Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di laurea

Sistema di controllo remoto via web basato sul modulo Rabbit RCM 5700

Relatore: Chiar.mo Prof. Oboe Roberto

Laureando: Miotti Massimo

ANNO ACCADEMICO 2010-2011

Indice

Prefazione

L'intenzione del presente lavoro è quella di descrivere l'attività svolta durante il periodo di tirocinio presso la ditta EAS Elettronica s.p.a. di Schio (VI). Il progetto sviluppato è stato commissionato da un'azienda che produce attrezzature e dispositivi per studi dentistici e laboratori odontotecnici, la Euronda s.p.a., anch'essa operante nel territorio vicentino. Lo scopo era quello di impiegare il modulo RCM 5700 della Rabbit per realizzare un sistema di gestione remota, via web, da applicare alle autoclavi per sterilizzazione prodotte dalla ditta committente.

La tesi inizia con una discussione sulle specifiche di progetto e sugli scopi da raggiungere in relazione ai numerosi vantaggi che si possono ottenere con una gestione remota via web di una macchina. Viene riportata poi una descrizione sommaria del principio di funzionamento delle autoclavi e dei cicli di sterilizzazione tipicamente usati. I capitoli successivi entrano nel merito della progettazione descrivendo l'hardware e il sistema di sviluppo a disposizione, il metodo di interfacciamento con la macchina da gestire, l'interfaccia di rete TCP/IP e alcuni aspetti di sicurezza della pagine web. Un capitolo viene dedicato all'utilizzo delle nuove tecnologie web per l'aggiornamento in tempo reale delle variabili di interesse sulle pagine web. Infine, vengono tratte le conclusioni sul lavoro svolto e vengono esposte alcune considerazioni sugli ulteriori possibili sviluppi della gestione e del monitoraggio via web di macchine.

Capitolo 1

Introduzione

Il rapido sviluppo che l'elettronica digitale ha avuto in questi ultimi anni ha molto cambiato la progettazione, e quindi le modalità di utilizzo e le funzionalità, di ogni tipo di dispositivo impiegato in ambito civile, industriale e medico. Talvolta, lo sviluppo di un'elettronica più sofisticata in un dispositivo ha portato ad una consistente diminuzione della componente meccanica presente nel dispositivo stesso e ad una conseguente diminuzione del costo del materiale e di produzione. Si pensi per esempio all'evoluzione che hanno avuto i riproduttori di musica portatili: dal lettore di audiocassette utilizzato negli anni '80, con limitate funzionalità e con una preponderante presenza di parti meccaniche, al lettore di MP3 dei giorni nostri, con funzionalità molto varie e con una totale assenza di parti meccaniche in movimento.

I passi da gigante compiuti nella progettazione e nella produzione di circuiti integrati hanno permesso di aumentare le potenzialità dei chip mantenendo inalterati costo e quantità di silicio impiegato. Molte periferiche, che per poter essere utilizzate prima richiedevano l'utilizzo di un chip progettato ad hoc, ora vengono inserite nello stesso die del microprocessore che le va ad utilizzare, consentendo un enorme risparmio di spazio nella scheda ed una importante semplificazione nella progettazione del circuito. Questo ha spinto tutti i costruttori di dispositivi dotati di elettronica di controllo ad aggiungere ai propri prodotti caratteristiche accessorie di vario tipo come la connettività di rete, porte di comunicazione seriale, la possibilità di riprodurre suoni o la presenza di schermi a colori di tipo touch.

Anche prodotti di nicchia, come le autoclavi per dentisti, hanno subito una forte evoluzione nel corso degli anni, che le ha rese più funzionali, più efficienti e più pratiche da utilizzare. In questo contesto si vuole poter dotare le autoclavi di una scheda di espansione opzionale che permetta il

collegamento della macchina alla rete ethernet.

I vantaggi che si possono trarre dalla connessione di una macchina alla rete ethernet sono innumerevoli e possono essere riassunti nei seguenti punti:

- possibilità di gestire una o più macchine contemporaneamente attraverso un computer o un qualsiasi dispositivo che si possa connettere ad una rete ethernet e che possa navigare sul web. Le possibilità di gestione sono limitate solamente dalle funzionalità messe a disposizione sulle pagine web e dalla memoria del sistema. Questa funzionalità può essere particolarmente utile per la gestione contemporanea di più macchine sia in fase di normale utilizzo che di collaudo.

- Dato che la gestione della macchina può avvenire utilizzando un semplice browser, è sufficiente utilizzare un dispositivo di limitate prestazioni ed inoltre non è necessario installare alcun applicativo. La macchina è quindi gestibile immediatamente da qualunque postazione di una rete ethernet.

- È possibile proteggere l'accesso alle pagine web, e quindi alla gestione della macchina, tramite un nome utente e una password. Tutti i dati relativi alle operazioni svolte da ciascun operatore possono essere memorizzati per una futura consultazione.

- È possibile prevedere l'esistenza di più gruppi di utenti aventi differenti diritti di accesso alle pagine web. Si può per esempio creare un gruppo di utenti standard che possa eseguire solamente le operazioni di gestione ordinaria e un gruppo di utenti amministratori che possa eseguire anche operazioni di manutenzione.

- Le operazioni di gestione ordinaria e straordinaria potrebbero venire eseguite anche da un utente collegato ad internet dall'altra parte del mondo se la scheda di rete dell'autoclave è resa raggiungibile attraverso internet. Questo potrebbe facilitare enormemente, per esempio, la rilevazione di eventuali guasti a distanza, senza la presenza di un tecnico di fronte alla macchina.

- È possibile sfruttare la connessione di rete per scaricare, anche in maniera automatica, dati della macchina di particolare interesse, in un computer che svolga la funzione di server.

1.1 Obiettivi da raggiungere

Come già anticipato, l'obiettivo di questo lavoro di tesi è stato quello di progettare un dispositivo che permetta la gestione da remoto di autoclavi per dentisti tramite pagine web. La scheda da aggiungere deve essere dotata delle seguenti caratteristiche:

- deve poter gestire l'intera suite di protocolli, dal HTTP all'ethernet, necessari per l'utilizzo di pagine web;
- deve dare la possibilità di scrivere delle funzioni CGI (Common Gate Interface) per poter innescare delle elaborazioni lato server da una pagina web;
- deve dare la possibilità di condividere variabili C con le pagine HTML del server;
- deve memorizzare le pagine web necessarie per la gestione dell'autoclave;
- deve essere in grado di dialogare con l'autoclave mediante lo standard di comunicazione seriale RS-232.

Le capacità di gestione che vengono pretese sono basilari e richiedono principalmente di poter avviare ed interrompere i vari cicli di sterilizzazione raccogliendo dalla macchina e visualizzando in tempo reale un certo numero di informazioni. In particolare le variabili che devono essere acquisite con una frequenza di almeno un campione ogni 2 secondi sono:

- temperatura interna della camera di sterilizzazione;
- temperatura della fascia esterna;
- temperatura del generatore di calore;
- pressione interna della camera.

Poiché il modulo web dovrebbe poter funzionare sia con i nuovi che con i vecchi modelli di autoclave, è richiesto di prevedere la compatibilità del webserver con i due differenti set di comandi utilizzati per controllare le macchine tramite porta seriale. Inoltre, per la progettazione delle pagine web viene richiesto di riprodurre la medesima interfaccia grafica che si può trovare nei nuovi modelli di autoclave.

1.2 Descrizione di un'autoclave

Con il termine autoclave viene generalmente indicato un contenitore nel quale la chiusura ermetica del coperchio viene garantita da una pressione interna al recipiente maggiore rispetto a quella esterna.

In senso più stretto, vengono chiamate autoclavi anche le macchine utilizzate principalmente in ambito medico, come ospedali o studi dentistici, per la sterilizzazione di attrezzature di vario tipo. In questo caso l'autoclave è essenzialmente una caldaia capace di produrre vapore acqueo ad elevata temperatura all'interno di una camera sotto pressione. In commercio esistono autoclavi di varie dimensioni; il volume della camera può variare dai 3 ai 150 litri per le sterilizzatrici da laboratorio e dai 500 ai 1500 litri per quelle ad uso ospedaliero. Un esempio di autoclave per dentisti è riportato nelle figure 1.1a e 1.1b.

(a) Autoclave Euronda E9 Insection

(b) chiusura dell'autoclave

Principio di funzionamento di un'autoclave

La camera di sterilizzazione di un'autoclave è costituita da due involucri a forma di bicchiere posti uno all'interno all'altro. Come si può vedere dalla figura 1.1, nell'intercapedine che si forma tra le due parti della camera circola il vapore acqueo. All'apertura della valvola di esercizio (operating valve) il vapore entra nella camera di sterilizzazione e, essendo più leggero dell'aria, tende ad occupare il volume superiore del contenitore. L'aria presente è quindi costretta ad uscire dal basso attraverso la valvola termostatica che si richiude solamente quando inizia a passare del vapore. La totale espulsione dell'aria è necessaria per l'ottenimento di vapore saturo, cioè in equilibrio con la fase liquida, e quindi per l'ottenimento di vapore

ad elevata temperatura che garantisca la completa sterilizzazione del carico inserito. Nelle autoclavi di nuova generazione il vapore saturo viene prodotto da un generatore esterno e poi iniettato nella camera già svuotata dall'aria tramite una pompa.

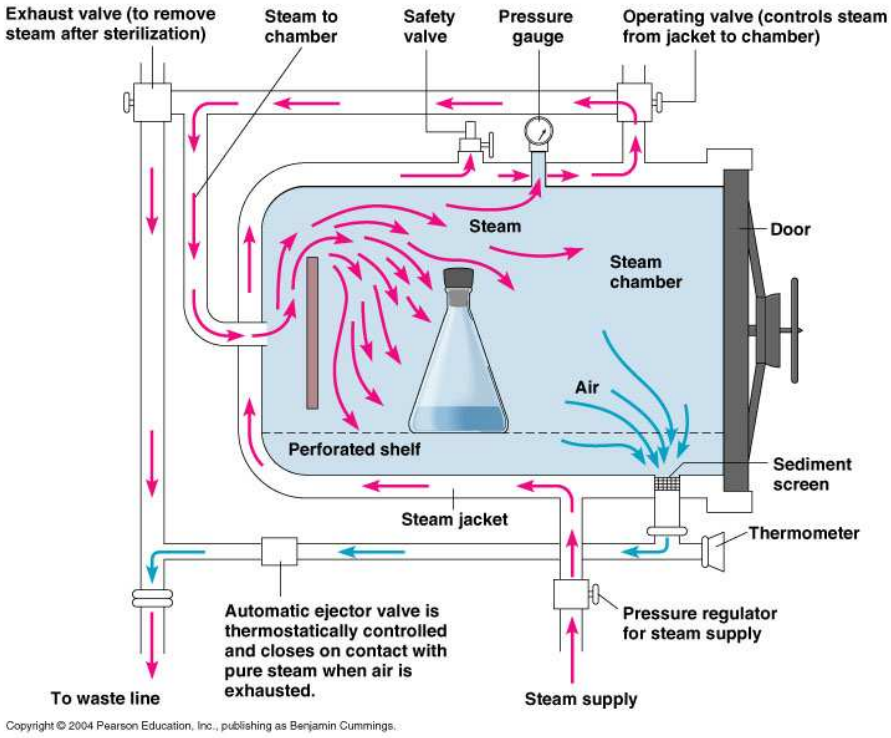


Figura 1.1: principio di funzionamento dell'autoclave

Fasi della sterilizzazione

Le autoclavi utilizzano un ciclo di sterilizzazione standardizzato composto da varie fasi. Nella prima di queste fasi, nella camera di sterilizzazione viene prodotto il vuoto mediante l'utilizzo di una pompa meccanica. Generalmente si fa distinzione tra vuoto meccanico singolo e frazionato. Nel primo caso la pompa aspira in un'unica volta le sacche d'aria presenti nella camera, permettendo così la corretta sterilizzazione di strumenti singoli o imbustati. Nel decondo caso, invece, una pompa a doppia testa effettua ripetute fasi di depressione (solitamente 3) garantendo la perfetta aspirazione delle sacche d'aria presenti nella camera di sterilizzazione e all'interno di oggetti cavi e porosi. La seconda fase del processo di sterilizzazione prevede che la macchina generi vapore facendo aumentare la

pressione della camera fino al valore nominale previsto dal ciclo (per esempio 2.05 bar). Segue poi un periodo di mantenimento della temperatura e della pressione nella camera, per un tempo predeterminato, al fine di ottenere la sterilizzazione. Alla fine di questa fase avviene lo scarico del vapore. Successivamente il carico sterilizzato viene asciugato, in ambiente rigorosamente sottovuoto, eventualmente con l'ausilio di ricircolazione di aria sterile. Quest'ultima fase è estremamente importante dato che gli strumenti umidi sono più facilmente aggredibili dai microrganismi presenti nell'atmosfera.

Esempi di cicli di sterilizzazione

Di seguito vengono riportati alcuni esempi di cicli di sterilizzazione disponibili nelle autoclavi di tipo B prodotte da Euronda.

- **B 134:** la fase di sterilizzazione ha una durata di 4 minuti. La temperatura e la pressione mantenute durante questo periodo sono rispettivamente di 135.5 °C e di 2.16 bar. La fase di asciugatura ha una durata di 15 minuti. Per un'autoclave con camera da 24 litri, il massimo carico solido sterilizzabile è di 6 Kg, mentre il massimo carico poroso è di 2 Kg.

- **B 134 PRION:** come la B 134 solo che la fase di sterilizzazione ha una durata di 18 minuti.

- **B 121:** la fase di sterilizzazione ha una durata di 20 minuti, mentre la temperatura e la pressione di esercizio sono rispettivamente di 122.5 °C e 1.16 bar. La fase di asciugatura ha una durata di 15 minuti. Per un'autoclave con camera da 24 litri, il massimo carico solido sterilizzabile è di 6 Kg, mentre il massimo carico poroso è di 2 Kg.

