Istituto Nazionale di Fisica Nucleare
Laboratori Nazionali di Frascati

STAGE ESTIVO PRESSO I LNF

SVILUPPO E IMPLEMENTAZIONE IN LABVIEW DI UN TOOL DI DIAGNOSTICA PER CPU SU BUS VME

Tutor
Ing. Baccarelli Gianfranco
Dott. Masciarelli Mario

Studenti
Celletti Federico
Di Nicola Massimiliano
Fei Emanuele
Izzi Emiliano

INDICE

1 – Introduzione

Lo stage tenutosi presso i laboratori dell'INFN di Frascati si propone di sviluppare e implementare in Labview un tool di diagnostica per CPU su bus VME. Il luogo in cui si è tenuto il nostro stage è la sala di controllo dell'acceleratore Dafne, più precisamente nella sala LAT. Durante lo stage abbiamo interagito con vari dispositivi come *server* e *workstation* Solaris, dispositivi industriali in bus VME , Labview, processore RabbitCore RCM2100 e rete Ethernet. Il nostro obbiettivo era quello di riuscire a creare un pannello di controllo in Labview in grado di elaborare, visualizzare e comunicare attraverso la rete Ethernet i dati acquisiti da un processore Rabbit e i parametri di lavoro delle CPU remote di acquisizione. Per fare ciò è stato necessario creare dei driver per il processore Rabbit in modo da rendere possibile la comunicazione via rete, poi abbiamo collegato al processore un convertitore ADC il quale legge la tensione e contribuisce con le altre porte del processore, ad inviare i dati nella rete in modo da poter rendere utilizzabili i dati al pannello di controllo in Labview. Per realizzare quanto richiesto, è stato necessario fissare alcuni aspetti concettuali e teorici di base in modo da poter lavorare con una certa facilità. Purtroppo le nostre conoscenze scolastiche non ci permettevano di realizzare il nostro progetto (essendo principalmente nozioni concettuali e non pratiche), e allora è stato necessario per i nostri tutor effettuare una introduzione teorica, molto dettagliata, al nostro lavoro. La parte teorica affrontata nello stage riguarda:

- Sistemi Operativi (Unix, Solaris);
- Architettura dei bus di comunicazione dati;
- Il bus VME;
- Le reti (struttura ISO/OSI e Protocollo TCP/IP);
- Labview
- Architettura del microprocessore Rabbit

2 - Il sistema operativo Unix

Il primo argomento affrontato riguarda il Sistema Operativo Unix. Ma che cosa è un sistema operativo? In termini molto semplicistici si può affermare che un sistema operativo è l'insieme del software che gestisce l'hardware di un elaboratore. Dove per software si intendono i programmi realizzati per essere eseguiti su un dato elaboratore dotato di una o più CPU che usa un insieme definito di istruzioni di macchina; mentre con il termine hardware si identifica l'insieme delle parti elettriche, elettroniche, meccaniche e ottiche che costituiscono la parte fisica di un elaboratore. Approfondendo la definizione del sistema operativo dal punto di vista dell'utente, esso è l'insieme del software che rende semplice e conveniente la gestione di un elaboratore ed utilizza l'hardware secondo modalità trasparenti all'utente in modo da ottenere le massime prestazioni complessive. In altri termini si dice anche che il compito del sistema operativo è quello di gestire le risorse di un elaboratore, dove con il termine risorse sono identificati i seguenti componenti hardware:

1. Processori
2. Memoria primaria e secondaria
3. Dispositivi di I/O
4. Dispositivi di comunicazione

Esistono vari tipi di sistema operativo:

1. sistemi monolitici
2. sistemi stratificati
3. sistemi virtuali

Unix è un sistema operativo che rientra nella seconda categoria. Unix infatti presenta una struttura a strati che permette l'iterazione solo tra strati adiacenti. Unix è un sistema operativo molto potente e multi utente, perchè permette all'utente che possiede i diritti di interagire direttamente con l'hardware con una certa semplicità.
Per comprendere al meglio il funzionamento del Sistema Operativo Unix è necessario conoscere le componenti più importanti del Sistema Operativo:

_ il File System
_ la Shell

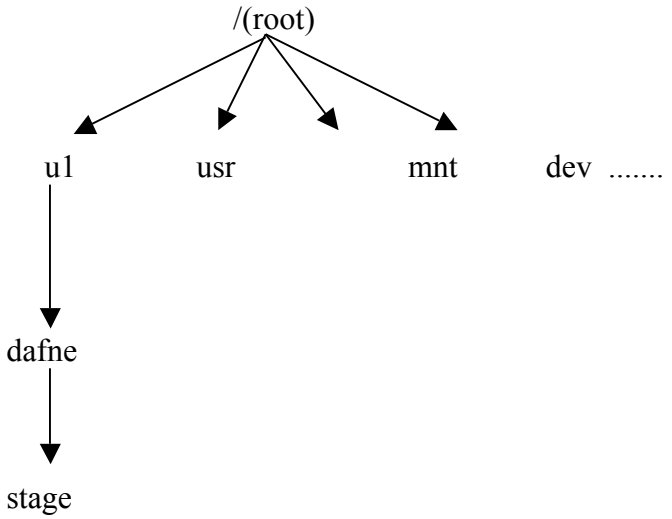
1 - In Unix tutte le informazioni, i dati, i programmi, i dispositivi di I/O ed il sistema operativo stesso sono organizzati come file e visibili secondo una struttura gerarchica, che è appunto, **il File System.** Quindi in Unix tutto è gestito come un file. Per capire bene come funziona e cosa sia il File System Unix è quindi necessario dare una definizione di file. Il file è una struttura di tipo avanzato, come l'array e il record,che ha cardinalità (numeri che lo compongono) infinita. La forma più semplice di un file è quella

sequenziale. La sequenza è un organizzazione di dati elementari. Il File System Unix è di tipo gerarchico e presenta una struttura di tipo ad albero che ha una radice (root) che si dirama ad albero indicando tutte le directory. Per raggiungere un file o una qualsiasi directory, esistono due modi:

_Assoluto: fa riferimento alla radice e vuole l'intero percorso per raggiungere il file che ci interessa. Per esempio volendo raggiungere la sotto directory dafne contenuta in /u1 si può digitare /u1/dafne .

_Relativo: non inizia mai con /, e per arrivare al file però si deve essere già nella directory esatta e poi basta aggiungere il nome del file.

Tutto ciò è possibile grazie alla gestione dell'hard disk da parte del File System. Infatti l'H.D. è diviso in quattro blocchi:
Blocco di boot che contiene il codice di boostrap che serve per far partire il Sistema Operativo, Super Blocco che descrive lo stato del file system (la sua grandezza in byte, quanti file può contenere, dove reperire lo spazio libero, ecc.), la lista i-node dove sono contenuti gli i-node che realizzano la struttura logica del file system Unix (perchè ogni file e ogni directory è identificata da un i-node), il blocco dati che contiene i dati dei file e delle directory e questo è uno dei principali motivi per cui il file system è sotto forma di albero.



Schema del file system gerarchico di unix

2**- La shell** è l'interfaccia dell' utente con il sistema operativo infatti con la shell si possono inviare comandi al sistema operativo. In Unix esistono due tipi di shell: grafiche e (GUI: Graphic User Interface) e non grafiche (CLI: Command Line Interface).
I principali comandi della Shell Unix sono:
ls = fornisce l'elenco dei file contenuti nella directory corrente;
ls -a = permette di vedere i file nascosti e molti dettagli;
ls –F = per visualizzare i diversi tipi di file;
ls –l = per informazioni sui diritti dei file e delle directory;
cd <directory> = permette di muoversi nella directory indicata;
cd ~ = ovunque ci si trovi riporta alla directory home;

cd .. = permette di spostarsi nella directory immediatamente precedente;
cd / = permette di tornare alla root;
man = apre la pagina informazioni del comando digitato;
who = fornisce l'elenco degli utenti connessi al sistema;
& = scritto alla fine di un comando permette l'esecuzione di un programma in *background*;
kill = permette di uccidere un processo;

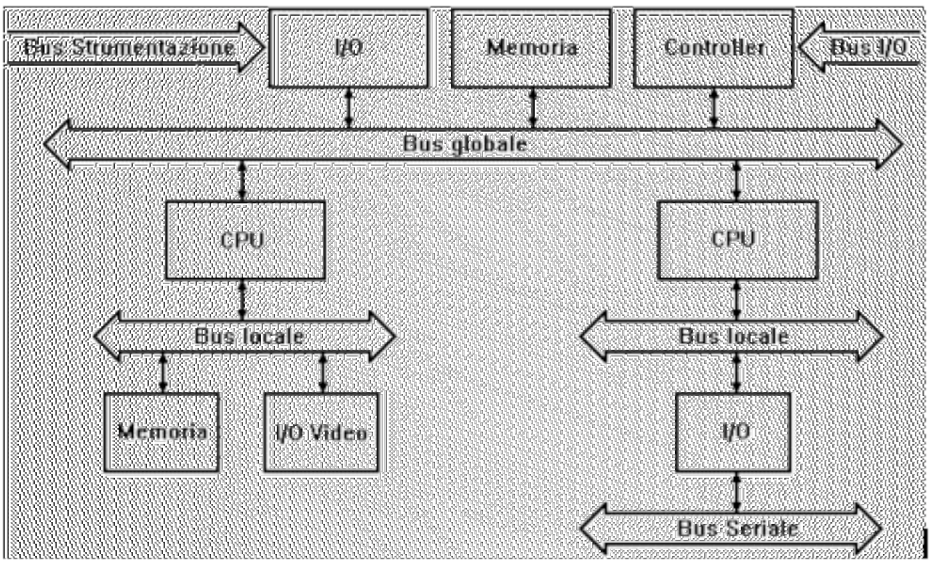
I principali comandi per operare con i file e con le directory:

mkdir <directory> = crea una directory con il nome indicato;
rmdir <directory> = cancella la directory con il nome indicato;
rm <file> = cancella il file con il nome indicato;
cp <file sorgente> <destinazione> = copia e incolla il file nella destinazione indicata;
mv <file sorgente> <destinazione> = taglia e copia il file nella destinazione indicata;

Unix, come già detto è un sistema multiutente, ossia permette l'accesso con la stessa macchina a più utenti, ovviamente questa caratteristica ha i suoi vantaggi e i suoi svantaggi, infatti se i diritti dell'utente fossero gestiti male questo permetterebbe a qualsiasi utente di entrare nel sistema e causare danni anche irreparabili. Per risolvere il problema il Unix associa ad ogni file 3 tipi di diritti: R= read, W= write e X= execute, e riconosce 3 tipi di utenti:il proprietario del file (Owner:u) un utente dello stesso gruppo del proprietario del file (Group:g) e un utente generico (Others:o). In questo modo il proprietario del file può scegliere quale utente del sistema ha i diritti desiderati su ogni file. L'utente root è l'unico che può gestire la macchina al 100%. L'istruzione che permette di cambiare i diritti da parte della root è **chmod < attributo><nome file>**.

