CAP. 6

6. IL TEMPO E I CALCOLATORI

Nella sezione precedentemente dedicata alla caratterizzazione del tempo nel cap. 2, ne è stata proposta una modellizzazione astratta, anche se coerente con la filosofia funzionale dei calcolatori digitali. Vediamo ora di affrontare in modo più concreto le problematiche temporali nel funzionamento dei sistemi di elaborazione.
Notiamo anzitutto che i meccanismi di percezione del tempo non fanno parte, in senso stretto, delle normali architetture dei calcolatori alla Von Neumann. Infatti a questo scopo vengono generalmente adottati dispositivi adatti per il conteggio di eventi esterni che solo accidentalmente risultano essere eventi temporali perchè generati da un oscillatore a frequenza regolare e nota.
Nella maggior parte dei casi quindi il sistema di temporizzazione assume la struttura di una unità esterna da interfacciare col calcolatore analogamente a quanto si fa con altre unità periferiche, e questo è il motivo per cui questo capitolo segue le considerazioni generali sulle problematiche di interfacciamento. D'altra parte i problemi di temporizzazione permeano molte delle funzionalità degli interfacciamenti sia digitali (trattati nel prossimo cap.7) che analogici (cap.8) e ciò ha portato ad anticipare queste considerazioni che dal punto di vista dei meccanismi implementativi ricadrebbero nella trattazione del prossimo capitolo.

NOTA.
E' molto importante adottare una buona visione ingegneristica riguardo al tempo, per una corretta attribuzione degli aggettivi *trascurabile, breve, lungo, infinito*, ecc.
Infatti nei calcolatori i tempi di volta in volta significativi spaziano dai nanosecondi dei tempi di risposta dei circuiti digitali, ai microsecondi necessari per l'esecuzione di singole istruzioni, ai millisecondi richiesti per completare l'esecuzione di procedure, e così via. Come per molti altri casi le modellizzazioni traggono vantaggio dalle semplificazioni che assimilano a nulli i tempi trascurabili e ad infiniti quelli relativamente molto lunghi, ma le fasce di queste classificazioni vanno valutate con una particolare sensibilità alle problematiche degli errori e delle loro accettabilità nel particolare contesto applicativo.

6.1 I RUOLI DEL TEMPO NEI SISTEMI DI CALCOLO

Dal punto di vista del sistema di elaborazione il tempo può assumere i seguenti ruoli distinti:

a) - Generatore di eventi temporali (flusso di eventi)
b) - Misura della distanza temporale da un'origine prefissata (valore di stato)
c) - Gestore di temporizzazioni su dispositivi HW

6.1.1 a) - Il tempo come generatore di EVENTI

Il tempo può essere considerato come una sorgente di stimoli "impliciti", non prodotti cioè da particolari fenomeni del mondo esterno ma che tuttavia devono attivare particolari elaborazioni che possono riguardare:

- - campionamenti di stati
- - emissione di informazioni
- - attività di controllo del sistema operativo
- - aggiornamento di variabili di tipo "data e ora"

Gli orologi ("*clock*" o "*timer*") destinati a realizzare questi aspetti devono essere dei dispositivi **"attivi"**, in grado cioè di prendere iniziative autonome ed in particolare di richiamare, generalmente con interrupt, l'attenzione del sistema di elaborazione per notificare gli eventi temporali in modo che vengano attivate le elaborazioni corrispondenti.

6.1.2 b) - Il tempo come valore di STATO

La misura della distanza temporale da un'origine arbitraria può essere considerata come uno **stato** numerico discreto crescente a tempo discreto, presentato in ingresso all'elaboratore da cui può essere letto negli istanti a cui occorra assegnare un valore temporale, come ad esempio:

- - istante di inizio e di fine di un intervallo temporale di cui occorre calcolare la durata;
- - istante di cui si vuole conoscere il tempo da associare (*time stamping*) a determinate informazioni di misure, eventi, ecc.

Gli orologi utilizzabili per questi scopi possono essere dispositivi **"passivi"**, nel senso che devono essere sempre disponibili ad operazioni di lettura da parte del calcolatore, senza assumere in proprio alcuna iniziativa.

6.1.3 c) - Il tempo come condizionatore di dispositivi circuitali

In questo caso il tempo gioca in qualche modo un ruolo simile a quello di generatore di eventi, salvo il fatto, qualificante, che sensibili a tali eventi sono primariamente dei dispositivi elettronici e solo in subordine l'eventuale attivazione di elaborazioni interne al calcolatore.

Gli orologi utilizzabili per questi scopi devono essere "**attivi verso il mondo esterno**", nel senso che devono generare segnali fisici adatti ad essere recepiti dai dispositivi da temporizzare, e possono essere **"attivi rispetto al calcolatore"** se occorre generare un evento, cioè uno stimolo, correlato temporalmente con la temporizzazione dei dispositivi esterni.

Si noti che questo terzo caso potrebbe essere considerato pertinente al mondo esterno e visto solo come generatore di eventi "generici" per il sistema di calcolo, ma varie considerazioni logiche ed implementative suggeriscono in molti casi di considerarlo strettamente legato al calcolatore stesso.

6.2 COMPONENTI TEMPORALI

Il comportamento temporale dei sistemi di calcolo presenta:

- - ritardi tra stimoli e risposte
- - periodo e irregolarità (*jitter*) delle azioni periodiche.

Per quantificare i tempi osservabili nel comportamento esterno del calcolatore è necessario scomporli nelle parti da attribuire a diverse cause e quantificare separatamente distinguendo, come in un diagramma PERT, le attività che possono essere fisicamente svolte in parallelo da quelle a cui sono imposti vincoli di precedenza e i cui tempi sono quindi “additivi”.
Si ricordi che i meccanismi di elaborazione di tipo *pipeline* sfruttano il parallelismo per aumentare la cadenza (*throughput*) senza ridurre però il ritardo di risposta.
Nelle sezioni seguenti analizziamo le componenti temporali che intervengono nell’esecuzione di elaborazioni.

6.2.1 LATENZE

Sono i "tempi morti" tra l'istante in cui si verifica l'evento di stimolo e l'istante in cui si completa l'azione preposta a prendere atto dello stimolo, cioè gli intervalli di tempo durante i quali l'esecuzione dell'azione stimolata non può **iniziare** o non può **proseguire** perchè le risorse necessarie sono dedicate ad altre attività. Le latenze sono quindi tipicamente dovute alla **scarsità di risorse** (tipicamente la CPU) e alla necessità di evitare interruzioni in regioni critiche di altre attività.
Generalmente si **unifica** in un’unica latenza la somma degli intervalli di tempo in cui l’azione stimolata non può iniziare e degli intervalli in cui tale azione viene sospesa e rimane inattiva per concedere le risorse di esecuzione ad altre azioni più prioritarie.

Latenze tipiche sono:
-- latenza di risposta a richieste di interruzione
-- latenza di attivazione di un processo "ready".