- **B 134 RAPIDO:** rispetto al ciclo B 134 ha una fase di sterilizzazione della durata di 3 minuti e 30 secondi e una fase di asciugatura di appena 5 minuti. Per un'autoclave con camera da 24 litri, il massimo carico solido sterilizzabile è di 0.6 Kg, mentre il massimo carico poroso è di 0.2 Kg.

- **B 134 PRION RAPIDO:** simile al ciclo B 134 PRION ad eccetto del tempo di asciugatura che è di soli 5 minuti e dei carichi massimi che sono di 0.6 Kg per oggetti solidi e di 0.2 Kg per oggetti porosi.

Nelle autoclavi possono essere a disposizione anche i seguenti cicli speciali di test che permettono di controllare e certificare l'efficacia della sterilizzazione:

- **test di Bowie-Dick:** con questo test viene verificata la capacità di rimozione dell'aria e penetrazione del vapore nei carichi porosi. Si inserisce nella camera un contenitore chiuso al cui interno si trova un campione di carta speciale che cambia di colore se esposta ad una certa pressione di vapore acqueo saturo. La conferma del corretto funzionamento della macchina si ha se alla fine del test la cartina è totalmente e uniformemente virata.
- **Helix test:** permette di verificare la penetrazione del vapore saturo all'interno di piccole cavità. Il campione di test è costituito da un viratore (cartina che cambia colore) posto all'interno di una capsula sigillata, messa in comunicazione con l'esterno attraverso una canula lunga 1.5 m e con diametro del foro di 2 mm.
- **Vacuum test:** valuta la capacità della sterilizzatrice di raggiungere e mantenere il vuoto ad una pressione di 0.86 bar.

Capitolo 2

Il modulo Rabbit RCM 5700

Per lo sviluppo del progetto è stato deciso di impiegare il modulo RCM 5700 prodotto dalla Rabbit semiconductor (vedi figura 2.1). Questo dispositivo non è altro che una scheda dalle dimensioni molto compatte (51mm x 31mm) dotata di tutta la componentistica necessaria per l'implementazione di un piccolo webserver. L'impiego di questa modulo è conveniente in applicazioni nelle quali un semplice microcontrollore risulterebbe insufficiente per via della scarsa disponibilità di memoria, mentre l'utilizzo di un personal computer risulterebbe eccessivo per le potenzialità, ma soprattutto per l'ingombro, il consumo di energia ed il costo.

Le principali caratteristiche messe a disposizione dal modulo sono:

- microcontrollore Rabbit 5000 con clock a 50 MHz;
- una porta ethernet 10/100Base-T;
- una memoria flash da 1 MByte;
- una memoria SRAM da 128 KByte;
- fino a 35 linee di GPIO;
- 6 porte seriali configurabili;
- un orologio Real-Time battery-backed;
- 10 timer a 8 bit;
- 1 timer a 10 bit;
- un watchdog timer;
- 4 canali PWM.

Molte di queste caratteristiche non sono necessarie per lo sviluppo di questo progetto, altre, come la memoria flash e la memoria RAM di discrete dimensioni, sono invece di fondamentale importanza. Anche la disponibilità di porte seriali configurabili è una caratteristica indispensabile dato il bisogno di comunicare con la macchina attraverso l'interfaccia RS-232.

L'RCM 5700 è dotato di un connettore PCI Express sul quale giungono tutte le linee necessarie per l'alimentazione e per l'interfacciamento con l'hardware di un'altra scheda. In questo modo possono essere sfruttate appieno tutte le potenzialità messe a disposizione dal microcontrollore montato sul modulo.

Come si vedrà nei seguenti paragrafi i moduli Rabbit sono venduti corredati di un buon sistema di sviluppo e, soprattutto, di una notevole quantità di librerie utili alla realizzazione di varie applicazioni.

Figura 2.1: modulo Rabbit RCM 5700

2.1 Composizione hardware del modulo

Come risulta evidente dalla figura 2.1 il modulo utilizzato è principalmente composto da tre componenti: il microcontrollore, la memoria flash, ed il chip per l'implementazione della comunicazione 10/100Base-T. Questi tre integrati verranno qui brevemente descritti.

2.1.1 Il microcontrollore R5000

Il Rabbit 5000 è un microcontrollore che presenta ridottissime emissioni di disturbi elettromagnetici, progettato espressamente per l'impiego in sistemi di controllo embedded, in sistemi di comunicazione e nella connettività di rete. A livello di architettura questo microcontrollore eredita molte caratteristiche dello Z80 e dello Z180 dei quali, però, rappresenta un'importante evoluzione. In particolare il Rabbit 5000 ha un bus interno a 16 bit che ha permesso di migliorare le performance di utilizzo di memorie esterne a 16 bit. La frequenza massima di funzionamento è di 100 MHz, mentre la massima quantità di memoria che può indirizzare è di 16 MB. Il core ha una tensione di funzionamento di 1.8 V che permette di limitare i consumi; i pin di I/O, invece, lavorano con una tensione di 3.3 V. Il Rabbit 5000 può vantare numerose periferiche, tra le quali: 8 canali di DMA, 6 porte seriali configurabili e compatibili con le specifiche IrDA, fino a 48 pin di I/O generici, 4 canali PWM, capture e compare, watchdog timer, decoder di posizione a quadratura e clock con batteria. Inoltre, il microcontrollore comprende anche 128 KB di memoria ad elevate prestazioni, utilizzabile sia per istruzioni sia per dati. Infine, per quanto riguarda la connettività, l'R5000 è dotato di un MAC Ethernet 10/100 Base-T con interfaccia standard per il collegamento con il blocco PHY, e di un MAC 802.11 b/g compatibile con vari ricetrasmettitori wireless.

Il Rabbit permette l'esecuzione del boot da memorie seriali esterne, evitando quindi l'utilizzo di una memoria parallela che richiederebbe una notevole occupazione di area per le piste. Importante è anche la disponibilità di 4 livelli di priorità per le interruzioni che permettono una risposta in real-time a vari eventi.

Caratteristiche del microcontrollore

Il Rabbit 5000 è dotato di alcuni particolari accorgimenti che hanno permesso il totale annullamento dei problemi relativi ai disturbi EMI irradiati. L'ampiezza di tali radiazioni è infatti attenuata dalla presenza dello *spectrum spreader*, dallo spegnimento dei clock interni non utilizzati e dalla divisione tra i piani di riferimento del core del processore e delle GPIO (il che riduce il rumore da crosstalk). Lo spreader spectrum, quando abilitato, continua a comprimere e allargare il clock principale in maniera tale da distribuire l'energia del segnale in un ampio range di frequenze. In figura 2.2 è possibile constatare l'effetto dello spectrum spreader nella diminuzione dei picchi delle armoniche del clock.

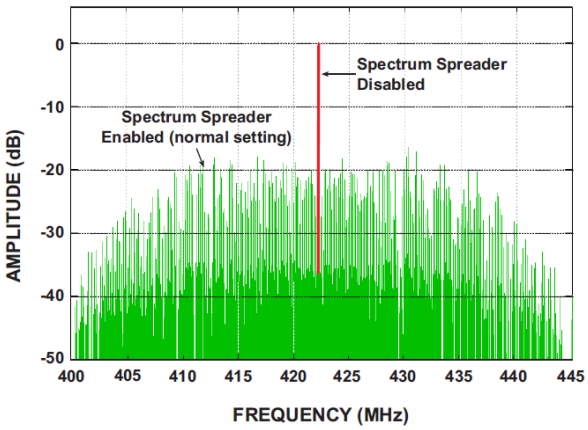


Figura 2.2: effetto dello spectrum spreader nella riduzione dei disturbi

Il set di istruzioni a disposizione permette di ottenere dalla compilazione del codice C prestazioni elevate in termini di efficienza e di velocità di esecuzione. È possibile effettuare operazioni di load e store di dati ed operazioni logiche o aritmetiche a 16 o 32 bit in soli 2 cicli di clock. La moltiplicazione di dati a 16 bit richiede invece 12 colpi di clock.

Il Rabbit 5000, grazie all'indirizzamento a 24 bit, ad un bus dati a 8/16 bit, 3 linee chip select, 2 di output enable e 2 di write enable, può interfacciarsi direttamente con 6 memorie per una totale capacità di storage di 16 MB massimi. Inoltre, la presenza di una memoria SRAM, interna, di 128 KB permette la rapida memorizzazione di dati temporanei a 16 bit.

Il microcontrollore è dotato di 6 porte di I/O parallele a 8 bit, per un totale di 48 GPIO e 6 porte seriali delle quali tutte sono configurabili per comunicazioni asincrone, 4 sono configurabili anche come SPI e 2 sono configurabili anche per lo standard SDLC/HDLC oramai poco utilizzato. Tra le numerose periferiche a disposizione abbiamo 2 ingressi capture temporizzati da un timer interno (che possono essere utilizzati per la misura di periodi), 2 decoder di quadratura (ciascuno dei quali dotato di due ingressi interfacciabili direttamente con encoder ottici e un contatore a 8 o 10 bit), 4 canali PWM a 1024 livelli. I segnali PWM possono essere filtrati per creare un convertitore D/A a 10 bit. Inoltre, sono a disposizione 3 sistemi di timer. Il Timer A consiste in 10 contatori a 8 bit programmabili, 6 dei quali possono essere configurati in cascata. Il Timer B invece comprende un contatore a 10 bit, due match register e due step register. Al raggiungimento del valore contenuto nel match register può essere generata un'interruzione oppure può essere modificato il valore di un pin e il valore di match può essere incrementato del valore memorizzato nello step register.

Il Timer C è invece un contatore a 16 bit programmabile ed utilizzabile per generare segnali PWM o segnali per il decoder di quadratura.

L'R5000 permette di lavorare con sistemi operativi protetti con due livelli di priorità: system e user. Questo consente di eseguire codice per applicazioni critiche senza che venga interrotto da codice utente a bassa priorità. Inoltre, uno spazio di memoria di 4 KB può essere protetto contro la scrittura accidentale ad opera di codice utente.

Otto sono i canali di DMA utilizzabili con periferiche e memorie sia esterne che interne che consentono di ridurre notevolmente il carico di lavoro del processore dovuto al trasferimento di dati da e verso le periferiche collegate.

La connettività di rete ethernet è implementabile grazie alla presenza del MAC Ethernet 10/100 Base-T collegabile ad un canale DMA per facilitare il trasferimento dei dati. La periferica è totalmente compatibile con lo standard 802.3, incluso il supporto per l'autonegoziazione, il link detection, indirizzo multicast ed indirizzo broadcast. L'interfacciamento con il chip che implementa il layer fisico è possibile grazie all'interfaccia MII (Media Indipendent Interface) standard. L'R5000 è dotato anche di un MAC 802.11b/g, anch'esso utilizzabile con il DMA, per la realizzazione di reti wireless ad infrastruttura oppure ad-hoc. L'interfacciamento con i diversi ricetrasmettitori è consentito dalla presenza di due convertitori, A/D e D/A, ad elevata velocità e di una porta seriale sincrona di controllo. Inoltre, è disponibile un convertitore A/D a basse prestazioni per il monitoraggio della potenza di trasmissione. Tutti i convertitori possono essere utilizzati per scopi generici qualora non si faccia uso della connessione wireless.

Schema a blocchi del microcontrollore

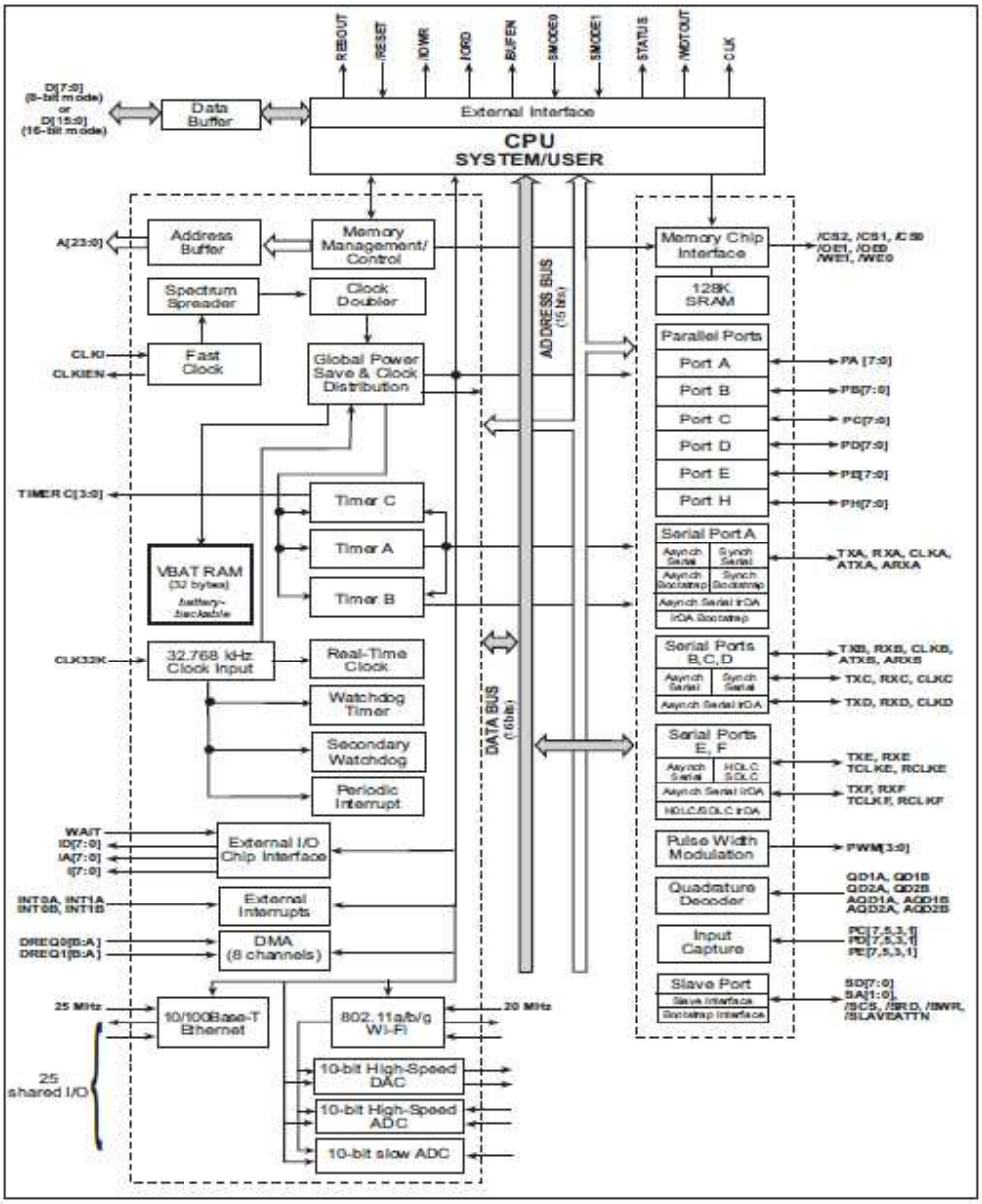


Figura 2.3: schema a blocchi del microcontrollore R5000

2.1.2 Integrato ICS1893

Questo componente implementa il livello fisico di una comunicazione ethernet di tipo 10/100 Base-TX con protocolli di accesso multiplo e rilevazione della portante e delle collisioni e si interfaccia direttamente, attraverso un bus configurabile, al sottolivello MAC del layer di collegamento.

