PROTEUS ES

Manuale di programmazione

Volume 1

Programmazione mediante interfaccia grafica
Programmazione in linguaggio C

L' interfaccia grafica

L'interfaccia di Proteus dispone apparentemente di pochi componenti di base, ma grazie alla possibilitá di creare e di utilizzare oggetti custom, e grazie ai metodi e alle proprietà dei componenti esistenti, studiati per la massima flessibilità, è possibile con tale strumento creare Forms e Applicazioni di qualsiasi genere e grado di complessità.

1- Oggetti e componenti

L'interfaccia grafica di Proteus permette di disporre sui FORM del progetto (le pagine visualizzate) vari oggetti, a cui sono associate delle proprietá e degli eventi.
Proteus assegna automaticamente un nome di default per gli oggetti creati. Possiamo cambiare questo nome, cambiando la proprietá nametype. Il nome di un oggetto deve essere una stringa alfanumerica di lettere e numeri non contenente spazi e deve iniziare con una lettera minuscola. Nel nome è consentito usare il carattere speciale "_" (sottolineatura).

Un oggetto fa parte del form in cui è definito.
Le proprietá di un oggetto fanno parte dell' oggetto stesso.
Ad esempio:
Se in pagina page1 abbiamo l'oggetto label1:
-label1 è conosciuta dal programma come page1->label1
-la sua proprietá caption è `page1->label1->caption`.

Per favorire la portabilitá del codice, ovvero per scrivere delle routines che possono essere utilizzate anche su oggetti diversi o pagine diverse, possiamo utilizzare i seguenti nomi di **default:**
All'interno della pagina (ad esempio negli eventi dell'oggetto stesso o di altri oggetti della pagina), la pagina stessa puo' essere chiamata come **Local.**
All'interno degli eventi dell' oggetto, l'oggetto stesso puó essere identificato come **me** o **This**.

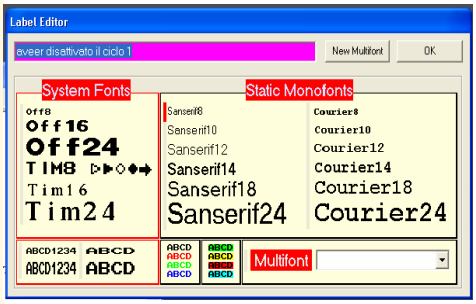
Ad esempio:
Nel codice associato agli eventi degli oggetti di page1,
`page1->label1->caption` equivale a `Local->label1->caption`
Nel codice associato agli eventi di label1,
`page1->label1->caption` equivale a `This->caption`
oppure a `me->captino`

Gli oggetti di base predefiniti

Label:

permette di visualizzare una scritta sul form. La scritta visualizzata è la proprietá caption dell' oggetto.

Le proprietá **font**, **fontcolor** e **color** permettono di cambiare rispettivamente il font, il colore ed il colore del fondo della scritta.



Con un doppio click su una label si apre una finestra in cui è possibile cambiare le principali proprietá (font, color, fontcolor, caption) in modo semplice ed intuitivo.

La proprietá Caption merita un discorso un po' piú approfondito.

Se Caption contiene un semplice testo, questo testo verrà visualizzato, oppure tradotto nella lingua corrente nel caso in cui sia abilitata l' opzione multilingua (vedi opzione multilingua).

In alternativa Caption puó contenere il nome di un puntatore.
Ad esempio:

caption	*pippo
caption	msg[page+7]

In tal caso la scritta sará assegnata dinamicamente.

Per cambiare la caption da programma, possiamo utilizzare piú modi:

```
1- page1->label1->caption="abcd";
2- sprintf(page1->label1->caption,"prova%d",varx);
3- strcpy(page1->label1->caption,page3->label4->caption);
```

Nel primo esempio cambiamo il puntatore, mentre nei seguenti riscriviamo il contenuto della stringa. In questo secondo caso dobbiamo far ben attenzione a non scrivere una stringa piú lunga di quella originale (quella definita nella proprietá caption) altrimenti non sappiamo cosa andiamo a sovrascrivere in memoria.

Edit:

Permette di visualizzare il valore di una variabile associata (proprietá **var**). La proprietà **caption** permette di aggiungere una eventuale scritta a destra del campo stesso.

A differenza dell' oggetto Label, un campo di Edit permette di visualizzare variabili alfanumeriche (stringhe, ovvero char *) e numeriche, specificando il numero di caratteri (**digits**), la posizione di un eventuale punto decimale (**decimals**), lo stile della cornice (**bevel**, **bordercolor**, **borderwidth**).

Se la proprietá **enabled** vale 1, l' operatore puó cambiare il valore della variabile associata da tastiera o con il mouse o il touch-screen.
In ogni caso, il valore visualizzato mostra sempre il valore attuale della variabile.
Se vogliamo aumentare il tempo di aggiornamento della variabile (utile per variabili che cambiano molto rapidamente), possiamo immettere nella proprietá **frequency** il tempo minimo (in millisecondi) di aggiornamento del campo.

Le proprietá **rows Y** e **columns X** permettono di definire con un solo oggetto un vettore o una matrice di campi di edit.
In tal caso è possibile assegnare alla proprietá **var** (variabile associata) un vettore o una matrice, usando l' indicizzazione intrinseca **[ ].**

Ad esempio se per edit1 definiamo:

var	pippo[]
Rows Y	4
Columns X	1

Avremo 4 campi di edit allineati verticalmente, il primo associato a pippo[0], il secondo a pippo[1] ecc....

var	caio[][]
Rows Y	4
Columns X	2

Avremo una matrice di due colonne e 4 righe di campi di edit, associati a caio[0][0],caio[0][1] ecc....

Anche per l' oggetto Edit, per quanto riguarda la caption vale quanto detto per le label, ed in caso di matrici di campi di edit, potremo utilizzare per la caption, matrici intrinseche di stringhe.

ad esempio:

```
char  scritta[][]={"line1","line2","line3","line4"};
```

caption	scritta[]
Rows Y	4
Columns X	1

Button:

è il classico bottone di Windows.
La variabile associata (proprietá **var**), di tipo byte, varrà 1 quando il bottone è premuto, e 0 quando il bottone non è premuto. Le proprietá **caption** e **font** consentono di gestire la scritta sul bottone.

Se vogliamo una pulsantiera con piú bottoni mutuamente esclusivi,

- tutti i bottoni devono avere la **stessa variabile** (di tipo byte, ovvero char).
- i bottoni devono avere la proprietá **refresh=1**
- i bottoni devono avere le proprietá **tag** diverse.

In tal modo, premendo un bottone, la variabile assume il valore della corrispondente proprietá tag, e l' unico bottone che risulterà premuto è quello con la propria proprietá tag uguale al valore della variabile.

Per avere piú pulsantiere indipendenti, basta usare variabili var diverse.

Con un doppio click sul bottone, come per la label, si apre una finestra che consente di impostare in modo intuitivo lo stile ed il colore della scritta.

La scritta è di norma centrata, ma per alcuni font non monospaced può risultare un po' decentrata. In tal caso è possibile allinearla inserendo all' inizio o alla fine della stessa dei caratteri blank.

Per una corretta animazione, la proprietá bevel deve essere settata rised2 (default) o rised1.

Checkbox:

Equivale al checkbox e al radiobutton di Windows.

La variabile associata (proprietá **var**), di tipo byte, varrà 1 quando il checkbox è attivo, e 0 quando è disattivato. Le proprietá **caption** e **font** consentono di gestire la scritta a destra del checkbox stesso.

Se vogliamo una serie di checkbox mutuamente esclusivi (radiobuttons),

- tutti i checkbox devono avere la **stessa variabile** (di tipo byte, ovvero char).
- i checkbox devono avere la proprietá **refresh=1**
- i checkbox devono avere le proprietá **tag** diverse.

In tal modo, premendo un checkbox , la variabile assume il valore della corrispondente proprietá tag, e l' unico checkbox che risulterà attivo è quello con la propria proprietá tag uguale al valore della variabile.
Per avere più serie di checkbox indipendenti, basta usare variabili (proprietà var) diverse.

Panel:

Può essere usato come semplice elemento decorativo, per dividere la pagina con cornici rialzate, ribassate o contornate (proprietá **bevel**), oppure come contenitore attivo. Infatti, oltre alla pura funzione decorativa, un Panel ha le seguenti funzioni operative:

- Gli oggetti che vengono creati all' interno di un Panel, fanno parte del pannello stesso, e quindi spostando il pannello, sposteremo anche gli oggetti che si trovano su di esso.
- Se un pannello cambia il proprio stato da visibile a invisibile (proprietá **visible**), anche tutti gli oggetti in esso contenuto saranno invisibili.

Per un pannello, le proprietá caption, var, refreshvartype, glyph, numglyph, frequency, sebbene presenti, non hanno alcun senso in sé .
La proprietá **enabled** rende sensibile il pannello al touch-screen ed al mouse se vale 1, attivando gli eventi on_click, on_mousedown, on_mousepress e on_mouseup.
La proprietá **canfocus**, se ad 1, rende il pannello selezionabile da tastiera con i tasti freccia-destra e freccia-sinistra, ed abilita eventuali campi di Edit in esso contenuto alla selezione mediante i tasti freccia-su e freccia-giú.
È cosí possibile, per apparecchiature con sola tastiera, un accesso riga-colonna ai campi di edit, in cui il pannello rappresenta la colonna.

Anche per gli oggetti Panel, come per i campi di edit, è possibile settare le proprietá **rows Y** e **column X** per creare vettori o matrici di pannelli.

Bitmap:

È' l' oggetto che consente di visualizzare disegni, anche animati.
La proprietá **bitmap** è il nome del disegno da visualizzare. Tale proprietá si assegna in Proteus, scegliendo la figura (di tipo BMP) e regolandone la tonalitá, le dimensioni e la trasparenza con l' apposito tool presente su Proteus e che viene chiamato automaticamente con un doppio click del mouse sul' oggetto bitmap.

Se il disegno è animato, ovvero se consiste di piú figure che possono cambiare, è necessario partire da un' immagine di tipo BMP in cui siano presenti, affiancati orizzontalmente e **tutti della stessa larghezza in pixels**, tutti i disegni alternativi.
A questo punto metteremo nella proprietá **numglyphs** il numero di sottodisegni e nella proprietá glyph quello che vogliamo vedere.
Durante l' esecuzione del programma il disegno visualizzato sará quello individuato dal valore della variabile associata **var(glyph)** (al valore 0 corrisponde il primo disegno, ovvero quello a sinistra).
Se per la proprietà' glyph usiamo un valore maggiore del numero di disegni (maggiore di **numglyphs** –1) il bitmap non sara' visualizzato nella finestra di edit ed al suo posta apparira' il nome del bitmap stesso.

Anche per gli oggetti Bitmapl, come per i campi di edit, è possibile settare le proprietá **rows Y** e **column X** per creare vettori o matrici di immagini.

Ad esempio:

Abbiamo un disegno che rappresenta un led nei suoi due stati spento-acceso. Creiamo un bitmap bitmap1 associato a questo disegno, in cui mettiamo:

var	ingr[]
Rows Y	1
Columns X	8
Numglyphs	2
Bitmap	AALED2

In questo modo creiamo una fila di 8 led.
Se vogliamo vedere su questi led lo stato dei primi 8 ingressi del cn400, possiamo ad esempio scrivere:

```
char ingr[8];
void Event(bitmap1_onentry)()
{
   char x;
   for (x = 0;x < 8;x++)
      ingr[x] = I(i+1);
}
```

Object:

è l'oggetto **componente**, ovvero un oggetto complesso precedentemente costruito e che si trova in libreria degli oggetti. Un componente è costruito utilizzando gli oggetti di base sopra visti, ed eventualmente altri componenti. La modularitá è un punto di forza di Proteus, in quanto consente al programmatore di utilizzare componenti prefabbricati anche molto sofisticati (PID, Posizionatori, Levigatrici ecc..) e consente inoltre di costruire da soli i propri componenti, assemblando o modificandone altri precedentemente creati o scelti tra quelli standard a corredo di Proteus.

Ad ogni object è assegnato un nome che coincide con quello della propria proprietá **var**. Tale nome deve avere tassativamente la prima lettera minuscola. Un object è identificato dal proprio nome, e in fase di stesura del programma si puó decidere se gli oggetti siano entitá assolute o, come tutti gli altri oggetti, facciano parte della pagina che li contiene. In quest' ultimo caso selezionare il checkbox oggetti locali. Un oggetto assoluto non ha bisogno di essere richiamato con pagina->oggetto ma per contro, non possono apparire su pagine diverse oggetti assoluti con lo stesso nome.

Un componente, oltre al nome-variabile non ha altre proprietá ed eventi, ma eredita quelli del componente di libreria originale. Come tale, puó avere associate molte variabili-proprietá, a seconda della sua complessitá..
- Tutti i componenti hanno la proprietá val, che coincide con il nome-variabile del componente stesso.
- Le proprietá appartengono al modello di libreria, che ha lo stesso nome del componente, ma con l' iniziale MAIUSCOLA..

Facciamo un esempio.

Prendiamo un interruttore di tipo Switch8-40, che chiameremo interrutt1 (scriviamo interrutt1 nella proprietá var e scriviamo quello che vogliamo in caption).

Con l' istruzione `interrutt1=1` l' interruttore va on (e vedremo l' immagine con la levetta alzata).
La stessa cosa se facciamo `Interrutt1->val=1.` (notare la I maiuscola).

Nella parte dedicata ai componenti standard, per ogni componente sono elencate le variabili associate. Per quanto riguarda i componenti creati dal programmatore, sará compito di quest' ultimo quello di ricordare il nome delle variabili associate (quelle dichiarate **friend** nel form di creazione del componente). Vedremo piú avanti nei dettagli l' operazione di creazione e modifica di un componente.

Static Label:

equivale all' oggetto Label, ma è un oggetto fissi che non permette di cambiare la caption da programma.

Static Bitmap:

equivale all' oggetto Bitmap ma è un oggetto fisso, che contiene un ' unica immagine statica.

Questi tipi sono presenti per ragioni storiche di compatibilità' con versioni precedenti, ma **non debbono essere utilizzati per nuovi progetti**, in quanto non saranno piu' presenti nelle prossime versioni di Proteus.

Multipanel:

è un oggetto che serve per creare la base per schedari. Vediamo con un esempio pratico la creazione e l' utilizzo di un multipanel:

-Selezioniamo con il mouse sinistro sulla barra del menu l' oggetto multipanel.
-Clicchiamo sul form per piazzare il multipanel dove vogliamo. Notiamo che il checkbox multipannello si è selezionato automaticamente.
-Con il mouse destro ridimensioniamo il pannello e con quello sinistro lo spostiamo dove desiderato.
-A questo punto scegliamo sulla barra del menu il componente panel e lo piazziamo dentro il multipanel appena creato. Vedremo che il pannello assume le dimensioni del multipanel, che appaiono due diagonali per dire che il pannello non è abilitato, e vedremo la scritta Pag. 1 in alto a sinistra.
-Ripetiamo la stessa operazione altre due volte, creando Pag 2 e Pag3.
-A questo punto selezioniamo sulla barra degli strumenti il componente Button, e piazziamolo dentro il multipanel. Vedremo che il bottone si allinea in alto a sinistra ed assume il caption Pag 1.
-Ripetiamo la costa creando i bottoni Pag 2 e Pag 3.
- Deselezioniamo adesso il checkbox multipannello. In questo modo abbiamo il fuoco sulle singole pagine e non piú sul multipagina globale. Le barre diagonali scompaiono e potremo cambiar pagina con i bottoni in alto, oppure con un doppio click del mouse all' interno della pagina stessa.
-Ogni pagina è a tutti gli effetti un panel, e ad un dato momento saranno visibili solo gli oggetti che abbiamo creato sulla pagina selezionata del multipanel.
-Non è necessario creare i bottoni di selezione, ma se non lo facciamo potremo cambiare pagina solo da programma (cambiando la proprietá glyph del multipanel) ma non manualmente, non essendoci i bottoni per farlo.
-Una volta creati i bottoni di selezione, possiamo cambiarne il nome riscrivendo la proprietá caption.

*I componenti fino ad ora visti si trovano sulla barra dei **componenti standard**. La barra dei **componenti Applicazioni** contiene altri oggetti (Applicazioni, Page, Hardware Timer, Serial Port ecc…) che vedremo dopo.*
A differenza dei componenti standard questi ultimi sono sempre attivi, anche se non siamo sulla pagina in cui sono stati definiti.
L' unico componente interessante di tale menu, per ore è il

Timer:

E' un oggetto che ha le sole proprietá **enabled** e **interval** e il metodo **on_timer**.

Se la proprietá enabled è vera (diversa da 0) l' evento on_timer è eseguito ogni "interval" millisecondi.

Le proprietá interval e enabled possono essere cambiate dal programma in ogni momento.

Le proprietá

Ogni oggetto ha delle proprietá, ovvero delle variabili che determinamno il suo aspetto e che se cambiate dal programma modificano l' oggetto stesso.
Le variabili in questione possono essere dei char, dei word, delle stringhe o altro. Ogni proprietá ha il suo tipo, o i suoi tipi, di variabile.
Le proprietá possono essere cambiate dall' interno dell' oggetto stesso, da routines (o metodi) di altri oggetti della pagina, da routines in altre pagine, dal ciclo di PLC …. In altre parole, qualunque parte del codice puó cambiare le proprietá di qualunque oggetto giá definito (vedi piú avanti, definizione e visibilitá degli oggetti).
A questo scopo le proprietá debbono essere variabili globali,.
Per fare questo tutte le proprietá degli oggetti sono contenute in una struttura associata all' oggetto stesso. Tutte le strutture degli oggetti di una pagina sono puntate da una ulteriore struttura associata alla pagina. Perció una proprietá è determinata mediante il percorso:

Pagina->oggetto->proprietá

Solo per gli oggetti di tipo componente, è selezionabile l' opzione di rendere gli oggetti globali (non legati alla pagina). Se tale opzione è attivata, la proprietá è raggiungibile mediante il percorso:

Oggetto->proprietá

Esaminiamo ora nei dettagli il significato delle proprietá degli oggetti.
Teniamo conto che non tutti gli oggetti hanno tutte le proprietá (ad esempio una Label non ha Bitmap né cornice di testo), ma che le proprietá presenti hanno lo stesso significato per tutti gli oggetti (ad es. la proprietá Top ha lo stesso significato, indipendentemente dal fatto che si tratti di un bitmap, di una label o di un button).

nametype

E' il nome dell' oggetto. In effetti non è una vera proprietá ma è il nome della struttura che racchiude tutte le altre proprietá dell' oggetto.
E' possibile usare lo stesso nametype per oggetti diversi in pagine diverse. Se invece utilizziamo lo stesso nametype per oggetti diversi nella stessa pagina, gli oggetti successivi al primo saranno **dei cloni** del primo oggetto.
Un oggetto clone funziona benissimo, ed in genere viene utilizzato questo metodo per non moltiplicare inutilmente gli oggetti di una pagina ed il codice relativo, nel caso di oggetti simili. Ad esempio i vettori di panel ed i vettori di campi di edit sono fatti mediante clonazione.
Gli oggetti cloni hanno tuttavia alcune limitazioni:
- Gli eventi vengono ereditati dal primo oggetto, che è l' unico in cui i campi relativi esistono.
- Le proprietá fondamentali esistono e sono distinte per ogni oggetto clone. Le proprietá fondamentali sono:

Caption
Var
Left
Top
Tag

- Tali proprietá sono tuttavia statiche e non possono essere modificate da programma nemmeno per l' oggetto padre.

Tutte le altre proprietá (width, height, color …) sono statiche, non possono essere modificate, sono visibili e settabili solo sull' oggetto padre, e sono uguali per tutti gli oggetti clonati.

Nametype è una label, e pertanto deve obbedire alle regole del linguaggio c per i nomi delle label, ovvero deve essere una serie di lettere e numeri, il primo carattere deve essere una lettera, è ammesso il solo carattere speciale underscore (_), non puó avere il nome di parole chiave del c (for, if, sin, cos, printf ecc.)

caption

E' una stringa associata all' oggetto.
Puó rappresentare il testo (label bottoni, checkbox) o un commento (edit). Se è una stringa fissa deve contenere non piú di 64 caratteri e viene gestita dal tool di traduzione automatica, nel caso in cui nel progetto sia dichiarato un numero di lingue maggiore di 1.
In effetti la traduzione automatica è "automatica" solo nel senso che, nell' istante in cui varia il valore della variabile Lang, tutte le caption di tutti gli oggetti visibili cambiano e mostrano il testo tradotto nella nuova lingua. Non esiste tuttavia alcun vocabolario universale di traduzione, e in effetti tale vocabolario lo dobbiamo fare noi, nel seguente modo.
Dopo aver compilato il programma, nella directory del programma stesso troveremo un file che si chiama LANG2.H. In esso si trovano, ripetute n volte, dove n è il numero di lingue indicate nel progetto, tutte le caption di tutti gli oggetti del progetto.
Al momento della creazione, tutte le stringhe sono uguali a quella presente nel campo caption. Sará nostra cura editare tale file con un editor di testo e tradurre le varie ricorrenze delle stringhe nelle lingue che vogliamo.
Quando la variabile Lang vale 1, si vedrá la prima stringa, con 2 la seconda ecc..

Ad esempio, se numero lingue=2 ed abbiamo

nametype	label1
caption	Prova

Troveremo dopo la compilazione, in LANG2.H, una riga

```
const char *L_Prova[]={"Prova","Prova"};
```

Che tradurremo ad esempio

```
const char *L_Prova[]={"Prova","Test"};
```

Ovviamente, traducendo dobbiamo fare attenzione a scrivere delle stringhe che non siano troppo lunghe e non vadano a debordare dai campi. In linea di principio non c' è alcuna limitazione al numero di caratteri di una stringa tradotta. Volendo, possiamo anche modificare la prima stringa (quella nella lingua originale).

Un altro modo di inizializzare la variabile caption e' quello di scrivere nel campo stesso il nome di un puntatore, ad esempio:

```
*string1
string1[0]
String1[]
String1[][]
```

La terza forma (indice implicito) è utilizzabile solo per vettori e matrici di oggetti (vedi campi di edit e label)
La quarta forma (doppio indice implicito) è utilizzabile solo per matrici di oggetti.

Nel caso di oggetti di tipo componente, la proprietá caption viene di solito utilizzata per passare un parametro opzionale al componente stesso, ed il suo significato cambia a seconda del componente.

var, vartype

La proprietá var è la principale differenza tra Proteus e gli altri RAD, come Borland Builder, Visual Basic, Delphi ecc.
In Proteus ogni oggetto è legato dinamicamente ad una variabile globale del progetto. Se la variabile cambia, il componente reagisce a tale cambiamento (ad es. un bitmap mostra un‘ immagine diversa, un campo di edit mostra un valore diverso). Inoltre se effettuiamo determinate azioni sul componente (ad es. se premiamo un pulsante) la variabile associata cambia.

Var è il nome di una variabile globale (che deve essere definita dal programmatore)
Vartype è il tipo di tale variabile, e puó essere scelta da menu tra i seguenti tipi:

Byte	**(char)**
Word	**(short int)**
Doubleword	**(long int)**
Real	**(float)**
String	**(char*)**

In vartype dobbiamo indicare il tipo con cui è stata definita la variabile. Se ci sbagliamo (ad es. indichiamo come byte una variabile short int), il programma puó dare risultati incontrollati o bloccarsi.

In var possiamo mettere anche una espressione, purché la sua forma sia compatibile con **un’ espressione che sta a sinistra del segno uguale**. Ad esempio

```
Quotax[testa1*para2+3]
Quotax[quotay[zz2]]
```

Sono forme corrette mentre

```
Quota1 + 3
```

Non va bene, perché

```
Quota1 +3 = 0
```

è un errore in C.

Per i vettori e le matrici di oggetti è possibile, come per la proprietá caption, utilizzare la forma ad indice implicita

Var[] e
Var[][]

Qualora la variabile cambi molto rapidamente (ad esempio la quota durante un posizionamento), si possono porre dei limiti alla frequenza di aggiornamento della variabile su display, settando la proprietá frequency ad un valore abbastanza alto (è il tempo minimo di aggiornamento dell' oggetto in millisecondi).
Ogni oggetto puó avere la proprietá frequency diversa dagli altri.
Al momento della creazione all' oggetto viene associata la variabile VAR1 di default. Questa è una variabile di sistema giá definita e puó essere lasciata se la proprietá var non interessa (ad es. per le label).

E' una variabile unica per ogni oggetto, ed è quella che, insieme a tag, distingue un oggetto clone da un altro
.