Le latenze sono in genere di difficile valutazione **perchè dipendono dal contesto di elementi che si contendono l’uso delle risorse**, dalle politiche di assegnazione di tali risorse (*scheduling*) e presentano un'elevata varianza statistica per la variabiltà dinamica di tale contesto.

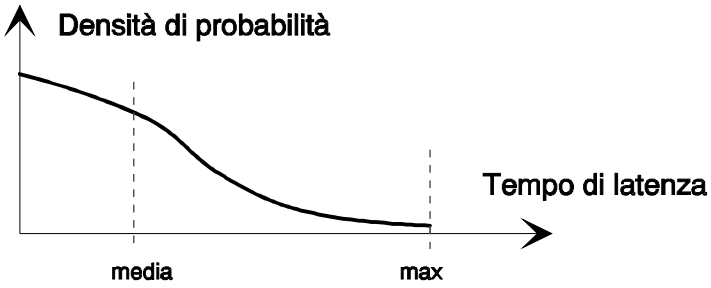

Fig. 6.1 . Tipica distribuzione probabilistica dei tempi di latenza.

Generalmente in sede di stima o di verifica del comportamento temporale vengono calcolate le **latenze medie** pesate con la loro probabilità, per le azioni ripetute frequentemente e non vincolate singolarmente a particolari requisiti temporali, mentre si calcolano le **latenze massime** (*upper bound*) per le azioni per cui è specificata una scadenza.
Quando per un'azione la latenza massima tende ad infinito si parla di *starvation* (letteralmente *morte per inedia*) di quell'azione, cosa che costituisce ovviamente un'anomalia da correggere.
Per definizione l'estremo inferiore (*lower bound*) delle latenze è nullo, mentre gli estremi superiori, a parità di contesto possono differire tra le diverse azioni in competizione per le risorse, ed in generale sono più elevati per le azioni (servizi di interrupt e processi) con priorità meno elevata.
Gli algoritmi di scheduling (v. cap. 10) hanno lo scopo di mantenere entro limiti prevedibili le latenze delle azioni temporalmente critiche.
Nei dispositivi HW, in cui le risorse sono generalmente adeguate (cioè non scarse e condivise), le latenze sono in genere molto ridotte e con precisi limiti superiori.

6.2.2 TEMPI DI OVERHEAD

Sono i tempi che i **meccanismi di base** (HW e soprattutto SW, cioè Sistema Operativo) spendono per la corretta gestione delle attivazioni delle azioni applicative.
Poichè si tratta di impieghi del tempo necessari ma non produttivi, è significativo parlare di **efficienza** dei meccanismi di base come rapporto (classico) tra tempo dedicato alle azioni utili (quelle applicative) e tempo totale.
In molti casi si accettano elevati *overhead* (e quindi basse efficienze) in cambio di prestazioni **più ricche, più robuste e più prevedibili** da parte del sistema operativo.
Poichè le figure responsabili dei tempi di *overhead* (progettista del sistema operativo) e dei tempi di esecuzione delle azioni applicative (progettista dell'applicazione) sono diverse, è corretto tenere concettualmente separati questi contributi, ma spesso per semplicità si preferisce accorparli **attribuendo al tempo di elaborazione netto** (vedere seguito) **di un'azione anche la quotaparte di overhead necessario**, cosa che faremo salvo esplicita dichiarazione contraria.
I tempi di *overhead* sono relativamente stabili e, tranne qualche eccezione, poco dipendenti dal contesto e vengono in genere dichiarati nella documentazione per l'utente del sistema operativo, sia pure in forma indiretta (numero di cicli di clock) o con riferimento ad una specifica piattaforma hardware.

6.2.3 TEMPO DI COMUNICAZIONE

Tra i tempi in gioco per valutare un **ritardo globale**, una tipica componente *additiva* è costituita dai tempi richiesti dalle comunicazioni di informazioni. Il peso relativo rispetto ad altre componenti temporali dipende largamente dal mezzo trasmissivo e dalla tecnica di comunicazione utilizzata, ed in molti casi il tempo di comunicazione è superiore a quello di elaborazione, costituendo il vero “collo di bottiglia” anche rispetto ai ritardi di elaborazione (*throughput*) oltre che per i tempi di risposta. Si noti che in molti casi sono disponibili risorse dedicate alla comunicazione che possono lavorare in parallelo alla CPU, consentendo quindi un effetto “*pipeline*” nel qual caso il tempo di comunicazione non è additivo rispetto al *throughput*, ma solo rispetto al *ritardo* delle risposte.

E’ significativo individuare due componenti dei tempi di comunicazione.

- TEMPO IMPIEGATO PER IL PROTOCOLLO

Si tratta di tempi, di tipo "*overhead*", necessari per la corretta conduzione delle operazioni di comunicazione, che possono comprendere:

- meccanismi di accesso al mezzo trasmissivo e sincronizzazione
- meccanismi di gestione ed instradamento
- meccanismi per rilievo e correzione di errori

- TEMPO IMPIEGATO PER IL TRASFERIMENTO

Sono i tempi effettivamente dedicati a trasferire le informazioni "utili" dalla sorgente alla destinazione. Questa componente diventa preponderante rispetto alla componente di *overhead* per il protocollo (aumentando con ciò l'efficienza) se le informazioni utili sono organizzate in "blocchi" di dimensioni sostanziose.
Questi tempi sono prevedibili con buona approssimazione una volta note la quantità di informazione da trasmettere e la velocità (*transfer rate*) del canale trasmissivo.

6.2.4 TEMPO DI ELABORAZIONE NETTO

Il tempo di elaborazione è una componente essenziale. Nell’ambito del tempo totale richiesto per completare un’elaborazione è in genere utile distinguere la componente ineliminabile costituita dal tempo effettivamente dedicato dalla CPU ad eseguire le istruzioni dell'algoritmo implementato, che chiamiamo tempo di elaborazione "**netto**", da altre componenti, come latenze, overhead e tempi di comunicazione sopra citati, che si sommano per portare al tempo effettivamente trascorso per completare l'elaborazione che chiamiamo tempo di elaborazione "**lordo**".

La valutazione del tempo netto di elaborazione presenta una difficoltà molto dipendente dai costrutti di programmazione utilizzati (costrutti condizionali, cicli, ecc.) e per essa sarebbe auspicabile un supporto da parte dei compilatori.
In genere questi tempi sono misurati a posteriori eseguendo le elaborazioni sotto il controllo di strumenti SW detti *profiler*, che rilevano i tempi totali e le distribuzioni relative alle varie elaborazioni.