3 - Il bus

Il bus è un elemento molto importante nell'architettura di un calcolatore, i bus nascono dal esigenza di dover collegare tra loro i vari elementi di un calcolatore (CPU, memoria, unità di I/O, etc..). Un bus ha molti vantaggi: in particolare permette di mettere insieme parti costruite da case diverse e questo permette un gran risparmio di tempo e denaro. Quindi per le grandi industrie serve che il bus sia standardizzato.



Architettura di un calcolatore

Il disegno in figura è un esempio generico di bus.
In base al tipo di trassmissione che utilizzano i bus possono essere classificati in:

- Bus paralleli: corti,veloci e in grado di inviare molte informazioni contemporaneamente, come il bus VME.
- Bus seriali: economici e in grado di coprire distanze apprezzabili, ma capaci di trasmettere solo un bit alla volta, come Ethernet.

Anche la trasmissione permette di fare una nuova divisione dei bus:

- Sistemi single-master: in cui è previsto un solo master(che svolge le funzioni di CPU) e più slave (che ricevono i comandi);
- Sistemi multi-master: in cui un controller si occupa di gestire l'arbitraggio tra i vari dispositivi master e slave presenti quando essi richiedono l'accesso al bus;

L'ultimo parametro che serve per classificare i bus è la sincronizzazione del trasferimento infatti esistono:

- Bus sincroni: in cui esiste un segnale di riferimento temporale, il clock, a scandire la successione degli eventi. I segnali agiscono in corrispondenza del clock e devono avere tutti la stessa velocità.

- Bus asincroni: in cui la successione degli eventi è scandita dai sottosistemi che trasferiscono i dati. L'unico vincolo nella trasmissione è il tempo di timeout entro cui i dispositivi devono rispondere.

In un bus, per eseguire il trasferimento di un dato servono oltr alle linee di alimentazione anche fili che portano l'indirizzo, il dato stesso, le linee di interrupt, le linee di arbitraggio etc...
Le linee più importanti in un bus sono:
DATA: sono le linee che trasmettono le informazioni
DATA STROBE: stabiliscono le modalità di trasferimento. Ciascun standard adotta un metodo diverso per compiere questa operazione. Ad esempio il bus VME utilizza quattro segnali distinti, ognuno dei quali controlla l'accesso a un byte della parola. La selezione di ampiezze dati maggiori di un byte avviene attivando più segnali contemporaneamente.
ADDRESS: specificano gli indirizzi che individuano l'unità di I/O;
ADDRESS STROBE: comunica se ciò che è in transito nel bus è un segnale valido o meno . Tipicamente, questo segnale rimane a livello durante l'intera trasmissione;
ARBITRAGGIO: decidono chi ha l'accesso al bus in caso di più master che ne chiedono simultaneamente l'utilizzo, evitando che questi ultimi vadano in conflitto.

Alcuni Bus inoltre sono anche multiplexati: ossia più segnali sono riservati ad un'unica linea, ed un'unica linea trasmette alternativamente segnali il cui significato varia nel tempo. Questo procedimento è vantaggioso perchè riduce il numero delle linee,le dimensioni ed il costo del bus.

Esempi di schede per bus industriali

4-Le Reti

Una rete è un particolare infrastruttura che permette la comunicazione tra più macchine (stampanti, server, calcolatori) contemporaneamente. Le reti, a seconda della loro estensione geografica possono essere divise in:

1. **LAN**: Le reti locali (LAN), sono il più comune tipo di rete presente nei piccoli uffici con le seguenti caratteristiche:

• opera su di un'area ristretta. Può trattarsi di un piano di un edificio o di un singolo edificio;

• gli host all'interno della LAN sono collegati tra di loro per mezzo di connessioni di rete ad elevata capacità di banda, tipo ethernet o token ring;

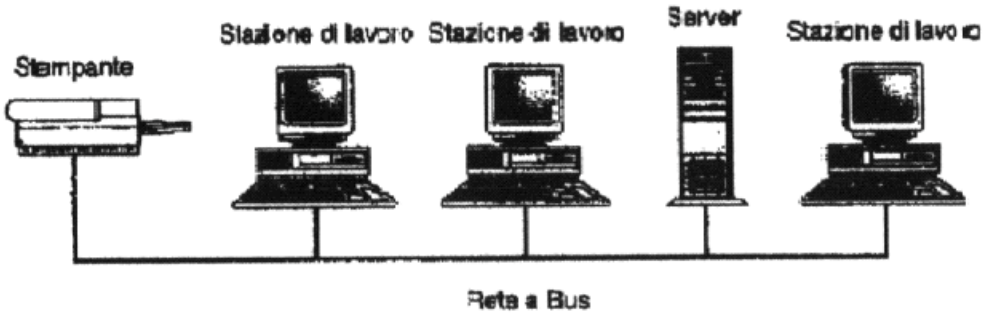
• spesso, tutti gli aspetti della rete locale sono gestiti privatamente. Non occorre l'intervento di terze parti per le soluzioni che riguardano la connettività;

• i servizi delle reti locali sono disponibili 7 giorni alla settimana, 24 ore su 24.

Ora che si è parlato delle reti LAN prima di proseguire con gli altri tipi di rete è giusto parlare della rete ETHERNET.

Ethernet è il più diffuso tipo di rete locale esistente al mondo. Però il fatto che sia diventato uno standard internazionale, non implica che essa sia la migliore tecnologia in assoluto. Sicuramente, però, si tratta della più economica e della più semplice da implementare. Originariamente, Ethernet, utilizzava un solo cavo per collegare decine di stazioni di lavoro, ciascuna delle quali riceveva contemporaneamente (o quasi) tutto quello che passava sulla rete. Solo una stazione alla volta, invece, aveva la possibilità di trasmettere. Si tratta della topologia a bus, abbinata alla tecnica CSMA/CD. In realtà, al giorno d'oggi, Ethernet gestisce alla perfezione anche reti LAN basate su di una topologia a stella. Il CSMA/CD, comunque, rimane la prerogativa preponderante di questo standard.

La rete a bus é il metodo più semplice utilizzato per mettere in rete i computer. Consiste in un singolo cavo che connette tutti i computer, i server e le varie periferiche in un singolo segmento di rete. Gli host su una rete a bus, comunicano tra loro mettendo le informazioni sul cavo, indirizzate all'indirizzo fisico della scheda di rete usata per connettere il computer destinatario al segmento di rete. Questo indirizzo fisico prende il nome di indirizzo Media Access Control (MAC). I dati messi sulla rete vengono inviati a tutti i computer che fanno parte della rete stessa. Ciascun computer esamina questi dati, per scoprire se l'indirizzo di destinazione delle informazioni corrisponde al proprio indirizzo MAC. In caso affermativo, il computer legge le informazioni, altrimenti le scarta. Le reti ethernet costituiscono l'implementazione più comune delle reti a bus. Esse si servono di un metodo chiamato "Carrier Sense Multiple Access with Collision Detection" (CSMA/CD). Ciò significa che un solo computer per volta può inviare dati sulla rete a bus. Se un host volesse trasmettere dati e scopre che vi sono già altri dati in transito sulla rete, esso deve aspettare che questa sia libera prima di trasmettere le sue informazioni.



Rete a bus

Se due host iniziano contemporaneamente a trasmettere dati sulla rete, avviene un fenomeno chiamato “collisione”. Gli host possono accorgersi di questa situazione ed inviare sulla rete un segnale di ingorgo. Questo fa sì che la collisione duri abbastanza a lungo perché tutti gli altri host la riconoscano. Ciascun host trasmittente aspetta un periodo di tempo casuale prima di tentare di nuovo l’invio dei dati. Questo intervallo di tempo è reso casuale per ovvi motivi. Se infatti due host rilevassero una collisione e dopo un periodo di tempo T, uguale per tutti, tornassero entrambi ad inviare il messaggio, vi sarebbe nuovamente una collisione. Si creerebbe insomma un ciclo di durata virtualmente infinita, durante il quale tutti gli host sarebbero esclusi dalla rete.

2. RETE WAN: Le implementazioni delle reti locali, soffrono di limitazioni fisiche e geografiche. Col passare del tempo aumentano i fabbisogni di collegamenti in rete che prevedono la connettività su distanze ben maggiori di quelle tra le singole stanze di un edificio. Molte reti geografiche estese (WAN) sono semplici combinazioni di reti locali e collegamenti aggiuntivi per le comunicazioni tra le varie LAN. Per descrivere la portata o le dimensioni della WAN, si usano i seguenti termini:

• *Reti di area metropolitana:* le MAN (Metropolitan Area Network) sono WAN disposte in piccole aree geografiche. Dimensionalmente possono essere identificate come reti che collegano singole città o regioni;

• *Reti universitarie:* le CAN (Campus Area Network) sono di solito WAN che collegano fra loro dipartimenti e facoltà universitarie.

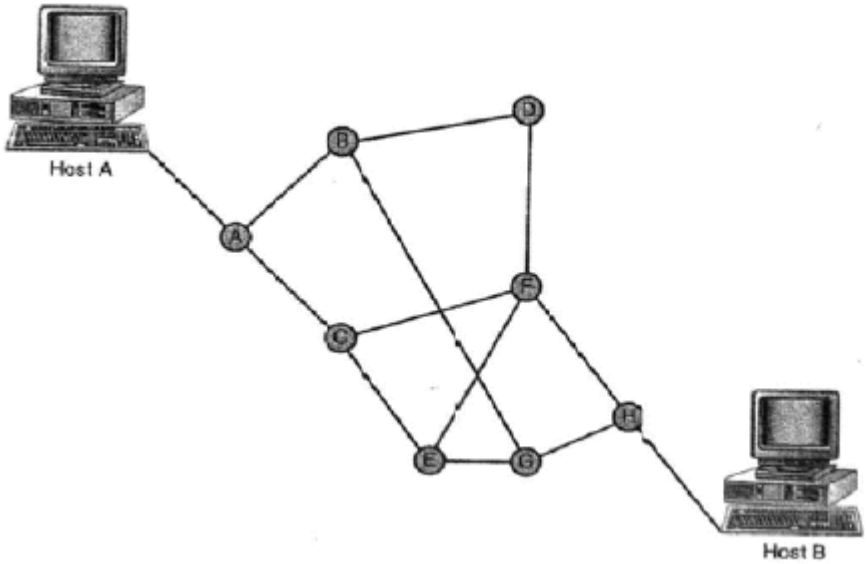
Per cercare di dimensionare a dovere i due concetti qui sopra esposti potremmo arrivare a dire che:

• le MAN sono reti che ricoprono circa l’area di una grossa città e/o di una piccola provincia;

• le WAN sono reti molto più estese, che possono interessare anche una o più regioni di medie dimensioni.