Tag

E’ un numero, che puó essere usato per scopi diversi, a discrezione del programmatore.
In genere è utilizzato per oggetti simili (pulsantiere, vettori di checkbox ecc.) per individuarne il numero.
Per buttons e checkbox mutuamente esclusivi, contiene il numero da assegnare alla variabile associata quando l’ oggetto è attivo.
Per oggetti cloni, insieme alla proprietá var, è l’ unico modo per distinguere un clone da un altro.
Deve essere definita con un numero intero nel range 0-65535. Non puó essere una variabile né altro.

visible enabled

Visible, se vero (diverso da 0) rende l'oggetto visibile, altrimenti lo cancella dal display.
Enabled, se vero, rende l'oggetto sensibile al touch-screen (o al click del mouse).

Queste proprietá, a differenza di tutte le altre, sono puntatori alla rispettiva proprieta', perció il programma per accedervi deve usare la deferenziazione. Ad esempio:

```
*page1->edit1->visible=0;
*page2->button3->enabled=1;
```

Notare l' asterisco davanti al nome della proprietá.

Un oggetto non abilitato è attivo e funziona regolarmente (ad esempio un campo di edit visualizza i cambiamenti della variabile associata), ma non è sensibile al tocco.

frequency

Questa proprietá assegna il tempo minimo di refresh di un oggetto.
Gli oggetti vengono aggiornati (ridisegnati) quando cambia una proprietá che comporta una variazione della rappresentazione dell' oggetto stesso. In certi casi oltre all' oggetto deve essere ridisegnato anche quello che gli sta attorno (ad esempio se un oggetto viene spostato, occorre ridisegnare lo sfondo per cancellare il vecchio oggetto).
Tutto questo richiede del tempo, e se uno o piú oggetti variano molto rapidamente (pensiamo alla quota di un asse durante il posizionamento) il continuo refresh puó provocare rallentamenti dell' interfaccia utente (i processi PLC non ne risentono in quanto girano in un altro task), flickering e fastidiosi effetti visivi. Per evitare questo, possiamo assegnare a tali oggetti un tempo di visualizzazione piú alto (10-20 è un buon compromesso). Oppure, nel caso in cui sia previsto che cambiano delle proprietà che richiedono di ridisegnare l' oggetto, è bene includerlo dentro un pannello non trasparente. In tal modo sará ridisegnato solo íl pannello e non tutto il form.

Oltre al valore della variabile associata, le proprietá che, in caso di variazione richiedono di ridisegnare l'oggetto sono:

Borderstyle (bevel)
BorderColor
Color
FontColor
Glyph
Tag

Le proprietá che in caso di variazione richiedono di ridisegnare anche lo sfondo sono:

BorderStyle (se con il nuovo stile la cornice è piú sottile)
Border
Caption
DecimalPointPosition (decimals)
NumDigits (digits)
Height
Left
Top
Width

proprietá specifiche dell' oggetto Edit

I campi di edit hanno alcune proprietá dedicate.

Font: è il font del campo di edit e anche della stringa caption.
Color: è il colore di fondo della finestra di edit.
Bordercolor: è il colore del bordo della finestra di edit e della caption.
Fontcolor: è il colore della scritta nel campo di edit.
Borderwidth: è l' extra-size (in pixel) dello sfondo della finestra di edit.
Rows Y: se > 1 definisce un vettore verticale di campi di edit.
Columns X: se > 1 definisce un vettore orizzontale di campi di edit.
(se sia Rows che Columns sono > 1 è una matrice di campi di edit)
Bevel: è lo stile del bordo del campo di edit (in rilievo, ribassato ecc.)
Decimals: definisce il numero di decimali dopo la virgola. Se il tipo di variabile è intero, definisce la posizione fittizia del punto decimale. Ad esempio: se la variabile è un Word e vale 1000 e decimals=2, il campo editerá il valore 10.00, se battiamo il nuovo valore 3 si visualizzerá il numero 3.00 e la variabile assumerá il valore 300).
Digits: è la larghezza fissa del campo (il numero di cifre o lettere che puó contenere, compreso il segno ed il punto decimale.

proprietá specifiche dell' oggetto Bitmap

I campi di tipo bitmap hanno alcune proprietá dedicate.

Color: è il colore di fondo del campo (visibile nelle zone trasparenti dell' immagine)
Bordercolor: è il colore del bordo del disegno.
Borderwidth: è l' extra-size (in pixel) dello sfondo del disegno.
Rows Y: se > 1 definisce un vettore verticale di campi di edit.
Columns X: se > 1 definisce un vettore orizzontale di campi di edit.
(se sia Rows che Columns sono > 1 è una matrice di campi di edit)
Bevel: è lo stile del bordo del campo di edit (in rilievo, ribassato ecc.)
Bitmap: è il nome dell' immagine da visualizzare.
Numglyphs: indica il numero di sotto-immagini in cui è divisa orizzontalmente l' immagine da visualizzare.
Select: indica il numero della sottoimmagine da visualizzare.
Se Numglyphs e' > 1, glyph assume automaticamente nel programma il valore della variabile associata. Tale campo è presente in Proteus solo per un preview degli effetti grafici durante la stesura del programma.
Scalable: se posto ad 1 permette di variare le dimensioni dell' oggetto, consentendone uno zoom o una riduzione a piacere.
Selezionando la casella fix zoom saranno consentite solo particolari valori per lo zoom (50%, 75%, 100%, 150%, 200%).
Lo zoom e' sempre lo stesso per i due assi, e non consente distorsioni dell' immagine.
La proprietà' Scalable esiste anche per gli oggetti di tipo Componente, ma in tale caso non è modificabile, ovvero: **se un componente e' stato costruito per essere scalabile, il suo nome deve finire con il suffisso _r, e la proprietà scalable sará attivata automaticamente.**

Altre proprietá:

Le proprietá **width, height, top, left** danno le dimensioni e la posizione di un oggetto (all' interno della pagina, o del pannello che lo contieene)

Le proprietá **row, column, canfocus, canedit** sono usate per campi di edit, ed utilizzate soprattutto con interfaccia tastiera senza touch screen.
Row specifica la posizione verticale del campo
Column specifica la posizione orizzontale del campo.
Mediante i tasti freccia si puo' navigare tra i campi di una pagina.
I campi accessibili sono quelli degli oggetti di tipo Edit che:

- siano contenuti in un pannello visibile
- Abbiano le proprietà row e column diverse da zero
- Abbiano la proprietà canfocus=1

I campi editabili sono quelli che, oltre alle condizioni sopra viste, abbiano la proprietà canedit=1

La proprietá **refresh,** che puo' valere **0** oppure **1,** ha significati diversi, a seconda del tipo di oggetto:
- ***Edit****: se refresh vale 1 il campo verrá aggiornato anche se il valore della sua variabile non cambia.*
- ***Button***, ***Checkbox****: se refresh vale 1 l' oggetto è mutuamente esclusivo con quelli che hanno la stessa var, ed è attivo quando il valore della var è uguale al proprio tag.*

Fontx e **Fonty** vengono usate, in abbinamento alla proprietá Font e limitatamente ai font scalabili (OFFx e TIMx) per lo zoom orizzontale e verticale dei font stessi

Align è utilizzata in pratica solo per i pannelli, e serve ad allinearli alla pagina o al pannello che li contiene.

Mobile è usata solo per i pannelli e consente di renderli mobili, ovvero trascinabili da un posto all' altro del Form mediante il touch screen.

regole pratiche per la corretta utilizzazione degli oggetti

Il sistema Proteus è molto flessibile e permette di creare l' interfaccia grafica in modo interattivo e con pochi tocchi del mouse, tuttavia se non si osservano alcune regole pratiche, il risultato puó essere scadente e dare luogo a flicker e a rallentamenti nella visualizzazione.

-Dobbiamo tener presente che quando nel programma cambiamo le proprietá degli oggetti, se questo porta ad un cambiamento fisico dell' aspetto dell' oggetto (visibile/invisibile, ridimensionamento, spostamento, cambiamento della cornice di testo, cambiamento del caption o del colore ecc..) l' oggetto verrá immediatamente ridisegnato.
In piú verranno ridisegnati tutti gli oggetti ad esso sovrapposti.
Nel caso in cui il semplice ridisegnare l' oggetto non sia sufficiente (ad esempio se riscriviamo una scritta con fondo trasparente, se rimpiccoliamo un pannello, se cambiamo un disegno) il motore grafico puó ritenere necessario di ridisegnare il fondo per cancellare la parte dell' oggetto non piú visibile. Questo comporta che verranno ridisegnati anche tutti gli oggetti sovrapposti al fondo ridisegnato.
Come conseguenza, se ad esempio cambiamo il caption di una label, tutta la pagina verrebbe ridisegnata, il che, a parte il rallentamento, provoca un effetto visivo fastidioso.
Per ovviare a questo, è bene collocare gli oggetti che possono essere ridisegnati all' interno di un pannello di colore non trasparente, in modo da limitare l' area da ridisegnare.
Non occorre prendere tale precauzione per gli oggetti quando vengono ridisegnati automaticamente, come per i campi di edit quando cambia il valore visualizzato, i bottoni quando vengono premuti, i checkbox quando vengono selezionati o deselezionati, i multipanel quando cambiamo pannello.
Se utilizziamo dei bitmap che possono cambiare, se questi hanno delle zone trasparenti, è bene definire il fondo del bitmap con un colore non trasparente, altrimenti quando cambia l' immagine, potrebbe restare in sottoimpressione la vecchia immagine nelle zone trasparenti della nuova immagine.
E' bene inoltre non creare immagini animate molto grandi, in quanto l' aggiornamento sarebbe lento e visibile. In tal caso è meglio creare un' immagine grande fissa, e in sovraimpressione delle piccole immagini animate per i particolari che cambiano.
Queste norme non sono tassative, ma dettate dal buon senso. L' esperienza del programmatore puó suggerire altre astuzie per rendere piú fluido il programma.

Gli eventi

Ad ogni oggetto sono assegnabili degli eventi, ovvero delle funzioni definite dal programmatore, che vengono eseguite quando si verificano determinate condizioni.
Gli eventi sono funzioni senza parametri, tuttavia quando vengono chiamati, vi è un certo numero di variabili globali che assumono determinati valori:

MouseX (o **mouse_x**) posizione orizzontale del mouse (o del touch sullo schermo) relativa all'oggetto stesso (0,0=angolo in alto a sinistra).
MouseY (o **mouse_y**) posizione verticale del mouse (o del touch sullo schermo) relativa all'oggetto stesso (0,0=angolo in alto a sinistra).
This (**me**) puntatore all' oggetto stesso.
Local puntatore alla pagina attuale.
Var struttura alla variabile associata.
-**Var.Byte** (**_byte**) variabile associata (caso char,unsigned char)
-**Var.Word** (**_word**) variabile associata (caso short int)
-**Var.Dword** (**_dword**) variabile associata (caso long int)
-**Var.Real** (**_real**) variabile associata (caso float)
-**Var.String** (**_string**) variabile associata (caso char [ ]).

Ad esempio, se all' interno dell' evento onentry di un oggetto scriviamo:

```
if (_word > 10) _word=10;
```

Limitiamo al valore massimo 10 il valore della variabile associata.

Nel codice di un evento possiamo inoltre leggere e modificare le proprietá dell'oggetto stesso (This->proprietá), degli altri oggetti della pagina (Local->object->proprietá) e di tutti gli altri oggetti del progetto (page->object->proprietá).
Se il codice è scritto utilizzando le variabili di default (This,Local,_byte,_word ecc..) anche altri oggetti della pagina possono utilizzare lo stesso evento, e il codice non richiede alcuna modifica se cambiamo il nome della pagina o se lo copiamo per utilizzarlo su un oggetto di un' altra pagina.
Occorre fare attenzione, riferendosi ad altri oggetti, che questi ultimi esistano, ovvero che siano giá stati creati. È buona norma usare un codice del tipo:

```
if (page1->label1) page1->label1->caption="abcd";
```

Vediamo ora il significato degli eventi degli oggetti.

on_first

Questo evento viene chiamato per ogni oggetto nel momento in cui si apre una nuova pagina.
Possiamo scrivervi ad esempio il codice di inizializzazione.

on_create

Questo evento viene chiamato ogni volta che l'intera pagina viene ridisegnata. Salvo il caso in cui si forzi da programma un refresh della pagina o si esca da un form child, equivale alla proprietá on_first.

on_entry

Questo evento viene chiamato sempre, prima di qualsiasi altro evento (salvo on_first e on_create) quando si fa il polling dell' oggetto (in media ogni millisecondo). In questa subroutine possiamo mettere ad esempio il codice di controllo dei parametri dell' oggetto, della visibilità e dell' abilitazione dello stesso.

on_draw

Viene chiamato, nel caso in cui un oggetto debba essere disegnato o ridisegnato, prima di disegnare l' oggetto stesso.
È qui che si possono fare ulteriori controlli ed eventualmente disegnare diversamente ed annullare l' azione standard di disegno (**mettendo a zero la variabile ActionEnable** che è il flag di abilitazione all' azione, ovvero al disegno)
Nel caso di oggetti di tipo Panel, se presente l' evento on_draw, esso si dovra' occupare anche dell' eventuale disegno del fondo del pannello (se non trasparente) in quanto, solo per i componenti di tipo Panel, l' evento on_draw **sostituisce l' azione di default**, la quale non sará eseguita. Questo consente l' utilizzo di un pannello come finestra attiva associata a subroutines particolari.

on_click

Viene chiamato quando l' oggetto viene toccato sul touch-screen, oppure quando vi si clicca con il mouse, o quando l'oggetto è evidenziato e si batte un tasto su tastiera. Anche in questo caso si possono fare controlli ed eventualmente annullare l' azione di default **mettendo a zero la variabile ActionEnable.**

on_mousedown

Viene chiamato una sola volta nell' istante in cui si clicca con il mouse sull' oggetto o si tocca sul touch-screen.

on_mousepress

Viene chiamato per tutto il tempo in cui il mouse (o il touch-screen) resta premuto sull' oggetto

on_mouseup

Viene chiamato una sola volta al rilascio del mouse (o del touch-screen).

on_exit

Come on_entry viene chiamato all' inizio del polling di un oggetto, cosí on_exit Viene chiamato alla fine del polling stesso.
Puó essere usato per completare altre azioni precedenti. Ad esempio, nella libreria dei componenti standard, per i componenti del tipo strumento analogico, in questo evento viene ridisegnata la lancetta dello strumento se lo strumento è stato ridisegnato o se il valore della variabile associata è cambiato.
A tale scopo, viene conservata una traccia delle chiamate degli eventi dell' oggetto nelle variabili:

_abilita_azione_click =1se eseguito on_click
_abilita_azione_repaint =1 se eseguito on_repaint (*)
_abilita_azione_draw =1 se eseguito on_draw

(*) L'azione on_repaint è una subroutine interna e viene chiamata quando l' oggetto viene ridisegnato completamente. Ad esempio per il campo di Edit on_repaint ridisegna tutto (cornice, sfondo, caption), mentre on_draw riaggiorna solo il valore del dato. Essendo on_repaint subordinato alla gestione della grafica e non ad azioni esterne, non è prevedibile e non esiste un evento corrispondente usabile dal programmatore. L' unico posto per fare qualcosa in questo caso è in on_exit usando il flag _abilita_azione_repaint).

2– La struttura del progetto

Un progetto-tipo di Proteus consiste di piú files, tutti contenuti nella directory del progetto, di cui vedremo il significato. Con Proteus **una directory puó contenere un solo progetto.**

Le Pagine

Per default i files delle pagine vengono chiamati page1.c, page2.c ecc.. Ma nessuno vieta di assegnare loro un altro nome al momento della creazione.
La prima pagina viene creata automaticamente quando si crea un nuovo progetto. Le pagine successive possono essere aggiunte dal menu Archivio->Nuova Pagina.
Appena creata il testo della pagina contiene solo l'include dell' header e la dichiarazione del form.
Vi possiamo aggiungere le routines per gli eventi e dichiarare le variabili (globali) della pagina.
Le variabili debbono essere dichiarate prima dell' include dell' header, mentre il corpo delle routines associate agli eventi viene **creato automaticamente** con un doppio click sul campo dell' evento scelto, nello schedario dell' ispettore degli oggetti.

Esempio:

```
unsigned char pippo;  //  variabile globale
#include "page1.h"
// ------------------------------+
//    form  page1
// ------------------------------+
void form_page1(void)
{
  object(TPage,&page1,page1_Body,0,0,&Null,Byte,1,0,"PAGE");
}
// ------------------------------
// methods:
// ------------------------------
void Event(edit1_onentry)()
{
//  scrivere qui il proprio codice
}
```

Effettivamente la parte testuale della pagina, scritta in linguaggio standard C contiene solo la dichiarazione delle variabili usate e il codice relativo agli eventi che vogliamo trattare in modo non standard. Nella

maggior parte dei casi non occorre scrivere niente, lasciando la gestione di default degli oggetti.
La costruzione vera e propria della pagina, il cui codice C viene generato automaticamente nel file header (pagexx.h) è eseguita in modo grafico interattivo usando la finestra grafica, la barra degli strumenti e l' ispettore degli oggetti.

La costruzione delle pagine ci permette di implementare l' interfaccia uomo-macchina. Oltre a questo, in un normale programma di automazione, è necessaria la parte operativa, che puó essere scritta in linguaggio C o/e in linguaggio Ladder.

La prima compilazione di un progetto, che contenga almeno una pagina con almeno un oggetto sopra, crea automaticamente la traccia di tutti i restanti files che possiamo editare, e che andiamo ad esaminare.
I files sottostanti si editano nella finestra di testo, e si aprono cliccando con il mouse destro all' interno di essa, scegliendo il menu Apri File.

COMMON.H

In questo files dobbiamo dichiarare le variabili globali che saranno usate dalle varie pagine e dal programma di lavoro. Pure all' inizio della pagina, come abbiamo visto, possiamo dichiarare delle variabili globali, ma solo se vengono utilizzate unicamente dalla pagina stessa.
In common.h scriveremo anche la dichiarazione di eventuali strutture, union, dichiarazioni di #define, funzioni e subroutines che useremo negli eventi delle pagine e nel ciclo macchina.

COMMONPLC.H

Qui dichiareremo le variabili, le strutture e le funzioni che NON saranno usate dagli eventi degli oggetti, ma che utilizzano come parametri proprietá e/o eventi degli oggetti, e che quindi debbono essere definiti dopo la definizione delle pagine (appunto in commonplc.h).

START.C

Contiene il codice di inizializzazione, ovvero tutto quello che dobbiamo fare una sola volta all' avvio del programma (caricamento parametri, inizializzazione delle porte seriali, degli encoders, delle quote degli assi ecc…) Le variabili usate debbono essere dichiarate in common.h o in commonplc.h.

PLC.C

È la parte del codice di automazione della macchina che è scritto in linguaggio C.
Appena creato è un file del tipo:

```
//----------------------------------------
// PLC CODE program Base
// inserire qui il codice del PLC
//----------------------------------------
if (first_cycle)  //INIZIALIZZAZIONE//
{
//----------------------------------------

//----------------------------------------
}
if(is_running)    //CICLO PLC//
{
//----------------------------------------

//----------------------------------------
}
idle();
//----------------------------------------
```

La prima condizione comprende il codice da eseguire all' avvio, la seconda il codice della macchina a regime. NON deve contenere loop di attesa o routines chiuse su sé stesse.
Puó richiamare funzioni ed utilizzare variabili dichiarate in common.h.

LANG2.H

Contiene il vocabolario delle frasi da tradurre se si è attivato il supporto multilingua. Sará nostro compito quello di tradurre le occorrenze successive alla prima nelle lingue che vogliamo utilizzare.

3– Le funzioni di libreria

Proteus nasce principalmente come tool di programmazione per apparecchiature con PLC, pertanto è corredato di base con una serie di funzioni in grado di gestire i devices dei PLC, quali

Kernel
Encoders
Timers
Orologio realtime
Derivatori
Ritardatori
Monostabili
Funzioni grafiche
Tastiera
Porte seriali
Flash SD/MMC
Penna usb
Filesystem
Can
Canopen
Ethernet
Espansioni i-o

Esaminiamo adesso le funzioni ed il significato dei relativi parametri.

KERNEL

```
int  exec_task(void(*fun)(),int numtask,int priority);
int  resume_task(int numtask);
int  suspend_task(int numtask);
int  remove_task(int numtask);
void _idle(void);
int  _p(int event);
int  _v(int event);
void _waitfor(int event);
void _event(int event);
```

ENCODERS

```
void load_asse1_fast(volatile long int *quota);
void load_asse2_fast(volatile long int *quota);
void load_asse3_fast(volatile long int *quota);
void load_asse4_fast(volatile long int *quota);
void load_asse1(void(*)(),void(*)());
void load_asse2(void(*)(),void(*)());
void load_asse3(void(*)(),void(*)());
void load_asse4(void(*)(),void(*)());
void cerca_zero(unsigned char n,long int quota);
void azzera_asse(unsigned char n);
```

INTERRUPTS

```
void load_interrupt(int num,void(*)());
```

INGRESSI ANALOGICI

```
void set_anal(int num,int level);
void refresh_anal(int num);
void pot_init(void);
void pot_start(void);
void pot_stop(void);
void pot_refresh(void);
```

INGRESSI-USCITE DIGITALI

```
unsigned char I(unsigned int n);
unsigned char O(unsigned int n);
void out(unsigned int n,unsigned char x);
void __i(void);
void __o(void);
```

TIMERS

```
int  wait(time_t n);
int  every(time_t n);
void time_init(void);
int  exec_timer(void(*subroutine)(),time_t _interval,time_t fra_quanto);
int  resume_timer(void(*subroutine)());
int  suspend_timer(void(*subroutine)());
int  remove_timer(void(*subroutine)());
int  sleep_timer(void(*subroutine)(),time_t tempo);
time_t timer(void);
float  _ftime(void);
```

OROLOGIO REALTIME

```
char *get_time(void);
char *get_date(void);
unsigned char get_oro(unsigned char c);
void set_time(char *s);
void set_date(char *s);
void set_oro(unsigned char c,unsigned char val);
```

DERIVATORI

```
unsigned char PulseOn(unsigned char *p,unsigned char i);
unsigned char PulseOff(unsigned char *p,unsigned char i);
```

RITARDATORI

```
unsigned char Delay(unsigned long int *h,int n,unsigned char i);
unsigned char DelayUp(unsigned long int *h,int n,unsigned char i);
unsigned char DelayDown(unsigned long int *h,int n,unsigned char i);
```

MONOSTABILI

```
unsigned char MonostableUp(unsigned long int *h,int n,unsigned char i);
unsigned char MonostableDown(unsigned long int *h,int n,unsigned char i);
```

FUNZIONI GRAFICHE

```
void ioinit(int priority);
void crt(int dx,int dy,int displ_cieco);
void display(int dx,int dy,int displ_cieco);
void crt_res(int dx,int dy);
```

```
void v_send(char c1);
void v_stop(void);
void v_cls(void);
void v_locate(int x,int y);
void v_llocate(int y,int x);
void _font(int font_num,int scala_x,int scala_y);
void v_font(int font_num,int scala_x,int scala_y);
void v_setfont(int fontn,int fontx,int fonty);
void v_color(int c1,int c2);
void _color(int bg,int fg);
void v_line(int x0,int y0,int dx,int dy);
void v_linec(int x0,int y0,int dx,int dy,int color);
void v_rect(int x0,int y0,int dx,int dy);
void v_rectc(int x0,int y0,int dx,int dy,int color);
void v_rectfill(int x0,int y0,int dx,int dy);
void v_rectfillc(int x0,int y0,int dx,int dy,int color);
void v_printchar(unsigned char c);
void v_print(char *s);
void v_paint(const unsigned char *s);
void v_paint_ram(unsigned long int ram,const unsigned char *s);
void v_copy(int xs,int ys,int xd,int yd,int dx,int dy);
void v_copy_from_ram(unsigned long int ram,int step,int xd,int yd,int
dx,int dy);
void v_fill_mode(unsigned char mode,int level,int color0,int color1,int
color2);
void lock(void);
void unlock(void);
```

TASTIERA

```
void v_taston(int colonna,int riga,int nrighe,int ncolonne);
void v_tastoff(void);
void v_spot(unsigned long int leds);
void v_all(unsigned char);
void v_tik(unsigned char x0);
void v_freq(unsigned int KBAUD);
unsigned char v_inkey(void);
void keyon(int lin,int col,int nx,int ny,char *show_keys);
void keyoff(void);
void key_msg(char *msg);
void key_wait(void);
void key_top(void);
void key_release(void);
void keyboard(const char* keyboar_keys);
void key_show(int lin,int col,int nx,int ny,char *show_keys,const char*
keyboar_keys);
unsigned char t_inke(void);
unsigned char t_inkey(void);
void t_input(char * string,short int max);
void t_edit(char * string,short int max);
```