6.3 ENTITA' RESPONSABILI DEI TEMPI DI ESECUZIONE

- **Hardware.**
  Gli elementi rilevanti sono il tipo di CPU, la frequenza del suo clock, gli stati di *wait* introdotti negli accessi alle memorie, ecc. Cause di **variabilità** dei tempi sono l'eventuale concomitanza di accessi in DMA, il *prefetch* di istruzioni e l'uso di memorie *cache*.
- **Supporto run-time.**
  Gli elementi principali sono i tempi di esecuzione delle primitivedel Sistema Operativo, le politiche di *scheduling* e la gestione delle risorse, con notevole impredicibilità nel caso di gestione di memoria virtuale, *garbage collection*, ecc..
- **Algoritmi applicativi.**
  Per queste elaborazioni un elemento di incertezza tipico è costituito dalla presenza di **cicli a condizione**, la cui durata di esecuzione può dipendere anche in modo complesso dai valori variabili dei dati.
- **Sottosistema di comunicazione.**
  I tempi di comunicazione dipendono dal volume di traffico, dalle politiche di risoluzione dei conflitti di accesso al mezzo trasmissivo e dalle eventuali necessità di ritrasmissione dei messaggi ricevuti con errori a causa dei disturbi ambientali.

Si noti che i tempi risultanti da interazioni di diverse funzionalità (latenze, protocolli, ecc.) presentano una varianza molto maggiore e risultano pù complicati da calcolare: spesso si preferisce rilevarli "in vivo" con opportuni *benchmark* sul prototipo.

E' infatti tipico che vengano valutate con buona approssimazione le prestazioni "locali" di singole entità (HW e SW), mentre le prestazioni complessive, dette anche "*end-to-end*", vengono per lo più stimate grossolanamente e solo talvolta limitate superiormente. Ciò è contrario allo spirito dei sistemi *hard real-time* per cui è fondamentale la predicibilità. Se non si può dimostrare l'accettabilità di queste incertezze nella particolare applicazione, esse dovranno essere ridotte con opportune semplificazioni che riducano le cause di variabilità.

6.4 ERRORI DI TEMPORIZZAZIONE

I tempi coinvolti nell'esecuzione delle azioni, che abbiamo brevemente analizzato nelle sezioni precedenti, hanno un impatto sulle **precisioni delle operazioni temporali**. Mentre torneremo su queste considerazioni in relazione a particolari situazioni concrete, accenniamo qui brevemente ad alcuni aspetti generali. Si noti che trascuriamo per ora gli errori di precisione e di quantizzazione dell'orologio adottato, considerando esclusivamente gli errori **aggiuntivi** dovuti ai ritardi di esecuzione di azioni.

6.4.1 Rilievo del tempo assoluto di un evento

La concatenazione dei tempi tra il verificarsi di un evento e l'acquisizione del tempo relativo è la seguente:

Ta - **latenza** del **rilievo** dell'evento (variabile)
Tb - **tempo di esecuzione** del **riconoscimento** dell'evento (costante)
Tc - **latenza** dell'operazione di **lettura** del valore del tempo (variabile)
Td - **tempo di esecuzione** della **lettura** del valore tempo (costante).

Le componenti Ta e Tb si hanno solo per eventi esterni che devono essere recepiti a controllo di programma o ad interrupt.
La componente Tc può essere ridotta a zero se si rende non interrompibile la sequenza di operazioni di lettura del tempo, ad esempio disabilitando gli interrupt.
La componente Td è ineliminabile, ma piuttosto costante e breve.
Gli errori temporali, a prescindere da quelli dovuti alla *rappresentazione discreta* del tempo, sono sempre per eccesso e si hanno le seguenti situazioni.
L'errore temporale **minimo** è quindi pari a:

per eventi esterni Tb + Td
per eventi interni Td

L'errore temporale **massimo** è:

per eventi esterni Tamax + Tb + Tcmax + Td
per eventi interni Tcmax + Td

6.4.2 Misura dell'intervallo tra due eventi

La misura di un intervallo di tempo viene generalmente effettuata rilevando il tempo assoluto dell'evento di inizio, quello dell'evento di fine e calcolandone la differenza. Tutte le componenti temporali costanti, nel rilievo dei tempi assoluti, contribuiscono a formare il ritardo della disponibilità della misura, ma non ne inficiano il valore i **cui errori dipendono solo dalle componenti variabili** che sono tipicamente costituite dalle latenze.
Gli errori di misura massimi per eccesso e per difetto (cioè in modulo) sono quindi pari alla somma delle latenze massime (Tamax e Tcmax visti precedentemente) dato che le latenze minime sono nulle.

6.4.3 Esecuzione a tempo prefissato di un'azione

La concatenazione dei tempi tra l'evento temporale e il completamento dell'azione è la seguente:

Ta - latenza del rilievo dell'evento temporale
Tb - tempo di esecuzione del riconoscimento dell'evento temporale
Tc - latenza dell'azione
Td - tempo di esecuzione dell'azione.

Le componenti Ta e Tb sono relative ai meccanismi di *percezione* del tempo ad interrupt o, più raramente, a controllo di programma.
La componente Tc può essere ridotta a zero se si rende non interrompibile la sequenza di operazioni, ad esempio disabilitando ulteriori interrupt. Quando non sia conveniente fare ciò e per di più l'azione non venga eseguita nell'ambito della risposta all'interrupt ma da parte di un processo da attivare con sincronizzazione di secondo livello, questa latenza può avere valori massimi anche piuttosto lunghi (attesa della CPU da parte del processo), in dipendenza del contesto applicativo, e diventare così la componente preponderante dell'errore.
La componente Td è ineliminabile.
Gli errori sono **sempre per eccesso** (ritardo) e vanno da

un **minimo** di Tb + Td
a un **massimo** di Tamax + Tb + Tcmax + Td

6.5 REQUISITI DEI TEMPORIZZATORI

I requisiti dei temporizzatori influenzano la scelta dei meccanismi di base, da cui dipendono le componenti di errore presentate nel paragrafo precedente, e la scelta delle granularità temporali da cui dipendono gli errori di quantizzazione.
E' buona regola ingegneristica che la granularità dei valori che una grandezza può assumere sia dello stesso ordine di grandezza (o poco superiore) delle incertezze su tali valori. Utilizzeremo quindi le valutazioni sulle incertezze nei diversi ruoli del tempo individuati nel precedente paragrafo 6.1, per scegliere le corrispondenti granularità.

A) - Le attivazioni di processi e le loro elaborazioni mettono in gioco l'esecuzione dell'ordine delle centinaia di istruzioni macchina, quindi per i tempi utilizzati per il "risveglio" di attività adotteremo una granularità di *percezione* del tempo dell'ordine del millisecondo.

B) - Il tempo fisicamente necessario per leggere il tempo da associare ad un evento (ad es. l'esecuzione di una certa azione) a scopo di misura, è quello occorrente per eseguire alcune istruzioni macchina, e per esso adotteremo perciò una granularità di *misura* del tempo dell'ordine dei 10 microsecondi.