Grazie alla presenza di un digital signal processor (DSP) questo componente può trasmettere e ricevere dati su cavi non schermati (UTP) di categoria 5 con attenuazioni superiori ai 24 dB a 100 MHz, effettuando inoltre una correzione virtuale degli errori generati dai pacchetti killer, ossia pacchetti composti in modo tale da produrre un segnale costante capace di far perdere la sincronia al ricevitore.

L'interfaccia verso il mezzo trasmissivo (Media Dependent Interface) può essere configurata per operare in modalità half-duplex o full-duplex a data rates di 10 o 100 Mbits, oppure per utilizzare la funzione di autonegoziazione al fine di concordare con gli altri utenti della rete una modalità di comunicazione comune. Inoltre, è possibile realizzare sistemi in cui il dispositivo funzioni da semplice nodo della rete, da ripetitore oppure da switch.

Per configurare il modo di funzionamento desiderato per il dispositivo sono disponibili due diversi metodi: quello hardware, che prevede di settare alcuni pin del componente, e quello software, che invece prevede di configurare alcuni registri interni accessibili mediante la porta MII Serial Management. Quando il pin denominato HW/SW è al valore logico 1, allora è selezionata la modalità di configurazione software e, viceversa, quando è al valore logico 0 è selezionata la modalità di configurazione hardware.

Il componente è montato su un contenitore TQFP da 64 pin.

Funzionamento del dispositivo

L'ICS1893 è un componente che processa un flusso di dati bidirezionale tra il livello MAC e il livello fisico di un altro NIC (Network Interface Controller). Quando si devono trasmettere dati, questi vengono passati al componente sotto forma di una sequenza di blocchetti di 4 o 5 bit a seconda del tipo di interfaccia utilizzata. I dati vengono poi serializzati in un unico stream di bit, debitamente codificati e poi trasmessi sul cavo attraverso un trasformatore di isolamento. Durante la ricezione, invece, avviene il processo contrario: lo stream di bit raggiunge il componente attraverso un ulteriore trasformatore, viene decodificato e poi ridotto nuovamente in blocchetti di 4 o 5 bit che vengono inviati al MAC.

Nel funzionamento a 100 Mbits, durante la trasmissione, l'ICS1893 ricostruisce ogni frame MAC dai pacchetti ricevuti e lo incapsula con uno Start-of-Stream Delimiter (SSD) e uno End-of-Stream Delimiter (ESD). In particolare, l'SSD va a sostituire i primi 8 bit del preambolo di ogni frame, mentre l'ESD viene messo in coda per segnalare la fine del frame stesso. Durante la ricezione, invece, l'ICS1893 sostituisce i bit degli SSD con il pezzo di preambolo originale ed elimina l'ESD, di modo che il MAC locale riceva una copia inalterata del frame inviato dal MAC remoto. Durante il periodo in cui non si sta né ricevendo né trasmettendo, il componente segnala e rileva la condizione di IDLE della linea.

Nel funzionamento a 10 Mbits, invece, dato che il preambolo dei frame MAC ricostruiti contiene già una sequenza di start, l'ICS1893, durante la fase di trasmissione, aggiunge solo una delimitatore di fine (IDL) in coda al frame MAC. In ricezione il preambolo viene utilizzato dal componente per ottenere la sincronizzazione del clock e l'IDL viene tolto al fine ottenere una copia precisa del messaggio inviato dal MAC remoto. Durante le fasi in cui il componente non sta ricevendo o trasmettendo, l'integrità del canale di comunicazione è sempre monitorata grazie ad una sequenza di impulsi (Normal Link Pulses) inviati e rilevati sulla linea dai partner della rete.

Oltre all'incapsulamento, i segnali che transitano attraverso l'ICS1893 subiscono numerose elaborazioni necessarie al fine di rendere possibili le varie modalità di comunicazione. Nei prossimi paragrafi verranno illustrati i blocchi principali responsabili di queste elaborazioni e alcune delle interfacce più importanti disponibili all'interno del componente. Uno schema a blocchi dell' ICS1893 è rappresentato in figura 2.4.

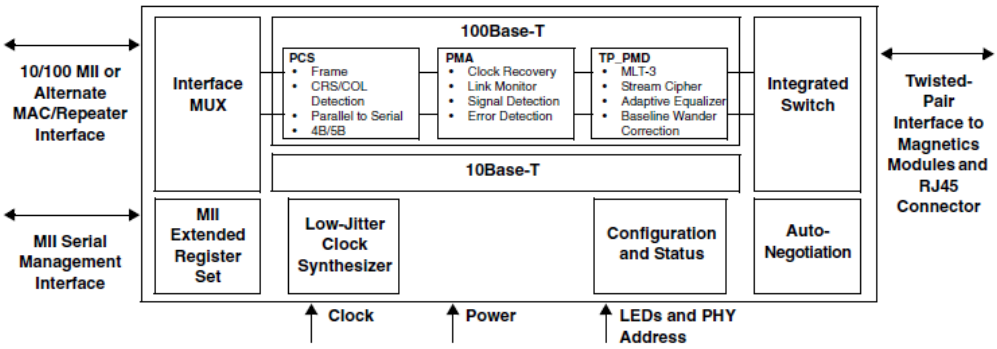


Figura 2.4: diagramma a blocchi dell'ICS1893

Blocchi di elaborazione del segnale

L'ICS1893 implementa al suo interno il livello fisico del modello OSI così come è definito dallo standard ISO/IEC 8802-3, e cioè composto dai seguenti livelli di elaborazione dei segnali:

- Physical Coding Sublayer (PCS);
- Physical Medium Attachment sublayer (PMA);
- Physical Medium Dependent sublayer (PMD);
- Auto-Negotiation sublayer.

Ciascuno dei primi due sublayer è implementato per mezzo di un modulo per la trasmissione e uno per la ricezione, mentre gli altri due sublayer sono realizzati da un unico modulo. Le operazioni che è necessario compiere sui segnali durante il funzionamento a 10 Mbits sono notevolmente più semplici rispetto a quelle necessarie nel funzionamento a 100 Mbits; questo è essenzialmente legato al fatto che per velocità di trasferimento inferiori, la banda occupata è più limitata e di conseguenza è sufficiente una codifica dei segnali più semplice per garantire il funzionamento della comunicazione.

Per non appesantire ulteriormente la trattazione di questo argomento, di seguito verranno illustrate le interfacce di maggior interesse e le operazioni svolte nei vari livelli solo nel caso del funzionamento a 100 Mbits.

Modulo PCS di trasmissione

Questo modulo riceve i blocchi di dati dal MAC e li converte, secondo la codifica 4B/5B, in blocchi di 5 bit i quali prendono il nome di simboli. I dati così ottenuti vengono poi serializzati e passati al sublayer PMA. Il PCS ha anche il compito di incapsulare il frame proveniente dal MAC con lo Start-of-Stream Delimiter (SSD) e l' End-of-Stream Delimiter (ESD). Infine il PCS si occupa anche della rilevazione e segnalazione delle collisioni. Si noti che la codifica 4B/5B, oltre a consentire la risoluzione di problemi relativi alla trasmissione di lunghe sequenze di bit uguali, mette a disposizione 16 simboli, da 5 bit ciascuno, utili per le funzioni di controllo della trasmissione (SSD, ESD, IDLE, etc..).

Modulo PMA di trasmissione

Questo modulo effettua la conversione dello stream di dati ricevuto dal PCS nel formato NRZI (Non Return to Zero Inverted) per poi inviare la

sequenza di bit ottenuta al modulo Twisted-Pair Physical Medium Dependent (TP-PMD). Il PMA inoltre utilizza un PLL digitale per generare il clock di trasmissione (25 MHZ nel caso 100Mbit) a partire da un clock interno di riferimento.

Modulo PCS di ricezione

Il modulo PCS di ricezione acquisisce dal layer PMA il segnale di clock e lo stream di bit oggetto della trasmissione. Quando il link è nello stato di riposo, il modulo riceve solamente simboli di IDLE (sequenza di bit tutti a 1). Nel momento in cui lo stream di bit presenta due zeri non contigui, il PCS analizza i bit che seguono per rilevare un eventuale SSD. Nel caso in cui una trasmissione stia per avere inizio, il modulo sostituisce i bit dello SSD con il primo byte del preambolo standard ed inizia a spezzettare lo stream in blocchetti di 5 bit, i quali vengono sottoposti alla decodifica 4B/5B e poi inviati all'interfaccia MAC. Queste operazioni vengono ripetutamente eseguite fino a che non si verifica una delle seguenti situazioni:

- si riceve un ESD: in questo caso il modulo non trasmette al PCS i bit relativi al delimitatore di fine dello stream e passa allo stato di IDLE;
- si riceve un simbolo non identificato come dato o come segnale di controllo: in questa condizione il PCS mette a 1 il segnale di errore di ricezione (RX ER) e continua a processare il flusso di bit;
- si riceve prematuramente un segnale di IDLE: il modulo segnala la presenza di un errore in maniera analoga al caso precedente e passa allo stato di riposo.

Infine, il modulo PCS di ricezione è anche responsabile della rilevazione della portante (carrier sensing).

Modulo PMA di ricezione

Questo modulo effettua la decodifica NRZI (Non Return to Zero Inverted) dello stream di bit in arrivo dal layer PMD e trasmette il segnale così ottenuto al PCS. Questo modulo è anche responsabile della generazione di un segnale di clock della stessa frequenza di quello utilizzato dal trasmettitore. Per fare ciò il PLL interno al modulo si sincronizza automaticamente con il clock estratto dal segnale ricevuto. Nel caso in cui, per una particolare sequenza di bit, il PLL non sia in grado di agganciarsi al segnale in

ricezione, il PMA attiva un segnale di errore che può essere rilevato dalla lettura di un registro di stato.

Modulo TP-PMD

Il modulo PMD si interfaccia direttamente con il mezzo trasmissivo, che in questo caso è un cavo di rame intrecciato, ed è responsabile di numerose operazioni, sui segnali che transitano, necessarie per poter trasmettere a data rate di 100 Mbits. Le prime due elaborazioni effettuate su questo layer, *stream cipher scrambling/descrambling* e *coding/decoding MLT-3*, hanno essenzialmente lo scopo di limitare le emissioni di disturbi elettromagnetici e la quantità di energia trasmessa nella banda di frequenza 20 - 100 MHz. Il segnale così elaborato può contenere, però, lunghe sequenze di bit uguali le quali causano la fluttuazione della componente DC del segnale. Questo effetto, anche chiamato *baseline wander* riduce l'immunità ai disturbi dato che il ricevitore può misurare una tensione media molto scostata dal valore nominale. Per ovviare a questo problema il modulo PMD realizza un efficace ripristino della componente continua del segnale rendendo la comunicazione molto robusta. Un'altra elaborazione eseguita sul segnale ricevuto è quella che viene chiamata *adaptive equalization* che consiste nel compensare le distorsioni, di fase e di ampiezza, originate dalla trasmissione, ad elevato data rate, su un cavo di rame. Infine il PMD mette a disposizione la funzione di *auto polarity correction* che interviene qualora venga invertita la polarità del segnale in ingresso.

Modulo di autonegoziazione

Il modulo di autonegoziazione, quando attivato, scambia con gli altri utenti della linea informazioni riguardanti le possibili modalità di comunicazione alla scopo di determinare le massime prestazioni raggiungibili da tutti.

Media Indipendent Interface

Abbiamo già detto che la comunicazione tra il livello MAC e l'ICS1893 è realizzata tramite l'interfaccia standard MII, che permette di svincolare dal mezzo trasmissivo utilizzato la progettazione della parte di sistema a monte del layer fisico. Il bus, per consentire una trasmissione di tipo Full-Duplex, utilizza linee diverse per la trasmissione e la ricezione.

I segnali dedicati alla trasmissione sono:

- TXD[3:0]: sono le linee impiegate per trasferire in parallelo blocchetti di 4 bit;

- TXCLK: è il segnale di clock per sincronizzare la trasmissione dei blocchetti;
- TXEN: è il segnale che abilita la trasmissione;
- TXER: è il segnale che avverte il layer fisico dell'esistenza di un problema nella trasmissione dei blocchetti.

Analogamente i segnali dedicati alla ricezione sono:

- RXD[3:0]: sono le linee impiegate per ricevere i blocchetti di 4 bit;
- RXCLK: è il segnale di clock per sincronizzare la ricezione dei blocchetti;
- RXEN: è il segnale che abilita la ricezione;
- RXER: è il segnale con cui il layer fisico avverte il MAC di aver ricevuto una sequenza di bit errata.

Il clock che sincronizza i due interlocutori è sempre fornito da un apposito blocco dell'ICS1893. In aggiunta il componente fornisce al sottolivello MAC anche il segnale di Carrier Sense (CRS) e di Collision Detect (COL).

