamate (c.ca 250 msec)	15