Cap. 12

12. UN MODELLO IMPLEMENTATIVO

In questo capitolo viene presentata una struttura implementativa adatta per applicazioni di acquisizione dati e controllo di macchine o piccoli impianti, mediante microcalcolatori a 16 bit della classe del 80186 o del V-25, corredati di nucleo multitask.

Le caratteristiche generali della classe di applicazioni sono le seguenti.

- Si tratta di macchine o di piccoli impianti la cui evoluzione deve essere automatizzata e soggetta a supervisione ed eventuale conduzione da parte di un operatore.
- Interfaccia con operatore, tramite tastiera e video standard o con versioni ridotte di tastiere e display.
- Acquisizione di valori di misure e di stati digitali.
- Gestione allarmi.
- Anelli di regolazione con periodi di campionamento di decine di ms.
- Eventuali comunicazioni con altri calcolatori di supervisione.

La presentazione della struttura implementativa è di tipo esemplificativo e non si propone di descrivere in dettaglio e risolvere tutti i problemi.

12.1 STRUTTURA GENERALE

Adottiamo la seguente scomposizione funzionale:

GESTIONE INFORMAZIONI DIGITALI
GESTIONE MISURE
GESTIONE OPERATORE
GESTIONE COMUNICAZIONI

12.1.1 GESTIONE INFORMAZIONI DIGITALI

La gestione delle informazioni digitali prevede generalmente operazioni piuttosto semplici, e quindi eseguibili velocemente e con basso carico di lavoro per la CPU. E' in genere conveniente svolgere queste funzioni in modo ciclico con approccio *time-driven*.
Questa gestione comprende tipicamente le seguenti funzioni.

I/O di segnali ON/OFF
Allarmi
Temporizzatori SW
Conteggi lenti SW
Blocchi e consensi.

12.1.2 GESTIONE MISURE

La gestione delle misure prevede operazioni relativamente complesse, sia dal punto di vista computazionale, quindi con relativamente elevati tempi di esecuzione e carico di lavoro per la CPU, sia per quanto riguarda i dispositivi di interfaccia. In particolare la complessità delle funzioni di interfaccia in molti casi comporta tempi di attesa di eventi del tipo "periferica pronta", che spesso suggeriscono una gestione ad interrupt e sincronizzazione di secondo livello con il *task* principale. Per questi motivi è in genere opportuno eseguire queste funzioni solo quando è richiesto, con approccio *event-driven*.
Questa gestione comprende tipicamente le seguenti funzioni.

Conteggi veloci HW e misure di tempi.
I/O di segnali analogici con relativa conversione A/D e D/A.
Conversione in misure ingegneristiche con correzioni e linearizzazioni.
Calcolo di misure derivate.
Filtraggi vari SW.
Regolatori vari (tipicamente PID) SW.
Generatori di forme d'onda (tipicamente rampe) SW.

12.1.3 GESTIONE OPERATORE

La gestione dell'interazione con l'operatore è basata su un automa a stati che riceve simboli di ingresso prodotti dai comandi dell'operatore e da eventi del sistema, e che manifesta le sue uscite anche in forma visibile, su un normale video oppure su display di formato ridotto (ad es. 2 righe di 40 caratteri).
La gestione operatore riguarda in generale le seguenti funzioni.

Introduzione o modifica dei parametri di configurazione, coefficienti, ecc.
Comandi di avviamento e arresto.
Comandi di esecuzione di particolari funzioni: ad es. taratura.
Scelte delle diverse pagine di informazioni da visualizzare.
Comandi di tacitazione di allarmi.
Comandi e visualizzazioni diagnostiche per *debug*.

12.1.4 GESTIONE COMUNICAZIONI

La gestione delle comunicazioni assume forme varie in dipendenza del tipo di rapporto funzionale che si vuole instaurare tra il calcolatore in oggetto e gli altri calcolatori con cui avviene la comunicazione.
Un caso relativamente generale prevede che si abbia un collegamento punto-punto con un sistema supervisore che funge da master del collegamento, e un collegamento multipunto (bus di campo) con piccoli sistemi di acquisizione dati di livello inferiore.

12.2 MODELLO FUNZIONALE

Il modello funzionale che ci proponiamo di seguire è basato su

- un **automa** a stati che modella il comportamento reattivo "**discreto**" del sistema (o più automi se è conveniente scomporre il sistema in sottosistemi relativamente autonomi)
- una rete di **blocchi funzionali** che rappresentano dei **dispositivi software** che modellano il comportamento trasformazionale "**continuo**".

12.2.1 BLOCCHI FUNZIONALI

Questi "dispositivi" software sono pensabili secondo un paradigma dichiarativo come se fossero dotati di funzionalità intrinseca (cioè come se fossero realizzati ad HW), anche se sappiamo che il loro comportamento è ottenuto mediante la classica tecnica della **ripetizione ciclica** di azioni discrete con opportuna temporizzazione.
Questi dispositivi utilizzano i valori delle variabili che sono rappresentate dai loro ingressi e producono valori periodicamente aggiornati per le variabili associate alle loro uscite.
I dispositivi sono dotati di proprietà

- **statiche** - identificatore, tipo, morsetti di ingresso e uscita, funzione di elaborazione, ecc.
- **semistatiche** - parametri e coefficienti, modificabili con operazioni specifiche, ma non volatili. Per alcuni dispositivi è significativo prevedere diverse modalità di funzionamento per il normale esercizio oppore per simulazione a scopo di debug, impostabili mediante appositi indicatori.
- **dinamiche** - variabili di stato interno del dispositivo e valori di uscita.

12.2.2 AUTOMA A STATI

All'automa a stati è assegnato il compito di descrivere l'evoluzione generale (modi, fasi, passi) del sistema di automazione i cui stati sono rappresentativi degli stati del sistema complessivo. Quando l'evoluzione generale del sistema è prevalentemente guidata dall'operatore è significativo adottare un unico automa che governa il colloquio con l'operatore e determina anche la funzionalità generale. In altri casi si possono adottare diversi automi associati alle funzionalità che è conveniente assumere come attività che evolvono in parallelo tra loro.
Gli *Statecharts* presentati nel cap. 3 costituiscono un'interessante strumento descrittivo di questi automi.
Le **azioni** prodotte dagli automi sono associate sia alle transizioni che agli ingressi e uscite dai singoli stati, mentre **l'attività** associata agli stati ha lo scopo di supervisione e di procurarsi simboli per le transizioni di stato, ed è realizzata con funzioni cicliche dette in gergo *loop-function*.

12.2.3 COMUNICAZIONI

Le comunicazioni possono essere viste come svolgere due tipi di funzioni.

1. **Aggiornamento periodico e automatico di variabili di stato** che rappresentano copie di variabili (dette talora variabili di rete) prodotte dai sistemi remoti. Le informazioni interessate da questa funzionalità sono del tipo **eventi assoluti periodici**, con rilievo degli errori per **scartare** i valori errati. Questa funzione simula una **memoria globale** condivisa tra diversi calcolatori.
2. **Esecuzione su richiesta, di particolari transazioni**. Le informazioni in gioco sono **eventi incrementali sporadici**, con rilievo degli errori e richiesta di **ripetizione** (*datagram*).
   Possibili transazioni sono
   a bassa priorità:
   trasferimento di programmi (*download*) o *file*,
   upload di configurazioni (ad es. parametri, curve di taratura), ecc.
   ad alta priorità:
   invio di comandi
   Questa funzionalità virtualizza un **insieme di canali** trasmissivi punto-punto a funzionamento corretto, che cioè preservano integrità, completezza e ordinamento dei vari pacchetti in cui sono suddivise le informazioni e i file.