C) - I tempi di attivazione diretta di dispositivi circuitali sono dell'ordine delle centinaia di nanosecondi (per alcuni dispositivi anche molto meno) e quindi tali tempi avranno una granularità che, se le specifiche lo richiedono, potrà scendere anche al centinaio di nanosecondi. Per gli errori temporali di quantizzazione, dovuti al meccanismo temporale discreto, si ricordi quanto detto nel cap. 2.

Per chiarire questi concetti si può notare una **similitudine** significativa con l'uso del tempo da parte delle **persone**.

A) - Il ruolo "attivo" del tempo corrisponde all'uso di una funzione di "sveglia", per cui è accettabile una granularità di 1..5 minuti, compatibile con il fatto che questo è circa il tempo che impieghiamo per mettere in moto la nostra attività al risveglio. Mentre nel caso del risveglio la latenza è nulla, diverso è il caso della sveglia che suoni mentre stiamo già compiendo azioni non interrompibili.

B) - Il ruolo "passivo" di misura è basato sull'osservazione di una lancetta nel quadrante di un orologio rispetto alla quale la nostra capacità di cogliere l'indicazione istantanea giustifica una graduazione in secondi che è la granularità di cui siamo in grado di tener conto.
Una variante di questo uso è costituita dal cronometro con start e stop a pulsante, e successiva lettura "con calma" che consente di apprezzare 0.1..0.2 secondi, che sono il tempo di reazione muscolare della mano. Si noti che in questa similitudine la riduzione delle latenze viene ottenuta dedicando tutta la nostra attenzione all'operazione di cronometraggio e disabilitando quindi di fatto altre azioni.

C) - L'uso di attivazione diretta di dispositivi potrebbe essere paragonato al timer di un forno da cucina in cui l'azione di spegnimento viene direttamente effettuata dal timer, mentre un suono richiama l'attuazione, differibile con una certa larghezza, delle operazioni conclusive da parte dell'uomo. La granularità richiesta da quest'ultimo esempio è grossolana in dipendenza delle tempistiche relative a operazioni di cottura.
Un altro esempio, stavolta con granularità temporale molto più fine, è costituito dal meccanismo di temporizzazione che gestisce l'otturatore di una macchina fotografica, in grado di effettuare operazioni con temporizzazioni che la nostra mano non potrebbe, e che ci avverte con un "click" che la fotografia è conclusa.

6.6 TIPI DI TEMPORIZZAZIONI ATTIVE

Per la classe di temporizzazioni che abbiamo definite attive e generatrici di eventi, è interessante accennare ad una tipica classificazione che distingue diversi tipi di timer.
Nel seguito utilizzeremo i seguenti simboli.

ST evento di *start*
V_S evento di *start valido*
RES evento di *reset*
OUT stato della temporizzazione (attivo = TRUE)
T valore (parametrico) del tempo di temporizzazione

6.6.1 Temporizzazione a durata

Si tratta di una temporizzazione di tipo "monostabile" attivata da un evento (subìto) *trigger* e conclusa dall'evento (prodotto) di *fine attivazione* dopo l'intervallo di tempo prefissato.

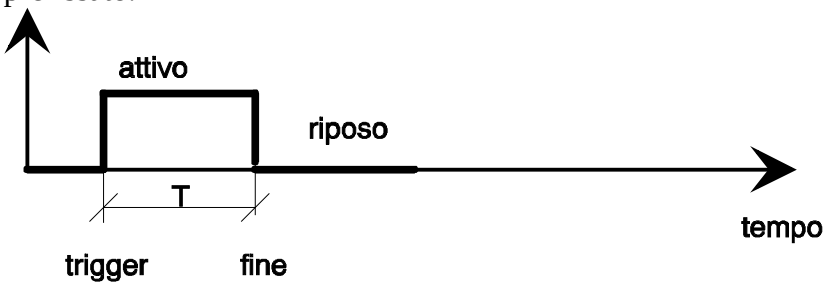

Fig. 6.2 - Temporizzazione a "durata".

Sono detti "*retriggerabili*" i temporizzatori per cui ogni nuovo evento di *trigger* determina la ripartenza da zero del conteggio del tempo, prolungando così l'attivazione.

SPECIFICA **TRIO** DI TEMPORIZZAZIONE NON RETRIGGERABILE
Definizione di start valido V_S

$\text{Always}(V\_S \leftrightarrow ST \wedge \neg OUT)$

Definizione del comportamento

$\text{Always}(OUT \leftrightarrow \text{WithinP}_{ei}(V\_S, T))$

La formula dice che l'uscita è attiva se nel precedente intervallo (inizio escluso - fine inclusa) di tempo T si è varificato un evento *valid start*.

SPECIFICA **TRIO** DI TEMPORIZZAZIONE RETRIGGERABILE
Definizione di start valido

$\text{Always}(V\_S \leftrightarrow ST)$

Definizione del comportamento

$\text{Always}(OUT \leftrightarrow \text{WithinP}_{ei}(V\_S, T))$

La formula dice che l'uscita è attiva se nel precedente intervallo (inizio escluso - fine inclusa) di tempo T si è varificato un evento *valid start*, e in questo caso tutti gli eventi di *start* sono anche *valid start*.

6.6.2 Temporizzazione a ritardo

In questo caso l'evento *trigger* non attiva nulla, se non il conteggio dell'intervallo specificato, al termine del quale si verifica la produzione dell'evento *attivazione*.
In genere si prevede un comando di ripristino al quale vengono forniti eventi di *reset* per ottenere la disattivazione con il ritorno nello stato di riposo. Se l'evento di *reset* si verifica prima della scadenza del tempo T non si ha attivazione.

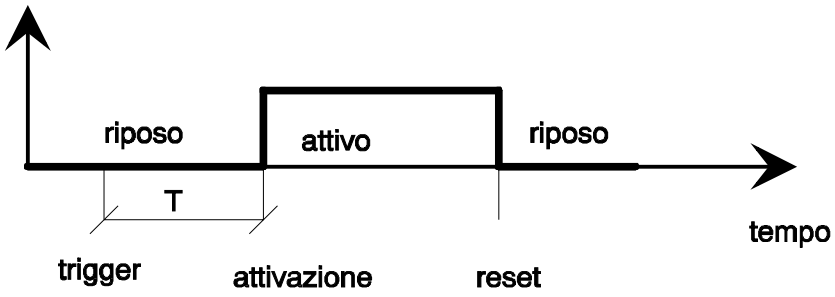

Fig. 6.3 - Temporizzazione a “ritardo”

SPECIFICA **TRIO** DI TEMPORIZZAZIONE A RITARDO

Definizione di start valido V_S

$$\text{Always}(V\_S \leftrightarrow ST \wedge \forall t\ (\text{Past}\ (ST, t) \rightarrow \text{WithinP}\ (RES, t))$$

La formula dice che uno *start* è valido sse gli *start* precedenti, in qualunque tempo si siano verificati, sono stati seguiti da un evento di *reset*.

