Cap. 9

9. TECNICHE DI COMUNICAZIONE DIGITALE

9.1 INTRODUZIONE

9.1.1 SIMBOLI

Le informazioni destinate ad essere comunicate sono messe in corrispondenza biunivoca con sequenze di simboli (entità astratte) i quali a loro volta per essere "prodotti" dal mittente e "recepiti" dal destinatario sono rappresentati (concretamente) da fenomeni fisici detti **segnali** proprio per la loro funzione di veicolo di informazione.

La corrispondenza tra informazioni e sequenze di simboli è detta **codifica**. Le codifiche sono a priori **arbitrarie**, e ciò consente di ottimizzare fattori di merito significativi, ma devono poi essere **scrupolosamente seguite** da mittente e destinatario, pena la totale "incomprensione".

Tra gli obbiettivi, nello sfruttamento dell'arbitrarietà sopra citata, ricordiamo l'economicità dei dispositivi fisici da utilizzare per i segnali, la velocità di generazione delle sequenze di simboli, il margine di tolleranza ai disturbi, l'analogia con simboli di uso ormai consolidato negli anni o secoli.

9.1.2 SEGNALI E MEZZI TRASMISSIVI

Nel seguito specializzeremo il nostro discorso sui **segnali elettrici**, che sono tuttora i più versatili ed efficienti da gestire con le attuali tecnologie, e sui **simboli binari**, cioè i 'bit' che supportano le codifiche tipicamente utilizzate per i trasferimenti di informazioni tra macchine.

I mezzi trasmissivi, che consentono la propagazione dei segnali, sono semplici cavi, cavi speciali (ad es. coassiali) e l'etere, con le opportune apparecchiature, tipicamente elettroniche, di generazione, amplificazione, conversione, ecc. dei segnali.

Le caratteristiche dei mezzi trasmissivi, spesso collegati “in cascata” tra loro, determinano il tipo di segnali più adatti per una efficiente trasnissione, cioè in sostanza il modo con cui sono “concretizzati” i bit.

Nei calcolatori elettronici, ed in generale nei circuiti elettronici digitali, che costituiscono i tipici mittenti e destinatari di informazioni, i bit sono messi in corrispondenza con fasce di valori della tensione elettrica, usata quindi come segnale a livello, mentre nei vari mezzi trasmissivi le grandezze usate come segnale possono

essere ancora la tensione oppure la corrente, ma i valori dei bit non sono sempre associati a fasce di valori dei segnali e anzi spesso si ricorre a qualche tipo di **modulazione**, come vedremo in un successivo paragrafo. Per ora ricordiamo solo che un aspetto caratterizzante dei segnali è la loro banda di frequenze, che deve essere compatibile con il mezzo trasmissivo.

Un'analisi approfondita delle caratteristiche e dei tipi di mezzi trasmissivi esula dagli obbiettivi di queste note ed è oggetto di pubblicazioni specialistiche. Vengono quindi solo elencati caratteristiche significative e tipi principali di mezzi trasmissivi.

Le principali caratteristiche di un mezzo trasmissivo sono:

- - banda passante espressa in Hz (100 KHz..1000 MHz), da cui dipende la massima velocità di trasmissione;
- - attenuazione espressa in dB/Km (0.4..1.5), da cui dipende la massima distanza copribile senza ripetitori;
- - velocità di propagazione in Km/s (15000..150000);
- - impedenza caratteristica in Ohm (50..1000);
- - immunità ai disturbi.

I tipi principali di mezzi trasmissivi per segnali elettrici e distanze superiori a qualche metro, sono:

- - doppini intrecciati
- - doppini intrecciati e schermati
- - cavi coassiali

e per segnali elettromagnetici:

- - l'etere

E' opportuno notare che le **fibre ottiche** sono ormai diventate un mezzo trasmissivo molto interessante per alcune applicazioni, ma ciò non toglie validità alle considerazioni seguenti sui segnali elettrici, dato che possiamo ragionare ai “morsetti” elettrici dei dispositivi “terminali” di tali mezzi trasmissivi.
Pregi delle fibre ottiche sono immunità ai disturbi, isolamento galvanico, elevata banda passante.
Difetti delle fibre ottiche sono il costo elevato e la difficoltà di installazione (curvature, giunzioni e terminazioni), che richiede competenze e attrezzature specifiche.

NOTA. - Non si confonda la velocità di **trasmissione** di informazioni, che si misura in baud = simboli/s, con la velocità di **propagazione** del segnale che si misura in m/s ed è poco inferiore alla velocità della luce nel vuoto (300.000 Km/s).

9.2 PROBLEMATICHE GENERALI DELLA TRASMISSIONE DIGITALE

E' utile, prima di entrare in dettagli sulle tecniche di trasmissione di informazioni binarie, una sintetica rassegna delle problematiche da risolvere perchè tali trasmissioni siano effettuate correttamente.

9.2.1 DIREZIONI DEL TRASFERIMENTO DI INFORMAZIONI

Si distinguono i casi *simplex*, *half-duplex* e *full-duplex*.

- SIMPLEX
  Prevede un solo canale monodirezionale che consente trasferimenti da un mittente ad uno o più destinatari.

- HALF-DUPLEX
  Prevede un canale bidirezionale o due canali monodirezionali con orientamento l'uno opposto all'altro su cui, con **mutua esclusione**, ogni dispositivo può o trasmettere o ricevere. L'approccio *half-duplex* è tipico delle reti locali e viene spesso utilizzato anche quando si dispone di due canali che consentirebbero il sistema *full-duplex*, dato che la rigida alternanza di trasmissione e ricezione semplifica molto i protocolli di conferma o ripetizione dei messaggi errati.

- FULL-DUPLEX
  Richiede l'uso di due canali distinti per ricezione e trasmissione, che così possono avvenire **contemporaneamente**. Mentre molte interfacce seriali (Es. RS-232) sono fisicamente di questo tipo, come sopra accennato si preferisce spesso utilizzarle nel modo *half-duplex*.

9.2.2 BIT, CARATTERI E MESSAGGI

Le informazioni da trasmettere sono generalmente strutturate in modo gerarchico. Avremo quindi gruppi di **bit** (7, 8 o 9) che costituiscono i cosiddetti **caratteri**. Si noti che in molti casi si tratta di usuali caratteri in codifica ASCII, ma talvollta si tratta piuttosto di *byte* che possono quindi rappresentare informazioni con diverse codifiche.
Opportune sequenze di caratteri costituiscono i cosiddetti **messaggi**. L'organizzazione dei caratteri in messaggi non è intrinsecamente necessaria, ma è in genere suggerita da esigenze di protocollo di comunicazione.
Ad esempio in un collegamento *half-duplex* il termine di un messaggio dà al partner l'opportunità di rispondere, ed in genere nelle reti i messaggi (detti spesso pacchetti) consentono un intreccio di diverse comunicazioni "concorrenti". Ci sono poi esigenze connesse con il riconoscimento di errori e la loro correzione mediante ripetizione che limitano ad una massima lunghezza conveniente le sequenze di caratteri consecutivi.

9.2.3 SINCRONIZZAZIONE E RICONOSCIMENTO DEI SIMBOLI

Dato che i simboli sono trasmessi sequenzialmente, è necessaria una **sincronizzazione** tra mittente e destinatario, cioè un accordo su quali **finestre temporali** sono destinate alla rappresentazione di quali simboli.
Generalmente si usano **segnali ausiliari**, che con le loro commutazioni indicano gli istanti significativi per la sincronizzazione.
Nelle comunicazioni parallele, o comunque a breve distanza, è generalmente conveniente adottare uno o più cavi aggiuntivi per trasportare questi segnali ausiliari.
Invece nelle comunicazioni (seriali) a distanze elevate è preferibile associare ad uno stesso segnale sia il compito di convogliare le informazioni significative che quello di fornire le temporizzazioni. In questi casi i segnali ausiliari sono generati localmente

presso il mittente e presso il destinatario, in modo tale da seguire le necessarie temporizzazioni.

Si noti che la struttura gerarchica delle informazioni, organizzate a bit, caratteri e messaggi, richiede una corrispondente gerarchia di sincronizzazioni che consenta di riconoscere oltre agli istanti in cui sono validi i singoli bit, anche quali gruppi di bit costituiscono un carattere e quali sequenze di caratteri formano un messaggio.

9.2.4 PREVENZIONE DEGLI ERRORI

Una fonte di errori, cioè di errato **riconoscimento** di simboli, è tipicamente costituita dai **disturbi**, cioè fenomeni parassiti che **deformano** i segnali. Sono quindi importanti tecniche di:

riduzione dei fenomeni disturbanti,
protezione dei segnali da tali fenomeni,
scelta di segnali particolarmente “robusti”.

9.2.5 RICONOSCIMENTO E CORREZIONE DEGLI ERRORI

In ogni caso gli errori introdotti dai disturbi sono riducibili ma non completamente eliminabili. Nelle specifiche di progetto delle applicazioni che comportano trasmissione di informazioni si dovrà perciò accettare un certo **tasso d'errore non nullo**.
Per tasso d'errore si intende solitamente il rapporto tra simboli ricevuti in modo errato e numero totale di simboli trasmessi.

E' interessante notare che in genere si è disposti ad accettare un tasso di errori relativamente elevato, purchè la maggior parte di tali errori vengano **riconosciuti** come tali.
Infine si vuole che, almeno per alcuni errori, al loro riconoscimento faccia seguito la **ricostruzione** delle informazioni corrette.

Avremo quindi, in ordine non decrescente di valore, i seguenti tassi:

- tasso di errori non rilevati
- tasso di errori riconosciuti con scarto delle informazioni errate
- tasso di errori riconosciuti con ricostruzione delle informazioni corrette.

Si noti che le tecniche di **riconoscimento** degli errori sono basate sull'uso di simboli **ridondanti**, mentre le tecniche di correzione sono basate su ridondanza **intrinseca** (codici autocorrettori) o a **richiesta** (ripetizione dei messaggi riconosciuti come errati).

9.2.6 GESTIONE SPAZIALE E TEMPORALE DELLE COMUNICAZIONI

Un aspetto molto importante, soprattutto quando si ha a che fare con comunicazioni tra più di due partner (cioè nel caso di reti), è costituito dal coordinamento temporale e/o spaziale delle varie azioni di comunicazione, affinchè vengano evitati conflitti di accesso ai mezzi trasmissivi e le informazioni giungano alla loro destinazione voluta in un tempo accettabile e nella sequenza corretta. La gestione delle sequenze temporali è affidata ad opportuni **protocolli**, mentre la gestione spaziale è effettuata mediante opportune politiche di **instradamento** (*routing*) dei messaggi.

