G C T R

Grifo® C To Rom

MANUALE UTENTE

grifo®
ITALIAN TECHNOLOGY

Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
E-mail: grifo@grifo.it
http://www.grifo.it http://www.grifo.com
Tel. +39 051 892.052 (r.a.) FAX: +39 051 893.661

GCTR Edizione 3.40 *Rel. 04 Febbraio 2002*

G C T R

Grifo® C To Rom

MANUALE UTENTE

Il **GCTR** é un completo e potente pacchetto software che consente di sviluppare programmi applicativi in C, avvalendosi di ambienti di sviluppo e debug comodi, veloci ed efficaci. E' disponibile per tutte le schede basate sui microprocessori della famiglia **Intel 86**, appartenenti al vasto carteggio industriale della **grifo®**. Con la possibilità di salvare il codice sviluppato sulla FLASH EPROM della scheda in uso il **GCTR** semplifica anche la fase finale di installazioni ed aggiornamento che può essere quindi facilmente effettuato anche sul campo. Una serie di funzioni di libreria consente di gestire immediatamente una ricca serie di interfaccie operatore, normalmente presenti nella maggioranza delle macchine automatiche.

grifo®
ITALIAN TECHNOLOGY

Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
E-mail: grifo@grifo.it
http://www.grifo.it http://www.grifo.com
Tel. +39 051 892.052 (r.a.) FAX: +39 051 893.661

GCTR Edizione 3.40 *Rel. 04 Febbraio 2002*

Vincoli sulla documentazione **grifo**® Tutti i Diritti Riservati

IMPORTANTE

Tutte le informazioni contenute sul presente manuale sono state accuratamente verificate, ciononostante **grifo**® non si assume nessuna responsabilità per danni, diretti o indiretti, a cose e/o persone derivanti da errori, omissioni o dall'uso del presente manuale, del software o dell' hardware ad esso associato.
grifo® altresi si riserva il diritto di modificare il contenuto e la veste di questo manuale senza alcun preavviso, con l' intento di offrire un prodotto sempre migliore, senza che questo rappresenti un obbligo per **grifo**®.
Per le informazioni specifiche dei componenti utilizzati sui nostri prodotti, l'utente deve fare riferimento agli specifici Data Book delle case costruttrici o delle seconde sorgenti.

LEGENDA SIMBOLI

Nel presente manuale possono comparire i seguenti simboli:

Attenzione: Pericolo generico

Attenzione: Pericolo di alta tensione

Marchi Registrati

INDICE GENERALE

INDICE DELLE FIGURE

INTRODUZIONE

L'uso di questi dispositivi é rivolto - IN VIA ESCLUSIVA - a personale specializzato.

Scopo di questo manuale é la trasmissione delle informazioni necessarie all'uso competente e sicuro dei prodotti. Esse sono il frutto di un'elaborazione continua e sistematica di dati e prove tecniche registrate e validate dal Costruttore, in attuazione alle procedure interne di sicurezza e qualità dell'informazione.

I dati di seguito riportati sono destinati - IN VIA ESCLUSIVA - ad un utenza specializzata, in grado di interagire con i prodotti in condizioni di sicurezza per le persone, per la macchina e per l'ambiente, interpretando un'elementare diagnostica dei guasti e delle condizioni di funzionamento anomale e compiendo semplici operazioni di verifica funzionale, nel pieno rispetto delle norme di sicurezza e salute vigenti.

Per un corretto rapporto coi prodotti, é necessario garantire leggibilità e conservazione del manuale, anche per futuri riferimenti. In caso di deterioramento o più semplicemente per ragioni di approfondimento tecnico ed operativo, consultare direttamente l'Assistenza Tecnica autorizzata.

Al fine di non incontrare problemi nell'uso di tale ambiente di programmazione, é conveniente che l'utente - PRIMA DI COMINCIARE AD OPERARE - legga con attenzione tutte le informazioni contenute in questo manuale. In una seconda fase, per rintracciare più facilmente le informazioni necessarie, si può fare riferimento all'indice generale e all'indice analitico, posti rispettivamente all'inizio ed alla fine del manuale.

La **grifo®** non garantisce che questo sofware soddisfi le richieste dell'utente, che la produzione non cessi o sia priva di errori o che tutti gli eventuali errori siano corretti. La **grifo®** non é inoltre responsabile dei problemi causati dalle modifiche dell'hardware dei calcolatori e dei sistemi operativi che si possono verificare nel tempo.

VERSIONE

Il presente manuale é riferito al **GCTR** versione **3.5** e successive. La validità delle informazioni riportate é quindi subordinata al numero di versione del software in uso e l'utente deve quindi sempre verificare la giusta corrispondenza tra le due indicazioni. Il numero di versione é riportato sulle etichette dei dischetti **GCTR** ricevuti ed in alcuni altri programmi, esempi, ecc.
In questo manuale sono inoltre presenti le informazioni relative ad altri programmi che costituiscono una parte integrante del **GCTR**: ognuno di questi ha il proprio numero di versione che, quando necessario, viene presentato in questo manuale.
In caso di necessità di assistenza tecnica é di fondamentale importanza che l'utente oltre alla descrizione del problema, fornisca il/i numeri di versione del/dei programmi in uso.
Per quanto riguarda le versioni ordinabili del **GCTR** e la descrizionedelle modifiche effettuate nel tempo, fare riferimento al capitolo "VERSIONI GCTR".

INFORMAZIONI GENERALI

Questo manuale fornisce tutte le informazioni hardware e software per consentire all'utente il miglior utilizzo del **GCTR** (**Grifo®** C To Rom).
Nel presente manuale si utilizzano le seguenti indicazioni:

Programma applicativo: é il programma sviluppato dall'utente che gestisce, dal punto di vista software, l'applicazione da realizzare.

Scheda di controllo: é la scheda **grifo®** su cui usare il **GCTR** e per cui sviluppare il programma applicativo.

Il **GCTR** é un completo e potente pacchetto software che consente di sviluppare programmi applicativi in C, avvalendosi di comodi ambienti di sviluppo e di debug. E' disponibile per tutte le schede basate sui microprocessori della famiglia **Intel 86**, appartenenti al vasto carteggio industriale della **grifo®**.
Il **GCTR** consente di operare sfruttando un ambiente di lavoro molto evoluto, che non necessita nemmeno di una conoscenza approfondita dell'hardware usato ed è stato realizzato con l'obiettivo di semplificare e velocizzare le fasi di sviluppo, prova ed installazione del sistema da realizzare.
Il pacchetto é composto da vari sottogruppi indipendenti e non, che insieme soddisfano le odierne esigenze dei programmatori, in accordo con le loro esperienze lavorative. In paticolare, oltre al codice che consente di **romare** il programma sviluppato dall'utente, sono compresi un **compilatore** e **linker**, un **debugger**, un **programmatore**, dei **programmi di utilità** generale e degli **esempi** pronti all'uso.
Il **C** é uno dei linguaggi preferiti dalla maggioranza dei programmatori e grazie alla sua modularità, compattezza, flessibilità ed efficienza dispone di una enorme quantità di codice già scritto e spesso pronto all'uso. Questo linguaggio consente di gestire direttamente l'hardware di sistema, sfruttare delle strutture dati di vario tipo e composizione, gestire facilmente gli interrupts, disporre di potenti istruzioni di controllo e di sfruttare tutti i vantaggi di una programmazione ad alto livello.
Il **GCTR** utilizza il compilatore C della Borland che é senza dubbio uno dei più diffusi e quindi più conosciuto. Quest'ultimo viene fornito solo nelle sue parti essenziali ed é quindi compito dell'utente procurarsi il pacchetto completo, in modo da poter usufruire di tutti i tools, delle funzioni di libreria, della documentazione in linea e su carta, ecc.
Il **Borland C/C++** e' un ambiente di sviluppo per programmi **DOS e/o WINDOWS** quindi, anche se il suo compilatore e linker possono essere usati per sviluppare codice embedded, ha delle librerie ed un debugger che richiedono la presenza di un sistema operativo. Il **GCTR** fornisce le parti mancanti ai programmatori che vogliono usare il Borland C/C++ per sviluppare programmi per un hardware embedded che non dispone di un sistema operativo.
Sia il **GCTR** che ogni programma realizzato con il **GCTR** non è soggetto a licenze: l'utente può sviluppare un numero illimitato di programmi applicativi e di diverse versioni di questo, senza dover informare la **grifo®** in alcun modo.

- Completo ambiente di sviluppo romato, per le CPU della famiglia **I86** e compatibili.
- Programmazione in **Borland C/C++**.
- Gestione del debug sulla scheda in uso tramite il **Turbo Debugger** della Borland.
- Debugger remoto a livello **sorgente**.
- Alta velocità di scaricamento programma applicativo da debuggare (circa **6 KByte** al secondo).
- **Codice di partenza** romabile che esegue il programma applicativo in C, da un reset o

da un'accensione.
- Utilizzo del modello di memoria **large** per avere gli spazi massimi per codice, dati, stack ed heap.
- Disponibilità completa di **floating point**.
- **Libreria romabile** che comprende le funzioni piu' frequentemente usate (malloc, free, interrupt, delay, ecc.), soprattutto nel campo dell'automazione.
- Possibilità di utilizzare le funzioni di **console** ad alto livello (cprintf, cputs, getch, kbhit, ecc.) per gestire una serie di pannelli operatore come **QTP xxx**, **QTP xxxP**, terminali seriali o più semplicemente **display** alfanumerici e **tastiere** a matrice.
- **Locator** flessibile per microprocessori 80x86, predisposto per generare file in formato binario.
- Predisposto per poter operare anche in abbinamento a programmi applicativi che gestiscono **interrupt**, senza limiti nelle procedure di risposta.
- Codice salvato su **EPROM** o **FLASH EPROM**.
- Possibilita' di utilizzare solo una porzione della **RAM** come area dati, stack ed heap e di riservare la rimanente per **salvataggio parametri**, **data logher**, ecc.
- Possibilita' di utilizzare solo una porzione della **ROM** come area codice e di riservare la rimanente per configurazioni, **messaggi**, **tabelle**, ecc.
- Uso del **compilatore** e del **linker** standard della Borland.
- Gestione del **debug hardware**.
- Completa gestione della circuiteria di **watch dog** della scheda di controllo. L'utente può mantenere sempre collegata tale circuiteria, quando necessaria, incluso durante le fasi di scaricamento e debug.
- E' un ambiente di sviluppo **non intrusivo**, infatti non ha propri interrupt e non svolge alcuna operazione autonomamente. Solo durante la fase di debug utilizza una linea seriale della scheda remota ed il relativo interrupt di ricezione.
- E' **deterministico**: i tempi di esecuzione delle sue funzioni sono costanti e quindi si presta ad essere usato anche in applicazioni real time.
- Include apposito programma di **installazione** che si occupa di definire tutte le possibili configurazioni del pacchetto.
- **Nessuna licenza** o costi aggiuntivi.
- Fornito su **dischetti**, manuale utente (su **CD**) e dispositivo di memoria programmato.

Vista la naturale evoluzione dei pacchetti software, si faccia sempre attenzione all'eventuale presenza del file READ.ME nel disco o nella directory di lavoro. Tale file coincide con la serie di aggiunte, modifiche e miglioramenti apportati nel tempo, a tutto il pacchetto software e non ancora riportati sul manuale: se é presente lo si deve esaminare, stampare ed allegare al presente manuale.

MATERIALE NECESSARIO

Viene di seguito riportata una breve descizione del materiale (hardware e software), necessario per operare con il **GCTR**:

SCHEDA DI CONTROLLO

Coincide con una scheda di controllo appartenente al carteggio industriale **grifo®**, basata sui microprocessori della famiglia I86 come: **GPC® 188F**; **GPC® 188D**; **GPC® 884**; **GPC® 883**; ecc. La scheda di controllo, indipendentemente dalle richieste dell'applicazione da realizzare, deve essere dotata di:

- almeno 64K Byte di RAM
- una linea seriale asincrona in RS 232
- una EPROM oppure una FLASH EPROM con le seguenti indicazioni:

TDEB xxx	**FWR xxx**
Ver. ?.?	**Ver. ?.?**
zzM	**yyyK zzM**

dove:
- xxx = codice scheda
- ?.? = versione programma
- yyy = dimensione della FLASH EPROM (128K o 256K Byte)
- zz = frequenza di clock della scheda (20M, 26M o 40M)

Quanto sopra riportato è da intendersi come struttura minima di lavoro, infatti lo stesso sistema può essere espanso aumentando quindi le sue potenzialità. La scelta della configurazione della scheda di controllo deve comunque avvenire in relazione alle specifiche esigenze dell'applicazione che deve essere sviluppata.
In caso di acquisto contemporaneo di scheda di controllo e **GCTR**, il dispositivo di memoria EPROM o FLASH EPROM viene fornito già montato sulla scheda. Sull'etichetta di tale dispositivo sono presenti tutte le informazioni relative al tipo di scheda remota, alla versione del codice salvato, alle sue dimensioni ed alla frequenza di clock con una forma simile a quella sopra usata.

PERSONAL COMPUTERS

Il pacchetto **GCTR** necessita di un personal computer, che da ora in poi chiameremo **P.C. di sviluppo**, con le seguenti caratteristiche:

Personal Computer:	IBM compatibile (con CPU ≥ 386).
Memoria RAM:	Minimo 8M Bytes.
Sistema operativo:	WINDOWS 3.11, 95, 98, ME.
Monitor:	Colori.
Memorie di massa:	Drive per floppy disk da 3 pollici e 1/2. Hard Disk con almeno 4M Byte liberi.
Una Seriale:	COM 1 o 2 in RS 232, secondo specifiche V24 (in grado di gestire un Baud Rate di 115,2 KBaud)
Mouse:	Microsoft compatibile con relativo driver installato.

Un secondo personal computer, che da ora in poi chiameremo **P.C. di console**, é invece consigliato per poter utilizzare direttamente i programmi dimostrativi forniti e per avere una interfaccia utente tradizionale. Le uniche caratteristiche necessarie del P.C. di console sono una tastiera, un monitor, una linea seriale RS 232 ed un programma di comunicazione seriale in modo da poter svolgere le funzioni di un semplice terminale che rappresenta sul monitor quanto ricevuto e trasmette qunato premuto sulla tastiera.

CAVO DI COMUNICAZIONE SERIALE

Per le fasi di debug ed eventuale programmazione della FLASH EPROM della scheda di controllo é necessario effettuare un collegamento seriale tra una delle linee seriali del P.C. di sviluppo e la linea seriale A della scheda di controllo. Tale collegamento necessita solo dei segnali di ricezione, trasmissione e massa (RxD, TxD e GND) e deve avvenire seguendo le normative V24 del C.C.I.T.T. Un secondo cavo di comunicazione tra una delle linee seriali del P.C. di console e la linea B della scheda é eventualmente necessario se l'utente utilizza la console del **GCTR**. Tale cavo ha caratteristiche analoghe al precedente, come di seguito riportato:

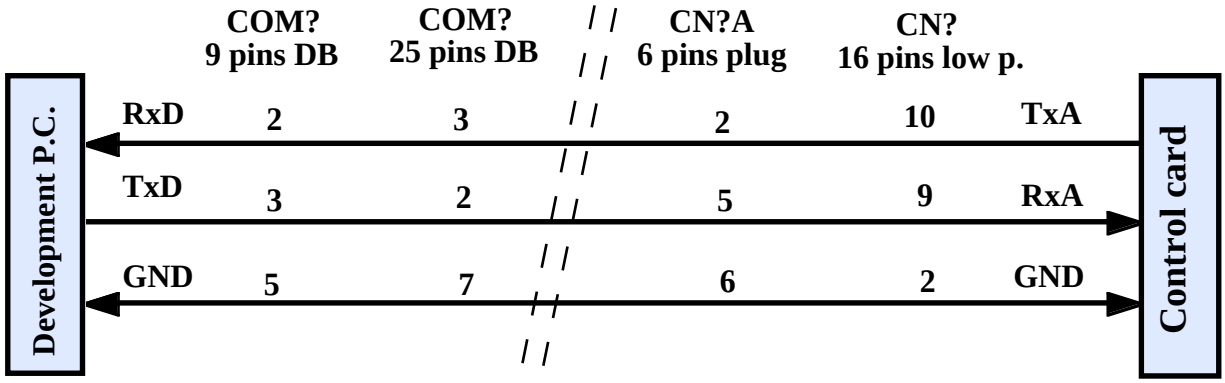

FIGURA 1: COLLEGAMENTO SERIALE TRA P.C. DI SVILUPPO E SCHEDA DI CONYROLLO

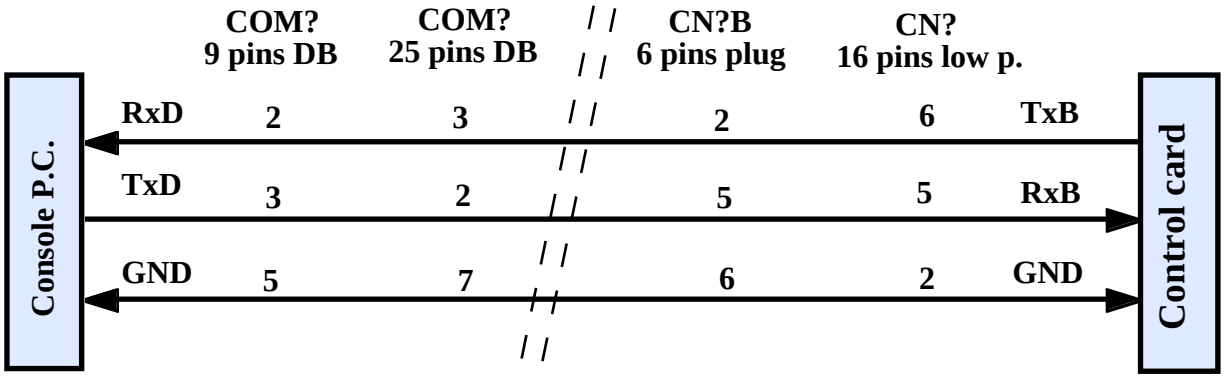

FIGURA 2: COLLEGAMENTO SERIALE TRA P.C. DI CONSOLE E SCHEDA DI CONTROLLO

Con le indicazioni **COM?** s'intene una delle linee di comunicazione seriale del P.C. mentre con **CN? 16 pins low p.** e **CN? 6 pins plug** s'intendono i connettori standardizzati della scheda di controllo **grifo®**, descritti nel relativo manuale tecnico. La tabella nella figura seguente riporta i nomi di tali connettori ed i codici degli accessori (cavi, schde, ecc.) che la **grifo®** é in grado di offrire per facilitare e velocizzare la fase di connessione. L'utente può quindi decidere se realizzare i cavi di collegamento

autonomamente oppure ordinarli direttamente alla **grifo®**.

SCHEDA DI CONTROLLO	CONNETTORE	CODICI ACCESSORI PER COLLEGAMENTO SERIALE
GPC® 188F	CN1	FLT 16+16; NCS 01; CCR 25+25 o CCR 25+9
GPC® 188D	CN1	FLT 16+16; NCS 01; CCR 25+25 o CCR 25+9
GPC® 883	CN3A, CN3B	CCR.PLUG25F o CCR.PLUG9F
GPC® 884	CN3A, CN3B	CCR.PLUG25F o CCR.PLUG9F

FIGURA 3: CONNETTORI SERIALI ED ACCESSORI DI COLLEGAMENTO

SOFTWARE DI LAVORO

Assieme all'hardware descritto per operare con il **GCTR** é necessario un software di lavoro con cui sviluppare e mettere a punto il programma applicativo. Tale software é composto da una serie di programmi e files, presenti nei dischi di distribuzione e può essere suddiviso in due gruppi principali, come di seguito descritto.

SOFTWARE PER SVILUPPO PROGRAMMA APPLICATIVO

Il **GCTR** si basa sul compilatore C, linker e debugger della Borland. Questi ultimi sono normalmente utilizzati per generare programmi applicativi per personal computer standard, su cui opera il sistema operativo MS DOS o WINDOWS. Al fine di evitare tutti i problemi relativi alle diverse versioni dei pacchetti software descritti, assieme al **GCTR** vengono forniti anche i pacchetti di compilazione e linker e quello di debug, nelle loro parti essenziali, già installati, configurati e quindi pronti all'uso. In questo modo l'uso del **GCTR** é notevolmente semplificato; é comunque indispensabile che l'utente si procuri i pacchetti Borland descritti in modo da regolarizzarne l'uso, da poterne consultare la documentazione e da poterne usare i vari programmi di utilità generale.
Riassumendo il software indispensabile per lo sviluppo programma applicativo comprende:

- Pacchetto software Borland TURBO C o Borland TURBO C++ o Borland C++ con relativa documentazione.
- Pacchetto software Borland TURBO DEBUGGER con relativa documentazione.

La versione ed il tipo di questi pacchetti non é importante per quanto già detto sopra.
E' invece consigliabile, ma non indispensabile, un programma di comunicazione generico in grado di gestire una classica emulazione terminale, con un protocollo fisico di comunicazione impostabile, sul P.C. di console. A questo scopo si ricordano i famosi, e diffusi, programmi CROSS TALK, PROCOMM, BITCOMM, TERMINAL, HYPERMINAL, ecc. od il **GET51** disponibile nel sito e/o nel CD della **grifo®**.

SOFTWARE E FIRMWARE PER SCHEDA DI CONTROLLO

Il **GCTR** si basa su una serie di programmi e files che si preoccupano di rendere romabile il programma applicativo scritto in C, tramite un linguaggio di programmazione che genera codice per sistemi basati su sistema operativo. Questo compito é svolto tramite librerie, codice di partenza, debugger remoti, programmi di supporto, utilità, ecc. che variano al variare delle caratteristiche hardware della scheda di controllo usata e sono realizzati e forniti dalla **grifo®**. Per maggiori informazioni sul software e firmware per la scheda di controllo, fare riferimento al sucessivo paragrafo "DIRECTORY C:\GCTRxxx".

MANUALE UTENTE GCTR

E' questo manuale che riporta tutte le informazioni tecniche relative al sistema operativo **GCTR**. In particolare si possono trovare connessioni hardware, sintassi dei comandi, descrizione delle librerie, procedure e programmi di supporto, organizzazione della memoria, ecc.

PROGRAMMATORE DI EPROM

Un programmatore di EPROM in grado di programmare file presenti sul P.C. di sviluppo é necessario per completare l'applicazione realizzata. Infatti il codice generato, una volta debuggato e provato in tutte le sue parti, deve essere salvato permanentemente su una EPROM da montare sulla scheda di controllo.
Da ricordare che il programmatore di EPROM é necessario solo se il **GCTR** in uso non è su FLASH EPROM in quanto in quest'ultimo caso l'operazione di salvataggio sulla stessa FLASH é autonomamente svolta dalla scheda di controllo tramite il P.C. di sviluppo ed un apposito firmware di programmazione (**FWR xxx**) già incluso nel **GCTR**.