PORTE SERIALI

```
int  com_enable(struct COM *com);
int  com_disable(struct COM *com);
int  com_open(struct COM *porta,long int baud);
int  com_close(struct COM *porta);
int  protocol_mode(struct COM *porta,char mode);
int  com_speed(struct COM *porta,long int baud);
int  com_tx_empty(struct COM *porta);
int  com_tx_full(struct COM *porta);
int  com_tx_size(struct COM *porta);
```

```
int  com_rx_empty(struct COM *porta);
int  com_rx_size(struct COM *porta);
int  com_tx(struct COM *porta,unsigned char c);
int  com_rx(struct COM *porta);
int  com_txsl(struct COM *porta,unsigned char *c,int len);
int  com_txs(struct COM *porta,unsigned char *c);
int  bload(struct COM *porta,void *buffer,int len);
int  bsave(struct COM *porta,void *buffer,int len);
int  rxchars(struct COM *porta);
int  com_rxlen(struct COM *porta);
int  com_rxcode(struct COM *porta);
int  onrx(struct COM *porta,void(*subroutine)());
int  ontx(struct COM *porta,void(*subroutine)());
int  polling(struct COM *porta,int numout,int numin,int tempo);
```

FLASH SERIALE

```
int sf_wdata(unsigned char *src,long int dst,long int num);
int sf_rdata(unsigned char *src,long int dst,long int num);
int sf_cdata(unsigned char *src,long int dst,long int num);
int sf_wcode(unsigned char *src,long int dst,long int num);
int sf_rcode(unsigned char *src,long int dst,long int num);
int sf_ccode(unsigned char *src,long int dst,long int num);
```

PENNA USB (S400)

```
void cf_init(void);
int  cf_dir(char *filename,char *result,char mode);
int  cf_open(char *filename,char mode);
int  cf_test(void);
int  cf_close(void);
int  cf_del(char *filename);
int  cf_rb(char *buffer,int len);
int  cf_wb(char *buffer,int len);
int  cf_seek(long int posiz);
int  cf_rd(char *buffer);
int  cf_wr(char *buffer);
int  cf_status(int mode);
int  cf_present(void);
```

GESTIONE FILES

```
FILE *fopen(const char *name,const char *mode);
int   fclose(FILE *file);
size_t fread(void *buffer,size_t count,size_t num,FILE *file);
int fgetc(FILE *file);
char *fgets(char *buffer,int max,FILE *file);
int  fscanf(FILE *file,const char *format, ... );
size_t fwrite(const void *buffer,size_t count,size_t num,FILE *file);
int fputc(int c,FILE *file);
int fputs(const char *buffer,FILE *file);
int  fprintf(FILE *file,const char *format, ... );
int  fseek(FILE *file,long offset,int whence);
long int ftell(FILE *file);
int feof(FILE *file);
long int flen(FILE *file);
long int fpointer(FILE *file);
void mount(void);
void unmount(void);
```

CAN

```
void can_time(int t);
void can_perif(int n,int t,int baud);
unsigned char can_busoff(void);
unsigned char can_msg(void);
void can_rxmsg(unsigned char n);
```

CANOPEN

```
void canopen_on(unsigned char sub);
void canopen_off(void);
unsigned char canopen_empty(void);
unsigned char canopen_full(void);
unsigned char canopen_msg(void);
void canopen_flush(void);
unsigned char canopen_rxmsg(void);
unsigned char canopen_txmsg(void);
unsigned char canopen_txmsg_len(unsigned char len);
unsigned char canopen_requestmsg(void);
unsigned char canopen_txsdo(void);
void canopen_onrx(void(*subroutine)());
int can_13_ok(void);
int can_14_ok(void);
void buson(void);
void canopen_standard_sub(void);
unsigned char canopen_perif(unsigned char i,unsigned char type,void
(*subrx1)(),void(*subrx2)());
```

ESPANSIONE I/O

```
void need(int,int,int,int);
void refresh(unsigned int perif,COM *com,long int baud,int numin,int fir-
stin,int numout,int firstout,int timeout);
```

VARIE (S400)

```
void dev_tr80(void);
int  cpufreq(void);
int  CPUFREQ(void);
```

Compatibile (VK-S4000)

```
void cf_init(void);
int  cf_dir(char *filename,char *result,char mode);
int  cf_open(char *filename,char mode);
int  cf_test(void);
int  cf_close(void);
int  cf_del(char *filename);
int  cf_rb(void *buffer,int len);
int  cf_wb(void *buffer,int len);
int  cf_seek(long int posiz);
int  cf_rd(void *buffer);
int  cf_wr(void *buffer);
int  cf_status(int mode);
int  cf_present(void);
```

crt calibrate (VK-S4000)

```
int screen_calibrate(void);
```

Ethernet (VK-S4000)

```
void EMAC_TxEnable(void);
void EMAC_TxDisable(void);
void EMAC_RxEnable(void);
```

```
void EMAC_RxDisable(void);
U32 EMACInit(int mode,int timeout);
U8 *EMACTxBuffer(void);
U32 EMACSend(U32 *EMACBuf,U32 length);U32 EMACReceive(U32 **EMACBuf,int
free);
U32 EMACFree(void);
void SetIp(IP *ip,U16 a,U16 b,U16 c,U16 d);
void SetMac(MAC *mac,U16 a,U16 b,U16 c,U16 d,U16 e,U16 f);
void SetMacFilter(MAC *mac);
HOST *CreateHost(U8 a,U8 b,U8 c,U8 d,U16 port);
void DeleteHost(HOST *host);
U32 EMAC_Tx(HOST *host,U8 *txbuffer,U16 flags,U16 txlen);
U32 EMAC_Rx(HOST *host,U8 *rxbuffer,U16 flags,U16 rxmaxlen);
U32 EMAC_Server(U16(*subio)(U8 *rxbuffer,U8 *txbuffer,U16 flags,U16
port,U16 rxlen),U32 flags);
```

Canbase (VK-S4000)

```
short canbase_init(unsigned short can_isrvect,unsigned int can_btr);
short canbase_txmsg (CANBASE_MSG *pTransmitBuf);
short canbase_rxmsg (CANBASE_MSG *pReceiveBuf);
```

Framebuffers (VK_S4000)

```
void crt_page(int page);
void crt_view(int page);
void crt_write(int page);
```

DOS (VK-S4000)

```
void exec_application(char *name);
int batch(char *filename,char *para1,char* para2);
```

VARIE (VK-S4000)

```
void key_autoshow(int nx,int ny,const char* keys);
int xfcopy (const char *oldname, const char *newname);
void start_download(void);
void stop_download(void);
void exec_download(void);
void status_download(void *s);
int str_width(char *s,int font);
int str_height(int font);
float fp_ok(void);
U32 heap_limit(void);
```

KERNEL

Il sistema operativo utilizzato nei programmi generati da Proteus ES versione S400 supporta **3 task** concorrenti accessibili, a cui è possibile assegnare delle prioritá.
Il task 1 gestisce l' interfaccia grafica uomo-macchina e per default ha prioritá 4.
Il task 2 gestisce i cicli di PLC, quello scritto in linguaggio Ladder e quello scritto in C, i quali vengono eseguiti alternativamente. La prioritá di default è 4.
Il task 3 è utilizzato per il controllo dell' interpolazione lineare-circolare, e se quest' ultima non è utilizzata, puó essere usato liberamente dal programmatore. Si consiglia una prioritá non superiore a 2.
Il sistema operativo ha un **full preemptive round robin time-sharing**, con un time-slice settabile da 100 usec. a 1 msec. e con un tempo di latenza massima da 0.9 mSec. a 2 mSec.
Sono previsti ed utilizzabili dal programmatore le classiche funzioni di gestione dei task (exec, remove, suspend, resume), i semafori di sincronizzazione e di mutua esclusione, e la gestione di eventi significativi (p,v,waitfor, event).
Normalmente non è necessario utilizzare alcuna di queste funzioni in un normale programma PLC-Interfaccia Grafica, tuttavia esse esistono e sono documentate, nel caso il programmatore volesse usarle.

Nella versione ARM (VK3-V8 ecc.) sono supportati **32 task**.
In tal caso il comando exec_task puó contenere come numero di task il numero ZERO, e il sistema provvederá automaticamente ad assegnargli il numero di un task libero.
In questa versione è possibile eseguire i comandi suspend_task, remove_task ecc. passando nel parametro **numero del task** il **nome** del task stesso.

Esempio:

```
exec_task(mytask,0,4);
.
.
.
suspend_task(mytask);
```

```
int  exec_task
(void(*fun)(),int numtask,int priority);
```

Esegue un task.

Fun è il nome della funzione senza argomenti che viene eseguita come task. Deve contenere un loop chiuso e non deve mai terminare, altrimenti il risultato è imprevedibile.
Il task 1 è eseguito automaticamente allo start della macchina e la funzione associata è il main.
Se esiste ed è attivato il PLC esso viene avviato automaticamente come task 2 e contiene giá un loop chiuso.
Se non abbiamo bisogno del PLC standard e vogliamo usare un task 2 definito da noi, dobbiamo includere in COMMON.H la linea:

```
#define NO_PLC
```

Che esclude la generazione del codice e l' inclusione del plc standard.
Il task 3 è riservato al programmatore, salvo il caso in cui nel progetto siano utilizzati dei componenti di interpolazione.

Numtask è il numero del task (1, 2 o 3 per S400), (0..32 per ARM)
Il task 0 è allocato automaticamente dal sistema in un task libero.

Priority è la prioritá del task o meglio il numero di slot temporali ad esso riservato.
Uno slot temporale dura 100 uSecondi.
I task 1 e 2 hanno per default prioritá 4, e quindi girano alternativamente per 400 uSec ciascuno. In tal modo il tempo totale di timesharing è di 800 uSec. E il massimo tempo di latenza, ovvero il tempo in cui un task non gira, è di 400 uSec.

Codice di ritorno:
0 = operazione riuscita
-1 = errore.
Viene generato errore e l'operazione non viene eseguita se si tenta di eseguire un task in uno slot in cui è giá attivo un altro task.
Affinché l'operazione riesca occorre che lo slot indicato non sia assegnato ad alcun task, oppure che quest'ultimo sia sospeso o rimosso.

Esempio:

In common.h definiamo i task e disabilitiamo il PLC

```
#define NO_PLC
// task che fa lampeggiare un' uscita
void lampeg(void)
{
   while(1)
   {
      o1=o1 ^1;
      wait(1 SEC);
   }
}

// task che attiva un' uscita 1 sec alla settimana
void settim(void)
{
   while(1)
   {
      o2=1;
      wait(1 SEC);
      o2=0;
      wait(1 WEEK – 1 SEC);
   }
}
```

Da qualche parte nel codice del programmatore ...

```
// esecuzione del task lampeg come task 2 prioritá 1
// e del task settim come task 3 prioritá 1
exec_task(lampeg,2,1);
exec_task(settim,3,1);
```

```
int  resume_task(int numtask);
```

Attiva un task sospeso.

Numtask è il numero dello slot in cui si trova il task da riattivare..
(oppure il nome del task nella versione per ARM, purché la routine avente tale nome non sia stata lanciata su piú task)

Il **codice di errore** ritorna con il valore

0 se l'operazione è andata a buon fine
-1 se lo slot non è assegnato a nessun task, oppure è assegnato ad un task giá attivo o rimosso.

n.b. Suspend e Remove sono due comandi molto simili, ed hanno entrambi come effetto quello di disattivare un task.
Un task sospeso con suspend puo' essere definitivamente rimosso con remove oppure riattivato con resume.
Un task rimosso con remove è rimosso definitivamente e se si vuole rimandarlo in esecuzione, occorre un nuovo exec ed il task ripartirá dall' inizio.
Un task può sospendere o rimuovere se stesso, nel qual caso, affinché l'azione avvenga immediatamente e non attenda la fine del tempo di slot, occorre far seguire al suddetto un comando di **idle()** che forza il reskeduling.
Ovviamente un task non puó fare il resume o l' exec di sé stesso.

Esempio:

```
// con rif. all' esempio in exec_task ...
resume(2);     //ripristina il task lampeg
resume(lamp);  //solo versione ARM
```

```
int  suspend_task(int numtask);
```

Sospende un task.

Numtask è il numero dello slot in cui si trova il task da sospendere. (oppure il nome del task nella versione per ARM, purché la routine avente tale nome non sia stata lanciata su piú task)

Il **codice di errore** ritorna con il valore

0 se l'operazione è andata a buon fine
-1 se lo slot non è assegnato a nessun task, oppure è assegnato ad un task giá sospeso o rimosso, oppure se il task impegna dei semafori di risorse. (ad esempio, nessuno puó sospendere o rimuovere un task che ha riservato il video o la penna USB, altrimenti tali risorse resterebbero bloccate per sempre)

n.b. Suspend e Remove sono due comandi molto simili, ed hanno entrambi come effetto quello di disattivare un task.
Un task sospeso con suspend puo' essere definitivamente rimosso con remove oppure riattivato con resume.
Un task rimosso con remove è rimosso definitivamente e se si vuole rimandarlo in esecuzione, occorre un nuovo exec ed il task ripartirá dall' inizio.
Un task puó sospender o rimuovere se stesso, nel qual caso, affinché l'azione avvenga immediatamente e non attenda la fine del tempo di slot, occorre far seguire al suddetto un comando di **idle()** che forza il reskeduling.
Ovviamente un task non puó fare il resume o l' exec di sé stesso.

Esempio:

```
// con rif. all' esempio in exec_task ...
suspend(2);  //sospende il task lampeg
```

```
int  remove_task(int numtask);
```

Rimuove un task.

Numtask è il numero dello slot in cui si trova il task da rimuovere (oppure il nome del task nella versione per ARM, purché la routine avente tale nome non sia stata lanciata su piú task)

Il **codice di errore** ritorna con il valore

0 se l'operazione è andata a buon fine
-1 se lo slot non è assegnato a nessun task, oppure è assegnato ad un task giá rimosso, oppure se il task impegna dei semafori di risorse (per il perché vedi suspend_task).

n.b. Suspend e Remove sono due comandi molto simili, ed hanno entrambi come effetto quello di disattivare un task.
Un task sospeso con suspend puo' essere definitivamente rimosso con remove oppure riattivato con resume.
Un task rimosso con remove è rimosso definitivamente e se si vuole rimandarlo in esecuzione, occorre un nuovo exec ed il task ripartirá dall' inizio.
Un task puó sospender o rimuovere se stesso, nel qual caso, affinché l'azione avvenga immediatamente e non attenda la fine del tempo di slot, occorre far seguire al suddetto un comando di **idle()** che forza il reskeduling.
Ovviamente un task non puó fare il resume o l' exec di sé stesso.

Quando un task è rimosso da uno slot, a tutti gli effetti è come se nello slot non avesse mai girato alcun task.

Esempio:

```
// con rif. all' esempio in exec_task ...
remove(2);  //rimuove il task lampeg
```

void _idle(void);

Passa al task successivo

Questo comando fa passare immediatamente il time-sharing al task successivo, senza attendere la fine dello slot temporale a lui assegnato. Puó essere usato per velocizzare la macchina, nel caso in cui stiamo attendendo che succeda un evento e non vogliamo impegnare inutilmente il tempo macchina nell' attesa.

Esempio:

```
// attesa sblocco
while(!i22)
{
   _idle();
}
// seguito del programma
```

Attendiamo che l' ingresso 22 si attivi per proseguire. Nell' attesa lasciamo tempo agli altri task.

```
int  _p(int event);
int  _v(int event);
```

Sono i semafori di mutua esclusione. Viaggiano sempre in coppia.
Event è il numero di una risorsa a cui si vuole fare accesso.

p(event) prenota la risorsa, se questa è occupata autosospende il proprio task, altrimenti continua.
v(event) libera una risorsa precedentemente prenotata e fa il resume di eventuali altri task che siano stati sospesi in attesa di tale risorsa.

Ritornano con **codice di errore 0** se l' operazione ha avuto successo.
p(event) ritorna con **codice di errore –1** se il semaforo è giá prenotato dal proprio task.
v(event) ritorna con **codice di errore –1** se il semaforo è libero o se è prenotato da un altro task task.
Solo il task che ha prenotato una risorsa ha il diritto di liberarla.
Il sistema operativo di Proteus 2007 puó gestire **32 semafori**.
Il **semaforo 1** è riservato per il CRT (vedi lock e unlock)
Il **semaforo 2** e' riservato alla penna USB (vedi mount,unmount)
I semafori da 3 a 31 sono liberamente utilizzabili dal programmatore.

Esempio:

```
// in task 1...
_p(5);
v_font(OFF8,1,1);
v_color(1,254);
v_llocate(3,3);
v_print("hello !");
_v(5);

// in task 2...
_p(5);
v_font(OFF16,1,1);
v_color(0,100);
v_llocate(3,23);
v_print("ciao !");
_v(5);
```

```
void _waitfor(int event);
void _event(int evento);
```

Sincronizzazione tra task

Evento è il numero di un evento (un numero compreso tra 1 e 127).

Queste due funzioni consentono la sincronizzazione tra loro di due task diversi.
_waitfor sospende il task mettendolo in attesa dell' evento passato come argomento.
_event dichiara l'evento passato come argomento e fa il resume degli eventuali task che sono sospesi in attesa di tale evento. Se ve ne erano, passa immediatamente il controllo al primo di essi, senza attendere la fine del proprio slot temporale.
Casi particolari:
_waitfor(0) disabilita l'interrupt di sistema
_waitfor(254) autosospende il task.
_waitfor(255) autoremove il task..
_event(254) fa il resume di tutti i task sospesi

Esempio:

```
// in task 1…
While(timo)
{
  _waitfor(33);
  v_font(OFF8,1,1);
  v_color(1,254);
  v_llocate(3,3);
  printf("timeout %d",timo);
}

// in task 2...
timo=xx;
_event(33);
```

Il task 1 è in attesa che timo vada a zero e si autosospende. Il task 2, quando cambia la variabile timo, notifica il cambiamento tramite l'evento 33 risvegliando il task 1.

ENCODERS

I PLC della serie S400 hanno fino a 4 controlli asse ad encoders a bordo, espandibili a 16 (**) con moduli aggiuntivi.
Supportano encoders incrementali bidirezionali con tacca zero operanti fino a frequenze di 1 Mhz. (10KHz. In ricerca di zero).
Gli assi da 1 a 4 sono abilitati con la dichiarazione **`load_asseN_fast`**
Gli assi oltre al quarto sono gestiti in modo trasparente al software, e vengono abilitati con la dichiarazione **`can_perif`** (vedi capitolo CAN).

Le quote,sono doppi word con segno, ed il sistema definisce automaticamente le seguenti variabili:

encoder1...encoder16 la quota (in impulsi encoder)
enczero1...enczero16 la quota di zero (*)
searching_z[0...15] flag ricerca di zero in corso (read only)
search_z[0...15] flag start ricerca di zero (write only)

Agli assi interni possiamo anche associare un nome della quota diverso (*se non utilizziamo le routines automatiche di azzeramento su tacca zero*) ed eventualmente possiamo anche scrivere delle routines personalizzate per gli interrupts.
La sola cosa che diversifica gli assi interni da quelli esterni è il diverso modo per dichiararli. Una volta dichiarati, vengono trattati esattamente nella stessa maniera.

(*) Si tratta di una variabile di supporto, che non influenza il valore letto della quota, ma è solo un promemoria, utilizzabile se vogliamo come offset per calcolare il valore reale della quota.

(**) *Espandibili a 32 con moduli can-open.*

```
void load_asse1_fast(volatile long int
*quota);
void load_asse2_fast(volatile long int
*quota);
void load_asse3_fast(volatile long int
*quota);
void load_asse4_fast(volatile long int
*quota);
```

Abilitazione assi interni

quota variabile long int associata alla quota

Con queste dichiarazioni, che d' abitudine si trovano nel file **START.C**, vengono definiti ed attivati gli assi ad encoder interni al modulo principale S400 (quello che contiene il programma).
Se vogliamo utilizzare le routines di azzeramento assi su tacca di zero, per la variabile quota dobbiamo usare le variabili di sistema **encoder1,encoder2,encoder3 ed encoder4** rispettivamente.
Volendo, possiamo dichiarare ed utilizzare per le quote qualsiasi variabile, purché di tipo **long int**.

Se nel progetto è dichiarato un numero di assi maggiore di zero, l' inizializzazione degli assi dichiarati è fatta automaticamente dal compilatore e non occorre che il programmatore dia il comando load_asseX_fast.

Esempio:

```
// in start.c
load_asse1_fast(encoder1);
load_asse2_fast(encoder2);
load_asse3_fast(encoder3);
load_asse4_fast(encoder4);
```

```
void load_asse1(void(*)(),void(*)());
void load_asse2(void(*)(),void(*)());
void load_asse3(void(*)(),void(*)());
void load_asse4(void(*)(),void(*)());
```

Routines di interrupt personalizzate

Con questi comandi che d' abitudine si trovano nel file **START.C**, possiamo definire delle routines di interrupt personalizzate per gli encoders.
Ad ogni impulso UP dell' encoder sará chiamata la funzione dichiarata nel primo parametro.
Ad ogni impulso DOWN dell' encoder sará chiamata la funzione dichiarata nel secondo parametro.
Le funzioni debbono essere senza parametri e senza variabile di ritorno (**void(*)()**).
ATTENZIONE: Questa gestione degli encoders puó ridurre notevolmente la frequenza massima di lavoro degli stessi.
QUESTA FUNZIONE NON È UTILIZZABILE SE IL PROGRAMMA DEVE GIRARE SU PC.

Esempio:

```
// in COMMON.H…
void up1(void)
{
  if (encoder1<100) encoder1++;
}
void dn1(void)
{
  if (encoder1>0) encoder1–;
}

// in START.C
load_asse1(up1,dn1);
```

In questo esempio gestiamo la quota dell' asse 1 limitata tra 0 e 100.
A cosa serve? A nulla, è un esempio!.

```
void cerca_zero(unsigned char n,long int q);
```

ricerca tacca di zero.

n numero dell' asse (1..16)
q valore da assegnare a enczero dell'asse.

Questo comando setta il flag search_z dell' asse corrispondente.
Il flag **searching_z** dell' asse corrispondente (**meno 1, vedi esempio**) va ad 1 e vi resta fino a quando la tacca di zero è stat trovata. Quando ció avviene la quota dell'asse (encoderX) va a **ZERO**.
Il flag searching_z, se ad 1 indica che la ricerca è in corso, ma per fare un controllo dobbiamo attendere il reskeduling per gli assi interni ed almeno 5 mSec. Per quelli esterni, altrimenti non possiamo sapere, se tale flag vale zero, se la ricerca è giá finita o non ancora iniziata. Non è detto inoltre che tale flag vada ad 1, in quanto se il comando viene dato quando l'encoder è giá sulla tacca di zero, non lo vedremo mai ad 1.

La sequenza corretta è dunque:
1- cerca_zero(..)
2- idle() (obbligatorio se c' è il 3)
3- while(searching_z[..]) {} (facoltativo)

Affinché la tacca di zero sia trovata, occorre che duri almeno 100 uSec., ovvero che la frequenza dell' encoder **non superi i 10 KHz**.

Esempio:

```
// ricerca tacca di zero asse 1
cerca_zero(1,1000);
idle();
while(searching_z[0]);
// ricerca tacca di zero asse 12
cerca_zero(12,0);
wait(5);
while(searching_z[11]);
```

Il valore 1000 di q va in enczero1.
Quando la tacca di zero è trovata encoder1 va a ZERO.

```
void azzera_asse(unsigned char n);
```

Azzera la quota di un asse.

n numero dell' asse (1..16)

Questa funzione è utilizzabile per assi interni ed esterni.
Per azzerare la quota di un asse non è sufficiente mettere a zero la relativa variabile encoderX, in quanto questa è controllata dall' hardware.
Solo per gli assi interni, e solo se il programma non gira su PC, è possibile assegnare un valore ad una quota nel seguente modo:

```
IEN=0;
encoderX=valore;
IEN=1;
```

In tutti gli altri casi, possiamo solo mettere a zero la quota mediante l' istruzione azzera_asse.
Dobbiamo vedere le quote encoderX come contatori di impulsi di un encoder assoluto, piuttosto che come quote vere e proprie, ed utilizzare un offset ed un fattore di moltiplicazione per avere la quota in unitá fisiche.

Esempio:

```
// azzera asse 1
azzera_asse(1);
// azzera asse 12
azzera_asse(12);
```

INTERRUPTS (solo versione S400)

Il PLC S400 gestisce un certo numero di risorse mediante routines di interrupt. Il programmatore puó personalizzare tali routines sostituendo il proprio codice a quello di default. Si tratta di una operazione molto delicata e per programmatori esperti, e puó dar luogo a malfunzionamento del programma e al blocco della macchina se non si sa esattamente quello che si sta facendo.
La spiegazione dettagliata della struttura dell' interrupt della cpu dell'S400 è fuori dagli intenti di questo manuale, per cui si rimanda all' user manual della cpu (c167cs_um_v2.0_2000_07.pdf) liberamente scaricabile dal sito
www.keil.com/dd/docs/datashts/infineon/**c167cs_um**.pdf
per approfondimenti.