Per l’implementazione pratica, queste reti apparentemente molto complesse, non differiscono molto da una LAN piuttosto estesa. Le comunicazioni su di una WAN, si servono principalmente della trasmissione a commutazione di pacchetto.

Le reti a commutazione di pacchetto consentono di trasmettere dati su una connessione “chiunque con chiunque”. A volte, una rete di questo tipo viene detta “rete ibrida”. Quando si trasmettono le informazioni sulla rete, non è possibile sapere in anticipo quale sarà il percorso che intraprenderanno i dati nel raggiungere il destinatario. I dati originali sono suddivisi in pacchetti più piccoli, ciascuno dei quali è contrassegnato con l’indirizzo di destinazione ed un numero sequenziale. Quando il pacchetto attraversa la rete tra l’host di origine e quello destinatario, viaggia sul miglior percorso disponibile al momento della spedizione. In questo modo, se un collegamento della rete si interrompe durante la trasmissione del flusso di pachetti, non occorre inviarli tutti una seconda volta, poiché alcuni avranno trovato una strada alternativa quando il collegamento si è interrotto. La figura disegnata alcune righe sopra, mostra i possibili percorsi tra l’host A e l’host B.



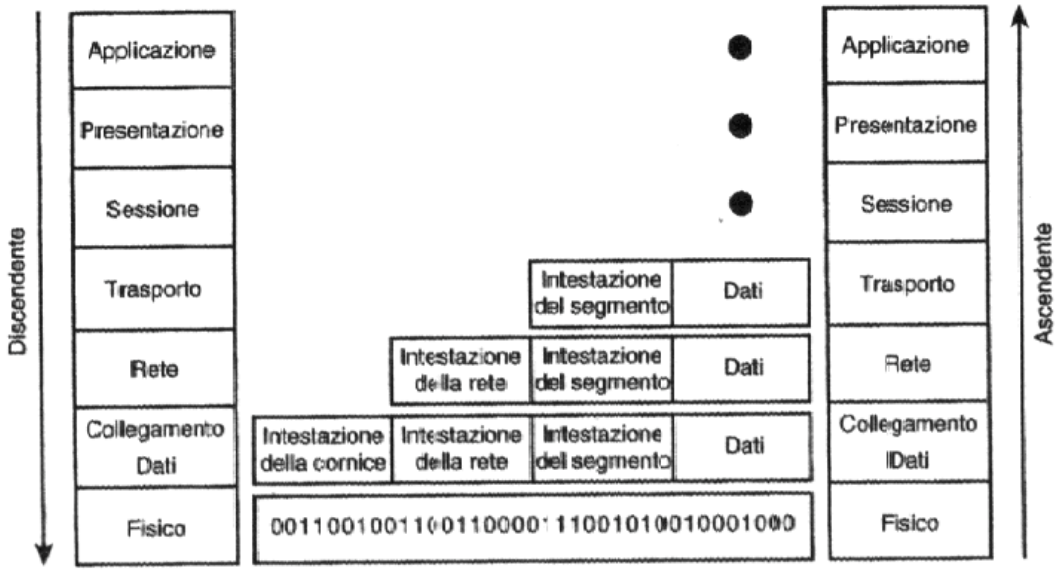
Esempio di rete a commutazione di pacchetto

Ipotizziamo che, in una condizione identica a quella del diagramma, un pacchetto sia stato instradato da A a B attraverso le reti che si trovano in A,C,F ed H. Se la rete che si trova in F si blocca, i pacchetti che sono arrivati alla rete C devono trovare una strada alternativa per la rete H. Una alternativa possibile è attraversare le reti E e G, arrivando così ad H. All'host destinatario, i pacchetti potrebbero arrivare in momenti diversi o comunque non in sequenza. Poiché, però, ogni pacchetto è contraddistinto da un numero sequenziale, il messaggio originale si può ricostituire senza sbagliare. L'host destinatario può anche richiedere un nuovo invio dei pacchetti eventualmente persi, in base ai numeri mancanti nella sequenza. Le reti a commutazione di pacchetto sono rapide ed efficienti, avendo un loro metodo per gestire il traffico di instradamento. Dopo aver parlato delle reti ora si può parlare dell'insieme delle regole, e delle operazioni da svolgere per far parlare tra loro due o più macchine ossia i protocolli. Due tra i principali tipi standard sono: il modello ISO-OSI e il modello TCP/IP.

- **Modello ISO/OSI**

Il modello di riferimento OSI é costituito dai seguenti sette strati distinti:
1) Strato fisico
2) Strato del collegamento dati
3) Strato della rete
4) Strato del trasporto
5) Strato della sessione
6) Strato della presentazione
7) Strato dell'applicazione

Esso descrive il modo in cui le informazioni si fanno strada da un'applicazione su di un host ad un'applicazione su di un altro host. Mentre le informazioni discendono attraverso gli strati della rete sull'host mittente, cambiano il proprio formato in ciascuno strato. I dati che provengono dagli strati superiori sono incapsulati in informazioni di intestazione dallo strato immediatamente inferiore



Modelli ISO-OSI

Questo diagramma mostra che, come i dati discendono attraverso l'host a sinistra, la combinazione tra l'intestazione dello stato precedente ed i dati é incapsulata nell'intestazione dello strato successivo; per esempio i dati originali di un messaggio e-mail sono incapsulati in un'intestazione del segmento. Questa assicura che gli host coinvolti nel messaggio siano in grado di comunicare in modo attendibile l'uno con l'altro. Al livello della rete, i dati (che ora comprendono l'intestazione del segmento ed i dati originali ricevuti dagli strati superiori) si sistemano in un pacchetto che contiene un'intestazione di rete. Questa intestazione di rete comprende gli indirizzi logici dell'origine e della destinazione. In una rete interconnessa TCP/IP, sono gli indirizzi IP

degli host mittente e destinatario ed aiutano nell'instradamento dei pacchetti tra i due host attraverso la rete. Allo strato del collegamento dati, l'intestazione della rete e i suoi dati sono incapsulati in un'intestazione della cornice, che definisce il modo in cui le informazioni saranno trasportate attraverso l'interfaccia della rete sulla rete fisica. Ciascun dispositivo sulla rete richiede l'inclusione in una cornice per connettersi con il dispositivo successivo della rete. L'intestazione della cornice include anche gli indirizzi fisici degli host mittente e destinatario. Infine, allo strato fisico, l'intestazione della cornice ed i suoi dati si convertono in un formato che consente la trasmissione delle informazioni su di un mezzo come i cavi di rete. Quando i dati arrivano all'host destinatario, i bit sono nuovamente convertiti in un'intestazione della cornice con i suoi dati. Quando le informazioni si spostano verso l'alto attraverso gli strati della rete, ciascuna intestazione serve a determinare in che modo spostare i dati verso gli strati superiori. A ciascuno strato si staccano le informazioni di intestazione dello strato precedente, in modo tale che i dati abbiano di nuovo lo stesso formato che avevano al momento della trasmissione dallo strato corrispondente dell'host mittente. Ciascuno strato deve compiere una funzione predeterminata. Mentre i dati discendono attraverso gli strati, l'intestazione ed i dati provenienti dal livello superiore diventano la sezione dei dati del livello immediatamente inferiore. I dati non possono saltare uno strato, mentre scendono attraverso il modello OSI. Questo semplifica il processo della trasmissione e consente lo sviluppo di nuovi protocolli, poiché questi devono semplicemente interagire con gli strati sopra e sotto a quello in cui sono implementati. Ora per capire meglio il funzionamento della ISO/OSI descriveremo tutti gli strati.

- **Lo strato fisico** definisce le correnti elettriche, gli impulsi fisici o gli impulsi ottici che sono coinvolti nel trasporto dei dati. Lo strato fisico è responsabile per la trasmissione fisica dei bit da un computer all'altro.

- **Lo strato del collegamento** dati invia i dati dallo strato della rete allo strato fisico. Quando lo strato del collegamento dati riceve i bit dallo strato fisico, li traduce. Una cornice dati in genere comprende le seguenti componenti:
• *ID destinatario:* questo ID, di solito, é l'indirizzo dell'host di destinazione o del gateway predefinito
• *ID mittente:* questo ID, di solito, é l'indirizzo dell'host sorgente
• *Informazioni di controllo:* includono informazioni e notizie sull'instradamento e la segmentazione
• *Controllo ciclico di ridondanza:* effettua la correzione degli errori e verifica che i dati siano arrivati intatti all'host destinatario di riferimento

- Lo strato della rete determina il modo migliore per spostare i dati da un host all'altro. Gestisce l'indirizzamento dei messaggi e la traduzione degli indirizzi logici (come per esempio gli indirizzi IP) in indirizzi fisici. Lo strato della rete determina anche la strada che i dati percorrono tra l'host mittente e quello destinatario. Se i pacchetti in corso di trasmissione sono troppo grandi per l'host destinatario, lo strato della rete compensa, suddividendo i dati in pacchetti più piccoli, che saranno poi riassemblati una volta giunti a destinazione.

- **Lo strato del trasporto** segmenta e riassembla i dati in un flusso di dati. Provvede ad una connessione da un capo all'altro tra l'host mittente e quello destinatario. Quando si

verifica una trasmissione di dati dall'uno all'altro, tali dati sono segmentati in insiemi di informazioni più piccoli. I segmenti ricevono una numerazione sequenziale e sono inviati all'host destinatario. Quando questo riceve i segmenti, invia una conferma. Nel caso che un segmento non arrivi, l'host destinatario può richiede un secondo invio del segmento specifico. In questo modo sussiste un controllo degli errori per il trasporto dei dati.

- **Lo strato della sessione** consente a due applicazioni su host separati di stabilire una sessione. Le sessioni assicurano che i messaggi siano inviati e ricevuti con un alto grado di attendibilità. Lo strato della sessione svolge funzioni di sicurezza, per assicurare che due host siano autorizzati a comunicare attraverso la rete. Lo strato della sessione coordina le richieste di servizio e le risposte che hanno luogo quando le applicazioni comunicano tra gli host.

- Lo strato della presentazione determina il modo in cui i dati sono formattati nello scambio tra computer in una rete. I dati ricevuti dallo strato dell'applicazione sono tradotti in un formato intermediario comunemente riconosciuto. Lo strato della presentazione è anche responsabile per tutte le traduzioni e le codifiche dei dati e per le conversioni dei set di caratteri e dei protocolli. E' inoltre responsabile per le conversioni della sintassi tra due host in comunicazione, per esempio nel caso che uno degli host usi lo standard ASCII per le sue rappresentazioni di testo e dati, mentre l'altro host si serva di EBCDIC.