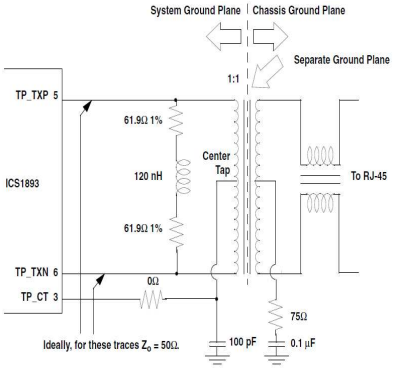
Interfaccia verso il canale trasmissivo

L'ICS1893, come abbiamo più volte sottolineato, è progettato per realizzare connessioni tramite coppie di cavi di rame intrecciati (Twisted Pair). I pin dell'integrato, tuttavia, non possono essere collegati direttamente al mezzo trasmissivo ma tra i due estremi deve essere interposto un sistema di isolamento e di adattamento d'impedenza che tipicamente è incluso nel connettore di rete. In figura sono rappresentati gli schemi dell'interfaccia di trasmissione e di ricezione.

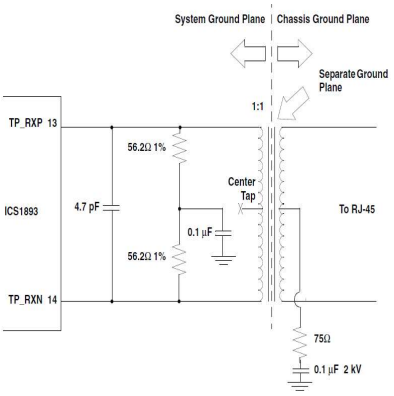
2.1.3 Memoria Flash

Il codice programma, che caratterizza il funzionamento del modulo, è contenuto in una memoria flash di 1 MB montata esternamente al microcontrollore. Il chip utilizzato, un S29AL008D prodotto dalla Spansion, ha caratteristiche che lo rendono particolarmente adatto all'impiego su questi moduli.

Uno dei primi dati di rilevanza che si può ricavare dal manuale del componente è, infatti, che il chip non necessita di un'alimentazione supplementare per consentire le operazioni di scrittura e cancellazione dei dati, permettendo un risparmio di spazio nella scheda del modulo.



(a) Interfaccia di trasmissione



(b) interfaccia di ricezione

I tempi di accesso alla memoria, che raggiungono i 55 ns, permettono al microcontrollore di recuperare il codice delle istruzioni da eseguire, e altri dati, senza avere dei cicli di attesa (wait states) che limiterebbero le prestazioni del sistema.

La memoria può essere organizzata in 1048576 bytes oppure in 524288 word (16 bit) e il bus dati può funzionare sia a 8 bit, sfruttando metà delle linee a disposizione, che a 16 bit. Il microcontrollore che equipaggia il modulo Rabbit è un dispositivo a 8 bit, di conseguenza la memoria è configurata per lavorare a byte. Oltre alle 16 linee del bus dati, come si può vedere in figura 2.5, l'intefaccia è costituita da altre 19 linee di indirizzamento e 6 linee di controllo. Quest'ultime sono:

- il pin CE di abilitazione del chip;
- il pin OE di abilitazione dell'uscita;
- il pin WE di abilitazione della scrittura della memoria;

- il pin BYTE, che permette di impostare in modo hardware il funzionamento a 8 o 16 bit;
- il pin RESET, attivo basso, permette di terminare immediatamente l'operazione in esecuzione e di resettare la macchiana a stati interna mettendola in modalità di lettura della prima locazione di memoria. Questo consente al microcontrollore di leggere il firmware di bootup direttamente da questa memoria in caso di reset del modulo;
- il pin RY/BY che permette di rilevare in modo hardware se un processo di scrittura o cancellazione della memoria è terminato

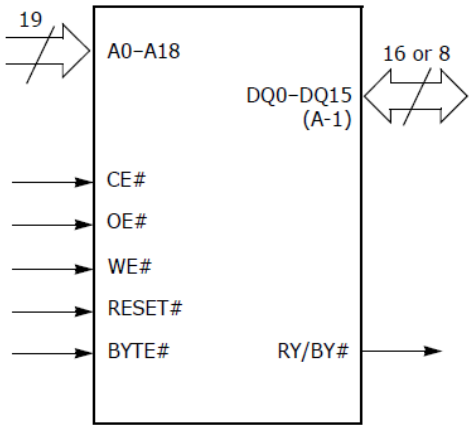


Figura 2.5: rappresentazione schematica della memoria

L'utilizzo dei pin CE, OE e WE permette di evitare la contesa del bus dati in presenza di più memorie interfacciate con lo stesso microcontrollore.

Il set di istruzioni per il controllo della memoria è interamente compatibile con lo standard JEDEC per le memorie flash. I comandi, che tramite il microcontrollore vengono scritti nell'apposito registro della memoria, servono come input per la macchina a stati interna che controlla la circuiteria di lettura, scrittura e cancellazione. Durante queste ultime due operazioni, i valori della porta indirizzi e della porta dati vengono temporaneamente salvati mediante dei latch, mentre le temporizzazioni adeguate vengono garantite dall'esecuzione di algoritmi interni al dispositivo.

La cancellazione dei dati memorizzati può avvenire mediante eliminazione totale dei dati oppure per cancellazione di singoli blocchi.

Il dispositivo presenta alcune caratteristiche atte ad evitare la perdita di dati. In particolare, la circuiteria di controllo inibisce in maniera automatica le operazioni di scrittura durante le transizioni della tensione di

alimentazione ed è inoltre possibile abilitare una protezione hardware che consente solamente la lettura dei dati.

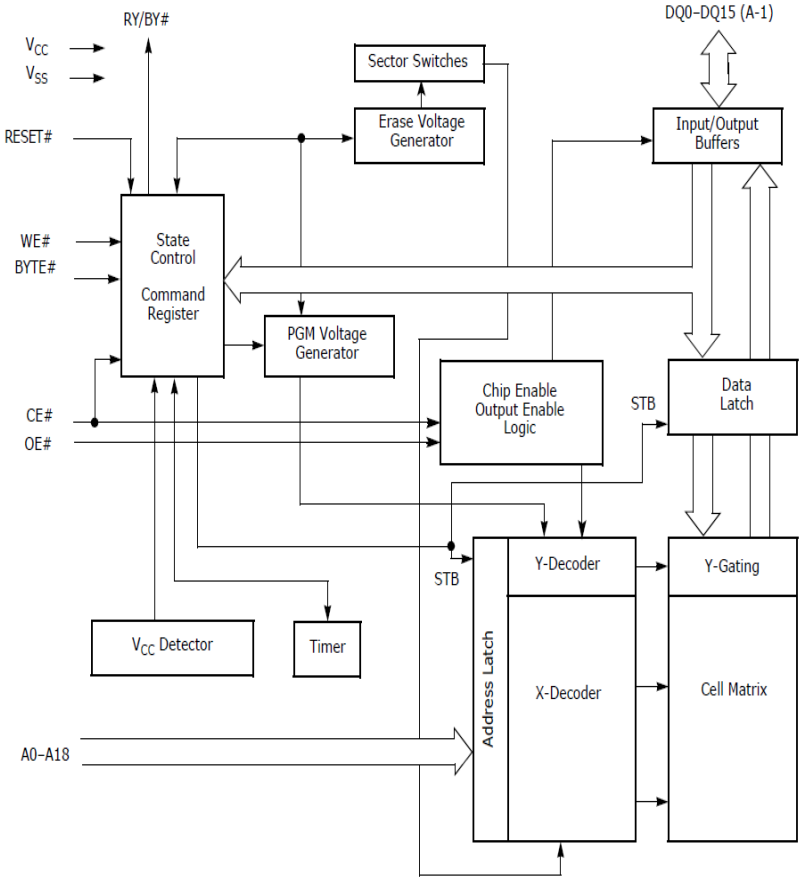


Figura 2.6: schema dei segnali della memoria

2.2 Il sistema di sviluppo Dynamic C

I moduli Rabbit vengono forniti corredati di un sistema di sviluppo integrato molto potente, il Dynamic C, e di una serie di librerie, in codice sorgente, che facilita la realizzazione di molte applicazioni. Questo ambiente di sviluppo, giunto alla versione 10, integra in un unico programma le seguenti funzioni:

- editor ASCII;
- linker;

- compilatore C;
- debugger;
- loader del firmware;
- guida in linea.

Il linguaggio utilizzato per la programmazione del modulo è basato sull'ANSI C, rispetto al quale tuttavia presenta alcune importanti differenze. Prima di tutto il Dynami C, attraverso l'utilizzo di un'apposita libreria, consente l'aggiornamento del firmware del modulo da remoto. Questa funzionalità è molto utile perché attraverso una connessione di rete o l'utilizzo di una SD card è possibile scaricare sulla memoria programma del microcontrollore un software precedentemente compilato in formato .bin. Una seconda funzionalità prevista dal Dynamic C è quella delle *function chain*, le quali facilitano le operazioni di inizializzazione del microcontrollore andando ad eseguire tutti i vari spezzoni di codice che appartengono alla catena. La _GLOBAL_INIT è un caso particolare di function chain che viene eseguito automaticamente all'inizio del programma. Altre caratteristiche importanti di questo "dialetto" del linguaggio C sono quelle che hanno a che fare con l'esecuzione in parallelo di più task del codice programma. Come verrà approfondito nel prossimo paragrafo, mediante l'impiego delle parole chiave *costate*, *cofunction* e *slice* è possibile gestire il multitask, di tipo cooperativo o con prelazione, all'interno di un unico programma.

Per consentire l'ottimizzazione del codice e una migliore gestione delle risorse del microcontrollore è possibile scrivere codice in linguaggio assembly e integrarlo nel programma in C. Sono inoltre a disposizione delle parole chiave, root, xmem, #memmap, che permettono un maggior controllo dell'utilizzo della memoria supportata da parte del programmatore.

D'altro canto il Dynamic C non consente l'indirizzamento in memoria di singoli bit (procedura genericamente indicata come bit fields) e non prevede la possibilità di compilare separatamente parti di un codice programma.

Il Dynamic C mette a disposizione la direttiva #use che consente di includere librerie scritte dall'utente oppure librerie del Dynamic C.

2.2.1 Implementazione del multitasking con il Dynamic C

Per sfruttare al meglio le potenzialità del microcontrollore è possibile utilizzare il multitasking, ossia l'esecuzione in parallelo di più task. È evidente

che un unico microcontrollore può eseguire una sola istruzione al colpo, quindi una vera e propria parallelizzazione non è possibile. Il multitasking di fatto prevede solo che ogni processo esegua parte del suo lavoro mentre gli altri processi sono in attesa di un evento che li faccia ripartire. Se, per esempio una porta seriale sta attendendo di ricevere un flusso di dati, il microcontrollore nel frattempo può occuparsi di svolgere altri compiti.

In generale si possono distinguere due tipi di multitasking: quello di tipo cooperativo (o senza diritto di prelazione) e quello con prelazione. I due tipi di schedulazione hanno un funzionamento notevolmente diverso perché mentre nel primo caso è il task in esecuzione che decide di cedere il controllo del processore, nel secondo caso il passaggio da un task all'altro (cambiamento di contesto) è determinato sulla base delle priorità dei task e dal tempo di esecuzione.

In seguito verranno brevemente illustrati i metodi di implementazione dei due tipi di multitasking tramite il Dynamic C.

Multitasking cooperativo

Il funzionamento del multitasking di tipo cooperativo può essere descritto come l'esecuzione di un ciclo infinito contenente un determinato numero di macchine a stati, ognuna delle quali rappresenta un singolo task. La precedente affermazione risulta molto chiara se si osserva l'immagine in figura 2.7

È facile comprendere che il loop indicato nella figura rappresenta il ciclo infinito di esecuzione del programma, mentre le macchine a stati rappresentano il codice dei singoli task che viene eseguito man mano che gli stati cambiano di valore. Il Dynamic C è dotato di particolari estensioni che permettono di implementare la struttura appena descritta in modo molto facile e pulito. In particolare, non occorre gestire alcun tipo di variabile di stato ma è sufficiente usare delle semplici dichiarazioni. Qui di seguito si può vedere un esempio della struttura del codice C per la gestione di più task in modo cooperativo:

```
while(1){
        costate{ ... } // task 1
        costate{ // task 2
                waitfor( buttonpushed() );
                turnondevice1();
                waitfor( DelaySec(60L) );
                turnondevice2();
                waitfor( DelaySec(60L) );
```

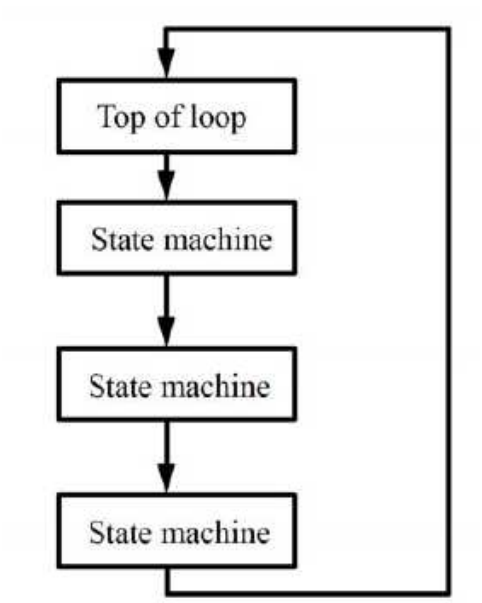


Figura 2.7: loop del multitasking cooperativo

```
                turnoffdevice1();
                turnoffdevice2();
        }
costate{ ... } // task n

}
```

L'istruzione *costate* è proprio quella che permette di costruire una macchina a stati. Come si può vedere il corpo di ciascun costate è racchiuso tra parentesi graffe e altro non è che una lista di istruzioni da eseguire in sequenza. Ciò che invece non è visibile dal programmatore è tutta la gestione degli stati dei vari task. Ogni costate ha un suo puntatore che tiene traccia dell'istruzione che sarà eseguita la volta successiva in cui il processore si occuperà di questo task. Tutte le macchine a stati sono inizializzate quando la function chain _GLOBAL_INIT viene richiamata automaticamente all'inizio del programma.