Definizione del comportamento

$$\text{Always}(OUT \leftrightarrow \exists t\ (\ (t > T) \wedge \text{LastTime}\ (V\_S,t) \wedge \text{Lasted}\ (\neg\ RES, t))$$

La formula dice che l’uscita è attiva sse in un tempo t precedente e maggiore di T si è avuto un *valid start* e nessun *reset* nel frattempo.

Si assume $\text{LastTime} \equiv \text{Past}\ (A, t) \wedge \text{Lasted}\ (\neg A, t)$

6.6.3 Temporizzazione ciclica

I temporizzatori a ciclo continuo dopo l’evento di trigger producono eventi con cadenza regolare e di periodo temporale prefissato T.

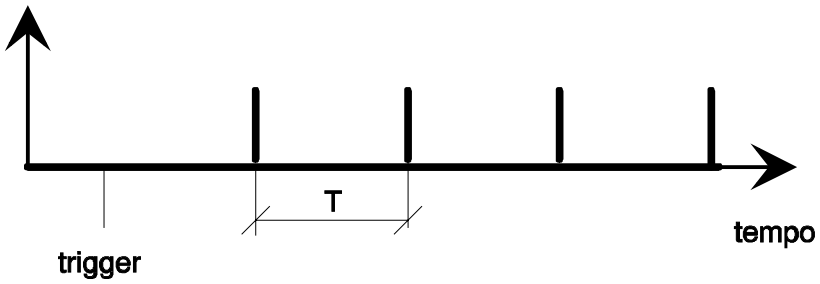

Fig. 6.4 - Temporizzazione ciclica.

In genere questi temporizzatori sono dotati di un comando di *reset* che ne arresta il funzionamento, che riprenderà solo dopo un nuovo evento di *trigger*.

SPECIFICA **TRIO** DI TEMPORIZZAZIONE CICLICA

Definizione di start valido (come nel caso a ritardo)

$$\text{Always}(V\_S \leftrightarrow ST \wedge \forall t\ (\text{Past}\ (ST, t) \rightarrow \text{WithinP}\ (RES, t))$$

La formula dice che uno *start* è valido sse gli *start* precedenti, in qualunque tempo si siano verificati, sono stati seguiti da un evento di *reset*.

Definizione del comportamento

$$\text{Always}(OUT \leftrightarrow \text{Since}(\neg\, RES, V\_S) \wedge \text{Past}(V\_S \vee OUT, T))$$

La formula dice che l'uscita è attiva se dall'ultimo *valid start* non ci sono stati *reset* e se nell'istante passato a distanza temporale T c'è stato un *valid start* o un'attivazione di OUT.

6.7 TECNICHE IMPLEMENTATIVE DI TEMPORIZZATORI

Le tecniche implementative di orologi e temporizzatori nei calcolatori sono varie in dipendenza dei diversi requisiti da soddisfare per diverse categorie di applicazioni, e sono basate sull'integrazione di funzioni HW e SW.
Innanzitutto vengono brevemente presentati i circuiti detti *calendar/clock*, che svolgono il ruolo di permettere di conoscere data e ora.
Successivamente vengono presentati sistemi di temporizzazione adatti per applicazioni real-time, nei tre approcci rappresentativi di tre livelli di complessità (e prestazioni) crescenti.

6.7.1 CIRCUITI CALENDAR/CLOCK

Si tratta di circuiti integrati previsti per svolgere le funzioni di orologio e calendario.
Contengono al loro interno l'oscillatore (generalmente a 32768 Hz) che fornisce il tempo base, eventualmente utilizzando un quarzo esterno, e tutti i contatori necessari per tener conto di secondi, minuti, ore giorni, giorno della settimana, mese e anno. Sono organizzati per tener conto delle diversità dei vari mesi e degli anni bisestili.

In genere le operazioni di impostazione o lettura di data e ora avvengono tramite un opportuno protocollo seriale, che consente di ridurre al minimo il numero di pin necessari (6 o 8) ma rende relativamente lente (centinaia di microsecondi) tali operazioni.
In effetti questi circuiti, con la loro risoluzione temporale di 1 secondo, non sono previsti per effettuare operazioni di temporizzazione, ma solo per letture periodiche da parte del calcolatore quando occorre conoscere data e ora, come ad esempio in occasione della scrittura di file, ecc.

Naturalmente la funzionalità di orologio/calendario è significativa soprattutto se è continua e non interrotta dagli spegnimenti dell'alimentazione del calcolatore su cui esso è montato. A tale scopo questi circuiti sono corredati di batterie in tampone che ne mantengono la continuità di funzionamento. Trattandosi di circuiti in tecnologia CMOS con funzionamento a bassa frequenza l'assorbimento è così ridotto (nanoampere) da consentire un funzionamento per diversi anni anche con piccolissime batterie.
E' interessante e frequente l'abbinamento di questi circuiti con dei dispositivi di memoria RAM, così da condividere le circuiterie di decodifica degli indirizzi senza richiedere ulteriori interfacce dedicate
Per concludere notiamo che questi circuiti spesso convivono nello stesso calcolatore con i timer presentati nei paragrafi seguenti.

6.7.2 IMPLEMENTAZIONI MINIMALI

Le realizzazioni di orologi più semplici sono basate su una sorgente di richieste di interruzione (tick) a cadenza regolare, detta RTC (Real Time Clock) ottenibile, ad esempio, con un oscillatore quarzato e un divisore di frequenza (*prescaler*) che produce un segnale fornito al circuito di generazione di richieste di interruzione.

Tutte le funzioni sono svolte a livello SW dalla routine di servizio delle interruzioni dei tick di RTC. Questa routine è parte essenziale di tutti i nuclei di sistemi operativi (in particolare quelli per il real-time) e svolge almeno le seguenti funzioni:

- aggiorna una variabile (eventualmente aggregata) di tipo "ora" (ed eventualmente "data") che funge da ora corrente per le operazioni di "timbro orario" (*time stamping*) di eventi, e misura di intervalli (ruolo passivo di misura).

- verifica se uno o più degli intervalli temporali relativi ai processi scade, e nel caso effettua le corrispondenti transizioni di stato (da *waiting* a *ready* - vedi cap.10) sui processi interessati (ruolo attivo).

Con questa tecnica le **granularità** di *percezione* e di *misura* **coincidono** e corrispondono al periodo **Tck** dei tick.
La frequenza dei tick è scelta come compromesso tra una fine granularità temporale ed un overhead accettabile, corrispondente ad un tempo di esecuzione medio della routine di servizio inferiore ad un decimo del periodo di tick.
Valori tipici di periodo di tick sono di 1..10 ms per applicazioni real-time, e fino a 100 ms per applicazioni senza particolari requisiti temporali.