- Lo strato dell'applicazione consente ai programmi di accedere ai servizi di rete. Non ha a che fare con i programmi che necessitano solo di risorse locali. Per usare lo strato dell'applicazione, un programma deve avere una componente della comunicazione che richiede risorse di rete.
Di seguito, si trova un elenco di tipi di programmi che coinvolgono lo strato dell'applicazione:
• Posta elettronica
• Applicazioni per le teleconferenze
• World Wide Web (WWW)

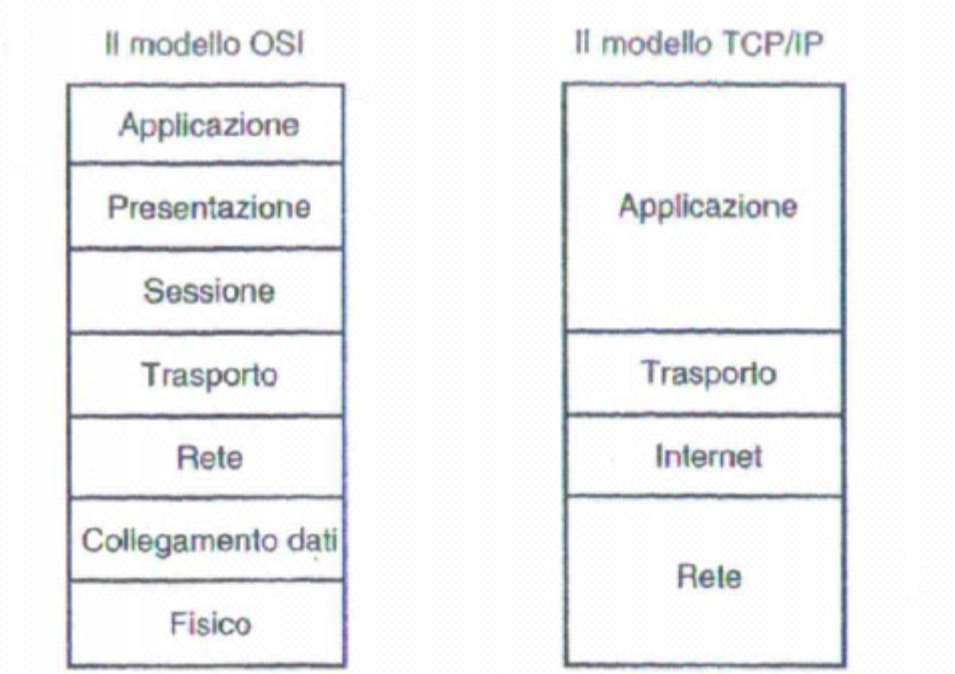
- **Modello TCP/IP**

Il nome completo è "TCP/IP Internet Protocol Suite" ed è un insieme di protocolli di trasmissione, i cui due principali sono appunto il TCP (Transmission Control Protocol) e l'IP (Internet Protocol). Tra il modello di riferimento OSI a sette strati ed il modello TCP/IP a quattro strati, si possono fare i seguenti confronti:
- Il modello stratificato TCP/IP unifica lo strato fisico a quello del collegamento dati del modello OSI, nello strato della rete TCP/IP. Non fa distinzione tra le schede di rete in senso fisico ed i loro driver e questo permette di implementare il TCP/IP in qualunque topologia di rete;

- Lo strato Internet del modello TCP/IP corrisponde allo strato della rete del modello di riferimento OSI. Entrambi si occupano dei servizi di indirizzamento ed instradamento;

- Lo strato del trasporto in tutti e due i modelli permette che si stabiliscano sessioni di comunicazione da un capo all'altro tra due host;

- Lo strato della sessione del modello TCP/IP combina gli strati della sessione, della presentazione e dell'applicazione del modello OSI. Il modello TCP/IP comprende tutte le questioni che riguardano la rappresentazione dei dati e il mantenimento delle sessioni nell'ambito delle definizioni di una applicazione.



Confronto tra il modello ISO-OSI e il modello TCP-IP

Anche del modello TCP/IP presenteremo una descrizione degli strati che lo compongono:

- **Lo strato della rete** infila sul cavo le cornici (frame) in partenza e toglie dal cavo quelle in arrivo. Il formato utilizzato da queste cornici dipende dalla topologia di rete implementata. Lo strato della rete aggiunge un preambolo all'inizio della cornice, nonché un controllo ciclico di ridondanza per assicurare che i dati non si danneggino durante il transito. Se la cornice arriva intatta, viene trasmessa al livello superiore nel modello a strati della rete; se invece essa é danneggiata, a questo punto viene scartata e si rende necessaria una nuova spedizione da parte dell'host mittente.

- **Lo strato Internet** svolge tre funzioni principali: l'indirizzamento, la suddivisione in pacchetti e l'instradamento. L'Internet Protocol (IP) risiede in questo strato dell'insieme stratificato di protocolli TCP/IP. Questo significa che il protocollo IP non svolge alcun controllo o misurazione per assicurarsi che l'host destinatario abbia ricevuto con successo le informazioni. I pacchetti possono andare perduti o arrivare non in sequenza. Quando le informazioni arrivano dallo strato del trasporto, il protocollo IP vi aggiunge un'intestazione che contiene le seguenti notizie:

• *Indirizzo IP dell'origine*: indirizzo IP assegnato all'host mittente;

• *Indirizzo IP della destinazione*: indirizzo IP assegnato all'host destinatario;

• *Protocollo del trasporto:* il protocollo usato dallo strato del trasporto é immagazzinato all'interno dell'intestazione IP. In questo modo, quando il diagramma arriva al sistema dell'host, lo strato Internet sa se trasferirlo utilizzando il protocollo TCP o il protocollo UDP;

• *Somma di controllo*: assicura che i dati che arrivano a questo strato non si siano danneggiati durante il transito;

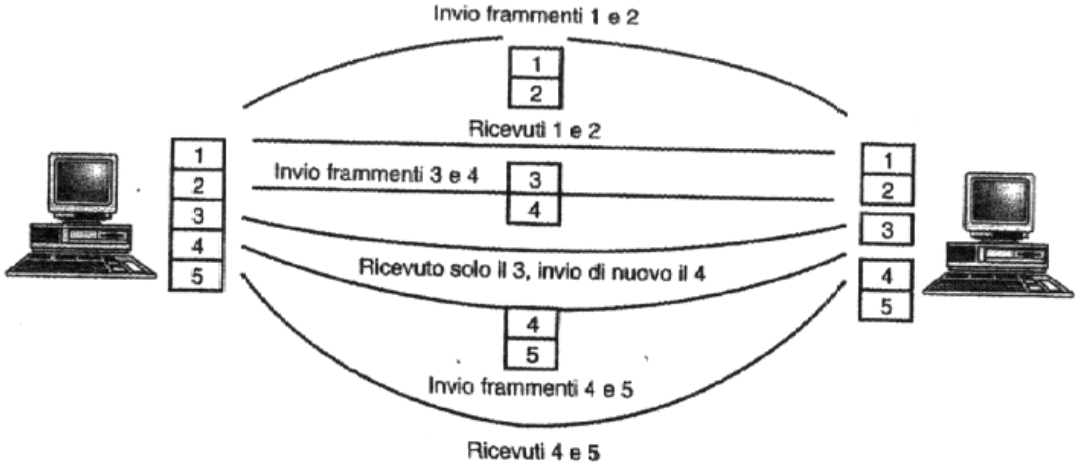
Altri processi che avvengono nello strato Internet sono la *frammentazione* ed il *riassemblaggio*. A volte, quando le informazioni sono trasferite tra i segmenti di rete, questi possono non servirsi della stessa topologia di rete. La topologia di rete del ricevente può non lavorare con la medesima dimensione del datagramma che usa la rete dell'host mittente. In questo caso, IP scompone i dati in pezzi più piccoli. Quando i dati arrivano all'host destinatario, i pezzi più piccoli vengono riassemblati,nel pacchetto dati originale.

- **Lo strato del trasporto** fornisce una comunicazione da un capo all'altro tra gli host utilizzando le *porte*. Nel modello a strati del TCP/IP, si trovano i seguenti due protocolli con il compito di trasportare i dati:

- Trasmission Control Protocol (TCP)

- User Datagram Protocol (UDP)

Il TCP si occupa delle comunicazioni orientate alla connessione su una rete TCP/IP. Quando due host comunicano utilizzando il protocollo TCP, bisogna che tra i due si stabilisca una sessione. Questo avviene perché ciascun host possa determinare il numero sequenziale successivo che l'altro host utilizzerà. Una connessione TCP garantisce un livello di attendibilità. Le trasmissioni si servono di numeri sequenziali e di conferme per assicurarsi che l'host destinatario riceva con successo i dati. Se infatti non riceve uno specifico segmento, può richiedere all'host mittente di inviarlo una seconda volta.



Esempio di trasmissione di pacchetti

Nella figura qui sopra, l'host sulla sinistra ha segmentato un pacchetto dati in 5 pezzi più piccoli. Invia i primi due all'host sulla destra, il quale, al momento della ricezione, dà una conferma che l'invio ha avuto successo. Quindi l'host sulla sinistra invia i due frammenti successivi (il 3 ed il 4). Per qualche ragione, l'host sulla destra riceve solo il terzo frammento. L'host sulla sinistra invia di nuovo il frammento 4 ed insieme il frammento 5. Al momento della ricezione, l'host destinatario dà la conferma per entrambi i frammenti. Ora può riassemblare i dati nel loro formato originale.

- Le applicazioni che si basano sulla rete lavorano sullo strato dell'applicazione nel modello stratificato TCP/IP. Il termine "applicazioni che si basano sulla rete" si riferisce a quelli applicazioni che si connettono o che comunicano con host su reti remote.

Non si può parlare delle reti senza parlare di indirizzi IP. Gli indirizzi IP identificano in modo univoco ciascun host su una internet TCP/IP. Un host può essere un computer o un terminale. In generale, possiamo dire che un host é un qualunque dispositivo fisico sulla nostra rete, che corrisponde ad una delle seguenti proprietà:

• si usa per accedere ad altri dispositivi sulla rete;
• ci si connette a questo dispositivo in qualità di componente condiviso della rete.