Passiamo ora ad esaminare un po' più in concreto i principali aspetti tecnici delle più diffuse comunicazioni di informazioni con codifica digitale.

9.3 COMUNICAZIONI PARALLELE

Nei collegamenti a breve destanza (fino a pochi metri), quando si vuole rendere veloce la trasmissione si utilizzano le comunicazioni dette “parallele” perchè gruppi di bit, in genere byte (8 bit) o word (16 bit), vengono trasmessi contemporaneamente su altrettanti conduttori.

In ogni caso però i byte o le word vengono a loro volta trasmessi in sequenza: si tratta cioè, a rigore, di trasmissioni **bit-parallele** e **byte-seriali**.

A parte le connessioni interne tra i moduli dei calcolatori, spesso organizzate appunto a bus di sistema, le più diffuse connessioni parallele sono quelle descritte dagli standard IEEE-488 e “Centronics” brevemente presentate nel seguito.

Si noti l’importante differenza tra i bus di sistema e quelli di comunicazione, relativamente alla tecnica di indirizzamento:

- - nei bus di sistema il *master* impone su **apposite linee** (bus indirizzi) l’indirizzo del partner,
- - nei bus di comunicazione il **mittente** inserisce nei **dati** trasmessi anche l’indirizzo del destinatario.

Nelle connessioni parallele la **sincronizzazione** dei trasferimenti (di *byte* o di *word*) viene effettuata con **segnali ausiliari** appositi, in genere con una tecnica di *handshaking* del tipo di quella riportata sotto.

Questa tecnica consente di adeguare le velocità del trasmettitore e del ricevitore, in modo da non perdere informazioni e tenendo conto delle tolleranze sui diversi tempi di commutazione dei vari bit in parallelo.

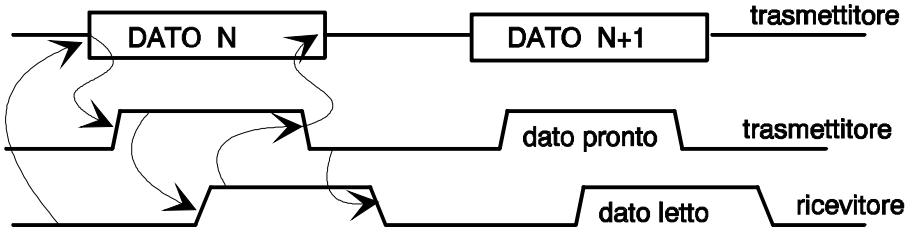


Fig. 3.1 - Segnali di sincronizzazione di trasferimenti dati.

9.3.1 COMUNICAZIONI CON BUS IEEE-488

Il bus standard IEEE-488 è detto anche GP-IB (cioè *General Purpose Interface Bus*) ed è stato proposto dalla Hewlett Packard come sistema di interconnessione per strumentazione da laboratorio, adatto a realizzare una specie di rete locale tra calcolatori e strumenti “intelligenti”, cioè gestiti da microprocessori.

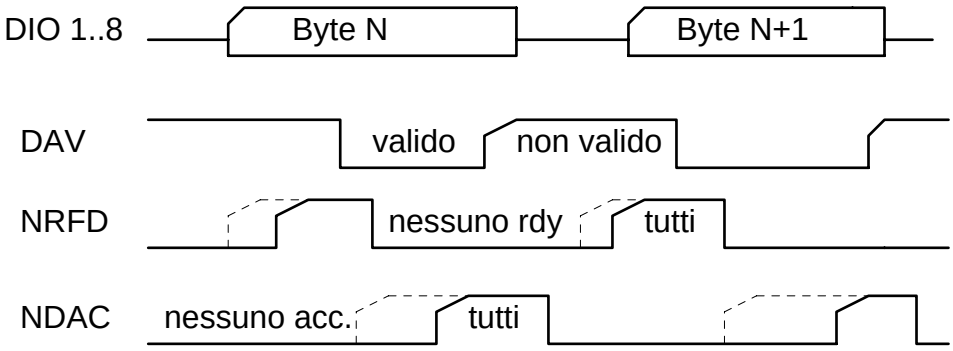
Le caratteristiche tecniche principali di questo tipo di collegamento sono:

BUS costituito da 16 linee di segnale così suddivise:
- 8 linee bus dati
- 3 linee di controllo (*handshaking*)
- 5 linee di gestione delle interfacce

I segnali sono a livelli TTL con logica negativa (il livello basso indica asserzione del livello dominante del segnale) con pilotaggio "*open-collector*" che consente la funzione di "OR di collettore" tra diversi dispositivi.

Sul bus sono previsti dispositivi che possono assumere i ruoli detti "***talker***" (trasmittente) "***controller***" (controllore) e "***listener***" (ricevente). Mentre lo standard specifica l'uso dei segnali per effettuare correttamente le trasmissioni, lascia libero il formato dei messaggi che i vari tipi di dispositivi sono in grado di interpretare.

La velocità di trasferimento dei byte viene adeguata, tramite *handshaking*, alle caratteristiche dei vari dispositivi in comunicazione e può arrivare a 1 Mbyte/s. E' interessante notare come, con la tecnica dell'OR di collettore, si ottenga un adeguamento automatico della velocità di comunicazione al più lento tra i vari dispositivi coinvolti dal trasferimento.



DIO 1..8 8 bit DATI bidirezionali
DAV Data Available
NRFD Not Ready for Data
NDAC Not Data Accepted

BUS IEEE-488 Segnali di Handshake

Fig, 3.2 Handshake su bus IEEE-488

NRFD - Segnale in logica negativa gestito *open-collector* dai **riceventi**, i quali asseriscono il valore basso (dominante) se non sono pronti ad accettare il prossimo dato. Il livello di questo segnale sarà quindi **alto solo se tutti i riceventi sono pronti**. Ciò consente di attendere anche il più lento dei riceventi prima di trasmettere il prossimo byte.
DIO - Il livello di questi 8 bit è imposto dal **mittente**, e rappresenta un byte da trasmettere.

DAV - Segnale in logica negativa gestito dal **mittente**, che asserisce il livello basso solo dopo aver impostato i bit DIO e aver rilevato il segnale NRFD alto.
NDAC - Segnale in logica negativa gestito *open-collector* dai **riceventi**, i quali asseriscono il valore basso (dominante) se non hanno ancora acquisito il nuovo byte. Solo quando si ha un livello alto su questo segnale il mittente potrà rilasciare DAV a livello alto (indicando dati non più validi) e modificare il byte DIO con il nuovo valore.

9.3.2 INTERFACCIA CENTRONICS

Costituisce uno **standard di fatto**, introdotto dalla grande diffusione dei *Personal Computer* che in genere sono dotati di un'interfaccia parallela per stampanti (di tipo Centronics, appunto) che si presta anche a collegamenti veloci a breve distanza.

I segnali sono a livelli TTL e prevedono 8 linee per i dati e alcuni segnali di controllo, disposti su connettori di tipo Cannon a 25 contatti.

Mentre nelle prime interfacce di questo tipo i dati erano previsti solo in uscita dal calcolatore, in molte realizzazioni più recenti si adotta un collegamento bidirezionale (*half-duplex*) per gli 8 bit dei dati.

9.4 COMUNICAZIONI SERIALI.

Generalmente, soprattutto per distanze superiori a pochi metri, le trasmissioni sono di tipo seriale, con l'uso di un singolo segnale per trasmettere un bit alla volta in successione temporale.

Nel seguito vengono brevemente presentate le caratteristiche qualificanti di vari tipi di trasmissione.

9.4.1 VELOCITA' DI TRASMISSIONE (baud rate)

La velocità di trasmissione (*baud rate*), viene espressa in simboli al secondo (***baud***) o in bit/s. Si noti che se si utilizzano segnali binari (simboli a due valori) i valori di baud e bit/s coincidono.
I valori generalmente utilizzati sono:

- 75, 110, 150, 300, 600 bit/s
  basse velocità standard, usate per telex, teletype, accoppiatori acustici telefonici, ormai in disuso.
- 1200, 2400, 4800, 9600 bit/s
  velocità medie standard, molto diffuse nelle comunicazioni con terminali e unità periferiche, microcontrollori, PLC, ecc. e adatte anche alle linee telefoniche con l'uso di opportuni dispositivi MODEM.
- 19200, 38400 bit/s
  velocità elevate standard, raramente utilizzate, ma di interesse crescente.
- 100 .. 1000 Kbit/s

velocità medie per collegamenti speciali, bus di campo, ecc.

- 1 .. 10 Mbit/s

velocità elevate utilizzate in genere in reti locali (LAN).

9.4.2 TECNICA DI SINCRONIZZAZIONE DI BIT

Obbiettivo della sincronizzazione di bit è di mettere il ricevente in grado di conoscere le finestre temporali in cui sono validi, e quindi campionabili, i livelli del segnale ricevuto.

Le tecniche utilizzate per la sincronizzazione a livello di bit sono classificabili in isocrona, asincrona e sincrona come nel seguito descritte, con preferenza per la tecnica asincrona per le basse velocità standard e collegamenti tra apparecchiature di diversi costruttori, come PLC, strumenti con microprocessore, stampanti, plotter, ecc.

In molti **bus di campo** e nelle **reti locali** viene invece preferita la sincronizzazione di bit di tipo sincrono.

9.4.2.1 TRASMISSIONE ISOCRONA

La sincronizzazione di bit è basata su un **unico** segnale di *clock*, tipicamente ad onda quadra, in **comune** tra mittente e destinatario. Sui fronti di salita del *clock* (fronti di **commutazione**) il mittente imposta il livello del prossimo bit che il destinatario campiona sui fronti di discesa (fronti di **campionamento**) (fig. 9.1).