12.3 MODELLO IMPLEMENTATIVO

L'approccio implementativo che qui si propone è **globalmente** di tipo *event-driven* e basato su un insieme di processi che si appoggiano su un supporto di esecuzione costituito da un nucleo *multitask*.
Per semplificare la realizzazione e limitare il *gap* semantico tra funzionalità intrinseca che vogliamo virtualizzare e funzionalità programmata reale, diversi processi sono **localmente** impostati come periodiche ripetizioni di azioni cicliche, secondo lo schema *time-driven* che consente più regolari e meglio valutabili tempi di risposta. I periodi di queste ripetizioni saranno scelti in base alle granularità temporali caratteristiche delle diverse funzioni.

Per rendere più strutturata e modificabile la configurazione dell'applicazione si adotta una tabella dei dispositivi (detta spesso *data-base* dell'applicazione) in cui sono elencati tutti i blocchi funzionali con le relative caratteristiche e interconnessioni. Questa tabella fa riferimento, tipicamente con puntatori, a procedure e ad aree di memoria dei vari tipi: volatili, persistenti e ROM.
Le informazioni di stato prodotte da alcuni blocchi funzionali e utilizzate da altri sono descritte con variabili globali. Due particolari sottoinsiemi delle variabili globali sono costituiti dalle **immagini degli ingressi** e le **immagini delle uscite** che hanno lo scopo di **disaccoppiare** in qualche misura le attività interne dal mondo esterno.

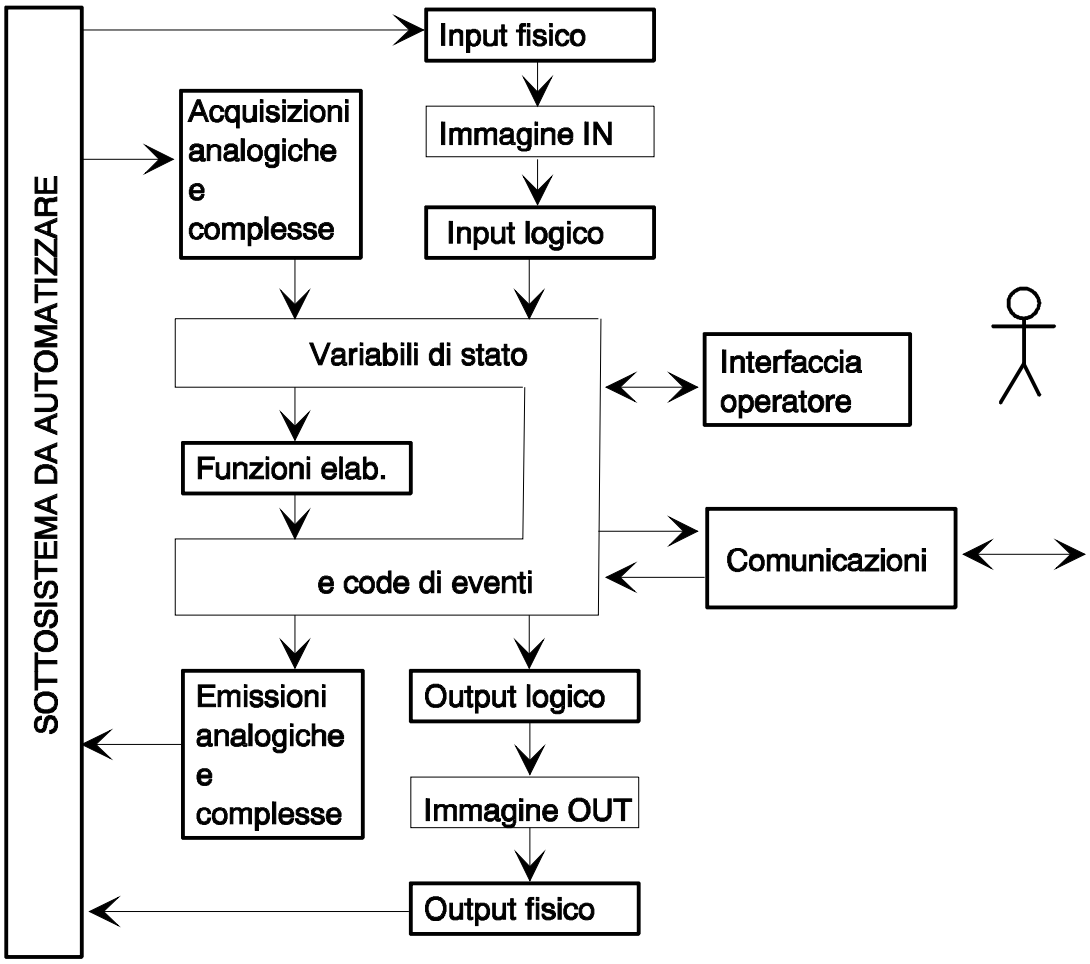

Fig.12.1 - Schema a flusso di dati del modello implementativo.
I rettangoli in grassetto rappresentano funzioni, mentre quelli sottili rappresentano strutture di dati.

12.3.1 MEMORIA

Innanzitutto notiamo che la memoria viene considerata composta di tre aree concettuali, non necessariamente connesse e contigue, che chiamiamo

- RAM per le variabili dinamiche
- TAM per i parametri semifissi
- ROM per il codice, le tabelle e i descrittori statici.

Queste aree potranno corrispondere a dispositivi fisicamente distinti e dotati delle proprietà di scrivibilità e persistenza opportune, come suggerito dal nome, oppure potranno essere porzioni di una normale memoria (RAM) gestite in modo da presentare il comportamento voluto. Ad esempio le tabelle su ROM possono risiedere su un'area RAM che viene inizializzata ad ogni accensione o *reset* del sistema copiando una vera ROM o un file da memoria di massa o altro equivalente (es. *download* da calcolatore supervisore).

12.3.1.1 Immagine degli ingressi.

Area di memoria RAM in cui sono previsti tanti byte quante sono le porte di ingresso utilizzate per segnali di tipo ON/OFF o di tipo impulsivo "lenti", cioè con stati di durata

minima superiore alla granularità temporale caratteristica. Si noti che invece non si fa l'immagine di eventi, impulsi veloci, ingressi analogici, ecc.
Durante il normale funzionamento l'esecuzione periodica di apposite procedure mantiene aggiornati i valori di questi byte in modo che rispecchino i valori dei segnali presenti ai morsetti di ingresso.
Quando si vogliano eseguire prove per *debug* di funzioni software o per diagnostica dei collegamenti dei segnali sulle interfacce e delle funzioni hardware, si possono disattivare gli aggiornamenti automatici di queste immagini, globalmente o selettivamente, in modo che l'operatore (tecnico di installazione o manutenzione) possa forzare ai valori voluti i bit che ritiene significativi.
I comandi per abilitare o disabilitare l'aggiornamento automatico delle immagini degli ingressi e per forzarne i valori, costituiscono un sottoinsieme del colloquio operatore, generalmente attivabile con opportune protezioni (*password*) per evitare abusi involontari o malevoli.

12.3.1.2 Variabili globali.

Area di memoria RAM vista come insieme di variabili dei consueti tipi `boolean, integer, float`, molte delle quali sono aggregate di tipo *record* (`struct` in C).
Queste variabili sono la rappresentazione delle informazioni che costituiscono i "segnali" generati ed utilizzati dai blocchi funzionali. Queste variabili rappresentano anche lo stato interno dei blocchi funzionali che le scrivono. La strutturazione con diversi campi rispecchia il significato di queste variabili: ad esempio le misure saranno corredate di *time-stamp* per poterne calcolare derivate e integrali, gli allarmi avranno indicatori dello stato, ecc., come vedremo più in dettaglio descrivendo i vari blocchi funzionali.

