Misure di intervalli temporali

Misure di tempo

Abbiamo visto che è possibile settare intervalli regolari di tempo (utilizzando pause o timers) e misurarne la durata con la funzione `gettimeofday`; abbiamo utilizzato questo "orologio" per regolare e memorizzare il tempo in cui abbiamo eseguito le misure con il multimetro.

A volte è però necessario eseguire misure di intervalli temporali determinati da apparati esterni: sia nel caso di fenomeni periodici (ad esempio il periodo di oscillazione di un pendolo, la frequenza di un sistema oscillante) che nel caso di fenomeni singoli o stocastici (il tempo di caduta di un grave, l'intervallo tra il passaggio di due raggi cosmici).

Reazione ad un evento

Come prima cosa dobbiamo essere capaci di comunicare al computer che è successo qualcosa. Un modo di farlo è utilizzare il registro **STATUS** della porta parallela: il passaggio da 0 a 1 di una di tali linee sarà l'evento di cui vogliamo misurare il timing.

Sarà quindi necessario trasformare l'evento fisico che ci interessa nella (rapida) transizione di una tensione da 0 a 5 Volts (trasduzione).

Dal punto di vista del computer, si dovrà semplicemente leggere uno dei bits del registro `STATUS` della porta parallela.

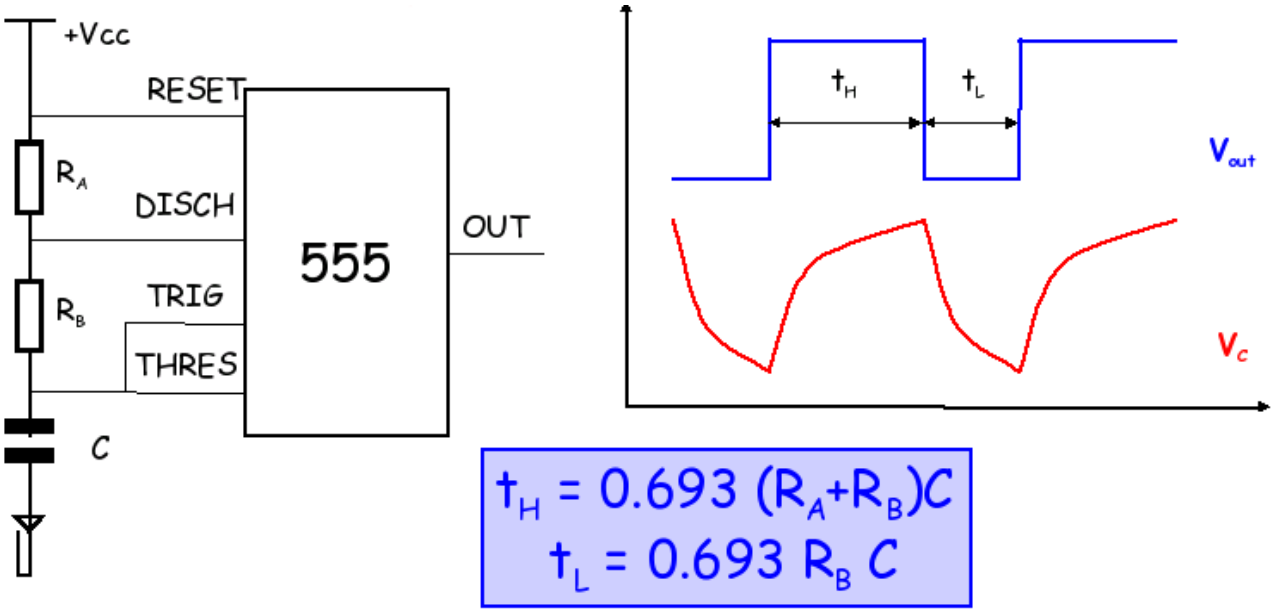
Pin	Bit	SPP Signal	I/O	Logica
10	S6	Ack	IN	
11	S7	Busy	IN	INV
12	S5	Paper Out/End	IN	
13	S4	Select	IN	
15	S3	Error/Fault	IN	

In laboratorio utilizzeremo `Ack` (il primo pin dopo il registro dati).

L'oscillatore

Per provare i vari metodi di misura utilizziamo (come strumento di test e debugging) un oscillatore basato sul circuito integrato 555.

Non sono importanti i dettagli del suo funzionamento: basta sapere che il 555 induce continuamente una carica del condensatore C attraverso R_A ed R_B seguita da una scarica attraverso R_B. L'uscita rimane al valore 0 V per un tempo t_L e quindi a $+5$ V per un tempo t_H.



Il polling

Il metodo del "polling" è molto intuitivo: si chiede in continuazione al dispositivo se è successo qualcosa (nel nostro caso una transizione 0-1) e si misura il tempo a cui è avvenuto.

Possibile schema per fare n misure di tempo

```
bool second=false;
int oldbit, bit=1;
do {
  oldbit = bit;
  // lettura del bit
  if (oldbit==0 && bit == 1){
    // salvo il t in t_old
    // calcolo t
    // a partire dal secondo tempo: calcolo delta_t = t-t_old
    //                              e aggiorno n
  }
} while(n<=nmis);
```

Il polling (II)

Per controllare la transizione 0-1 si deve leggere il registro `STATUS` e controllare se il pin `Ack` è 0 o 1.

Per fare quest'ultima operazione si può eseguire un and bitwise con `PARPORT_STATUS_ACK` (che vale per la cronaca `0x40`)

```
unsigned char status;
ioctl(fd_par,...,...);
if (status & PARPORT_STATUS_ACK)
  // il bit e' 1
else
  // il bit e' 0
```

Difetti del polling

Il polling costituisce una buona tecnica nel caso di una macchina single process - single user, ovvero una macchina che non ha null'altro da fare che interrogare il dispositivo.