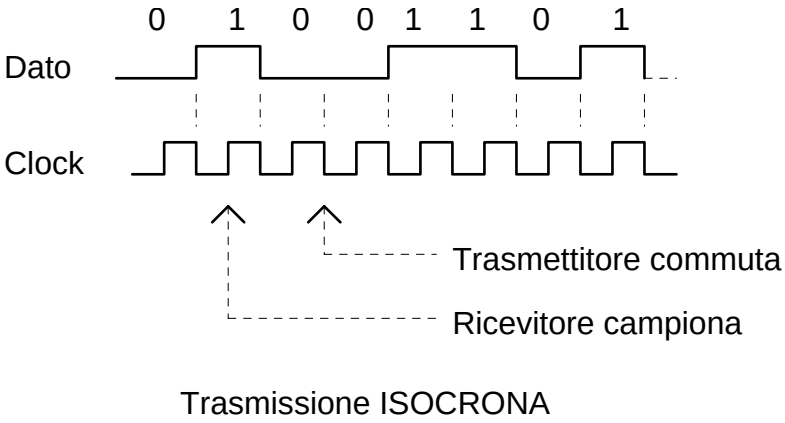


Fig. 9.1 - Trasmissione isocrona.

E' necessario effettuare una sincronizzazione di *byte* ottenibile con speciali "**caratteri di sincronismo**", con segnali ausiliari o con una tecnica simile a quella asincrona presentata nel seguito.

Non è richiesta precisione nè stabilità sulla frequenza del clock e le uniche specifiche temporali precisano i valori minimi del periodo "alto" e "basso" del clock in funzione del tempo di propagazione dei segnali.
A proposito del tempo di propagazione si devono distinguere i due casi seguenti.

1. Clock generato **presso il trasmettitore**. In questo caso il clock e i dati si propagano nella stessa direzione e quindi mantengono una corretta relazione di fase qualunque sia la distanza di trasmissione.
2. Clock generato **presso il ricevitore**. In questo caso presso il ricevitore i dati si presentano con un ritardo rispetto al clock, pari a due volte il tempo di propagazione

che è proporzionale alla distanza. Il corretto campionamento dei dati richiede quindi che il clock presenti tra il fronte di commutazione e il fronte di campionamento una distanza temporale maggiore (con un certo margine) del doppio del tempo di propagazione. Questa esigenza impone un limite alla velocità di trasmissione.

Per le trasmissioni isocrone occorre un conduttore a parte per il clock, che tra l'altro è critico rispetto ai disturbi, e si deve tener conto dei tempi di propagazione, come appena discusso, per cui questa tecnica viene generalmente riservata a collegamenti relativamente brevi.
Un esempio tipico di comunicazione isocrona è costituito dalla connessione tra apparecchiature digitali e modem secondo lo standard RS-232 completo.

9.4.2.2 TRASMISSIONE ISOCRONA CON HANDSHAKING

Costituisce una variante della tecnica isocrona, di cui mantiene l'impiego di segnali di temporizzazione separati dai dati, che però vengono gestiti da entrambi i partner, per consentire l'adattamento della cadenza di trasmissione alle disponibilità sia del trasmittente che del ricevente.

- Variante a doppio segnale di handshaking

  un segnale indica dato trasmesso e l'altro dato ricevuto, con la normale tecnica di *handshacking* vista precedentemente.
  Utilizzata in trasmissioni parallele a bus asincrono. (Es. Bus IEEE-488)

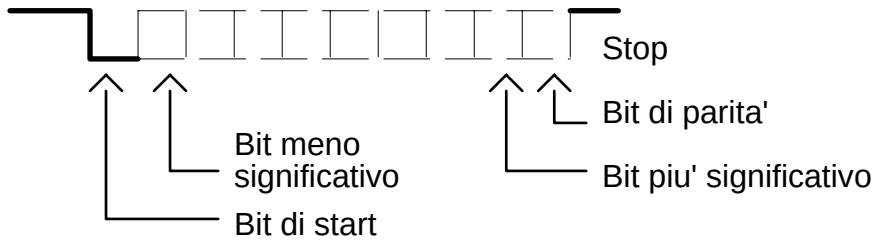
- Variante a clock unico condizionato

  un segnale unico, ma gestito *open collector* dai partner, consente la sincronizzazione con *handshaking*. Un esempio è il bus I2C della Philips, in cui il ricevente può mantenere a livello basso il clock finchè non è pronto a ricevere il prossimo bit, ritardandone così, se necessario, la trasmissione. A sua volta il trasmittente quando percepisce il fronte di salita (di commutazione) del clock, dopo aver impostato il livello del bit di dato forza sul clock un livello basso (fronte di campionamento) per un certo tempo.

9.4.2.3 TRASMISSIONE ASINCRONA

Mittente e destinatario devono essere corredati ognuno di un proprio generatore di clock.
Questi oscillatori devono avere frequenze **simili** (+/- 1..3 %) e **stabili**, in genere scelte tra una serie di valori standard (300, 600, 1200, 2400, 4800, 9600 Hz).
Ovviamente per quanto abbiano frequenze simili i due clock non saranno generalmente in fase tra loro, richiedendo quindi periodiche sincronizzazioni.
La sincronizzazione dei due clock viene effettuata **ad ogni byte trasmesso**, sul fronte di commutazione costituito dal primo bit, detto "**bit di start**", mentre tra un byte ed il successivo viene inserito un **livello di "stop"** della durata minima di uno (talvolta due) bit (fig. 9.2).
Il trasmittente non è tenuto a trasmettere i caratteri a ritmo "serrato", ma può dilazionarne a piacere la trasmissione. Naturalmente se in sede di progetto del protocollo di comunicazione si adotta un periodo di pausa superiore ad un certo T-out come indicatore di fine messaggio, non si dovrà superare questo intervallo tra un carattere e l'altro all'interno del messaggio.

Una volta sincronizzati i due clock essi si comportano come l'unico clock del caso isocrono e la similitudine delle loro frequenze deve essere tale da garantire che per tutti i bit del carattere lo sfasamento non superi l'intervallo di +/- metà della durata di un bit.
Infatti i fronti di **commutazione** sono forniti dal clock del **trasmittente**, mentre i fronti di **campionamento** sono forniti dal clock del **ricevente** e, per una corretta acquisizione, dovranno cadere all'interno della finestra temporale del bit.
Si noti che **l'errore di fase** accettabile, che nel caso teorico corrisponde a differenza tra le frequenze di +/-5% (su 10 bit complessivi di carattere), si riduce quando i **tempi di commutazione** dei bit non sono più considerabili come nulli. Soprattutto alle elevate velocità di trasmissione il peso **percentuale** del tempo di commutazione rispetto al periodo di bit diventa anche notevole. Si ricordi che più la banda del mezzo trasmissivo è **limitata** maggiori sono i tempi di commutazione dei segnali.



Trasmissione ASINCRONA con caratteri a 8 bit + parita'

Fig. 9.2 Trasmissione asincrona.

La sincronizzazione di byte avviene contestualmente alla sincronizzazione del primo bit e quindi non richiede tecniche particolari.

La tecnica asincrona è molto utilizzata, per la sua semplicità, per brevi collegamenti punto - punto a velocità fino a 19200 bit/s, tipicamente per mouse, stampanti, PLC, ecc.

9.4.2.4 TRASMISSIONE SINCRONA

Nella trasmissione sincrona il mittente ha un suo clock che oscilla liberamente, mentre il destinatario è dotato di generatore di clock con frequenza simile a quella del mittente, ma con la capacità di **aggancio di fase** (PLL = *Phase Lock Loop*) con il clock del trasmittente, di cui deduce appunto la fase dalle commutazioni dei bit ricevuti.

La sincronizzazione di bit avviene automaticamente dopo alcune commutazioni che consentono l'aggancio di fase, mentre la sincronizzazione di byte viene normalmente basata sul riconoscimento di appositi **caratteri di sincronismo** che il mittente antepone ad ogni messaggio in numero sufficiente (3..5). In molti casi i caratteri di sincronismo possono essere scelti dal progettista, il quale dovrà aver cura di scegliere caratteri ricchi di commutazioni, per favorire l'aggancio di fase, ma anche con una sequenza di zeri e uni che sia univocamente riscontrabile per la sincronizzazione di carattere. Ad esempio il carattere 10101010 va molto bene per l'aggancio di fase ma non è certo riconoscibile univocamente in una sequenza di bit, mentre è ad esempio opportuna una scelta del tipo 10100110. Si ricordi infatti che i bit iniziali di ogni messaggio non sono correttamente ricevuti finchè non è avvenuto l'aggancio di fase.

I bit che compongono un messaggio devono poi essere trasmessi **senza pause** ed evitando che si abbiano lunghe sequenze senza commutazioni (1111111 oppure 0000000) che farebbero perdere l'aggancio di fase. Se necessario il trasmettitore inserisce automaticamente bit aggiuntivi di valore opposto che il destinatario è in grado di riconoscere ed eliminare.
Questa tecnica richiede dispositivi più complessi delle precedenti (ma ormai la complessità circuitale non costituisce un costo eccessivo) consentendo però una maggiore efficienza e velocità di trasmissione.

- Efficienza maggiore del 20% rispetto alla trasmissione asincrona perchè si evitano i bit di start e di stop.
- Velocità perchè l'aggancio di fase segue più accuratamente le effettive temporizzazioni dei bit, senza doversi affidare solo a precisione e stabilità dell'oscillatore del clock.

La trasmissione sincrona è tipica nelle trasmissioni ad elevate velocità ed in generale nelle reti di calcolatori. In particolare viene utilizzata per massimizzare lo sfruttamento della limitata banda telefonica per le trasmissioni via modem.

9.4.3 SINCRONIZZAZIONE DI CARATTERE E MESSAGGIO

Nelle più semplici comunicazioni asincrone la sincronizzazione di carattere è automatica (v. sopra) mentre quella di messaggio viene spesso effettuata con l'inserimento volontario di un intervallo temporale di "silenzio" (*time-out* di fine messaggio).

Nelle comunicazioni sincrone ed isocrone si utilizzano **caratteri di sincronismo** (SYNC) e specifici **caratteri di inizio messaggio** (SOH = *Start Of Header*) che consentono un inequivocabile riconoscimento di ogni nuovo messaggio e della sua parte significativa (al netto dei SYNC).

9.4.4 SEGNALI, CODIFICA E MODULAZIONE

Data la complessità dell'argomento e la ricchezza di varianti e relative problematiche, che vanno oltre gli scopi di queste note, si propone nel seguito una classificazione semplificata delle tecniche di segnalazione di sequenze di bit.