La sintassi per inserire un costate è la seguente:

```
costate [nome [stato]]
{...istruzioni...
        yield;
        abort;
```

```
        waitfor (espressione);
}
```

Il nome permette di richiamare la struttura dati che sta alla base del funzionamento del costate per una gestione particolare della macchina a stati, ma non è obbligatorio. I valori possibili per il campo `stato` sono invece `always_on` e `init_on`. Nel primo caso il costate viene eseguito in continuazione durante il funzionamento del processore. Nel secondo caso, invece, il costate diventa inattivo, ossia non viene eseguito una volta che è stata completata l'ultima istruzione o che ha subito un aborto. Lo stato di attivazione della macchina a stati può essere controllato mediante le funzioni CoBegin(), CoResume() e CoPause(). Se in un costate con nome non è presente il campo `stato`, allora esso è inizializzato automaticamente nella condizione di `init_on` in pausa e viene eseguito, una sola volta, dopo che è stata chiamata la funzione CoBegin(). Infine, costate dichiarati senza il nome sono automaticamente sempre attivi.

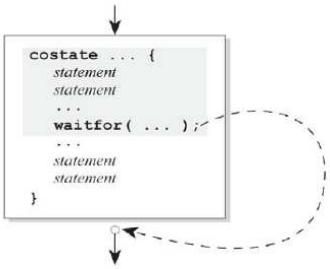
Come si è potuto notare nel frammento di codice sopra riportato, all'interno del costate erano state inserite istruzioni `waitfor`. Esse, insieme a `yeald` e `abort`, permettono di controllare il funzionamento della macchina a stati. Di seguito verranno presentate le caratteristiche di ognuna di queste istruzioni.

waitfor: ogni volta che questa istruzione viene eseguita, il risultato dell'espressione che segue, racchiusa tra parentesi, determina il flusso del programma. Se il risultato è vero (qualunque valore diverso da zero) al prossimo passo sarà eseguita l'istruzione immediatamente successiva al waitfor, se il risultato è falso (zero) il programma salterà direttamente alla chiusura della parentesi graffa del costate. Alla prossima iterazione del task, l'esecuzione riprenderà direttamente dal waitfor. La figura 2.8 dà una rappresentazione schematica del funzionamento di questa istruzione.

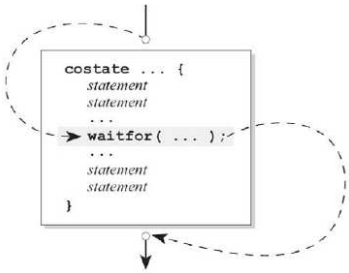
yield: questa istruzione alla prima esecuzione del task opera un salto incondizionato alla fine del costate. L'esecuzione riprenderà poi dall'istruzione successiva a quella di salto.

abort: consente di terminare l'esecuzione di un costate prima di arrivare all'ultima istruzione. Se il costate è di tipo *always_ on* la prossima volta che il programma riprenderà questo task inizierà dalla prima istruzione.

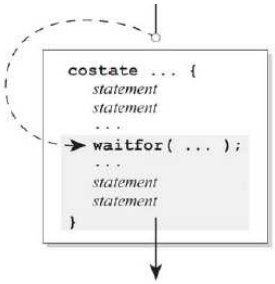
Come già anticipato per poter implementare una macchina a stati il costate utilizza una serie di variabili inglobate in una struttura dati. Que-



(a) Prima esecuzione con espressione falsa

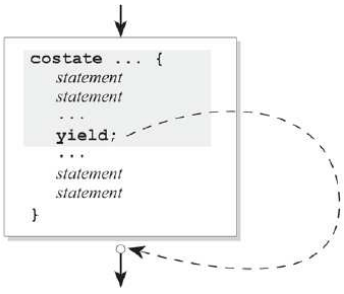


(b) Esecuzioni successive con espressione falsa

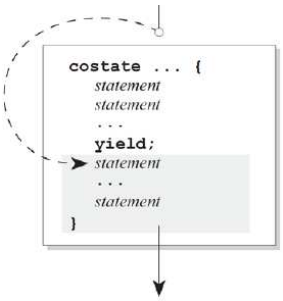


(c) Esecuzioni con espressione vera

Figura 2.8: funzionamento del comando waitfor



(a) Esecuzione di uno yield

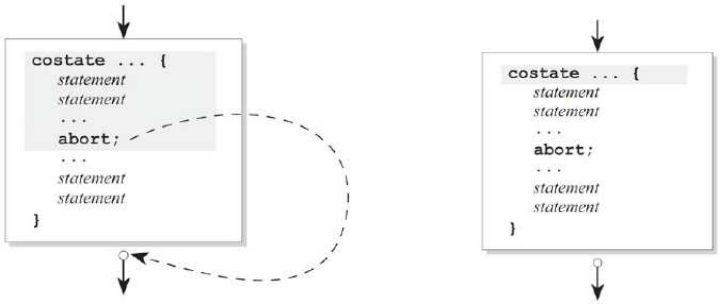


(b) Esecuzione di un task dopo lo yield

Figura 2.9: funzionamento del comando yield

sta struttura è definita come tipo di dato e prende il nome di CoData. Le variabili che la compongono sono:

- CSState: che contiene due flag che indicano lo stato di attivazione e inizializzazione del task.
- Last location: è un dato composto di due campi per un totale di 24 bit contenente l'indirizzo da dove riprendere l'esecuzione del task.



(a) Esecuzione di un abort

(b) Esecuzione di un task dopo l'abort

Figura 2.10: funzionamento del comando abort

- Check Sum: è il check sum dell'indirizzo Last location. Se il check sum non è consistente con l'indirizzo viene generato un errore e il microcontrollore viene resettato.
- First time: è un flag che viene messo a 1 quando un waitfor o waitfordone viene usato per la prima volta. Permette alle funzioni di ritardo di ricavare il tempo trascorso.
- Content: viene utilizzato per memorizzare un eventuale valore di ritardo.

Un altro strumento messo a disposizione dal Dynamic C per la realizzazione di sistemi multitasking sono le *cofunction*. Esse si comportano in modo simile ai costate, ma hanno una struttura simile alle funzioni: infatti, possono ricevere parametri e restituire valori. La dichiarazione di una cofunction è simile a quella di una funzione qualunque, eccetto il fatto che deve essere preceduta dalla parola chiave `cofunc`.

```
cofunc tipo[nome][dim](tipo arg1,..,tipo argN)
{...istruzioni...
                yield;
                abort;
                waitfor (espressione);
}
```

Inoltre, una funzione di questo tipo può essere chiamata solo all'interno di un costate o di un'altra cofunction e solo attraverso la parola chiave `waitfordone` (anche abbreviata con `wfd`). All'interno di queste funzioni possono essere eseguite qualsiasi tipo di istruzioni comprese `abort`, `yield`,

`waitfor` e `wfd`. A differenza della `waitfor`, l'istruzione `wfd` non attende che sia soddisfatta una condizione booleana ma verifica che una o più funzioni siano eseguite completamente, eventualmente ponendo una ulteriore condizione sul valore restituito. Inoltre, per una maggior flessibilità di programmazione anche il `wfd` può restituire valori. Qui di seguito si possono vedere alcuni esempi di utilizzo di questa estensione del Dynamic C.

```
{
        int j,k,x,y,z;
        j = waitfordone x = Cofunc1;
        k = wfd{ y=Cofunc2(...); z=Cofunc3(...); }
        wfd {
                clrscr();
                putat(5,5,"name:");
                putat(5,6,"password:");
                echoon();
        }
}
```

Multitasking con diritto di prelazione

In questo tipo di multitasking i processi non cedono volontariamente il controllo, ma tutto è regolato da un schedulatore che decide quale task eseguire in base ad un livello di priorità e ad un certo limite di tempo. Il Dynamic C consente di realizzare questo tipo di controllo in due modi: attraverso l'istruzione *slice*, metodo che verrà brevemente illustrato qui di seguito, e mediante l'impiego del sistema operativo real-time μC/OS-II.

L'istruzione *slice* consente al programmatore fare interrompere automaticamente l'esecuzione di un blocco di codice dopo che è trascorso un certo tempo. La sintassi generale del comando è:

```
{
slice (context_buffer_size, time_slice)[nome]
{
...istruzioni...
yield;
abort;
waitfor(espressione);
}
}
```

dove:

context_buffer_size è un valore che specifica il numero di byte per il context_buffer (area di memoria per il salvataggio del contesto quando inizia lo slice) e deve essere maggiore delle dimensioni dello stack nel caso peggiore;

time_slice è il numero di tick (1 tick = 1/1024 secondi) per il quale lo slice può rimanere in esecuzione prima di cedere il controllo del processore;

nome è il nome dato alla struttura dati interna allo slice.

Dato che questa applicazione non ha l'esigenza di implementare un controllo in tempo reale, lo slice non è mai stato utilizzato ed è stato descritto solo per completezza di informazione.

2.2.2 Le librerie del Dynamic C

Come già accennato all'inizio del capitolo il sistema di sviluppo è accompagnato da una vasta gamma di librerie che aiutano nell'utilizzo delle periferiche del sistema, nell'elaborazione di dati, nell'utilizzo di memorie di vario genere e nella realizzazione di applicazioni web complesse. Le librerie che in questo lavoro hanno svolto il ruolo più importante sono senza dubbio quelle che implementano lo stack di comunicazione TCP/IP. Esse permettono di realizzare le seguenti applicazioni:

- server HTTP (Hypertext Transfer Protocol);
- client HTTP;
- server FTP (File Transfer Protocol);
- client FTP;
- client SMTP (Simple Mail Transfer Protocol);
- cliente POP3 (Post Office Protocol v3);
- telnet;

L'enorme vantaggio fornito dalle librerie dello stack è quello di rendere il funzionamento delle librerie completamente invisibile al programmatore il quale si può limitare alla configurazione di alcuni parametri tramite la direttiva *#define* e all'utilizzo di un numero limitato di funzioni di interfacciamento. Va detto che è inoltre possibile realizzare connessioni HTTP

sicure mediante l'utilizzo del protocollo SSL (Security Socket Layer) e della crittografia AES (Advanced Encription Standard).

Alcuni aspetti delle librerie TCP/IP verranno maggiormente discussi nei prossimi capitoli.

2.2.3 RabbitWeb

Il RabbitWeb è una particolare estensione del Dynamic C che consente lo sviluppo di un server HTTP evoluto, eliminando, in alcuni casi, l'esigenza di scrivere complesse funzioni CGI (Common Gateway Interface) per effettuare elaborazioni lato server di dati. Esso è composto principalmente da due componenti:

- un semplice linguaggio di scripting, chiamato ZHTML, che va inserito all'interno del codice della pagina HTML;
- una serie di direttive interpretate dal compilatore che vanno inserite nel codice C che implementa il webserver.

L'utilizzo congiunto di questi due elementi consente, per esempio, di aggiornare parametri appartenenti al codice C mediante i form HTML, effettuando eventualmente un controllo automatico sul valore inviato dalla pagina web. Questi parametri, che non sono altro che delle variabili, possono essere eventualmente protette da una login. Ossia, è possibile rendere una variabile modificabile solamente da un utente, o da un gruppo di utenti, che si siano autenticati fornendo al browser web una username e una password. I tipi di autenticazione supportati sono: quella di base, che trasmette le credenziali in chiaro sui pacchetti TCP/IP, e quella DIGEST che elabora la password mediante un algoritmo di crittografia prima di inviarla. Il grosso limite di questo sistema è quello che non permette di creare facilmente pagine web dinamiche, ossia pagine in cui gli elementi cambiano il loro stato in modo automatico senza richiedere un aggiornamento dell'intera pagina. L'utente che sta dietro ad una pagina web, per accorgersi del cambiamento di una variabile del server deve necessariamente richiedere un rinfresco della pagina. Anche il passaggio di valori dal client al server, dato che necessita della pressione del pulsante di *submit* di un form HTML, richiede un aggiornamento della pagina. Questi problemi di dinamicità devono quindi essere risolti senza l'utilizzo delle funzionalità di RabbitWeb ma attraverso l'esecuzione di codice javascript all'interno delle pagine HTML del client e di funzioni CGI sul server.

Un'importante vantaggio derivante dall'utilizzo di RabbitWeb è quello di aver a disposizione un metodo molto facile per indicare al server i MIME

Type da utilizzare per interpretare i file di vari formati e per organizzare in cartelle il contenuto del webserver. Qui di seguito è riportato un esempio di codice che implementa i punti appena descritti:

```
ximport "samples/tcpip/rabbitweb/pages/page2.zhtml"
page2_zhtml
ximport "samples/tcpip/rabbitweb/pages/home.zhtml" home_zhtml
SSPEC_MIMETABLE_START
        SSPEC_MIME_FUNC(".zhtml", "text/html", zhtml_handler),
        SSPEC_MIME(".html", "text/html")
SSPEC_MIMETABLE_END
SSPEC_RESOURCETABLE_START
        SSPEC_RESOURCE_XMEMFILE("/index.zhtml", page2_zhtml),
        SSPEC_RESOURCE_XMEMFILE("/admin/index.zhtml", home_zhtml)
SSPEC_RESOURCETABLE_END
```

Per poter utilizzare il RabbitWeb è necessario aggiungere all'inizio del codice sorgente la direttiva `#define TCPCONFIG 1`.