12.3.1.3 Immagini delle uscite.

Analogamente agli ingressi, è conveniente mantenere un'area di memoria che rispecchia i valori dei bit delle porte di uscita per segnali ON/OFF o impulsivi lenti.
Nel normale funzionamento i valori di questi byte di memoria sono periodicamente aggiornati in modo da rispecchiare i valori delle variabili globali che sono destinate alle uscite. Questo aggiornamento automatico può essere globalmente o selettivamente sospeso per consentire un controllo diretto dei valori da parte dell'operatore in sede di prova. Infine un processo periodico emette sulle porte di uscita i valori delle immagini.
Non è prevista l'immagine delle uscite impulsive veloci e degli eventi, a causa dei diversi requisiti temporali, non compatibili con una semplice elaborazione ciclica.

12.3.1.4 Parametri in tampone.

Si tratta di un'area di memoria TAM a lettura e scrittura ma non volatile, destinata a contenere delle variabili i cui valori costituiscono parametri di configurazione o di taratura, o comunque che devono sopravvivere agli spegnimenti e riaccensioni del sistema. Una possibile soluzione consiste in una memoria RAM CMOS dotata di una batteria in tampone: il bassissimo consumo statico dei CMOS permette l'uso di batterie incorporate nello stesso *chip* di memoria, con durata decennale.
L'importanza di mantenere validi i valori dei parametri suggerisce l'uso di opportune tecniche, ad esempio secondo lo schema seguente.

12.3.1.5 GESTIONE DI RAM TAMPONE
Doppia memorizzazione con checksum.

Si realizza una duplicazione identica dell'area delle variabili che costituiscono i parametri in tampone, associando ad ogni copia un byte di checksum (o due byte di CRC = *Cyclic Redundancy Check*).
A livello hardware il pin di WRITE della memoria è condizionato con un segnale di consenso generato da un bit di una porta di uscita. Ciò consente di confinare a ben precise e volontarie finestre temporali la possibilità di scrittura, in modo che eventuali errori software non modifichino i parametri.
Ogni aggiornamento dei valori dei parametri è effettuato da apposite procedure che:

1. abilitano la scrittura con l'apposito bit in uscita,
2. aggiornano i valori della prima copia,
3. aggiornano il checksum della prima copia,
4. aggiornano i valori della seconda copia,
5. aggiornano il checksum della seconda copia,
6. disabilitano la scrittura.

Può essere interessante associare ai singoli parametri diversi livelli di protezione. In questo caso viene accettata ed introdotta la modifica di un parametro solo se la *password* introdotta dall'operatore corrisponde ad un livello di accesso maggiore o uguale a quello richiesto dal parametro.
Protezioni più raffinate, necessarie per motivi di sicurezza in particolari impianti, possono prevedere che i diritti di accesso dipendano non solo dal **ruolo dell'operatore**, ma anche dallo **stato del processo**: automatico, manuale, manutenzione, fermo, ecc., che determina quali modifiche dei parametri sono da considerarsi **intrusive** e quali no

Ad ogni accensione del sistema, in fase di inizializzazione si effettua la procedura di verifica con l'algoritmo seguente.

```
Se le due copie hanno checksum corretto e sono eguali
     vengono accettate e si procede nell'avviamento.
altrimenti
     se la prima copia ha checksum corretto viene
     copiata sulla seconda e si procede
altrimenti
     se la seconda copia ha checksum corretto viene
     copiata sulla prima e si procede
altrimenti
     si segnala l'anomalia e il sistema accetta solo
     il comando di impostazione parametri o adotta una
     copia di default conservata su ROM.
```

Questa tecnica conferisce una buona robustezza al sistema ed evita il suo avviamento con valori non significativi dei parametri.

12.3.2 BLOCCHI FUNZIONALI

I blocchi funzionali hanno lo scopo di rappresentare **dispositivi virtuali** la cui funzionalità è ottenuta da una esecuzione periodica di opportune procedure, dipendenti dal tipo di dispositivo, utilizzando i valori delle variabili, considerate come segnali di ingresso al dispositivo, tenendo conto dei valori dei parametri associati al dispositivo e producendo variabili che rappresentano lo stato e le uscite del dispositivo.
Il descrittore di base di un'applicazione (detto talora un po' impropriamente base dati) è costituito da una tabella di dispositivi, realizzata come array dei descrittori statici elementari e tipicamente contenuta in area di memoria ROM.

Per l'esecuzione dei blocchi funzionali si adottano due diverse modalità in dipendenza della complessità delle funzioni che tali blocchi devono svolgere.
Per i blocchi con operazioni semplici e di rapida esecuzione (aritmetica intera, operazioni logiche) si tende ad usare l'approccio **ciclico** che ad ogni ciclo ripete la valutrazione di funzioni combinatorie su informazioni di **stato**, con la modalità **scrittore-lettori**.
I blocchi con funzionalità complessa, calcoli in *floating-point* ecc., per motivi di efficiente utilizzo della CPU, sono gestiti con la modalità **produttore-consumatore**, cioè attivati sulla base di **eventi**, tramite le primitive di PUT/GET di singoli codici su buffer circolari o SEND/RECEIVE di messaggi.

12.3.2.1 I descrittori statici elementari

Costituiscono la struttura descrittiva comune a tutti i blocchi funzionali, che risiede in area ROM e dalla quale si può risalire a tutti i dettagli di ogni singolo dispositivo. I descrittori statici elementari comprendono ad esempio i seguenti campi.

- **Codice tipo dispositivo**. Determina univocamente il formato e l'interpretazione di tutte le strutture di dati annesse al dispositivo stesso.
- **Nome simbolico**. Stringa di caratteri che rappresentano il nome in chiaro del singolo dispositivo. Questo nome consente il riferimento simbolico anche *run-time.*
- **Puntatore alla procedura esecutiva di base**. La procedura esecutiva può essere quella prevista come standard dal tipo di dispositivo, oppure potrà essere una procedura specifica che il programmatore applicativo ha scritto appositamente.
- **Puntatore al descrittore statico esteso**. Punta alla porzione di descrittore statico che è peculiare del particolare tipo di dispositivo e risiede in area ROM.

12.3.2.2 I descrittori statici estesi

I descrittori statici estesi hanno formato dipendente dal tipo di dispositivo e conterranno tipicamente le seguenti informazioni.

- Puntatore al descrittore dinamico, costituito da una variabile aggregata di struttura e contenuto caratteristici del tipo di dispositivo, e che costituisce l'uscita del dispositivo stesso.
- Puntatore al descrittore parametrico, costituito da una variabile aggregata che presenta una struttura specifica del tipo di dispositivo.
- Puntatori alle variabili utilizzate come ingressi.

12.3.2.3 TIPI DI BLOCCHI FUNZIONALI (Dispositivi)

Elenchiamo i più comuni tipi di blocchi funzionali, che verranno analizzati nelle sezioni successive, descrivendone la funzionalità e un tipico contenuto dei descrittori statico, parametrico e dinamico.

INPUT (logico)
OUTPUT (logico)
TIMER
CONTATORI
ALLARMI
MISURE
FILTRI
REGOLATORI
SIMULATORI

12.3.2.3.1 INPUT (logico)

FUNZIONE
In base al corrispondente bit dell'immagine di ingresso produce una variabile `boolean` (byte) tenendo conto della polarità (logica positiva o negativa).
La funzionalità può essere estesa a semplici operazioni di eliminazione di spurii, antirimbalzo, ecc. (v. cap.7)
DESCRITTORE STATICO
- Nome del segnale
- Indirizzo e posizione del bit corrispondente
- Ptr. funzione di estrazione del bit

DESCRITTORE PARAMETRICO
- Indicatore di logica positiva o negativa
- Eventuali parametri di filtraggio temporale

DESCRITTORE DINAMICO
- Valore boolean
- Eventuale stato di transitorio e regime

12.3.2.3.2 OUTPUT (logico)

FUNZIONE
Inserisce nell'immagine di uscita e nella posizione opportuna il valore di un bit in base al valore di una variabile booleana, tenendo conto della logica positiva o negativa.
DESCRITTORE STATICO
- Nome del segnale.
- Indirizzo della variabile booleana.
- Indirizzo e posizione del bit dell'immagine di uscita.