6.7.3 IMPLEMENTAZIONI TIPICHE

Come discusso precedentemente, i requisiti di granularità sono diversi per i diversi ruoli del tempo.
Per differenziare la granularità (fine) dei tempi usati come misure rispetto a quella (più grossolana) usata per l'attivazione di processi si adotta in genere un sistema di temporizzazione con due contatori indipendenti, secondo lo schema di fig. 6.5.

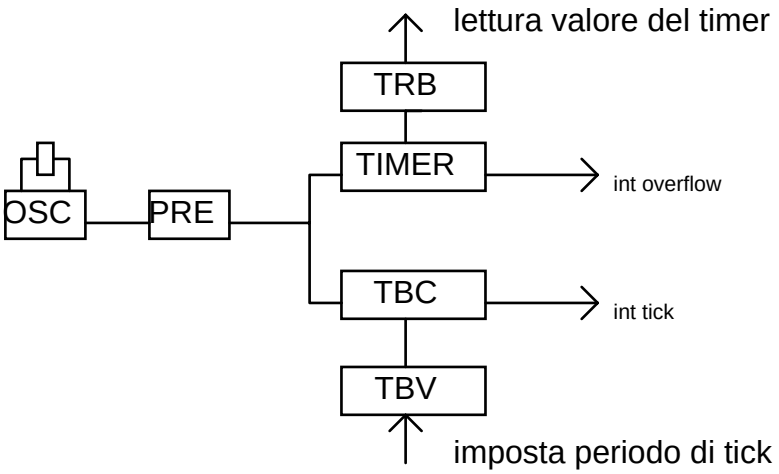

Fig. 6.5 - Schema di sistema tipico per la gestione del tempo.

Nello schema a blocchi di fig. 6.5 si ha:

- **OSC** è un oscillatore di frequenza F (1..20 MHz)
- **PRE** è un divisore di frequenza per 2^K (1..256)
- **TIMER** (per il ruolo passivo di misura) è un contatore binario ciclico di N bit, previsto per generare, se necessario, una richiesta di interruzione ad ogni "*overflow*" e dotato di un registro "buffer" di lettura TRB che cattura (*latch*) tutti gli N bit del TIMER ogni volta che si inizia una operazione di lettura che può comportare il trasferimento di diversi byte: spesso N = 16 ma talora si ha N = 24.

- **TBC** (per il ruolo attivo di stimolo) è un contatore che genera la "base tempo" (Time Base Counter) mediante generazione di una richiesta di interruzione seguita da un ripristino del valore iniziale, ogni TBV conteggi. Il valore TBV è generalmente impostabile in fase di inizializzazione del sistema, in modo da scegliere la frequenza di *tick* più opportuna.

Si noti che per semplicità si è adottata un'unica catena *oscillatore-prescaler* per entrambi i contatori. Questa scelta è comunque ragionevole, dato che spesso si adotta un rapporto di circa 1 a 100 tra granularità temporale del TIMER e dei tick di TBC.

Il comportamento temporale di questo modello, che in realtà è costituito da due orologi, è caratterizzato dai valori seguenti.

- Sezione **TIMER**

TIG = TIR = Granularità = risoluzione $= 2^K / F$ secondi
TIP = Periodo = Range $= TIG \cdot 2^N$ secondi

Per questo orologio si ha
unità di tempo locale utl = TIG
base tempo locale BTL = 1

- Sezione **Time Base**

TBR = risoluzione $= 2^K / F$ secondi
TBP = TBG = Periodo = granularità $= TBV * TBR$ secondi

Per questo orologio si ha
unità di tempo locale utl = TIG
base tempo locale BTL = TBV

Si noti che la precisione di questi valori per entrambi gli orologi è la stessa dell'oscillatore a frequenza F.

Per il meccanismo di ripristino ciclico del contatore TBC rispetto al valore TBV, si trovano generalmente due tipi di soluzioni.

- Ricaricamento automatico (*auto-reloading*) che provvede, a livello HW, al ripristino del contatore, contestuale con la generazione di richiesta di interruzione, utilizzando il valore TBV. Questo valore è stato caricato, con un'operazione di scrittura SW in fase di inizializzazione, sul registro ausiliario apposito.

- Ricaricamento esplicito con un'operazione di scrittura effettuata dalla routine di risposta all'interruzione.
Si noti che in questo secondo caso, dopo aver generato la richiesta di interrupt il conteggio prosegue, modulo 2^M se il contatore ha M bit, fino a quando viene ricaricato il valore di ripristino, portando ad un accumulo degli eventuali (ma in molti casi probabili) ritardi nelle risposte alle richieste di interruzione, cioè con latenze maggiori di TIG. In altre parole il conteggio di un nuovo periodo non inizia esattamente al

termine del periodo precedente, ma solo dopo che è trascorso tutto il tempo necessario per l'attivazione ed esecuzione della routine di servizio dell'interruzione, che provvede al nuovo caricamento.

Un'efficace tecnica per evitare l'accumulo di questi ritardi consiste nel **sommare** il valore TBV al valore di conteggio corrente raggiunto nel frattempo, invece che *ricaricare* il valore TBV, defalcando in tal modo le unità di tempo eventualmente già trascorse durante la latenza della risposta all'interrupt.
N.B. - Si è fatta l'ipotesi di contatore a **decremento** che costituisce il caso tipico.
Ovviamente in caso contrario si dovrebbe operare una sottrazione del valore TBV dal valore di conteggio raggiunto dal timer.

Con questo approccio a doppio orologio la **gestione temporale dei processi** è effettuata dalla routine di risposta ai tick di TBC, **l'aggiornamento di data e ora** può essere effettuato dalla risposta ai tick di TBC o agli overflow di TIMER per i normali usi a "grana grossa", mentre **le misure di intervalli di tempo** più raffinate utilizzeranno i valori di conteggio leggibili dal contatore TIMER.

NOTE - Poichè il TIMER ha **periodo** TIP, questo è il più lungo intervallo di tempo misurabile senza ambiguità. Misure di tempo di intervalli più lunghi possono ottenersi utilizzando la variabile data-ora aggiornata a SW per risolvere le ambiguità del tempo ciclico ricavato da TIMER, ponendo però attenzione allo spiazzamento temporale tra l'evento "overflow di TIMER" e l'evento "aggiornamento di data-ora". Infatti durante l'intervallo di tempo tra l'evento HW di overflow e l'evento SW di aggiornamento, si ha inconsistenza tra la parte più significativa della data (SW) e quella meno significativa (contenuto del registro TIMER).

Tipica granularità di TIMER	TIG = 1..100 microsecondi.
Tipica granularità di TBC	TBG = 1..50 millisecondi.

6.7.4 IMPLEMENTAZIONI COMPLESSE

Nei (micro)processori progettati per applicazioni di tempo reale stretto sono talvolta presenti interessanti meccanismi HW che consentono di effettuare particolari operazioni relative al tempo con la granularità molto fine che solo a livello HW è efficacemente possibile, grazie a latenze e tempi di esecuzione trascurabili rispetto ai corrispondenti della versione SW.
Un interessante esempio è costituito dal microcontroller Intel 80C196KB che fornisce le prestazioni dette HSI (High Speed Input) e HSO (High Speed Output) di cui presentiamo brevemente le caratteristiche principali.