Capitolo 3

Descrizione dell'interfacciamento con la macchina

L'autoclave è dotata di quattro porte di comunicazione seriale, utilizzabili con lo standard RS232, che le consentono di interfacciarsi con diversi dispositivi master. La macchina si serve di una di queste porte per interfacciarsi con un data logger il quale consente il salvataggio dei dati relativi ai cicli di sterilizzazione e di test svolti. Il protocollo di comunicazione usato si basa sul MODBUS, uno standard largamente impiegato a livello industriale, che permette di connettere più dispositivi ad una stessa rete. In questo caso, essendo la comunicazione di tipo seriale, necessariamente un dispositivo, il datalogger, svolge la funzione di master che può andare ad interrogare le autoclavi le quali svolgono la funzione di dispositivi slave. Abbiamo utilizzato una delle porte seriali dell'autoclave per realizzare la comunicazione con il modulo Rabbit, il quale ha anch'esso a disposizione ben quattro seriali. In questo caso, dato che tale modulo diviene parte integrante della macchina, esso svolge la funzione di master collegato all'unità di controllo che quindi risulta essere l'unico slave della rete.

In questo capitolo verrà descritto il protocollo di comunicazione già implementato nell'autoclave e i passi seguiti per la realizzazione dell'interfaccia master nel modulo Rabbit.

3.1 Protocollo di comunicazione

Nel protocollo MODBUS utilizzato, per la descrizione del valore del byte vengono impiegati i caratteri della tabella ASCII. Per esempio, per trasmettere il valore di un campo che può andare da 0 a 600 servono 10 bit

che equivale a dire 3 cifre esadecimali che vengono rappresentate da 3 byte contenenti il loro codice esadecimale.

3.1.1 Struttura del frame

Il frame utilizzato per la comunicazione seriale è composto da 6 campi:

il campo di start: rappresenta il byte iniziale del frame e permette la sincronizzazione tra master e slave. Quando un dispositivo riceve questo byte sa che sta iniziando una trasmissione di dati che potrebbe essere a lui destinata. Per il master è riservato il valore 0x02, mentre ogni slave, quando intende rispondere, deve impostare come byte di start il valore 0x01 nel caso abbia interpretato correttamente il messaggio del master, oppure il valore 0x15 per segnalare un errore di trasmissione.

Il campo slave: questo campo segue immediatamente il byte di start e rappresenta l'indirizzo identificativo del dispositivo destinatario del messaggio. Secondo lo standard MODBUS, tale campo deve permettere l'indirizzamento di 255 dispositivi. Per fare ciò il protocollo utilizzato da queste macchine per il campo slave riserva due byte, uno per ogni carattere ASCII necessario per rappresentare i 255 slave in codifica esadecimale. I valori attualmente supportati vanno da 1 a 4 più il valore 0 per indicare il master e il valore 255 per messaggi broadcast. Il campo ha lo stesso valore sia nelle trasmissione (da master a slave) che nelle risposte (da slave a master).

Il campo lunghezza: segue immediatamente i due byte relativi allo slave, indica la lunghezza del campo dati che seguirà e il valore rappresentato può variare da 2 a 511. Ha una lunghezza di 4 byte, che permettono di trasmettere i caratteri relativi alle cifre esadecimale che servono per rappresentare il numero di dati.

Il campo dati: contiene il messaggio da trasmettere e ha una lunghezza variabile tra 1 e 255 byte. I dettagli di questo campo verranno maggiormente discussi nel prossimo paragrafo.

Il campo CRC: 4 byte contenenti i caratteri della codifica esadecimale del codice di controllo a ridondanza ciclica il quale permette di verificare la correttezza del messaggio ricevuto. Nel calcolo del CRC sono esclusi il byte di inizio e di fine trasmissione ed il CRC stesso. Qualora il CRC calcolato e quello ricevuto non corrispondano, lo slave

risponde con un NACK. L'algoritmo di generazione del codice verra descritto dettagliatamente nei prossimi paragrafi.

Il campo di Stop: è un byte che indica la fine del frame del messaggio ed ha sempre valore 0x03.

In figura 3.1 vengono riportati i frame che sono coinvolti nella comunicazione tra master e slave.

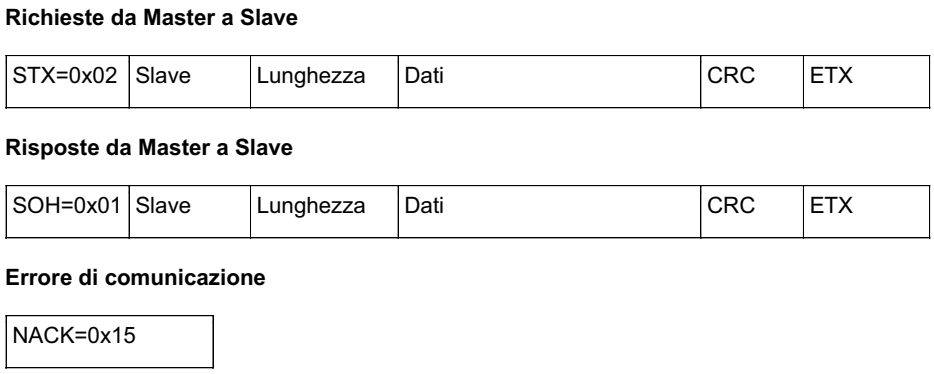


Figura 3.1: struttura dei frame seriali

Non è prevista la gestione di ripetizioni di pacchetti da parte del protocollo, perciò la situazione di errore o mancata risposta deve essere controllata dall'applicativo. Il master prevede un timeout di 2 secondi passati i quali, se non ha ricevuto risposta, ritorna all'applicativo. Per ridurre i tempi in caso di frame ricevuto con forma corretta ma codice CRC errato lo slave invia un NACK, ossia un byte di valore 0x15.

3.1.2 Il campo DATI nel messaggio di protocollo

Questo campo riporta il vero messaggio che deve essere recapitato al destinatario. È a sua volta composto di due sottocampi: il codice del pacchetto, di due byte di lunghezza, indicativo dell'operazione da svolgere, e i parametri del comando che possono avere una lunghezza massima di 253 byte e un formato variabile. Esistono parametri di lunghezza fissa come numeri binari (byte, integer e long), data e ora e parametri a lunghezza variabile come numeri reali con segno e separatore decimale e stringhe in formato UNICODE. I codici pacchetto implementati sono riportati in tabella 3.1 Ognuno di questi codici ha una sua serie di parametri che vanno a completare il campo *dati* e che consentono il totale controllo della macchina.

Codice	Operazione
0x02	lettura/scrittura data autoclave
0x03	lettura/scrittura ora autoclave
0x04	lettura fuso orario
0x06	lettura sensori analogici
0x07	lettura input digitali
0x08	lettura uscite
0x09	setup delle uscite
0x0A	dati ultimo ciclo in memoria
0x0B	cancellazione del ciclo
0x0C	dati ciclo in corso
0x0E	lettura/scrittura dati relativi allo studio che utilizza l'apparecchiatura
0x10	lettura/scrittura dati relativi allo studio che utilizza l'apparecchiatura
0x14	lettura/scrittura dati relativi agli operatori della macchina
0xF1	calibrazione sonde PTC
0xA2	lettura dei contatori ciclo macchina
0xA1	reset della EPROM
0xA3	blocco del ciclo in corso
0xA4	selezione della zona di funzionamento
0xB0	comando avvio ciclo
0xB1	comando interruzione ciclo
0xF0	comando di simulazione tasti della HMI

Tabella 3.1: Valori validi per il sottocampo **Codice pacchetto** del dato

3.1.3 Cyclic Rendundancy Check

Il CRC è un codice che consente la rilevazione di errori casuali in una sequenza di bit ed è tipicamente utilizzato nelle trasmissioni digitali e nei dispositivi di memorizzazione. In una trasmissione seriale il messaggio è costituito da una sequenza di bit, che può avere lunghezza arbitraria, dalla quale viene calcolato un codice di rilevazione degli errori di lunghezza fissata. In generale, un CRC di n bit può rilevare correttamente errori casuali non più lunghi di n bit. Questo tipo di codice viene detto ridondante perché aggiunge una certa quantità di bit che non portano informazioni aggiuntive. Viene inoltre detto ciclico perché l'algoritmo di calcolo è basato sui codici ciclici (ossia codici ottenuti elaborando ciclicamente il dato in ingresso attraverso una rete logica). I CRC, grazie alla loro semplicità di implementazione su hardware e alla loro efficacia nella rilevazione di errori nelle linee di comunicazione interessate da elevato rumore, sono largamente impiegati.

Per specificare un codice CRC utilizzato per la rilevazione degli errori è necessario fornire quello che viene chiamato polinomio generatore. In questo caso, invece, il costruttore della macchina ha fornito direttamente l'algoritmo che permette di implementare il calcolo via software del codice con lunghezza fissa di 2 byte.

Algoritmo di calcolo del CRC

La regola prevede di ricevere in ingresso tutti i byte dei quali bisogna calcolare il codice e di processarli uno ad uno partendo da quello che viene trasmesso per primo. Il calcolo parte con un'operazione di xor tra il numero 0xFFFF e il primo byte allineato a destra per ottenere il primo risultato parziale. Il prossimo valore intermedio sarà ottenuto da un semplice shift logico a destra con inserimento di uno zero nel bit più significativo, nel caso il primo bit del risultato precedente sia uno 0, e dallo shift più un'operazione di xor con un numero prestabilito (0xA001 in questo caso) nella situazione opposta. Questa procedura tra risultati intermedi, iterata per otto volte, consente di ottenere il CRC valido per il primo byte di dati. Al fine di calcolare il CRC finale è necessario effettuare le operazioni appena elencate per tutti i byte che compongono il messaggio, utilizzando però come valore per per la prima operazione di xor il risultato intermedio ottenuto dal byte precedente anziché il numero 0xFFFF. Si noti che il valore ottenuto alla fine di tutte queste operazioni presenta i due byte scritti in ordine inverso rispetto a quello corretto.

3.2 Implementazione del protocollo di comunicazione sul modulo

IL modulo Rabbit è dotato di 4 porte seriali che possono funzionare sia in modo sincrono che in modo asincrono. In questo caso la connessione tra il webserver e la macchina deve essere conforme allo standard RS-232 che è un'interfaccia di tipo asincrono. Inoltre la velocità di trasferimento dei dati deve essere impostata a 19200 baud.

3.2.1 Configurazione delle seriale

Per la realizzazione di questo progetto si è deciso di utilizzare la libreria RS232.lib inclusa nel set di librerie C fornite assieme al modulo. Essa mette a disposizione una serie di strumenti che consentono un utilizzo semplice ed efficace delle porte seriali in modalità asincrona. La libreria è essenzialmente composta da due buffer circolari per ognuna delle porte, le quali vengono distinte tramite le lettere A,B,C e D, di una interrupt service routine (ISR) e di una serie di funzioni di interfacciamento. I due buffer hanno il compito di immagazzinare temporaneamente i dati in attesa di trasmissione e quelli già ricevuti ma non ancora processati dall'applicativo. Le funzioni della libreria permettono di utilizzare la seriale semplicemente andando ad inserire e ad estrarre valori in questi due buffer senza doversi occupare della gestione degli interrupt di ricezione e di verificare l'effettivo invio di ogni byte. La dimensione di questi array può essere impostata andando a definire, tramite la direttiva *#define*, le costanti `XINBUFSIZE` e `XOUTBUFSIZE`, dove al posto della lettera X va messa la lettera indicativa della porta seriale. Questi valori non sono particolarmente critici se si garantisce un'efficace gestione della seriale. Quando la porta sta ricevendo dati, questi devono essere processati prima che il buffer sia completamente riempito dalla ISR della seriale, pena la perdita di dati. Per quanto riguarda la trasmissione invece, è evidente che se il buffer di uscita è pieno l'applicativo non può inserire altri dati da spedire. Questa situazione deve quindi essere gestita in modo intelligente al fine di evitare che il processore rimanga troppo tempo in attesa di poter inserire dati nel buffer. Dato che i buffer devono necessariamente avere dimensioni date dalla legge $2^n - 1$ e che 255 byte sono insufficienti per contenere un'intero messaggio del protocollo delle dimensioni massime, si è adottato per entrambi il valore 511. Oltre a definire le dimensioni delle FIFO (che altrimenti sarebbero impostate di default a 31 byte), per utilizzare una porta seriale è anche necessario configurare i vari parametri della comunicazione. Anche in questo, siamo

facilitati dall'utilizzo delle funzioni di libreria. All'inizio dell'esecuzione del programma la seriale viene inizializzata tramite le seguenti funzioni:

- `serXdatabit(PARAM_8BIT)` che configura il numero di bit utilizzati dalla porta;
- `serXparity(PARAM_NOPARITY)` che disabilita il bit di parità;
- `serXopen(19200)` che abilita la porta a comunicare alla velocità di 19200 bps.

3.2.2 Funzioni di interfacciamento con la seriale

Le funzioni che consentono l'interfacciamento con i buffer delle porte seriali sono elencate qui di seguito:

- serXgetc() legge il prossimo carattere nel buffer di ricezione;
- serXread() legge uno specifico numero di byte dal buffer di ricezione;
- serXpeek() guarda il valore del prossimo byte memorizzato nel buffer di ricezione;
- serXputc() scrive un carattere nel buffer di trasmissione;
- serXputs() scrive una stringa (array di caratteri con terminatore) nel buffer di trasmissione;
- serXwrite() scrive uno specifico numero di byte sul buffer di ricezione;

Per la corretta comprensione delle parti di codice riportate in questo capitolo e in appendice è importante evidenziare i seguenti fatti:

- serXpeek(), a differenza di serXread() e serXgetc(), quando viene richiamata va a vedere il valore del prossimo dato sul buffer della seriale senza prelevarlo definitivamente;
- serXputs() serXwrite() sono funzioni bloccanti, ossia non ritornano finché non hanno inserito sul buffer di uscita tutti i dati che devono inviare;
- dopo che i dati da inviare sono stati inseriti nel buffer di uscita è la ISR della seriale ad occuparsi dell'effettivo invio dei byte;

- tutte le funzioni, ad eccetto di serXpeek(), bloccano il buffer all'inizio della loro esecuzione e lo sbloccano quando ritornano. In conseguenza di ciò e del fatto che i buffer sono variabili globali, un unico task alla volta può operare su una determinata porta seriale.
- esistono anche le versioni cooperative di queste funzioni che migliorano l'utilizzo di tempi morti. In particolare le funzioni di trasmissione passa ad un altro task se il buffer di uscita è pieno e non può completare l'inserimento dei dati.