DESCRITTORE PARAMETRICO
- Indicatore di polarità positiva o negativa

DESCRITTORE DINAMICO
- Nessuno

12.3.2.3.3 TIMER

FUNZIONE
Produce un valore ON/OFF temporizzato, con la modalità dipendente dal tipo di temporizzatore: (v. cap. 6.6).
1. a tempo
2. a ritardo
3. ciclico

In alcuni casi può essere utile poter mettere in pausa il temporizzatore, "congelandone" il conteggio temporale raggiunto, con un comando di pausa. Il comando *Resume* servirà per riprendere il conteggio dal punto in cui è arrivato. Il comando di *Reset*, invece, ferma il temporizzatore e lo azzera.
L'uscita dei timer può essere utilizzata per attivare e disattivare a tempo determinate funzioni, indurre transizioni di stato negli automi, generare segnalazioni di time-out, ecc.
La possibilità di mettere in Pausa un temporizzatore è utile se il tempo che interessa controllare si riferisce solo allo stato di funzionamento di una macchina. Ad es. per le operazioni di manutenzione preventiva da eseguirsi dopo un certo numero di ore di funzionamento, si potrà utilizzare un temporizzatore di tipo 2 (a ritardo) che viene mantenuto in Pausa durante i periodi di fermo della macchina.
DESCRITTORE STATICO

Nome del temporizzatore
Tipo di temporizzatore
DESCRITTORE PARAMETRICO
Durata intervallo di temporizzazione
DESCRITTORE DINAMICO
Uscita ON/OFF
Contatore tempo corrente
Stato = [Riposo, Conteggio, Finito, Pausa]
COMANDI
Start, Pausa, Resume, Reset

12.3.2.3.4 CONTATORI

FUNZIONE
Mantenere aggiornate variabili (di tipo `int` o `long`) di conteggio di eventi. In genere è previsto che si possa sospendere il conteggio o azzerarlo.
Si possono avere due tipi di contatori, implementativamente molto diversi, per conteggi **lenti** e per conteggi **veloci**.

1. Per conteggi **lenti**, acquisiti come impulsi in ingresso a porte di IN e con durata maggiore della granularità temporale caratteristica, cioè del ciclo del processo di gestione digitale, il contatore rileva i fronti che rappresentano gli eventi da contare come variazioni di stato dei bit acquisiti in ingresso.
2. Per conteggi **veloci** avremo dei contatori primari HW che richiedono una loro specifica inizializzazione e gestione.

Generalmente i contatori prevedono un doppio conteggio parziale e totale, separatamente azzerabili.
In genere il conteggio totale è mantenuto in area TAM non volatile. Talora è utile che il contatore mantenga aggiornato un bit di stato che indica se è stato raggiunto un conteggio prefissato.
I contatori possono essere utilizzati per il conteggio di materiali, energia, spostamenti, cicli di lavorazione, ecc. Si noti infatti che il segnale di conteggio può provenire da macchine esterne di vario tipo, ma anche da processi SW interni al calcolatore di controllo.
DESCRITTORE STATICO
Nome del contatore
Ptr. al segnale di conteggio (per contatori lenti)
Ptr. alla funzione di conteggio
DESCRITTORE PARAMETRICO
Soglia di conteggio per segnalazione di raggiunto valore prefissato
Contatore totale
DESCRITTORE DINAMICO
Uscita ON/OFF di segnalazione di raggiunto conteggio
Contatore parziale
Stato = [Azzerato, Conteggio, Pausa]
COMANDI
Start, Pausa, Resume, Reset-parziale, Reset-totale

12.3.2.3.5 ALLARMI

FUNZIONE

Gestione di allarme associato ad un'anomalia. Contribuisce ad aggiornare anche indicatori globali di allarmi del gruppo di allarmi a cui appartiene (v. seguito).
L'allarme potrà prevedere un ritardo all'attivazione se l'anomalia può presentare degli spurii da ignorare.
Per ogni allarme sono previsti i modi:

- escluso - non viene gestito;
- controllo - gestisce solo le segnalazioni di allarme
- blocco - effettua anche il blocco di qualche macchina.

In alcuni casi si prevede anche la possibilità di attivare per prova gli allarmi, in modo da verificare che funzionino correttamente tutte le segnalazioni audio-visive correlate.

DESCRITTORE STATICO
- Nome allarme
- N. gruppo di allarmi di appartenenza
- Ptr. segnale ON/OFF di anomalia
- Ptr. funzione di blocco eventuale
- Messaggio identificatore

DESCRITTORE PARAMETRICO
- Ritardo all'attivazione
- Modo = [Escluso, Controllo, Bloccante]

DESCRITTORE DINAMICO
- Stato = [Riposo, Conteggio, Allarme, Tacitato, Rientrato]
- Conteggio corrente del tempo per ritardo all'attivazione

COMANDI
- Tacitazione, Prova

GRUPPO DI ALLARMI

Per ogni gruppo di allarmi è opportuno avere degli indicatori complessivi, che segnalino se almeno uno degli allarmi è in allarme, se almeno uno richiede il blocco, se almeno uno richiede l'attivazione della sirena, ecc. Questi indicatori costituiscono la funzione logica OR dei corrispondenti indicatori dei singoli allarmi.

- Ingressi - Tacitazione, Prova, Enable
- Uscite - OR-Presenti, OR-Bloccanti, OR-Acustici, OR-Rientranti

12.3.2.3.6 MISURE

FUNZIONE

Acquisire i valori grezzi tramite la gestione di opportune catene di misura, e convertirli in unità ingegneristiche (v. cap.8).

La conversione in unità ingegneristiche (U.I.) tramite caratteristica statica ricavata mediante operazioni di taratura software effettua contestualmente linearizzazione e correzioni di offset e di guadagno.

Con catene di misura a guadagno variabile può essere effettuato un *auto-range*.

Si ricorda che alcune misure sono derivate da altre, mediante il calcolo di opportune espressioni, mentre altre misure (ad es. di velocità) sono ricavate dal rilievo di conteggi, tempi e frequenze (v. cap.7). In questi casi la funzione di acquisizione non avrà in genere una diretta interazione con le interfacce di acquisizione fisica, ma utilizzerà valori acquisiti da altre misure, da routine di servizio di interrupt o da altre funzioni specifiche.

Per ogni misura sono previsti i modi:

- reale - normale acquisizione, o calcolo, dei valori di misura;
- simulato - si utilizza il valore calcolato da un apposito blocco di simulazione (v. seguito);
- escluso - non si aggiorna il valore della misura.

Generalmente per ogni misura sono previste delle soglie di allarme ai limiti inferiore e superiore del campo di valori della misura accettabili senza problemi. Il blocco misura fornirà quindi le segnalazioni di anomalia allarme basso e allarme alto, utilizzabili come ingressi per i corrispondenti blocchi di allarme. Nei casi più raffinati si potranno addirittura prevedere sia delle soglie di preallarme, sia delle soglie di allarme. Mentre le prime daranno luogo ad allarmi di controllo, le seconde corrisponderanno in genere ad allarmi bloccanti.