ATTENZIONE: Utilizzando routines personalizzate ad interrupt ci si lega al particolare hardware del PLC, cosa che è fuori dalla filosofia di Proteus, e si creano programmi che non possono girare su PC e che sicuramente non saranno compatibili con successive versioni del PLC S400.

`void load_interrupt(int num,void(*)());`

Carica il vettore di interrupt della routine nel sistema operativo.

esempio

```
#define T0TRAP 0x20
static interrupt (T0TRAP) using (_RBANKT0) void
t0_int( void )
{
    step_passo=step_passo ^ 1;
    o18=step_passo;
    if (!step_passo) step_quota--;
   _OUT2=_lout2;
    if (!step_quota) T0IE=0;
}
load_interrupt(T0INT,t0_int);
T01CON=(T01CON & 0xff00) | 0x40;
T0IC=0x2c;
```

INGRESSI-USCITE ANALOGICHE

Gli ingressi ed uscite sono trattate dal programma come variabili che vengono gestite dal sistema operativo, il quale, al momento opportuno, invia fisicamente i valori attuali all' I/O.
Il ritardo, dal cambiamento di una variabile al cambiamento dello stato dell' ingresso-uscita è inferiore al millisecondo (massimo 2 mSecondi per gli I/O delle espansioni) e con apposite istruzioni possiamo forzare l' aggiornamento immediato.

Ingressi analogici: **pot1,pot2**… oppure **Pot[0],Pot[1]…**
Uscite analogiche: **anal1,anal2**.. Oppure **Anal[0],Anal[1**]...

Le due forme sono equivalenti.

Il controllore S400 **ha 4 uscite analogiche interne**+12 esterne con i moduli di espansione,operanti nel range **–32000…+32000**.

Ha inoltre **8 ingressi analogici interni**+24 esterni con i moduli di espansione, operanti nel range **0...4095**.

```
void set_anal(int num,int level);
```

Cambia fisicamente il livello di una uscita analogica

num numero dell' uscita analogica
level valore da assegnare

Questo comando cambia fisicamente il livello dell' uscita analogica (interna) ed aggiorna il valore della variabile associata.
Si usa nel caso in cui sia importante un aggiornamento immediato del livello.

Esempio:

```
void refresh_anal(int num)
{
   int i;
   num++;
   for (i=1;i<num;i++)
     if(anol[i]!=anal[i]) set_anal(i,anal[i]);
}
```

L'esempio è l' implementazione in libreria del comando refresh_anal (vedi pagina seguente), il quale utilizza set_anal per aggiornare le uscite analogiche che sono cambiate.

```
void refresh_anal(int num);
```

Esegue fisicamente l' aggiornamento delle uscite analogiche che sono cambiate.

num numero di uscite analogiche interne (= 4)

Questa routine è chiamata automaticamente dal sistema operativo all' incirca 1 volta ogni millisecondo, per cui, se non ci sono motivi per un aggiornamento piú rapido, non occorre chiamarla nel programma.
A differenza della routine set_anal questa ultima puó aggiornare piú uscite analogiche, e solo quelle che sono cambiate.
Nella pagina precedente possiamo vedere il listato di questa funzione.

Esempio:

```
// ---------------
anal1=i1*1000-500;
anal2=i2*1000-500;
anal4=i3*1000-500;
refresh_anal(4);
// ---------------
```

In questo caso refresh_anal esegue l' aggiornamento fisico solo quando l' ingresso digitale relativo cambia di livello.

```
void pot_init(void);
```

Inizializza l'interrupt e il DMA per la lettura dei potenziometri (ingressi analogici)

Questa routine è utilizzata internamente dal sistema operativo e non deve mai essere usata nel programma.

```
void pot_start(void);
void pot_stop(void);
```

Avvio ed arresto della lettura dei potenziometri.

La lettura dei potenziometri è un' operazione piuttosto onerosa dal punto di vista del tempo macchina, per cui è previsto di poterla far partire quando serve e di stopparla quando non serve piú.

All' avvio della macchina, la lettura è disattivata, per cui, per poter leggere i potenziometri, è necessario un comando di pot_start.
Di solito questo comando si trova in START.C.

Esempio:

```
// ---------------
pot_start();
// ---------------
```

```
void pot_refresh(void);
```

Aggiorna la lettura dei potenziometri.

Questo comando esegue la media sui valori di lettura dei potenziometri e aggiorna il valore delle variabili degli ingressi analogici (pot[n]).
Viene chiamato periodicamente dal sistema operativo, e **non deve mai essere usato nel programma**.

INGRESSI-USCITE DIGITALI

Gli ingressi ed uscite sono trattate dal programma come variabili che vengono gestite dal sistema operativo, il quale, al momento opportuno, invia fisicamente i valori attuali all' I/O.
Il ritardo, dal cambiamento di una variabile al cambiamento dello stato dell' ingresso-uscita è inferiore al millisecondo (massimo 2 mSecondi per gli I/O delle espansioni) e con apposite istruzioni possiamo forzare l' aggiornamento immediato.

Ingressi digitali: **i1,i2...**
Uscite digitali: **o1,o2...**

Il controllore S400 **ha 32 uscite digitali interne** + 96 esterne con i moduli di espansione.
Ha inoltre **32 ingressi analogici interni** + 96 esterni con i moduli di espansione.

Gli ingressi e uscite suddette vengono visti come variabili predefinite di tipo bit **i1...i128** e **o1...o128**.

Con ulteriori moduli RS-232 e can-open possiamo arrivare ad un massimo di **160 ingressi** e **160 uscite** indirizzabili direttamente (i160, o160) e fino a **2048 ingressi** e **2048 uscite** gestite a gruppi di 8.

Esempio:

```
// ——————————————————
if ( i1 & !i2) o1=1;
o2=i3 | i4;
// ——————————————————
```

```
void __i(void);
void __o(void);
```

Refresh immediato degli I/O

Qualora per scopi particolari volessimo un aggiornamento immediato delle uscite, dobbiamo dare **il comando __o() dopo** le espressioni che cambiano le uscite.
Allo stesso modo se vogliamo leggere immediatamente lo stato attuale degli ingressi, dobbiamo dare **il comando __i() prima** di esaminare gli ingressi.

Tali comandi agiscono solo sui **32 ingressi e uscite a bordo** dell' S400 e **non sono utilizzabili per programmi che girano sul PC**.

Esempio:

```
// ----------------
__i();
if ( i1 & !i2) o1=1;
o2=i3 | i4;
__o();
// ----------------
```

```
unsigned char I(unsigned int n);
unsigned char O(unsigned int n);
void Out(unsigned int n,unsigned char x);
```

Questi comandi consentolno la lettura e l' aggiornamento degli i/o digitali in forma vettorizzata

Ad esempio,

```
if ( i1 & !i2) o1=1;
o2=i3 | o4;
```

Puó essere anche scritto:

```
int x1=1,x2,2,x3=3,x4=4;
if ( I(x1) & !I(x2) ) out( x1 , 1 );
Out( x2 , I(x3) | O(x4) );
```

TIMERS

Il sistema operativo di Proteus 2007 gestisce un system clock con risoluzione di 1 mSec. con il quale si possono creare tempi di attesa o eseguire delle funzioni a cadenza regolare.

Il **minimo** intervallo di tempo gestibile è **1 millisecondo**.
Il **massimo** intervallo di tempo gestibile è di 2^31-1 millisecondi pari a **24 giorni + 20 ore + 31 minuti + 23 secondi**, che è il tempo di cadenza massima di ripetizione di una routine a tempo o il tempo di attesa massimo.

Il tempo, ove non diversamente specificato, viene passato in mSec. (10 = 10 mSec.) e si possono usare le seguenti parole chiave per specificare unitá di misura di tempo diverse:

MSEC millisecondo/i
SEC secondo/i
MIN minuto/i
HOUR ora
DAYgiorno
WEEK settimana
HOURS ore
DAYS giorni
WEEKS settimane
ONCE una sola volta
NOW adesso

Ad esempio sono espressioni corrette:

```
23 SEC
1 DAY + 3 HOURS + 18 MSEC
2 WEEKS + 1 HOUR + 5 SEC
```

Il sistema operativo per la serie S400 è in gradio di gestire fino a un **massimo** di **32 routines a tempo attive contemporaneamente** , mentre per la serie ARM si possono gestire **fino a 100 routines a tempo attive contemporaneamente**.

```
int exec_timer(
void(*subroutine)(),
time_t interval,
time_t fra_quanto);
```

Esegue una routine a tempo specificando la cadenza e il momento di prima esecuzione.

Subroutine è il nome di una funzione senza argomenti e senza codice di ritorno, che deve essere eseguita periodicamente. Deve durare un tempo ragionevole, per non bloccare il sistema.

Interval è il tempo di ripetizione. Se deve essere eseguita una sola volta, usare per questo parametro il valore ONCE oppure 0.

Fra_quanto è il ritardo di esecuzione. Se si vuole l' esecuzione immediata (ovvero allo scadere del millisecondo corrente) usare il valore NOW oppure 0.

Questo comando puó essere usato per mandare in esecuzione una routine a tempo, oppure per cambiare il tempo di esecuzione o per ritardare una routine a tempo giá in esecuzione.

Esempio:

```
// routines a tempo
void lampeggia(void) {o1=!o1;}
void attiva(void) {o2=1;}
// . . . .
// da qualche parte nel programma ...
// esegue lampeggia a cadenza 1 secondo
// attiva o2 tra 1 ora
exec_timer(lampeggia,1 SEC, 10 MSEC);
exec_timer(attiva,ONCE, 1 HOUR);
// . . . .
// cambia la cadenza a 1/2 sec.
exec_timer(lampeggia,500 MSEC, NOW);
```

```
int  resume_timer(void(*subroutine)());
```

Ripristina una routine a tempo

Subroutine nome della routine a tempo, precedentemente mandata in esecuzione con il comando exec_timer

Questo comando ripristina la normale esecuzione della routine a tempo specificata, con la cadenza precedentemente dichiarata nel comando exec_timer.

Esempio:

```
// routine a tempo
void lampeggia(void) {o1=!o1;}
// . . . .
// da qualche parte nel programma ...
// esegue lampeggia a cadenza 1 secondo
exec_timer(lampeggia,1 SEC, 10 MSEC);
// . . . .
// cambia la cadenza a 1/2 sec.
exec_timer(lampeggia,500 MSEC, NOW);
// . . . .
// sospende lampeggia
suspend_timer(lampeggia);
// . . . .
// riattiva lampeggia
resume_timer(lampeggia);
// . . . .
// sospende lampeggia per 1 minuto
sleep_timer(lampeggia, 1 MIN);
// . . . .
// rimuove lampeggia
remove_timer(lampeggia);
```

int **suspend_timer**(void(*subroutine)());

Sospende una routine a tempo

Subroutine nome della routine a tempo, precedentemente mandata in esecuzione con il comando exec_timer

Questo comando sospende a tempo indeterminato la normale esecuzione della routine a tempo specificata. La routine resta comunque in vita e continua ad occupare il suo posto nel vettore delle 32 routines a tempo dichiarabili contemporaneamente.
Se pensiamo che essa non debba piú essere mandata in esecuzione, è meglio dare un comando di remove_timer per far posto ad eventuali altre routines a tempo.

Esempio:

```
// routine a tempo
void lampeggia(void) {o1=!o1;}
// . . . .
// da qualche parte nel programma ...
// esegue lampeggia a cadenza 1 secondo
exec_timer(lampeggia,1 SEC, 10 MSEC);
// . . . .
// cambia la cadenza a 1/2 sec.
exec_timer(lampeggia,500 MSEC, NOW);
// . . . .
// sospende lampeggia
suspend_timer(lampeggia);
// . . . .
// riattiva lampeggia
resume_timer(lampeggia);
// . . . .
// sospende lampeggia per 1 minuto
sleep_timer(lampeggia, 1 MIN);
// . . . .
// rimuove lampeggia
remove_timer(lampeggia);
```

int **remove_timer**(void(*subroutine)());

Rimuove una routine a tempo

Subroutine nome della routine a tempo, precedentemente mandata in esecuzione con il comando exec_timer

Questo comando rimuove una routine a tempo dalla lista delle 32 routines a tempo che possono coesistere contemporaneamente.
La routine ha assolto il suo compito, lasciamo posto alle altre.

Esempio:

```
// routine a tempo
void lampeggia(void) {o1=!o1;}
// . . . .
// da qualche parte nel programma ...
// esegue lampeggia a cadenza 1 secondo
exec_timer(lampeggia,1 SEC, 10 MSEC);
// . . . .
// cambia la cadenza a 1/2 sec.
exec_timer(lampeggia,500 MSEC, NOW);
// . . . .
// sospende lampeggia
suspend_timer(lampeggia);
// . . . .
// riattiva lampeggia
resume_timer(lampeggia);
// . . . .
// sospende lampeggia per 1 minuto
sleep_timer(lampeggia, 1 MIN);
// . . . .
// rimuove lampeggia
remove_timer(lampeggia);
```

```
int  sleep_timer(void(*subroutine)
(),tempo);
```

Sospende per un certo tempo una routine a tempo

Subroutine nome della routine a tempo, precedentemente mandata in esecuzione con il comando exec_timer

Questo comando sospende per un certo tempo una routine a tempo. Ripetendo piú volte tale comando i tempi di sospensione NON SI SOMMANO ma VALE L' ULTIMO.
Evidentemente, se ripetiamo questo comando con cadenza piú frequente del tempo di sospensione impostato, la routine non verrá mai eseguita. Possiamo usare questo metodo per creare un **WATCHDOG**, un SAVESCREEN o funzioni simili.

Esempio:

```
// routine a tempo
void lampeggia(void) {o1=!o1;}
// . . . .
// da qualche parte nel programma ...
// esegue lampeggia a cadenza 1 secondo
exec_timer(lampeggia,1 SEC, 10 MSEC);
// . . . .
// cambia la cadenza a 1/2 sec.
exec_timer(lampeggia,500 MSEC, NOW);
// . . . .
// sospende lampeggia
suspend_timer(lampeggia);
// . . . .
// riattiva lampeggia
resume_timer(lampeggia);
// . . . .
// sospende lampeggia per 1 minuto
sleep_timer(lampeggia, 1 MIN);
// . . . .
// rimuove lampeggia
remove_timer(lampeggia);
```

```
int  wait(time_t n);
```

Sospende per un certo tempo il task corrente.

n tempo di sospensione.

Questo comando sospende per un certo tempo il task in cui viene eseguito. Per le unitá di tempo vedi all' inizio del capitolo TIMERS.

Esempio:

```
// o1 attivo per 100 mSec.
o1=1;
wait(100 MSEC);
o1=0;
```

```
int  every(time_t n);
```

Sospende per un certo tempo il task corrente per regolarizzare la durata di un loop.

n cadenza.

Questo comando sospende per un certo tempo il task in cui viene eseguito se non è passato almeno il tempo specificato dalla volta precedente.
Per le unitá di tempo vedi all' inizio del capitolo TIMERS.

Esempio:

```
// loop1.
for (i=1;i<100;i++)
{
  wait(90 MSEC);
  o1=1;
  wait(10 MSEC);
  o1=0;
  … altre cose
}
// loop2.
for (i=1;i<100;i++)
{
  every(100 MSEC);
  o1=1;
  wait(10 MSEC);
  o1=0;
  … altre cose
}
```

A differenza del loop 1, il loop 2 ha un tempo di ciclo di **esattamente 100 millisecondi**, in quanto every() attende il tempo mancante prima di proseguire.

```
time_t _time(time_t *dst);
float  _ftime(void);
```

Legge il system timer.

L' argomento dst è facoltativo. Se non usato mettere 0.

Questo comando legge la variabile unsigned long int system timer.
La prima versione (_time) è valida per PC e per S400.
La versione floating point, valida per S400, dá il valore del tempo con maggior risoluzione (microsecondo) ma riduce il tempo massimo da 2^31-1 a 2^24-1 millisecondi.
Puó servire per misurare il tempo di esecuzione di una routine, o per altri scopi.

Esempio:

```
// quanto è il tempo di esecuzione di SIN?
unsigned long int t1;
volatile float arg=1.2345,ris;
int i;
t1=_time(0);
for (i=0;i<5000;i++)
   ris=sin(arg);
t1=_time(0)-t1;
printf("sin(arg) dura %f mSec",(float)t1/5000.);
```

Nell' esempio abbiamo usato il prefisso volatile per ris per forzare l' esecuzione della funzione, bypassando l' ottimizzazione.
Si è usato un loop di 5000 per avere una sufficiente precisione.

OROLOGIO REALTIME

Il sistema operativo di Proteus 2007 gestisce un orologio-calendario utilizzabile per statistiche, controllo di produttivitá, o semplicemente per mostrare sullo schermo che ora è.

Il programma in C comunica con l' orologio tramite i comandi che vedremo nelle pagine successive, scambiando i dati mediante stringhe di otto caratteri, nel formato:

HH.MM:SS per l'ora
e
GG-MM-AA per il giorno (serie S400)
GG-MM-AAAA per il giorno (serie ARM)

Sono inoltre previste due funzioni (**get_oro e set_oro**) che possono leggere e scrivere il singolo byte dell'orologio. Queste due funzioni sono valide solo per programmi su S400 e non garantiscono la compatibilita'con successive versioni del compilatore.

```
char *get_time(void);
```

Legge l’ora

Questa funzione ritorna con il puntatore ad una stringa di otto caratteri ascii che contiene l’ ora attuale nel momento in cui è stata eseguita la funzione, con il formato **HH.MM:SS.**

Esempio:

```
// stampa l’ora in alto a sinistra sullo schermo
v_font(OFF8,1,1);
v_color(0,255);
v_llocate(1,1);
printf(“%s”,get_time());
```

```
char *get_date(void);
```

Legge la data odierna

Questa funzione ritorna con il puntatore ad una stringa di otto caratteri ascii (dieci caratteri nella versione ARM) che contiene la data attuale nel momento in cui è stata eseguita la funzione, con il formato **GG-MM-AA.**

Esempio:

```
// stampa la data e l'ora in alto
// a sinistra sullo schermo
v_font(OFF8,1,1);
v_color(0,255);
v_llocate(1,1);
printf("%s  %s",get_date(),get_time());
```

```
void set_time(char *s);
```

setta l'ora

s stringa contenente l' ora nel formato **HH.MM:SS**

Questa funzione aggiorna l' orologio con i valori contenuti nella stringa passata come parametro.
E' possibile aggiornare anche solo l' ora oppure ora e minuti, passando una stringa piú corta.
Per i caratteri di separazione tra ore, minuti e secondi possiamo utilizzare qualunque carattere ascii.

Esempio:

```
// setta ora e minuti
// a mezzogiorno e trenta
set_time("12-30");
```

```
void set_date(char *s);
```

setta la data attuale

s stringa contenente l' ora nel formato **GG.MM:AA (GG-MM-AAAA nella versione ARM)**

Questa funzione aggiorna l' orologio con i valori contenuti nella stringa passata come parametro.
E' possibile aggiornare anche solo il giorno oppure giorno e mese, passando una stringa piú corta.
Per i caratteri di separazione tra giorno, mese ed anno possiamo utilizzare qualunque carattere ascii.

Esempio:

```
// setta la data al 24 maggio
// a mezzogiorno e trenta
set_time("12-30");
set_date("24-05");
```

```
unsigned char get_oro(unsigned char c);
void set_oro(unsigned char c,unsigned
char val);
```

Sono due funzioni che possono leggere e scrivere il singolo byte dell'orologio. Queste due funzioni sono valide solo per programmi su S400 e non garantiscono la compatibilita'con successive versioni del compilatore.

C posizione del carattere nell' orologio dell' S400
V valore da scrivere.

I singoli bytes contengono valori esadecimali codificati in BCD

C	significato
0	**secondi**
1	**minuti**
2	**ore**
3	**giorno**
4	**mese**
5	**—**
6	**anno**

Esempio:

```
// legge e setta i secondi
unsigned char sec;
sec=get_oro(0);    // legge i secondi in BCD
sec-=(sec>>4)*6;   // li converte in numero
set_oro(0,0x35);   // setta i secondi a 35
```

DERIVATORI

```
unsigned char PulseOn(unsigned char
*p,unsigned char i);
unsigned char PulseOff(unsigned char
*p,unsigned char i);
```

Derivatori

p variabile unsigned char di servizio
i variabile unsigned char di ingresso

Queste funzioni danno come risultato il **valore 1** se la variabile di ingresso è **cambiata** rispetto alla volta precedente in cui la funzione è stata chiamata, altrimenti danno **valore 0**.
PulseOn rivela cambiamenti **da 0 a 1** della variabile, mentre **PulseOff** rivela cambiamenti **da 1 a 0**.
La variabile p, che deve essere dichiarata dal programmatore ed opportunamente inizializzata ad un valore iniziale, serve a memorizzare lo stato precedente e deve essere una variabile diversa per ingressi diversi.
Come ingresso si intende una qualsiasi variabile di tipo byte o bit, e non necessariamente un ingresso fisico del PLC.

Esempio:

```
// contatore fronti positivi ingresso 2
int contai2(void)
{
   static unsigned char h2=0;
   static int conta=0;
   conta+=PulseOn(&h2,i2); // conta fronti pos. I2
   return conta;
}
```

Se chiamata abbastanza frequentemente rivela i fronti di salita di i2.

RITARDATORI

```
unsigned char Delay(unsigned long int
*h,int n,unsigned char i);
unsigned char DelayUp(unsigned long int
*h,int n,unsigned char i);
unsigned char DelayDown(unsigned long int
*h,int n,unsigned char i);
```

Ritardatori di segnale

h variabile di servizio unsigned long int
n tempo di ritardo in millisecondi
i variabile da ritardare

Queste funzioni ritardano di un tempo costante il valore di una variabile.
Delay è un ritardatore puro.
DelayUp ritarda la transizione da 0 ad 1
DelayDown ritarda la transizione da 1 a 0

Di norma sono usate nel task di PLC, ma possono essere usate anche altrove, purché vengano chiamate regolarmente ed abbastanza frequentemente. **NON possono essere usate** se il ciclo di PLC non è attivo (se è stato dichiarato **`#define NO_PLC`**).
La variabile p, che deve essere dichiarata dal programmatore, serve a memorizzare lo stato interno e deve essere una variabile diversa per ingressi diversi.

Esempio:

```
// conta ingresso 2 (solo impulsi > 230 mSec)
int contai2(void) {
   static unsigned char h2=0;
   static unsigned long int tt2=0;
   static int conta=0;
   char tmp;
   tmp=DelayUp(&tt2,230,i2); // ritarda i2 up
   conta+=PulseOn(&h2,tmp ); // conta fronti pos. I2
   return conta; }
```

MONOSTABILI

```
unsigned char MonostableUp(unsigned long
int *h,int n,unsigned char i);
unsigned char MonostableDown(unsigned
long int *h,int n,unsigned char i);
```

monostabili

h variabile di servizio unsigned long int
n tempo di ritardo in millisecondi
i variabile da ritardare

MonostableUp dá il valore 1 quando la variabile i va ad 1, dá il valore 0 dopo il tempo indicato che la variabile i è ritornata a 0.
dá il valore 0 quando la variabile i va ad 0, dá il valore 1 dopo il tempo indicato che la variabile i è ritornata a 1.

Di norma sono usate nel task di PLC, ma possono essere usate anche altrove, purché vengano chiamate regolarmente ed abbastanza frequentemente. **NON possono essere usate** se il ciclo di PLC non è attivo (se è stato dichiarato **#define NO_PLC**).
La variabile p, che deve essere dichiarata dal programmatore, serve a memorizzare lo stato interno e deve essere una variabile diversa per ingressi diversi.

Esempio:

```
// o2=1 per 1 secondo su fronte salita di i2
// a condizione che i2 duri almeno 200 mSec.
void ripeti_i2(void) {
   char temp;
   static unsigned char dd2=0;
   static unsigned long int tt1=0;
   static unsigned long int tt2=0;
   tmp=DelayUp(&tt1,200,i2);          // ritardatore
   temp=PulseOn(&dd2,temp);           // derivatore
   o2=MonostableUp(&tt2,1 SEC,temp); // monostabile
}
```

FUNZIONI GRAFICHE

Le funzioni qui descritte permettono di definire la risoluzione del CRT, di inizializzarlo e di scrivere e disegnare direttamente sul Canvas dello schermo.
Lo schermo è visto come una matrice XY di punti.
Il punto di coordinate 0,0 è quello in alto a sinistra.