In una internet TCP/IP, ciascun host ha bisogno di un indirizzo IP che lo identifichi in modo univoco. Questo indirizzo IP, naturalmente, deve essere unico in tutta internet. La scelta degli indirizzi in una LAN é generalmente opera dell'amministratore, ma quando si ha a che fare con reti molto più estese geograficamente e come numero di utilizzatori, occorre una autorità che si occupi dell'assegnazione dei numeri. Per Internet, la *Internet Assigned Number Authority* (IANA) stabilisce le linee di condotta che riguardano l'assegnazione degli indirizzi IP ed è preposta a tale assegnazione. Ciascun indirizzo IP è una sequenza di 32 cifre binarie, quindi 1 o 0. Questo è il motivo per cui la versione corrente dell'attribuzione di indirizzi IP è conosciuta come assegnazione di indirizzi a 32 bit.
Gli indirizzi sono tuttavia anche divisi in tre classi (A,B,C) queste tre classi possono essere assegnate agli host su una rete. In ciascuna di queste prime tre classi di indirizzi IP, gli ID sono composti da una porzione relativa alla rete e da una relativa all'host.

Classi di indirizzamento IP

Ora descriviamo brevemente le caratteristiche delle tre classi:

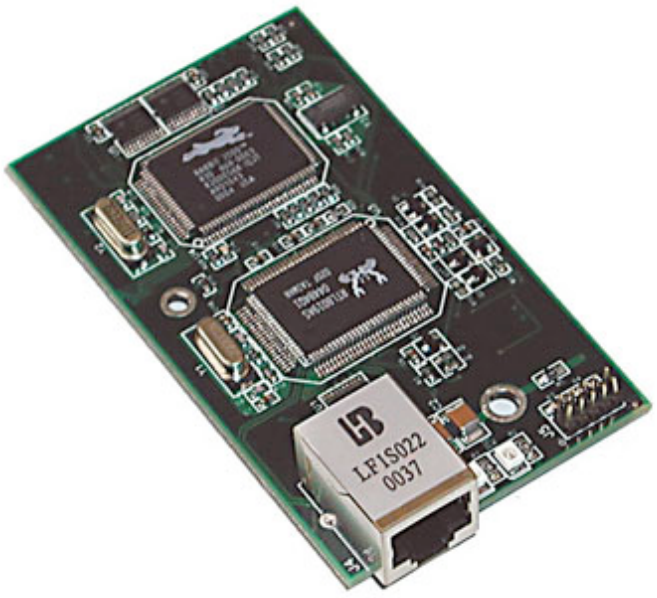
- Un indirizzo di Classe A, assegna 8 bit dell'indirizzo alla porzione della rete e 24 bit alla porzione dell'host. In un indirizzo di classe A, il valore del primo ottetto é compreso tra 1 e 126. Questi numeri sono rappresentati in binario. Questo consente 126 reti distinte di 16'774'214 host ciascuna. Questi numeri si ricavano mediante i seguenti calcoli:

- Un indirizzo di Classe B, assegna 16 bit alla porzione della rete e 16 bit alla porzione dell'host. Il valore del primo ottetto è compreso tra 128 e 191. Questi numeri sono rappresentati in binario e ciò si traduce in 16'384 reti univoche, con 65'534 host ciascuna.

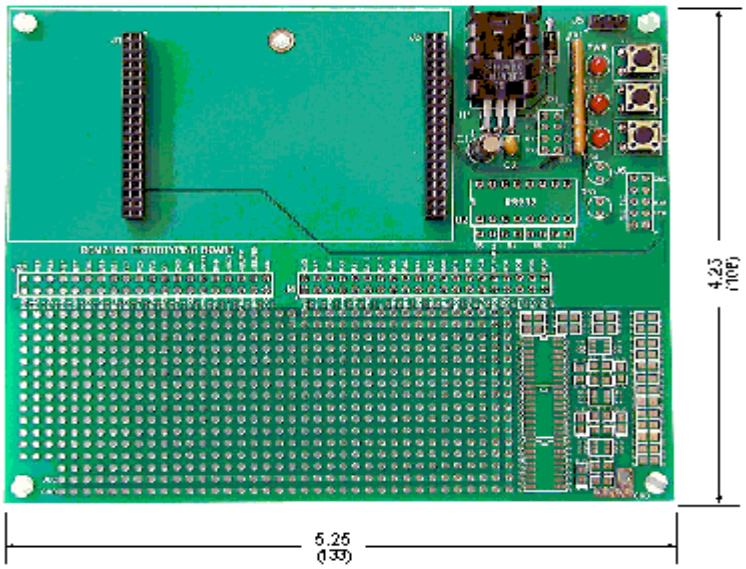
- Un indirizzo di Classe C, assegna 24 bit alla porzione della rete e 8 bit alla prozione dell'host. Il valore del primo ottetto è compreso tra 192 e 223. Questi numeri sono rappresentati in binario. Ciò si traduce in 2'097'152 reti univoche con 254 host ciascuna.

5-Rabbit core modules

I moduli Rabbit, sono delle schede già montate, di dimensioni molto contenute su cui è alloggiata tutta la circuiteria necessaria ad un completo sistema a microprocessore. Tra l'ampia gamma di modelli e versioni disponibili, esamineremo i versatili moduli **RCM2100** che rappresentano il modello più economico della gamma tra quelli dotati di connessione **Ethernet**. Esistono anche moduli più potenti dotati di CPU a 16 bit a clock più elevato e moduli più economici, più piccoli e privi di connessione Ethernet.



Modulo Rabbit RCM2100



RCM2100** Prototyping **Board

I moduli sono dotati, dal lato saldature, di due serie di connettori per poter essere ospitati direttamente sul circuito stampato della nostra applicazione in modo da poterla dotare di funzionalità di controllo e comunicazione avanzate senza dover affrontare la complessità di progettazione hardware e software che queste funzionalità richiedono. Per realizzare i nostri progetti dovremo semplicemente occuparci di realizzare l'interfaccia elettrica tra il

mondo esterno ed il modulo Rabbit e sviluppare in linguaggio C il programma adatto alla sua gestione senza preoccuparci dello sviluppo di una serie di dettagli hardware e software già risolti ed implementati sui moduli Rabbit quali:

- Le connessioni tra il microprocessore e le memorie per il programma ed i dati
- La circuiteria di clock
- La circuiteria per la connessione alla rete Ethernet
- La programmazione sul campo del microprocessore con il software della nostra applicazione
- La gestione dei protocolli di comunicazione, delle linee di I/O, delle porte seriali e tutta una serie di funzioni software già disponibili con le librerie in dotazione (complete di source).

L'AMBIENTE DI SVILUPPO DYNAMIC IN C SE

I moduli RCM2100 sono dotati di un ottimo ambiente di sviluppo integrato, disponibile solo per sistemi operativi Windows dalla versione 95 alla XP, ed una incredibile varietà di librerie di funzioni e applicazioni d'esempio.

Funzioni integrate del Dynamic C SE:

- Editor ASCII
- Compilatore C
- Debugger
- Loader (per scaricare il codice compilato sul modulo Rabbit)
- Documentazione in linea (solo in inglese)

Librerie di funzioni per la gestione dei protocolli TCP/IP

- **HTTP** client e server
- **FTP** client e server
- **TELNET** client e server
- **POP** client
- **SMTP** client

APPLICAZIONI PER LA QUALE E' CONSIGLIATO L'USO DEL RABBIT

I Core Modules Rabbit RCM2100 sono adatti per tutte quelle applicazioni di complessità intermedia in cui un semplice microcontroller (quale ad esempio un PICmicro) può risultare insufficiente per via della sua memoria troppo limitata, delle funzioni troppo elementari e dove, al contrario, un Personal Computer risulterebbe decisamente eccessivo per via delle sue dimensioni, il consumo di corrente, il costo, ecc.ecc. Nei core modules Rabbit coesistono caratteristiche tipiche dei microcontroller quali la memoria flash, l'assenza di parti meccaniche in movimento, le dimensioni ridotte, il basso assorbimento, le linee di I/O TTL, ecc. e caratteristiche tipiche dei Personal Computer moderni quali la connessione di Rete, la gestione dei protocolli TCP/IP, il file system (la capacità di

memorizzare informazioni sotto forma di file), l'ambiente di programmazione IDE in linguaggio C, il multitasking, ecc. I Rabbit sono facili da programmare ed alla portata non solo degli appassionati di elettronica ma anche degli sviluppatori di software su PC. L'uso del linguaggio C e la disponibilità di una nutrita serie di librerie, permettono lo sviluppo di applicazioni con rapidità nascondendo molti dettagli di implementazione a basso livello.

6-LabView: Introduzione

LabView è un ambiente di sviluppo software a livello grafico che conserva la flessibilità di un linguaggio di programmazione semplificando notevolmente l'implementazione di applicazioni scientifiche.
E' linguaggio di programmazione *data flow*, in cui cioè è il flusso di dati e non il tempo a scandire l'esecuzione delle operazioni.
Ciò consente lo svolgimento di molteplici operazioni in parallelo.
I programmi eseguibili con LabView sono detti *Virtual Instruments* (**VIs**) e includono un *Front Panel* e un *Block Diagram*:

- Il *Front Panel* (*Pannello Frontale*) rappresenta l'interfaccia utente del programma. Utilizzando gli strumenti predefiniti contenuti nella *Control palette* atti a misurare e automatizzare un segnale, si può creare e gestire un'interfaccia flessibile per numerose applicazioni.

- Il *Block Diagram* (*Diagramma a Blocchi*) contiene il source code (codice sorgente) grafico, ovvero una rappresentazione schematica e intuitiva della VI.

Accedendo alla ricca libreria di funzioni contenuta nella *Function palette* è possibile acquisire segnali (input); scrivere i dati su file, rappresentare i segnali sottoforma di funzioni grafiche, display o indicatori che li analizzino e li misurino (output); creare dispositivi di controllo; eseguire operazioni aritmetiche, booleane, comparative; convertire dati in stringhe, operare con stringhe; scambiare dati tra applicazioni di 2 host via TCP o UDP.

Pregi:

LabView rende più intuitivo e veloce lo sviluppo del software, poichè il programmatore non deve riscrivere il codice per le funzioni che vuole implementare, in quanto molte di esse sono già presenti nelle librerie del programma.
I tipi di dati gestiti dai programmi si differenziano visivamente in base al colore dei cavi che collegano le VI o le funzioni di libreria, ad esempio nel modo seguente:

- Percorsi (dati non standard, per es. indirizzi di memoria);
- Stringhe;
- Booleani (Vero o Falso)
- Linee di errore;
- Intero a 32 bit.

Per facilitare l'utilizzo delle funzioni di libreria, LabView possiede un efficace sistema di aiuto contestuale (*Context Help*): per avere informazioni su un qualsiasi componente (anche VI) del programma è sufficiente posizionare il cursore del mouse su di esso.