DESCRITTORE STATICO

Nome misura e relative unità
Descrittore del canale, N. , indirizzo
Ptr. funzione di acquisizione
Ptr. funzione di conversione in U.I.
Ptr. funzione di simulazione

DESCRITTORE PARAMETRICO

Periodo di acquisizione
Descrittore/i delle caratteristiche statiche (corrispondenza Misura - valore Letto)
Soglie per cambio range
Coeff. di correzione
Soglia allarme alto
Soglia allarme basso
Modo = [Reale, Simulato, Disattivato]

DESCRITTORE DINAMICO

Valore grezzo binario intero
Valore ingegneristico float
Tempo istante di acquisizione (*time-stamp*)
Delta-t da acquisizione precedente, per derivate e integrali
Progressivo misure acquisite
Boolean anomalia errori di acquisizione
Boolean anomalia sopra soglia sup.
Boolean anomalia sotto soglia inf.

COMANDI

Nessuno

12.3.2.3.7 FILTRI

FUNZIONE
I filtri (software) sono generalmente utilizzati per produrre valori di misure che presentino andamento “morbido”, cioè senza brusche e continue variazioni, come è desiderabile, ad es., per i valori numerici da visualizzare o da memorizzare.
Alcuni filtri hanno lo scopo di calcolare un valore più probabile, ad esempio con lo scarto di valori probabilmente affetti da errori, mediante algoritmi adatti. Questo aspetto è particolarmente importante quando si acquisiscono valori per la taratura SW.

DESCRITTORE STATICO
- Nome del filtro
- Ptr. alla funzione di filtraggio
- Ptr. alla misura da filtrare

DESCRITTORE PARAMETRICO
- N. campioni su cui effettuare il filtraggio
- Coefficiente di filtraggio

DESCRITTORE DINAMICO
- Array di valori delle misure precedenti.
- Valore filtrato
- Tempo istante di acquisizione campione precedente
- Delta-t da acquisizione precedente
- Progressivo misure acquisite

COMANDI
- Nessuno

12.3.2.3.8 REGOLATORI

FUNZIONE
La funzione dei regolatori è principalmente quella di produrre opportuni comandi in uscita, in forma numerica o di tipo ON/OFF, da inviare ad attuatori in grado di agire sulla grandezza manipolata. Generalmente i regolatori utilizzano due misure, eventualmente a loro volta derivate da altre, che rappresentano il valore voluto (riferimento o *set-point*) e il valore attuale della grandezza regolata.
Funzioni ausiliarie riguardano la generazione di indicatori (booleani) di anomalie, come *set-point* non accettabile, eccessivo scarto di regolazione (errore integrale), ecc., che potranno essere destinati come ingressi per blocchi di tipo allarme.
I modi di funzionamento previsti sono:

- Anello aperto - usato per il comando “manuale” in cui alla grandezza manipolata si impone un valore dipendente solo dal set-point;
- Anello chiuso - normale funzionamento del regolatore.
- Fermo - alla grandezza manipolata viene imposto il valore di riposo, indipendentemente dal set-point.

DESCRITTORE STATICO
- Nome del regolatore
- Ptr. misura del riferimento
- Ptr. misura della grandezza
- Ptr. funzione di regolazione

Ptr. funzione di emissione all'attuatore
DESCRITTORE PARAMETRICO
Kp, Ki, Kd (parametri del PID)
Banda morta
Coeff. guadagno attuatore
Intervallo valori set-point accettabili
Massimo gradiente in uscita
Soglia dell'errore integrale per eccessivo scarto regolazione
DESCRITTORE DINAMICO
Valore misura precedente
Valore misura attuale
Delta-t misure
Valore integrale corrente
Valore numerico di uscita
Modo = [Fermo, Anello aperto, Anello chiuso]
COMANDI
Manuale (*feed forward*), Automatico (*feed back*), Arresto.

12.3.2.3.9 SIMULATORI

FUNZIONE
Possono essere utili, nelle fasi preliminari di prove al banco dei blocchi che producono dei valori di misure che non è possibile o conveniente rilevare dal mondo esterno (ad es. perchè il processo fisico non è disponibile o è critico).
Lo scopo dei simulatori qui considerati non è l'accurata riproduzione dei comportamenti simulati, ma molto più semplicemente la generazione interna di valori che consentano di verificare le funzionalità di massima.
In particolare le grandezze da regolare potranno essere simulate con un semplice sistema del primo ordine con costante di tempo eventualmente anche ridotta rispetto ai valori reali per rendere più rapide le prove.
DESCRITTORE STATICO
Nome segnale simulato
Ptr. regolatore della grandezza manipolata
Ptr. misura simulata
Ptr. funzione di simulazione
DESCRITTORE PARAMETRICO
Coefficiente di guadagno
Costante di tempo
DESCRITTORE DINAMICO
E' ragionevole che i simulatori agiscano direttamente sugli elementi del descrittore dinamico della misura simulata.
COMANDI
Nessuno

12.3.3 PROCESSI

La scomposizione in processi (task) di un'applicazione e l'attribuzione delle varie funzioni ai vari processi presenta gradi di libertà che vanno risolti con compromessi tra diversi obbiettivi. Talora le esigenze di chiarezza e buona modularità possono entrare in conflitto con le esigenze di efficienza e prestazioni in tempo reale.
In linea di massima si suggerisce di seguire le considerazioni presentate brevemente nel seguito.

Evitare un eccessivo frazionamento in numerosi processi.
Infatti ogni processo induce un *overhead* di memoria per il suo descrittore, per i meccanismi di comunicazione (mailbox, semafori, ecc.), e soprattutto per lo *stack*.
Si ha inoltre *overhead* di tempo CPU per le funzioni di *scheduling* e per le commutazioni di contesto.
Per ogni CPU un ragionevole limite superiore del numero di processi è di circa una dozzina, nel senso che raramente è necessario avere più flussi concorrenti.
Adottare un processo per ogni entità esterna che evolve secondo una storia indipendente.

- Un esempio tipico è costituito dal processo che gestisce l'interazione con l'operatore.
- Altre eventuali situazioni classiche sono costituite dagli eventuali processi *server* uno per ogni unità periferica, come la stampante, la memoria di massa, ecc. Questa scelta si mappa molto bene con il comportamento sequenziale di queste unità e con le esigenze di utilizzo in mutua esclusione delle loro funzioni.
- Le comunicazioni seriali tramite rete o collegamento punto-punto meritano una ulteriore considerazione. Un processo di gestione delle comunicazioni è generalmente utile per disaccoppiare nella temporizzazione le funzioni dei livelli superiori del colloquio (ISO-OSI) da quelle dei livelli inferiori, nascondendo i dettagli di queste ultime. A questo processo sono affidate le funzioni eseguite **ciclicamente** di esportazione delle variabili prodotte localmente e di aggiornamento delle immagini locali delle variabili prodotte remotamente e di cui si riceve periodicamente un nuovo valore, simulando così un insieme di variabili di stato globali e condivise.

Molto spesso è utile affiancare a questo processo un secondo processo a cui affidare le comunicazioni **sporadiche** che generalmente sono basate su un meccanismo a pacchetti. Le informazioni inviate con questa tecnica sono assimilabili a file e sono ad esempio programmi, configurazioni di parametri, comandi. Si ricordi che per queste informazioni devono essere garantiti integrità, completezza e ordinamento, così da suggerire per esse un flusso indipendente, ad alto livello, da quello delle variabili di stato aggiornate periodicamente, anche se nei livelli inferiori i due flussi vengono a fondersi.

- Per la gestione di interfacciamenti con numerosi segnali di I/O si suggerisce un approccio più attento al tempo che all'eleganza programmativa.