UTILIZZO DEL GCTR

Per utilizzare correttamente il **GCTR** é necessario compiere una serie di operazioni sequenziali e non, come descritto nei seguenti paragrafi. Al fine di verificare il corretto funzionamento dell'intero pacchetto software e di ottenere un sistema pronto all'uso in breve tempo, seguire le informazioni riportate nel sucessivo capitolo "COME INIZIARE".

INSTALLAZIONE

Da ricordare che il **GCTR** é un pacchetto software in cui tutte le operazioni per la creazione del programma applicativo, a parte naturalnte il debug, vengono effettuate sul P.C. di sviluppo, non sulla scheda di controllo e per questa ragione é conveniente che l'utente scelga un P.C. potente, veloce e sicuro su cui effettuare l'installazione.
Viene di seguito riportata la serie di operazioni che l'utente deve effettuare per installare correttamente il pacchetto software **GCTR**. E' importante che i seguenti passi siano tutti effettuati nell'ordine in cui sono riportati:

1- Installare sul P.C. di sviluppo, a meno che non sia già stato fatto, il pacchetto software Borland TURBO C o TURBO C++ o C++ seguendo le indicazioni dello stesso pacchetto.
Questo passo é opzionale, ma consigliato per facilitare le sucessive operazioni di scrittura e verifica sintattica dei programmi applicativi da sviluppare.
2- Verificare che l'hard disk del P.C. di sviluppo siano dsponibili almeno 4M Bytes liberi.
3- Inserire il disco 1 "**GCTR xxx**" nel floppy disk drive del P.C. di sviluppo e da questo lanciare il programma di installazione INSTALL.EXE, con un doppio click sulla sua icona.
4- Leggere la finestra informativa presentata dal programma d'installazione, premere il pulsante "Next" per proseguire.
5- A questo punto il programma d'installazione verifica i requisiti hardware e software del P.C. di sviluppo e se sufficienti prosegue, altrimenti avvisa delle mancanze riconosciute invitando ad eliminarle.
6- Attendere la fine della sucessiva fase di ricerca di un ambiente di sviluppo Borland (I.D.E.) sull'hard disk del P.C. di sviluppo. Al termine di questa ricerca viene presentato l'eventuale programma individuato oppure l'editor standard EDIT.COM nel caso che nessun compilatore C della Borland sia già installato. Qualora l'utente non gradisca l'ambiente di sviluppo proposto, lo può modificare manualmente digitando il percorso del suo programma preferito nella finestra visualizzata, oppure modificando il file GCTR.IDE, come descritto nel paragrafo "CONFIGURAZIONI UTENTE", anche al termine dell'installazione.
7- Selezionare la seriale che s'intende utilizzare sul P.C. di sviluppo (COM1 o COM2).
8- Selezionare le due dimensioni di memoria (CODE AREA SIZE e DATA AREA SIZE) che si vogliono dedicare al **GCTR**, facendo attenzione che tali dimensioni siano rispettivamente inferiori od uguali alla quantità di EPROM/FLASH EPROM e SRAM presenti sulla scheda di controllo.
9- A questo punto il programma d'installazione inizia a copiare il software di lavoro su hard disk. In questa fase un'apposita finestra riporta il file attualmente copiato ed una barra a scorrimento che indica la percentuale di lavoro svolta. In corrispondenza delle richieste visualizzate, inserire i sucessivi dischi di "**GCTR xxx**" e premere un tasto per proseguire.
10- Attendere il termine della fase di copia evidenziata da una finestra che informa della corretta riuscita dell'installazione e quindi premere il pulsante "Next" per continuare.
11- Compilare le opzioni della sucessiva finestra d'installazione del **GET188**, provvedendo a

riselezionare la seriale del P.C. di sviluppo (COM1 o COM2), il baud rate di comunicazione (115200), la lingua di rappresentazione (Italiano o Inglese), il tipo di monitor usato ed il nome dell'utilizzatore e della ditta. Una volta effettuate tutte le scelte descritte premere il pulsante "Install".

12- Leggere gli aggiornamenti non ancora riportati sul manuale che vengono visualizzati in questa fase in un'apposita finestra. Si ricorda che tali aggiornamenti sono salvati nel file READ.ME e che possono essere consultati e stampati anche in un secondo tempo.

13- Verificare che sull'hard disk del P.C. di sviluppo siano state correttamente create le tre directory: C:\TC_GCTR, C:\TD_GCTR e C:\GCTRxxx e che queste contengano i file descritti nei paragrafi seguenti.

14- A questo punto l'installazione é terminata.

Per completezza, segue una breve descrizione di tutti i files installati sull'hard disk del P.C. di sviluppo; le informazioni d'uso sono invece riportate nel prossimo capitolo.

DIRECTORY C:\TC_GCTR

Tale directory contiene 9 files e/o programmi che permettono di compilare e linkare il sorgente C del programma applicativo per ottenere l'eseguibile. La documentazione di questi files la si trova nella manualistica Borland C++, TURBO C, TURBO C++ .

DIRECTORY C:\TD_GCTR

Tale directory contiene 15 files e/o programmi che permettono di debuggare il programma applicativo che si stà sviluppando; da ricordare che il debugger previsto nel pacchetto software **GCTR** é un potentissimo debugger simbolico a livello sorgente in grado di gestire direttamente l'hardware della scheda di controllo**,** breakpoint, trace, visualizzazione delle strutture dati, ecc. tramite modalità semplici ed intuitive con rappresentazioni a finestre multiple, menù a tendina, tasti funzione, ecc. Anche in questo caso, la documentazione dei files la si trova nella manualistica Borland TURBO DEBUGGER.

DIRECTORY C:\GCTRXXX

Al termine dell'installazione tale directory contiene 80 files e/o programmi che permettono di utilizzare tutto il pacchetto software **GCTR**. L'utente deve sempre lavorare all'interno di questa directory in quanto i files qui salvati permettono l'accesso alle altre directory del pacchetto e possono essere svolte tutte le operazioni necessarie (edit, compilazione, link, debug, preparazione immagine EPROM, ecc) sul programma applicativo.
In particolare i files presenti sono:

STDEB.OBJ ->
STBIN.OBJ -> Sono i cosiddetti file con il codice di partenza per la scheda di controllo da linkare al programma applicativo durante la fase di debug e durante la generazione dell'immagine binaria; tali codici provvedono a settare ed inizializzare opportunamente l'hardware in uso, in modo che il main C del programma applicativo possa prendere il controllo della scheda. Tali file variano al variare della scelta memorie effettuata al punto 8 dell'installazione.

LCTR_T.LIB ->	E' un file di libreria per la scheda di controllo da utilizzare durante il link del programma applicativo, dove é salvato il codice che consete di romare il programma sviluppato a partire dall'equivalente codice per sistema operativo.
CL.LIB ->	E' la libreria standard della Borland, per il modello di memoria Large, modificata in alcune funzioni che ne consentono l'utilizzo sulla scheda di controllo; anche questo file, deve essere utilizzato in fase di link. Per maggiori informazioni fare riferimento ai capitoli sucessivi.
EMU,MATL.LIB ->	Sono file di libreria che contengono le procedure matematiche del C nella versione con coprocessore matemetico emulato, necessarie per gestire operazioni in floating point e per usare la ricca serie di funzioni matematiche.
*.H ->	Sono una serie di files header in cui sono salvate le dichiarazioni delle funzioni di libreria Borland e possono essere inclusi nel main del programma applicativo che ne fà uso, seguendo le regole della documentazione Borland.
LOC.EXE ->	E' un programma che consente di trasformare l'eseguibile .EXE creato a seguito della compilazione e del link, nella corrispondente immagine binaria da utilizzare per la programmazione della EPROM o della FLASH EPROM per la scheda di controllo,
EXETOBIN.LOC ->	E' un file che definisce i parametri di trasformazione per il LOC.EXE relativi alla configurazione memorie della scheda di controllo. Naturalmente questo file varia al variare della configurazione memorie selezionata al punto 8 dell'installazione.
CTODEB.* ->	Costituiscono un programma di supporto che provvede a lanciare in sequenza l'I.D.E. Borland o il text editor, il compilatore, il linker ed il debugger per il programma applicativo indicato.
CTOBIN.* ->	Costituiscono un programma di supporto che proovede a lanciare in sequenza il compilatore, il linker ed il locator per il programma applicativo indicato.
FLASHWR.* ->	Costituiscono un programma di supporto che consente di programmare la FLASH EPROM della scheda.
GCTR.IDE ->	E' un file di testo che contiene il percorso del programma di I.D.E. o text editor che deve essere utilizzato durante lo sviluppo del programma applicativo. Il CTODEB provvede a lanciare il programma qui indicato.
TDxxx.IMG ->	E' l'immagine binaria del programma eseguibile sulla scheda di controllo che si preoccupa di comunicare con il TURBO DEBUGGER e quindi gestisce tutta la fase di messa a punto del programma applicativo. Questo file varia al variare della configurazione memorie selezionata al punto 8 dell'installazione.
GET188.EXE ->	E' il programma di emulazione terminale intelligente utilizzato dal FLASHWR per programmare la FLASH EPROM.
G188HELP.HLP ->	File di help in linea del programma **GET188**.
GET188IN ->	Programma di installazione del **GET188**.
GHEX2.COM ->	E' un programma di utility che consente di trasformare un file binario nell'equivalente file in formato HEX Intel.
INSTALL.LOG ->	E' un file di testo che contiene l'elenco di files installati.
UNINSTALL.EXE ->	E' il programma di disinstallazione del **GCTR** e può essere usato per eliminare i file installati dall'hard disk del P.C. di sviluppo.
*.C ->	Sono una serie di programmi dimostrativi direttamente utilizzabili con il **GCTR** sulla scheda di controllo.
READ.ME ->	E' il file che contiene gli ultimi aggiornamenti non ancora riportati sul manuale.

USO

Qui segue la descrizione d'uso del pacchetto software **GCTR**, facendo riferimento alle precedenti informazioni; si ricorda che la procedura d'uso é stata semplificata rendendola utilizzabile da tutti gli utenti programmatori che conoscono il C.
I passi che seguono devono essere svolti per ottenere un programma applicativo scritto in C, completamente debuggato, da installare sulla scheda di controllo e devono essere eseguiti nell'ordine riportato. Per completezza vengono indicate sia le modalità d'uso da MS-DOS che WINDOWS, quando queste si differenziano:

1 - Portarsi nella directory C:\GCTRxxx dell'hard disk del P.C. di sviluppo.

2 - Verificare ed eventualmente modificare le configurazioni utente del GCTR relative all'editor ed all'I.D.E Borland C, come descritto nel paragrafo "CONFIGURAZIONI UTENTE".

3 - Lanciare il programma di supporto CTODEB:
*digitando C:\GCTRxxx>**CTODEB <nomefile>.C<ENTER>*** *da MS-DOS*
*trascinando l'icona del **<nomefile>.C** sull'icona del **CTODEB*** *da WINDOWS*
con cui viene eseguito l'I.D.E. Borland o l'editor specificato in fase d'installazione, sul file <nomefile>.C. In questo ambiente l'utente deve sviluppare il programma C seguendo le regole proprie del linguaggio di programmazione, facendo attenzione a non modificare il nome del sorgente C su cui opera; terminata la scrittura del programma applicativo (o della parte di questo che s'intende provare) se si stà utilizzando l'I.D.E., é conveniente verificarne la correttezza sintattica selezionando il comando di compilazione. A questo punto si può uscire dall'I.D.E., o dall'editor, ed automaticamente il programma applicativo scritto viene compilato e linkato con gli appositi programmi della directory C:\TC_GCTR. E' importante controllare l'assenza di errori durante questa fase e se presenti bisogna terminare l'esecuzione del CTODEB e ripetere il punto 3 dall'inizio eseguendo le correzioni dei suddetti errori. Se la compilazione ed il linker si concludono con successo, viene automaticamente lanciato il debugger dalla directory C:\TD_GCTR. A questo punto, usando tutte le potenzialità del TURBO DEBUGGER Borland, si deve verificare la correttezza funzionale del programma applicativo realizzato provandolo direttamente sull'hardware finale, ovvero sulla scheda di controllo collegata al resto dell'eventuale elettronica ed al P.C. di sviluppo.
A verifica effettuata, l'utente può uscire dal debugger e ritornare al controllo del sistema operativo del P.C. di sviluppo visto che il CTODEB termina in coincidenza della fine del debugger. Durante l'esecuzione di questa fase vengono creati o modificati i files: <nomefile>.OBJ, <nomefile>.MAP,<nomefile>.EXE, <nomefile>.C, <nomefile>.BAK ma di questi, solo il sorgente <nomefile>.C riguarda l'utente.

4 - Se dalla verifica di correttezza effettuata alla fine del puto 3, non emerge alcun errore, l'utente può proseguire passando al punto sucessivo, viceversa deve ripetere il punto 3 fino alla completa verifica di tutte le parti e modalità del programma. Naturalmente gli errori individuati con il debugger vengono risolti dall'utente tramite modifiche al sorgente del programma applicativo che si sta sviluppando.

5 - Terminata la fase di debug, si prosegue con il salvataggio del programma applicativo sulla scheda di controllo. Questa operazione, che é completamente automatica, viene effettuata lanciando il programma di supporto CTOBIN:
*digitando C:\GCTRxxx>**CTOBIN <nomefile>.C<ENTER>*** *da MS-DOS*
*trascinando l'icona del **<nomefile>.C** sull'icona del **CTOBIN*** *da WINDOWS*
dove <nomefile>.C coincide con il nome del file sorgente C usato al punto 3. Anche in questo caso l'utente dovrà controllare l'assenza di errori ed al termine, sarà presente il file <nomefile>.IMG che coincide con la codifica binaria del codice con cui programmare la

EPROM o la FLASH EPROM per la scheda di controllo; in questa fase viene creato anche il file <nomefile>.ABM.

6 - L'utente deve procedere con la programmazione della EPROM o della FLASH EPROM con il file <nomefile>.IMG. Per una dettagliata descrizione delle modalità di esecuzione di queste operazioni, fare riferimento ai paragrafi PROGRAMMAZIONE EPROM e PROGRAMMAZIONE FLASH EPROM.

7 - A questo punto in caso di **GCTR** su EPROM l'utente deve spegnere la scheda di controllo, smontare la EPROM "**TDEB xxx ...**" e sostituirla con quella programmata al punto 6; in caso di **GCTR** su FLASH EPROM l'utente deve spegnere la scheda di controllo e quindi selezionare il RUN mode (vedere paragrafo "MODIFICA DI APPLICAZIONE GIA' INSTALLATA"). Una volta ricollegata ed alimentata la scheda, il programma applicativo parte automaticamente.

L'utente deve lavorare con il **GCTR** mantenendo i file del suo programma applicativo (sorgenti, include, macro, ecc.) nella directory C:\GCTRxxx; per semplificare l'operazione di trascinamento icone quando si lavora con WINDOWS si può eventualmente copiare sulla scrivania i collegamenti ai file di supporto CTODEB e CTOBIN e quindi usare queste copie.

PROGRAMMAZIONE EPROM

Le modalità di utilizzo del programmatore di EPROM non riguardano questa documentazione, quindi in questo paragrafo vengono riportate solo delle informazioni collaterali
Il file <nomefile>.IMG generato dal programma di supporto CTOBIN é un file binario di dimensione pari a quella della EPROM selezionata in fase d'installazione del **GCTR**; tale file deve essere quindi programmato a partire dall'indirizzo 00000H del dispositivo.
Se il programmatore in uso richiede il formato HEX Intel provvedere a trasformare il file binario descritto nell'equivalente HEX tramite il programma GHEX2.COM La sintassi di utilizzo del GHEX2 è:

*C:\GCTRxxx>**GHEX2 <nomefile>.IMG<ENTER>***

da fornire sotto il controllo dell'MS-DOS o nella finestra Avvio | Esegui di WINDOWS e da cui si ottiene il file <nomefile>.HEX nel formato HEX Intel esteso.
Nel caso in cui il programma applicativo da programmare faccia uso di altri dati da salvare in EPROM (dati di configurazione, messaggi, tabelle, ecc), quest'ultimi devono essere salvati al termine del codice precedentemente salvato. Visto che il file binario generato dal **GCTR** riempe l'intero spazio della EPROM, spetta all'utente individuare l'indirizzo di fine del codice come descritto nel paragrafo "MEMORIA RISERVATA".

PROGRAMMAZIONE FLASH EPROM: FLASH WRITER

Una delle caratteristiche di particolare interesse del **GCTR** è la possibilità di poter autonomamente gestire la FLASH EPROM montata a bordo scheda. Questa caratteristica facilita notevolmente lo sviluppo dell'applicazione, infatti non è più necessario un programmatore di EPROM esterno che viene sostituito dal semplice P.C. di sviluppo collegato in seriale alla scheda. Le fasi di aggiornamento, verifica, manutenzione del software che la scheda stà eseguendo possono essere eseguite comodamente anche sul campo, ad esempio con l'aiuto di un semplice P.C. portatile.

La gestione della FLASH EPROM con il **GCTR** è una operazione assistita in cui l'utente tramite un apposito programma é in grado di modificare il contenuto di alcune aree della FLASH, partendo da files salvati su tutti i drives del P.C. di sviluppo. Tutte queste operazioni sono ad alto livello e sono corredate di messaggi di aiuto che assistono l'utente in tutte le fasi.
Si ricorda che al fine di garantire l'integrità dei dati salvati in FLASH EPROM e di assicurare sempre la presenza del programma FLASH WRITER, questo occupa sempre l'ultimo settore del componente e su questo può essere salvato solo dalla **grifo®**. L'utente può ottenere altre FLASH EPROM di lavoro o da installare su applicazioni finite, ordinandole con il codice **FWR xxx** o **FWR xxx.512K**, come riportato nel capitolo "VERSIONI GCTR".

CONFIGURAZIONE SCHEDA

Per gestire correttamente il FLASH WRITER, l'utente deve effettuare le seguenti configurazioni hardware:

1 - Collegare la linea seriale A della scheda di controllo alla linea seriale del P.C.di sviluppo, scelta durante la fase d'installazione, sfruttando il cavo di comunicazione descritto in figura 1.
2 - Montare una FLASH EPROM: "**FWR xxx ...**", sull'apposito zoccolo della scheda.
3 - Montare almeno 128K Byte di RAM sugli appositi zoccoli della scheda.
4 - Configurare i jumpers della scheda di controllo in base alla sua configurazione hardware e selezionare il DEBUG mode, seguendo le indicazioni del manuale tecnico o la sucessiva figura 4.
N.B. Per la **GPC® 188F** e **GPC® 188D** non collegare i jumpers J16 e J17.

AREE DELLA FLASH EPROM

Facendo riferimento alle figure 5, 6 solo le aree contrassegnate con **(*)** possono essere modificate dal programma FLASH WRITER. Viene di seguito riportata una breve descrizione di tali aree:

1- Area FLASH WRITER: coincide con gli ultimi 16K Bytes della FLASH e contiene il codice del programma di gestione FLASH. La scheda a seguito di un reset od un power on riparte sempre con l'esecuzione di questo codice, immediatamente seguita dalla verifica dello stato di RUN o DEBUG e dalla sucessiva esecuzione del programma applicativo salvato (RUN mode) o del FLASH WRITER (DEBUG mode). Tale area non può essere modificata dall'utente in alcun modo, al fine di evitare situazioni errate che pregiudicherebbero il funzionamento stesso della scheda.
2- Area non usata: é eventualmente presente sulle FLASH EPROM in cui le dimensioni del settore sono superiori ai 16K Bytes dell'area FLASH WRITER. Le dimensioni di quest'area sono quindi variabili (ad esempio 0K Bytes per FLASH da 128Kx8 e 48K Bytes per FLASH da 512Kx8), ma comunque corrisponde sempre ad un area non utilizzabile per nessuna operazione.
3- User Area: occupa lo spazio rimanente tra la somma delle due precedenti aree e le dimensioni della FLASH (ad esempio 112K Bytes per FLASH da 128Kx8, 448K Bytes per FLASH da 512Kx8) e può contenere codice e/o dati come il programma applicativo, dati di configurazione, messaggi, tabelle, ecc. In RUN mode la scheda parte sempre eseguendo il codice presente all'inizio di quest'area.

Nella User Area possono essere salvati uno o più files binari salvati sui dispositivi di memoria di massa del P.C. di sviluppo (floppy disk, hard disk, ecc) precedentemente generati dal **GCTR** o da altri pacchetti software. Questi files possono essere scritti su FLASH EPROM a partire da un indirizzo specificato dall'utente fino ad esaurimento dei files od al riempimento dell'area. La User Area non può essere scritta due volte con dati differenti (in questo caso un errore di malfunzionamento FLASH EPROM viene visualizzato), quindi la User Area deve essere prima cancellata.
Il programma applicativo sviluppato con il **GCTR** deve essere sempre salvato all'inizio della User Area in modo da essere eseguito in RUN mode.

Per ulteriori informazioni sulle aree di memoria descritte si faccia riferimento ai paragrafi ORGANIZZAZIONE DELLA MEMORIA.

ESECUZIONE DEL FLASH WRITER

Per eseguire correttamente il programma di gestione della FLASH EPROM, si devono eseguire i seguenti passi, con le solite distinzioni in caso d'uso da MS-DOS o da WINDOWS, quando necessarie.

1- Lanciare il programma **GET188** sul P.C. di sviluppo collegato alla scheda:
*digitando C:\GCTRxxx>**GET188 /T<ENTER>*** *da MS-DOS*
*effettuando un doppio click sull'icona **FLASHWR*** *da WINDOWS*
ed attendere la scomparsa della finestra di presentazione che viene visualizzata.

2- Preparare la scheda di controllo come indicato nel precedente paragrafo "CONFIGURAZIONE SCHEDA", quindi resettare od accendere la scheda in modo da eseguire il FLASH WRITER.

3- A questo punto il FLASH WRITER parte, presentando la sua versione, il size della FLASH, l'inizio dell'area libera della FLASH e la prima pagina di help.

4- Leggere attentamente le pagine di help, selezionandole con i tasti "N" e "P" e quindi proseguire l'esecuzione premendo il tasto "ENTER".

5- Selezionare l'operazione richiesta premendo il tasto numerico associato ("0"÷"3"), come indicato nel menù che compare.