Caso S400 e PC:
La larghezza del canvas dello schermo è di 1024 pixels.
L' altezza dipende dalla quantitá di memoria video istallata.
In genere:
1024 pixels per CRT 320x240
2048 pixels per CRT 640x480
3072 pixels per CRT 800x600
5120 pixels per CRT 1024x768
L' area visibile è sempre quella in alto a sinistra.
La restante porzione è utilizzata per operazioni di copia, per le animazioni, come buffer per i bitmap e viene gestita dal sistema operativo.

Caso ARM:
La larghezza del canvas dello schermo è pari alla risoluzione orizzontaledel display.
L' altezza del canvas dello schermo è pari alla risoluzione verticale del display.
Vengono gestiti due canvas, ed in ogni momento possiamo decidere quale canvas visualizzare e su quale scrivere.
Questo puó risultare particolarmente utile per animazioni senza flickering (tecnica del doppio buffer. Per ulteriori spiegazioni vedi esempio DX nella sezione ESEMPI ARM)

Se piú task accedono al video, occorre che le attivitá siano gestite da un semaforo, onde evitare che i vari comandi si mescolino, dando luogo ad errori di visualizzazione. A tale scopo si possono usare i comandi di LOCK ed UNLOCK.
Salvo rari casi, come per esempio se dobbiamo tracciare dei grafici o costruire un nuovo componente, tali funzioni sono gestite internamente dal sistema operativo, e il programmatore, usando Proteus con la programmazione ad oggetti, non è costretto ad utilizzare direttamente le funzioni grafiche.

```
void ioinit(int priority);
void crt(int dx,int dy,int displ_cieco);
void display(int dx,int dy,int
displ_cieco);
void crt_res(int dx,int dy);
void v_send(char c1);
void v_stop(void);
void v_paint_ram(unsigned long int
ram,const unsigned char *s);
void v_copy_from_ram(unsigned long int
ram,int step,int xd,int yd,int dx,int
dy);
void v_paint(const unsigned char *s);
void v_copy(int xs,int ys,int xd,int
yd,int dx,int dy);
void v_fill_mode(unsigned char mode,int
level,int color0,int color1,int color2);
```

Sono funzioni interne, valide solo per la versione V5/V10 del compilatore, non utilizzabili direttamente dal programmatore.

void **v_cls**(void);

Cancella l' area visibile dello schermo.

Questo comando colora lo schermo con il colore del fondo (vedi v_color)

Esempio:

```
// cancella lo schermo colorandolo di blu
v_color(BLUE,YELLOW);
v_cls();
```

```
void v_locate(int x,int y);
```

Posiziona il cursore.

Questo comando posiziona il cursore a coordinate stabilite.

Esempio:

```
// ____________________-
// in start.c
// ____________________-
v_color(BLUE,YELLOW);
v_cls();
v_font(OFF8,1,1);
v_color(TRASP,RED);
v_locate(12,12);
v_print("prova");
v_color(TRASP,YELLOW);
v_locate(13,13);
v_print("prova");
while(1);
```

```
void v_llocate(int riga,int colonna);
```

Posiziona il cursore.

Questo comando posiziona il cursore a coordinate stabilite. A differenza di v_locate, le coordinate sono invertite, e sono espresse in caratteri (multipli di 8). Ad es. v_llocate(2,5) equivale a v_locate(40,16).

Esempio:

```
// ____________________
// in start.c
// ____________________
v_color(BLUE,YELLOW);
v_cls();
v_font(OFF8,1,1);
v_color(TRASP,RED);
v_llocate(2,2);
v_print("prova");
v_color(TRASP,YELLOW);
v_llocate(3,2);
v_print("prova");
while(1);
```

```
void _font(int font_num,int
scala_x,int scala_y);
```

Seleziona il font di scrittura e la scala.

Questo comando seleziona il font di scrittura e il formato di zoom x e y.
I fonts standard utilizzabili direttamente sono:

off8 (OFF8)	0	abcdABCD01234
off16 (OFF16)	1	abcdABCD01234
off24 (OFF24)	2	abcdABCD01234
tim8 (TIM8)	3	abcdABCD01234
tim16 (TIM16)	4	abcdABCD01234
tim24 (TIM24)	5	abcdABCD01234
tim32 (TIM32)	6	abcdABCD01234
MS_Sans_Serif_8	7	abcdABCD01234
MS_Sans_Serif_10	8	abcdABCD01234
MS_Sans_Serif_12	9	abcdABCD01234
MS_Sans_Serif_14	10	abcdABCD01234
MS_Sans_Serif_18	11	abcdABCD01234
MS_Sans_Serif_24	12	abcdABCD01234
Arial_Black_8	13	*abcdABCD01234*
Arial_Black_10	14	*abcdABCD01234*
Arial_Black_12	15	*abcdABCD01234*
Arial_Black_14	16	*abcdABCD01234*
Arial_Black_18	17	*abcdABCD01234*
Arial_Black_24	18	*abcdABCD01234*
Courier_New_8_bold	19	abcdABCD01234
Courier_New_10_bold	20	abcdABCD01234
Courier_New_12_bold	21	abcdABCD01234
Courier_New_14_bold	22	abcdABCD01234
Courier_New_18_bold	23	abcdABCD01234
Courier_New_24_bold	24	abcdABCD01234

I fonts offxx e timxx sono scalabili, gli altri no, per cui per i fonts successivi a 6 gli ultimi due parametri non hanno influenza.

Esempio:

```
// ____________________-
// in start.c
// ____________________-
v_color(BLUE,YELLOW);
v_cls();
_font(off8,1,1);
v_color(TRASP,RED);
v_llocate(2,2);
v_print("prova");
v_color(TRASP,YELLOW);
v_llocate(3,2);
v_print("prova");
while(1);
```

```
void v_font(int font_num,int scala_x,int
scala_y);
void v_setfont(int fontn,int fontx,int
fonty);
```

Come font (vedi pagina precedente)

Questi comandi esistono per ragioni storiche per compatibilitá con le versioni precedenti. È meglio utilizzare _font(..)

```
void v_color(int bg,int fg);
void _color(int fg,int bg);
```

Imposta il colore del fondo e del carattere.

bg colore del background (del fondo)
fg colore del carattere

Questo comando setta i colori da utilizzare per scrivere su video.
I colori sono numeri da 0 a 255, con **palette fissa a 256 colori RGB 3-3-2**.
Esistono degli mnemonici per i colori fondamentali:

BLACK	0
DARKBLUE	2
DARKRED	128
DARKMAGENTA	130
DARKGREEN	16
DARKCYAN	18
DARKYELLOW	144
DARKGRAY	73
GRAY	146
BLUE	3
RED	224
MAGENTA	227
GREEN	28
CYAN	31
YELLOW	252
WHITE	255
TRASP	**4 (trasparente)**
COMPL	**32 (complementa il colore attuale)**

Esempio:

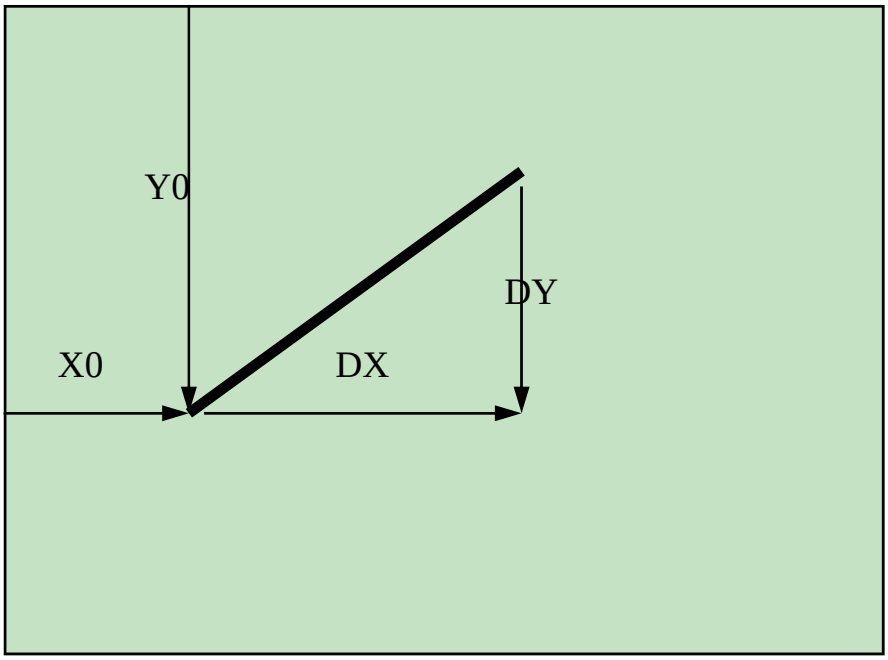
```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
```

```
void v_line(int x0,int y0,int dx,int dy);
void v_linec(int x0,int y0,int dx,int
dy,int color);
```

Disegna una linea.

x0 ascissa del punto di inizio
y0 ordinata del punto di inizio
dx ascissa del punto finale meno ascissa del punto di inizio
dy ordinata del punto finale meno ordinata del punto di inizio
color colore della linea (in v_line il colore è settato con v_color)

Questo comando disegna una linea sullo schermo.



Esempio:

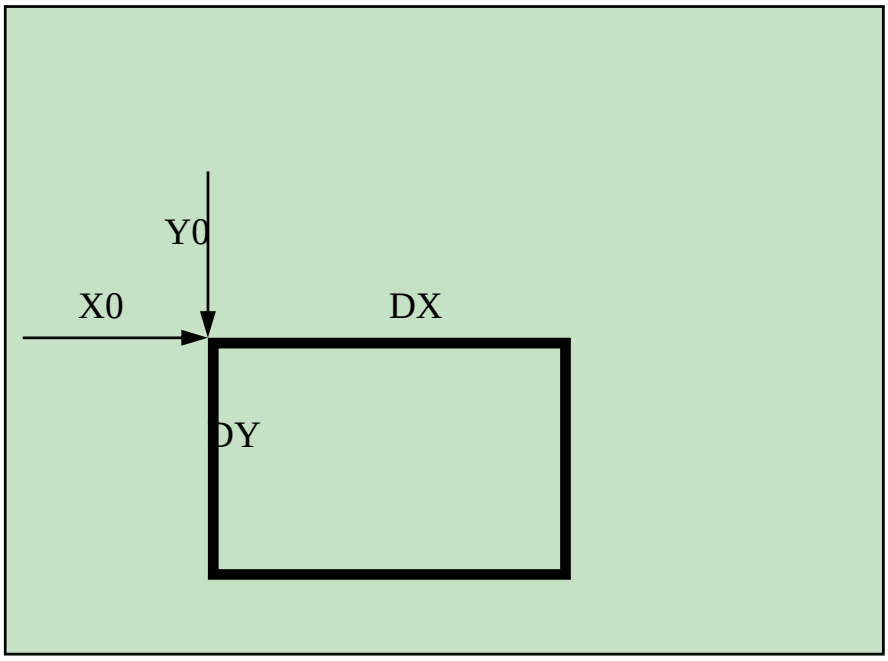
```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
v_linec(10,20,50,80,RED);
while(1);
```

```
void v_rect(int x0,int y0,int dx,int dy);
void v_rectc(int x0,int y0,int dx,int
dy,int color);
```

Disegna il contorno di un rettangolo vuoto.

x0 ascissa del punto di inizio
y0 ordinata del punto di inizio
dx larghezza
y0 altezza
color colore del bordo (in v_rect il colore è settato con v_color)

.



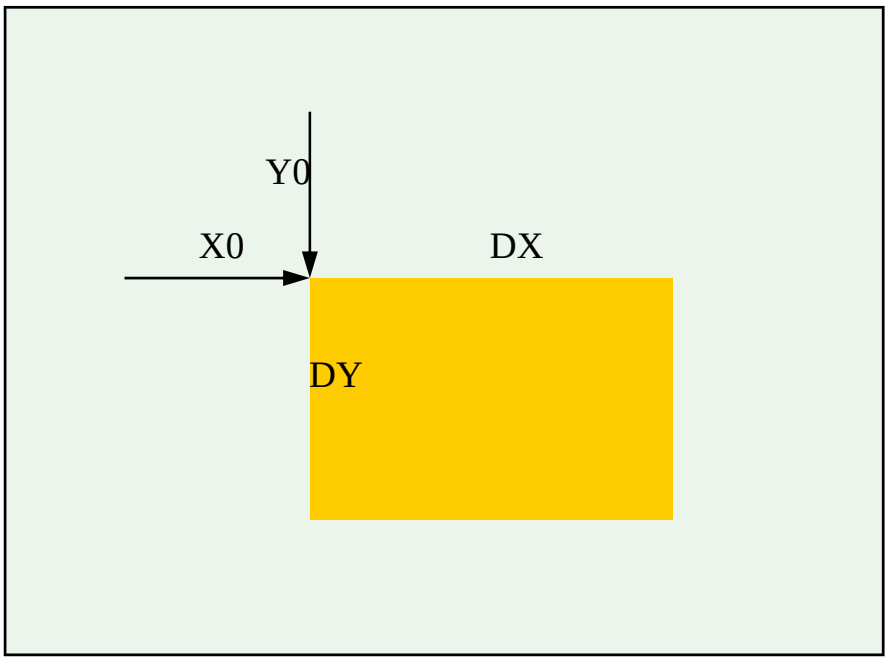
Esempio:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
v_rectc(10,20,50,80,RED);
while(1);
```

```
void v_rectfill(int x0,int y0,int dx,int dy);
void v_rectfillc(int x0,int y0,int dx,int dy,int color);
```

Disegna un rettangolo pieno.

x0 ascissa del punto di inizio
y0 ordinata del punto di inizio
dx larghezza
y0 altezza
color colore del rettangolo (in v_rectfill il colore è settato con v_color)



Esempio:

```
// ---------------------
// in start.c
// ---------------------
v_color(BLUE,YELLOW);
v_cls();
v_rectc(10,20,50,80,RED);
while(1);
```

```
void v_printchar(unsigned char c);
```

Stampa un carattere

c carattere da stampare

Stampa un carattere ascii alla posizione attuale del cursore. Il cursore viene incrementato della lunghezza della stringa.

Esempio:

```
// ____________________-
// in start.c
// ____________________-
v_color(BLUE,YELLOW);
v_cls();
font(off8,1,1);
v_color(TRASP,RED);
v_llocate(2,2);
v_printchar(49);  //stampa la lettera A
v_printchar('b'); //stampa la lettera B
while(1);
```

```
void v_print(char *s);
```

Stampa una stringa

s stringa da stampare

Stampa una stringa che termina con il carattere ascii 0 alla posizione attuale del cursore. Il cursore viene incrementato della lunghezza della stringa.
Se la stringa contiene un carattere ascii 13 (return) il cursore va a capo allineato all' inizio della stringa.

Esempio:

```
// ----------------------
// in start.c
// ----------------------
v_color(BLUE,YELLOW);
v_cls();
font(off8,1,1);
v_color(TRASP,RED);
v_llocate(2,2);
v_print("prova 1\n");
v_print("prova 2\n");
while(1);
```

```
void lock(void);
void unlock(void);
```

Riserva e libera la scrittura su schermo

Il comando lock riserva l' accesso alla scrittura su video al task corrente. Il comando unlock libera l' accesso precedentemente riservato con lock.

Se piú task accedono al video, occorre che le attivitá siano gestite da un semaforo, onde evitare che i vari comandi si mescolino, dando luogo ad errori di visualizzazione. A tale scopo si possono usare i comandi di lock ed unlock.

Esempio:

```
// ---------------------
// in task1
// ---------------------
lock();
v_color(BLUE,YELLOW);
font(off8,1,1);
v_llocate(2,2);
v_print("task 1");
unlock();
// ---------------------
// in task2
// ---------------------
lock();
v_color(RED,GREEN);
font(off16,1,1);
v_llocate(5,2);
v_print("task 2");
unlock();
```

TASTIERA

I progetti sviluppati con Proteus 2007 possono girare su hardware diversi, quali :

S400 con touch_screen
S400 con touch_screen e tastiera
S400 con tastiera
ARM con touch_screen
ARM con touch_screen e tastiera
ARM con tastiera
Console con LCD 2 x 20.
PC

Se si utilizzano gli oggetti ed i componenti standard il programma è indipendente dall' hardware, tuttavia in casi particolari (programma su console LCD o utilizzo particolare della tastiera) è possibile accedere direttamente alle funzioni di tastiera.
Occorre tener presente che utilizzando anche una sola delle funzioni qui riportate, il programma sará legato ad un hardware particolare e dovrá essere modificato per funzionare su altri hardware.

```
void v_tik(unsigned char x0);
```
(solo v5-v10-ARM con touch screen)

Il display emette un tick.

x0 modo:
- 0 tick touch screen hardware
- 1 tick touch screen da programma
- n>1 emette un tick di n millisecondi

Esempio:

```
// in una pagina…
void Event(bitmap1_onmousedown)()
{
   v_tik(2);
}
```

In Proteus un oggetto Bitmap dá un tick, se toccato, solo se cambia di immagine. Nell' esempio di forza il tick anche se l' immagine resta la stessa.

```
void v_spot(unsigned long int leds);
```
(solo console S400 con tastiera a leds)

Settaggio dei leds dei tasti.

ledsconfigurazione on-off dei leds:

Leds è una variabile di 32 bits. Ad ogni bit è associato uno dei 32 leds associati ai tasti della tastiera. Se il bit corrispondente è ad 1 il led è acceso, e viceversa.

Esempio:

```
// in COMMON.H…
unsigned long int ledimage=0;

// in una pagina…
void Event(bitmap1_onmousedown)()
{
   v_spot(ledimage |(1<<14));
}
```

In questo esempio accendiamo il led 15 quando tocchiamo l'oggetto bitmap bitmap1.

`void keyboard(const char* keyboar_keys);`

(console S400 e ARM con tastiera)

Settaggio della configurazione dei tasti **della tastiera hardware**.

Keyboar_keys tabella dei valori ascii associati ai tasti.

La tastiera, quando premuta, dá un codice (scancode) che individua il tasto che è stato premuto. Con questa funzione associamo il valore ascii che vogliamo ad ogni tasto.
Per la tastiera Touch-Screen vi sono altre funzioni.
Per default, la tastiera dá i seguenti 63 scan codes:

code	ascii	code	ascii	code	ascii	code	ascii
1	0	2	1	3	2	4	3
5	4	6	5	7	6	8	7
9	8	10	9	11	F1	12	F2
13	F3	14	F4	15	F5	16	F6
17	F7	18	F8	19	F9	20	F10
21	u (up)	22	d (down)	23	l (left)	24	r (right)
25	x	26	y (yes)	27	n (no)	28	b (back)
29	h	30	-	31	.	32	e (enter)
33	A	34	B	35	C	36	D
37	E	38	F	39	G	40	H
41	I	42	J	43	K	44	L
45	M	46	N	47	O	48	P
49	Q	50	R	51	S	52	T
53	U	54	V	55	W	56	X
57	Y	58	Z	59	+	60	*
61	=	62	/	63	blank		

Non tutti i tipi di tastiera hanno il set completo di tasti mostrato nella tabella sopra riportata, ma in ogni caso, per i tasti presenti, il codice è lo stesso per tutte le tastiere che hanno il tasto corrispondente.
Ad esempio, gli hardware che hanno solo tastiera numerica, non avranno alcun tasto corrispondente ad A o a B, ma il tasto =, se presente, corrisponderà sempre al codice di tastiera 61.
In tale modo è possibile creare del codice portabile.

Esempio:

```
// in COMMON.H…
```

```
const char tast1[]={0,
      '0','1','2','3','4','5','6','7','8','9',
      200,201,202,203,204,205,206,207,208,209,
      'u','d','l','r','x','y','n','b','h','-',
      '.','e','A','B','C','D','E','F','G','H',
      'I','J','K','L','M','N','O','P','Q','R',
      'S','T','U','V','W','X','Y','Z','+','*',
      '=','/',' '};

// in start.c…
keyboard(tast1);
```

In questo esempio associamo la tastiera standard ai valori standard dei tasti.

```
void keyboard(const char* keyboar_keys);
```
(v5/v10/ARM con touch screen)

Settaggio della configurazione dei tasti **della tastiera sul touch screen**.

Keyboar_keys tabella dei valori ascii associati ai tasti.

Nel caso di tastiera su touch screen, la funzione keyboard dichiara i caratteri da associare alla tastiera, scandita da sinistra verso destra e dall' alto verso il basso. In questo caso non esiste una mappatura standard per i tasti essendo questi definiti dal comando **keyon**.

Per ulteriori spiegazioni e per esempi si rimanda ai comandi **keyon** e **keyshow**.

```
void key_show(int lin,int col,int nx,int
ny,char *show_keys,const char* keyboar);

void key_msg(char *msg);

void keyoff(void);
```
(solo v5/v10/ARM con touch screen)

key_show Abilita la visualizzazione della tastiera sul touch screen..
key_msg Visualizza un messaggio sul display della tastiera appena abilitata da key_show.
keyoff fa scomparire la tastiera dallo schermo.

lin linea di inizio della tastiera (angolo in alto a sinistra).
col colonna di inizio della tastiera (angolo in alto a sinistra).
nx numero di tasti in orizzontale.
ny numero di tasti in verticale.
show_keys stringa dei caratteri da scrivere sui tasti (da sinistra verso destra, dall' alto verso il basso)
keyboar tabella dei valori ascii associati ai tasti.
msg stringa aascii del messaggio da visualizzare.

Key_show mostra la tastiera in sovrimpressione, definisce i caratteri da sovrascrivere ai tasti e i caratteri ascii corrispondenti. Le dimensioni in pixel dei tasti dipendono dalla risoluzione del monitor.
Key_msg mostra un messaggio che in genere spiega ció che ci si attende che venga battuto sulla tastiera. Tale messaggio scompare appena si batte un tasto.
Alla fine, keyoff fa scomparire la tastiera, ripristinando gli oggetti sottostanti.
Queste funzioni sono chiamate automaticamente dagli oggetti Edit, e vengono usate solo in casi particolari. Ove possibile, evitare di usarle direttamente, perché non sono compatibili con la versione PC.

Queste funzioni vengono sempre utilizzate congiuntamente, e nell' ordine sopra visto.

Esempio:

```
void bitmap1_onmouseup()
{
   const  char tastiera_1[]={"1234567890.-uzxe"};
   static char strinput[8];
   long int chiave=0;
   key_show(2,2,4,4,"1234567890.-u xe",tastiera_1);
   key_msg("Password ?");
   t_input(strinput,8);
   chiave=atol(strinput);
   if ((chiave==12345) || (chiave==23456))
   {
     page_end=1;
     page_num=_page1;
   }
   keyoff();
}
```

```
void key_wait(void);

void key_release(void);
```
(console con tastiera)

Attendono rispettivamente la **pressione** ed il **rilascio** di un tasto.