Altre funzioni importanti che sono state impiegate nella progettazione del controllo delle porte seriali sono serXrdFlush() e serXwrFlush()che permettono di svuotare brutalmente il buffer di uscita e di ingresso di una porta.

3.2.3 Gestione cooperativa della comunicazione seriale

Per una gestione quanto più robusta possibile della comunicazione seriale, si è scelto di attendere sempre la ricezione della risposta dopo l'invio di un messaggio verso la macchina. In questo modo è possibile verificare ad ogni trasmissione la funzionalità del canale di comunicazione ed eventualmente segnalare un'anomalia attraverso l'accensione di un led nella scheda oppure attraverso un messaggio visualizzato nella pagina web. Dato che in questo progetto, specialmente durante i cicli di sterilizzazione, si fa uso intensivo della comunicazione seriale, è stato di fondamentale importanza gestire tutti i tempi morti che la trasmissione e la ricezione possono introdurre a causa di anomalie nella comunicazione. Durante questi istanti, in cui non c'è attività nella porta seriale, è importante che il processore sia libero di eseguire le altre funzioni implementate nel server. Per mettere in atto quanto appena detto, nello sviluppo del software si è fatto largo impiego delle varie estensioni del Dynamic C che permettono di realizzare un sistema multitasking.

La variabile che svolge il ruolo di "semaforo" per le trasmissione dal modulo alla macchina è `busy_serial`. Questo flag ha valore 1 quando è stato inviato un messaggio ma non è ancora stata ricevuta la risposta (in questo caso non può avvenire una ulteriore trasmissione) e viene azzerato quando si riceve una risposta di qualsiasi tipo o quando scade un timeout di 2 secondi. In linea generale, una comunicazione tra master e slave si sviluppa nei seguenti passi:

1. viene inviato un messaggio attraverso la funzione

```
cofunc char
send_command_with_serial
(int port,char *ptr_com, int lenght)
{
    waitfor((busy_serial==0)||DelayMs(100));
    if (busy_serial==1){
       #ifdef DEBUG_MODE
       printf("Seriale occupata o cavo
         seriale scollegato\n");
       #endif
       return -1;
    }
    busy_serial=1;
    serXwrite (port,ptr_com,lenght);
    return 1;
}
```

I parametri che devono essere passati a questa funzione sono: la porta seriale dalla quale trasmettere, un puntatore alla struttura che contiene i dati da inviare e il numero di byte da inviare. Come si può comprendere dal codice riportato, grazie all'istruzione `waitfor` l'esecuzione della funzione viene sospesa qualora la seriale sia occupata o sia scaduto il timeout di 100 ms. In questa situazione, il processore si occupa dell'esecuzione di altre istruzioni. Non appena il flag `busy_serial` viene messo a 0 dopo aver ricevuto una risposta, l'esecuzione della funzione riprende dall'istruzione successiva al `waitfor` ed effettua l'invio dei lenght byte che trova a partire dall'indirizzo puntato da ptr_com. La dichiarazione della funzione appena descritta inizia con la parola chiave *cofunc* ad indicare l'adozione di un comportamento di tipo cooperativo. Questa funzione deve quindi essere richiamata all'interno di un *costate* o di un'altra cofunction attraverso l'istruzione *wfd*.

2. lettura in polling del buffer della seriale mediante la funzione `int leggi_frame (int port, char *ptr_command)`. I parametri da passare sono: la porta sulla quale andare a leggere e il puntatore alla struttura dati adibita a conservare il frame ricevuto privato dei byte di start e di stop. Tale funzione, dopo aver verificato la presenza di dati sul buffer di ingresso, controlla se il primo di essi è uno Start Of Heading (SOH) o un Non Acknowledge (NACK). Nel primo caso inizia il travaso dei byte successivi, fino all'End of Text (ETX), dal

buffer alle locazioni di memoria indirizzate dal puntatore. Nel secondo caso, invece, la funzione riconosce che la richiesta precedentemente inviata non è stata interpretata correttamente dalla macchina e non gestisce ulteriormente la ricezione. Al termine dell'esecuzione di tutte le istruzioni, viene restituito il valore 1 se è stato riconosciuto un frame corretto, -1 se è stato ricevuto un NACK e 0 se il buffer è stato trovato vuoto o se dai byte ricevuti non è stato individuato un frame corretto.

Data la lunghezza della funzione, il codice della stessa sarà riportato in appendice.

3. interpretazione del frame ricevuto mediante la funzione `void interp_frame (char *ptr_comm, frame_t *frame)`. I parametri da passare sono: il puntatore alla struttura dati che contiene il messaggio ricevuto privato dei byte di start e di stop e il puntatore alla struttura di nome frame_t dove inserire i valori dei vari campi in un formato opportuno. Tale struttura, riportata nel codice sottostante, è stata creata appositamente per facilitare l'elaborazione delle informazioni contenute nel frame.

```
typedef struct frame{
        char slave;
        int dimension;
        char code[3];
        char data[255];
        char crc[5];
        char flag_CRC_OK;
}frame_t;
```

Come si può osservare, la struttura di `frame_t` rispecchia quella dei pacchetti del protocollo descritto nel paragrafo 3.1. La prima variabile riporta il numero dello slave dal quale è stato ricevuto il messaggio. Il suo valore, in questo caso, sarà sempre 0 dato che la seriale del microcontrollore comunica con un'unica macchina. La seconda variabile riporta invece un numero relativo al campo *dimensione* del pacchetto seriale. La variabile viene rappresentata da un numero intero, risultando ciò conveniente per facilitare l'estrazione del campo *dati* dallo stream di byte. Il terzo dato appartenente alla struttura contiene in formato stringa il codice del pacchetto che può assumere uno dei valori riportati in tabella 3.1. Questo valore verrà poi utilizzato per identificare l'operazione svolta. Segue poi la variabile

`char data[255]` che contiene i byte relativi al messaggio ricevuto. Le ultime due variabili della struttura riguardano invece il codice di rilevazione degli errori: `char crc[5]` contiene in formato stringa il CRC ricevuto con il messaggio, mentre `char flag_CRC_OK` è una variabile che assume il valore 1 se il codice CRC ricevuto corrisponde a quello calcolato e 0 in caso contrario. Per il calcolo del CRC si impiega la funzione `void calcola_CRC (char *ptr_CRC, char *ptr_data, char lunghezza)` che implementa l'algoritmo descritto nei paragrafi precedenti. I parametri che devono essere passati a questa funzione sono: il puntatore alla regione di memoria dove andare a scrivere il CRC calcolato, il puntatore al primo byte di dati dei quali bisogna calcolare il codice ed infine il numero di byte da considerare nel calcolo.

La funzione per il calcolo del CRC è riportata in appendice.

4. Elaborazione del frame mediante la funzione `elabora_frame(frame_t *frame)`. L'unico parametro da passare a questa funzione è il puntatore alla struttura dati di tipo frame_t che contiene le informazioni relative al pacchetto ricevuto. In base al contenuto di questa struttura la funzione esegue istruzioni diverse al fine di estrarre in modo corretto le informazioni provenienti dalla macchina. Di seguito è riportato il codice eseguito nel caso la variabile `code` contenuta in un frame sia 06, ossia lettura sensore.

```
if (!strcmp((newframe->code),"06"))
{ //06 è il codice di lettura di un sensore
        #ifdef DEBUG_MODE
  printf("Misurazione di un valore\n");
  #endif
  for (i=0;i<4;i++)
  {
                sensor[i]=newframe->data[i];
  }
  sensor[4]=0;
  #ifdef DEBUG_MODE
  printf("Sensore N %s\n", sensor);
  #endif
  strcat(svalue,&(newframe->data[5]));
  strcat(svalue,&(newframe->data[7]));
  strcat(svalue,&(newframe->data[9]));
```

```
strcat(svalue,&(newframe->data[11]));
strcat(svalue,&(newframe->data[13]));
if(strcmp(&(newframe->data[7]),"")!=0)
{
                strcat(svalue,&(newframe->data[15]));
}
svalue[6]=0;
#ifdef DEBUG_MODE
printf("\n Valore misurato"\
"dal sensore %s = %s\n", sensor, svalue);
#endif
if (!strcmp(sensor,"0000"))
{
      strcpy(ptc1,svalue);
  #ifdef DEBUG_MODE
   printf("\n Valore misurato" \
   "dalla ptc1 = %s\n", svalue);
  #endif
}
if (!strcmp(sensor,"0001"))
{
      strcpy(ptc2,svalue);
  #ifdef DEBUG_MODE
   printf("\n Valore misurato"
   "dalla ptc2 = %s\n", svalue);
  #endif
}
```

L'invio di pacchetti sulla porta seriale è quasi sempre scatenato dall'esigenza di aggiornare qualche valore visualizzato sul web. Dato che non è possibile prevedere quali pagine del server sono aperte nei browser degli utenti, le richieste di aggiornamento delle variabili arrivano in modo totalmente asincrono. Tuttavia le funzioni CGI che rispondono a tali richieste non possono gestire in modo cooperativo la trasmissione dei pacchetti e l'attesa della risposta della macchina; infatti, le cofunction possono essere richiamate solamente all'interno di un costate che viene ciclicamente eseguito all'interno del main. Per consentire di recuperare dalla macchina, in modo cooperativo, solo i valori che effettivamente servono per tenere aggiornate le pagine visualizzate, si è scelto di implementare nel server dei flag che identificano, per classi, le variabili che di volta in volta devono essere aggiornate. Per esempio, durante il ciclo di sterilizzazione,

si devono acquisire dall'autoclave il valore di 4 sensori di temperatura e di un sensore di pressione. Quando inizia un ciclo, viene abilitato il flag `acquisizione_ciclo` che attiva, all'interno di un costate, l'esecuzione del codice responsabile dell'invio dei pacchetti seriali per l'aggiornamento dei cinque valori. A questo punto i valori possono essere acquisiti dalla macchina periodicamente ed eventualmente immagazzinati in una FIFO in attesa di essere trasmessi ad una pagina web che li richieda. Così facendo non vi è il pericolo che a causa del malfunzionamento della comunicazione tra master e slave il programma rimanga fermo all'interno di una funzione CGI che sta attendendo un messaggio di risposta.

Questo meccanismo funziona anche se si vogliono inviare dei comandi alla macchina. Volendo progettare un funzionamento robusto dell'applicazione è sempre conveniente verificare che un comando inviato alla macchina sia stato ricevuto correttamente. Per fare questo è necessario attendere la risposta dello slave che in questi casi, se il pacchetto è stato interpretato correttamente, restituisce una sequenza di byte esattamente uguale a quella ricevuta, ad eccetto del byte di start. Una tecnica che è possibile utilizzare per implementare questo comportamento è simile a quella già descritta nel caso dell'aggiornamento delle variabili. Se per esempio si vuole avviare un ciclo da una pagina web, è possibile, tramite la chiamata di una funzione CGI, andare ad attivare un flag. Questa variabile, a sua volta, causa l'esecuzione, all'interno di un costate, della cofunction responsabile di inviare alla macchina il pacchetto di inizio ciclo e di attendere ed interpretare la risposta. Se il messaggio è stato ricevuto correttamente e la macchina ha quindi dato inizio al ciclo, il webserver metterà a 1 una variabile ad indicare che la sterilizzazione è iniziata. La pagina web, con un piccolo ritardo programmabile può andare a verificare tramite un'altra funzione CGI che il comando abbia avuto effetto.

Capitolo 4

Descrizione dell'interfaccia di rete

Come descritto nel capitolo di introduzione, in questo progetto si è voluto realizzare un controllo via web di una macchina collegata alla rete ethernet. L'applicazione web, in questo caso il server HTTP, è situato al livello più alto di una pila di protocolli che permette di instaurare la comunicazione con altri sistemi client. Le librerie del Dynamic C permettono di implementare sul modulo la maggior parte dei protocolli di interesse per questo tipo di applicazioni e mettono a disposizione del programmatore una serie di funzioni e di procedure che consentono di utilizzarli senza conoscere i minimi dettagli dei protocolli stessi. In questo capitolo si cercherà di dare una descrizione sommaria dei vari livelli dello stack TCP/IP e di come sono stati utilizzati i mezzi messi a disposizione dalle librerie per implementarli.

4.1 Il modello di riferimento ISO/OSI

La comunicazione tra dispositivi appartenenti ad una rete passa attraverso l'utilizzo di una serie di protocolli. Per dare una definizione di questo termine potremmo dire che un protocollo è una descrizione formale della struttura dei messaggi e delle regole che due o più macchine devono rispettare per poter scambiare quei messaggi. L'implementazione di un protocollo può essere realizzata via software, via hardware o come combinazione dei due. A seconda del tipo di comunicazione che si vuole instaurare, si devono utilizzare protocolli diversi.

L'ISO (Intenational Organization for Standardization) per favorire un'approccio comune e l'interoperabilità tra apparecchiature diverse ha sviluppato un modello di riferimento, l'OSI (Open System Interconnection Re-

ference Model), per la realizzazione di sistemi di comunicazione. Questo modello descrive un'architettura di rete suddivisa in 7 strati, ognuno dei quali, per mezzo dei protocolli, svolge nella comunicazione una particolare ruolo. Ogni layer si interfaccia con il sottostante, del quale utilizza le funzioni, e con il soprastante, al quale invece fornisce le proprie funzioni.

Per comprendere appieno i differenti livelli del modello proposto dall'ISO bisogna prima analizzare quali sono i passi che si susseguono in una comunicazione tra due dispositivi. Qui di seguito abbiamo una descrizione di una comunicazione nel caso di un dispositivo A che invia un file ad un dispositivo B:

1. il dispositivo A invia un file al dispositivo B utilizzando dei servizi che dipendono dalla natura del messaggio stesso. Il file, prima di essere inviato, viene convertito in un formato standard ed eventualmente criptato e compresso;

2. dopo la conversione del file, il dispositivo A deve trovare l'indirizzo del dispositivo B ed avviare una nuova sessione di comunicazione;

3. i dati sono divisi in gruppi di pacchetti, che prendono il nome di frame, e inviati al destinatario. Il sistema che gestisce la comunicazione aggiunge ai frame alcune informazioni, degli header, che garantiscono la corretta trasmissione dei pacchetti e la ricomposizione del dato di partenza. Se mancano alcuni frame viene gestita la ritrasmissione.