6- Se viene scelta l'operazione di scrittura della User Area (tasto "0"), l'utente deve inserire il nome del file da scrivere in quest'area. Questo file deve avere un formato binario (.IMG, .BIN, ecc.) salvato sulla directory corrente del drive attuale. Una volta inserito il file, il programma ne verifica la presenza ed in caso affermativo prosegue, viceversa richiede se si desidera reinserire un nuovo nome file.
In caso di file trovato, l'utente deve inserire l'indirizzo di segmento da cui la programmazione del file scelto deve iniziare; automaticamente il programma presenta già il primo indirizzo libero della FLASH che può quindi essere confermato o modificato, inserendolo in esadecimale maiuscolo. Ad indirizzo inserito il FLASH WRITER verifica che questo sia compreso nella User Area ed in caso affermativo prosegue, viceversa richiede se si desidera reinserire un nuovo indirizzo.
In caso di indirizzo valido viene verificato se il file selezionato può essere completamente programmato nella User Area a partire dall'indirizzo specificato ed in caso affermativo prosegue, viceversa richiede una conferma di programmazione con troncamento della parte finale che eccede dalla User Area. Se l'utente conferma la programmazione il programma prosegue, altrimenti tutta l'operazione viene abortita.

La sucessiva fase di scrittura è illustrata da un apposito messaggio di stato che mostra l'indirizzo attualmente in programmazione; in questa fase l'utente deve solo attendere il completamento dell'operazione e poi verificarne l'esito.
Il nome file richiesto dal programma é nel formato **<drive>:<nome>.<estensione>** e come drive possono essere utilizzati tutti quelli gestiti dal P.C. di sviluppo. Nel caso in cui il file da selezionare non si trovi nella directory attuale, si deve selezionare quella corretta tramite l'opzione "File | Change dir..." del **GET188**.

7- Se viene scelta l'operazione di cancellazione della User Area (tasto "1") compare una richiesta di conferma alla cancellazione ed in caso affermativo viene cancellata l'intera User Area della FLASH, viceversa l'operazione viene abortita.
In caso di cancellazione confermata l'utente deve solo attendere il completamento dell'operazione, il cui proseguimento é segnalato da puntini che vengono rappresentati sul monitor, e poi verificarne l'esito.
Questa operazione è normalmente utilizzata per cancellare il precedente contenuto della FLASH EPROM e rendere quindi possibile la sucessiva programmazione con nuovi files; la cancellazione é comunque definitiva e deve essere quindi selezionata e confermata con estrema attenzione.

8- Se viene scelta l'operazione di rappresentazione delle finestre di help (tasto "2") vengono ripresentate le finestre di aiuto su richiesta dell'utente come indicato al punto 4.

9- Se viene scelta l'operazione di uscita (tasto "3") il FLASH WRITER presenta un messaggio che informa della fine esecuzione e sulle possibili operazioni sucessive.

10- Durante l'esecuzione nella maggioranza delle fasi è sempre possibile interrompere l'esecuzione del programma premendo il tasto "ESC" con cui si termina definitivamente l'esecuzione del FLASH WRITER. La pressione di questo tasto é equivalente alla scelta dell'opzione di uscita.

11- In ogni fase del programma vengono sempre verificati i possibili mafunzionamenti (errore di accesso al file system, errore di cancellazione FLASH, errore di scrittura FLASH, ecc.) e nel caso se ne presenti uno, è immediatamente presentato da un apposito messaggio informativo.

12- Uscire dal programma **GET188** sul P.C. di sviluppo premendo contemporaneamente i tasti <ALT>+<X>, con cui si ritorna al controllo del sistema operativo.

MODIFICA APPLICAZIONE INSTALLATA: SVILUPPO ED ESECUZIONE

Una richiesta molto comune in tutti i sistemi é quella di poter facilmente intervenire sui programmi applicativi in modo da aggiornarli, modificarli o verificarne il funzionamento quando questi sono già operativi. Il **GCTR** risponde a questa richiesta fornendo la possibilità di effettuare queste operazioni di aggiornamento, modifica e verifica in modo semplice ed efficace, sempre sfruttando il solo P.C. di sviluppo. La tecnica fornita consiste nell'interrompere l'esecuzione del programma applicativo già installato (modo ESECUZIONE) e ritornare nella condizione di verifica e modifica descritta nel paragrafo USO (modo SVILUPPO). Entrambe le modalità di ESECUZIONE e SVILUPPO sono già state ampiamente descritte nei paragrafi precedenti, quindi in questo paragrafo si descrive solo il passaggio da una modalità all'altra, distinguendo la versione **GCTR** in EPROM da quella in FLASH EPROM.

***GCTR** in EPROM*

- Il modo ESECUZIONE é selezionato montando sulla scheda di controllo la EPROM con il programma applicativo ottenuta seguendo le indicazioni del paragrafo PROGRAMMAZIONE EPROM.

- Il modo SVILUPPO é selezionato montando sulla scheda di controllo l'apposita EPROM: "**TDEB xxx ...**" e ricollegando la linea seriale A della scheda al P.C. di sviluppo.

***GCTR** in FLASH EPROM*

- Il modo ESECUZIONE é selezionato prima settando la modalità DEBUG sulla scheda, poi cancellando la User Area della FLASH EPROM, quindi salvando il programma applicativo all'inizio della User Area ed infine riselezionando la modalità RUN.
- Il modo SVILUPPO é selezionato prima settando la modalità DEBUG sulla scheda, poi cancellando la User Area della FLASH EPROM, quindi programmando il programma di comunicazione con il TURBO DEBUGGER (TDxxx.IMG), all'inizio della User Area ed infine riselezionando la modalità RUN.

  Le operazioni di cancellazione e programmazione della User Area devono essere effettuate tramite il FLASH WRITER seguendo le indicazioni del paragrafo PROGRAMMAZIONE FLASH EPROM. Sia in modo ESECUZIONE che SVILUPPO una volta scelto il programma da salvare all'inizio della User Area, si dovrà confermare il salvataggio con troncamento dello stesso file.
  La selezione del modo operativo (RUN/DEBUG) avviene tramite la configurazione di un apposito jumper, come descritto nella seguente tabella:

SCHEDA	JUMPER DI RUN DEBUG
GPC® 188F	**J18**
GPC® 188D	**J18**
GPC® 883	**J1**
GPC® 884	**J1**

FIGURA 4: TABELLA JUMPERS SELEZIONE RUN E DEBUG MODE

Dove: jumper non connesso -> seleziona modalità RUN
jumper connesso -> seleziona modalità DEBUG

N.B. A seguito di ogni variazione dello stato del jumper di selezione RUN e DEBUG mode, la scheda di controllo deve essere sempre riaccesa o resettata, infatti il FLASH WRITER verifica lo stato di questo jumper solo alla sua partenza.

Il passaggio tra i due modi operativi é senza dubbio più comodo nel caso di **GCTR** in FLASH EPROM, infatti in questo caso non si deve intervenire fisicamente sulla scheda di controllo se non per variare lo stato di un comodo jumpers.
Il **GCTR** in FLASH EPROM é quindi consigliabile nella fase di messa a punto dell'applicazione o comunque per produzione di pochi sistemi, mentre il **GCTR** in EPROM é senz'altro da preferire nel caso di grosse produzioni con un programma applicativo stabile.

COME INIZIARE

In queso capitolo vengono descritte quelle che sono le operazioni da effettuare per un primo elementare utilizzo del pacchetto **GCTR**. In particolare viene riportata la giusta sequenza di operazioni con degli esempi pratici di utilizzo. In caso di incomprensioni, fare riferimento ai precedenti capitoli in cui ogni operazione viene illustrata in modo più approfondito. Gli esempi riportati in questo paragrafo sono relativi ad un **GCTR** in FLASH EPROM per una **GPC® 884** con 128K FLASH EPROM ("**FWR xxx ...**") e 128K RAM.

1- Leggere tutta la documentazione ricevuta.

2- Installare un pacchetto di programmazione in C della BORLAND
Esempio: installare il Borland C++ Ver. 3.1 seguendo le informazioni e le possibilità del relativo programma d'installazione.

3- Installare il **GCTR** selezionando: l'ambiente di sviluppo (I.D.E.), la seriale del P.C. di sviluppo e la configurazione memorie presente a bordo scheda.
Esempio: confermare la scelta dell'I.D.E della Borland BC.EXE ; selezionare la COM1 nella finestra di richiesta linea seriale; selezionare dimensione della CODE AREA SIZE a 128K; selezionare dimensione della DATA AREA SIZE a 128K Bytes; al termine della copia, riselezionare la COM1 nella finestra d'installazione del **GET188** ed inserire i dati dell'utilizzatore.

4- Montare la EPROM "**TDEB xxx ...**" o la FLASH EPROM "**FWR xxx ...**" sull'apposito zoccolo della scheda e selezionare la modalità di SVILUPPO.
Esempio: montare la FLASH EPROM "**FWR 884 ...**" sullo zoccolo IC 5; chiudere il jumper J1; lanciare il FLASHWR sul P.C. di sviluppo; alimentare la scheda. Con il programma FLASH WRITER cancellare la User Area della FLASH EPROM, quindi programmare il programma TD884.IMG all'inizio della User Area (E000H) e confermando il suo troncamento; uscire dal **GET188** premendo <ALT>+<X>; aprire il jumper J1 e riaccendere la scheda.

5- Collegare la linea seriale A della scheda di controllo alla linea seriale selezionata del P.C. di sviluppo, provvedendo a collegare opportunamente, secondo lo standard RS 232, i segnali GND, TxD ed RxD. Per ulteriori informazioni a riguardo di questo collegamento fare riferimento alla figura 1 o ai manuali tecnici della scheda e del P.C.
Esempio: effettuare un cavo a tre fili che colleghi rispettivamente i pin 2,3,5 di un connettore a vaschetta DB 9 femmina ai pin 2,5,6 di un plug 6 vie maschio; collegare tale cavo al CN3A della **GPC® 884** ed al connettore COM1 del P.C. di sviluppo.

6- Collegare la linea seriale B della scheda di controllo alla linea seriale selezionata del P.C. di console, provvedendo a collegare opportunamente, secondo lo standard RS 232, i segnali GND, TxD ed RxD. Per ulteriori informazioni a riguardo di questo collegamento fare riferimento alla figura 2 o ai manuali tecnici della scheda e del P.C.
Esempio: effettuare un cavo a tre fili che colleghi rispettivamente i pin 2,3,5 di un connettore a vaschetta DB 9 femmina ai pin 2,5,6 di un plug 6 vie maschio; collegare tale cavo al CN3B della **GPC® 884** ed al connettore COM1 del P.C. di console.

7- Lanciare sul P.C. di console un programma di comunicazione seriale e configurare il protocollo fisico di comunicazione a 19200 Baud, 8 Bit x chr, 1 stop Bit, No parity sulla porta collegata al punto 6.
Esempio: lanciare il programma di comunicazione **GET51** (disponibile nel sito e/o CD della **grifo®**); settare i parametri sopra riportati nell'apposita finestra "Option | Serial Port" ed attivare la comunicazione con il comando "Option | Terminal".

8- Lanciare sul P.C. di sviluppo il programma di supporto per il debugger sul programma di apprendimento TEST.C:
*digitando C:\GCTRxxx>**CTODEB TEST.C<ENTER>*** *da MS-DOS*
*trascinando l'icona del file **TEST.C** sull'icona **CTODEB*** *da WINDOWS*
e verificare che venga eseguito l'I.D.E.selezionato al punto 3 e che venga aperto il programma TEST.C. Quest'ultimo é il classico programma che provvede a rappresentare sulla console i numeri primi inferiori a 100 ed ha una semplice funzione dimostrativa e didattica.
Esempio: lanciare il programma di supporto per il debugger come sopra indicato.

9 - Se é stato selezionato l'I.D.E. Borland, continuare a leggere, altrimenti saltare al punto 12. Configurare l'I.D.E. Borland selezionando la segnalazione di tutti warnings e selezionando il modello di memoria large.
Esempio: Selezionare l'opzione "All" nella finestra "Display warnings" visualizzata con il comando "Options|Compiler|Messages". Selezionare l'opzione “Large” nella finestra “Model” visualizzata con il comando “Options|Compiler|Code generation”. Infine salvare i settaggi effettuati con il comando “Options|Save” e confermando con "OK".

10- Esaminare il programma TEST.C senza modificarlo usando i comandi dell'editor ed in seguito lanciare la compilazione con l'apposita opzione dell'I.D.E. Borland.
Esempio: scorrere il programma con i tasti freccia; verificare le potenzialità dei comandi nei menù edit e search senza modificare il programma; compilare il programma selezionando l'opzione "Compile | Compile".

11- Correggere l'errore segnalato durante la compilazione: alla riga 48 del programma c'é un errore di battitura, ovvero é stata persa una s e quindi "flag[i]" deve diventare "flags[i]". Una volta aggiunta la lettera ripetere la compilazione e se non vengono riscontarti errori procedere al punto sucessivo.
Esempio: portarsi sulla riga 48 in cui il compilatore ha individuato l'eerore ed aggiungere la "s" mancante; ripetere la compilazione descritta al punto 10.

12- Una volta verificata e corretta la sintassi del programma TEST.C uscire dall'I.D.E., verificare l'assenza di errori durante la sucessiva fase di ricompilazione e link, attendere la partenza del TURBO DEBUGGER e confermare la trasmissione del programma sulla linea seriale, come richiesto da un'apposita finestra che compare.
Esempio: eseguire le operazioni sopra descritte confermando da tastiera con il tasto <ENTER> o con il mouse premendo il pulsante "Yes", in corrispondenza del messaggio "Program out of date on remote, send over link?"

13- Attendere la fine della fase di trasmissione evidenziata dalla comparsa del sorgente del programma TEST.C sul monitor e selezionare il comando di esecuzione "Run | Run". Così facendo sul monitor del P.C. di console comparirà l'elenco dei numeri primi compresi tra 1 e 100 di cui ne possiamo verificare l'esattezza e la completezza.
Da questo esame appare subito la mancanza del primo numero primo = 1: il programma TEST.C ha un errore funzionale da individuare.
Esempio: eseguire le operazioni sopra descritte.

14- Interrompere l'esecuzione del programma con i tasti <CTRL>+<BREAK>, chiudere la finestra CPU che compare con <ALT>+<F3>, ricaricare il programma con il comando "Run | Program Reset", posizionarsi con il cursore al termine del ciclo di determinazione dei numeri primi (riga 61) e quindi fornire il comando "Run | Go To Cursor". Il programma viene eseguito fino al ciclo di rappresentazione dei risultati, quindi a questo punto il vettore booleano flags[ ] già contiene lo stato di numero primo per i primi 100 numeri. Con il comando "Data | Add watch" e specificando la variabile "flags" compare una finestra che mostra il contenuto del vettore flags[ ]; su questa si può facilmente verificare che la cella di indice 1 é correttamente

settata, quindi la mancanza individuata al punto 13 é da attribuire alla sucessiva rappresentazione non alla determinazione dei numeri primi.
Proseguendo l'esecuzione a passi tramite ripetuti comandi di "Run | Trace Into" si determina subito che il ciclo di rappresentazione ha un indice di inizio sbagliato, infatti il primo numero primo stampato é quello di indice "i=2".
Esempio: eseguire le operazioni sopra descritte.

15- Per correggere l'indice di inizio si deve uscire dal TURBO DEBUGGER con il comando "File | Quit" e ripetere il punto 8, correggere la riga 63 con "i=1" e rieffettuare in sequenza i punti 10 e 12. In questo caso anche il numero primo 1 verrà visualizzato sul P.C. di console e quindi il programma é stato completamente testato e verificato.
Esempio: eseguire le operazioni sopra descritte.

16- Il programma completamente testato ottenuto al punto precedente ora può essere salvato in EPROM o FLASH EPROM della scheda remota in modo che parta automaticamente all'accensione, anche senza il P.C. di sviluppo. Per effettuare questo salvataggio, prima lanciare sul P.C. di sviluppo il programma di supporto che provvede a generare l'immaggine binaria del programma:
*digitando C:\GCTRxxx>**CTOBIN TEST.C<ENTER>*** *da MS-DOS*
*trascinando l'icona del file **TEST.C** sull'icona **CTOBIN*** *da WINDOWS*
con cui il file TEST.IMG viene creato nella directory di lavoro.
Esempio: lanciare il programma di supporto come sopra indicato.

17- A questo punto si deve selezionare il modo ESECUZIONE:
in caso di **GCTR** in EPROM si deve programmare il file TEST.IMG in una EPROM vergine di dimensioni pari a quelle definite al punto 3 tramite un programmatore di EPROM collegato al P.C. di sviluppo;
in caso di **GCTR** in FLASH EPROM si deve spegnere la scheda di controllo; selezionare il DEBUG mode; lanciare il programma **GET188** sul P.C. di sviluppo
*digitando C:\GCTRxxx>**GET188 /T<ENTER>*** *da MS-DOS*
*effettuando un doppio click sull'icona **FLASHWR*** *da WINDOWS*
rialimentare la scheda di controllo; attendere la partenza del programma FLASH WRITER; confermare l'esecuzione premendo <ENTER>; selezionare l'opzione di cancellazione della User Area (premendo <1>) e confermare l'operazione premendo <Y>; attendere la fine cancellazione e premere un tasto per tornare al menù principale del FWR; selezionare l'opzione di scrittura della User Area premendo <0>; digitare il nome del file da salvare: C:TEST.IMG<ENTER>; confermare l'indirizzo di programmazione presentato premendo <ENTER>; confermare la programmazione con troncamento premendo <Y>; attendere la fine programmazione e premere un tasto per tornare al menù principale; selezionare l'opzione di uscita dal FWR premendo <3>; uscire dal **GET188** premendo <Alt>+<X>.
In entrambi i casi terminata l'operazione di programmazione il P.C. di sviluppo non é più necessario e può essere anche scollegato dalla scheda di controllo.
Esempio: eseguire le operazioni sopra descritte ricordando che il DEBUG mode é selezionato collegando il jumper J1 della **GPC® 884** e che in fse di programmazione l'indirizzo da confermare é = E000.

18- Spegnere la scheda di controllo e completare la selezione del modo ESECUZIONE che
in caso di **GCTR** in EPROM equivale a sostituire la EPROM della scheda di controllo con quella otteuta al punto 16;
in caso di **GCTR** in FLASH EPROM equivale a selezionare il RUN mode, aprendo l'apposito jumper.
Esempio: spegnere la **GPC® 884** e scollegare il jumper J1.

19- Riaccendere la scheda di controllo e verificare il programma applicativo realizzato parta

automaticamente, ovvero osservare che sul P.C. di console vengano nuovamente rappresentati i numeri primi compresi tra 1 e 100. La generazione del primo programma applicativo é così terminata.

Esempio: eseguire le istruzioni sopra descritte.

DESCRIZIONE GCTR

Seguono una serie di informazioni relative al pacchetto **GCTR** di carattere generale che devono essere utilizzate dall'utente durante la fase di sviluppo del programma applicativo per utilizzare al meglio la scheda di controllo. Tutte le differenze relative all'hardware della scheda sono naturalmente illustrate separatamente.

CODICE DI PARTENZA

Con codice di partenza s'intende quella parte di codice che viene sempre eseguita dalla scheda di controllo dopo un reset od un power on e che provvede a preparare tutte le condizioni necessarie alle fasi sucessive. Le principali operazioni effettuate dal codice di partenza sono di seguito elencate:

1- Predispone la scheda in termini di configurazione di memoria.
2- Predispone la scheda in termini di configurazioni dell'I/O.
3- Disabilita le circuiterie che possono influenzare l'esecuzione del programma come il watch dog interno, gli interrupts, il DMA, ecc.
4- Retriggera la circuiteria di watch dog esterna.
5- Inizializza un'opportuno stack di lavoro.
6- Azzera l'area per tutte le variabili globali.
7- Inizializza l'emulatore floating point.
8- Copia l'area dati inizializzati da EPROM o FLASH EPROM a RAM.
9- Installa un gestore di interrupt 21H per intercettare e segnalare eventuali chiamate al sistema operativo inaspettate.
10- Salta alla prima istruzione del main del programma applicativo in C.

Il codice di partenza del **GCTR** é indispensabile per qualsiasi programma applicativo in C e non può essere sostituito da quelli standard della Borland che sono basati su una esecuzione tramite sistema operativo MS-DOS o WINDOWS. Tale codice deve essere sempre utilizzato anche nella fase di sviluppo del programma infatti il TURBO DEBUGGER é in grado di eseguirlo senza problemi anche se la scheda di controllo non é stata appena resettata o riaccesa.
Il codice di partenza, relativo alla configurazione di memoria scelta in fase d'installazione, viene fornito compilato nei file STDEB.OBJ, STBIN.OBJ e viene linkato nel programma applicativo in modo completamente automatico dagli appositi programmi di supporto CTODEB e CTOBIN.

INDIRIZZAMENTO STRUTTURE HARDWARE IN I/O

Il codice di startup che viene sempre eseguito prima del main del programma applicativo in C, provvede anche a settare ed inizializzare i registri del microprocessore che riguardano il controllo dell'I/O (RELOC, PACS).
Un settaggio comune a tutte le schede di controllo é quello che setta gli indirizzi interni del microprocessore, in I/O a partire da **FF00H**, e quello che setta le periferiche presenti sulla scheda sempre in I/O agli indirizzi riportati nell'APPENDICE C di questo manuale.
Questi indirizzi, ed in particolare quelli delle periferiche, devono essere utilizzati direttamente dall'utente nel programma applicativo C e sono inoltre utilizzate dalle librerie romate fornite con il **GCTR**; non possono essere quindi variati per nessuna ragione.

LOCATOR

Il Locator del **GCTR** consente di allocare il codice ed i dati in ogni punto dello spazio di memoria convenzionale della scheda di controllo. Esso opera sui file eseguibili standard (.EXE) e sui .MAP file generati dal Borland C/C++ e crea un file binario in uscita che puo' essere programmato in EPROM o FLASH EPROM. Se necessario, puo' generare un file in formato Intel OMF assoluto, utilizzato dalla maggioranza degli emulatori.
I comandi di allocazione hanno un formato leggibile; essi indicano i file che devono essere usati dal Locator, gli indirizzi in ROM, gli indirizzi in RAM dei segmenti ed infine le aree di memoria disponibili. Tali comandi non devono essere definiti dall'utente infatti sono forniti gia' salvati nel file EXETOBIN.LOC, durante l'installazione del **GCTR**.
Il Locator esegue delle verifiche incrociate durante l'allocazione. Verifica infatti i trabocchi di ROM, le sovrapposizioni di codice, le sovrapposizioni dei dispositivi RAM e ROM, i fuori sincronismo dei file .EXE e .MAP, la completa allocazione del programma, ecc. Al termine del lavoro il Locator rappresenta inoltre la quantita' di ROM utilizzata, sia in percentuale che in numero di byte attualmente usati.
Il Locator genera un file .MAP assoluto, con estensione .ABM, che riporta l'indirizzo di tutte le strutture dati e le funzioni pubbliche; questo ha lo stesso formato del file .MAP generato dal linker della Borland ma con gli indirizzi modificati.