- **SEGNALI A DUE VALORI** (detti in "**banda base**")
  - tensioni con livelli TTL 0 / 5 V
  - tensioni bipolari -6 / +6 V (Es. RS-232)
  - tensioni differenziali (Es. RS-485)
  - "*loop* di corrente" 0 / 20 mA (Es. le vecchie *Teletype*)

Questi segnali sono utilizzati con cavi (eventualmente coassiali) dedicati alla trasmissione su brevi distanze e in reti locali.
Nella fig. 9.3 sono riportati esempi di forme d'onda delle più diffuse "codifiche in banda base", per la stessa sequenza di bit (1001101).
Si noti che la NRZ (*Not Return to Zero*) è la più utilizzata ed è quella a cui abbiamo fatto riferimento finora. Le "codifiche" Manchester sono utilizzate nelle trasmissioni sincrone per la loro caratteristica di presentare almeno un fronte di commutazione per ogni bit, favorendo l'aggancio di fase ed il suo mantenimento, ma ovviamente occupano

una maggior banda di frequenza, a pari velocità di trasmissione, rispetto a NRZ e NRZI (=*Inverted*)..

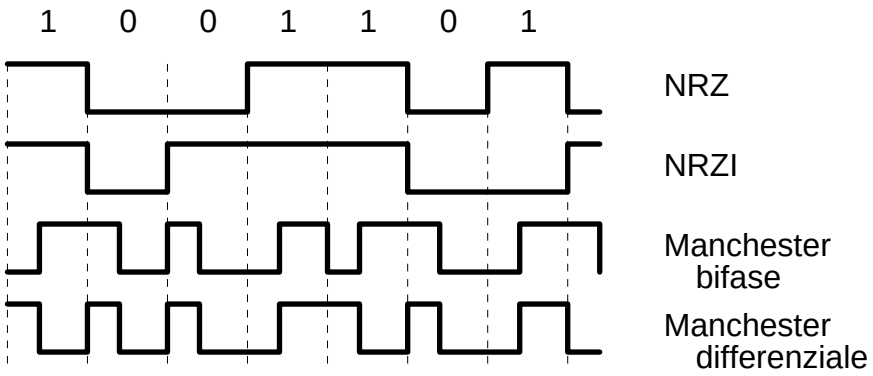


Fig. 9.3 Codifiche binarie in banda base.

- **SEGNALI SINUSOIDALI**
  - modulati in ampiezza (poco usati)
  - a modulazione di frequenza
  - a modulazione di fase

Questi segnali sono utilizzati per comunicazioni con modem (modulatori/demodulatori) su linee telefoniche commutate, su cavi coassiali o su ponti radio.
Al modem in trasmissione viene fornito il segnale in banda base, come abbiamo visto sopra, ed il modem in ricezione ne riproduce abbastanza fedelmente l'andamento, cosicchè ai morsetti "digitali" la modulazione risulta "trasparente".

9.4.5 GESTIONE DEGLI ERRORI

- RICONOSCIMENTO di errori a livello di **carattere**.
Viene spesso adottata, a livello di carattere, la tecnica del bit di parità, che consiste nell'associare ad ogni carattere un bit aggiuntivo tale da rendere pari (*even*) o dispari (*odd*) il numero totale di "uni".
Consente di riconoscere errori che modificano un numero **dispari di bit**.
Vengono talvolta usate particolari codifiche (ad es. codice autocorrettore di Hamming) che con una maggior ridondanza consentono di riconoscere errori su N bit e di correggere errori su M bit, con $N > M$. Spesso $N = 2$ e $M = 1$.

- RICONOSCIMENTO di errori a livello di **messaggio**.
Si utilizzano **caratteri aggiuntivi**, trasmessi alla fine dei messaggi, che il mittente calcola come funzione dei caratteri trasmessi ed il ricevente verifica con la stessa regola.
Le tecniche piu' usate sono:

- - CRC (*Cyclic Redundancy Check*) calcolato generalmente su 16 bit e trasmesso come 2 caratteri
- - checksum = somma dei caratteri, spesso effettuata su 8 bit, cioè modulo 256.

- - parità longitudinale, in cui ogni bit del carattere aggiuntivo costituisce il bit di parità dei bit omologhi dei vari caratteri del messaggio.

- CORREZIONE di errori.
L'uso dei codici autocorrettori sopra citati (Hamming), dato il costo in termini di ridondanza e di maggiore complessità circuitale che l'uso di tali codici impone, è generalmente riservato ad applicazioni critiche in presenza di collegamento di tipo *simplex* (v. sopra).
Normalmente la correzione degli errori rilevati viene effettuata con opportuni protocolli di richiesta di ritrasmissione del messaggio in cui si è rilevato un errore.

E' importante notare che la **lunghezza ottimale** dei messaggi dipende dal **tasso di errore** del canale di trasmissione usato, dato il costo della ridondanza introdotta dalla ritrasmissione e dalla probabilità che anche la ripetizione sia a sua volta affetta da errori.

9.4.6 STANDARD DI INTERFACCIA

Nel seguito vengono riportate sinteticamente le principali caratteristiche dei più diffusi standard di interfacciamento per trasmissioni seriali.

Gli standard specificano le caratteristiche **elettriche** e **temporali** dei segnali, con soglie max e min, il loro "**significato**" e talvolta anche il tipo di **connettori** e la disposizione sui contatti dei vari segnali.

9.4.6.1 RS-232-C Agosto 1969

E' uno standard molto diffuso, sia pure con varianti e semplificazioni.
Inizialmente è stato proposto per collegare dispositivi detti
DTE = *Data Terminal Equipment* (calcolatori, stampanti, ecc.)
con modem (tipicamente per linee telefoniche) detti
DCE = *Data Communications Equipment.*

SEGNALI ELETTRICI
0 = ON --> V > +3 V
1 = OFF --> V < -3 V
DISTANZA < 15 m
VELOCITA' < 20000 bit / s
SEQUENZA Bit meno signif.... Bit piu' signif. Event. parita'
CONNETTORE DTE Cannon maschio 25 contatti

DISPOSIZIONE DEI SEGNALI PRINCIPALI E SIGNIFICATO.

piedino	nome	significato
1	GND	massa di protezione
2	TXD	dati trasmessi (Transmitted Data) A questo morsetto sono passati, con la temporizzazione isocrona rispetto a TXC, i bit da trasmettere forniti al modem.
3	RXD	dati ricevuti (Received Data) Da questo morsetto si campionano con la temporizzazione isocrona rispetto a RXC, i bit ricevuti ed emessi dal modem.
4	RTS	richiesta di invio (Request To Send) Questo segnale deve essere asserito ON quando il modem deve porsi in stato di trasmissione, rilasciato OFF per passare in ricezione (la comunicazione è *half-duplex*).
5	CTS	consenso all'invio (Clear To Send) Con questo segnale il modem conferma di essere in stato di trasmissione, consentendo così l'invio dei bit.
6	DSR	*data set* (modem) pronto (Data Set Ready) Questo segnale emesso dal modem indica che esso è acceso e funzionante.
7	0-V	massa del segnale
8	CD	presenza di portante (Carrier Detect) Con questo segnale il modem indica se sta ricevendo la portante, cioè che il partner è attivo e in stato di trasmissione.
15	TXC	clock di trasmissione (Transmit Clock) Segnale generato dal modem, di temporizzazione isocrona per la commutazione dei bit in trasmissione.
17	RXC	clock di ricezione (Receive Clock) Segnale generato dal modem, di temporizzazione isocrona per il campionamento dei bit in ricezione.
20	DTR	*data terminal* (calcolatore) pronto (Data Terminal Ready) Segnale con cui il calcolatore (o il terminale) indica al modem di essere in funzione, agli effetti delle comunicazioni.

E' importante notare che questo standard viene spesso adottato in forma ridotta anche per connessione diretta tra dispositivi di tipo DTE, cioè senza interposizione di modem, ma con l'uso di un cavo (detto "*null modem*") in cui sono incrociati i collegamenti 2-3, 15-17, ecc.
Nelle versioni ridotte, in cui non c'è la necessità di gestire il tipico funzionamento *half-duplex* dei modem, spesso i costruttori assegnano significati leggermente diversi ai segnali (RTS,CTS,DSR,CD), creando talora qualche problema di compatibilità.

9.4.6.2 RS-422-A Dicembre 1978

Standard nato per superare i limiti di distanza e di velocità dell' EIA-RS-232, utilizzando segnali differenziali bilanciati di cui precisa le caratteristiche elettriche per velocità fino a 2 Mbit / s in ambienti anche relativamente rumorosi.
Prevede la connessione di un solo trasmettitore, ma eventualmente anche più ricevitori, sulla stessa linea.

9.4.6.3 RS-423-A Dicembre 1978

Simile al precedente, ma basato su segnali non bilanciati adatti per velocità fino a 100 Kbit / s ed in ambienti a basso livello di rumore.

9.4.6.4 RS-449 Novembre 1977

Questo standard si basa sui precedenti 422 e 423 per le caratteristiche elettriche e precisa caratteristiche meccaniche e disposizione dei segnali sui connettori di interfaccia.

9.4.6.5 RS-485

E' uno standard molto diffuso e limitato agli aspetti elettrici, che costituisce una interessante variante del 422 in quanto consente collegamenti tra più trasmettitori (*tri-state*) e ricevitori operanti sulla stessa linea, con velocità fino a 10 Mbit / s (con distanze < *M* m) e cavi lunghi fino a 1200 m (con velocità inferiori a *V* Kbit/s), dove i valori *M* e *V* sono correlati come nella figura seguente (fig. 9.4).