```
unsigned char t_inke(void);
unsigned char t_inkey(void);
unsigned char t_inkey(char mode);
```

Queste due funzioni danno il valore ascii del tasto premuto.
La prima forma (**t_inke**) dá il valore **istantaneo** del tasto premuto.
La seconda forma (**t_inkey**) equivale a t_inke nella versione S400.

Nella versione ARM t_inkey ha l' argomento mode:

Se mode=0 oppure se l' hardware ha la tastiera, equivale a t_inke.

Se l' hardware ha il touch screen, visualizza una tastiera, il cui layout dipende dal valore di mode:

mode= _**LOGIC**_ tastiera Yes/No/Esc
mode= _**FUNCTION**_ tastiera funzioniF1-F5/Esc
mode= _**NUMBER**_ tastiera numerica
mode= _**ALPHA**_ tastiera alfanumerica

La tastiera scompare automaticamente dopo che si è premuto un tasto.

Esempio:

```
void Event(onentry)()
{
   tasto=t_inke();
   if (tasto==202)
   {
      pagina=_page2;
      page_end=1;
   }
   if (tasto==203)
   {
      pagina=3;
      page_end=1;
   }
}
```

```
void t_input(char * string,short int
max);
```

string puntatore alla stringa da riempire
max numero massimo di caratteri da leggere.

Questa funzione legge da tastiera una stringa di caratteri e la mette nella variabile specificata.

Esempio:

```
void bitmap1_onmouseup()
{
   const  char tastiera_1[]={"1234567890.-uzxe"};
   static char strinput[8];
   long int chiave=0;
   key_show(2,2,4,4,"1234567890.-u xe",tastiera_1);
   key_msg("Password ?");
   t_input(strinput,8);
   chiave=atol(strinput);
   if ((chiave==12345) || (chiave==23456))
   {
     page_end=1;
     page_num=_page1;
   }
   keyoff();
}
```

PORTE SERIALI

Il PLC S400 dispone di 4 porte seriali asincrone:

COM1	10	…	625000 baud
COM2	4800	…	625000 baud
COM3	57600	…	5000000 baud
COM4	57600	…	5000000 baud

Le **baud rates poss**ibili sono solo quelle date dalla formula:

BAUD_RATE_MAX/N

con N intero

Se il PLC ha una console (monitor o lcd) la porta COM3 è utilizzata dal sistema operativo per la comunicazione con la console. Per apparecchiature cieche COM3 è utilizzabile dal programma.

I PLC della serie ARM dispongono di 2 porte seriali asincrone:

COM1	70	…	4500000 baud
COM2	70	…	4500000 baud

E' possibile per i PLC di questa serie utilizzare **qualunque baud rate**, disponendo l' UART di prescaler frazionario.

Il programmatore puó utilizzare la comunicazione via seriale a livello di singolo carattere (modo 0), di blocchi di dati (syel mode 1 e 2), o definendo un proprio protocollo di comunicazione.

In modo 0 (default) il sistema operativo gestisce le seriali ad interrupt con dei buffers circolari fifo di 256 bytes.

In syel mode 1 vengono gestiti messaggi lunghi al massimo 256 bytes. Se è definito un proprio protocollo, il dato ricevuto viene passato direttamente alla subroutine definita dal programmatore.

In syel mode 2 vengono gestiti messaggi di lunghezza massima definibile. Il protocollo syel mode 2 è avviato mediante il comando **pclib_open.**

pclib_open(COM *com,int baudrate,int max_buffer_len)

Occorre specificare il numero della centrale (il plc stesso) e della periferica (l' apparecchiatura remota, in genere un PC)

A questo punto è possibile definire i buffers dei messaggi di trasmissione e ricezione.

Esempio:

```
pclib_open(COM2,625000,4096);
COM2->centr=1;
COM2->perif=15;
_rx(COM2,1,(char *)&buffr1,sizeof(buffr1),subrx1);
_rx(COM2,2,(char *)&buffr2,sizeof(buffr2),subrx2);
_tx(COM2,1,(char *)&bufft1,sizeof(bufft1),subtx1);
_tx(COM2,2,(char *)&bufft2,sizeof(bufft2),subtx2);
```

Ogni porta COM è associata ad una struttura

```
typedef struct COM
{
  unsigned char open;     //0   flaag COM aperta
  unsigned char mode;     //1   tipo di protocollo
  long int baud;          //2   baud rate
  void(*subrx)();         //6   puntatore subroutine associata a rx
  void(*subtx)();         //10  puntatore subroutine associata a tx
  unsigned int  rx0;      //14  puntatore buffer circolare inizio rx
  unsigned int  rx1;      //16  puntatore buffer circolare fine rx
  unsigned int  tx0;      //18  puntatore buffer circolare inizio tx
  unsigned int  tx1;      //20  puntatore buffer circolare fine tx
  unsigned char *rxbuff;  //22  buffer di ricezione
  unsigned char *txbuff;  //26  buffer di trasmissione
  unsigned char rxstatus; //30  stato rx
  unsigned char txstatus; //31  stato tx
  unsigned char rxchk;    //32  checksum rx
  unsigned char txchk;    //33  checksun tx
  unsigned int  rxsize;   //34  n. di caratteri ricevuti
  unsigned int  txsize;   //36  n. di caratteri in attesa di trasm.
  unsigned char centr;    //38  n. della centrale (se stessa)
  unsigned char perif;    //39  n. periferica (destinatario)
  unsigned char echo;     //40  1= disabilita ricez. Durante trasm.
  unsigned char txrun;    //41  1=tx is running
  void(*protocol)(struct COM*,int); //42 custom interrupt subroutine
}COM;
```

In genere il programmatore non accede direttamente ai dati di questa struttura, salvo se usa un suo proprio protocollo.

Esempio:

Questo è il mio protocollo, che viene chiamato per ogni dato ricevuto, e gli viene passato il puntatore della com e il dato ricevuto.
Nell' esempio incremento il contatore pippo ogni volta che ricevo un carattere ascii A da COM2 e risponde con il carattere C ogni volta che riceve un carattere B.

In common.h

```
int pippo=0;
void com_int(COM * this_com,int dato)
{
   if (dato=='A') pippo++;
   if (dato=='B') com_tx(this_com,'C');
}
```

In start.c

```
com_open(COM2,57600);   //apro COM2 a 57600 baud
com_disable(COM2);      //la disabilito
COM2->protocol=com_int; //assegno il mio protocollo
com_enable(COM2);       //la riabilito
```

Nell' esempio, si fa accesso direttamente alla struttura della COM per assegnare l' indirizzo della routine di gestione del protocollo.
Il protocollo cosí definito, che viene eseguito a livello di routine di interrupt, è quello di ricezione.
Il protocollo di trasmissione è in ogni caso una funzione che invia i dati mediante delle istruzioni com_tx(COMn,dato).

```
int  com_open(struct COM *porta,long int baud);
```

Apre una porta COM.

porta nome della porta seriale (COM1,COM2,COM3 o COM4)
baud baud rate (la baud rate min. e max. dipedono dalla com)

COM1	min. 10	max. 625000 baud
COM2	min. 4800	max. 625000 baud
COM3	min. 57600	max. 5000000 baud
COM4	min. 57600	max. 5000000 baud

La baud rate massima è inoltre limitata dal tipo di interfaccia (RS-232, fibra ottica..) e dalla lunghezza del collegamento.
Per ulteriori informazioni a riguardo vedere le specifiche del PLC S400.

Esempio:

In start.c

```
com_open(COM1,57600);     //apro COM1 a 57600 baud
com_open(COM2,625000);    //apro COM2 a 625 KBaud
```

```
int  com_close(struct COM *porta);
```

Chiude una porta COM.

porta nome della porta seriale (COM1,COM2,COM3 o COM4)

In genere, una volta aperta una com, non vi è alcun motivo per chiuderla, salvo se vogliamo riaprirla con modalità diversa.

Esempio:

```
com_close(COM1);    //chiudo COM1
com_close(COM2);    //chiudo COM2
```

```
int  com_enable(struct COM *com);
int  com_disable(struct COM *com);
```

Abilita e disabilita una porta COM.

porta nome della porta seriale (COM1,COM2,COM3 o COM4)

Una porta seriale, una volta aperta, è automaticamente abilitata. Qualora dobbiamo cambiarne direttamente gli attributi, andando a modificare la struttura associata, è buona norma disabilitare la COM durante l' accesso alla struttura relativa, altrimenti potrebbero aversi risultati imprevedibili se vengono ricevuti dei caratteri durante la modifica dei parametri.

Esempio:

In start.c

```
com_open(COM2,57600);   //apro COM2 a 57600 baud
com_disable(COM2);      //la disabilito
COM2->protocol=com_int; //assegno il mio protocollo
com_enable(COM2);       //la riabilito
```

```
int  protocol_mode(struct COM *porta,char mode);
```

Assegna un protocollo alla COM.

porta nome della porta seriale (COM1,COM2,COM3 o COM4)
mode tipo di protocollo.

Attualmente sono disponibili tre tipi di protocollo:

0 protocollo a livello di singolo carattere, con buffer di ricezione (è il modo di default all' apertura della COM).
1 protocollo a livello di blocco (protocollo Syel block mode 1)
2 protocollo a livello di funzioni (protocollo Syel block mode 2), che si setta peró mediante il comando **pclib_open**.

Se il protocol mode è 1 e' possibile assegnare una routine di rx con il comando on_rx che verrá chiamata una volta ricevuto il blocco di dati, ed una routine di tx con il comando on_tx, che verrá chiamata prima dell' invio dei dati.

Esempio:

In start.c

```
com_open(COM2,57600);   //apro COM2 a 57600 baud
protocol_mode(COM2,1);  //in block mode 1
on_rx(COM2,my_rx_sub);  //assegno la subroutine di rx
```

Per ulteriori delucidazioni riguardo il protocollo mode 2 vedi l' esempio modo2_pclib nella directory ESEMPI_ARM

```
int  com_speed(struct COM *porta,long int baud);
```

Cambia la baud rate di una porta COM aperta.

porta nome della porta seriale (COM1,COM2,COM3 o COM4)
baud nuova baud rate.

Una porta seriale, una volta aperta, lavora alla baud rate specificata con il comando di apertura. Con il comando com_speed è possibile cambiare tale baud rate al volo, senza chiudere né disabilitare la COM. Se c' è un carattere in corso di ricezione, questo sará perduto.

Esempio:

In start.c

```
com_open(COM2,57600);   //apro COM2 a 57600 baud
com_speed(COM2,625000); //porta la baud rate a 625K
```

```
int  com_tx_empty(struct COM *porta);
```

Ritorna 1 se il buffer di trasmissione è vuoto (solo protocol mode 0)

```
int  com_tx_full(struct COM *porta);
```

Ritorna 1 se il buffer di trasmissione è pieno (solo protocol mode 0)

```
int  com_tx_size(struct COM *porta);
```

Ritorna con il numero di caratteri ancora disponibili nel buffer di trasmissione (solo protocol mode 0)

```
int  com_rx_empty(struct COM *porta);
```

Ritorna 1 se il buffer di ricezione è vuoto (solo protocol mode 0)

```
int  com_rx_size(struct COM *porta);
```

Ritorna con il numero di caratteri ancora non letti nel buffer di ricezione (solo protocol mode 0)

```
int  rxchars(struct COM *porta);
```

Ritorna con il numero di caratteri ricevuti (solo protocol mode 0)

Esempio:

```
com_open(COM2,57600);   //apro COM2 a 57600 baud
// ...
While(!com_rx_empty(COM2))
{
   msg[punta++]=com_rx(COM2);
}
```

```
int  com_tx(struct COM *porta,unsigned
char c);
```

Invia un carattere tramite una porta COM (solo protocol mode 0).

porta nome della porta seriale (COM1,COM2,COM3 o COM4)
c carattere da inviare.

Esempio:

```
com_open(COM2,57600);   //apro COM2 a 57600 baud
com_tx(COM2,'A');       //invio il carattere A
```

```
int  com_rx(struct COM *porta);
```

riceve un carattere tramite una porta COM (solo protocol mode 0).

porta nome della porta seriale (COM1,COM2,COM3 o COM4)

Esempio:

```
com_open(COM2,57600);   //apro COM2 a 57600 baud
// ...
While(!com_rx_empty(COM2))
{
   msg[punta++]=com_rx(COM2);
}
```

```
int  com_txs(struct COM *porta,unsigned
char *c);
int  com_txsl(struct COM *porta,unsigned
char *c,int len);
```

Invia una stringa di caratteri tramite una porta COM (solo protocol mode 0).

porta nome della porta seriale (COM1,COM2,COM3 o COM4)
c puntatore alla stringa di caratteri da inviare.
len numero di caratteri.

com_txs invia una stringa di caratteri **fino al primo carattere nullo** della stringa (modalitá text).
com_txsl invia **il numero specificato di caratteri**, compresi i caratteri nulli (modalitá binary).

Esempio:

```
char msg1[16];
Char msg2[]={0,0,0,3,32,0x44,0x55,3};
com_open(COM2,57600);   //apro COM2 a 57600 baud
strcpy(msg1,"Ciao !\n");
com_txs(COM2,msg1);     //invio il messaggio 1
com_txsl(COM2,msg2,8);  //invio il messaggio 2
```

```
int  bload(struct COM *porta,void
*buffer,int len);
```

Legge un blocco di dati

porta nome della porta seriale (COM1,COM2,COM3 o COM4)
buffer stringa in cui copiare il messaggio.
len numero di caratteri massimi da leggere.

Questo comando, valido solo in **BLOCK MODE 1**, legge il messaggio ricevuto sulla Com specificata, riabilitando la Com stessa a ricevere nuovi messaggi.
Se nessun messaggio è arrivato, ritorna con codice di errore –1.
Se c' e' un messaggio valido, ritorna con codice di errore 0.
Normalmente questo comando si trova nella routine di on_rx della Com, in tal caso siamo sicuri che verrà chiamata solo se c'è un messaggio da leggere.
Se la lunghezza del messaggio supera la lunghezza specificata nel campo len, solo i primi len bytes del messaggio verranno copiati nel buffer.
Un messaggio è considerato valido se ha il giusto header, se è destinato a noi (id=COMx->centr) e se il checksum è corretto.
Per il calcolo del checksum è preso in considerazione l' intero messaggio e non solo il numero di bytes che vogliamo leggere.

Esempio:

In common.h

```
char msg1[16];
void subrx2(void)
{
   bload(COM2,msg1,16);
}
```

In start.c

```
com_open(COM2,57600);
protocol_mode(COM2,1);
COM2->centr=4;
onrx(COM2,subrx2);
```

```
int  bsave(struct COM *porta,void
*buffer,int len);
```

Invia un blocco di dati

porta nome della porta seriale (COM1,COM2,COM3 o COM4)
buffer stringa in cui si trova il messaggio.
len numero di caratteri da inviare.

Questo comando, valido solo in **BLOCK MODE 1**, Invia il messaggio sulla Com specificata, alla periferica il cui indirizzo è specificato da COMx->perif.

Esempio:

```
char buffer1[16];
int num=1;
com_open(COM2,57600);
protocol_mode(COM2,1);
COM2->centr=4;
COM2->perif=12;
onrx(COM2,subrx2);
sprintf(buffer1,”MESSAGGIO %d”,num++);
bsave(COM2,buffer1,strlen(buffer1));
```

```
int  com_rxlen(struct COM *porta);
```

Ritorna con il numero di caratteri ricevuti (solo protocol mode 1)

```
int  com_rxcode(struct COM *porta);
```

Ritorna con il primo carattere ricevuto (solo protocol mode 1)

Esempio:

In common.h

```
char msg1[16];
char msg2[16];
void subrx2(void)
{
   if (com_rxcode(COM2)==1) bload(COM2,msg1,16);
   else bload(COM2,msg2,16);
}
```

In start.c

```
com_open(COM2,57600);
protocol_mode(COM2,1);
COM2->centr=4;
onrx(COM2,subrx2);
```

```
int  onrx(struct COM *porta,void
(*subroutine)());

int  ontx(struct COM *porta,void
(*subroutine)());
```

Associa una subroutine da eseguire quando si riceve un pacchetto in modo 1 (onrx) o prima di trasmettere un pacchetto in modo 1(ontx).

Esempio:

In common.h

```
char msg1[16];
char msg2[16];
void subrx2(void)
{
   if (com_rxcode(COM2)==1) bload(COM2,msg1,16);
   else bload(COM2,msg2,16);
}
```

In start.c

```
com_open(COM2,57600);
protocol_mode(COM2,1);
COM2->centr=4;
onrx(COM2,subrx2);
```

FLASH SERIALE

I PLC della serie S400 hanno una memoria non volatile flash seriale di dimensione variabile a seconda del modello da 2 Mbytes a 8 Mbytes.
I primi 1.5 Mbytes della flash sono riservati per il Dos e per la memorizzazione del programma di lavoro.
La parte successiva è a disposizione del programmatore, che puó scrivervi e leggervi i parametri della macchina, i programmi di utente o altro. Queste operazioni vengono eseguite mediante i comandi qui descritti.

I PLC della serie ARM hanno una memory card SD di 2 Gigabytes, alla quale si puó accedere mediante le funzioni standard C di gestione del file system (fopen,fread,fprintf,fscanf ecc.).
Per compatibilità con il S400 le funzionidi flash seriale sono implementate, e usano come supporto fisico una flash simulata dal file C:\FLASH.BIN, che viene creato automaticamente, se non presente, al primo accesso in scrittura alla flash simulata.

```
int sf_wdata(unsigned char *src,long int
dst,long int num);
```

Scrive dei dati in flash

src punta alla struttura o al vettore di dati da scrivere in flash
dst indirizzo della flash a partire dal quale scrivere i dati.
num numero di bytes da scrivere.

Questo comando vede solo l'area di memoria flash riservata al programmatore, sottraendo automaticamente l' offset di 1.5 Mbytes.
Il primo indirizzo di partenza valido è 0.
L' operazione ha successo solo se l' area di memoria in cui vogliamo scrivere è interamente contenuta nella flash.
Il codice di ritorno è il checsum ad 8 bit dei dati salvati.

Esempio:

In common.h

```
struct
{
   char nome[40];
   int  numero;
} my_record;
```

altrove ...

```
sf_wdata(&my_record,0,sizeof(my_record));
```

```
int sf_rdata(unsigned char *src,long int dst,long int num);
```

Legge dei dati dalla flash

src punta alla struttura o al vettore di dati da leggere da flash
dst indirizzo della flash a partire dal quale leggere i dati.
num numero di bytes da leggere.

Questo comando vede solo l'area di memoria flash riservata al programmatore, sottraendo automaticamente l' offset di 1.5 Mbytes.
Il primo indirizzo di partenza valido è 0.
L' operazione ha successo solo se l' area di memoria in cui vogliamo scrivere è interamente contenuta nella flash.
Il codice di ritorno è il checsum ad 8 bit dei dati letti.

Esempio:

In common.h

```
struct
{
   char nome[40];
   int  numero;
} my_record;
```

altrove ...

```
sf_rdata(&my_record,0,sizeof(my_record));
```

```
int sf_cdata(unsigned char *src,long int
dst,long int num);
```

Verifica dei dati in flash

src punta alla struttura o al vettore di dati da comparare con la flash
dst indirizzo della flash a partire dal quale confrontare i dati.
Num numero di bytes da confrontare.

Questo comando vede solo l'area di memoria flash riservata al programmatore, sottraendo automaticamente l' offset di 1.5 Mbytes.
Il primo indirizzo di partenza valido è 0.
L' operazione ha successo solo se l' area di memoria in cui vogliamo scrivere è interamente contenuta nella flash.
Il codice di ritorno dá il numero di errori, ovvero il numero dei bytes che la verifica ha trovato diversi.

Esempio:

In common.h

```
struct
{
   char nome[40];
   int  numero;
} my_record;
```

altrove ...

```
sf_cdata(&my_record,0,sizeof(my_record));
```

```
int sf_wcode(unsigned char *src,long int
dst,long int num);
int sf_rcode(unsigned char *src,long int
dst,long int num);
int sf_ccode(unsigned char *src,long int
dst,long int num);
```

Come i comandi visti sopra, ma riferiti alla parte della flash riservata al Dos.
Non usare mai tali comandi, pena la distruzione del firmware del PLC.

PENNA USB (Solo versione S400)

Nella versione ARM la penna USB è vista e gestita come VOLUME D: con i normali comandi del file system (fopen,fclose,fread ecc) ed accetta file-systems FAT12, FAT16 e FAT32, avando accesso alle sottodirectory di qualsiasi livello.

È possibile collegare ai PLC della serie S400 un' interfaccia USB in grado di leggere direttamente il contenuto di una memoria portatile di tipo Penna USB o, (con opportuno adattatore USB), una Compact Flash o una memoria SD.
Lo scopo è quello di poter leggere o scrivere dei files compatibili con il PC, per scambiare dati con quest' ultimo.
Una gestione completa del file system FAT è fuori dagli scopi e dalle possibilitá di memoria del PLC, tuttavia è possibile leggere e scrivere files con le seguenti limitazioni:

- Il supporto deve essere formattato in modalitá **FAT16**
- Si puó accedere solo alla **prima partizione**
- Tale partizione non deve superare i **2 Gbytes**
- I files debbono trovarsi nella **directory principale** (root)
- É possibile aprire **un solo file per volta**.
- Tenere sulla penna USB solo i files destinati allo scambio con il PLC.

Esistono due serie di comandi per accedere alla penna USB:

- un primo set di comandi dedicati (cf_init, cf_dir ecc..)
- un secondo set standard C (fopen,fread,fwrite ecc..)

Il set standard C è completo e permette di fare tutte le operazioni standard sui files.
Il set dedicato contiene una ripetizione dei comandi del set completo, piú alcuni comandi che non esistono nel C standard, (come ad es. il comando per la directory dei files o per vedere se la penna è inserita).
Mentre il set dedicato fa riferimento esclusivamente alla penna USB, il set standard puó essere usato anche per accedere ad altre periferiche di massa collegate al PLC, come l' hard disk.

I comandi dedicati ritornano con un codice di errore, che se è zero indicano il successo dell' operazione, altrimenti indicano il motivo dell' insuccesso.

Segue una lista dei codici di errore possibili:

```
;  codici di errore:
;
;  0   O.K.
;  1   ERRORE GENERICO DI I/O
;  2   CARD NON INSERITA
;  3   FILE ALREADY OPEN
;  4   FILE NOT FOUND
;  5   DISK FULL
;  6   FILE EXISTS
;  7   DIRECTORY FULL
;  8   FILE NOT OPEN
;  9   NUMERO PACCHETTO ERRATO
; 10   TIMEOUT
; 99   END OF DIRECTORY
```

Essendo l' interfaccia USB capace di gestire un solo file per volta, è bene che i vari task vi accedano con un semaforo.
A tale scopo esistono le istruzioni **mount** ed **unmount.**

```
void cf_init(void);
```

Inizializza l’interfaccia USB

Questo comando è per uso interno del sistema operativo e non è mai necessario usarlo direttamente nel programma.

```
int  cf_dir(char *filename,char
*result,char mode);
```

Verifica dei dati in flash

filename nome del file (puo' contenere il carattere *)
result punta ad una stringa in cui scrivere il nome completo del file trovato.
mode modo di accesso al comando.

Questo comando cerca nella directory principale un file il cui nome corrisponda a quello passato come primo argomento.
Se mode=0 cerca la prima occorrenza del file.
Se mode=1 cerca l'occorrenza successiva.
Ritorna con il codice di errore.

Esempio:

```
char dir[100][25];      //stringa dei risultati

int directory(char *filetype)
{
   static char s[128];  //stringa temporanea
   int er=0;            //azzero codice errore
   int first=0;         //parto da inizio directory
   int n=0;             //azzero num. Files trovati
   while(!er)
   {
      er=fdir(filetype,s,first);
      first=1;          //file successivo
      if(!er)           //se file trovato ...
      {
         strncpy(dir[n],s,23);
         dir[n++][23]=0;
         if(n>99) break;
      }
   }
   return n;
}

x=directory("*.dat");
```

```
int  cf_open(char *filename,char mode);
```

Apre un file

filename nome del file
mode modo di apertura.

Questo comando cerca nella directory principale un file il cui nome corrisponda a quello passato come primo argomento, quindi lo apre nella modalitá specificata dal secondo parametro

Mode

1	read
2	write
3	create
4	random r/w
5	append

Ritorna con il codice di errore.

Il nome del file deve essere compatibile DOS (in caratteri maiuscoli, massimo 8 caratteri + estensione max 3 caratteri)

Esempio:

```
cf_open(”FILE1.DAT”,1);
```

che equivale alla forma standard C:

```
FILE *f;
f=fopen(”FILE1.DAT”,”r”);
```

```
int  cf_test(void);
```

Testa la presenza della penna USB, chiude files aperti.

Questo comando interroga la penna USB per verificarne la presenza.Esso chiude inoltre eventuali files aperti e ritorna con il codice di errore.

Esempio:

```
int err;
if(err=cf_test()) printf(”error %d”,err);
```

```
int  cf_close(void);
```

Chiude il file aperto sulla penna USB.

Questo comando chiude il file aperto sulla penna USB.
Nella presente versione il sistema operativo consente di aprite un solo file per volta, per cui, prima di aprire un nuovo file, occorre chiudere il precedente.
Se nessun file era aperto, questo comando non fa nulla e non segnala errore.
Per precauzione è sempre bene dare un comando di chiusura files all' inizio del programma, in quanto la gestione dei files avviene nell' interfaccia USB e non nel PLC, per cui un eventuale Reset dell' apparecchiatura puó lasciare aperti dei files sull' interfaccia.

Equivale al comando fclose(FILE *f).

Esempio:

```
cf_open("FILE1.DAT",1);
. . . .
cf_close();
```

che equivale alla forma standard C:

```
FILE *f;
f=fopen("FILE1.DAT","r");
. . . .
fclose(f);
```

```
int  cf_del(char *filename);
```

Cancella uno o piú files sulla penna USB.

filename nome del file (puó contenere il carattere *)

Questo comando cancella tutti i files il cui nome corrisponde alla stringa passata come parametro.

Non ha equivalenti nella libreria standard C.

Esempio:

```
cf_del("*.*");
```

Cancella tutti i files sulla penna USB.