7-Progetto:Implementazione di un piccolo sistema di controllo su rabbit 2000

Il nostro progetto da una risposta specifica ad ogni richiesta (query) del Client, il microprocessore risponde in modo diverso a tre diverse richieste:

- Richiesta di dati (d)
- Richiesta di reset (r)
- Richiesta di test (t)

Viene creato un demone per controllare le richieste e leggere i valori di tensione dell' ADC.
Ma ora andiamo ad analizzare cosa viene inviato per risposta ad ogni singola richiesta

RICHIESTA DI DATI

La richiesta viene effettuata inviando al microprocessore il carattere 'd', che risponderà inviando un buffer contenente: dati digitali e i dati degli ADC.

RICHIESTA DI RESET

La richiesta viene effettuata inviando al microprocessore il carattere 'r', che eseguirà il reset del bus VME utilizzando il segnale di Sys Reset, come risposta all'esecuzione riuscita avvertirà con un segnale luminoso ed invierà al client il messaggio 'reset executed'.

RICHIESTA DI TEST

La richiesta viene effettuata inviando al microprocessore il carattere 't', che eseguirà un test per vedere lo stato di connessione del microprocessore, come risposta esatta alla richiesta avvertirà con un serie luminosa ed invierà al client il messaggio 'test ok'.

La comunicazione tra il microprocessore ed il client avviene seguendo il protocollo UDP/IP.
La gestione del protocollo è stata effettuata usando delle librerie già esistenti nel compilatore C.
Il programma inizia con l'apertura del socket, che ci avvisa tramite messaggio se l' "UDP Socket" si è aperto oppure no . Dopo di che si procede al collegamento con la conseguente apertura della porte tramite un ciclo dove vengono inviati 100 messaggi di ping, un altro messaggio ci avviserà della corretta apertura della porta.
Da qui incomincia un ciclo "FOR" infinito dove si interroga in continuazione la porta per vedere se ci sono dati. La prima istruzione è quella che va ad interrogare la porta dove risiede l'ADC per avere un aggiornamento continuo sulla tensione, poi si va ad interrogare la porta dove sono immessi i dati digitali che rappresentano gli stati del Bus VME, vengono monitorati i tre seguenti segnali:
-Sys Fail

-AC Fail
-Bus Err
Questi due dati vengono inseriti all'interno di un Array (buffer) di interi che viene aggiornato ad ogni ciclo del "FOR".
Appena arriva una richiesta sulla porta si va subito alla certificazione del dato, si controlla se corrisponde ad una delle tre opzioni gia elencate('a','r','t') e se la richiesta rispecchia uno dei tre casi il server invia al client i dati opportuni. Il server rimane poi in ascolto e continuerà ad aggiornare i dati dell'array aspettando una nuova richiesta. Tutto il programma è stato strutturato con l'utilizzo delle funzione per lasciare la sezione del "main", più pulita, e in questo modo se c'è bisogno si un comando già scritto non bisogna ripeterlo ma soltanto richiamare la funzione cambiando i parametri che gli passeremo.

Descrizione programma sorgente

```
void main()
{
   longword seq,ping_who;
   #GLOBAL_INIT
        {
   WrPortI(SPCR, &SPCRShadow, 0x84);// setup parallel port A as output
   WrPortI(PADR, &PADRShadow, 0xff);// turn off all LED's
             memset(x_buf, 0, sizeof(x_buf));
        }
```

In questa parte del “main” si setta porta parallela A come output Poi si mettono tutti i led della porta allo stato 0.

```
   datax[0]=1;
   datax[1]=70;
   datax[2]=5;
   datax[3]=16;          //inizializated array
   datax[4]=56;
   datax[5]=12;
   datax[6]=49;
   datax[7]=29;
   datax[8]=19;
   datax[9]=39;

   memmove (&buf_out,&datax,20); //copy to char buffer
   little2big(buf_out,20);
```

Qui invece si inizializza un Array di interi, che viene prima forzato in un Array di char di dimensione doppia, poiche un intero e’ formato da due byte ed un char da un byte, quindi di conseguenza per fare un intero occorrono due char. Nell’Array di char si invertono le posizioni dei byte poiche il rabbyt memorizza i dati in formato Little e Endian, per fare questa operazione si richiama la funzione “*little2big*” che inverte i byte.

```
   sock_init();
        printf("Opening UDP socket\n");

        if(!udp_open(&sock, LOCAL_PORT, resolve(REMOTE_IP), 0, NULL))
   {
             printf("udp_open failed!\n");
             exit(0);
        }
```

In questa parte del programma viene aperto il socket UDP.

```
        /* receive heartbeats */
   for(m=0; m<100; m++)
        {
        ping_who=resolve(PING_WHO);
      _ping(ping_who,seq++);
      }
      if(_ping(ping_who,seq++)!=0)
             {
             printf("error of connection on the door\n");
      exit (0);
      }
```

```
      printf("the door is open\n");
```

Qui si effettuano 100 comandi di ping per connettersi alla porta del server, e spedisce un messaggio sul video che ci avverte se la porta e' stata aperta o no.

```
for(;;)
   {
        read_ADC();
       read_portB();
               tcp_tick(NULL);
               if (1 == udp_peek(&sock, &udi))
       {
            answer_data();
         }

   }
}
```

Qui e' dove dentro il ciclo "for" infinito si interrogano le varie porte dove sono collegati, un ADC, dei dati digitali, e si interroga la porta di connessione per vedere se sono state inviate delle richieste al microprocessore. Tutti questi processi vengono effettuati tramite il richiamo di funzioni che ora andremo ad analizzare.

```
int read_ADC()
{
int j,k,data;
data=0;
for(j=0;j<14;j++)
{
BitWrPortI(PCDR, &PADRShadow, 0, adc_ck);
for(k=0;k<10;k++)
{};
BitWrPortI(PCDR, &PADRShadow, 1, adc_ck);
for(k=0;k<10;k++)
{};
if( (BitRdPortI(PCDR,3))==1)
{
switch(j){
case 0:
       break;

case 1:
       break;

case 2:
       data=data+2048;
       break;
case 3:
       data=data+1024;
       break;
case 4:
       data=data+512;
       break;
case 5:
       data=data+256;
       break;
case 6:
       data=data+128;
       break;
case 7:
       data=data+64;
```

```
       break;
case 8:
       data=data+32;
       break;
case 9:
       data=data+16;
       break;
case 10:
       data=data+8;
       break;
case 11:
       data=data+4;
       break;
case 12:
       data=data+2;
       break;
case 13:
       data=data+1;
       break;
       }//end switch
       }//end if
       }//end for
BitWrPortI(PCDR, &PADRShadow, 0, adc_ck);
datax[0]=data;
}
```

La funzione rappresenta il driver da noi creato che legge i valori dell'ADC, chiude ed apre la porta C, le operazioni vengono scandite da 14 cicli di clock, e da intervalli gestiti da due cicli "for", poi con un "if" si controlla se il 3° bit dell ADC e' uguale ad uno, se cio' e' vero in base al valore del contatore del ciclo addiziona alla variabile "data" un valore specifico, poi si continua col ciclo. Alla fine si richiude la porta definitivamente e si trasferisce il valore di "data" nella prima posizione dell'Array "datax".

```
int read_portB()
{
datax[1]=(RdPortI(PBDR) & 0x3F);
}
```

La funzione restituisce nella seconda posizione dell'array "datax" il valore della porta B.

```
int answer_data ()
{
         unsigned long int T0;
         n_byte=receive_packet(x_buf);


            if ((x_buf[0]== 'd')&(x_buf[1]== '\0')){
            rem_ip = udi.remip;
            rem_port = udi.remport;
            memmove (&buf_out,&datax,20); //copy to char buffer
             little2big(buf_out,20);
            send_packet(buf_out,20,rem_port, rem_ip);//(x_buf,n_byte);
            return 1;
            }
            if ((x_buf[0]== 'r')&(x_buf[1]== '\0'))
            {
            rem_ip = udi.remip;
            rem_port = udi.remport;
             BitWrPortI(PADR, &PADRShadow, 0, 1);// turn LED on
            T0=MS_TIMER;
            while((MS_TIMER-T0)<400){};
             BitWrPortI(PADR, &PADRShadow, 1, 1);// turn LED off
```

```
            send_packet("reset executed",14,rem_port, rem_ip);//(x_buf,n_byte);
            return 2;
            }
            if ((x_buf[0]== 't')&(x_buf[1]== '\0'))
            {
            rem_ip = udi.remip;
            rem_port = udi.remport;
                     BitWrPortI(PADR, &PADRShadow, 0, 0);// turn LED on
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                      BitWrPortI(PADR, &PADRShadow, 1, 0);// turn LED off
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                     BitWrPortI(PADR, &PADRShadow, 0, 1);// turn LED on
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};


                     BitWrPortI(PADR, &PADRShadow, 1, 1);// turn LED off
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                     BitWrPortI(PADR, &PADRShadow, 0, 2);// turn LED on
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                          BitWrPortI(PADR, &PADRShadow, 1, 2);// turn LED off
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                     BitWrPortI(PADR, &PADRShadow, 0, 3);// turn LED on
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                          BitWrPortI(PADR, &PADRShadow, 1, 3);// turn LED off
            send_packet("test ok",7,rem_port, rem_ip);//(x_buf,n_byte);
            return 3;
            }
            return 0;
}
```

La funzione controlla la richiesta inviata dal client, se la richiesta e' una di quelle che il microprocessore ha, invia la giusta risposta:
-per la richiesta di dati (d) invia l'array "datax" contenente i dati digitali e queli dell'ADC;
-per la richiesta di reset (r) accende e spenge un led e poi invia al client il messaggio "reset executed"
-per la richiesta di test (t) accende e spenge in sequenza dei led e poi invial al client il messaggio "test ok".

In appendice si trova il codice sorgente completo.

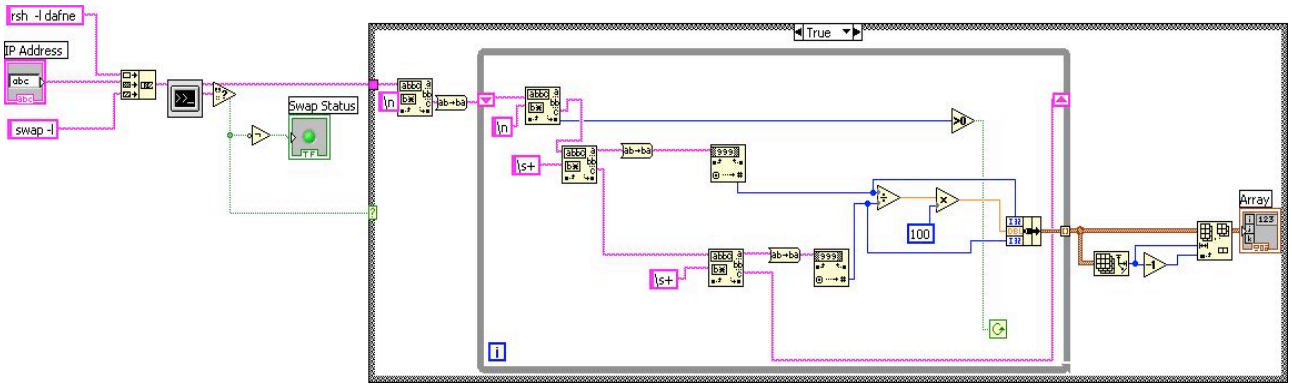
LabView: Esperienza pratica

Obiettivo dell'esperienza: Realizzare un tool di diagnostica per CPU installate su di un bus VME.