6.7.4.1 - HSI - cattura eventi con timbro orario

E' una funzione in grado di rilevare eventi associando *automaticamente* a livello HW un *time-stamping* con il valore prelevato da un contatore temporale (timer).

Possono essere tenuti sotto controllo fino a 4 flussi di eventi e la coda (FIFO) di eventi rilevati, corredati di *time-stamp* e non ancora "recepiti" a livello SW è lunga al massimo 8. E' possibile impostare la generazione di una richiesta di interruzione quando la coda è lunga 1,4,8 eventi.

Ogni elemento della coda è descritto con 20 bit, di cui 4 bit rappresentano i possibili (eventualmente concomitanti) eventi ed i rimanenti 16 bit rappresentano l'istante di tempo relativo (*time-stamp*).

6.7.4.2 - HSO - genera eventi a tempi prefissati

E' una funzione in grado di gestire fino ad 8 preimpostazioni di eventi. Le preimpostazioni sono inseribili in una memoria associativa (CAM *content-addressable*) le cui celle descrivono ciascuna un evento (scelto da un insieme di flussi di eventi possibili) e l'istante temporale in cui tale evento dovrà essere generato.

Ad ogni incremento del timer, mediante un accesso per contenuto alla CAM, viene verificata l'eventuale presenza di eventi da generare in quell'istante, ed in caso affermativo vengono generati gli eventi corrispondenti, dopo di che, in base ad un'opzione programmabile, le prenotazioni degli eventi che sono stati generati possono essere cancellate o mantenute. Le prenotazioni cancellate perchè soddisfatte lasciano posti liberi nella memoria associativa per ulteriori prenotazioni. Le prenotazioni non cancellate rimangono in vigore per il prossimo ciclo, consentendo di ottenere una generazione ciclica di sequenze di eventi.

6.7.4.3 Coprocessore di temporizzazione

Per raggiungere gli obbiettivi di elevate risoluzioni e precisioni con minimo sovraccarico (overhead) per la CPU, che nei sistemi convenzionali sono obbiettivi contrastanti, si può pensare di affidare ad un "coprocessore di temporizzazione" una gamma flessibile di funzioni sia relative alla misura dei tempi che all'attivazione temporizzata di processi.

Dispositivi ausiliari di gestione delle temporizzazioni, più o meno ricchi di funzioni, sono necessari se si vogliono utilizzare per applicazioni real-time anche processori "*general purpose*" come ad esempio 80386 e 80486 che sono potenti ma scarsamente dotati di funzioni specifiche per il tempo.

6.7.5 TEMPORIZZATORI SOFTWARE

In molte applicazioni si ha la necessità di gestire diverse, e talora numerose, temporizzazioni, anche tra loro concomitanti. Spesso non è conveniente adottare un temporizzatore HW per ogni distinta funzione di temporizzazione, ma risulta particolarmente efficace e flessibile una soluzione SW.
L'approccio consiste nel definire un certo numero di temporizzatori, dotati ognuno di una propria struttura dati che ne descrive lo stato, tutti gestiti a livello SW dalla routine di risposta alle interruzioni di un **unico temporizzatore** HW.

Questo approccio è così utile e significativo che quasi tutti i nuclei di sistemi operativi real-time lo adottano, sia pure con diverse modalità.
In genere i processi applicativi possono chiedere al SO, con apposite primitive, che venga loro assegnato un temporizzatore, che tale temporizzatore sia "lanciato" con un certo tempo DT, che alla scadenza dell'intervallo DT sia chiamata una data funzione, che un temporizzatore venga disattivato (*reset*), ecc.