```
int  cf_rb(char *buffer,int len);
```

Legge dei dati in binario dal file aperto.

buffer vettore in cui mettere i dati letti.
len numero di bytes da leggere

Questo comando legge il numero specificato di bytes dal file corrente, a partire dalla posizione corrente, ed incrementa la posizione.
All' apertura la posizione corrente è l' inizio del file (apertura in modo read o random).
Per cambiare la posizione corrente usare il comando cf_seek.

Equivale al comando fread del C standard.

Esempio:

```
cf_open("FILE1.DAT",1);
cf_seek(1200);
cf_read(my_buffer,200);
cf_close();
```

che equivale alla forma standard C:

```
FILE *f;
f=fopen("FILE1.DAT","rb");
fseek(1200,SEEK_SET);
fread(my_buffer,200,1,f);
fclose(f);
```

In questo esempio si leggono 200 bytes a partire dal byte n. 1200 del file FILE1.DAT

```
int  cf_wb(char *buffer,int len);
```

Scrive dei dati in binaario nel file aperto.

buffer vettore in cui si trovano i dati da scrivere.
len numero di bytes da scrivere

Questo comando scrive il numero specificato di bytes nel file corrente, a partire dalla posizione corrente, ed incrementa la posizione.
All' apertura la posizione corrente è l' inizio del file (apertura in modo write, create o random)., o la fine del file (apertura in modo append).
Per cambiare la posizione corrente usare il comando cf_seek.

Equivale al comando fwrite del C standard.

Esempio:

```
cf_open(”FILE1.DAT”,4);
cf_seek(1200);
cf_write(my_buffer,200);
cf_close();
```

che equivale alla forma standard C:

```
FILE *f;
f=fopen(”FILE1.DAT”,”wb”);
fseek(1200,SEEK_SET);
fwrite(my_buffer,200,1,f);
fclose(f);
```

In questo esempio si scrivono 200 bytes a partire dal byte n. 1200 nel file FILE1.DAT

```
int  cf_seek(long int posiz);
```

Setta la posizione corrente di lettura o scrittura nel file aperto.

posiz posizione corrente (relativa all‘ inizio del file)

Questo comando cambia la posizione corrente di lettura-scrittura.
Se ci si porta oltre la fine del file, in caso di scrittura, il file verrá riempito di caratteri zero fino alla posizione corrente.

Equivale al comando fseek del C standard.

Esempio:

```
cf_open(”FILE1.DAT”,1);
cf_seek(1200);
cf_read(my_buffer,200);
cf_close();
```

che equivale alla forma standard C:

```
FILE *f;
f=fopen(”FILE1.DAT”,”rb”);
fseek(1200,SEEK_SET);
fread(my_buffer,200,1,f);
fclose(f);
```

In questo esempio si leggono 200 bytes a partire dal byte n. 1200 del file FILE1.DAT

```
int  cf_rd(char *buffer);
```

Legge una stringa di dati dal file di testo aperto.

buffer vettore in cui mettere i dati letti.

Questo comando legge una stringa di bytes dal file corrente, a partire dalla posizione corrente, ed incrementa la posizione.
Si ferma se trova **un carattere CR** (ascii 13) oppure **la fine del file** oppure in ogni caso **dopo 240 bytes** letti.
All' apertura la posizione corrente è l' inizio del file (apertura in modo read).

Equivale al comando fgets del C standard.

Esempio:

```
cf_open("FILE1.DAT",1);
cf_read(my_record);
cf_close();
```

che equivale alla forma standard C:

```
FILE *f;
f=fopen("FILE1.DAT","r");
fgets(my_record,240,1,f);
fclose(f);
```

In questo esempio si legge la prima riga del file di testo FILE1.DAT.

```
int  cf_wr(char *buffer);
```

Scrive una riga di testo nel file aperto.

buffer vettore contenente il testo.

Questo comando scrive una riga di testo nel file aperto, terminandola con un carattere ascii CR (13), ed incrementa la posizione.
All' apertura la posizione corrente è l' inizio del file (apertura in modo write o create) o la fine (apertura in modo append).

Equivale al comando fputs del C standard.

Esempio:

```
cf_open("FILE1.DAT",2);
cf_wr(my_buffer);
cf_close();
```

che equivale alla forma standard C:

```
FILE *f;
f=fopen("FILE1.DAT","w");
fputs(my_buffer,f);
fclose(f);
```

In questo esempio si apre il file e vi si scrive una riga di testo.

```
int  cf_status(int mode);
```

legge lo status del file

Aggiorna le variabili:
fl_eof (1 se eof)
fl_pointer (posizione corrente nel file)
fl_size (dimensione del file)

mode modo del comando:

mode=0: ritorna con eof+ actual_byte pointer + file len
mode=1: ritorna con eof+ disk capacity
mode=2: ritorna con eof+ disk capacity + disk free

GESTIONE FILES

La versione corrente di Proteus prevede la penna USB come unico supporto di dati accessibile con le funzioni di file system standard del linguaggio C sui PLC della serie S400.
L'interfaccia USB dell'S400 supporta un solo FCB, per cui è possibile aprire un solo file per volta, nella directory principale della penna USB, che deve essere formattata in modo FAT16.

Ovviamente queste limitazioni non sussistono se sviluppiamo con Proteus un programma che gira su PLC ARM o su PC.
Per garantire l' accesso esclusivo al file system nel caso di piú task, sono state implementate le funzioni di semaforo:

```
void mount(void);
void unmount(void);
```

che permettono di creare una meccanismo di mutua esclusione durante l'accesso al file system.

Esempio:

```
FILE *f;                    //dichiaro l'FCB
```

In un task:

```
mount();                    //riserva l'FCB
f=fopen("FILE1.DAT","w");//apre il file
fputs(my_buffer1,f);        //scrive
fclose(f);                  //chiude il file
unmount();                  //libera l'FCB
```

In un altro task:

```
mount();                    //riserva l'FCB
f=fopen("FILE2.DAT","r");//apre il file
fgets(my_buffer2,f);           //legge
fclose(f);                  //chiude il file
unmount();                  //libera l'FCB
```

Le funzioni standard C per la gestione dei files sono:

```
FILE      *fopen(const char *name,const char *mode);
int       fclose(FILE *file);
size_t    fread(void *buffer,size_t count,size_t
          num,FILE *file);
int       fgetc(FILE *file);
char      *fgets(char *buffer,int max,FILE *file);
int       fscanf(FILE *file,const char *format,...);
size_t    fwrite(const void *buffer,size_t
          count,size_t num,FILE *file);
int       fputc(int c,FILE *file);
int       fputs(const char *buffer,FILE *file);
int       fprintf(FILE *file,const char *format,...);
int       fseek(FILE *file,long offset,int whence);
long int  ftell(FILE *file);
int       feof(FILE *file);
long int  flen(FILE *file);
long int  fpointer(FILE *file);
```

e **limitatamente ai PLC della serie ARM**:

```
void     mount(void);  monta il file system standard
void     unmount(void); flush file system standard
int      ffind(const char *filename,char *result,char
         mode);
void     set_current_drive(const char *drivename);
int      finit (void);
int      fdelete (const char *filename);
int      frename (const char *oldname, const char
         *newname);
int      fcopy (const char *oldname, const char
         *newname);
U32      ffree (const char *drive);
int      fformat (const char *drive,U32 size);
int      mkdir (const char *dirname);
int       remdir  (const char *dirname);
void     chgdir (const char *dirname);
void     set_current_dir (const char *dirname);
char     *get_current_dir(void);
```

Per la descrizione di tali funzioni si rimanda al manuale delle funzioni standard C, sezione STDIO.H

ESPANSIONE I/O seriale

IL PLC S400 supporta l' espansione degli I/O mediante moduli di interfaccia seriale in RS-232 o in fibra ottica.
E' possibile collegare tra loro in cascata fino a 16 di tali moduli, ciascuno dei quali puó avere fino a 128 ingressi, oppure fino a 128 uscite, oppure entrambi.
Affinché tali moduli siano gestiti correttamente dal sistema, occorre che siano dichiarati (normalmente in start.c) mediante il comando:

```
void refresh(unsigned int perif,COM
*com,long int baud,int numin,int
firstin,int numout,int firstout,int
timeout);
```

perif numero della periferica (0..15)
com porta seriale a cui il modulo è collegato (COM1...COM4)
baud baud rate (normalmente 125000)
numin numero di ingressi presenti sul modulo (0..128)
firstin primo ingresso a cui associare il modulo (>32)
numout numero di uscite presenti sul modulo (0..128)
firstout prima uscita a cui associare il modulo (>32)
timeout timeout massimo in mSec. (normalmente 20)

Limitazioni:

La porta COM3 è riservata al monitor se il PLC ne ha uno.
Moduli sulla stessa porta seriale debbono avere la stessa baud-rate.
Gli ingressi di moduli diversi non debbono sovrapporsi.

Esempio:

```
refresh(0,COM1,125000,64,33,0,0,20);
refresh(1,COM1,125000,0,0,64,33,20);
refresh(2,COM1,125000,64,97,64,97,20);
```

ESPANSIONE CAN

Il PLC S400 supporta in modo completamente trasparente al programmatore fino a 6 moduli di espansione CAN, ciascuno dei quali aggiunge agli I/O del PLC:

2 assi ad encoder bidirezionale con tacca di zero
2 uscite analogiche
4 ingressi analogici
16 ingressi digitali
16 uscite digitali

In questa struttura modulare il PLC implementa i primi due moduli (modulo 0 e 1) e i successivi sono costituiti dalle espansioni.

Modulo	2	3	4	5	6	7
Encoders	5-6	7-8	9-10	11-12	13-14	15-16
Anal.Out	5-6	7-8	9-10	11-12	13-14	15-16
Anal.In	9-12	13-16	17-20	21-24	25-28	29-32
Ingressi	i32-47	i48-63	i64-79	i80-95	i96-111	i112-127
Uscite	o32-47	o48-63	o64-79	o80-95	o96-111	o112-127

Questi moduli sono stati pensati per espandere gli assi gestiti dal PLC ma possono essere utilizzati anche per altri scopi.
Se occorre collegare piú di 6 moduli di espansione, questo è possibile gestendo gli stessi come periferiche CAN-OPEN.

Affinché il PLC gestisca correttamente le espansioni, occorre dichiararli mediante il comando:

```
void can_perif(int n,int t,int baud);
```

n numero di moduli esterni (da 0 a 6)
t tempo di refresh in millisecondi (normalmente 1)
baud baud rate del CAN in Kbaud (normalmente 1000)

Esempio:

```
can_perif(2,1,1000);  //2 moduli di espansione
```

Per i controllori della serie ARM esiste una forma alternativa di questo comando:

```
void can_perif_ext(int n,int t,int
baud,int first);
;
```

n	numero di moduli esterni (da 0 a 6)
t	tempo di refresh in millisecondi (normalmente 1)
baud	baud rate del CAN in Kbaud (normalmente 1000)
first	primo modulo esterno. **Se uguale a zero** il primo modulo di espansione è associato a i1-i16, o1-o16, encoder1-encoder2, anal1-anal2, pot1-pot4. **Se uguale ad 1**, il primo modulo di espansione è associato a i17-i32, o17-o32, encoder3-encoder4, anal3-anal4, pot5-pot8. **Se uguale a 2** equivale al comando can_perif.

Questa forma del comando è utile soprattutto nel caso in cui il programma giri su una **unitá senza i-o fisici**, per cui si demanda ad una periferica esterna il supporto fisico degli i-o.

Esempio:

```
can_perif_ext(2,1,1000,0);  //2 moduli di espansione
con inizio da i1, o1, encoder1, pot1, anal1
```

CAN-BASIC

Il compilatore Proteus ES ha una serie di funzioni che permettono di gestire la periferica CAN a basso livello.

Un messaggio CAN è inviato e ricevuto con la struttura:

```
typedef struct
{
  unsigned int Frame; // Bits 16..19: DLC - Data Length Counter
                      // Bit 30: Set if this is a RTR message
                      // Bit 31: Set if this is a 29-bit ID message
  unsigned int MsgID; // CAN Message ID (11-bit or 29-bit)
  unsigned int DatA;  // CAN Message Data Bytes 0-3
  unsigned int DatB;  // CAN Message Data Bytes 4-7
} CANBASE_MSG;
```

I comandi relativi alla gestione del CAN in modo di base sono:

Per il primo device CAN:
short **canbase_init**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_txmsg**(CANBASE_MSG *pTransmitBuf);
short **canbase_rxmsg**(CANBASE_MSG *pReceiveBuf);

Per il secondo device CAN (dove presente):
short **canbase_init_2**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_txmsg_2**(CANBASE_MSG *pTransmitBuf);
short **canbase_rxmsg_2**(CANBASE_MSG *pReceiveBuf);

short **canbase_init**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_init_2**(unsigned short can_isrvect,unsigned int can_btr);

can_isrvect è la priorità di gestione dei messaggi CAN. Questo parametro è presente per compatibilità con altre versioni di PROTEUS, non è influente, e generalmente gli viene assegnato il valore 4.

can_btr è la baud rate del CAN. Puó assumere i valori:
CANBitrate125k
CANBitrate250k
CANBitrate500k
CANBitrate1M

Questo comando inizializza la periferica CAN.
Se vogliamo utilizzare una baud rate diversa da quelle specificate nella lista dei valori di can_btr, possiamo cambiare in ogni momento la baud rate con il comando:

can_freq(U32 freq); per il CAN1
can_freq_2(U32 freq); per il CAN2

In cui **freq** puó assumere qualsiasi valore numerico.

short **canbase_txmsg**(CANBASE_MSG *pTransmitBuf);
short **canbase_txmsg_2**(CANBASE_MSG *pTransmitBuf);

Trasmette il messaggio CAN contenuto nella struttura passata come parametro, secondo le specifiche della struttura stessa.
Ritorna con codice ZERO se i buffers di trasmissione sono pieni o in caso di BUSOFF (bus del can interrotto)
Altrimenti ritorna con un valore diverso da 0.

Esempio:

```
CANBASE_MSG TXBuf;            // Buffer  tx CAN message
canbase_init(4,CANBitrate1M); // CAN 1 int vect 4 1 Mbaud
TXBuf.Frame = 0x00080000;     // dlc << 16
TXBuf.MsgID = 0x262;          // id pdor1
TXBuf.DatA  = 0x00050000;     // data 1-4
TXBuf.DatB  = 0x00000000;     // data 5-8
while(!canbase_txmsg(&TXBuf));// invia il pacchetto
```

short **canbase_rxmsg**(CANBASE_MSG *pReceiveBuf);
short **canbase_rxmsg_2**(CANBASE_MSG *pReceiveBuf);

Riceve un eventuale messaggio dal buffer del CAN.
I messaggi presenti sul bus del CAN vengono ricevuti automaticamente e memorizzati in un buffer circolare di 64 messaggi.
Se il buffer non è vuoto questa funzione scoda un messaggio dal buffer e ritorna con codice 1.
Se il buffer è vuoto, significa che non ci sono messaggi e ritorna con il codice ZERO.

Esempio:

```
CANBASE_MSG RXBuf;       // Buffer  rx CAN message
U32 can1a,can1b,can2a,can2b;
while(canbase_rxmsg(&RXBuf))
{
   if ((RXBuf.MsgID==((3<<7)|(98))))
   {
      can1a=RXBuf.DatA;  //legge i primi 4 bytes
      can1b=RXBuf.DatB;  //legge i secondi 4 bytes
   }
   if ((RXBuf.MsgID==((5<<7)|(98
   {
      can2a=RXBuf.DatA;  //legge i primi 4 bytes
      can2b=RXBuf.DatB;  //legge i secondi 4 bytes
   }
}
```

CAN-OPEN

Il compilatore Proteus ha una serie di funzioni che permettono di gestire sul PLC lo stack CAN-OPEN.

```
void canopen_on(unsigned char sub);
void canopen_off(void);
unsigned char canopen_empty(void);
unsigned char canopen_full(void);
unsigned char canopen_msg(void);
void canopen_flush(void);
unsigned char canopen_rxmsg(void);
unsigned char canopen_txmsg(void);
unsigned char canopen_txmsg_len(unsigned char len);
unsigned char canopen_requestmsg(void);
unsigned char canopen_txsdo(void);
void canopen_onrx(void(*subroutine)());
void buson(void);
void canopen_standard_sub(void);
unsigned char canopen_perif(U8 i,U8 type,void(*subrx1)(),void
(*subrx2)());
U8 send_sync(void);
```

L’ utilizzo di queste funzioni è illustrato nell’ esempio EX_CANOPEN.

ETHERNET

Le unitá basate sul micro ARM (serie VK e S4000) hanno un' interfaccia ethernet gestita mediante una serie di funzioni implementate nel BIOS.
Le funzioni essenziali sono:

Funzioni generali (host e server)

void **SetIp**(IP *ip,U16 a,U16 b,U16 c,U16 d);

Per implementare un Server:

U32 **EMAC_Server**(U16(*subio)(),U32 flags);

Per implementare un Host

HOST ***CreateHost**(U8 a,U8 b,U8 c,U8 d,U16 port);
U32 **EMAC_Tx**(HOST *host,U8 *txbuffer,U16 flags,U16 txlen);
U32 **EMAC_Rx**(HOST *host,U8 *rxbuffer,U16 flags,U16 rxmaxlen);

Per implementare un Host o Host/Server

HOST ***CreateHost**(U8 a,U8 b,U8 c,U8 d,U16 port);
U32 **EMAC_Tx**(HOST *host,U8 *txbuffer,U16 flags,U16 txlen);
U32 **EMAC_Server**(U16(*subio)(),U32 flags);

Altre funzioni

void **DeleteHost**(HOST *host);
void **emac_perif_on**(int n,int p,int ip,int f,int mo,int ma,int in,int ax,int np);

Queste funzioni sono sufficienti per gestire un colloquio server e/o client in rete locale con altre unitá VK o S4000 e con i PC.

void **SetIp**(IP *ip,U16 a,U16 b,U16 c,U16 d);

Questa funzione consente di specificare il proprio indirizzo IP.
Se a,b,c e d valgono tutti **zero** il comando non cambia l' indirizzo IP ma **disabilita la funzione di AUTO-IP**.
Se a,b,c e d valgono tutti **255** il comando non cambia l' indirizzo IP ma **abilita la funzione di AUTO-IP**.

La funzione di **AUTO-IP**, abilitata per default, consente alla periferica di esaminare i messaggi in rete, e di assumere automaticamente come proprio indirizzo IP quello di una eventuale richiesta **ARP**, indirizzata ad un nodo il cui indirizzo IP sia **xxx.xxx.xxx.n** , dove **n=200+perif**, essendo perif il numero della periferica stessa (contenuto in ram non volatile).
Questo meccanismo consente ai nodi di autoadeguarsi alla rete locale presente, senza bisogno di settare alcun parametro relativo alla rete, ma in certi casi puó essere controproducente, per cui è possibile disabilitarlo da programma.

Esempio:

```
SetIp(0,0,0,0);          //disabilito auto-ip
SetIp(192,168,1,33);     //setto il mio ip
```

HOST *__CreateHost__(U8 a,U8 b,U8 c,U8 d,U16 port);
void __DeleteHost__(HOST *host);

CreateHost consente di creare un Socket di supporto alla trasmissione o alla ricezione di messaggi UDP o TCP-IP.

Un socket di trasmissione contiene l' indirizzo IP del destinatario, ed il numero della porta UDP o TCP.
Occorre un socket diverso per ogni destinatario e per ogni porta a cui intendiamo inviare messaggi tramite il comando EMAC_Tx.

Il socket di ricezione puó essere unico, in quanto non sappiamo a priori da chi riceveremo dei messaggi, né su quale porta, e quindi possiamo crearlo lasciando a zero l' indirizzo IP e quello della porta. Sará la ricezione del messaggio stessa a riempire questi campi.

DeleteHost serve a disallocare la memoria riservata alla struttura host. Puó servire per distruggere un host di trasmissione temporaneo eventualmente creato come variabile locale di una subroutine.

Esempio:

```
Void send_a_message(void)
{
    HOST *h=CreateHost(192,168,1,100,1000);
    char buffer[]="il mio messaggio\n";
    EMAC_Tx(h,buffer,UDP,strlen(buffer));
    DeleteHost(h);
}
```

Manda il messaggio UDP "il mio messaggio" al nodo 192.168.1.100 sulla porta 1000.

U32 **EMAC_Tx**(HOST *host,U8 *txbuffer,U16 flags,U16 txlen);

Questa funzione permette di inviare un messaggio ad un nodo e su una porta associata al parametro host.

Host è il puntatore al socket di trasmissione da utilizzare, che identifica l' indirizzo IP e la porta del destinatario.
Txbuffer è il puntatore al buffer che contiene il messaggio da trasmettere
Flags puó essere UDP o TCP ed identifica il protocollo da utilizzare per la trasmissione
Txlen è la lunghezza del messaggio da trasmettere

Ritorna con il valore 1 se la trasmissione è andata a buon fine.

Il socket Host è utilizzato dall' ARP per associare l' indirizzo MAC all' indirizzo IP del destinatario.
Se si crea un nuovo socket, anche se si invia un messaggio ad un nodo a cui abbiamo giá inviato altri messaggi con altri socket, verrá fatta una nuova richiesta ARP per ricevere l' indirizzo MAC del destinatario, condizione indispensabile per poter inviare il messaggio.
Usando un socket globale, la richiesta ARP verrá eseguita solo la prima volta, e il socket memorizzerá nella propria struttura l'indirizzo MAC del nodo destinatario per i messaggi successivi.
In genere, sistemi operativi come Windows o Linux hanno una propria gestione dello stack ARP, in cui memorizzano l'indirizzo MAC dei nodi per un certo tempo (in genere 5 minuti) dopo di cui lo richiedono nuovamente.
Nel nostro caso l' indirizzo MAC, una volta acquisito, viene mantenuto indefinitamente, a meno di non mettere a 255 i 6 campi dell' indirizzo stesso nel socket Host.

Esempio:

```
Void send_a_message(void)
{
    HOST *h=CreateHost(192,168,1,100,1000);
    char buffer[]="il mio messaggio\n";
    EMAC_Tx(h,buffer,UDP,strlen(buffer));
    DeleteHost(h);
}
```

U32 **EMAC_Rx**(HOST *host,U8 *rxbuffer,U16 flags,U16 rxmaxlen);

Questa funzione consente la richiesta **a polling** di messaggi ethernet. **Non è una funzione indispensabile**, e puó essere sostituita piú efficacemente dalla funzione **EMAC_Server** che fa la stessa cosa in modo automatico e trasparente al programma, senza che quest' ultimo stia a preoccuparsi di esaminare se sono arrivati nuovi messaggi.

Host	è il socket di ricezione. Non deve necessariamente contenere l'indirizzo IP e la porta del messaggio da ricevere
Rxbuffer	è il puntatore al buffer dove la funzione deve copiare il messaggio ricevuto
Flags	puó essere UDP o TCP e identifica il protocollo
Rxmaxlen	è la lunghezza massima a cui eventualmente troncare il messaggio quando si copia in rxbuffer

Ritorna con la lunghezza del messaggio ricevuto (altrimenti 0)

Ethernet è gestito da un server, attivato all' accensione della macchina, che gestisce il traffico sul nodo (ARP, PING, TFTP ecc.). Quando il nodo riceve un messaggio a lui destinato, che non sia gestito dal sever, questo viene messo in un buffer (di 32 messaggi).
La funzione EMAC_Rx scoda gli eventuali messaggi presenti in tale buffer. Possiamo leggere in host->ip , host->mac e in host->port da chi è arrivato il messaggio e su quale porta.

Esempio:

```
Void receive_a_message(void)
{
    HOST *h=CreateHost(0,0,0,0,0);
    char buff[100];
    int port;
    if (EMAC_rx(h,buffer,UDP,100))
    {
       port=h->port;
       . . . . .
    }
    DeleteHost(h);
}
```

U32 **EMAC_Server**(U16(*subio)(),U32 flags);

Ethernet è gestito da un server, attivato all' accensione della macchina, che gestisce il traffico sul nodo (ARP, PING, TFTP ecc.).
Quando il nodo riceve un messaggio a lui destinato, che non sia gestito dal sever, se sono presenti funzioni di utente di gestione messaggi, questo viene proposto alle funzioni utente.
Se anche queste ultime ritornano con 0 (nessuna risposta) il messaggio viene messo in un buffer (di 32 messaggi).

La funzione **EMAC_Server** permette di definire tali funzioni:

U16(*subio)()	è la funzione di gestione dei messaggi
Flags	UDP (funzione gestione messaggi UDP) oppure TCP (funzione di gestione dei messaggi TCP-IP)

La funzione utente è del tipo:

U16 funzione(char *rxmsg,char *txmsg,int flags,int port,int length)

Dove:

Rxmsg	è il puntatore al messaggio ricevuto
Txmsg	è il puntatore al buffer in cui mettere il messaggio di risposta
Flags	puó essere UDP o TCP
Port	è la porta del messaggio
Length	è la lunghezza del messaggio ricevuto.

È buona norma che la funzione copi in un proprio buffer i messaggi rxmsg e txmsg per evitare problemi di allineamento.
Se tale funzione riempie il buffer txmsg e ritorna con un valore maggiore di ZERO, viene inviato su ethernet un messaggio di risposta contenente il buffer txmsg e di lunghezza pari al valore di ritorno della funzione.
Se EMAC_Server è utilizzato dal client, chiaramente la funzione utente si occuperá solo di gestire i messaggi in arrivo (le risposte del server), e ritornerá con valore ZERO.

È possibile definire solo due funzioni EMAC_Server per volta, una di tipo UDP e una di tipo TCP.
In ogni momento possiamo inviare un nuovo comando EMAC_Server per sostituire al volo la funzione di ricezione con un' altra.

Esempio server:
Se riceve sulla porta 1000 un messaggio contenente un numero (in ascii), risponde con un messaggio contenemte il doppio del numero (in ascii)
Se riceve sulla porta 1001 un messaggio contenente un numero (in ascii), risponde con un messaggio contenemte la metá del numero (in ascii)

```
EMAC_Server(pippo,UDP);

U16 pippo(char *rx,char *tx,int flags,int port,int len)
{
   int n=0;
   switch(port)
   {
      case 1000:
      sscanf(rx,"%d",&n);
      sprintf(tx,"%d\n",n*2);
      return strlen(tx);
      break;
      case 1001:
      sscanf(rx,"%d",&n);
      sprintf(tx,"%d\n",n/2);
      return strlen(tx);
      break;
      default:
      return 0;
      break;
   }
}
```

Il client corrispondente potrebbe essere del tipo:
E' il modo assolutamente piú complicato che si possa immaginare per fare semplicemente l' operazione n2=n1*2.