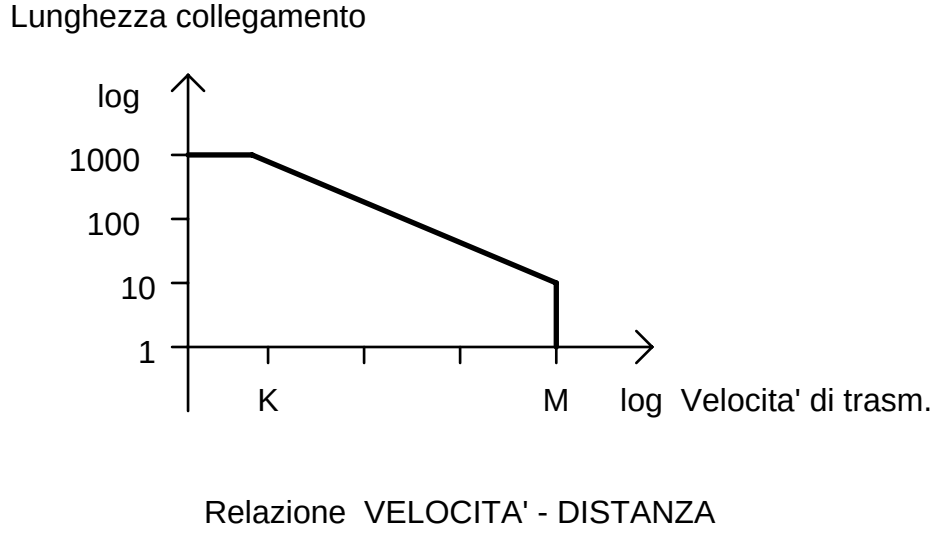


Fig. 9.4 Relazione velocità - distanza.

distanza velocità su cavo *twisted pair* bilanciato
m 1200 100kbit/s
m 15 10 Mbit/s
lineare su scale log/log

Questo standard elettrico sta assumendo una notevole diffusione nelle interconnessioni con topologia a "**bus**" utilizzate in impianti industriali per collegare calcolatori, PLC e strumentazione e viene spesso incorporato in standard che specificano anche i livelli superiori di funzionalità.

9.4.7 CIRCUITI USART

Le trasmissioni seriali, sincrone ed asincrone, sono così diffuse da suggerire la produzione di circuiti integrati (ad elevata scala di integrazione) detti appunto USART, cioè *Universal Synchronous Asynchronous Receiver Transmitter* in genere studiati per un facile interfacciamento con i più diffusi microprocessori. Spesso poi tali circuiti sono addirittura integrati con lo stesso processore nei "*microcontroller*" e nei "*single chip computer*".
Si tratta di circuiti integrati ad elevata densità che svolgono le operazioni necessarie per i livelli inferiori della gerarchia ISO-OSI nelle trasmissioni seriali.

Si noti che si parla di UART nel caso in cui i circuiti siano previsti per la sola, più semplice e più usata, comunicazione asincrona, realizzando così un sottoinsieme delle funzioni delle USART.

Senza la pretesa di discutere in dettaglio tutti i problemi di una gestione di comunicazioni seriali, si ritiene però utile una breve presentazione della struttura, funzionalità e gestione delle tipiche interfacce con USART.

9.4.7.1 REGISTRI DELLE USART

Le USART sono dotate internamente di un certo numero di registri, generalmente di 8 bit, che possono essere classificati come segue:

- buffer trasmissione
- shift trasmissione
- shift ricezione
- buffer ricezione
- registri di modo
- registri di comando
- registri di stato

9.4.7.2 FUNZIONALITA' TIPICA DELLE USART

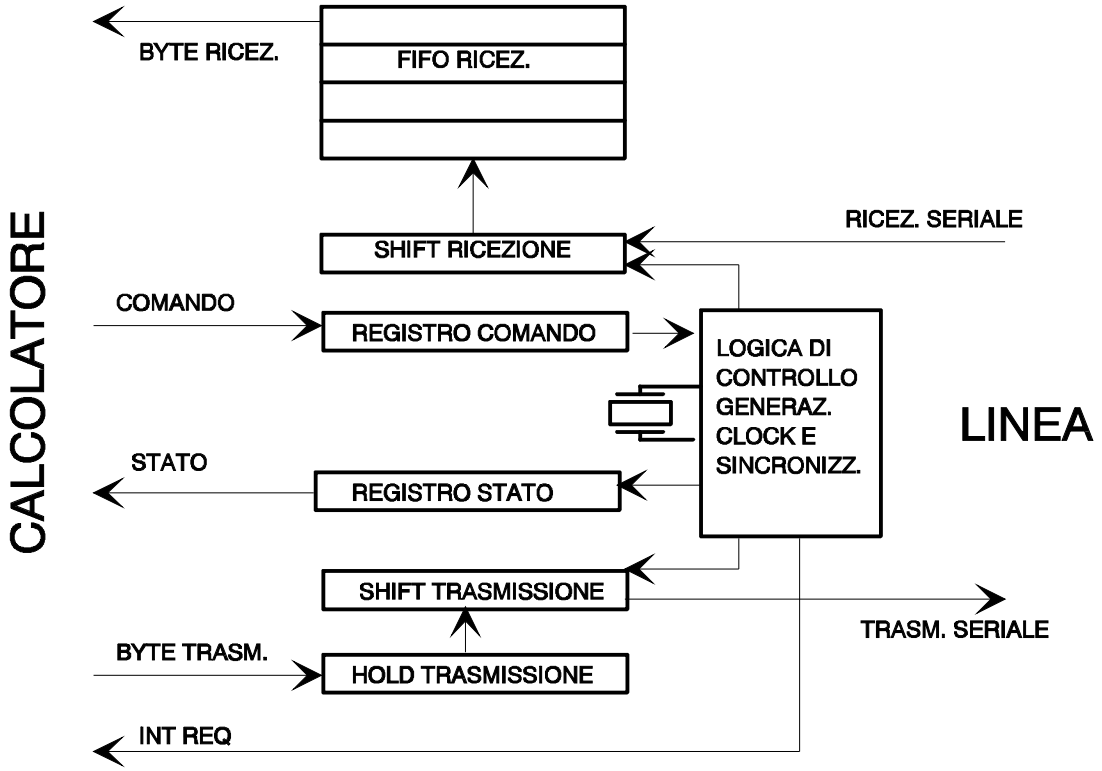


Fig. 9.5 - Schema a blocchi funzionali semplificato di una UART.

Scrivendo opportuni valori nei bit dei registri di modo si predispone il funzionamento asincrono o sincrono, la velocità di ricetrasmissione, la generazione / verifica di parità, la generazione di interrupt, ecc.

Nelle comunicazioni sincrone sono previsti anche registri in cui scrivere i caratteri di sincronismo. Questi caratteri sono confrontati con la sequenza di bit in ricezione e a corrispondenza verificata viene attivato il riconoscimento dei singoli caratteri e posizionato ad “uno” l'indicatore di sincronismo avvenuto (v. sotto registro di stato).

Scrivendo un carattere nel **buffer** (*hold*) **di trasmissione** si attiva il procedimento di trasmissione che porta tale carattere nel registro shift di trasmissione e poi ne emette in sequenza i vari bit, eventualmente preceduti dal bit di start (se trasmissione asincrona) e seguiti dai bit di parità e di stop.

Leggendo il contenuto dei **registri di stato**, dai vari bit si possono ricavare informazioni che sono tipicamente:

- sincronizzazione avvenuta, significativo solo nel caso di comunicazioni sincrone
- buffer di trasmissione vuoto, pronto per il prossimo carattere da trasmettere
- shift di trasmissione vuoto, cioè trasmissione dell'ultimo carattere completata
- buffer di ricezione pieno, contenente un carattere ricevuto
- errore di parità del carattere ricevuto

- errore di *framing* (formato) del carattere ricevuto, spesso indice di errata impostazione della velocità
- errore di *overrun* che indica la situazione di carattere ricevuto ma non estratto in tempo dal buffer di ricezione e quindi perso perchè sovrascritto dal carattere successivo
- richiesta di interrupt, nel caso essa sia stata abilitata

Leggendo il registro **buffer di ricezione** (o dalla FIFO se prevista) si ottiene il successivo carattere ricevuto, se l'indicatore di buffer di ricezione pieno vale ”uno”, e si azzera tale indicatore. Il carattere è da considerarsi errato se sono al valore “uno” uno o più bit indicatori di errori nel registro di stato.
Il registro di **comando** serve in genere per modificare volontariamente lo stato, spesso per azzerare indicatori di cui si è presa visione e tenuto conto, per abilitare o meno la generazione di richieste di interrupt e per gestire eventuali segnali di controllo di linea, come RTS, DTR, ecc..

9.4.7.3 UART NS8250 e NS16550

Presentiamo ora, come esempio concreto, una tipica UART adottata par gestire l’interfaccia seriale standard nei diffusi *Personal Computer*.
Si noti che la UART NS16550 è la versione più recente e potenziata della versione base NS8250 usata nei primi modelli di PC.
L’UART si presenta al programmatore come una serie di registri di 8 bit, che occupano 8 indirizzi consecutivi nell’area di indirizzamento di I/O. Nei PC standard l’indirizzo base (BA = Base Address) per COM1 è generalmente 3F8H con interrupt IRQ 4, e per COM2 è 2F8H con interrupt IRQ 3.
Nel seguito si adotta la normale convenzione di chiamare b7 il bit più significativo (MSB) di un byte e b0 il bit meno significativo (LSB).
Con R si indicano i registri di ingresso a sola lettura (Read), con W i registri in uscita a sola scrittura (Write) e con R/W i registri a lettura e scrittura.
Per ogni registro è riportato inoltre l’indirizzo relativo a BA. Si noti che all’indirizzo BA+0 sono allocati tre registri, uno R uno W e uno R/W. L’accesso è dirottato su uno dei primi due se il bit DLAB=0 (b7 del registro LCR), mentre il terzo registro è accessibile se DLAB=1.

REGISTRI

- **RBR - Receiver Buffer Register - R Ind. = BA - (con DLAB=0)**

Contiene l’ultimo carattere ricevuto. Se il registro non viene letto prima che sia ricevuto il carattere seguente, il carattere originale viene perso (*overrun*).
Nella versione NS16550 è presente una coda FIFO in grado di contenere fino a 16 caratteri ricevuti, prima che si verifichi un *overrun*.

- **THR - Transmitter Holding Register - W Ind = BA - (con DLAB=0)**

Scrivendo un carattere (byte) in questo registro, se ne provoca la trasmissione. Un’eventuale sovrascrittura prima che sia completata la trasmissione provoca la perdita sia del carattere originario che di quello sovrascritto.
Nel NS16550 è presente una coda FIFO in grado di accogliere fino a 16 caratteri in trasmissione.

- **DLR - Divisor Latch Register - R/W Ind = BA, BA+1 - (con DLAB=1)**

Si tratta di due registri destinati a impostare la velocità di trasmissione (*Baud Rate*).

Nel complesso formano una parola di 16 bit di cui la parte meno significativa (*Low Byte*) è a BA+0 e la più significativa (*High Byte*) a BA+1. Nella tabella seguente sono riportati i valori per le varie velocità standard.