Impostazione dell'esperienza:
Dopo aver avuto nozioni teoriche riguardanti le reti e in particolare, per la nostra esperienza, il protocollo UDP/IP si è passati alla realizzazione di un algoritmo diviso nei seguenti sottoprogrammi:

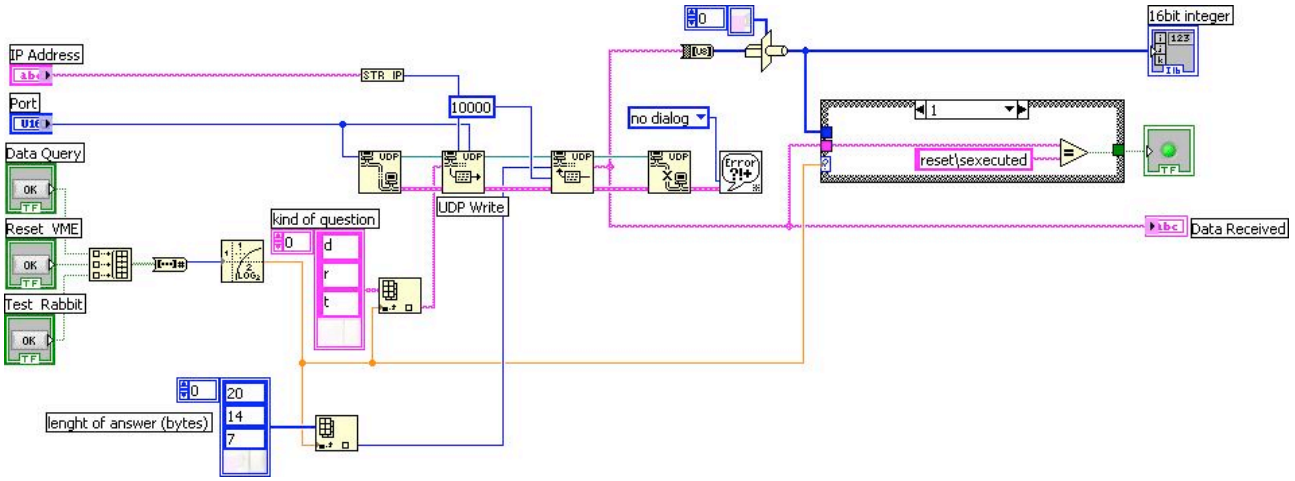
- Modulo SWAP (Calcolo della percentuale di memoria Swap disponibile)
- Modulo RABBIT QUERY (Interrogazione e Intepretazione dei dati ricevuti dal microprocessore Rabbit)
- Modulo FAN & TEMPERATURE (Calcolo della percentuale dell'utilizzo delle ventole e della temperatura di una o più CPU)
- Pannello di controllo principale (Permette la visualizzazione delle informazioni elaborate)

MODULO SWAP:



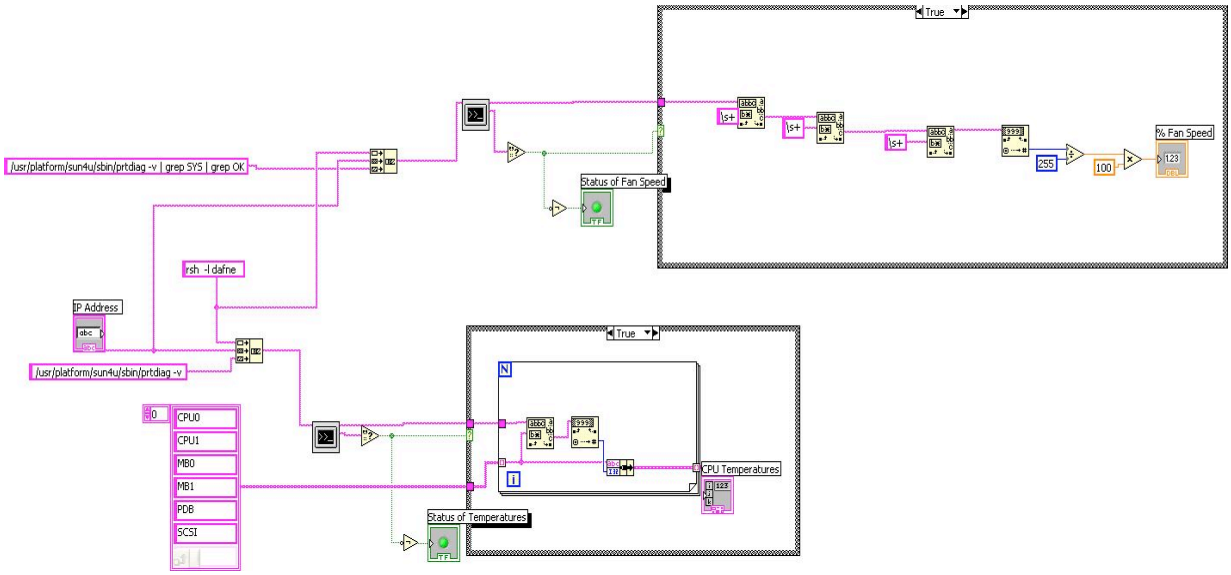
Come si vede dall'immagine del diagramma a blocchi sopra riportato a partire da sinistra si hanno i dati di input che prevedono una 2 stringhe costanti e una che verrà inserita da input. Queste 3 stringhe insieme formeranno una riga di comando che servirà al modulo per prelevare le informazioni relative alla memoria totale e libera di SWAP e calcolarne poi la percentuale relativa.
I dati così calcolati vengono inseriti ,infine, in un array, ovvero l'output del modulo.

MODULO RABBIT QUERY:



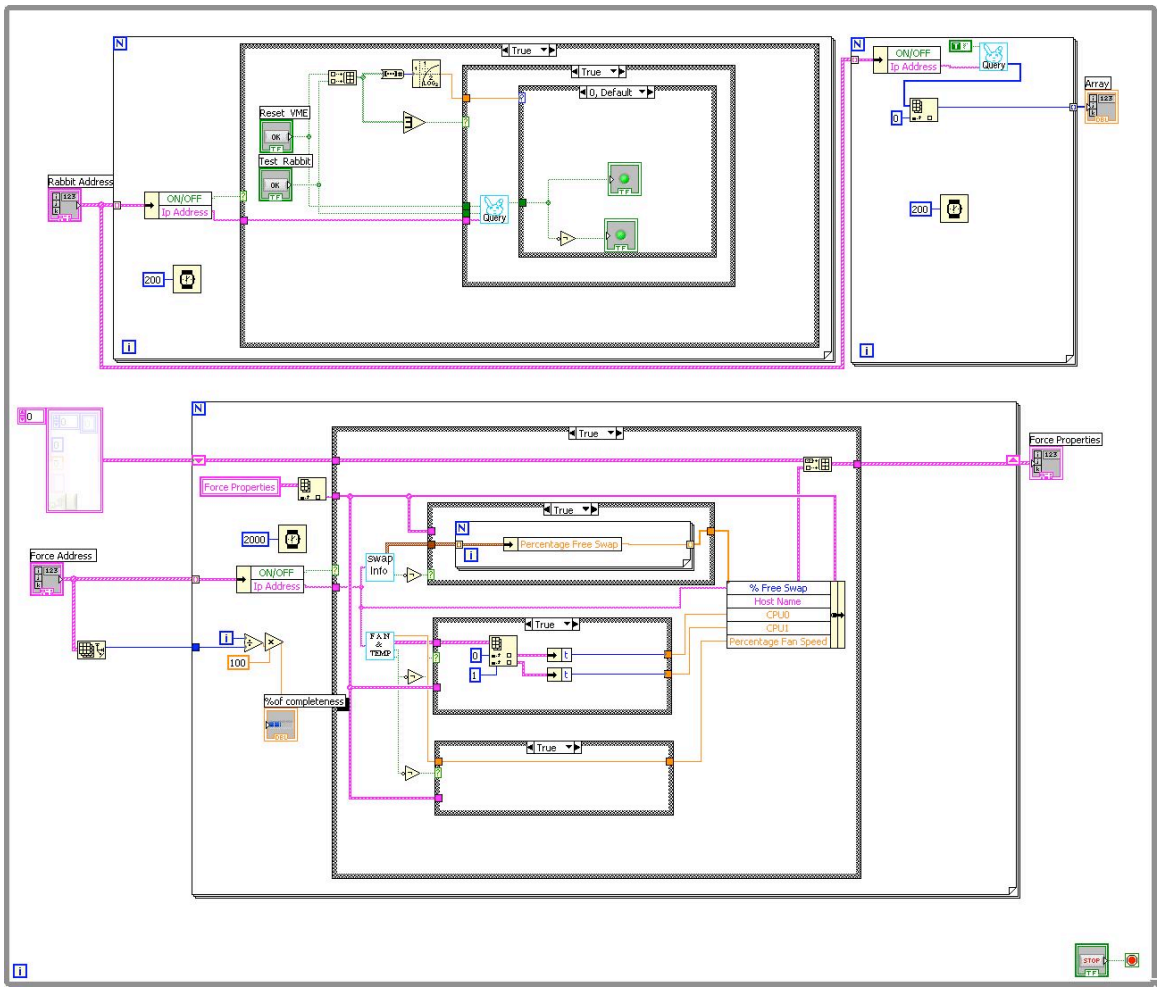
Per questo modulo i dati in input, a partire dalla sinistra dell'immagine, sono l'indirizzo IP inserito dall'utente , la porta di comunicazione come costante e le 3 possibili interrogazioni al Rabbit.
Il modulo prevede che una volta aperta la porta di comunicazione si invia sul protocollo UDP il comando scelto dall'utente; si rimane in attesa della risposta da parte del Rabbit per un periodo di tempo limitato a 10 secondi, se la risposta del Rabbit perviene entro il tempo massimo (TIMEOUT) allora i dati vengono convertiti a 16 bit e scritti in un array e si imposta l'indicatore booleano a "true", altrimenti i dati "muoiono" nella rete e l'indicatore booleano viene impostato a "false".
Sia l'array che l'indicatore booleano sono gli output di questo modulo.