FLOATING POINT

Nel **GCTR** e' compreso un preciso ed efficiente floating point con le relative funzioni matematiche e trigonometriche del C. Per le schede **grifo®** non provviste di coprocessore, la gestione matematica del Borland C/C++ coincide inevitabilmente con una emulazione software del floating point.
Gli errori matematici sono gestiti da una apposita funzione romata che termina l'esecuzione del programma applicativo e restituisce un codice numerico d'errore univoco, in modo da fornire informazioni sull'accaduto agli utenti. Un apposito programma d'esempio denominato DEMOFP.C, illustra le modalità di utilizzo di buona parte delle funzioni disponibili.

BREAKPOINT HARDWARE

Nella modalità di SVILUPPO del programma applicativo, l'utente dispone di due breakpoint che gli danno la possibilità di riprendere sempre il controllo della scheda di controllo, anche quando il programma applicativo che questa esegue, entra in loop infiniti o altre situazioni non previste.
Il primo breakpoint é quello classico del TURBO DEBUGGER, attivabile in ogni momento dalla tastiera del P.C. di sviluppo premendo contemporaneamente i tasti <CTRL>+<BREAK> ed é un interrupt software associato all'interrupt di ricezione della linea seriale A della scheda di controllo.
Il secondo invece é un vero e proprio breakpoint hardware associato al Not Mascherable Interrupt (/NMI) della scheda di controllo. L'attivazione dell'/NMI la si ottiene anche tramite un semplice pulsante che una volta premuto colleghi questo segnale (presente sui vari connettori della scheda) alla massa di alimentazione. A differenza del precedente breakpoint, che può essere disattivato via software, quest'ultimo é sempre attivo e quindi sempre utilizzabile. Una seconda differenza tra questi breakpoint é che il primo ferma il programma applicativo nel punto in cui questo era in esecuzione, mentre il secondo termina completamente l'esecuzione del programma che quindi deve essere ricaricato per eventuali fasi sucessive di debug. Naturalmente nessun breakpoint é disponibile e/o attivo nel modo ESECUZIONE del programma applicativo.

CONFIGURAZIONI UTENTE

Come descritto nei precedenti paragrafi il **GCTR** é caratterizzato da una serie di configurazioni con cui l'utente può definire una serie di funzionalità del pacchetto di sviluppo. Per completezza in questo paragrafo vengono descritte tutte queste configurazioni definendo per ognuna il significato e le modalità di settaggio:

- *EDITOR:* coincide con il programma di editor con cui l'utente scrive e/o modifica il sorgente del suo programma applicativo quando utilizza il CTODEB. E' preferibile utilizzare l'editor incluso nell'I.D.E. del Borland C in quanto comprende alcune caratteristiche come la codifica a colori, la verifica della correttezza sintattica, un help in linea sul C, ecc. La scelta dell'editor avviene in fase d'intstallazione oppure modificando il file ascii GCTR.IDE in cui é appunto salvato il percorso ed il nome del programma da utilizzare.
- *I.D.E. Borland C:* qualora sia stato selezionato l'I.D.E. del Borland C come editor del **GCTR** e l'utente intende sfruttare la sua possibilità di verificare la correttezza sintattica del programma applicativo scritto (comando Compile), si deve prima configurare lo stesso I.D.E. in termini di modello di memoria e livello di warnings. In dettaglio si deve manualmente selezionare il modello di memoria **Large** tra le opzioni del compilatore ed attivare tutti i messaggi di allarme (**All**) nelle opzioni dei messaggi. Queste configurazioni devono essere effettuate sfruttando le apposite modalità dell'I.D.E., come descritto nella sua documentazione, edevono essere salvate per renderle permanenti (vedere esempio al punto 9 del capitolo "COME INIZIARE").
- *CODE AREA SIZE:* coincide con la dimensione dell'area di memoria usata dal **GCTR** per il codice del programma applicativo. E' settabile solo ed esclusivamente in fase d'installazione.
- *DATA AREA SIZE:* coincide con la dimensione dell'area di memoria usata dal **GCTR** per i dati, lo stack e l'heap del programma applicativo. E' settabile solo ed esclusivamente in fase d'installazione.
- *Seriale P.C. sviluppo:* coincide con la linea seriale del P.C. di sviluppo collegata alla scheda di controllo con cui si effettua il debug ed il salvataggio in FLASH EPROM del programma applicativo. E' settabile solo ed esclusivamente in fase d'installazione.

WATCH DOG ESTERNO

Il **GCTR** si occupa autonomamente del retrigger periodico della circuiteria di watch dog esterno presente su tutte le schede di controllo. In questo modo l'utente che intende sviluppare un applicazione che fà uso del watch dog esterno può tranquillamente collegare tale circuiteria e non preoccuparsi del suo retrigger in tutte le fasi in cui il **GCTR** é in esecuzione (codice di partenza, TURBO DEBUGGER, librerie con un lungo tempo di esecuzione). Naturalmente quando invece é in esecuzione il programma applicativo sviluppato dall'utente spetta a quest'ultimo il compito di retriggerare periodicamente la circuiteria di watch dog affinché non intervenga resettando la scheda. Al fine di evitare retrigger indesiderati a seguito dell'intervento della circuiteria di watch dog quando il programma applicativo é in esecuzione, si deve sempre:

- riprendere la comunicazione con il TURBO DEBUGGER tramite un breakpoint da tastiera <CTRL>+<BREAK>;
- uscire dal TURBO DEBUGGER sul P.C. di sviluppo;
- resettare o rialimentare la scheda di controllo;
- riprendere il normale sviluppo del programma applicativo.

ORGANIZZAZIONE DELLA MEMORIA

Fondamentalmente in tutti i programmi scritti in C si distinguono tre aree di memoria fondamentali: l'area per il **codice**, l'area per i **dati** e l'area per lo **stack** e l'**heap** mentre sulla scheda di controllo normalmente sono presenti solo due tipi di memoria: la **ROM** (EPROM o FLASH EPROM) di sola lettura e la **RAM** di lettura e scrittura. Il **GCTR** si occupa di organizzare tutta la memoria presente sulla scheda di controllo, rendendola disponibile per il programma applicativo nel modo desiderato dall'utente. All'atto dell'installazione infatti, il **GCTR** richiede le dimensioni dell'area codice (in EPROM o FLASH EPROM) e dell'area dati+stack+heap (in RAM) e si configura autonomamente con le selezioni effettuate; per questo durante il lavoro non é neccessario preoccuparsi della configurazione hardware della scheda, ma é sufficiente rispettare le indicazioni fornite in questo paragrafo, limitandosi ad utilizzare solo le aree consentite.
Nelle schede di controllo **grifo®** basate sui microprocessori della famiglia I186, é possibile programmare le dimensioni e gli indirizzi dei dispositivi di memoria via software, tramite il settaggio di una serie di registri interni al microprocessore e l'eventuale circuiteria di MMU a bordo scheda. Il **GCTR** setta questi registri (UMCS, MPCS, MMCS, LMCS, MMU), nel codice di partenza che a sua volta é configurato in fase d'installazione. Il codice di partenza viene sempre eseguito dalla scheda di controllo dopo un reset od un power on, perciò la giusta configurazione di memorie é sempre garantita in ogni condizione operativa. Se le necessità di memoria del programma applicativo variano é sufficiente reinstallare il **GCTR** e ricompilare lo stesso applicativo senza nessuna ulteriore modifica.
Le seguenti figure illustrano le possibili configurazioni di memoria supportate dal **GCTR**:

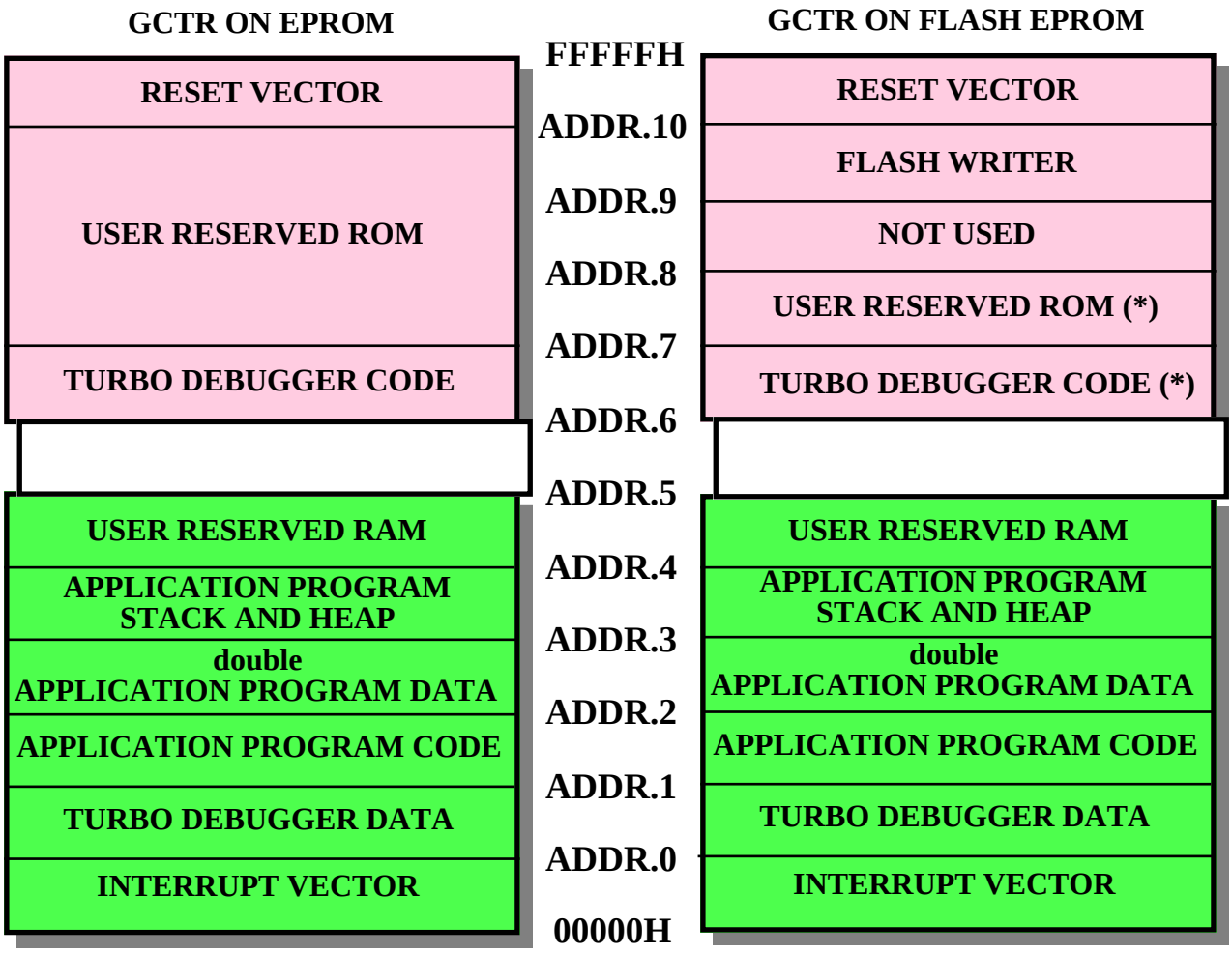

FIGURA 5: CONFIGURAZIONE MEMORIE IN MODO SVILUPPO

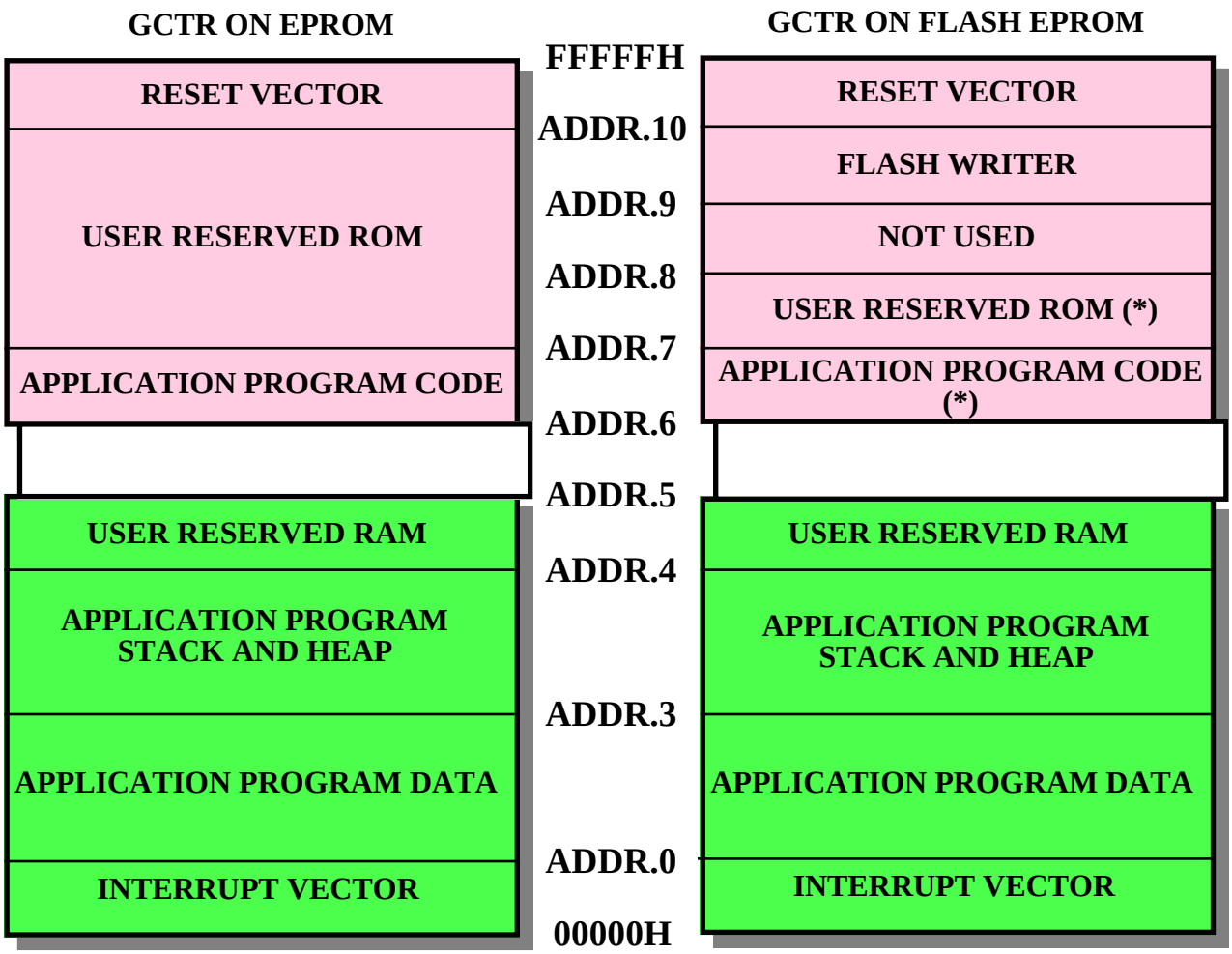

FIGURA 6: CONFIGURAZIONE MEMORIE IN MODO ESECUZIONE

I valori degli indirizzi riportati nelle figure 5 e 6 variano al variare della configurazione memorie presente sulla scheda di controllo e sulle dimensioni delle aree di memoria selezionate in fase d'installazione, come descritto nelle seguenti tabelle.

Dimensione area DATA da installazione	ADDR.0	ADDR.1	ADDR.2	ADDR.3	ADDR.4
64 Kbyte	00400H	00FC0H	?????H	?????H	10000H
128 Kbyte	00400H	00FC0H	?????H	?????H	20000H
256 Kbyte	00400H	00FC0H	?????H	?????H	40000H
512 Kbyte	00400H	00FC0H	?????H	?????H	80000H

FIGURA 7: VALORI INDIRIZZI MEMORIA RAM DA INSTALLAZIONE

Configurazione RAM su scheda	ADDR.5
128 Kbyte	10000H
256 Kbyte	20000H
512 Kbyte	80000H

FIGURA 8: VALORI INDIRIZZI MEMORIA RAM DA CONFIGURAZIONE SCHEDA

Dimensione area CODE da installazione	ADDR.6	ADDR.7	ADDR.8	ADDR.9	ADDR.10
128 KByte	E0000H	?????H	FC000H	FC000H	FFFF0H
256 Kbyte	C0000H	?????H	F8000H	FC000H	FFFF0H
512 Kbyte	80000H	?????H	F0000H	FC000H	FFFF0H

FIGURA 9: VALORI INDIRIZZI MEMORIA ROM DA INSTALLAZIONE

Configurazione ROM su scheda	ADDR.6
128 Kbyte	E0000H
256 Kbyte	C0000H
512 Kbyte	80000H

FIGURA 10: VALORI INDIRIZZI MEMORIA ROM DA CONFIGURAZIONE SCHEDA

Per gli indirizzi non definiti (?????H) non può essere specificato un valore preciso in quanto i size delle aree che delimitano variano al variare del programma applicativo in sviluppo e possono essere quindi stabiliti solo dall'utente.

NOTE SU USO MEMORIA

- Le informazioni sulle dimensioni sia dell'area dati che dell'area codice del programma applicativo possono essere ottenute dai file <nomefile>.MAP e <nomefile>.ABM generati dallo stesso **GCTR** durante il suo normale lavoro.
- La **User Area** a cui il paragrafo PROGRAMMAZIONE FLASH EPROM fà riferimento coincide con l'area della FLASH EPROM delimitata dai due indirizzi ADDR.6 e ADDR.8, come denota l'***** che identifica le parti che possono essere modificate dall'utente tramite il FLASH WRITER.
- Per il **GCTR** su FLASH EPROM, la scelta di avere un area non usata di dimensioni variabili al variare del size della memoria é legata all'organizzazione fisica della FLASH utilizzata. Infatti quest'ultima è divisa in 8 settori equivalenti e per ragioni di sicurezza sul codice del FLASH WRITER si é deciso di proteggere l'ultimo settore, rendendolo quindi non utilizzabile. Questa sicurezza và a scapito di una riduzione della User Area che é comunque un limite accettabile.
- Il **GCTR** utilizza il modello di memoria **large** in modo da disporre delle massime dimensioni per tutte le aree del programma applicativo. Si ricorda per completezza che il segmento DGROUP non può comunque eccedere i 64K complessivi, come descritto nella documentazione del Borland C.

- Come indicato in figura 5 il TURBO DEBUGGER raddoppia l'area dati del programma applicativo che salva in RAM, per un suo uso interno. Riassumendo sulla scheda di controllo deve essere installata una quantità di SRAM pari a: 0FC0H + codice applicativo + dati applicativo + dati applicativo + stack ed heap + eventuale memoria riservata.
- Nel modo SVILUPPO la quantità di RAM utilizzata é notevolmente superiore a quella necessaria nel modo ESECUZIONE, infatti nel primo caso il TURBO DEBUGGER salva sulla RAM anche altre aree del programma applicativo in sviluppo. Per semplicità, sicurezza ed ottimizzazione costi si può quindi usare una scheda di controllo con la massima configurazione di SRAM per la messa a punto del programma (modo SVILUPPO) e poi in produzione (modo ESECUZIONE) usare invece solo la configurazione necessaria.

MEMORIA RISERVATA

Una delle richieste più comuni nello sviluppo di applicazioni per l'automazione industriale é quella di disporre di aree memoria riservate, a completa disposizione dell'utente, in cui possono essere gestiti trasferimenti DMA, salvati dati acquisiti dal campo, memorizzate tabelle di parametri e/o messaggi, ecc.
Con il **GCTR** é possibile avere aree riservate sia in RAM che in ROM semplicemente montando sulla scheda di controllo una quantità di memoria sufficiente per il programma applicativo più aree riservate ed installando il **GCTR** con delle dimensioni di area dati e codice sufficienti per il solo programma applicativo. Così facendo tutta la memoria presente sulla scheda ma non dichiarata nell'installazione, é automaticamente riservata. Più in dettaglio il **GCTR** in nessuna condizione accede autonomamente ad indirizzi RAM nel range ADDR.4÷ADDR.5 ed ad indirizzi ROM nel range ADDR.7÷ADDR.10 nel caso di EPROM o nel range ADDR.7÷ ADDR.8 nel caso di FLASH EPROM.

Ad esempio, se l'utente necessita di 256 KBytes di RAM per il salvataggio di dati trasferiti in DMA e di 50+100 KBytes di RAM per l'area dati+stack ed heap del suo programma applicativo, dovrà configurare la scheda di controllo con 512 KBytes di RAM ed installare il **GCTR** con una dimensione dell'area dati di 256 KBytes. Così facendo il **GCTR** ed il programma applicativo sviluppato non accederenno mai ad indirizzi ≥ ADDR.4 = 40000H ed a partire da questo indirizzo l'utente potrà tranquillamente gestire i trasferimenti in DMA.

Se invece l'utente deve salvare 10 KBytes di messaggi in quattro lingue diverse (per un totale di 40 KBytes) sulla ROM ed ha un programma applicativo di 50 KBytes, dovrà configurare la scheda di controllo con 128 KBytes di EPROM o FLASH EPROM, installare il **GCTR** con un'area codice di 128 KBytes e salvare i messaggi a partire da F0000H. Quest'ultimo indirizzo é stato scelto 14 KBytes dopo ADDR.7 = EC800H in modo da lasciare libertà di crescita al programma applicativo, senza dover spostare i messaggi.

Per conoscere le dimensioni delle aree dati e codice usati dai programmi sviluppati con il **GCTR**, da sommare alle dimensioni delle aree riservate, in modo da poter correttamente installare il pacchetto e correttamente selezionare la quantità di memoria a bordo scheda, l'utente può esaminare il file .ABM generato dal programma di supporto CTOBIN. Tale file contiene infatti l'elenco di tutti i segmenti del programma con le relative lunghezze in bytes.
Nel caso in cui le configurazioni disponibili non soddisfano completamente le esigenze dell'applicazione, contattare drettamente la **grifo®**.