Baud Rate	High Byte	Low Byte
110	04H	17H
300	01H	80H
1200	00H	60H
2400	00H	30H
4800	00H	18H
9600	00H	0CH
19200	00H	06H
38400	00H	03H
57600	00H	02H
115200	00H	01H

- **IER - Interrupt Enable Register - R/W Ind = BA+1 (con DLAB=0)**

Questo registro è utilizzato per abilitare la UART a generare diversi tipi di richiesta di interrupt, in ogni combinazione, impostando ad 1 i corrispondenti bit, con i significati seguenti.
b7 . . b4 - Riservati, sempre = 0
b3 - Modem Status Interrupt
b2 - Receiver Line Status Interrupt
b1 - Transmitter Holding Register Empty Interrupt
b0 - Received Data Available Interrupt

- **IIR - Interrupt Identification Register - R Ind = BA+2**

Quando la UART genera una richiesta di interrupt, i bit di questo registro sono impostati in modo da riflettere il tipo di interrupt.
Per ogni condizione viene generata una distinta richiesta di interrupt, con la seguente scala di priorità decrescente.

Receiver Line Status (priorità massima)
Received Data Available
Transmitter Holding Register Empty
Modem Status (priorità minima)

b7, b6 - (solo per NS16550) Entrambi a 1 se la FIFO è abilitata.
b5, b4 - Riservati, sempre a 0.
b3 - (solo NS16550) Interrupt di Timeout. Se la FIFO è abilitata questo bit è posto a 1 insieme al bit b2, per indicare che è pendente un interrupt di time-out.
b2, b1 - Identificazione dell'interrupt, come da elenco seguente, in cui è riportato tra parentesi l'operazione che azzera la richiesta pendente.

b2	**b1**	**Indicazione**	**Azzeramento**
1	1	Receiver Line Status	lettura LSR
1	0	Received Data Available	lettura RBR
0	1	Transm. Hold. Register Empty	lett. IIR scritt. THR
0	0	Modem Status	lettura MSR

b0 - Interrupt pending. Vale 0 se è pendente una richiesta di interrupt, altrimenti vale 1.

- **FCR - FIFO Control Register - W Ind = BA+2**

Registro presente solo in NS16550.
b7, b6 - Livello di trigger della FIFO di ricezione.

b7	**b6**	**n. byte di riempimento per attivare trigger**
0	0	1
0	1	4
1	0	8
1	1	14

b5, b4 - Riservati, sempre a 0.
b3 - Modo di RxRdy/TxRdy. Normalmente a 0.
b2 - Reset della FIFO di trasmissione. Scrivendo 1 in questo bit si azzera la FIFO e il bit poi si azzera automaticamente.
b1 - Reset della FIFO di ricezione. Scrivendo 1 in questo bit si azzera la FIFO e il bit poi si azzera automaticamente.
b0 - FIFO Enable. 1=abilitata, 0=disabilitata

- **LCR - Line Control Register. - R/W Ind = BA+3**

b7 - DLAB Divisor Latch Access Bit. Deve essere posto a 1 per accedere ai registri di Baud Rate (Divisor Latch Register - v. sopra), mentre deve essere a 0 per accedere ai registri RBR, THR e IER.
b6 - Set Break. Quando è posto ad 1 viene generata sulla linea di trasmissione una condizione di “break”, corrispondente al livello continuo di un bit di start.
b5 - Riservato, sempre a 0.
b4, b3 - Tipo di parità
11 = parità pari
01 = parità dispari
00 = nessuna parità
10 = non definito
b2 - Bit di Stop.
0 = 1 bit di stop
1 = 2 bit di stop
b1, b0 - Lunghezza del carattere
00 = 5 bit
01 = 6 bit
10 = 7 bit
11 = 8 bit

- **MCR - Modem Control Register - R/W Ind = BA+4**

I bit di questo registro comandano segnali in uscita dalla UART.
b7 . . b3 - Riservati, sempre a 0.
b2 - Abilitazione globale di interrupt della UART. 1 = abilitato.
b1 - Segnale RTS (Request To Send) 1 = richiesta di trasmettere, 0 = ricezione.
b0 - Segnale DTR (Data Terminal Ready) 1 = ready.

- **LSR - Line Status Register - R Ind = BA+5**

Questo registro contiene informazioni sullo stato del trasferimento di dati. Quando si verifica un *Line Status Interrupt*, leggendo questo registro si rimuove l'interrupt e osservando i bit b4 . . b1 se ne individua la causa (condizioni di errore).
Notare che quando si funziona con la FIFO abilitata i valori di questi 4 bit rispecchiano errori relativi al carattere corrente (in testa alla FIFO).
b7, b6 - Don't Care = non significativi.
b5 Transmitter Holding Register Empty. Questo bit vale 1 se la UART è in grado di accettare un carattere da trasmettere (nel NS16550 FIFO vuota). Se abilitato viene generato anche un interrupt THRE.
b4 - Break. Vale 1 se è stata rilevata sulla linea una condizione di break, cioè un valore basso (come bit di start) di durata superiore ad un intero carattere.
b3 - Framing Error. Vale 1 se al termine del carattere non è stato rilevato un corretto bit di stop. Probabile causa è la differenza di baud-rate rispetto all'altro sistema collegato alla linea seriale.
b2 - Parity Error. Vale 1 se il carattere ricevuto non aveva il bit di parità corretto.
b1 - Overrun Error. Vale 1 se il carattere in RBR non è stato letto prima di essere sovrascritto da un successivo carattere ricevuto.
b0 - Data Ready. Vale 1 se è disponibile in RBR un carattere ricevuto e non ancora letto. Il bit va a 0 automaticamente con la lettura di RBR.

- **MSR - Modem Status Register - R Ind = BA+6**

I bit b7 . . b4 rispecchiano lo stato di segnali del modem, mentre i bit b3 . . b0 sono posti ad 1 quando sono rilevati fronti di commutazione di tali segnali (con eventuale generazione di un Modem Status Interrupt) e automaticamente posti a 0 con la lettura di questo registro MSR.
b7 - DCD - Data Carrier Detect
b6 - RI - Ring Indication.
b5 - DSR - Data Set Ready.
b4 - CTS - Clear To Send
b3 - DCD commutato
b2 - RI commutato.
b1 - DSR commutato.
b0 - CTS commutato

- **SR - Scratch Register - R/W Ind = BA+7**

Questo registro non ha nessun effetto, ma può essere usato come una cella di memoria per memorizzare informazioni relative alla gestione del collegamento seriale.

9.4.7.4 GESTIONE SOFTWARE DELLE USART

- - GESTIONE A CONTROLLO DI PROGRAMMA

Consiste in una periodica lettura del registro di stato LSR dai cui bit si ricavano informazioni sulla disponibilità di un carattere ricevuto, sul verificarsi di eventuali errori e sulla possibilità di trasmettere il prossimo carattere.

Quando si verificano le situazioni rilevanti si procede al trattamento corrispondente.

Questa tecnica, particolarmente semplice nella sua forma più elementare, si presta soprattutto nei casi in cui non si abbiano altre attività concomitanti da svolgere durante le comunicazioni.

- - GESTIONE AD INTERRUZIONE

In questo caso viene affidata alla USART la possibilità di generare una richiesta di interruzione (interrupt) al processore quando si verificano gli eventi rilevanti della comunicazione, cioè ricezione di un carattere, rilievo di errori e trasmissione di un carattere.

Il programma che nel frattempo sta svolgendo altre eventuali elaborazioni viene temporaneamente interrotto per dar luogo all'esecuzione delle apposite routine di servizio delle interruzioni che, dopo aver rilevato dal registro IIR la causa trattano gli eventi indicati. Infine si torna a riprendere le elaborazioni interrotte.

La tecnica dell'interrupt è la più utilizzata nelle applicazioni industriali se la bassa o media velocità di trasmissione lo consente (fino a circa 1000 byte /s cioè 9600 bit/s). Con le UART dotate di FIFO si possono sostenere velocità anche decisamente superiori.

- - GESTIONE A DMA

Il "*Direct Memory Access*" costituisce una tecnica più costosa delle precedenti, ma necessaria per le comunicazioni alle velocità piu' elevate.
Si utilizzano appositi circuiti (controllori di DMA oppure coprocessori di comunicazione) che sono in grado di effettuare direttamente i trasferimenti di informazioni dalla memoria operativa del processore principale alla USART e viceversa, ad elevate velocità (Mbyte/s) e richiedono l'intervento del processore stesso solo per le elaborazioni di fine messaggio o il trattamento degli errori.
Per questo tipo di gestione è necessario che la UART sia dotata di circuiti di richiesta al controllore di DMA, cosa che è prevista solo in particolari interfacce.

9.5 ESEMPI DI GESTIONE DELLA COMUNICAZIONE

9.5.1 Modello della USART

```
;*********************************************************
; Indirizzi dei registri e principali maschere dei bit
;*********************************************************

BA    EQU   3F8H        ;indirizzo base

RBR   EQU   BA+0        ;dato ricevuto
THR   EQU   BA+0        ;dato da trasmettere

DLRL  EQU   BA+0        ;seleziona baud L
DLRH  EQU   BA+0        ;seleziona baud H
B1200 EQU   0060H       ;valore per 1200 baud
B9600 EQU   000CH       ;valore per 9600 baud

IER   EQU   BA+1        ;reg. abilitaz. interrupt
MKEIT EQU   2           ;maschera int. trasmissione
MKEIR EQU   1           ;maschera int. ricezione

IIR   EQU   BA+2        ;reg. tipo interrupt
MKI   EQU   6           ;maschera bit di tipo

FCR   EQU   BA+2        ;reg. controllo FIFO
MKF1  EQU   0           ;int. ogni byte
MKF4  EQU   04H         ;int. ogni  4 byte
MKF8  EQU   08H         ;           8 byte
MKF14 EQU   0CH         ;           14 byte
MKFRT EQU   04H         ;mschera reset FIFO trasm.
MKFRR EQU   02H         ;maschera reset FIFO ricez.
MKFEN EQU   01H         ;maschera abilita FIFO

LCR   EQU   BA+3        ;reg. controllo linea
MKDLAB      EQU   80H
STDLINE     EQU   1BH   ;impostaz. standard pari, 1 stop, 8 bit

MCR   EQU   BA+4

LSR   EQU   BA+5        ;reg. stato linea
MKTHRE      EQU   20H   ;maschera THRE
MKBRK       EQU   10H   ;maschera Break
MKFRE       EQU   08H   ;maschera framing err.
MKPAE       EQU   04H   ;maschera parity err.
MKOVE       EQU   02H   ;maschera overrun err.
MKDRDY      EQU   01H   ;maschera dato ricevuto ready

MSR   EQU   BA+6
```