```
int n1
volatile int n2;
char buffer[100];
HOST *h=CreateHost(192,168,1,100,1000);
EMAC_Server(caio,UDP);
 . . . .
n1=120;
sprintf(buffer,"%d\n",n1);
EMAC_Tx(h,buffer,UDP,strlen(buffer));
```

```
U16 caio(char *rx,char *tx,int flags,int port,int
len)
{
   switch(port)
   {
      case 1000:
      sscanf(rx,”%d”,&n2);
      return 0;
      break;
   }
}
```

Il client trasforma il numero n1 in una stringa ascii e lo manda al nodo 192.168.1.100 alla porta 1000
Il server riceve il messaggio, converte la stringa in numero, lo moltiplica per due, lo riconverte in stringa e invia la risposta al mittente
Il client intercetta la risposta all’ arrivo, converte la stringa di risposta in un numero e lo mette in n2, quindi ritorna con zero (nessuna risposta per evitare un loop infinito).
È solo un esempio, per moltiplicare un numero per due non fate cosí !!!

void **emac_perif_on**(int n,int p,int ip,int f,int mo,int ma,int in,int ax,int np);

È una funzione estremamente potente, che permette di espandere gli i-o di una periferica (o di un PC) su altre periferiche della serie VK o S4000.
È l' equivalente per ethernet della funzione **can_perif_ext** del CAN, ma molto piú potente, versatile e veloce.

I parametri:

N numero dello slot virtuale. Deve essere 0 per la prima emac_perif_on, uno per la successiva e cosí via.
Una emac_perif_on con un numero uguale a uno precedente rimpiazza la precedente.
P numero della periferica fisica. Vedere le tabelle 1 e 2
IP quarto byte dell' indirizzo IP della periferica fisica di espansione.
F frequenza di polling (in genere viene messo ad 1=1 millisecondo).
Mo maschera delle uscite escluse (32 bit, ogni bit ad 1 esclude il controllo remoto del corrispondente bit di uscita)
Ma maschera delle uscite analogiche escluse (4 bit, ogni bit a 1 esclude il controllo remoto della corrispondente uscita analogica.)
In numero di ingressi digitali remoti validi per il modulo
Ax numero di assi remoti validi per il controllo
Np numero di ingressi analogici remoti validi per il modulo.

————————————-Tabella 1 (VK)————————————-

Perif(P)	in	out	encoders	pot	anal_out
0	i1-i16	o1-o16	encoder1-2	pot1-4	anal1-2
1	i17-i32	o17-o32	encoder3-4	pot5-8	anal3-4
2	i33-i48	o33-o48	encoder5-6	pot9-12	anal5-6
3	i49-i64	o49-o64	encoder7-8	pot13-16	anal7-8
4	i65-i80	o65-o80	encoder9-10	pot17-20	anal9-10
5	i81-i96	o81-o96	encoder11-12	pot21-24	anal11-12
6	i97-i112	o97-o112	encoder13-14	pot25-28	anal13-14
7	i113-i128	o113-o128	encoder15-16	pot29-32	anal15-16

————————————-Tabella 1 (S4000)————————————-

Perif(P)	in	out	encoders	pot	anal_out
0	i1-i32	o1-o32	encoder1-4	pot1-8	anal1-4
2	i33-i64	o33-o64	encoder5-8	pot9-16	anal5-8
4	i65-i96	o65-o96	encoder9-12	pot17-24	anal9-12
6	i97-i128	o97-o128	encoder13-16	pot25-32	anal13-16

È possibile mescolare periferiche VK e S4000, purché non si sovrappongano. Per le periferiche VK è possibile definire piú encoders (fino a 4 ciascuna) purché non si sovrappongono.

Non è necessario che le periferiche siano contigue e che coprano tutto l' intervallo degli i-o.
Possono esserci buchi, nel qual caso gli i/o relativi ai buchi non sono validi e non deve essere usati nel programma.
La sola cosa fondamentale è che il primo parametro N nella prima dichiarazione dei moduli parta da zero ed incrementi senza buchi.

Se si definisce un modulo con **P=0** gli i-o a bordo vengono disattivati e remotati su quelli della periferica.

Con il comando **com_disable(0);** è possibile remotare sulla prima periferica definita (quella con N=0) le COM locali.

Limitatamente al PC, il comando **com_disable(0);** remota sulla prima periferica definita anche le porte CAN.

Riassumendo, è possibile espandere con questa funzione:

- **Gli ingressi digitali fino a 128**
- **Le uscite digitali fino a 128**
- **Gli ingressi analogici fino a 32**
- **Le uscite analogiche fino a 16**
- **Gli encoders fino a 16**
- **Gli encoders monodirezionali fino a 16**
- **I motori passo-passo fino a 16**
- **Le porte COM sul primo modulo (COM1-COM3)**
- **Il CAN sul primo modulo(CAN1 e CAN2) (* solo per PC)**

Le periferiche non necessitano di alcun programma particolare per operare come slave.E' sufficiente lasciarle sulla videata di accensione in waiting connection.

È possibile inoltre avere piú periferiche master, a condizione che non operino sulle stesse uscite. È anche possibile definire periferiche che siano ognuna contemporaneamente master e slave dell' altra, una volta definito quali uscite digitali ed analogiche siano da gestire localmente e quali siano remotate.

I tempi di aggiornamento delle uscite e di lettura degli ingressi sono di 1 millisecondo, e quindi all' incirca uguali a quelli di aggiornamento delle risorse di i/o a bordo. Non ha senso per gli i/o rematati dare dei comandi __i() e __o(); in quanto in ogni caso non sarebbero aggiornati immediatamente.

Esempio 1: configurazione massima, con **un S4000 compact master e 3 S4000 slave**: scrivere in start.c del master:

```
emac_perif_on(1,2,201,1,0,0,32,4,8);
emac_perif_on(2,4,202,1,0,0,32,4,8);
emac_perif_on(3,6,203,1,0,0,32,4,8);
```

Esempio 2: configurazione massima, con **un S4000 senza i/o abordo e 4 S4000 slave**: scrivere in start.c del master:

```
emac_perif_on(0,0,241,1,0,0,32,4,8);
emac_perif_on(1,2,222,1,0,0,32,4,8);
emac_perif_on(2,4,208,1,0,0,32,4,8);
emac_perif_on(3,6,233,1,0,0,32,4,8);
com_disable(0);
```

Esempio 3: configurazione massima, con **un PC master e 4 S4000 slave**: scrivere in start.c del master:

```
emac_perif_on(0,0,201,1,0,0,32,4,8);
emac_perif_on(1,2,202,1,0,0,32,4,8);
emac_perif_on(2,4,203,1,0,0,32,4,8);
emac_perif_on(3,6,204,1,0,0,32,4,8);
com_disable(0);
```

Esempio 4: **configurazione massima, con un PC master e 8 VK3 slave**: scrivere in start.c del master:

```
emac_perif_on(0,0,201,1,0,0,16,2,4);
emac_perif_on(1,1,202,1,0,0,16,2,4);
emac_perif_on(2,2,203,1,0,0,16,2,4);
emac_perif_on(3,3,204,1,0,0,16,2,4);
emac_perif_on(4,4,205,1,0,0,16,2,4);
emac_perif_on(5,5,206,1,0,0,16,2,4);
emac_perif_on(6,6,207,1,0,0,16,2,4);
emac_perif_on(7,7,208,1,0,0,16,2,4);
com_disable(0);
```

APPENDICE

Comandi aggiunti bios 1.7

void **v_copy_line**(int x0,int y0,int dx,int dy,int offset);
void **memcpyfast**(U32 *dst,U32 *src,U32 size);
void **show_arrow**(void *arrow);
void **v_rectfillrotatec**(int xl,int yl,int dxl,int dyl,float a,int xc,int yc,S32 col,int mod);
U32 **fplay**(FILE * file,U32 xs, U32 ys, U32 dx, U32 dy, U32 sx, U32 sy);
void **emac_perif_ext**(int n,int t,int firstmac,int first);
void **emac_perif_on**(int n,int p,int ip,int f,int mo,int ma,int in,int ax,int np);
U32 **emac_perif_init**(void);

Comandi aggiunti bios 2.9

int **Step**(U32 n,U32 cmd,S32 op1,S32 op2,S32 op3,S32 op4);

Comandi aggiunti bios 2.9b

//canbase
short **canbase_init_2**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_txmsg_2**(CANBASE_MSG *pTransmitBuf);
short **canbase_rxmsg_2**(CANBASE_MSG *pReceiveBuf);
// can
void **can_timer_2**(void);
void **can_time_2**(int t);
void **can_perif_2**(int n,int t,int baud);
void **can_perif_ext_2**(int n,int t,int baud,int first);
U8 **can_busoff_2**(void);
U8 **can_msg_2**(void);
void **can_rxmsg_2**(U8 n);
void **buson_2**(void);
void **can_freq_2**(U32 freq);
void **canopen_on_2**(U8 sub);
void **canopen_off_2**(void);
U8 **canopen_empty_2**(void);
U8 **canopen_full_2**(void);
U8 **canopen_msg_2**(void);
void **canopen_flush_2**(void);
U8 **canopen_rxmsg_2**(void);
U8 **canopen_txmsg_2**(void);
U8 **canopen_txmsg_len_2**(U8 len);
U8 **canopen_requestmsg_2**(void);
U8 **canopen_txsdo_2**(void);

U8 **send_sync_2**(void);
void **canopen_standard_sub_2**(void);
void **canopen_standard_poll_2**(int n);
U8 **canopen_perif_2**(U8 i,U8 type,void(*subrx1)(),void(*subrx2)());

void **v_copy_line**(int x0,int y0,int dx,int dy,int offset);

Equivale al comando v_linec ma il quinto parametro, anziché il colore, specifica l' offset in memoria video da cui copiare il colore dei pixel.

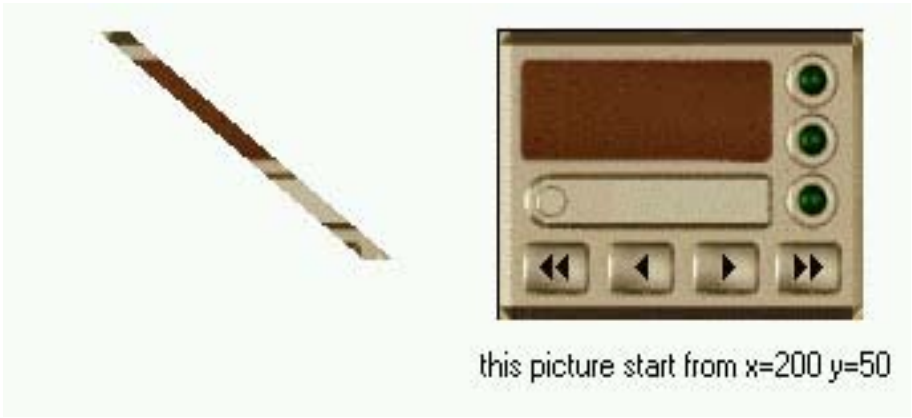
L' offset si puó calcolare come dx+dy*SCREEN_WIDTH.

Questo comando traccia una retta prendendo i colori dei pixels da una immagine che si trova in un' altra zona della memoria video.

Esempio:

```
void hdcopy(void)
{
   int x0;
   for (x0=0;x0<10;x0++)
   {
      v_copy_line(x0+50,50,100,80,150);
   }
}
```

Dá il seguente risultato:



I pixels della linea che parte da coordinate 50,50 vengono presi dallo schermo da coordinate 150 pixels piú a destra (200,50)

void **memcpyfast**(U32 *dst,U32 *src,U32 size);

Dst indirizzo in memoria della destinazione
Src indirizzo in memoria della sorgente
Sizenumero di bytes da copiare

Equivale a memcpy ma è piú rapida, effettuando la copia a 32 bit (memcpy copia 8 bit per volta).

La condizione per usare questo comando è che dst, src e size siano **multipli di 4.**

È particolarmente utile per copiare zone della memoria video.

Esempio

```
int vett1[100];
int vett2[100];
 . . . .
memcpyfast(vett2,vett1,sizeof(vett1));
```

void **show_arrow**(void *arrow);

È un comando che serve principalmenteper disegnare lancette di strumenti.
Fa riferimento ad una struttura di tipo ARROW

```
typedef struct ARROW
{
   int lun;
   int lar;
   int back;
   int col1;
   int col2;
   int x0;
   int y0;
   int dx;
   int dy;
   int traspa;
   float ang;
   float old_ang;
   float min_step;
   OBJ *center_box;
}ARROW;
```

Questo comando puó essere validamente sostituito dal comando v_rectfillrotatec che è molto piú flessibile e semplice da usare.
Per spiegazioni ed esempi si rimanda all' esempio EXAMPLE_ANIMATION_1.

void **v_rectfillrotatec**(int xl,int yl,int dxl,int dyl,float a,int xc,int yc,S32 col,int mod);

È un comando molto versatile che consente:

- Di disegnare un rettangolo ruotato intorno ad un punto
- Di ruotare un' immagine intorno ad un punto e di copiarla in un' altra parte dello schermo

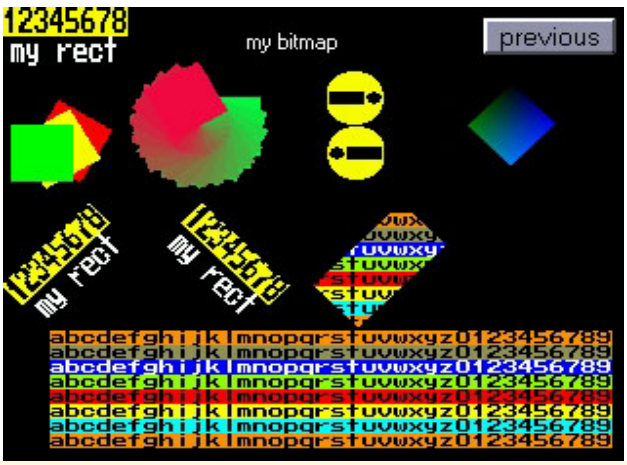
Parametri:

Xl coordinata x del centro di rotazione sul form
Yl coordinata y del centro di rotazione sul form
Dxl larghezza del rettangolo
Dyl altezza del rettangolo
A angolo di rotazione in senso orario
Xc ascissa del centro di rotazione del rettangolo rispetto all' angolo in alto a sinistra
Yc ordinata del centro di rotazione del rettangolo rispetto all' angolo in alto a sinistra
Color colore del rettangolo oppure ram_bitmap (dipende dal modo)
Mod modo. Sono previsti diversi modi, a seconda dei quali cambia il significato del parametro color:

modo colore:
- COLOR colore

modo rotazione immagine:
- BITMAP indirizzo del bitmap_ram nella cache dei bitmaps
- FROMADDR indirizzo assoluto del bitmap in ram
- FROMABSCRT indirizzo assoluto del bitmap in memoria video
- FROMABSFRAME indirizzo relativo a top screen

modo immagine non ruotata:
- FROMRIGIDREL offset del background in memoria video
- FROMRIGIDABS indirizzo assoluto del background



Le funzionalità di questo comando sono illustrate chiaramente nell' esempio EXAMPLE_ROTATIONS da cui è tratta l' immagine a lato.

U32 **fplay**(FILE * fil,U32 xs, U32 ys, U32 dx, U32 dy, U32 sx, U32 sy);

copia su video da coordinate xs,ys leggendo dal file fil, dx * dy pixels con scala sx e sy .
i pixels sono bytes (CRT_256COLORS) o word (CRT_32KCOLORS)
le coordinate sono relative a frame_vect
il file deve puntare al frame corrente

Questo comando in genere non viene usato direttamente, ma tramite le macro multimediali:

int **PlayVideo**(char *file,int xs,int ys,int zoom,int sync);
void **PlayAudio**(char *file);
void **PlayVideoStop**(void);
void **PlayAudioStop**(void);

Parametri di PlayVideo:

File	nome del file multimediale. Se il nome non ha estensione, (ex c:\\pippo) viene cercato il file pippo.vid per il video ed il file pippo.aud per l' eventuale audio. Se non c'è audio, basta non caricare sulla SD il file .aud, oppure specificare come parametro del PlayVideo il file con estensione (c:\\pippo.vid).
Xs	posizione x dell' angolo in alto a sinistra del filmato
Ys	posizione y dell' angolo in alto a sinistra della finestra del filmato.
Zoom	zoom del filmato (1 o 2)
Sync	**sync=0** il filmato è eseguito a velocitá normale **sync<0** viene mostrato il frame –sync*totframes/1000. ad esempio sync=-500 mostra il frame di metá filmato. **sync=1** viene mostrato il frame successivo **sync=2...4** velocitá ridotta di sync (1/2, 1/3, 1/4) **sync>4** viene mostrato il frame numero sync.

La funzione ritorna con il valore **ZERO** se il file esiste, non è giunto alla fine e se il frame indicato esiste, altrimenti ritorna con il valore 1.

PlayAudio esegue un file di tipo audio.
PlayVideoStop e **PlayAudioStop** fermano le relative funzioni.

Con queste funzioni non è possibile vedere filmati di tipo avi o mpeg né riprodurre audio di tipo wav o mp3, ma occorre prima preparare il materiale audio-video in forma non compressa, mediante l'apposita

utility AVICONVERTER.EXE che si trova nella distribuzione di Proteus nella cartella c:\commons\c\aviconverter.

Se vogliamo fare un file audio, possiamo convertirlo da mp3 o wav, in formato RAW 11kbps mono 8 bit, che è il formato utilizzato da PlayAudio.

Chi volesse vedere come sono fatte le macro PlayVideo e PlayAudio, il sorgente si trova in c:\commons\c\armgcc\arm-hitex-elf\include\stubs.h

ATTENZIONE: PlayAudio utilizza il TimerTask, per cui è incompatibile con eventuali programmi che utilizzino tale task.

Gli esempi EXAMPLE_FILM_V8 e EXAMPLE_FILM_VK3 mostrano applicazioni pratiche di questi comandi.

void **emac_perif_on**(int n,int p,int ip,int f,int mo,int ma,int in,int ax,int np);
void **emac_perif_ext**(int n,int t,int firstmac,int first);
U32 **emac_perif_init**(void);

Emac_perif_on è illustrato nel capitolo relativo all'ethernet
Emac_perif_ext ed **emac_perif_init** sono presenti solo per compatibilità e non sono da utilizzare in nuovi progetti.

int **Step**(U32 n,U32 cmd,S32 op1,S32 op2,S32 op3,S32 op4);

Questo comando permette di gestire la movimentazione di **motori passo-passo**, assicurando le corrette rampe di accelerazione e di decelerazione, la corretta velocitá ,e gestendo per ogni motore le uscite ENABLE, DIRECTION e STEP.
Le frequenze massime operative sono di 80KHz nel caso di 1 motore per unitá, di 40 KHz nel caso di 2 motori, di 20 KHz nel caso di 4 motori.

Parametri:

N numero del motore. La serie VK gestisce fino a due motori per ogni unitá, mentre la serie S4000 ne gestisce fino a 4. Se si utilizza l' espansione Ethernet (vedi il comando emac_perif_on) è possibile per l' unitá master gestire direttamente i motori passo-passo degli slave, fino a 16 motori. per questo N puó assumere valori compresi tra 1 e 16
Cmd È il comando da dare al motore (vedi tabella sottostante)
Op1 primo operando (vedi tabella sottostante)
Op2 secondo operando (vedi tabella sottostante)
Op3 terzo operando (vedi tabella sottostante)
Op4 quarto operando (vedi tabella sottostante)

Ogni motore ha una struttura di supporto del tipo:

```
typedef struct STEP_MOTOR
{
    S32 quota;    //quota attuale
    S32 target;   //quota da raggiungere
    S32 target1;  //quota fine acc.
    S32 target2;  //quota inizio dec.
    U32 rampa;    //rampa acc. e dec. in step
    U32 speed;    //v. max in (clock*8 @vmax)
    U32 status;   //0=non ist. 1=fermo 2=acc. 3=regime 4=dec. 5=stop
    U32 passo;    //passo parziale rampa
    U32 t;        //tempo attuale semiimpulso (in usec/8)
    U32 k;        //livello digitale UP-DOWN
    U32 overrun;  //vel. magg. di calcolo
    S32 ud;       //up/down
    U8  *onoff;   //usc. di onoff
    U8  *dir;     //usc. di dir
    U32 oldrampa;
    U32 oldspeed;
    U32 *tab;
}STEP_MOTOR;
```

Tabella dei comandi per il motore:

Comando	op1	op2	op3	op4	return
1=init	onoff	dir	quota	–	&quota (0 se error)
2=go	quota	speed	rampa	wait	1 (0 se posiz in corso)
3=delta	delta q	speed	rampa	wait	1 (0 se posiz in corso)
4=stop	–	–	–	–	1 se pos. in corso, else 0
5=busy	–	–	–	–	1 se pos in corso, else 0
6=errors	–	–	–	–	errors
7=query	–	–	–	–	&step_motor
8=0ff	wait	–	–	–	–

Nel comando **init** indichiamo con **onoff** e **dir** il numero dell' uscita corrispondente (es. se onoff=o1 porremo onoff=1)

Speed è indicata in impulsi al secondo.
Rampa è la rampa (numero di gradini di velocitá) di accelerazione e decelerazione, ed è indicata in impulsi (0=senza rampa, valore massimo = 5000)
Wait è il tempo di attesa in millisecondi che intercorre tra **Enable** e **Dir**.
&step_motor è il puntatore alla struttura di supporto del motore.

Le uscite del clock sono le stesse delle uscite analogiche (occorre selezionare un ponticello sulla scheda per commutare la funzione)

vk3-v5-v8-v10:

Step1	PWM0_4	P3.19
Step 2	PWM0_5	P3.20

s4000

Step 1	PWM0_1	P1.2
Step 2	PWM0_2	P1.3
Step 3	PWM0_3	P1.5
Step 4	PWM0_4	P1.6

Per maggiori chiarimenti vedere l' esempio EXAMPLE_STEP_MOTORS

short **canbase_init_2**(unsigned short can_isrvect,unsigned int can_btr);
short **canbase_txmsg_2**(CANBASE_MSG *pTransmitBuf);
short **canbase_rxmsg_2**(CANBASE_MSG *pReceiveBuf);
void **can_timer_2**(void);
void **can_time_2**(int t);
void **can_perif_2**(int n,int t,int baud);
void **can_perif_ext_2**(int n,int t,int baud,int first);
U8 **can_busoff_2**(void);
U8 **can_msg_2**(void);
void **can_rxmsg_2**(U8 n);
void **buson_2**(void);
void **can_freq_2**(U32 freq);
void **canopen_on_2**(U8 sub);
void **canopen_off_2**(void);
U8 **canopen_empty_2**(void);
U8 **canopen_full_2**(void);
U8 **canopen_msg_2**(void);
void **canopen_flush_2**(void);
U8 **canopen_rxmsg_2**(void);
U8 **canopen_txmsg_2**(void);
U8 **canopen_txmsg_len_2**(U8 len);
U8 **canopen_requestmsg_2**(void);
U8 **canopen_txsdo_2**(void);
U8 **send_sync_2**(void);
void **canopen_standard_sub_2**(void);
void **canopen_standard_poll_2**(int n);
U8 **canopen_perif_2**(U8 i,U8 type,void(*subrx1)(),void(*subrx2)());

Questi comandi gestiscono la seconda interfaccia CAN delle periferiche della serie S4000 e sono identici ai rispettivi comandi senza il suffisso _2, con la sola differenza che comandano la seconda linea CAN anziché la prima.

Le due linee CAN sono identiche e completamente indipendenti.