MODULO FAN & TEMPERATURE:



Questo modulo si occupa del calcolo della percentuale dell'utilizzo delle ventole e della temperatura delle CPU,delle schede madri e dei dispositivi SCSI. Nella parte più in alto dell'immagine a partire da sinistra,come input si ha una stringa costante inviata nella console dei comandi che restituisce una riga di testo dove è mostrata la velocità delle ventole che verrà poi calcolata in percentuale (primo dato di output).
Nella parte in basso dell'immagine si hanno nuovi dati di input i quali formeranno la linea di comando inviati alla shell dei comandi che restituirà una stringa nella quale verranno presi i valori delle temperature inseriti poi in un array di cluster (secondo dato di output).

PANNELLO DI CONTROLLO PRINCIPALE:



Il pannello di controllo principale permette all'utente di gestire le richieste d'informazioni mediante la seguente interfaccia grafica:

Tutto il programma è stato implementato in un ciclo infinito che viene arrestato quando l'utente preme il pulsante di "STOP";all'interno del ciclo principale sono stati implementati altri 3 cicli.
La caratteristica principale che permette l'esecuzione dei 3 cicli contemporaneamente è la diversa temporizzazione. I cicli sono cosi implementati: il primo ciclo (in alto a sinistra del diagramma a blocchi) si occupa della gestione delle informazioni inviate e ricevute dal Rabbit; il secondo ciclo (in alto a destra del diagramma a blocchi) permette di inviare continuamente ricezione di dati; infine il terzo ciclo (in basso nel diagramma a blocchi) gestisce e visualizza le informazioni delle schede FORCE grazie allo sviluppo delle funzioni sopra descritte (Swap,Fan&Temperature).

8 – Conclusioni

Alla fine di questo stage possiamo affermare che il nostro progetto ha rispettato tutti i punti previsti in partenza. Purtroppo non è stato possibile espandere il progetto su altre consolle e nemmeno su altri microprocessori Rabbit, per mancanza di tempo e a causa del fatto di aver lavorato su un prototipo disponibile in un solo esemplare.
In questo stage abbiamo potuto apprendere nuove conoscenze che ci saranno molto utili in futuro, e soprattutto in ambito lavorativo.
Non ci aspettavamo di riuscire a concludere il lavoro, perciò la completa riuscita del progetto ci è piaciuta molto, e ci ha gratificato.
Il rammarico è quello che lo stage sia durato poco, avremmo voluto avere la possibilità di esercitarci e cimentarci in altri progetti.

9 - Appendice

Programma sorgente in c

```
#class auto

/***********************************
 * Configuration                   *
 * -------------                   *
 * All fields in this section must *
 * be altered to match your local  *
 * network settings.               *
 ***********************************/

/*
 * Pick the predefined TCP/IP configuration for this sample.  See
 * LIB\TCPIP\TCP_CONFIG.LIB for instructions on how to set the
 * configuration.
 */
#define TCPCONFIG 1
#define PING_WHO                      "192.168.192.1"
#define adc_ck 0
#define adc_data 1
/*
 * Define the number of socket buffers that will be allocated for
 * UDP sockets.  We only need one UDP socket, so one socket buffer
 * will be enough.
 */
#define MAX_UDP_SOCKET_BUFFERS 1

/*
 * UDP demo configuration
 */

/* what local UDP port to use - we receive packets only sent to this port */
#define LOCAL_PORT    40000
//#define REMOTE_PORT 1144
/*
 * If this is set to "0", we will accept a connection from anybody.
 * The first person to connect to us will complete the socket with
 * their IP address and port number, and the local socket will be
 * limited to that host only.
 *
 * If it is set to all "255"s, we will receive all broadcast
 * packets instead.
 */
//#define REMOTE_IP                   "193.206.84.182"
#define REMOTE_IP                     "255.255.255.255" /*broadcast*/

/*******************************
 * End of configuration section *
 *******************************/

#memmap xmem
#use "dcrtcp.lib"

udp_Socket sock;
word   rem_port;
longword       rem_ip;
/*typedef struct {  longword remip;          // Remote host IP address
                                            word   remport;              // Remote host
port number
                  int      len;             // Length of datagram
                  byte     flags;           // Bit mask (defined below)
                  byte     iface;           // Interface number
                }
                    _udp_datagram_info;   */
```

```
int n_byte,m;          // state of virtual switch controlled by button S1
static char x_buf[128];
static char buf_out[20];
static int datax[10];
 _udp_datagram_info udi ;
/* receive one packet (heartbeat) */
int receive_packet(char in_buf[128])
{
int    byte_num,i;

    for(i=0;i<128;i++)
    in_buf[i]='\0';

        /* receive the packet */
   byte_num = sock_bytesready( &sock );
        if (-1 != byte_num)
        {
        udp_recv(&sock, in_buf, byte_num);
        printf("Received-> %s\n",in_buf);
       return byte_num;
                }
   /* no packet read. return */

        return 0;
}
/* send one packet (heartbeat) */
int send_packet        (char buf[128], int length, word     remport,
                                              longword       remip)
{
static long sequence;
  //    auto char      buf[128];
        auto int        retval;

        #GLOBAL_INIT
        {
                sequence = 0;
        }

 /*      fill the packet with interesting data (a sequence number)
        sequence++;
        sprintf(buf, "SEQ=%ld",sequence);*/
 //     length = strlen(buf);

        /* send the packet */
        retval = udp_sendto(&sock, buf, length, remip, remport);
        if (retval < 0) {
                printf("Error sending datagram!  Closing and reopening socket...\n");
                if (sequence == 1) {
                        printf("   (initial ARP request may not have finished)\n");
                }
                sock_close(&sock);
                if(!udp_open(&sock, LOCAL_PORT, remip, remport, NULL)) {
                        printf("udp_open failed!\n");
                        exit(0);
                }
        }
   else
   printf("sending data!!\n");



        //tcp_tick(NULL);
        return 1;
}

//conversion
void  little2big       ( char *data, int nbyte )
{
auto    char    tmp;
auto int i;
        for (i=0 ; i < nbyte ; i = i+2)
   { tmp = data [i];
```

```
     data [i] = data[i+1];
     data [i+1] = tmp;
   }
}

/* answer to receive data */


int answer_data ()
{
         unsigned long int T0;
         n_byte=receive_packet(x_buf);


            if ((x_buf[0]== 'd')&(x_buf[1]== '\0')){
            rem_ip = udi.remip;
            rem_port = udi.remport;
            memmove (&buf_out,&datax,20); //copy to char buffer
             little2big(buf_out,20);
            send_packet(buf_out,20,rem_port, rem_ip);//(x_buf,n_byte);
            return 1;
            }
            if ((x_buf[0]== 'r')&(x_buf[1]== '\0'))
            {
            rem_ip = udi.remip;
            rem_port = udi.remport;
             BitWrPortI(PADR, &PADRShadow, 0, 1);// turn LED on
            T0=MS_TIMER;
            while((MS_TIMER-T0)<400){};
             BitWrPortI(PADR, &PADRShadow, 1, 1);// turn LED off
            send_packet("reset executed",14,rem_port, rem_ip);//(x_buf,n_byte);
            return 2;
            }
            if ((x_buf[0]== 't')&(x_buf[1]== '\0'))
            {
            rem_ip = udi.remip;
            rem_port = udi.remport;
                     BitWrPortI(PADR, &PADRShadow, 0, 0);// turn LED on
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                      BitWrPortI(PADR, &PADRShadow, 1, 0);// turn LED off
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                     BitWrPortI(PADR, &PADRShadow, 0, 1);// turn LED on
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};


                     BitWrPortI(PADR, &PADRShadow, 1, 1);// turn LED off
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                     BitWrPortI(PADR, &PADRShadow, 0, 2);// turn LED on
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                          BitWrPortI(PADR, &PADRShadow, 1, 2);// turn LED off
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                     BitWrPortI(PADR, &PADRShadow, 0, 3);// turn LED on
                     T0=MS_TIMER;
                     while((MS_TIMER-T0)<400){};
                          BitWrPortI(PADR, &PADRShadow, 1, 3);// turn LED off
            send_packet("test ok",7,rem_port, rem_ip);//(x_buf,n_byte);
            return 3;
            }
            return 0;
}

int read_ADC()
{
int j,k,data;
data=0;
for(j=0;j<12;j++)
```

```
{
BitWrPortI(PCDR, &PADRShadow, 0, adc_ck);
for(k=0;k<10;k++)
{};
BitWrPortI(PCDR, &PADRShadow, 1, adc_ck);
for(k=0;k<10;k++)
{};
if( (BitRdPortI(PCDR,3))==1)
{
switch(j){
case 0:
       data=data+2048;
       break;
case 1:
       data=data+1024;
       break;
case 2:
       data=data+512;
       break;
case 3:
       data=data+256;
       break;
case 4:
       data=data+128;
       break;
case 5:
       data=data+64;
       break;
case 6:
       data=data+32;
       break;
case 7:
       data=data+16;
       break;
case 8:
       data=data+8;
       break;
case 9:
       data=data+4;
       break;
case 10:
       data=data+2;
       break;
case 11:
       data=data+1;
       break;
       }//end switch
       }//end if
       }//end for
BitWrPortI(PCDR, &PADRShadow, 0, adc_ck);
datax[0]=data;
}

int read_portB()
{
datax[1]=(RdPortI(PBDR) & 0x3F);
}




void main()
{
   longword seq,ping_who;
   #GLOBAL_INIT
        {
   WrPortI(SPCR, &SPCRShadow, 0x84);// setup parallel port A as output
   WrPortI(PADR, &PADRShadow, 0xff);// turn off all LED's
               memset(x_buf, 0, sizeof(x_buf));
        }
   //memmove (&buf,&data,24); //copy to char buffer
   datax[0]=1;
```

```
   datax[1]=70;
   datax[2]=5;
   datax[3]=16;          //inizializated array
   datax[4]=56;
   datax[5]=12;
   datax[6]=49;
   datax[7]=29;
   datax[8]=19;
   datax[9]=39;

   memmove (&buf_out,&datax,20); //copy to char buffer
   little2big(buf_out,20);

   sock_init();
        printf("Opening UDP socket\n");

        if(!udp_open(&sock, LOCAL_PORT, resolve(REMOTE_IP), 0, NULL))
   {
                printf("udp_open failed!\n");
                exit(0);
        }

        /* receive heartbeats */
   for(m=0; m<100; m++)
        {
        ping_who=resolve(PING_WHO);
       _ping(ping_who,seq++);
       }
       if(_ping(ping_who,seq++)!=0)
                {
                printf("error of connection on the door\n");
       exit (0);
       }
       printf("the door is open\n");
   for(;;)
   {
        read_ADC();
        read_portB();
                tcp_tick(NULL);
                if (1 == udp_peek(&sock, &udi))
        {
             answer_data();
         }

   }
}
```