9.5.2 Inizializzazione della UART

```
;imposta velocita' di comunicazione
      MOV   AL, MKDLAB
      MOV   DX, LCR
      OUT   [DX],AL
      MOV   AX, B9600
      MOV   DX, DLRL
      OUT   [DX],AL
      INC   DX
      OUT   [DX],AH
;imposta modalita' di comunicazione
```

```
      MOV   AL, STDLINE
      MOV   DX,LCR
      OUT   [DX],AL
;cancella eventuali interrupt pendenti
      MOV   DX, IIR
      IN    AL,[DX]
;inizializza la FIFO e la abilita
      MOV   AL, MKFRT+MKFRR+MKFEN
      MOV   DX, FCR
      OUT   [DX],AL
;aggancia la routine di servizio interrupt
      CALL  AGGANCIA_INT_UART
;abilita UART a generare richieste di interrupt
      MOV   AL, MKEIT+MKEIR
      MOV   DX, IER
      OUT   [DX],AL
```

9.5.3 Trasmissione di un carattere a controllo di programma

```
      mov   dx, LSR           ;indirizzo porta registro di stato
wait_tr:
      in    al,[dx]           ;legge parola di stato
      and   al,MKTHRE         ;test se buffer trasmissione libero
      jz    wait_tr
      mov   dx,THR            ;indirizzo porta trasmissione dati
      mov   al,[carattere]    ;carattere da trasmettere
      out   [dx],al
```

9.5.4 Ricezione di un carattere a controllo di programma

Il polling del registro di stato per rilevare la ricezione di un carattere deve essere effettuato con periodo minore del tempo di trasmissione di un carattere.
che a 9600 Baud deve essere effettuato ogni 1 ms. In realtà se si ha una coda in ricezione (FIFO) di N posizioni, si può adottare un periodo di polling fino a N*P, avendo però l'accortezza di estrarre dalla FIFO tutti i caratteri prima di passare ad altre elaborazioni.

```
      mov   dx, LSR           ;indirizzo porta registro di stato
wait_rc:
      in    al,[dx]           ;legge parola di stato
      mov   ah,al             ;salva parola di stato
      and   al,MKDRDY         ;test se dato ricevuto
      jz    wait_rc
      mov   dx,RBR            ;indirizzo porta ricezione dati
      in    al,[dx]           ;legge carattere ricevuto
      and   ah, MKBRK+MKFRE+MKPAE+MKOVE
      jnz   err_ricez         ;gestione errore ricezione
      mov   [carattere],al          ;deposita carattere letto
```

9.5.5 FORMATO DEI MESSAGGI

I messaggi sono in genere strutturati come un contenitore (*envelope*) e un contenuto (*text*).
Il contenitore prevede un'intestazione (*header*) che contiene informazioni di servizio gestite dal protocollo di colloquio, e una conclusione (*trailer*) che normalmente si limita a uno o due caratteri di controllo della correttezza (*checksum* o CRC).
Quando i protocolli sono organizzati a livelli, come avviene generalmente per le comunicazioni su reti, ciò si rispecchia anche nella struttura dei messaggi che si presenta con un annidamento di contenitori. (fig. xx)

Un possibile formato generale di messaggio è descritto nel seguito. Naturalmente alcuni campi potranno essere superflui in alcune applicazioni e per semplicità ed efficienza non usati.

Caratteri di inizio. Potranno essere uno o più caratteri. Dal numero di tali caratteri dipende la massima latenza di riconoscimento del messaggio che non ne provoca la perdita.

Codice destinatario. Usato nei collegamenti multi-punto (*multidrop*). Sono usati anche codici collettivi per i messaggi da trasmettere a gruppi (*multicast*) o a tutti (*broadcast*) i calcolatori collegati allo stesso mezzo trasmissivo.

Numero caratteri totali del messaggio. Indispensabile per riconoscere la fine dei messaggi con testo composto di byte in binario e non in codice ASCII.

Codice mittente. Usato solo nei collegamenti multi-punto.

Progressivo di trasmissione al destinatario. Numero incrementato modulo 256 che consente al ricevente di verificare la sequenza dei messaggi, evitando che passino inosservate perdite o che vengano fatte duplicazioni.

Valore del progressivo dell'ultimo messaggio correttamente ricevuto dal mittente, da cui il partner deduce l'eventuale necessità di ripetere messaggi.

Codice funzione. Indica univocamente quale è il formato della porzione dati del messaggio. Non usato se tutti i messaggi contengono le stesse informazioni.

Parte DATI del messaggio. In alcune applicazioni si decide di adottare solo caratteri in codice ASCII, mentre talvolta si adotta la rappresentazione binaria (a byte) che è più compatta per i valori numerici (2 byte binari invece che 6 caratteri, per un normale numero intero a 16 bit).

Codice di verifica. Si usa un checksum di un byte o vari tipi di CRC (Cyclic Redundancy Check) su 2 byte.

9.5.6 Ricezione di un messaggio

Il problema tipico è costituito dal fatto che non si sa quando arriva il messaggio, ma quando comincia la ricezione si pone un problema di tempo reale stretto per la ricezione di tutti i singoli caratteri. La successiva interpretazione ed elaborazione del messaggio può anche avere requisiti di tempo più laschi.
Per questo motivo generalmente si fa in modo che almeno il primo carattere di un messaggio generi un interrupt, per evitare un continuo polling del registro di stato.

Attesa inizio messaggio
Ricezione del messaggio
Verifica di correttezza
Passaggio del messaggio completo.

Ricezione del messaggio
Viene effettuata a polling o a interrupt. In questo secondo caso la risposta ad interrupt è generalmente strutturata come automa a stati che evolve in base al susseguirsi delle varie parti del messaggio. Si verifica la correttezza dei singoli caratteri ricevuti, e si aggiorna il calcolo del checksum per la verifica a fine messaggio.
Verifica di correttezza
Quando il messaggio termina si deduce la sua correttezza dal confronto delchecksum ricevuto con quello calcolato.
Passaggio del messaggio completo.

Se il messaggio è corretto viene passato al processo di elaborazione e di risposta.

9.5.7 Trasmissione di un messaggio

Con la gestione a controllo di programma non ci sono particolari problemi, se la trasmissione è di tipo asincrono, se non quello di evitare una pausa troppo lunga (oltre il valore di t-out) tra un carattere e l'altro. Con la gestione ad interrupt il primo carattere va scritto direttamente dal processo applicativo sul registro di trasmissione (THR), in modo da mettere in moto il meccanismo di interrupt, che poi richiederà i successivi caratteri del messaggio.
Come per la ricezione anche la risposta ad interrupt di trasmissione è generalmente organizzata con una struttura ad automa a stati.
Attenzione: quando arriva l'interrupt dopo aver emesso sulla USART l'ultimo carattere, questo carattere è ancora in corso di trasmissione nel registro *shift*, e quindi si deve aspettare prima di disabilitare la trasmissione.
La sequenza di operazioni con cui un processo applicativo lancia la trasmissione di un messaggio con gestione ad interrupt è sinteticamente la seguente.

inizializza puntatori e contatori al messaggio
inizializza checksum
abilita interrupt trasmissione
imposta stato S1 all'automa
scrive primo carattere
imposta indicatore trasmissione in corso
WAIT (fine_trasm, tout)

9.6 ESEMPIO DI PROTOCOLLO PUNTO-PUNTO

9.6.1 PROBLEMA

9.6.1.1 Livello applicativo

Collegamento punto-punto con comunicazione half-duplex.
I processi applicativi vedono messaggi col formato seguente:

`tipo`	byte di codice tipo messaggio
`ncar`	numero byte di testo (0 < `ncar` < 256)
`testo`	sequenza di byte di informazione utile

I processi applicativi vedono le funzioni seguenti:

`void ini_com (void)`	inizializza la comunicazione
`void put_tra (mess)`	inserisce messaggio in coda di trasmissione
`mess get_ric (void)`	estrae messaggio da coda di ricezione. Se il sistema remoto non aveva nulla da trasmettere si riceve un messaggio di tipo = 0 e contenente un solo byte = 0 di testo.

E' compito dei processi applicativi riconoscere l'anomalia di "mancato collegamento" deducendola dall'eccessivo riempimento della coda di trasmissione o dal trascorrere di un tempo eccessivo senza ricezione di messaggi corretti, dopo aver eventualmente forzato per un certo numero di volte una trasmissione.

9.6.1.2 Livello comunicazione

Si vuole effettuare la ritrasmissione dei messaggi non ricevuti correttamente.

Formato dei messaggi

`SOH`	carattere iniziale *Start Of Header*
`prog_t`	codice progressivo del messaggio corrente
`prog_r`	codice progressivo dell'ultimo messaggio ricevuto correttamente
`tipo`	codice tipo messaggio
`ncar`	numero byte del testo
`testo`	contenuto del messaggio
`cks`	checksum di tutti i byte da SOH escluso

Ogni carattere è di 8 bit + parità.
Tra un carattere trasmesso ed il successivo non si supera la distanza temporale di t_out ms, in modo da consentire il riconoscimento di fine messaggio anche in caso di errori.

Strutture dati.

`coda_ric` coda di messaggi correttamente ricevuti in attesa di essere consumati dai processi applicativi.

`coda_tra` coda di messaggi prodotti dai processi applicativi e da trasmettere.

`buf_ric` array di caratteri per la ricezione del messaggio corrente.

`buf_tra` array di caratteri per la trasmissione del messaggio corrente.

Funzioni di comunicazione

`mess get_tra (void)` estrae messaggio dalla coda di trasmissione. Se la coda è vuota si invia un messaggio di tipo vuoto (=0) con un solo byte (=0).

`void put_ric (mess)` inserisce messaggio nella coda di ricezione

`interrupt int_ric ()` funzione chiamata da interrupt di ricezione, cioè quando si è ricevuto un carattere o si ha un errore in ricezione. Organizzata come automa a stati che evolve con l'andamento del messaggio in corso di ricezione.

`interrupt int_tra ()` funzione chiamata da interrupt di trasmissione, cioè quando si ha *Transmit Register Empty*. Organizzata come automa a stati che evolve con l'andamento del messaggio in corso di ricezione.