Nella realtà però un computer esegue molte operazioni allo stesso tempo, ed alcune di queste (in particolare tutte quelle che hanno a che fare con la gestione dei dispositivi esterni e del sistema) hanno priorità maggiore rispetto ai processi dell'utente. Di conseguenza tali processi possono venire interrotti in qualunque momento, anche in funzione del carico della macchina, e ciò degrada la risoluzione della misura temporale.

Inoltre il continuo accesso al device consuma tutte le risorse del computer, e gli impedisce di fare qualcosa di utile durante l'attesa dell'evento.

Interrupt sulla porta parallela

Quando un dispositivo deve dialogare con la CPU, attira la sua a attenziona utilizzando un segnale digitale di interrupt.

Se vogliamo che la porta parallela registri un interrupt in corrispondenza della transizione 0-1 del bit `Ack` occorre caricare il modulo della parallela `parport_pc` con le seguenti opzioni

```
modprobe parport_pc io=0x378 irq=7
```

(quest'operazione è già stata eseguita sulle macchine dell'aula informatica)

Un interrupt può causare la chiamata di una funzione (interrupt handler).

Per controllare il numero di interrupt ricevuti dalle varie periferiche

```
cat /proc/interrupts
```

Scrivere un interrupt handler per la parallela (modificando ppdev o scrivendo un nuovo driver) va oltre gli scopi del corso. Qui ci limiteremo a contare gli "interrupt" registrati dalla porta parallela.

Interrupt (II)

La funzione `select` permette di restare in attesa finchè non diventa attivo un file descriptor in un insieme di file descriptor.

```
int select(int n, fd_set *rfds, fd_set *wfds, fd_set *exfds,
           struct timeval *timeout)
```

- `n` è il file descriptor maggiore + 1,
- `rfds, wfds, exfds` sono gli insiemi dei descrittori da controllare in lettura, in scrittura e per verifica di errori;
- il tempo dopo il quale la funzione ritorna se non si verifica nessun cambio di stato (se si passa un puntatore nullo la funzione aspetta indefinitamente).

Cos'è un insieme di file descriptor e come si gestisce ?

```
#include <select.h>
fs_set fd_coll;
FD_ZERO(fd_set *fd_coll);          // azzera l'insieme
FD_SET(int fd, fd_set *fd_coll); // assegna il file descriptor f
                                   // alla collezione fd_coll
```

Interrupt (III)

Per conoscere il numero di interrupt registrati dalla parallela e risettarne a zero il valore

```
ioctl(int fd,PPCLRIRQ,int *ncount);
```

Come combinare tutte queste funzioni per calcolare l'intervallo temporale tra due transizioni 0-1 ?

Normalmente un file descriptor diventa attivo in lettura quando il corrispondente file è pronto per essere letto.

Nel caso della parallela, configurata con interrupt su ack, il file descriptor diventa attivo lettura non appena viene registrato almeno una transizione 0-1 sul pin ack.

Interrupt (IV)

Schema della misura di intervallo con interrupt.

```
fs_set fd_coll; int ncount;
FD_ZERO(&fd_coll);          // azzera l'insieme
FD_SET(fd_par, &fd_coll); // assegna la parallela a fd_coll
...
ioctl(fd_par,PPCLRIRQ,&ncount);  // resetto gli interrupt
...
select(fd_par+1,&fd_coll,0,0,0); // aspetto una transizione 0-1
ioctl(fd_par,PPCLRIRQ,&ncount);  // resetto gli interrupt
// calcolo t0
select(fd_par+1,&fd_coll,0,0,0); // aspetto una transizione 0-1
// calcolo t1
ioctl(fd_par,PPCLRIRQ,&ncount);  // resetto gli interrupt
```

Vantaggi dell'interrupt (tramite select)

Siccome non c'è una funzione direttamente collegata all'interrupt (interrupt handler), l'interrupt, pur essendo in media più veloce del polling, è comunque soggetto a eventuali ritardi nella chiamata alla funzione che calcola il tempo (o a `ioctl`).

Tuttavia è possibile riconoscere le misure chiaramente falsate contando il numero di interrupt raccolti tra le due misure (deve essere uno solo).

Inoltre ha l'indubbio vantaggio di ottimizzare l'utilizzo della CPU.

Un caso particolare

Se il fenomeno è perfettamente periodico si può misurare il numero di interrupt registrati in un tempo noto:

```
...
ioctl(fd_par,PPCLRIRQ,&ncount);  // resetto gli interrupt
// calcolo t0
usleep(twait);
ioctl(fd_par,PPCLRIRQ,&ncount);  // resetto gli interrupt
// calcolo t1
...
dt = (t1-t0)/ncount;
```

Se il metodo è preciso (attesa lunga) non è veloce.

Esercitazione di laboratorio

Traccia

- Implementare una funzione per il polling ed una per l'interrupt
- Istogrammate i periodi ottenuti (valutando media ed errore)
- Confrontate la precisione dei due metodi variando il periodo da $\sim 10\ ms$ a $\sim 10\ \mu s$.

Consigli

- Fate funzioni "leggere" per non rallentare ulteriormente la lettura (no istogrammi, no output):

```
void polling(double *delta_t, double n);
void interrupt(double *delta_t, int *ninterr, double n);
```

- Gestite istogrammi e output nella funzione principale (e quindi in maniera identica per polling e interrupt)