GESTIONE CONSOLE

Il sistema di sviluppo **GCTR** si occupa della gestione di una serie di interfaccie operatore che possono essere gestite con le istruzioni ad alto livello del C, dedicate al comando della console. L'aspetto dell'interfacciamento operatore é sempre uno dei problemi più pesanti in tutti i programmi applicativi, quindi la disponibilità di strumenti già pronti che facilitino questa operazione semplifica il lavoro dell'utente e riduce i tempi di sviluppo.
Il dispositivo di console gestito dal **GCTR** può essere associato a dispositivi hardware predisposti per l'interfaccia operatore, come stampanti, terminali seriali, display alfanumerici, LEDs di stato, tastiere a matrice, ecc. Tra questi si possono ricordare sia prodotti **grifo®** (come **QTP xxx**, **QTP xxP**, **KDx x24**, **DEB 01**, **IAC 01**, ed altri) che quelli di terze parti.
All'interno del programma applicativo la gestione della console avviene tramite apposite istruzioni ad alto livello del C (cputs(), cprintf(), cscanf(), ecc.) che vanno a chiamare altrettante funzioni presenti nella libreria CL.LIB; attualmente sono presenti numerose funzioni che consentono di attivare i LEDs di stato con attributi e di acquisire e/o rappresentare dati sia numerici che alfanumerici, anche in modalità formattata su tutti i dispositivi hardware previsti. Per una dettagliata descrizione di tali funzioni di console fare riferimento all'APPENDICE B di questo manuale, in cui sono riportate la definizione, la descrizione dei parametri, ed un esempio di utilizzo.
Normalmente per gestire correttamente la console con il **GCTR** devono essre sempre effettuate in sequenza, le sucessive operazioni:

a) inizializzare il dispositivo hardware usato compatibilmente con l'eventuale sistema di console collegato (operazione necessaria solo in caso di sistemi seriali);
b) selezionare il dispositivo harware usato in ingresso;
c) selezionare il dispositivo hardware usato in uscita;
d) usare i dispositivi selezionati con le funzioni di libreria ad alto livello.

Si ricava immediatamente che con un programma in **GCTR** si possono gestire anche più dispositivi di ingresso ed uscita, provvedendo ad inizializzarli tutti ed a selezionare il dispositivo prima di utilizzarlo, anche se precedentemente ne era già selezionato uno diverso. La distinzione di dispositivo d'ingresso ed uscita consente inoltre di usare sistemi hardware diversi consentendo ad esempio di acquisire dati da una linea seriale e di rappresentarli su una stampante parallela.

DISPOSITIVI HARDWARE DI CONSOLE

Nelle librerie del **GCTR** é prevista la gestione di una serie di dispositivi hardware di console, che possono essere usati in operazioni di ingresso e/o uscita e che usano alcune risorse della scheda, come indicato nella seguente tabella:

Dispositivo hardware di console	I/O	Risorse usate su scheda di controllo
Terminali seriali, **QTP xxx**, P.C., ecc	I/O	Linea seriale A
Terminali seriali, **QTP xxx**, P.C., ecc	I/O	Linea seriale B
Display e tastiera esterna (**KDx x24**)	I/O	16 linee digitali di I/O
Display, tastiera e LEDs (**QTP xxP**)	I/O	16 linee digitali di I/O
Stampante parallela (**IAC 01**, **DEB 01**)	O	16 linee digitali di I/O

FIGURA 11: DISPOSITIVI HARDWARE DI CONSOLE

I dispositivi che usano 16 linee digitali di I/O sono provvisti di un connettore standardizzato e possono essere quindi collegati direttamente alla scheda remota, seguendo le indicazioni della figura 12, che include il collegamento anche della tensione di alimentazione. Nel caso in cui le interfaccie disponibili nel carteggio **grifo®** non soddisfino le esigenze dell'utente é possibili realizzare delle proprie interfaccie operatore seguendo le indicazioni dell'APPENDICE A in cui sono riportati gli schemi elettrici di alcune di queste interfaccie.

SCHEDA DI CONTROLLO	CONNETTORE	CAVO DI CONNESSIONE
GPC® 188F	CN2	FLAT 20+20
GPC® 188D	CN2	FLAT 20+20
GPC® 883	CN5	FLAT 20+20
GPC® 884	CN5	FLAT 26+20

FIGURA 12: COLLEGAMENTO DISPOSITIVI DI CONSOLE

Per avere ulteriori informazioni sui dispositivi di console nominati in questo paragrafo e sulle loro possibili configurazioni e potenzialità, fare riferimento all'apposita documentazione disponibile nel sito o nel CD della **grifo®**.

SIMBOLI PREDEFINITI PER CONSOLE

IL **GCTR** comprende un file di intestazione denominato GCLIBD.H, che include la definizione di una serie di simboli da utilizzarsi per la gestione ad alto livello del dispositivo di console:

QTP16P	->	Identifica il dispositivo hardware di console **QTP 16P**	(*)
QTP24P	->	Identifica il dispositivo hardware di console **QTP 24P**	(*)
KDxx24	->	Identifica il dispositivo hardware di console **KDx x24**	(*)
SER0	->	Identifica il dispositivo hardware di console seriale B	
SER1	->	Identifica il dispositivo hardware di console seriale A	
PRINTER	->	Identifica il dispositivo hardware di console stampante parallela	
LCD20x2	->	Identifica un display LCD da 20 caratteri su 2 righe	(#)
LCD20x4	->	Identifica un display LCD da 20 caratteri su 4 righe	(#)
LCD40x2	->	Identifica un display LCD da 40 caratteri su 2 righe	(#)
VFD20x2	->	Identifica un display fluorescente da 20 caratteri su 2 righe	(#)

Oltre a questi simboli nello stesso file sono presenti i prototipi di tutte le funzioni di libreria per la console a disposizione dell'utente; quest'ultime sono abbondantemente descritte in APPENDICE B. Si ricorda che tali simboli possono essre usati direttamente come parametri delle funzioni di libreria e che i simboli contrassegnati con (*) devono essre combinati in or logico con i simboli contrasseganti con (#) in modo da definire completamente il dispositivo di console utilizzato. Altre combinazioni di simboli non sono invece ammesse e causano un malfunzionamento delle librerie di console.
Per ulteriori informazioni relative all'uso dei simboli predefiniti fare riferimento all'apposito programma dimostrativo DEMOCONS.C che é in grado di gestire tutti i dispositivi di console supportati dal **GCTR**.

TASTIERA A MATRICE

Di seguito sono riportate le tabelle con i codici restituiti dal **GCTR** in corrispondenza della pressione di un tasto della tastiera a matrice del dispositivo di console. Per rendere generica la descrizione i tasti sono stati identificati dalla loro posizione nella matrice o, in altri termini, dai due segnali di riga e colonna presenti sul connettore della tastiera. Le tabelle di figura 13, 14, 15 e gli schemi elettrici di figura A2, A4, A5 consentono quindi di conoscere i codici dei tasti sia per le tastiere standard che per eventuali tastiere dell'utente.

PIN CN2 KDx x24	8	7	6	5	9	10
4	F = 70	E = 69	D = 68	C = 67	J = 74	N = 78
3	CR = 13	9 = 57	6 = 54	3 = 51	I = 73	M = 77
2	0 = 48	8 = 56	5 = 53	2 = 50	H = 72	L = 76
1	A = 65	7 = 55	4 = 52	1 = 49	G = 71	K = 75

FIGURA 13: CODICI TASTIERA KDX X24

PIN CN3 QTP 16P	8	7	6	5
4	D = 68	C = 67	B = 66	A = 65
3	# = 35	9 = 57	6 = 54	3 = 51
2	0 = 48	8 = 56	5 = 53	2 = 50
1	* = 42	7 = 55	4 = 52	1 = 49

FIGURA 14: CODICI TASTIERA QTP 16P

PIN CN3 QTP 24P	6	5	4	3	2	1
10	7 = 55	CR = 13	6 = 54	L = 76	H = 72	D = 68
9	ESC = 27	0 = 48	4 = 52	K = 75	G = 71	C = 67
8	5 = 53	9 = 57	3 = 51	J = 74	F = 70	B = 66
7	1 = 49	8 = 56	2 = 50	I = 73	E = 69	A = 65

FIGURA 15: CODICI TASTIERA QTP 24P

Quando il dispositivo hardware usato in ingresso coincide con una delle tastiere a matrice sopra descritte, sulla scheda di controllo oltre alle 16 linee di I/O viene usato anche il Timer 2 della CPU ed il relativo interrupt. Quest'ultimo é infatti usato per effettuare uno scanning periodico della tastiera in modo da riconoscere le eventuali pressioni di tasti con le classiche tecniche di debouncing ed autorepeat. In dettaglio il **GCTR** definisce:

tempo di debouncing	=	20 msec
tempo di autorepeat	=	100 msec
tempo di attesa autorepeat	=	500 msec
dimensione buffer tastiera	=	3 tasti

Con la gestione dell'autorepeat, se viene riconosciuto la pressione di un tasto per un tempo superiore a 500 msec il **GCTR** inizierà a salvare il suo codice nel buffer ogni 100 msec, fino a quando il tasto non viene rilasciato. Di ogni tasto premuto il codice viene salvato nel buffer di tastiera, organizzato in modalità FIFO, pronto per essere prelevato dalle funzioni di libreria; la pressione di più di tre tasti, senza prelevamento da parte del programma applicativo, implica quindi la perdita di tali caratteri in quanto fisicamente non salvabili nel buffer di tastiera.

COMANDI PER CONSOLE

In questo paragrafo vengono descritte tutte le sequenze di comando che possono essere utilizzate per usufruire di tutte le principali caratteristiche dei dispositivi hardware di console. Le funzioni di libreria del **GCTR** visualizzano tutti i caratteri aventi un codice compreso nel range **32÷255** (**20÷FF Hex**); se viene inviato un codice al di fuori di questo range, e questo non é un comando, viene ignorato. Il carattere viene visualizzato nella posizione attuale del cursore, e quest'ultimo avanzerà nella posizione successiva; se il cursore si trova sull'ultimo carattere del display (angolo in basso a destra), verrà posizionato nella posizione di Home (angolo in alto a sinistra).
Per ogni comando viene riportata una doppia descrizione dei codici che lo compongono: quella mnemonica, tramite caratteri ascii, e quella numerica espressa sia in forma decimale che esadecimale.
Tali comandi seguono lo standard **ADDS View-Point**, quindi tutte le sequenze iniziano con il carattere **ESC**, corrispondente al codice decimale 27 (**0x1B**). E' sottointeso che l'effetto dei comandi é subordinato al tipo di periferica hardware di console utilizzata; quindi ad esempio tutti i comandi elencati saranno correttamente gestiti da **QTP xxx** o **QTP xxP** ma non lo saranno su una stampante od un terminale seriale che non riconosce lo standard ADDS View-Point.
Si ricorda inoltre che per alcuni dei comandi descritti nei sucessivi paragrafi esistono anche delle funzioni di libreria che svolgono la stessa funzione: per maggiori informazioni si veda l'APPENDICE C di questo manuale.

CURSOR LEFT

Codice: ***21***
Codice Hex: ***0x15***
Mnemonico: ***NACK***

Il cursore viene spostato di una posizione a sinistra senza alterare il contenuto del display. Se il cursore si trova nella posizione di Home, verrà posizionato nell'ultimo carattere in basso a destra del display.

CURSOR RIGHT

Codice: ***6***
Codice Hex: ***0x06***
Mnemonico: ***ACK***

Il cursore viene spostato di una posizione a destra. Se il cursore si trova nell'ultimo carattere, in basso a destra del display, verrà posizionato nella posizione di Home.

CURSOR DOWN

Codice: ***10***
Codice Hex: ***0x0A***
Mnemonico: ***LF***

Il cursore viene posizionato nella riga successiva a quella in cui si trova, ma rimane nella stessa colonna. Se il cursore si trova nell'ultima riga del display, verrà posizionato nella prima riga del display.

CURSOR UP

Codice: ***26***
Codice Hex: ***0x1A***
Mnemonico: ***SUB***

Il cursore viene posizionato nella riga precedente a quella in cui si trova, ma rimane nella stessa colonna. Se il cursore si trova nella prima riga del display, esso verrà posizionato nell'ultima riga.

HOME

Codice: ***1***
Codice Hex: ***0x01***
Mnemonico: ***SOH***

Il cursore viene posto nella posizione di Home, corrispondente alla prima riga ed alla prima colonna del display, ovvero il carattere nell'angolo in alto a sinistra.

CARRIAGE RETURN

Codice: ***13***
Codice Hex: ***0x0D***
Mnemonico: ***CR***

Il cursore viene posizionato all'inizio della riga in cui si trova.

CARRIAGE RETURN + LINE FEED

Codice: ***29***
Codice Hex: ***0x1D***
Mnemonico: ***GS***

Il cursore viene posizionato all'inizio della riga successiva a quella in cui si trovava. Se il cursore si trova nell'utima riga del display, esso verrà posizionato all'inizio della prima riga, cioé nella posizione di Home.

POSIZIONAMENTO DEL CURSORE

Codice: ***27 89 r c***
Codice Hex: ***0x1B 0x59 r c***
Mnemonico: **ESC Y ASCII(r) ASCII(c)**

Il cursore viene posizionato nel punto assoluto, indicato tramite "r" e "c".
Questi codice esprimono i valori di riga e colonna del display, a cui va aggiunto un offset di **32** (**20 Hex**). Se ad esempio, si desidera posizionare il cursore nella posizione di Home (riga 0, colonna 0), sarà necessario inviare la sequenza:

27 89 32 32

Se i valori di riga e colonna non sono compatibili con il tipo di display installato, tale comando viene ignorato.

BACKSPACE

Codice: ***8***
Codice Hex: ***0x08***
Mnemonico: ***BS***

Il cursore si sposta a sinistra di un carattere, cancellando il contenuto della cella raggiunta.
Se il cursore si trova nella posizione di Home, verrà cancellato il carattere che si trova nell'ultima cella in basso a destra del display.

CLEAR PAGE

Codice: ***12***
Codice Hex: ***0x0C***
Mnemonico: ***FF***

Viene cancellato l'intero display ed il cursore viene posizionato in Home.

CLEAR LINE

Codice: ***25***
Codice Hex: ***0x19***
Mnemonico: ***EM***

Viene cancellata l'intera linea in cui si trova il cursore, e questo viene posto all'inizio di tale riga.

CLEAR END OF LINE

Codice: ***27 75***
Codice Hex: ***0x1B 0x4B***
Mnemonico: ***ESC K***

Vengono cancellati tutti i caratteri che si trovano nella riga in cui é posto il cursore, a partire dalla posizione del cursore stesso, fino al termine della riga. Il cursore rimane nella posizione in cui si trovava all'arrivo del codice di **Clear End Of Line**.
Se ad esempio, il cursore si trova all'inizio di una riga del display, verrà cancellata l'intera linea.

CLEAR END OF PAGE

Codice: ***27 107***
Codice Hex: ***1B 6B***
Mnemonico: ***ESC k***

Vengono cancellati tutti i caratteri dal punto in cui si trova il cursore, fino al termine del display. Il cursore rimane nella posizione in cui si trovava all'arrivo del codice di **Clear End Of Page**.
Se ad esempio, il cursore si trova nella posizione di Home, verrà cancellato l'intero display.

DISATTIVAZIONE DEL CURSORE

Codice: ***27 80***
Codice HEX: ***0x1B 0x50***
Mnemonico: ***ESC P***

Il cursore viene disattivato e non é più visibile.

ATTIVAZIONE DEL CURSORE FISSO

Codice: ***27 79***
Codice Hex: ***0x1B 0x4F***
Mnemonico: ***ESC O***

Il cursore viene attivato, quindi reso visibile, e rappresentato sotto forma di linea non lampeggiante posizionata al di sotto del carattere.
N.B. Questo comando non é disponibile quando é installato il display **fluorescente 20x4**.

ATTIVAZIONE DEL CURSORE "BLOCCO" LAMPEGGIANTE

Codice: ***27 81***
Codice Hex: ***0x1B 0x51***
Mnemonico: ***ESC Q***

Il cursore viene attivato, quindi reso visibile, e rappresentato sotto forma di rettangolo lampeggiante, visualizzato alternativamente con il carattere sovrapposto ad esso.

ATTIVAZIONE DI UN LED

Codice: ***27 50 n.LED Attr.***
Codice Hex: ***0x1B 0x32 n.LED Attr.***
Mnemonico: ***ESC 2 ASCII(n.LED) ASCII(Attr.)***

Viene attivato il LED indicato in "n.LED", con l'attributo specificato in "Attr.". I numeri dei LEDs sono compresi nel range **0÷15**, come rappresentato in figura B1.
Gli attributi disponibili sono i seguenti:

0 (00 Hex) -> LED disattivato
255 (FF Hex) -> LED attivato
85 (55 Hex) -> LED lampeggiante (blinking)

Se ad esempio, si vuole attivare il LED 5 con l'attributo di lampeggio, sarà necessario inviare la seguente sequenza:

27 50 5 85

Se il parametro con il numero del LED, o quello con l'attributo, non sono validi, il comando viene ignorato.

ATTIVAZIONE MASCHERA DI LEDS

Codice: ***27 52 mask1 mask2 mask3***
Codice Hex: ***0x1B 0x34 mask1 mask2 mask3***
Mnemonico: ***ESC 4 ASCII(mask1) ASCII(mask2) ASCII(mask3)***

Vengono gestiti contemporaneamente tutti i LEDs presenti sul sistema di console (come**QTP 24**, **QTP 24P**, **QTP 22**, **QTP G28**, ecc) come indicato in "mask1", "mask2" e "mask3", secondo la seguente corrispondenza:

mask1 (bit 0 …7) -> LED 0 … LED 7
mask2 (bit 0 …7) -> LED 8 … LED 15
mask3 -> (Nessuna funzione, mantenuto per motivi di compatibilità)

Se un bit é posto a 0, il LED relativo risulterà spento, viceversa questo sarà acceso se il bit in questione é posto a 1. Se alcuni dei LEDs possiedono l'attributo di blinking, questo viene disattivato. I numeri dei LEDs sono compresi nel range **0÷15**, come rappresentato in figura B1.

N.B. Il "mask3" deve essere comunque inviato anche se non ha alcun significato al fine della gestione dei LEDs del terminale.

LIBRERIE

Con il pacchetto software **GCTR** é perfettamente utilizzabile la struttura a librerie che caraterizza la programmazione in C. Nel pacchetto vengono fornite quattro file di libreria:

LCTR_T.LIB	per rendere romabile il codice;
CL.LIB	che coincide con la libreria standard del Borland TURBO C++ modificata nelle funzioni di console, di temporizzazione e gestione data ed ora;
MATHL.LIB	per poter usare la ricca serie di funzioni matematiche del Borland C;
EMU.LIB	per eseguire operazioni in floating point con una emulazione software del coprocessore matematico che infatti non é disponibile sulla scheda di controllo.

Il suffisso o prefisso L presente in molti dei nomi delle librerie corrisponde al modello di memoria utilizzato dal **GCTR**. Quest'ultimo é appunto il **Large** che soddisfa al meglio la configurazione memorie normalmente presente sulla scheda di controllo.
L'utente può liberamente intervenire sulle librerie ma solo modificando quelle esistenti infatti i programmi di supporto CTODEB e CTOBIN, che effettuano la fase di link, non possono usare file di libreria diversi da quelli sopra elencati. Si possono comunque aggiungere delle nuove funzioni in libreria aggiungedoli ad esempio alla libreria standard CL.LIB, senza restrizioni. Le modalità di modifica e/o aggiunta alle librerie sono quelle classiche del pacchetto Borland in uso gestite dall'apposito programma TLIB. Quando si dispone del codice sorgente delle funzioni di libreria da modificare o aggiungere si può utilizzare in alternativa la tecnica degli include: i file sorgenti saranno semplicemente "inclusi" nel sorgente del programma applicativo usato con CTODEB e CTOBIN. L'uso delle funzioni di libreria implica l'inclusione di adeguati file di intestazione (*.H) in cui sono presenti i prototipi delle stesse funzioni e sono dichiarate le eventuali strutture dati utilizzate. La serie completa di questi file .H viene salvata nella directory di lavoro del **GCTR** durante l'installazione e l'utente si deve limitare ad usare quelli necessari. Nell'APPENDICE B di questo manuale viene riportato l'elenco delle funzioni di libreria romate e/o modificate in cui compare anche il relativo file di intestazione necessario. Le funzioni delle librerie fornite rimpiazzano le ononime funzioni della libreria standard e possono essere quindi utilizzate direttamente nel programma applicativo sviluppato dall'utente.
Il sorgente delle librerie non é fornito nel pacchetto **GCTR** ma può essere direttamente richiesto alla **grifo®**, se ritenuto necessario. Sono invece forniti una serie di programmi dimostrativi che usano le funzioni di libreria in modo da renderle immediatamente utilizzabili.
Sono da segnalare le funzioni di gestione diretta della console ridirezionate su una linea seriale della scheda di controllo, infatti con questa caratteristica l'utente dispone immediatamente di una interfaccia utente minimale a cui era sicuramente abituato nella programmazione con Borland C++ su P.C. Al fine di sfruttarla correttamente dovrà però disporre del P.C. di console precedentemente descritto.

DIFFERENZE TRA BORLAND C++, TURBO C O TURBO C++ E GCTR

Tali differenze sono date dal fatto che il primi sono sviluppati per piattaforme hardware P.C. su cui opera anche un sistema operativo mentre, la scheda di controllo su cui opera il secondo pacchetto non ha questa struttura.

1- Il codice di partenza standard deve essere sostituito con uno apposito che possa operare da EPROM o FLASH EPROM, anche in assenza di sistema operativo (per maggiori informazioni si veda paragrafo CODICE DI PARTENZA).
2- Alcune delle funzioni di libreria che riguardano la data e l'ora non possono essere utilizzate. In particolare le funzioni di settaggio ed acquisizione data ed ora attuale sono state modificate in modo da gestire il real time clock (RTC) hardware della scheda di controllo mentre quelle basate sui cosiddetti "tick di sistema" non possono essere modificate e quindi usate.
3- Le funzioni di libreria che riguardano le periferiche hardware di un P.C. (files su hard disk o floppy disk, memorie di massa, monitor, grafica, stampante, comunicazione seriale, ecc.) non possono essere utilizzate sulla scheda di controllo.
4- Le funzioni di libreria che riguardano la console non possono essere utilizzate nella loro forma originale, in quanto fanno uso di chiamate al sistema operativo. Sono quindi state modificate tutte le funzioni di console diretta in modo da ridirezionarle su uno dei dispositivi hardware supportati, come descritto nel paragrafo GESTIONE CONSOLE.
5- Un programma applicativo sviluppato con il **GCTR** quando termina entra in un loop infinito in quanto il controllo non può essere passato al sistema operativo che non é presente sulla scheda di controllo. Le condizioni di termine di un programma applicativo sono quelle classiche come un errore in esecuzione, il raggiungimento della fine del main, oppure una chiamata al sistema operativo (INT 21H), l'uso della funzione "exit()", ecc.
6- Le tempistiche di esecuzione del codice generato sono costanti. La mancanza del sistema operativo MS-DOS o WINDOWS, e dei suoi caratteristici interrupts, garantisce dei tempi di esecuzione del codice generato invariabili, con la conseguente possibilità di prestabilire e misurare con certezza le prestazioni del programma.
7- Il programma generato dal linker deve essere trasformato in termini di memoria utilizzata prima di essere eseguito. Questa trasfformazione viene definita allocazione ed é eseguita da un apposito programma descritto nel paragrafo LOCATOR.
8- Nella modalità di SVILUPPO del programma applicativo la linea seriale A della scheda remota é sempre utilizzata dal TURBO DEBUGGER e non é quindi disponibile al programma applicativo. Sviluppando su P.C. il debug é effettuato tramite il monitor e la tastiera che invece non sono completamente impegnati e continuano quindi a svolgere le funzioni di console.