`int leggi_tempo (void)` lettura tempo corrente, usata per il rilievo di time-out di fine messaggio

Variabili statiche visibili dalle funzioni di comunicazione

`char`

`stato_t`	stato dell'automa di trasmissione
`stato_r`	stato dell'automa di ricezione
`prog_t_rem`	codice progressivo di trasmissione del sistema remoto
`prog_t_loc`	codice progressivo di trasmissione del sistema locale
`prog_r_rem`	codice progressivo di ricezione del sistema remoto
`prog_r_loc`	codice progressivo di ricezione del sistema locale
`cks_t`	checksum corrente del messaggio in trasmissione
`cks_r`	checksum corrente del messaggio in ricezione
`count`	numero caratteri del testo da ricevere

```
int
     it              indice corrente del buffer di trasmissione
     ir              indice corrente del buffer di ricezione
long int
     tempo_prec          tempo di ricez. carattere precedente (per t-out)
     tempo_corr          tempo di ricez. carattere attuale (per t-out)
```

9.6.2 IMPLEMENTAZIONE

INIZIALIZZAZIONE

Inizializza code di messaggi di ricezione e trasmissione
Inizializza le variabili statiche usate nelle risposte agli interrupt
Aggancia le risposte ad interrupt ai vettori corrispondenti
Inizializza i registri della UART

Si propone una forma semplificata di risposta ad interrupt per carattere ricevuto, eventualmente con errori, ed in seguito una risposta ad interrupt per la trasmissione di un carattere
Si suppone che sia prevista a monte la parte di risposta effettivamente agganciata all'interrupt della UART, che si fa carico delle operazioni iniziali (salvataggio registri, ecc.) ed in particolare dell'individuazione se l'interrupt riguarda la ricezione o la trasmissione, in base ai bit del registro di stato IIR.

SERVIZIO INT RICEZIONE

```
void leggi_byte_uart (void)
  {
  byte = leggi_rbr (); // registro ricezione UART
  errori = check_lsr ();     // line status per verifica errori
  if (errori)
      {tempo_prec = leggi_tempo (); // tempo arrivo byte
      stato_r = S_ER;}
  switch (stato_r)
  {case S0:
      if (byte == SOH)
            stato_r = S1;
      else
            {tempo_prec = leggi_tempo (); // t arrivo byte
            stato_r = S_ER;}
      break;
  case S1: // aspetta progressivo trasmesso remoto
      cks_r = 0;  // azzera checksum
      ir = 0;           // indice buffer a inizio
      prog_t_r = byte;
      cks_r += byte;
      stato_r = S2;
      break;

  case S2: // aspetta progressivo ricevuto remoto
      prog_r_r = byte;
      cks_r += byte;
      stato_r = S3;
      break;

  case S3: // aspetta codice tipo
      buf_ric [ir] = byte;
      ir++;
      cks_r += byte;
      stato_r = S4;
```

```
      break;

   case S4:  // aspetta numero byte testo (>=1)
      buf_ric [ir] = byte;
      ir++;
      count = byte;
      cks_r += byte;
      stato_r = S5;
      break;

   case S5:  // aspetta i byte del testo
      buf_ric [ir] = byte;
      ir++;
      cks_r += byte;
      if (--count == 0)
            stato_r = S6;
      break;

   case S6:  // aspetta byte di checksum
      if (byte == cks_r)      // verifica checksum
            {put_ric (buf_ric);     // nuovo messaggio corretto
            prog_r_l = prog_t_r;
            disab_ric ();
            enab_tra ();            // abilita trasmissione
            stato_r = S0;}
      else                    // errore checksum
            {tempo_prec = leggi_tempo (); // t arrivo byte
            stato_r = S_ER;}

      break;

   /* ERRORE - Lascia passare, ignorandoli, tutti gli eventuali
   caratteri rimanenti e aspetta l'inizio del messaggio successivo
   riconosciuto dal carattere SOH ricevuto dopo lo scadere di time-out
   */
   case S_ER:      // ERRORE -
      tempo_corr = leggi_tempo ();  // tempo arrivo byte
      if (((tempo_corr-tempo_prec)>=TOUT) && (byte==SOH))
            stato_r = S1;
      else
            tempo_prec = tempo_corr;
      break;
   }
```

SERVIZIO INT TRASMISSIONE

```
void scrivi_byte_uart (void)
   {
   switch (stato_t)
   case S0:
      {/* vede se l'ultimo messaggio trasmesso e' stato
       ricevuto correttamente da sistema remoto */
      if (prog_r_r == prog_t_l)
            prog_t_l++;
            buf_tra = get_tra ();   // nuovo messaggio
            if (buf_tra == NULL)    // nulla da trasmettere
                  {buf_tra[0] = 0;  // tipo
                  buf_tra[1] = 1;   // num. byte
                  buf_tra[2] = 0; } // byte testo
      byte = SOH;
      scrivi_thr (byte);
      stato_t = S1;
      break;

   case S1:  // trasmette progressivo trasmesso locale
      cks_t = 0;  // azzera checksum
      it = 0;           // indice buffer a inizio
      byte = prog_t_l;
```

```
    scrivi_thr (byte);
    cks_t += byte;
    stato_t = S2;
    break;

case S2: // trasmette progressivo ricevuto locale
    byte = prog_r_l;
    scrivi_thr (byte);
    cks_t += byte;
    stato_t = S3;
    break;

case S3: // scrive codice tipo
    byte = buf_tra [it];
    scrivi_thr (byte);
    it++;
    cks_t += byte;
    stato_t = S4;
    break;

case S4: // trasmette numero byte testo (>=1)
    byte = buf_tra [it];
    scrivi_thr (byte);
    it++;
    count = byte;
    cks_t += byte;
    stato_t = S5;
    break;

case S5: // trasmette i byte del testo
    byte = buf_tra [it];
    scrivi_thr (byte);
    it++;
    cks_t += byte;
    if (--count == 0)
          stato_t = S6;
    break;

case S6: // trasmette byte di checksum
    byte = cks_t;
    scrivi_thr (byte);
    stato_t = S7;
    break;

case S7: // termine trasmissione
    disab_tra ();
    enab_ric ();
    stato_t = S0;
    break;
}
```

9.6.2.1 Struttura del processo Master

Nel seguito viene presentata una trama sintetica di come potrebbe essere impostata una comunicazione di tipo *master-slave* con un processo che gira a bordo del calcolatore *master* e uno che gira a bordo del calcolatore *slave*.

```
/*********************************************
Processo di comunicazione MASTER
*********************************************/
void comm_proc(void)
{
    ini_comm(); //inizializza protocollo
    for (;;)
    {
      WAIT_TIMER (PERIODO);   // scadenza prossimo periodo
      prepara_mess();
      trasm();
      esito = WAIT (RISP, TIME);    // attesa risposta
      if (esito == T_OUT)
        manca_comm();  // lo SLAVE non ha risposto
      else
      {
        tratta_ric();  // gestione messaggio ricevuto
      }
    }
}

/*********************************************
 Inizializzazione del colloquio
*********************************************/
void ini_comm (void)
{
    /* varie inizializzazioni, come precedentemente accennato, di code,
variabili statiche, aggancio interrupt al vettore, UART, ecc. */
}

/*********************************************
Prepara messaggio da trasmettere
*********************************************/
void prepara_mess (void)
{
    /* inserisce in un'area di memoria, vista come array di caratteri,
    le informazioni da trasmettere, precedute dal codice di tipo del
    messaggio e dal numero di byte di testo effettivo. */
}

/*********************************************
Comando di trasmissione del messaggio
*********************************************/
int trasm (void)
{
    /* Abilita la trasmissione, ed in particolare la relativa
    generazione di richieste di interrupt, e scrive a controllo di
    programma il primo carattere sul registro THR della UART. I
    successivi saranno trasmessi a carico della routine ISR di risposta
    ad interrupt di trasmissione. */
}
/*********************************************
Gestisce informazioni messaggio ricevuto
*********************************************/
tratta_ric ()
{
    /* Dal codice di tipo del messaggio riconosce di quali informazioni
    si tratta, e di conseguenza le ricopia ed elabora opportunamente.
    Eventualmente si fa carico di gestire il protocollo a livello
```

```
    applicazione, decidendo quindi il tipo di messaggio con cui
    rispondere al messaggio appena ricevuto. */
}

/*********************************************
Segnala allarme di mancato collegamento
*********************************************/
manca_comm ()
{
    /* Modifica il valore di una variabile di stato che indica la
    funzionalita' o meno del collegamento. Eventualmente genera un
    evento di anomalia che produce una segnalazione per l'operatore. */
}
```

9.7 ALTRE LETTURE

J. Butler
UARTs make possible low-cost networks of embedded systems.
EDN March 30, 1995 pag. 87

S. Rothkin
PC UART Device Driver
The C User Journal - Dec 1991 pag.62

C. Bin
La Trasmissione Dati.
Ed. Buffetti Editore 1991

Tecniche per la trasmissione di dati, modem, reti e riferimenti a standard per reti pubbliche, con attenzione prevalente alle applicazioni gestionali.

G. Saccardi
Trasmissione dati - Dispositivi standard e protocolli.
Ed. Jackson 1986

Contiene molte informazioni di dettaglio su standard e raccomandazioni nazionali ed internazionali, con particolare attenzione alle reti telefoniche ed alle applicazioni gestionali e a grandi distanze.

R. L. Krutz
Interfacciamento nella progettazione di sistemi digitali.
Ed. Jackson 1988

Sviluppa diversi aspetti dei sistemi digitali, con particolare attenzione alle interconnessione tra i circuiti (integrati) interni, ma con interessanti presentazioni di tecniche di collegamento.

Tratta sommariamente anche aspetti software e sistemi basati su processori della famiglia 80x86.

Ricca bibliografia nei vari capitoli.

A.S. Tanenbaum
Computer networks.
Ed. Prentice-Hall 1988

Ottima presentazione, chiara e completa, delle problematiche delle reti di calcolatori, con analisi delle funzionalita' ai vari livelli ISO-OSI e riferimenti agli standard per reti locali e geografiche.

U.D. Black
Data networks: concepts, theory and practice.
Ed. Prentice-Hall 1989
H.C. Folts
Data Communications Standards
McGraw-Hill 1982

Corposa raccolta di standard elettrici e funzionali per le comunicazioni seriali.