E' noto che la coesione temporale, che aggrega cioè in uno stesso modulo delle operazioni per il solo fatto che esse vanno eseguite in successione temporale, non è considerata una coesione forte, ma nei sistemi che stiamo considerando (acquisizione dati e controllo di processo in tempo reale) sono particolarmente importanti le seguenti considerazioni:

> Evitare spreco di tempo in inutili cambiamenti di contesto.
> Utilizzo dei tempi di attesa per sincroniizzazione con gli eventi esterni tipicamente associati con le operazioni di interfaccia.

Adotteremo quindi un processo per le operazioni di I/O digitale di variabili di stato (ON/OFF) o impulsive lente, che non richiedono tempi di attesa, essendo disponibili su porte leggibili rapidamente ed in ogni istante.
Adotteremo un diverso processo per ogni multiplexer per le acquisizioni di segnali analogici che presentano diverse frequenze di campionamento, **attese** per la commutazione dei multiplexer e per le conversioni A/D. Le acquisizioni di diverse catene di misura possono essere infatti svolte in modo concorrente tra loro.

Altri processi (spesso uno solo) eseguiranno invece le operazioni di calcolo (CPU-*bound*) fungendo da consumatori delle informazioni prodotte dai processi di acquisizione.

Presentiamo ora una possibile attribuzione di funzioni a processi, che concretizza le indicazioni sopra riportate.

12.3.3.1 GESDIGI - Gestore di funzioni digitali

Processo periodico attivato da timer ciclico (con primitiva del tipo `WAIT_TIMER(periodo)`) con periodo tipicamente compreso tra 10 ms e 100 ms. Si noti che questo periodo va deciso in sede di progetto tenendo conto delle considerazioni sul rilievo di informazioni associate a stati, impulsi lenti, ecc., e determina la granularità temporale caratteristica della gestione “digitale”.
Questo processo esegue una funzionalità di tipo *time-driven* con ingressi e uscite sincroni, simile a quella normalmente adottata dai PLC. Si ricorda che rispetto alle funzioni svolte dal processo il ritardo massimo è pari a due volte il periodo di attivazione.
FUNZIONI DEL PROCESSO

- Input fisico
  Invoca la **procedura applicativa**, appositamente scritta da progettista, che effettua la lettura di tutte le porte di ingresso di segnali ON/OFF e aggiornamento delle corrispondenti immagini degli ingressi.
- Input logico
  Per ogni blocco INPUT LOGICO del *data base* in *modo reale* estrae dalle immagini degli ingressi la corrispondente variabile di tipo boolean (generalmente un byte) con eventuale inversione (se logica negativa) e attuazione di eventuali politiche di filtraggio per eliminazione di spurii.
  I bit di ingresso dichiarati in *modo simulato* invece non vengono toccati, e la gestione delle corrispondenti variabili boolean è lasciata alle apposite funzioni di comando da parte dell’operatore per *debug*.
- Timer
  Per ogni blocco TIMER attivo chiama la funzione associata che incrementa il conteggio corrente e, se scaduto, aggiorna il valore di uscita.
- Contatori
  Per ogni blocco CONTATORE attivo chiama la funzione associata che aggiorna il conteggio e lo stato.
- Allarmi
  Per ogni blocco ALLARME esegue l’automa associato, che eventualmente effettua transizioni di stato. Contestualmente vengono aggiornati gli indicatori del GRUPPO di allarmi a cui l’allarme appartiene.
- Output logico
  Per ogni variabile di stato booleano in uscita aggiorna il bit corrispondente nell’immagine di OUT, tenendo conto della eventuale logica negativa.
- Output fisico
  Invoca la **procedura applicativa**, appositamente scritta dal progettista, che scrive tutti i byte di immagine OUT sulle corrispondenti porte di OUT.

12.3.3.2 ACQMIS - Acquisizione periodica di misure

Processo periodico, attivato ciclicamente (con primitiva del tipo `WAIT_TIMER(periodo)`) con una temporizzazione adatta a tener conto delle frequenze di campionamento richieste e dei tempi di commutazione del multiplexer e di conversione del convertitore A/D.
Se si hanno diversi convertitori (probabilmente con altrettanti mux) si avrà un processo ACQMIS_X per ogni catena di acquisizione.

- Misure e Input analogici
  Per ogni blocco MISURA in modo reale aggiorna il conteggio del tempo dalla precedente acquisizione e se è scaduto il periodo di campionamento invoca la funzione di acquisizione che aggiorna il valore grezzo letto e il *time-stamp*, e poi la funzione di conversione che calcola e aggiorna il valore nelle unità ingegneristiche, incrementa modulo 256 il codice progressivo e aggiorna gli eventuali indicatori di supero delle soglie. Se la misura è destinata ad un regolatore o ad un filtro, invia un codice di sincronizzazione (ad es. il puntatore alla misura) al processo REGOL. Analoga attivazione del processo REGOL è effettuata se la misura è nel modo simulato.
  La sincronizzazione può utilizzare primitive tipo
  ```
  PUT (coda_regol, ptr_misura)
  ```

12.3.3.3 REGOL - Elaborazione di misure e regolazioni

Processo aperiodico, attivato dai codici di sincronizzazione delle misure rispetto a cui funge da consumatore.
Il ciclo base del processo può essere sincronizzato con la primitiva
```
ptr_misura = GET (coda_regol, FOREVER ).
```
Questo processo esegue le funzioni di elaborazione intensive rispetto ai calcoli.

- Filtri
  Invoca la funzione di filtraggio del filtro eventualmente associato alla misura.
- Regolatori
  Invoca la funzione di regolazione associata alla misura. Invoca la funzione di emissione del comando all'attuatore corrispondente.
- Simulatori
  Invoca l'eventuale funzione di simulazione che calcola e aggiorna i valori dinamici della misura simulata, tenendo conto del valore di attuazione precedentemente fornito dal regolatore eventualmente associato o dei criteri di generazione adottati.

12.3.3.4 GESOP - Gestione operatore

Il processo di gestione operatore può essere vantaggiosamente realizzato come **automa** misto asincrono / sincrono, cioè attivato dagli eventi che costituiscono simboli di ingresso o dallo scadere di un periodo di *time-out*, ad esempio mediante una primitiva tipo

```
simbolo = GET (coda_automa, periodo)
```

dove il periodo, tenendo conto dei tempi di reazione umani, sarà scelto tra 0.1 s e 1 s.
E' interessante notare che per la gestione di macchine a prevalente controllo da parte dell'operatore, si può talvolta suggerire la fusione dell'automa di macchina con l'automa di operatore. In questi casi il processo GESOP assume anche le particolari funzionalità connesse con la gestione della macchina e dipendenti dall'applicazione, e quindi i simboli inseriti nella `coda_automa` saranno prodotti sia dai comandi dell'operatore, sia dagli eventi del funzionamento della macchina.
Le principali funzioni svolte da questo processo riguardano la **tastiera** e le **visualizzazioni**.

- Tastiera
  La console operatore può presentarsi come:
  -- **tastiera autonoma**, dotata delle funzionalità per generare una richiesta di interrupt e fornire direttamente il codice (ASCII o equivalente) del tasto premuto, con antirimbalzo (*debounce*), ecc. In questo caso la routine di risposta all'interrupt costituirà uno dei produttori di simboli in ingresso all'automa, che invierà con la primitiva

  ```
  PUT (coda_automa, tasto).
  ```

  -- **collezione di contatti organizzati** a matrice da scandire ciclicamente per rilevare i tasti premuti, effettuando a SW le funzioni di antirimbalzo, riconoscimento degli eventi, ecc. Questa funzionalità SW può essere svolta vantaggiosamente da un processo periodico dedicato alla tastiera (v. appendice di questo capitolo). La soluzione di affidare a GESOP questa funzione è meno generale ed elegante, anche se praticabile.
  -- **collezione di contatti non aggregati**. In questo caso i segnali ON/OFF sono trattabili come normali ingressi digitali ed i bit corrispondenti gestiti dal processo GESDIGI visto sopra che produce variabili di stato booleane. Se non si adottano provvedimenti *ad hoc* questi comandi vengono gestiti con funzioni combinatorie chiamate ad ogni scadenza di *time-out*, secondo la modalità tipica degli automi sincroni.
- Visualizzazioni
  Le modifiche della pagina video, con apertura e chiusura di finestre, ecc., saranno effettuate dalle funzioni associate all'ingresso e all'uscita degli stati dell'automa. Gli aggiornamenti di valori visualizzati, di messaggi, di curve e sinottici saranno effettuati nell'ambito delle funzioni cicliche di stato (*loop function*) con cadenza da una a quattro volte al secondo.