Ulteriori differenze possono essere facilmente individuate dall'utente durante la fase di debug e quindi opportunamente gestite mentre per maggiori informazioni sulle modifiche alle librerie fornite vedere sucessivo paragrafo LIBRERIE e l'APPENDICE B.

PROGRAMMI DIMOSTRATIVI

In abbinamento al **GCTR** sono forniti una serie di esempi che illustrano le modalità d'uso del pacchetto di sviluppo e consentono di utilizzare le risorse della scheda di controllo nel minor tempo possibile. Di seguito viene riportato l'elenco di questi programmi dimostrativi accompagnato da una breve descrizione:

C:\GCTRxxx\DEMOCONS.C
Dimostrativo per la gestione di tutti i dispositivi hardware di console tramite le funzioni di libreria.

C:\GCTRxxx\DEMOFP.C
Dimostrativo per la gestione delle principali funzioni matematiche su variabili floating point.

C:\GCTRxxx\DEMORIT.C
Dimostrativo per la gestione delle funzioni di libreria per la generazione di ritardi temporali.

C:\GCTRxxx\DEMORTC.C
Dimostrativo per la gestione delle funzioni di libreria per la gestione della data ed ora tramite real time clock.

C:\GCTRxxx\IRQxxx.C
Dimostrativo per la gestione degli interrupt generati dalle periferiche hardware della scheda di controllo **GPC® xxx**.

C:\GCTRxxx\PRxxx.C
Dimostrativo per la gestione di tutte le sezioni hardware della scheda di controllo **GPC® xxx**.

C:\GCTRxxx\TEST.C
Programma d'apprendimento per le modalità di utilizzo del **GCTR**, usato nel capitolo COME INIZIARE.

C:\GCTRxxx\DEB01*.C
Dimostrativi per la gestione della scheda didattica **DEB 01**, collegata alla scheda di controllo **GPC® xxx**, in tutte le sue sezioni.

Tutti questi programmi sono naturalmente forniti in formato sorgente, sono ben documentati e sono strutturati in modo da poter essere usati direttamente dall'utente. Quest'ultimo potrà decidere se usare parte di questi esempi (ad esempio le procedure) senza alcuna modifica oppure studiare il sorgente e modificarlo a seconda delle proprie esigenze.

VERSIONI GCTR

Di base sono disponibili le seguenti versioni del pacchetto software:

GCTR xxx
Ambiente di sviluppo per la scheda **GPC® xxx** in EPROM.

FGCTR xxx
Ambiente di sviluppo per la scheda **GPC® xxx** in FLASH EPROM da 128K Byte.

FGCTR xxx.512K
Ambiente di sviluppo per la scheda **GPC® xxx** in FLASH EPROM da 512K Byte.

FWR xxx
FLASH EPROM da 128 KByte per la scheda **GPC® xxx**, con il FLASH WRITER programmato nel primo settore.

FWR xxx.512K
FLASH EPROM da 512 KByte per la scheda **GPC® xxx**, con il FLASH WRITER programmato nel primo settore.

Le cinque sigle sopra riportate si adattano alla scheda con frequenza di clock di base (20 MHz per **GPC® 188F/D** e 26 MHz per **GPC® 884**, **GPC® 883**); qualora l'utente intenda utilizzare il **GCTR** con frequenze di clock superiori, deve utilizzare le apposite versioni identificate dal suffisso .xxM (ad esempio .40M per la **GPC® 884** a 40 MHz di clock). Si ricorda inoltre, che le sigle sopra riportate possono essere utilizzate direttamente per gli eventuali ordini.

Come ogni software e firmware, anche il **GCTR** é soggetto a continue evoluzioni e modifiche, con l'intento di soddisfare nel modo migliore le nuove richieste dell'utenza e di eliminare gli eventuali problemi riscontrati. Di seguito viene quindi riportata una breve descrizione delle modifiche che il pacchetto di sviluppo ha subito a seconda del numero di versione:

Ver. 1.0	->	Prima versione.
Ver. 1.1	->	Aggiunta programma d'installazione.
Ver. 2.0	->	Aggiunta gestione completa del floating point.
Ver. 3.1	->	Aggiunta gestione FLASH EPROM; scelto modello memoria large; aggiunta gestione **GPC® 884**.
Ver. 3.2	->	Migliorato settaggio del codice di partenza.
Ver. 3.3	->	Aggiunta selezione quantità memoria della scheda di controllo; aggiunta gestione breakpoint hardware.
Ver. 3.4	->	Ampliate funzioni di libreria; aggiunta selezione seriale su PC di sviluppo; aumentato livello di warning; aggiunto utilizzo da WINDOWS 3.1, 95, 98, ME.
Ver. 3.5	->	Aggiunto retrigger watch dog esterno; migliorati programmi dimostrativi; aggiunta gestione **GPC® 883**.

Ogni eventuale aggiunta o miglioria che l'utente ritiene interessante può essere proposta contattando direttamente la **grifo®**.

BIBLIOGRAFIA

E’ riportato di seguito, un elenco di manuali e note tecniche, a cui l'utente può fare riferimento per ottenere ulteriori informazioni, che semplificano l'utilizzo del **GCTR**:

Titolo	Autore/i
LINGUAGGIO C	W. Kernighan e M. Ritchie
BORLAND C++ - Guida all'uso	Borland
BORLAND C++ - Guida alla programmazione	Borland
TURBO DEBUGGER - Guida all'uso	Borland
BORLAND C++ - Libreria	Borland

Per avere tutti gli aggiornamenti di tali manuali, programmi applicativi d'esempio, trattazioni particolari, ecc., fare riferimento anche agli appositi siti INTERNET.

APPENDICE A: SCHEMI ELETTRICI

In questa appendice sono disponibili gli schemi elettrici di alcuni dei dispositivi harware di console, gestiti dal **GCTR**. Tutte queste interfaccie possono essere prodotte autonomamente dall'utente oppure possono essere ordinate direttamente alla **grifo®**.

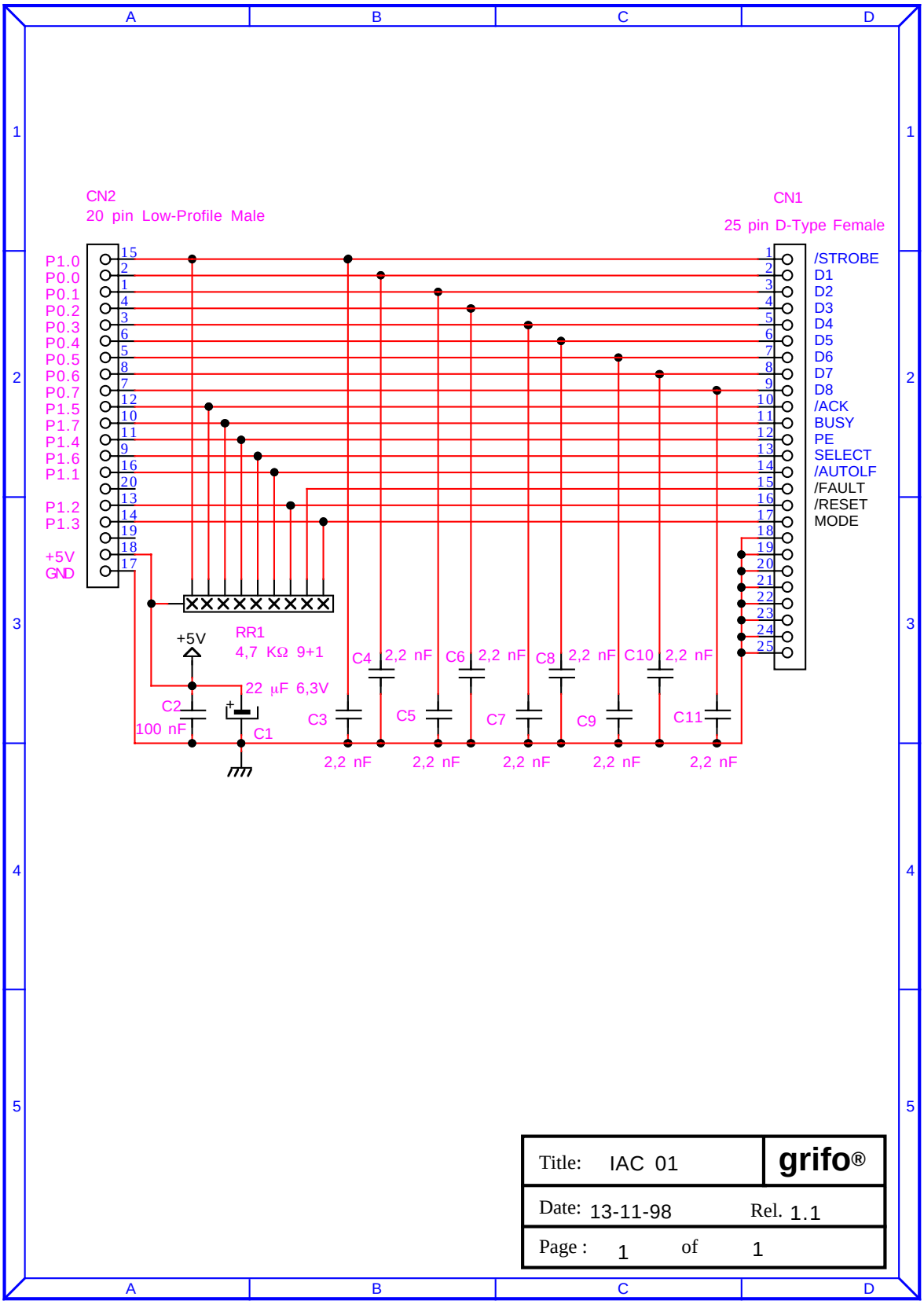

FIGURA A1: SCHEMA ELETTRICO IAC 01

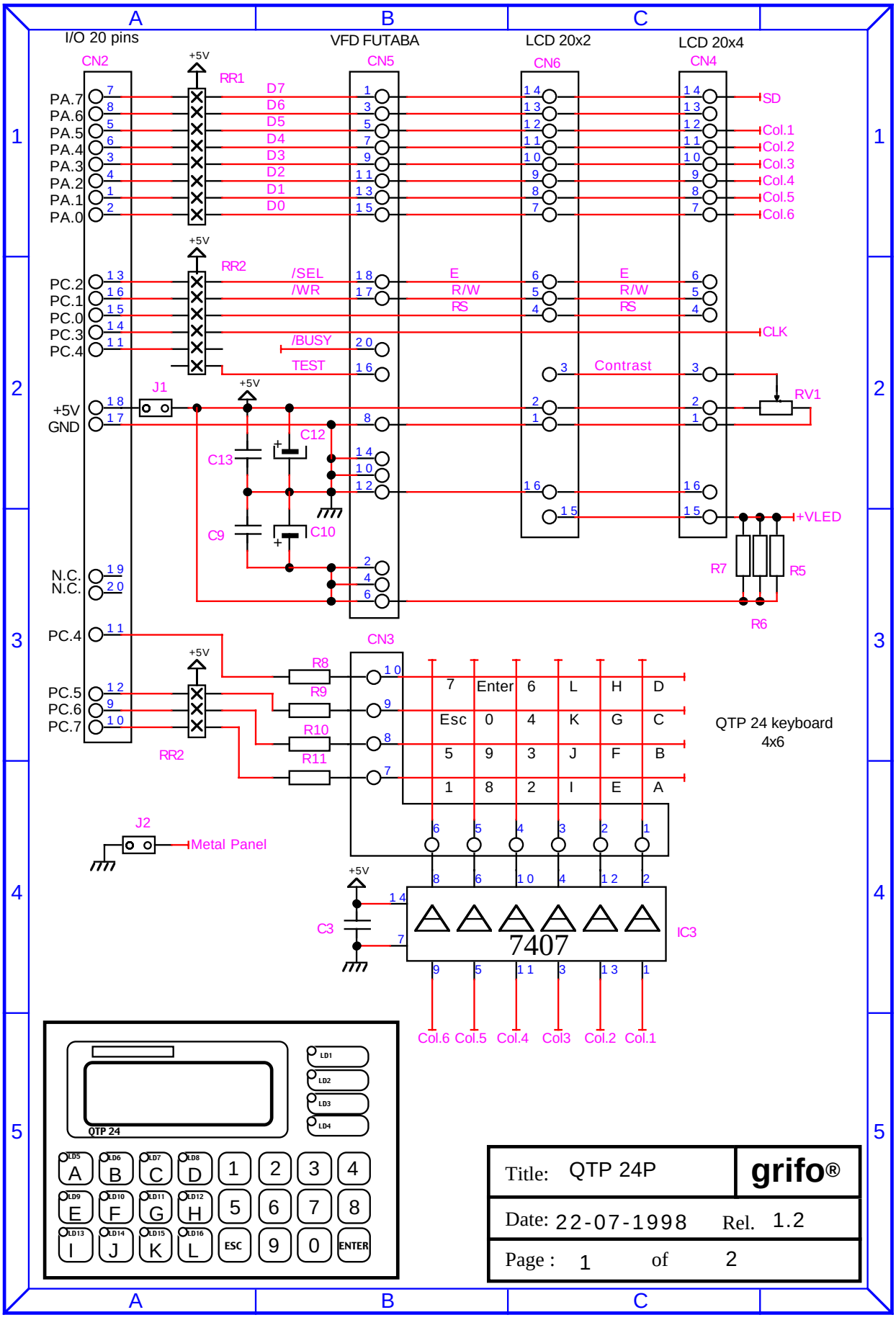

FIGURA A2: SCHEMA ELETTRICO QTP 24P (1 DI 2)

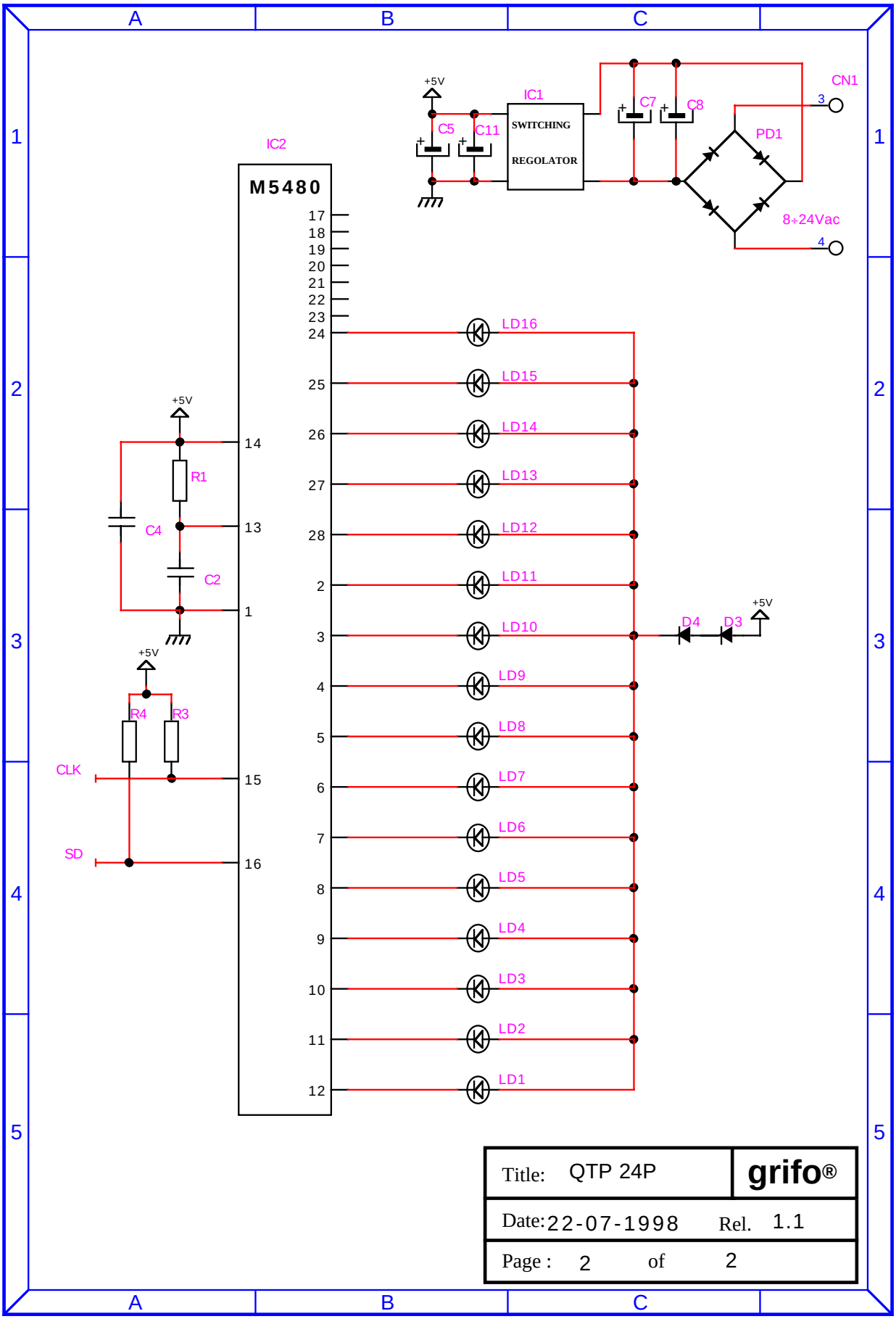

FIGURA A3: SCHEMA ELETTRICO QTP 24P (2 DI 2)

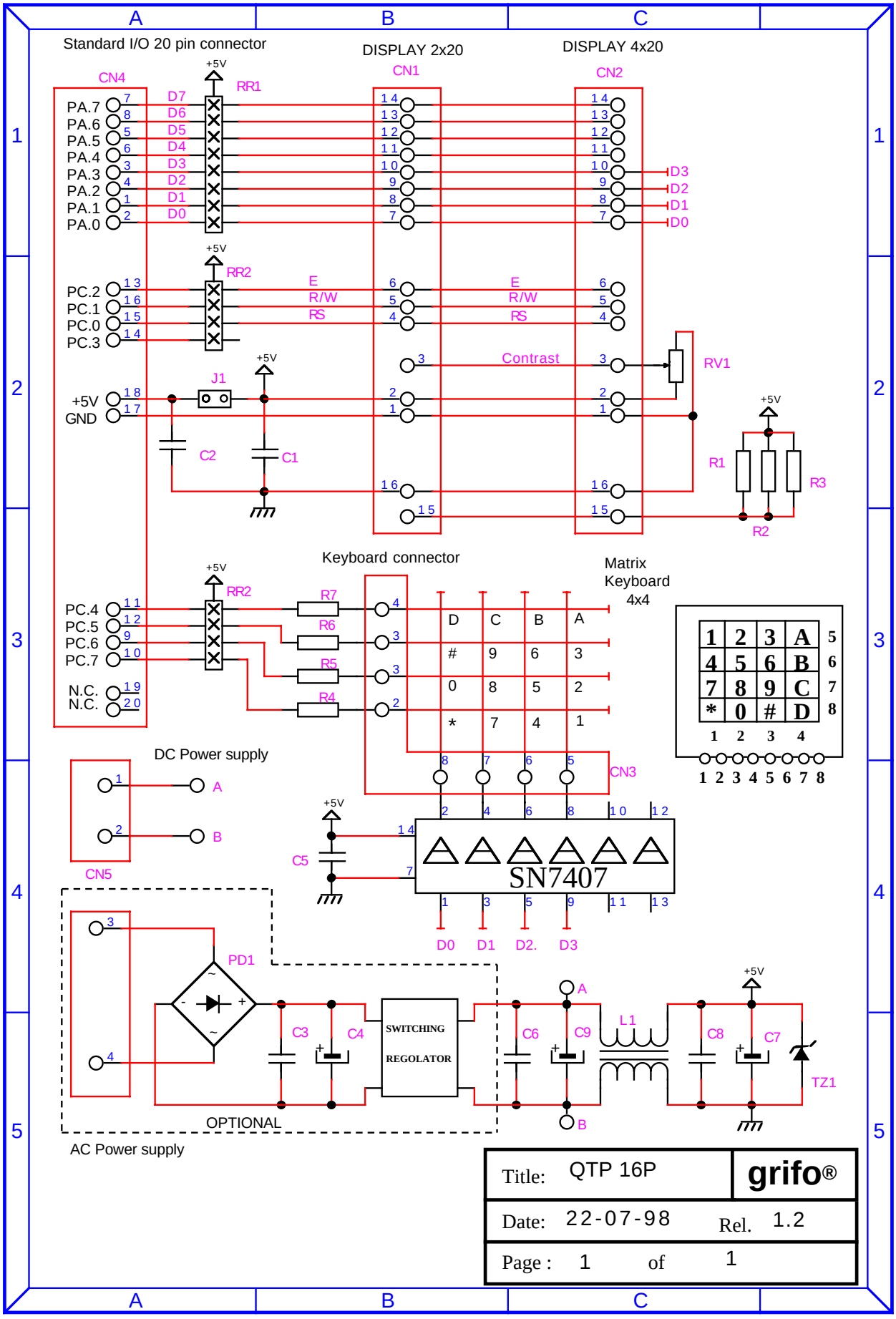

FIGURA A4: SCHEMA ELETTRICO QTP 16P

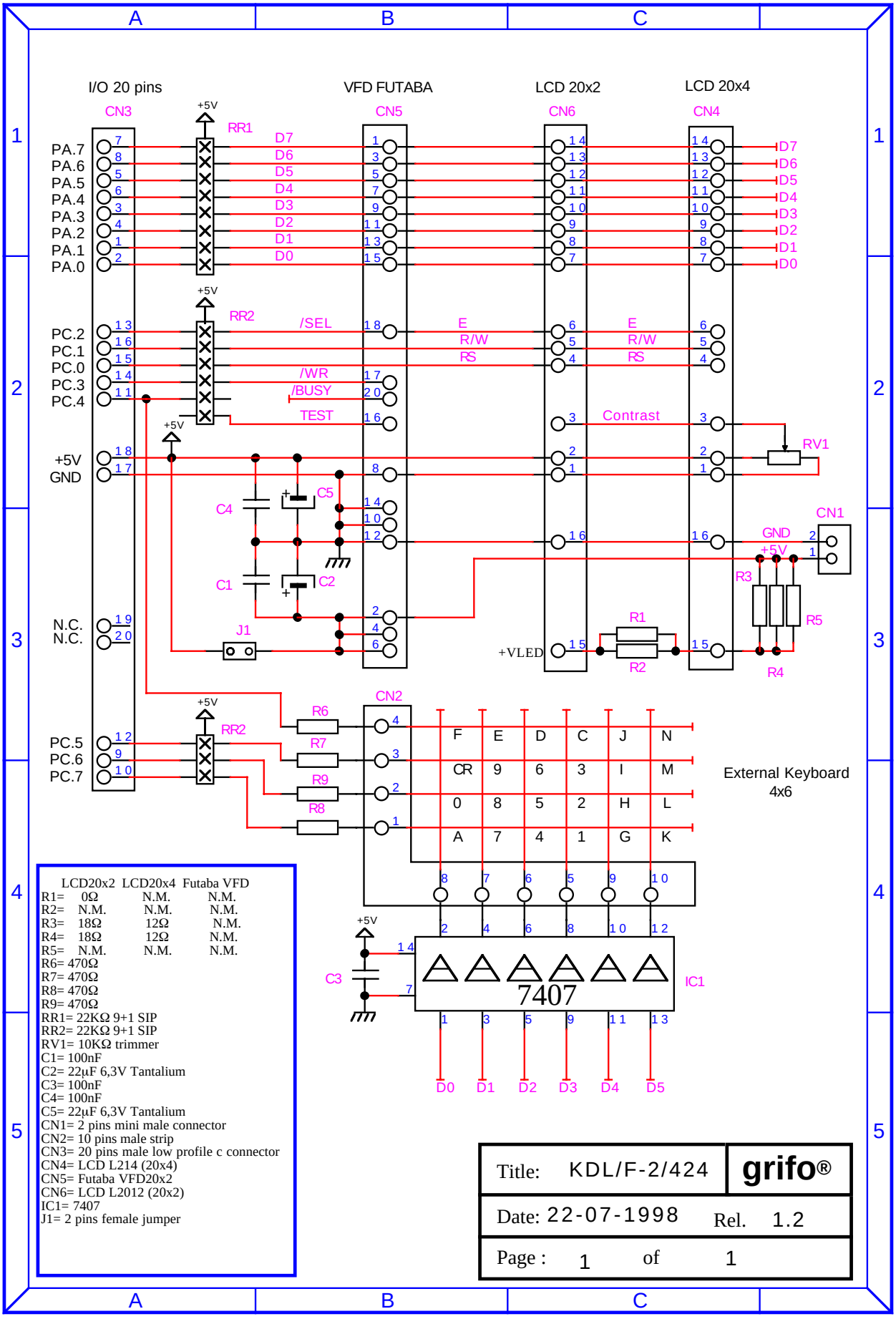

FIGURA A5: SCHEMA ELETTRICO KDX X24

APPENDICE B: FUNZIONI DI LIBRERIA MODIFICATE

CALLOC

Definizione:
#include <ALLOC.H>
void* calloc(unsigned int items, unsigned int size);

Libreria:
LCTR_T.LIB

Descrizione:
La funzione calloc alloca un'area di memoria e la azzera. La memoria allocata da una chiamata a questa funzione corrisponde allo spazio occupato da items*size, deve essere inferiore ai 64K Bytes ed é allocata nell'area destinata all'heap.