Nel seguito viene riportata una possibile implementazione in linguaggio C di un insieme di temporizzatori che possono essere inseriti in applicazioni con esigenze di temporizzazione che non trovano adeguato supporto nei servizi offerti dal SO disponibile.
La gestione di questi temporizzatori richiede che sia predisposto un meccanismo con cui ad ogni interrupt del Real-Time Clock di sistema sia invocata l'apposita funzione riportata nell'esempio.
L'esempio è volutamente presentato sotto forma di parte di programma, per dare maggior concretezza, ma naturalmente lascia incomplete e imprecisate molte parti necessarie per una effettiva realizzazione funzionante.
Notiamo infine che l'esempio riportato costituisce solo uno dei possibili modi per realizzare delle temporizzazioni multiple con gestione SW.

```
/**************************
GESTIONE DI TIMER SOFTWARE
**************************

Sono previsti N_TIM timer software, gestiti da alcune routine di
sistema mediante primitive invocate dagli applicativi, e da una
routine agganciata alla risposta ad interrupt di Real Time Clock.
Nelle primitive i timer vengono citati per numero 0..N_TIM-1
All'atto dell'inizializzazione viene associata ad ogni timer una
funzione,
      che deve essere breve perche' eseguita sotto risposta a
      interrupt di Real Time Clock, e che viene invocata
      automaticamente allo scadere del tempo impostato.
Sono previsti i tre tipi classici di temporizzazione:
1) - DURATA - L'uscita diventa ON allo start e dopo il tempo DT
      l'uscita va OFF e si invoca la funzione.
2) - RITARDO - La temporizzazione parte con lo start. Dopo il
      tempo DT l'uscita va ON e si invoca la funzione.
      L'uscita va a OFF con la primitiva di reset.
3) - CICLICO - Con start parte la temporizzazione e dopo ogni
      intervallo DT si invoca la funzione e l'uscita va ON per
      il periodo di un tick di RTC.
      La temporizzazione ciclica termina con il comando di reset.
I tempi sono espressi in ms e la costante RTCP indica di quanti ms
e' il periodo tra i tick di RTC (multiplo intero di ms).
```

```
L'uso di long consente intervalli DT fino a 2E31 ms, circa 2 milioni
di secondi, cioe' oltre 23 giorni.
*/


/*********************************************************
Struttura dati per N_TIM timer gestiti a software
*********************************************************/

/*==== PARAMETRI CONFIGURABILI PER APPLICAZIONE ====*/
#define N_TIM xxx  // numero di timer previsti
#define RTCP 5    // periodo di RTC in ms

/* --- TIPI DI TIMER -----*/
#define DURATA 1
#define RITARDO 2
#define CICLICO 3
/* --- STATI DEI TIMER ----*/
#define RIPOSO 1
#define CONTA 2
#define SCADUTO 3

/* --- USCITE DEI TIMER -----*/
#define OFF 0
#define ON 1

#define esito int
#define OK 1
#define ERR 0xff        // codice esadecimale di errore


typedef uint unsigned int;
typedef PFUN (*void)();       // puntatore a funzione


typedef struct
{
      char  tipo;
      char  stato;
      char  uscita;
      long  tempo_tot;
      long  tempo_corr;
      PFUN  azione;
}TIMER;

TIMER t_sw [N_TIM - 1]; // Array dei timer
```

```
/*******************************************************
*******************************************************
PRIMITIVE DI USO DEI TIMER DA PARTE DEL SW APPLICATIVO
*******************************************************
*******************************************************/


/*****************************************************************
INIZIALIZZA TIMER - Definisce il tipo di timer e associa la funzione
parametri
      nt          numero del timer (0..N_TIM-1)
      tipo        enumerativo (DURATA, RITARDO, CICLICO)
      funzione    puntatore a funzione da chiamare a scadenza
                  temporale
ritorna esito = OK se inizializzazione effettuata
        esito = ERR se nt o tipo e' scorretto
-----------------------------------------------------------------*/
esito ini_timer (uint nt, char tipo, PFUN funzione)
{
      if (nt >= N_TIM)
            return (ERR);
      switch (tipo)
      {
            case DURATA:
            case RITARDO:
            case CICLICO:
                  t_sw [nt].tipo = tipo;
                  t_sw [nt].stato = RIPOSO;
                  t_sw [nt].uscita = OFF;
                  t_sw [nt].tempo_tot = 0;
                  t_sw [nt].tempo_corr = 0;
                  t_sw [nt].azione = funzione;
                  return (OK);
            default:
                  return (ERR);
      }
}


/*****************************************************************
LANCIA TIMER - Attiva il timer (gia' inizializzato) specificando il
tempo
parametri
      nt          numero del timer (0..N_TIM-1)
      tempo       valore in ms
ritorna esito = OK oppure esito = ERR se nt e' scorretto
-----------------------------------------------------------------*/

esito start_timer (uint nt, long tempo)
{
      if (nt >= N_TIM)
            return (ERR);
      if (t_sw [nt].stato != RIPOSO)
            return (ERR);
      t_sw [nt].stato = CONTA;
      t_sw [nt].tempo_tot = tempo;
      t_sw [nt].tempo_corr = 0;
      if (t_sw [nt].tipo == DURATA)
            t_sw [nt].uscita = ON;
      return (OK);
}


/*****************************************************************
RESET DEL TIMER - Lo porta nello stato di riposo con uscita OFF
parametri
      nt          numero del timer (0..N_TIM-1)
ritorna esito = OK oppure esito = ERR se nt e' scorretto
```

```
-------------------------------------------------------------------*/

esito reset_timer (uint nt)
{
      if (nt >= N_TIM)
            return (ERR);
      t_sw [nt].stato = RIPOSO;
      t_sw [nt].uscita = OFF;
      return (OK);
}


/*******************************************************************
RIPORTA LO STATO DEL TIMER
parametri
      nt          numero del timer (0..N_TIM-1)
ritorna stato = RIPOSO, CONTA, SCADUTO
            oppure esito = ERR se nt e' scorretto
-------------------------------------------------------------------*/

char get_timer_status (int nt)
{
      if (nt >= N_TIM)
            return (ERR);
      return (t_sw [nt].stato);
}


/*******************************************************************
RIPORTA USCITA DEL TIMER
parametri
      nt          numero del timer (0..N_TIM-1)
ritorna uscita = OFF, ON
            oppure esito = ERR se nt e' scorretto
-------------------------------------------------------------------*/

char get_timer_output (uint nt)
{
      if (nt >= N_TIM)
            return (ERR);
      return (t_sw [nt].uscita);
}
```

```
/*****************************************************************
ROUTINE DA CHIAMARE NELLA RISPOSTA A INTERRUPT DI RTC
Questa routine deve essere chiamata ogni RTCP ms, come
precisato dalla #define RTCP nei parametri configurabili.
-----------------------------------------------------------------*/
void update_timers (void)
{
int  i;
TIMER *ptim;

     for (i=0; i<N_TIM; i++)  // ciclo su tutti i timer
     {
          ptim = &t_sw [i];   // puntatore al timer in esame
          if (ptim->stato == CONTA)
          {              // solo quelli attivi
               if (ptim->tipo == CICLICO)
                    ptim->uscita = OFF;
               ptim->tempo_corr += RTCP;  // incrementa tempo
               if (ptim->tempo_corr >= ptim->tempo_tot)
               {              // qui scaduto il tempo
                    (*(ptim->azione))();  // chiama la funzione
                    switch (ptim->tipo)
                    {          // tratta i diversi tipi di timer
                    case DURATA:
                         ptim->stato = RIPOSO;
                         ptim->uscita = OFF;
                         break;
                    case RITARDO:
                         ptim->stato = SCADUTO;
                         ptim->uscita = ON;
                         break;
                    case CICLICO:
                         ptim->tempo_corr -= ptim->tempo_tot;
                         ptim->uscita = ON;
                         break;

                    default:
                         ptim->stato = RIPOSO;
                         ptim->uscita = OFF;

                    }
               }
          }
     }
}
```

6.8 ESERCIZI

-- Discutere il compromesso tra granularità e massima durata nella misura di intervalli di tempo.

-- Perchè la latenza di attivazione dei processi è maggiore della latenza delle risposte ad interrupt?.

-- Discutere i compromessi nella scelta del periodo dei tick di RTC.

-- Perchè i temporizzatori a durata non richiedono necessariamente un comando di reset, come invece avviene per quelli a ritardo e quelli ciclici?.

-- Individuare possibili applicazioni che richiedano temporizzatori a durata di tipo retriggerabile e di tipo non retriggerabile.

-- Analizzare i problemi che si pongono nella misura di lunghi intervalli di tempo.

-- Sono corrette le misure di intervalli di tempo se il contatore nel frattempo subisce un overflow?. E se ne subisce due?.

-- Quali possono essere secondo voi, le risoluzioni temporali ottenibili con i meccanismi HSI e HSO presentati in questo capitolo?.

-- Discutere pro e contro degli approcci HW e SW nel caso in cui si abbia necessità di svariati temporizzatori.

- Discutere le due scelte di eseguire le azioni nell'ambito della routine di risposta ad interrupt oppure, con sincronizzazione di secondo livello, nell'ambito di un processo.

6.9 ALTRE LETTURE

Connie U. Smith, Lloyd G. Williams
Software Performance Engineering. A Case Study Including Performance Comparison with Design Alternatives.
IEEE Trans. on Software Eng. Vol.19 N.7 jul.1993

Interessante trattazione di aspetti legati alle prestazioni e ricca bibliografia anche generale su Real-time.

L. Mezzalira, A. Morzenti.
Relating Specified Time Tolerances to Implementation Performances.
IEEE C.S.Press - 6th Euromicro Workshop on Real-Time Systems,
Vaesteraas, Sweden, June 15-17, 1994

Ruoli del tempo, primitive di temporizzazione e relative specifiche TRIO.