12.3.3.5 GESCOMM - Gestore comunicazioni

Questo processo si fa carico della gestione del protocollo di comunicazione, eseguendo tutte le funzioni dei livelli **inferiori** OSI (Trasporto, Rete, Data-link) che spesso vengono basate su una connessione di tipo *half-duplex* che prevede una rigida alternanza di trasmissioni e ricezioni.

Ai messaggi di questa alternanza di base viene generalmente affidato anche il compito di trasportare le immagini di **variabili da condividere** e quindi da aggiornare ciclicamente (considerate quindi **eventi assoluti**), senza ripetizione in caso di errori.
Il compito di aggiornamento automatico delle immagini di variabili "distribuite" può essere svolto da questo processo sulla base di apposite tabelle di corrispondenza tra la posizione dei valori nei messaggi (formato dei dati nei messaggi) e la posizione delle corrispondenti variabili in memoria (indirizzi nell'area dati globali).
Il processo GESCOMM interagisce con funzioni di interfaccia (gestione USART), e in particolare dovrà sincronizzarsi con le routine di risposta agli interrupt di ricezione e trasmissione che in molti casi costituiscono il meccanismo di sincronizzazione di basso livello e genera informazioni sulla regolarità della connessione (ad es. con *time-out* di mancata ricezione di messaggi).
Questo processo assumerà un comportamento un po' diverso se è previsto per svolgere la funzione di *master*, con iniziativa di iniziare trasmissioni, o di *slave* che si limita a rispondere su richiesta del master.

12.3.3.6 GESPACK - Gestore comunicazione di pacchetti

Questo processo si occupa dei livelli OSI Sessione e **superiori**, per le comunicazioni (considerate **eventi incrementali**) che richiedono la correzione dei messaggi errati, con la ricostruzione della corratta e completa sequenza di pacchetti. Ciò richiede una gestione di code di pacchetti, sia in ricezione che in trasmissione. A questo proposito va notato che occorrono tante code quante sono le sessioni di comunicazione aperte, e che in genere è opportuno prevedere sessioni diverse per i pacchetti urgenti contenenti **comandi**, da quelle destinate alla trasmissione, generalmente molto meno urgente, di **file**, configurazioni, ecc.
Questo processo si sincronizza con il processo GESCOMM oltre che con i processi produttori o consumatori dei pacchetti trasmessi e ricevuti.

12.4 SVILUPPO DI UN'APPLICAZIONE

Vediamo ora per grandi linee come si sviluppa un'applicazione con l'impostazione presentata in questo capitolo.
Innanzitutto notiamo che tutta una serie di elementi SW sono relativamente indipendenti dalla particolare applicazione, ed in particolare:

- processo GESDIGI
- processo ACQMIS
- processo REGOL
- funzioni standard dei BLOCCHI funzionali

Questi elementi, una volta sviluppati, costituiranno una libreria di base destinata a consolidarsi ed arricchirsi ad ogni nuovo progetto.
Nel seguito consideriamo quindi solo gli interventi del progettista per la particolare applicazione.

Si noti, nell'approccio proposto, il suggerimento di descrivere le funzionalità il più possibile con strutture dati (v. anche cap.11) invece che con strutture di programma. Le procedure esecutive in tal modo diventano degli "interpreti" delle strutture dati, con i seguenti vantaggi.

- Le procedure sono indipendenti dall'applicazione e quindi meglio collaudate.

- La funzionalità presenta una buona persistenza (v. cap.1)
- Le strutture dati costituiscono una specie di linguaggio orientato al problema e quindi più vicino alla forma con cui sono scritte le specifiche.
- Le estensioni sono molto facilitate, con la semplice aggiunta di elementi nelle strutture dati
- Si separa nettamente il collaudo dei programmi (interpreti) dal collaudo della funzionalità (tabelle di descrittori)
- Le procedure non standard sono ridotte al minimo e sono più facili da collaudare perchè sono inserite in un ambiente ben assestato e che "sta in piedi".
- Si può facilmente realizzare un sistema di sviluppo, o almeno una serie di strumenti, che facilitino molto il progettista nella compilazione dei descrittori che costituiscono le strutture dati.

12.4.1 Processo principale

Il processo principale potrà in molti casi essere costituito dallo stesso processo GESOP che abbiamo visto precedentemente. La "trama" generale di questo processo è relativamente indipendente dall'applicazione, contenendo nei punti strategici le chiamate ai sottoprogrammi che invece dovranno essere appositamente scritti.

12.4.1.1 Inizializzazioni

Oltre alle inizializzazioni standard, le parti dipendenti dall'applicazione sono principalmente le seguenti.

- Inizializzazioni delle variabili, verifica della TAM, eventuali copiature di tabelle da EPROM.
- Inizializzazioni delle interfacce specifiche dell'applicazione.
- Emissione sulle uscite dei valori di riposo.
- Impostazione del periodo di RTC voluto.

12.4.1.2 Configurazione dell'automa

La configurazione dell'automa richiede che si compilino le strutture dati degli stati e delle transizioni e che si scrivano le funzioni di stato e di transizione opportune.

12.4.1.3 Funzioni di visualizzazione

La gestione delle visualizzazioni, pur con criteri comuni, è una parte tipicamente dipendente dall'applicazione. In particolare si dovrà decidere le pagine video da adottare, quando (cioè in quali stati dell'automa) queste dovranno essere visualizzate e quali informazioni dovranno essere presentate in quali posizioni e con quale forma (numeri, barre, grafici, ecc.).
Anche qui ci sarà di grande vantaggio basarci su strutture dati che chiameremo Descrittori pagine video. Ogni descrittore di pagina sarà costituito da una tabella (array) di tanti elementi quante sono le informazioni da visualizzare. Ogni elemento sarà un record con (ad esempio) i seguenti campi principali.

- puntatore alla variabile che rappresenta l'informazione
- puntatore alla funzione che la scrive su video, che definisce implicitamente il formato
- coordinate della posizione su video
- attributi come colore, *reverse*, *blinking*, ecc.
- indicatore se la visualizzazione è costante, da aggiornare periodicamente, modificabile dall'operatore.

12.4.2 Data base

Come si è detto, qui per *data base* si intende una struttura a tabella che elenca tutti i blocchi funzionali che sono adottati nell'applicazione, e ne costituisce quindi il principale descrittore.
Nel compilare questa tabella si dovrà scegliere il nome di ogni blocco funzionale, precisare tutte le variabili di ingresso del blocco, e si dovrà fornire il valore di *default* di tutti i parametri.

12.4.3 Procedure applicative

12.4.3.1 Input fisici

Si scriverà una procedura che legge tutte le porte di semplici ingressi digitali e aggiorna conseguentemente le immagini di ingresso. Questa procedura sarà chiamata ciclicamente dal processo GESDIGI.
Per ogni interfaccia analogica (catena di misura) e per ogni interfaccia complessa, si scriverà una procedura (con eventuale corredo di risposte ad interrupt associati) chiamata dai blocchi funzionali di tipo MISURA.