Esempio:
struct zoo*park1;
park1=calloc(105,sizeof(struct zoo));

Parametri restituiti:
La funzione calloc restituisce un puntatore alla memoria allocata in caso di successo oppure un puntatore NULL in caso di errori. Il puntatore restituito é NULL se la memoria libera disponibile é insufficiente a soddisfare completamente la richiesta.

CLREOL

Definizione:
#include <CONIO.H>
void clreol(void);

Libreria:
CL.LIB

Descrizione:
La funzione clreol cancella tutti i caratteri che si trovano dalla posizione attuale del cursore fino al termine della riga, senza spostare lo stesso cursore. Se come dispositivo di console d'uscita é stato selezionata una linea seriale, su questa verranno trasmessi i relativi codici ADDS View-Point.

Esempio:
integer i;
cputs("Inserire numero pezzi=").
clreol();
cscanf("%d",&i);

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione clreol.

CLRSCR

Definizione:
#include <CONIO.H>
void clrscr(void);

Libreria:
CL.LIB

Descrizione:
La funzione clrscr cancella tutti i caratteri rappresentati e posiziona il cursore in Home (ovvero il carattere nell'angolo in alto a sinistra). Se come dispositivo di console d'uscita é stato selezionata una linea seriale, su questa verranno trasmessi i relativi codici ADDS View-Point.

Esempio:
```
clrscr();
cputs("Videata di aiuto: scegliere la voce con i tasti freccia").
cputs("      :                    :                         : ");
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione clrscr.

CPRINTF

Definizione:
#include <CONIO.H>
int cprintf(const char *format [,argomento,.....]);

Libreria:
CL.LIB

Descrizione:
La funzione cprintf gestisce la rappresentazione formattata (caratteristica della nota funzione printf) sul dispositivo hardware di console d'uscita selezionato, che quindi dovrà essere precedentemente selezionato e/o inizializzato. Tutti i formati sono utilizzabili per l'apposita costante format e per maggiori informazioni fare riferimento al manuale del C.

Esempio:
```
integer i;
float d[100];
cprintf("Indice= %d Valore= %f \r\n",i,d[i]);
```

Parametri restituiti:
cprintf restituisce il numero di caratteri rappresentati.

CPUTS

Definizione:
#include <CONIO.H>
int cputs(const char *str);

Libreria:
CL.LIB

Descrizione:
La funzione cputs gestisce la rappresentazione di una stringa sul dispositivo hardware di console d'uscita selezionato, che quindi dovrà essere precedentemente selezionato e/o inizializzato. La stringa str deve essere terminata dal carattere nullo e la funzione non aggiunge la rappresentazione di CR o LF.

Esempio:
cputs("Produzione arrestata");

Parametri restituiti:
cputs restituisce il codice dell'ultimo carattere rappresentato.

CSCANF

Definizione:
#include <CONIO.H>
int cscanf(const char *format [,indirizzo,.....]);

Libreria:
CL.LIB

Descrizione:
La funzione cscanf gestisce l'acquisizione formattata (caratteristica della nota funzione scanf) dal dispositivo hardware di console d'ingresso selezionato, che quindi dovrà essere precedentemente selezionato e/o inizializzato. Tutti i formati sono utilizzabili per l'apposita costante format e per maggiori informazioni fare riferimento al manuale del C.

Esempio:
integer i;
float d;
cscanf("%d %f",&i,&d);

Parametri restituiti:
cscanf restituisce il numero di valori correttamente acquisiti.

DELAY

Definizione:
#include <DOS.H>
void delay(unsigned int milliseconds);

Libreria:
CL.LIB

Descrizione:
La funzione delay effettua un ritardo calibrato della durata definita dal parametro milliseconds, in millisecondi.

Esempio:
```
outp(PA,0x01);          // Attiva uscita per 50 msec
delay(50);
outp(PA,0x00);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione delay.

_DISABLE

Definizione:
#include <DOS.H>
void _disable(void);

Libreria:
LCTR_T.LIB

Descrizione:
La funzione _disable disabilita il bit per la gestione interrupts del microprocessore, salvato nel registro dei flags.

Esempio:
```
_disable();
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione _disable.

DELLINE

Definizione:
#include <CONIO.H>
void delline(void);

Libreria:
CL.LIB

Descrizione:
La funzione delline cancella l'intera linea in cui si trova il cursore, e questo viene posto all'inizio di tale riga. Se come dispositivo di console d'uscita é stato selezionata una linea seriale, su questa verranno trasmessi i relativi codici ADDS View-Point.

Esempio:
integer i;
delline();
cputs("Inserire numero pezzi=").
cscanf("%d",&i);

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione delline.

_DOS_GETDATE

Definizione:
#include <DOS.H>
void _dos_getdate(struct dosdate_t *datep);

Libreria:
CL.LIB

Descrizione:
La funzione _dos_getdate preleva la data attuale dal Real Time Clock della scheda di controllo e la salva nella variabile strutturata datep. Quest'ultima deve essere una variabile definita con il tipo dosdate_t, dichiarato sempre nell'header DOS.H, che coicide con 4 variabili: giorno (unsigned char), mese (unsigned char), anno (unsigned int) e giorno della settimana (unsigned char).

Esempio:
struct dosdate_t dd;
_dos_getdate(&dd);
cprintf("Data attuale: %2d/%2d/%2d", dd.day, dd.month, dd.year);

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione _dos_getdate.

_DOS_GETTIME

Definizione:
#include <DOS.H>
void _dos_gettime(struct dostime_t *timep);

Libreria:
CL.LIB

Descrizione:
La funzione _dos_gettime preleva l'ora attuale dal Real Time Clock della scheda di controllo e la salva nella variabile strutturata timep. Quest'ultima deve essere una variabile definita con il tipo dostime_t, dichiarato sempre nell'header DOS.H, che coicide con 4 variabili: ore, minuti, secondi, centinaia di secondi (tutte unsigned char). Il RTC della scheda di controllo non gestisce le centinaia di secondi, che sono quindi restituite sempre azzerate.

Esempio:
```
struct dostime_t tt;
_dos_getdate(&tt);
cprintf("Ora attuale: %2d:%2d:%2d", tt.hour, tt.minute, tt.second);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione _dos_gettime.

_DOS_GETVECT

Definizione:
#include <DOS.H>
void interrupt (*_dos_getvect(unsigned int intnum)) ();

Libreria:
LCTR_T.LIB

Descrizione:
La funzione _dos_getvect preleva l'indirizzo della procedura di risposta all'interrupt intnum dall'apposita area di memoria destinata ai vettori d'interrupt, in modo sicuro (disabilitando gli stessi interrupts). La famiglia di microprocessori Intel 86 gestisce 256 interrupts diversi, numerati da 0x00 a 0xFF i cui vettori sono salvati sequenzialmente nel primo K di memoria da 0x0000 a 0x0400.

Esempio:
```
void interrupt(*oldfunc) (__CPPARGS);
oldfunc=_dos_getvect(5);
```

Parametri restituiti:
L'indirizzo di tipo far (word per segmento e word per offset) della procedura di risposta all'interrupt specificato é restituito da questa funzione.

_DOS_SETDATE

Definizione:
#include <DOS.H>
unsigned char _dos_setdate(struct dosdate_t *datep);

Libreria:
CL.LIB

Descrizione:
La funzione _dos_setdate setta sul Real Time Clock della scheda di controllo la data salvata nella variabile strutturata datep. Quest'ultima deve essere una variabile definita con il tipo dosdate_t, dichiarato sempre nell'header DOS.H, che coicide con 4 variabili: giorno (unsigned char), mese (unsigned char), anno (unsigned int) e giorno della settimana (unsigned char).

Esempio:
```
struct dosdate_t dd;
dd.day=1;                    // Setta RTC ad inizio secolo
dd.month=1;
dd.year=0;
_dos_setdate(&dd);
```

Parametri restituiti:
La funzione _dos_setdate restituisce sempre il valore 0 per indicare l'avvenuto corretto settaggio.

_DOS_SETTIME

Definizione:
#include <DOS.H>
unsigned char _dos_settime(struct dostime_t *timep);

Libreria:
CL.LIB

Descrizione:
La funzione _dos_settime setta sul Real Time Clock della scheda di controllo il tempo salvato nella variabile strutturata timep. Quest'ultima deve essere una variabile definita con il tipo dostime_t, dichiarato sempre nell'header DOS.H, che coicide con 4 variabili: ore, minuti, secondi e centinaia di secondi (tutti unsigned char). Il RTC della scheda di controllo non gestisce le centinaia di secondi, quindi questo parametro d'ingresso non é utilizzato dalla funzione.

Esempio:
```
struct dostime_t tt;
tt.hour=tt.minute=tt.second=1;          // Setta RTC ad inizio giorno
_dos_settime(&tt);
```

Parametri restituiti:
La funzione _dos_settime restituisce sempre il valore 0 per indicare l'avvenuto corretto settaggio.

_DOS_SETVECT

Definizione:
#include <DOS.H>
void _dos_setvect(unsigned int intnum, void interrupt far(*isr) ());

Libreria:
LCTR_T.LIB

Descrizione:
La funzione _dos_setvect setta l'indirizzo della procedura di risposta all'interrupt intnum nell'apposita area di memoria destinata ai vettori d'interrupt, in modo sicuro (disabilitando gli stessi interrupts). La famiglia di microprocessori Intel 86 gestisce 256 interrupts diversi, numerati da 0x00 a 0xFF i cui vettori sono salvati sequenzialmente nel primo K di memoria da 0x0000 a 0x0400.

Esempio:
```
_dos_setvect(5,intris);
void interrupt intris(void)
{
}
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione _dos_setvect.

_ENABLE

Definizione:
#include <DOS.H>
void _enable(void);

Libreria:
LCTR_T.LIB

Descrizione:
La funzione _enable abilita il bit per la gestione interrupts del microprocessore, salvato nel registro dei flags.

Esempio:
```
_enable();
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione _enable.

_EXIT

Definizione:
#include <STDLIB.H>
void _exit(int value);

Libreria:
LCTR_T.LIB

Descrizione:
La funzione _exit é diversa dalla normale funzione di uscita del C infatti non ritorna al sistema operativo MS-DOS o WINDOWS ma semplicemente provvede a salvare il codice di uscita nella variabile _exit_status, ad abilitare gli interrupts ed infine ad entrare in un loop infinito. La procedura exit può essere utilizzata con profitto durante la fase di debug, infatti settando un breakpoint su questa procedura ed esaminando il codice di uscita é sempre possibile stabilire la causa che ha terminato l'esecuzione del programma, anche se non esplicitamente chiamata dal programma applicativo. Per attivare questo controllo con il TURBO DEBUGGER si deve appunto settare un breakpoint in corrispondenza dell'etichetta exit e quando questo viene raggiunto proseguire a passi (F8) fino al loop infinito della procedura _abort ed a questo punto ispezionare il valore salvato nella variabile _exit_status.
Di seguito viene riportata la corrispondenza tra le cause d'uscita ed i relativi codici d'uscita numerici:

Normale ritorno dal main	->	0x80
Assegnamento ad un puntatore NULL	->	0x81
Overflow dello stack	->	0x82
Chiamata all'INT 21H del sistema operativo	->	0x83
Emulatore floating point non inizializzato	->	0x90
Divisione per 0	->	0x91
Overflow da divisione	->	0x92

Esempio:
exit(0);

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione _exit e lei stessa non ritorna al chiamante.

FAR_FREE

Definizione:
#include <ALLOC.H>
void far_free(void far *block);

Libreria:
LCTR_T.LIB

Descrizione:
La funzione far_free libera una porzione di memoria allocata dalla funzione far_malloc. E' equivalente alla funzione free nel modello di memoria large e rilascia l'area di memoria solo se l'identificatore block é valido.

Esempio:
```
int huge *array;
array=far_malloc(50000L * sizeof(int));
   :                    :
far_free(array);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione far_free.

FAR_MALLOC

Definizione:
#include <ALLOC.H>
void far *far_malloc(unsigned long nbytes);

Libreria:
LCTR_T.LIB

Descrizione:
La funzione far_malloc alloca e quindi riserva un blocco di memoria di nbytes Bytes nell'heap. Questa funzione può allocare tutta la memoria disponibile ed é particolarmente interessante per la gestione di grossi vettori.
Visto che la funzione restituisce un puntatore di tipo far, far_malloc non ha il limite dei 64K caratteristico della funzione malloc che opera sui modelli small.

Esempio:
```
float matrix far *array;
array=far_malloc(80000L * sizeof(float));
```

Parametri restituiti:
La funzione far_malloc restituisce un puntatore far alla memoria allocata in caso di successo oppure un puntatore NULL in caso di errori. Il puntatore restituito é NULL se la memoria libera disponibile é insufficiente a soddisfare completamente la richiesta.

FREE

Definizione:
#include <ALLOC.H>
void free(void *block);

Libreria:
LCTR_T.LIB

Descrizione:
La funzione free libera una porzione di memoria allocata dalla funzione malloc. Nel caso in cui l'identificatore block non é stato allocato o viene liberato due volte, si possono presentare dei risultati imprevedibili.

Esempio:
```
char *buffer;
buffer=malloc(10000);
   :                  :
free(buffer);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione free.

GETCH , GETCHE

Definizione:
#include <CONIO.H>
int getch(void);
int getche(void);

Libreria:
CL.LIB

Descrizione:
Le funzioni getch e getche gestiscono l'acquisizione di un carattere dal dispositivo hardware di console d'ingresso selezionato, che quindi dovrà essere precedentemente selezionato e/o inizializzato. L'acquisizione é sospensiva nei confronti dell'esecuzione del programma chiamante e nel caso della getche ne viene effettuato l'eco sul dispositivo di console d'uscita.

Esempio:
```
unsigned char scelta;
if getch()=='S'
  scelta=getche();
```

Parametri restituiti:
getch e getche restituiscono il primo carattere disponibile dalla console d'ingresso.

GETDATE

Definizione:
#include <DOS.H>
void getdate(struct date *datep);

Libreria:
CL.LIB

Descrizione:
La funzione getdate preleva la data attuale dal Real Time Clock della scheda di controllo e la salva nella variabile strutturata datep. Quest'ultima deve essere una variabile definita con il tipo date, dichiarato sempre nell'header DOS.H, che coicide con 3 variabili: anno (int), giorno (char), mese (char).

Esempio:
```
struct date d;
getdate(&d);
cprintf("Data attuale: %2d/%2d/%2d", d.da_day, d.da_mon, d.da_year);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione getdate.

GETTIME

Definizione:
#include <DOS.H>
void gettime(struct time *timep);

Libreria:
CL.LIB

Descrizione:
La funzione gettime preleva l'ora attuale dal Real Time Clock della scheda di controllo e la salva nella variabile strutturata timep. Quest'ultima deve essere una variabile definita con il tipo time, dichiarato sempre nell'header DOS.H, che coicide con 4 variabili: minuti, ore, centinaia di secondi, secondi (tutte char). Il RTC della scheda di controllo non gestisce le centinaia di secondi, che sono quindi restituite sempre azzerate.

Esempio:
```
struct time t;
gettime(&t);
cprintf("Ora attuale: %2d:%2d:%2d", t.ti_hour, t.ti_min, t.ti_sec);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione gettime.

GOTOXY

Definizione:
#include <CONIO.H>
void gotoxy(int x, int y);

Libreria:
CL.LIB

Descrizione:
La funzione gotoxy posiziona il cursore nella posizione specificata dai due parametri d'ingresso x ed y, corrispondenti a collonna e riga della console. Se come dispositivo di console d'uscita é stato selezionata una linea seriale, su questa verranno trasmessi i relativi codici ADDS View-Point. Qualora le coordinate fornite sono invalide per la console selezionata la funzione non svolge alcuna operazione.

Esempio:
```
int npz;
gotoxy(2,5);
cprintf("Numero pezzi prodotti=%d",npz).
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione gotoxy.

KBHIT

Definizione:
#include <CONIO.H>
int kbhit(void);

Libreria:
CL.LIB

Descrizione:
La funzione kbhit gestisce la verifica di un carattere disponibile dal dispositivo hardware di console d'ingresso selezionato, che quindi dovrà essere precedentemente selezionato e/o inizializzato. La funzione verifica se é stato ricevuto un carattere, senza sospendere l'esecuzione del programma.

Esempio:
```
while (!kbhit());          // Ciclo di attesa tasto
```

Parametri restituiti:
kbhit restituisce un valore diverso da 0 in caso di carattere disponibile e viceversa.

LEDBLINKSTATUS

Definizione:
#include <GCLIBD.H>
unsigned int ledblinkstatus(void);

Libreria:
CL.LIB

Descrizione:
La funzione restituisce lo stato di lampeggiamento degli eventuali LEDs del dispositivo hardware di console collegato (**QTP 24** e **QTP 24P**), che quindi dovrà essere precedentemente selezionato e/o inizializzato.

Esempio:
unsigned int blink;
blink=ledblinkstatus();

Parametri restituiti:
La funzione ledblinkstatus restituisce una parola a 16 bits in cui il bit meno significativo (bit 0) corrisponde al LED0 ed il bit più significativo (bit15) corrisponde al LED15, come indicato in figura B1. Se un bit della parola restituita é settato (1) il relativo LED é lampeggiante e viceversa.

LEDSTATUS

Definizione:
#include <GCLIBD.H>
unsigned int ledstatus(void);

Libreria:
CL.LIB

Descrizione:
La funzione restituisce lo stato di attivazione degli eventuali LEDs del dispositivo hardware di console collegato (**QTP 24** e **QTP 24P**), che quindi dovrà essere precedentemente selezionato e/o inizializzato.

Esempio:
unsigned int led;
led=ledstatus();

Parametri restituiti:
La funzione ledstatus restituisce una parola a 16 bits in cui il bit meno significativo (bit 0) corrisponde al LED0 ed il bit più significativo (bit15) corrisponde al LED15, come indicato in figura B1. Se un bit della parola restituita é settato (1) il relativo LED é acceso o lampeggiante, viceversa se il bit é resettato (0) il LED é spento.

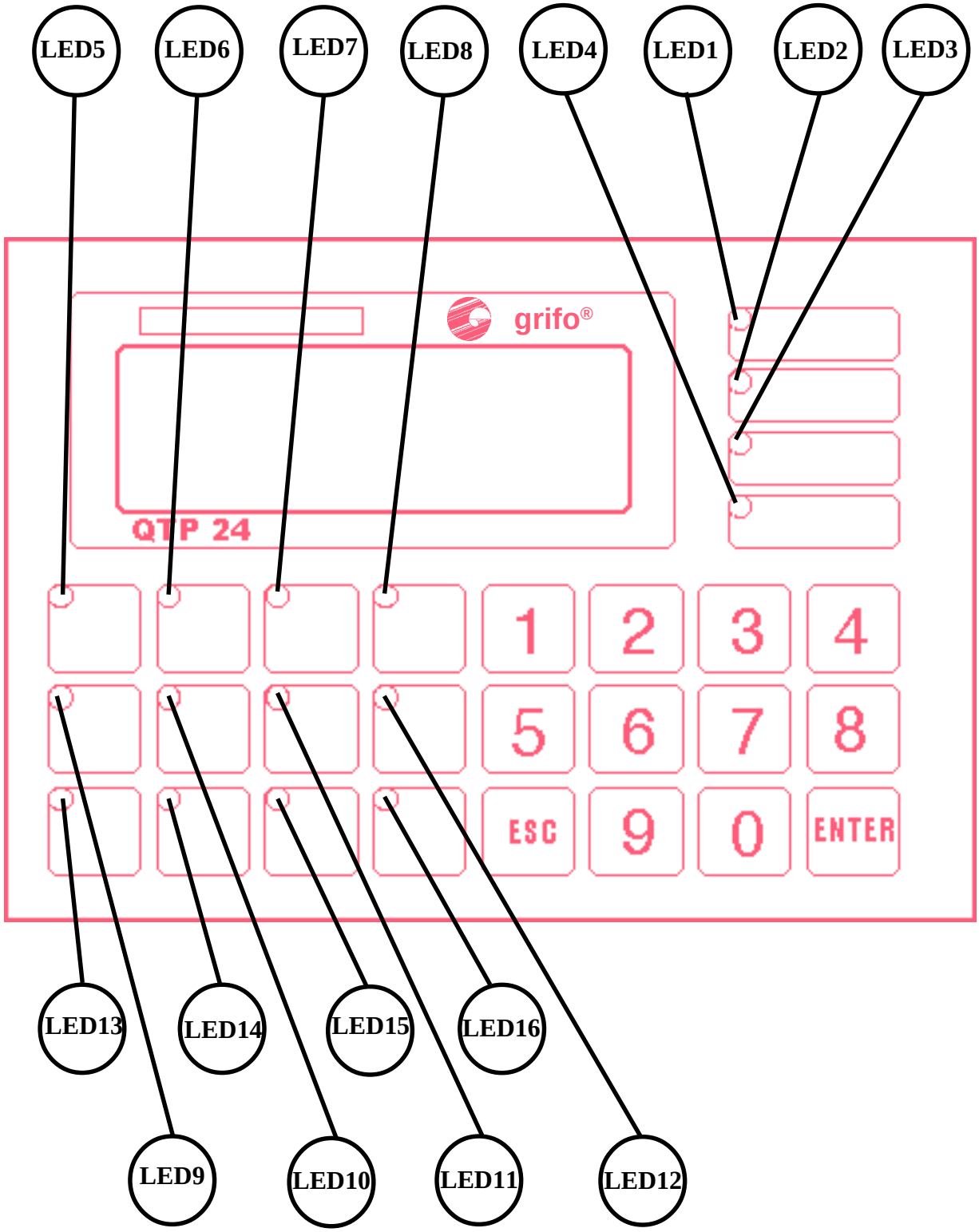

FIGURA B1: NUMERAZIONE LEDS SU QTP 24 E QTP 24P

MALLOC

Definizione:
#include <ALLOC.H>
void *malloc(unsigned int nbytes);

Libreria:
LCTR_T.LIB

Descrizione:
La funzione malloc alloca e quindi riserva un blocco di memoria di nbytes Bytes nell'heap. Questa funzione può allocare tutta la memoria disponibile fino ad un massimo di 64K Bytes.

Esempio:
unsigned char *tmp;
tmp=malloc(8000);

Parametri restituiti:
La funzione malloc restituisce un puntatore far alla memoria allocata in caso di successo oppure un puntatore NULL in caso di errori. Il puntatore restituito é NULL se la memoria libera disponibile é insufficiente a soddisfare completamente la richiesta.

PUTCH

Definizione:
#include <CONIO.H>
int putch(int ch);

Libreria:
CL.LIB

Descrizione:
La funzione putch gestisce la rappresentazione di un singolo carattere sul dispositivo hardware di console d'uscita selezionato, che quindi dovrà essere precedentemente selezionato e/o inizializzato.

Esempio:
putch('\r');

Parametri restituiti:
putch restituisce il carattere rappresentato.

QTPLED

Definizione:
#include <GCLIBD.H>
void qtpled(unsigned char nled, unsighed char attr);

Libreria:
CL.LIB

Descrizione:
La funzione gestisce il LED indicato in nled, con l'attributo specificato in attr, del dispositivo hardware di console collegato (**QTP 24** e **QTP 24P**), che quindi dovrà essere precedentemente selezionato e/o inizializzato. I numeri dei LEDs sono compresi nel range **0÷15**, come rappresentato in figura B1, mentre gli attributi disponibili sono i seguenti:

0 (00 Hex) -> LED disattivato
255 (FF Hex) -> LED attivato
85 (55 Hex) -> LED lampeggiante (blinking)

Se i parametri non sono validi, la funzione non svolge alcuna operazione.

Esempio:
```
qtpled(5, 85);          // Setta LED5 lampeggiante
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione qtpled.

SERIN

Definizione:
#include <GCLIBD.H>
unsigned char serIn(unsigned char nser);

Libreria:
CL.LIB

Descrizione:
La funzione serIn gestisce la ricezione di un carattere dalla linea seriale nser, che quindi dovrà essere precedentemente inizializzata. La funzione può utilizzare una delle seriali della scheda di controllo a seconda del parametro nser: 0 -> seriale B ed 1 -> seriale A ed attende la ricezione del carattere, sospendendo l'esecuzione del programma.

Esempio:
```
unsigned char rx[10];
for (i=0; i<5; i++)
  rx[i]=serIn(0);          // Ciclo di ricezione 5 chr da seriale B
```

Parametri restituiti:
serIn restituisce il codice del carattere ricevuto.

SEROUT

Definizione:
#include <GCLIBD.H>
void serOut(unsigned char nser, unsigned char c);

Libreria:
CL.LIB

Descrizione:
La funzione serOut gestisce la trasmissione del carattere c sulla linea seriale nser, che quindi dovrà essere precedentemente inizializzata. La funzione può utilizzare una delle seriali della scheda di controllo a seconda del parametro nser: 0 -> seriale B ed 1 -> seriale A.

Esempio:
```
unsigned char tx[10];
for (i=0; i<5; i++)
  serOut(0, tx[i]);          // Ciclo di trasmissione 5 chr su seriale B
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione serOut.

SERSTATUS

Definizione:
#include <GCLIBD.H>
unsigned char serStatus(unsigned char nser);

Libreria:
CL.LIB

Descrizione:
La funzione serStatus gestisce la verifica di ricezione di un carattere dalla linea seriale nser, che quindi dovrà essere precedentemente inizializzata. La funzione può utilizzare una delle seriali della scheda di controllo a seconda del parametro nser: 0 -> seriale B ed 1 -> seriale A e non sospende l'esecuzione del programma.

Esempio:
```
unsigned char rx;
if (serStatus(0))          // Se carattere ricevuto da seriale B
  rx=serin(0);             // lo preleva
```

Parametri restituiti:
serStatus restituisce un valore diverso da 0 in caso di carattere ricevuto e viceversa.

SETIN

Definizione:
#include <GCLIBD.H>
unsigned int setIn(unsigned int device);

Libreria:
CL.LIB

Descrizione:
La funzione setIn consente di selezionare il dispositivo hardware di console d'ingresso e lo predispone per le sucessive operazioni svolte dalle altre funzioni di console. Il parametro d'ingresso device definisce la console collegata alla scheda di controllo e deve essere definito come descritto nel paragrafo SIMBOLI PREDEFINITI PER CONSOLE.

Esempio:
```
unsigned int devin;
devin=QTP16P | LCD20x4;          // Console su QTP 16P con display LCD 20x2
setIn(devin);
```

Parametri restituiti:
setIn restituisce un valore uguale a 0 in caso di dispositivo scelto non valido e viceversa.

SETOUT

Definizione:
#include <GCLIBD.H>
unsigned int setOut(unsigned int device);

Libreria:
CL.LIB

Descrizione:
La funzione setOut consente di selezionare il dispositivo hardware di console d'uscita e lo predispone per le sucessive operazioni svolte dalle altre funzioni di console. Il parametro d'ingresso device definisce la console collegata alla scheda di controllo e deve essere definito come descritto nel paragrafo SIMBOLI PREDEFINITI PER CONSOLE.

Esempio:
```
unsigned int devout;
devout=SER0;                    // Console su seriale B
setOut(devout);
```

Parametri restituiti:
setOut restituisce un valore uguale a 0 in caso di dispositivo scelto non valido e viceversa.

SETSERIAL

Definizione:
#include <GCLIBD.H>
void setSerial(unsigned char nser, unsigned long baud, unsigned char bitxchr, unsigned char parity,
unsigned char stopbit);

Libreria:
CL.LIB

Descrizione:
La funzione setSerial gestisce l'inizializzazione della linea seriale nser, in modo da poterla utilizzare con le sucessive funzioni che la usano. I cinque parametri d'ingresso hanno il seguente significato:

nser	seriale da inizializzare	0 -> seriale B 1 -> seriale A
baud	baud rate	50÷115200
bitxchr	numero bit per carattere	5÷8
parity	gestione bit di parità	0 -> nessuna 1 -> dispari 2 -> pari
stop bit	numero di stop bit	1 o 2

A seconda della scheda di controllo utilizzata i valori ammessi per i precedenti parametri possono variare; spetta quindi all'utente individuare quelli validi, sul manuale tecnico della scheda, ed utilizzarli per chiamare correttamente la funzione. La gestione di handshake hardware e/o software delle linee seriali non viene mai abilitata dalla funzione setSerial.

Esempio:
```
setSerial(0, 19200, 8, 0, 1);          // Inizializza seriale B
serOut(0, 65);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione setSerial.

SETDATE

Definizione:
#include <DOS.H>
void setdate(struct date *datep);

Libreria:
CL.LIB

Descrizione:
La funzione setdate setta la data attuale del Real Time Clock della scheda di controllo con i dati salvati nella variabile strutturata datep. Quest'ultima deve essere una variabile definita con il tipo date, dichiarato sempre nell'header DOS.H, che coicide con 3 variabili: anno (int), giorno (char), mese (char).

Esempio:
```
struct date d;
d.da_day=1;                    // Setta RTC ad inizio secolo
d.da_mon=1;
d.da_year=0;
setdate(&d);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione setdate.

SETTIME

Definizione:
#include <DOS.H>
void settime(struct time *timep);

Libreria:
CL.LIB

Descrizione:
La funzione settime setta l'ora attuale del Real Time Clock della scheda di controllo con i dati salvati nella variabile strutturata timep. Quest'ultima deve essere una variabile definita con il tipo time, dichiarato sempre nell'header DOS.H, che coicide con 4 variabili: minuti, ore, centinaia di secondi, secondi (tutte char). Il RTC della scheda di controllo non gestisce le centinaia di secondi, quindi questo parametro d'ingresso non é utilizzato dalla funzione..

Esempio:
```
struct time t;
t.ti_hour= t.ti_min= t.ti_sec= 1;      // Setta RTC ad inizio giorno
settime(&t);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione settime.

SLEEP

Definizione:
#include <DOS.H>
void sleep(unsigned int seconds);

Libreria:
CL.LIB

Descrizione:
La funzione sleep effettua un ritardo calibrato della durata definita dal parametro seconds, in secondi.

Esempio:
```
outp(PA,0x02);          // Attiva uscita per 5 sec
sleep(5);
outp(PA,0x00);
```

Parametri restituiti:
Non ci sono parametri restituiti dalla funzione sleep.

_STRDATE

Definizione:
#include <TIME.H>
char *_strdate(char *buf);

Libreria:
CL.LIB

Descrizione:
La funzione _strdate converte la data attuale del Real Time Clock della scheda di controllo in una stringa salvata nel buffer *buf. La stringa generata é terminata dal classico carattere nullo, ed ha la notazione americana MM/GG/AA dove MM, GG, AA sono tutti numeri a 2 cifre per il mese, giorno ed anno. Si ricava quindi che il buffer di salvataggio deve essere lungo almeno 9 caratteri.

Esempio:
```
char datebuf[9];
_strdate(datebuf);
cprintf("Data: %s",datebuf);
```

Parametri restituiti:
_strdate restituisce buf, ovvero l'indirizzo della stringa in cui é stata salvata la data.

_STRTIME

Definizione:
#include <TIME.H>
char *_strtime(char *buf);

Libreria:
CL.LIB

Descrizione:
La funzione _strtime converte il tempo attuale del Real Time Clock della scheda di controllo in una stringa salvata nel buffer *buf. La stringa generata é terminata dal classico carattere nullo, ed ha la notazione OO:MM:SS dove OO, MM, SS sono tutti numeri a 2 cifre per le ore, minuti e secondi. Si ricava quindi che il buffer di salvataggio deve essere lungo almeno 9 caratteri.

Esempio:
```
char timebuf[9];
_strtime(timebuf);
cprintf("Ore: %s",timebuf);
```

Parametri restituiti:
_strtime restituisce buf, ovvero l'indirizzo della stringa in cui é stata salvata il tempo.

WHEREX

Definizione:
#include <CONIO.H>
int wherex(void);

Libreria:
CL.LIB

Descrizione:
La funzione wherex restituisce l'attuale posizione orizzontale x (colonna) del cursore, sulla console.

Esempio:
```
int col, row;
col=wherex();           // Sposta cursore 5 chr avanti
row=wherey();
gotoxy(col+5,row);
```

Parametri restituiti:
wherex restituisce la colonna in cui si trova il cursore ed il suo range di variazione cambia a seconda della console selezionata.

WHEREY

Definizione:
#include <CONIO.H>
int wherey(void);

Libreria:
CL.LIB

Descrizione:
La funzione wherex restituisce l'attuale posizione verticale y (colonna) del cursore, sulla console.

Esempio:
```
int col, row;
col=wherex();              // Sposta cursore 3 chr in basso
row=wherey();
gotoxy(col,row+3);
```

Parametri restituiti:
wherex restituisce la riga in cui si trova il cursore ed il suo range di variazione cambia a seconda della console selezionata.

APPENDICE C: INDIRIZZI I/O

DISP.	REG.	IND.	R/W	SIGNIFICATO
ABACO® I/O BUS	IOBUS	F000H÷F0FFH	R/W	Indirizzi **ABACO®** I/O BUS
RUN/DEB.	RUNDEB	F100H	R	Registro acquisizione stato jumper input utente
Real Time Clock	SEC1	F100H	R/W	Registro dati per unità secondi
	SEC10	F101H	R/W	Registro dati per decine secondi
	MIN1	F102H	R/W	Registro dati per unità minuti
	MIN10	F103H	R/W	Registro dati per decine minuti
	HOU1	F104H	R/W	Registro dati per unità ore
	HOU10	F105H	R/W	Registro dati per decine ore e AM/PM
	DAY1	F106H	R/W	Registro dati per unità giorno
	DAY10	F107H	R/W	Registro dati per decine giorno
	MON1	F108H	R/W	Registro dati per unità mese
	MON10	F109H	R/W	Registro dati per decine mese
	YEA1	F10AH	R/W	Registro dati per unità anno
	YEA10	F10BH	R/W	Registro dati per decine anno
	WEE	F10CH	R/W	Registro dati per giorno della settimana
	REGD	F10DH	R/W	Registro di stato e controllo D
	REGE	F10EH	R/W	Registro di stato e controllo E
	REGF	F10FH	R/W	Registro di stato e controllo F
W. DOG	RWD	F600H	R/W	Registro di retrigger watch dog

FIGURA C1: INDIRIZZI REGISTRI DI I/O SU GPC® 884

DISP.	REG.	INDIRIZZO	R/W	SIGNIFICATO
W.DOG	RWD	F000H	R/W	Retrigger watch dog
EEPROM	RE2	F000H	R/W	Accesso seriale ad EEPROM
MMU	MMU	F000H	R/W	Impaginazione memorie con MMU
LD3,4	LED	F000H	R/W	Registro gestione LEDs di attività
BT1	BAT	F000H	R	Registro acquisizione stato batteria
SCC	RSB	F080H	R/W	Registro stato linea seriale B
85C30	RDB	F081H	R/W	Registro dati linea seriale B
	RSA	F082H	R/W	Registro stato linea seriale A
	RDA	F083H	R/W	Registro dati linea seriale A
DMA	DMA	F100H	R/W	Registro disattivazione richiesta DMA
A/D	IRL0÷7	F180H÷F18EH (pari)	R/W	Registro istruzioni low 0÷7 del sequencer
LM12458	IRH0÷7	F181H÷F18FH (dispari)	R/W	Registro istruzioni high 0÷7 del sequencer
	CNTL	F190H	R/W	Registro di configurazione low
	CNTH	F191H	R/W	Registro di configurazione high
	INTENL	F192H	R/W	Registro abilitazione interrupt low
	INTENH	F193H	R/W	Registro abilitazione interrupt high
	INTSTL	F194H	R	Registro di stato interrupt low
	INTSTH	F195H	R	Registro di stato interrupt high
	TMRL	F196H	R/W	Registro per timer low
	TMRH	F197H	R/W	Registro per timer high
	FIFOL	F198H	R	Registro per conversioni in FIFO low
	FIFOH	F199H	R	Registro per conversioni in FIFO high
	LIMSTL	F19AH	R	Registro stato limiti low
	LIMSTH	F19BH	R	Registro stato limiti high

FIGURA C2: INDIRIZZI REGISTRI DI I/O SU GPC® 188F (1 DI 2)

DISP.	REG.	INDIRIZZO	R/W	SIGNIFICATO
PPI 82C55	PA	F200H	R/W	Registro dati del port A
	PB	F201H	R/W	Registro dati del port B
	PC	F202H	R/W	Registro dati del port C
	RC	F203H	R/W	Registro di controllo e comando
RTC 62421	S1	F280H	R/W	Registro unità secondi
	S10	F281H	R/W	Registro decine secondi
	MI1	F282H	R/W	Registro unità minuti
	MI10	F283H	R/W	Registro decine minuti
	H1	F284H	R/W	Registro unità ore
	H10	F285H	R/W	Registro decine ore; AM/PM
	D1	F286H	R/W	Registro unità giorno
	D10	F287H	R/W	Registro decine giorno
	MO1	F288H	R/W	Registro unità mese
	MO10	F289H	R/W	Registro decine mese
	Y1	F28AH	R/W	Registro unità anno
	Y10	F28BH	R/W	Registro decine anno
	W	F28CH	R/W	Registro giorno della settimana
	REGD	F28DH	R/W	Registro di stato e controllo D
	REGE	F28EH	R/W	Registro di stato e controllo E
	REGF	F28FH	R/W	Registro di stato e controllo F
WRITE PROTECT	WRP	F300H	W	Registro sprotezione scrittura SRAM
DIP SWITCH	DSW1	F300H	R	Registro acquisizione dip switch

FIGURA C3: INDIRIZZI REGISTRI DI I/O SU GPC® 188F (2 DI 2)

DISP.	REG.	IND.	R/W	SIGNIFICATO
W.DOG	RWD	F000H	R	Retrigger watch dog
EEPROM	RE2	F000H	R/W	Accesso seriale ad EEPROM
SCC	RSB	F080H	R/W	Registro stato linea seriale B
85C30	RDB	F081H	R/W	Registro dati linea seriale B
	RSA	F082H	R/W	Registro stato linea seriale A
	RDA	F083H	R/W	Registro dati linea seriale A
DMA	DMA	F100H	R/W	Registro disattivazione richiesta DMA
ABACO® I/O BUS	IOBUS	F180H÷F1FFH	R/W	Indirizzi per la gestione dell'**ABACO®** I/O BUS
PPI 82C55	PA	F200H	R/W	Registro dati del port A
	PB	F201H	R/W	Registro dati del port B
	PC	F202H	R/W	Registro dati del port C
	RC	F203H	R/W	Registro di controllo e comando

FIGURA C4: INDIRIZZI REGISTRI DI I/O SU GPC® 188D (1 DI 2)

DISP.	REG.	IND.	R/W	SIGNIFICATO
RTC	S1	F280H	R/W	Registro unità secondi
72421	S10	F281H	R/W	Registro decine secondi
	MI1	F282H	R/W	Registro unità minuti
	MI10	F283H	R/W	Registro decine minuti
	H1	F284H	R/W	Registro unità ore
	H10	F285H	R/W	Registro decine ore; AM/PM
	D1	F286H	R/W	Registro unità giorno
	D10	F287H	R/W	Registro decine giorno
	MO1	F288H	R/W	Registro unità mese
	MO10	F289H	R/W	Registro decine mese
	Y1	F28AH	R/W	Registro unità anno
	Y10	F28BH	R/W	Registro decine anno
	W	F28CH	R/W	Registro giorno della settimana
	REGD	F28DH	R/W	Registro di stato e controllo D
	REGE	F28EH	R/W	Registro di stato e controllo E
	REGF	F28FH	R/W	Registro di stato e controllo F
WR PROT	WRP	F300H	W	Registro sprotezione scrittura
DIP SWITCH	DSW1	F300H	R	Registro acquisizione dip switch
LEDS ATTIVITA'	LED	F340H	W	Registro di gestione LEDs attività

FIGURA C5: INDIRIZZI REGISTRI DI I/O SU GPC® 188D (2 DI 2)

DISP.	REG.	INDIRIZZO	R/W	SIGNIFICATO
ABACO® I/O BUS	I/O BUS	F000H÷F0FFH	R/W	Indirizzi gestione **ABACO®** I/O BUS
RUN / DEB.	RUNDEB	F100H	R	Registro acquisizione stato jumper J1
RTC 62421	SEC1	F100H	R/W	Registro dati per unità secondi
	SEC10	F101H	R/W	Registro dati per decine secondi
	MIN1	F102H	R/W	Registro dati per unità minuti
	MIN10	F103H	R/W	Registro dati per decine minuti
	HOU1	F104H	R/W	Registro dati per unità ore
	HOU10	F105H	R/W	Registro dati per decine ore e AM/PM
	DAY1	F106H	R/W	Registro dati per unità giorno
	DAY10	F107H	R/W	Registro dati per decine giorno
	MON1	F108H	R/W	Registro dati per unità mese
	MON10	F109H	R/W	Registro dati per decine mese
	YEA1	F10AH	R/W	Registro dati per unità anno
	YEA10	F10BH	R/W	Registro dati per decine anno
	WEE	F10CH	R/W	Registro dati per giorno della settimana
	REGD	F10DH	R/W	Registro di controllo D
	REGE	F10EH	R/W	Registro di controllo E
	REGF	F10FH	R/W	Registro di controllo F
A/D MAX197	DASCTRL	F500H	W	Registro di controllo A/D converter
	DASL	F500H	R	Registro dati low A/D converter
	DASH	F501H	R	Registro dati high A/D converter
PPI 82C55	PA	F540H	R/W	Registro dati del port A
	PB	F541H	R/W	Registro dati del port B
	PC	F542H	R/W	Registro dati del port C
	RC	F543H	R/W	Registro di controllo e comando
W. DOG	RWD	F580H	R/W	Registro di retrigger del Watch Dog
DIP SWITCH	DSW1	F5C0H	R	Registro di acquisizione dip switch
UART CAN SJA 1000	CAN	F600H÷F67FH	R/W	Registri per la gestione dell'UART CAN SJA 1000, in modalità BasicCAN o PeliCAN (i registri sono gli stessi riportati nel data sheet del componente, con un offset di **F600H**).

FIGURA C6: INDIRIZZI REGISTRI DI I/O SU GPC® 883

APENDICE D: INDICE ANALITICO

D

E

F

G

H

I

J

K

L

M

P

Q

R

S

T

U

V

W