12.4.3.2 Output fisici

Si scriverà una procedura che scrive su tutte le porte di semplici uscite digitali i valori contenuti nelle immagini di uscita. Questa procedura sarà chiamata ciclicamente dal processo GESDIGI.
Per ogni uscita analogica (o interfaccia complessa PFM, PWM, ecc., v. cap.7) si scriverà una procedura chiamata da blocchi funzionali di tipo REGOLATORE.

12.4.3.3 Funzioni personalizzate

Per tutti i blocchi funzionali, se non sono già disponibili le funzioni volute, si dovranno scrivere le procedure che svolgono la funzionalità richiesta dall'applicazione. Ad esempio calcolo di misure derivate, particolari filtri, particolari regolatori, particolari simulatori, ecc.

12.4.4 Comunicazioni

La struttura di base dei processi di comunicazione dipende dalla politica e dalla temporizzazione che si vuole o si deve adottare. Potremo quindi individuare alcune classi di applicazioni che richiederanno altrettante versioni dei processi GESCOMM e GESPACK.
Il formato dei messaggi, inteso come collocazione delle varie informazioni nell'ambito del campo "testo", sarà invece specifico della particolare applicazione e, come al solito, sarà vantaggiosamente descritto da opportune strutture dati.

12.4.4.1 Descrittori dei messaggi

Per ogni tipo di messaggio trasmesso (quindi da comporre) e ricevuto (quindi da interpretare) ci sarà una tabella che per ogni informazione contiene un elemento descritto con i seguenti campi.

- puntatore alla variabile in memoria
- tipo e numero byte della variabile
- puntatore alla funzione di trasformazione e copiatura

Per i messaggi da trasmettere le funzioni di trasformazione e copiatura leggono la variabile in memoria e, eventualmente trasformata opportunamente, la aggiungono al messaggio che si sta compilando.
Per i messaggi ricevuti invece le funzioni estraggono dal messaggio le informazioni e aggiornano i valori delle variabili in memoria.

12.5 APPENDICE

12.5.1 GESTIONE DI TASTIERA A MATRICE

L'acquisizione di tasti (o in generale di contatti) mediante la tecnica a matrice consente un notevole risparmio di porte di ingresso a parità di tasti gestiti.
Il cablaggio assume una struttura a matrice con tante righe (NR) quanti sono i bit di uscita e tante colonne (NC) quanti sono i bit di ingresso consentendo di acquisire lo stato ON/OFF di NR*NC contatti.
I contatti, in serie ad un diodo, sono collocati in corrispondenza delle intersezioni di righe e colonne, come nella fig. 12.2.

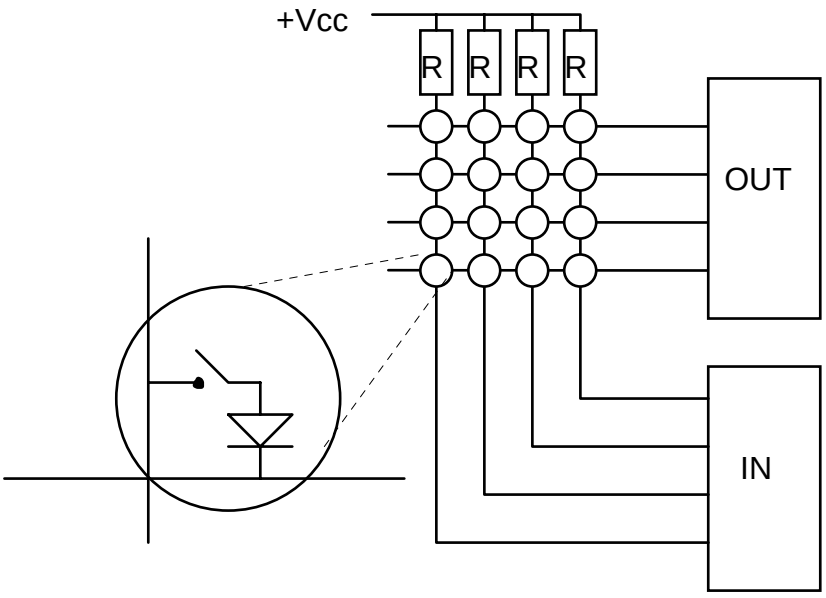

Fig. 12.2 - Schema di tastiera a matrice di contatti.

Con lo schema di fig. 12.2 si hanno 16 contatti acquisibili con 4 bit di uscita e 4 bit di ingresso.
Il procedimento di acquisizione è il seguente.

- Si emette una configurazione di bit composta di un solo 0 (riga da attivare) e tutti gli altri 1
- Si legge la porta di ingresso i cui bit sono 0 in corrispondenza dei contatti chiusi nella riga attivata.
- Si ripete ciclicamente attivando in sequenza tutte le righe.

NOTE.
Se si è certi che non sia lecito premere più di un tasto alla volta si possono risparmiare i diodi in serie ai contatti.
La scansione completa di tutte le righe deve avere un periodo minore del tempo di permanenza significativa di un contatto chiuso (o aperto), e generalmente va effettuata ogni 50-100 ms.
Se non si tratta di generici contatti, ma di una vera e propria tastiera, a valle dell'acquisizione periodica dello stato fisico (aperto / chiuso) dei contatti vengono

generalmente eseguite elaborazioni di livello superiore per una significativa generazione di eventi opportunamente codificati (ad es. con codice ASCII).

Eliminazione dei rimbalzi.

Viene ottenuta convalidando la pressione o il rilascio di un tasto solo alla seconda volta consecutiva che lo si trova nello stesso stato.

Eliminazione di tasti contemporanei.

I conflitti di contemporaneità sono risolti con criteri di precedenza (il primo tasto premuto inibisce i successivi) o di priorità spaziale (alcuni tasti prevalgono sempre sugli altri).

Generazione di un evento col codice del tasto premuto.

Si associa un codice (ASCII o convenzionale) al tasto premuto e si genera un evento con una delle normali tecniche (indicatore di tasto premuto, automa di sincronizzazione, inserimento in buffer circolare, ecc)

Generazione di *autorepeat*

Se un tasto rimane premuto oltre il tempo Tp si genera un evento con il relativo codice ogni Tr unità di tempo. Valori tipici sono Tp = 0,5 s e Tr = 0,1 s (fig.12.3).

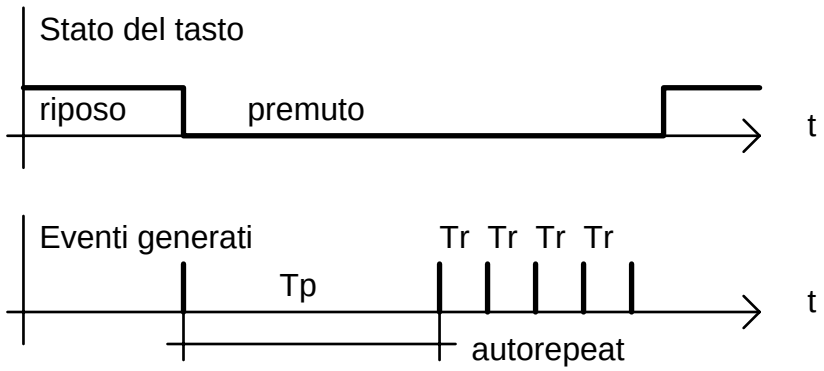

Fig. 12.3 - Andamento temporale della pressione di un tasto e corrispondente generazione di eventi con *autorepeat*.

12.6 ALTRE LETTURE

D.M. Auslander, C.H. Tham
Real-time software for control. Program examples in C.
Ed. Prentice-Hall 1990