4. il dato viene trasmesso sul canale di comunicazione come una sequenza di bit.

I passi di una trasmissione appena elencati possono essere mappati su differenti livelli del modello OSI riportato schematicamente in figura 4.1.

Qui di seguito sono brevemente descritte le caratteristiche dei vari layer.

Strato fisico: è il livello più basso dello stack è descrive le caratteristiche che deve avere l'hardware per poter connettere un dispositivo alla rete (caratteristiche fisiche del mezzo trasmissivo e del connettore, aspetti elettrici e meccanici, modulazione e codifica del segnale).

Collegamento dati: definisce le regole di comunicazione tra due dispositivi connessi direttamente tra loro e determina la suddivisione dei dati in frame. Nelle reti di tipo broadcast (per esempio ethernet) è stato aggiunto il sottolivello MAC (Media Access Control) per consentire la condivisione dello stesso mezzo trasmissivo tra più dispositivi. Questo sublayer controlla l'accesso al mezzo trasmissivo e prevede l'assegnazione di un indirizzo hardware, unico e fissato, ad ogni

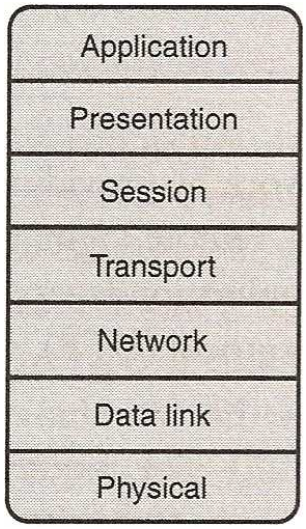


Figura 4.1: modello di riferimento OSI

dispositivo connesso alla rete. Per ogni frame trasmesso viene atteso l'acknowledge del destinatario prima di inviarne un altro, altrimenti avviene una ritrasmissione. Inoltre ad ogni frame viene aggiunto un codice CRC per il riconoscimento di errori di comunicazione.

Strato di rete: fornisce servizi di internetworking, ossia collegamento tra più reti. Per fare questo assegna indirizzi logici al mittente e al destinatario di un pacchetto e determina il migliore percorso da seguire attraverso le reti per la consegna di ogni pacchetto.

Strato di trasporto: imposta la connessione dal punto di partenza al punto di arrivo all'interno di una sessione di comunicazione e mette a disposizione un elevato grado di controllo sullo spostamento dei dati tra i due interlocutori. I dati sono trasmessi sotto forma di flusso di pacchetti numerati in successione. Lo strato di trasporto del destinatario ricompone l'informazione correttamente. Questo strato supporta l'indirizzamento dei dati su porte diverse(comunicazione tramite socket), ognuna delle quali identifica una diversa applicazione utilizzata negli strati superiori.

Strato di sessione: ha il compito di stabilire e mantenere una sessione di comunicazione tra due dispositivi. In questo strato vengono anche decise le modalità di comunicazione: simplex, half-duplex o full-duplex.

Strato di presentazione: funge da traduttore per i servizi dello strato applicazione. Tipicamente è utilizzato per convertire dati da un formato proprietario ad un formato standard, e viceversa, in modo da presentare al livello 5 dati in un formato conosciuto. In questo livello si eseguono anche operazioni di compressione/decompressione e crittatura/decrittatura.

Strato di applicazione: fornisce una serie di protocolli direttamente utilizzati dall'utente della rete tramite applicazioni software. Non riguarda l'interfaccia che l'utente utilizza per accedere ai servizi del protocollo. Alcuni protocolli appartenenti a questo strato sono: HTTP, FTP, TFTP, POP, SMTP, telnet.

Un sistema di comunicazione basato sul modello OSI deve prevedere uno stack di protocolli classificati secondo lo strato di appartenenza. Un dato che deve essere trasmesso parte dal livello più alto dello stack, quello applicativo, e man mano che scende viene elaborato da diversi protocolli i quali aggiungono una certa quantità di informazioni (headers) necessarie per il riconoscimento e la rielaborazione dei dati da parte degli stessi protocolli implementati nel dispositivo di rete del destinatario. Un dato ricevuto, invece, percorre lo stack nella direzione opposta a quella appena descritta: dallo strato fisico sale verso lo strato applicativo passando per i protocolli dei livelli intermedi. Va inoltre evidenziato che i protocolli che operano in un layer comunicano solo con i protocolli nello stesso layer dell'altro dispositivo.

4.2 Il modello di riferimento TCP/IP

L'organizzazione dello stack di protocolli TCP/IP ha una struttura diversa rispetto al modello ISO/OSI appena descritto. Quest'ultimo ha infatti validità più teorica che pratica, e va considerato come modello utile per comprendere i diversi servizi che possono andare a costituire un sistema di comunicazione.

Il modello pragmatico TCP/IP è costituito di soli 4 livelli: applicazione, trasporto, intenet e interfaccia di rete. La figura 4.2 mette in relazione i livelli appena elencati con quelli del modello ISO/OSI.

Il livello applicativo: permette all'utente di accedere alla rete per mezzo di alcuni servizi. Alcuni esempi di protocolli che operano a questo livello sono: File Transfer Protocol (FTP), Trivial FTP (TFTP) per

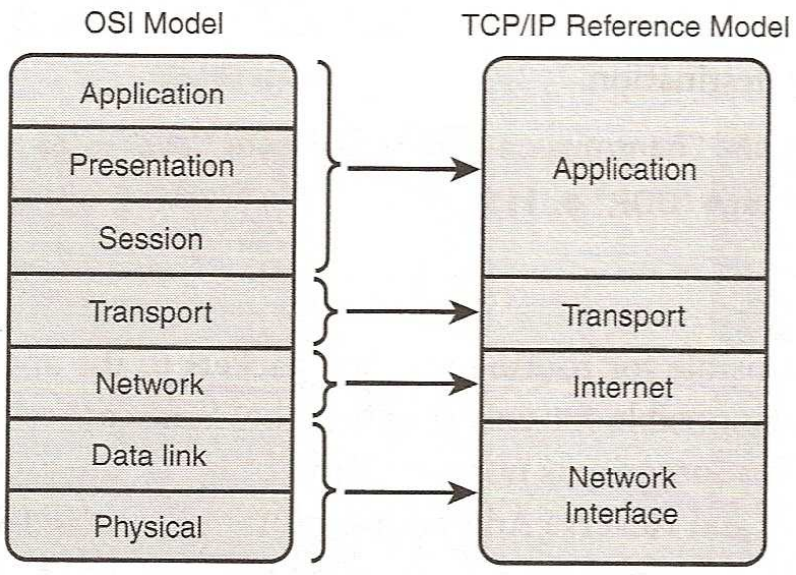


Figura 4.2: corrispondenza tra livelli OSI e TCP/IP

il traferimento di file, Hyper Text Tranfer Protocol (HTTP) per il trasferimento di testo, Simple Mail Tranfer Protocol (SMTP) per l'invio di e-mail, Telnet per il login in sistemi remoti e Domain Name System per la traduzione di alias in indirizzi IP numerici. Questo strato interagisce con il sistema operativo e il file system per effettuare la compressione, conversione e crittatura dei dati.

Il livello di trasporto: gestisce la comunicazione tra due dispositivi e comprende principalemente due protocolli: Trasmission Control Protocol (TCP) e User Datagram Protocol (UDP). Il primo offre un trasferimento dei dati estremamente controllato dalla presenza di errori, mentre il secondo non applica alcun tipo di controllo. Per realizzare questi controlli il protocollo TCP, quando deve trasferire dati, inizia una sessione di comunicazione all'interno della quale, se si verifica un errore, si prende la responsabilità di ritrasmettere pacchetti e quando non ci sono più dati da trasferire chiude la comunicazione. Per questo motivo TCP è detto protocollo orientato alla connessione. Diversamente, il protocollo UDP è detto senza connessione dato che prevede di trasferire dati senza prima aver instaurato un rapporto con il destinatario e senza preoccuparsi che la trasmissione sia andata a buon fine. Il livello di trasporto riceve i dati del livello applicativo e prima di trasmetterli li divide in unità logiche chiamate pacchetti.

Il livello internet: il protocollo più importante di questo livello è l'Internet Protocol (IP), il quale è responsabile di assicurare che i pacchetti del livello superiore giungano a destinazione. Non è tuttavia di sua competenza verificare l'integrità dei dati. L'IP interagisce con i protocolli ARP (Address Resolution Protocol) e RARP (Revers ARP), appartenenti al livello collegamento, per la risoluzione degli indirizzi IP. In particolare, ARP consente di ottenere l'indirizzo MAC (anche chiamato indirizzo fisico) corrispondente all'indirizzo IP del destinatario di un pacchetto e viceversa il RARP permette di risalire all'indirizzo IP associato ad un indirizzo MAC del mittente. Si noti che il livello collegamento è implementato sia per mezzo di software che di hardware e funge da interfaccia tra il livello internet e il livello di rete.

Al livello internet esiste anche il protocollo ICMP(Internet Control Message Protocol) che genera errori in caso di problemi nella trasmissione di dati.

Il livello rete: è responsabile della suddivisione dei dati in arrivo dal livello internet in blocchetti chiamati frame. A seconda del fatto che la comunicazione del livello trasporto sia di tipo con connessione o senza, questo livello aggiunge una diversa intestazione ad ogni frame. Nel primo caso, l'header riporta il numero di frame e l'ordine con cui devono essere riassemblati. Questo layer inoltre assicura che tutti i frame siano ricevuti correttamente applicando un codice CRC (Cyclic Redundancy Check).

In figura 4.3 è riportato in maniera schematica il flusso di dati da un computer sorgente ad un computer destinazione e tutti i protocolli fondamentali che sono coinvolti in tale comunicazione.

Si noti che ad ogni passaggio da un layer più alto ad uno più basso, ai dati che devono essere trasmessi si aggiungono delle informazioni nella forma di header (intestazioni) e trailer (un'ulteriore intestazione messa in coda al pacchetto). Queste informazioni sono utilizzate dal corrispondente protocollo del sistema ricevitore. Una rappresentazione schematica di tale comportamento è dato dalla figura 4.4. In figura 4.5 è invece riportato, come esempio, l'inserimento dell'intestazione al livello network. Il campo *Data* rappresenta l'informazione spedita dal livello superiore, ossia il livello internet.

Nei prossimi sottoparagrafi verranno trattati i protocolli che sono di maggior importanza nella realizzazione di questo progetto: il TCP, l'IP e HTTP.

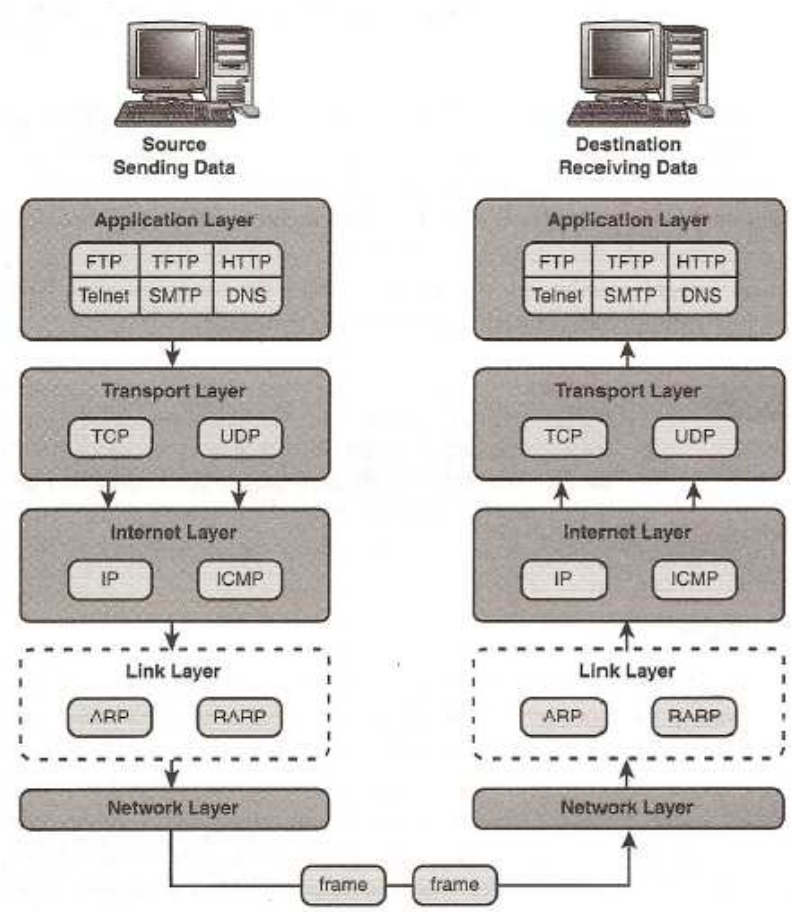


Figura 4.3: protocolli che operano in una comunicazione TCP/IP

4.2.1 Il protocollo TCP

Il protocollo TCP può essere descritto come uno standard di comunicazione che divide i dati ricevuti dal livello applicazione in pezzettini più piccoli, chiamati segmenti, e che tiene sotto controllo il loro trasferimento verso il destinatario. Il funzionamento di questo protocollo si basa principalmente sull'esplicazione di due servizi: quello che permette di stabilire una connessione e quello responsabile del trasferimento di dati.

Come abbiamo già detto in precedenza il TCP è un protocollo basato sulla connessione. Questo significa che prima di iniziare a trasferire dati, un dispositivo deve aprire, con il destinatario, una sessione di comunicazione, la quale verrà poi chiusa una volta terminato lo scambio di dati tra i due interlocutori. Gli elementi che concorrono all'apertura di una nuova connessione sono gli indirizzi e il numero della porta del mittente e del

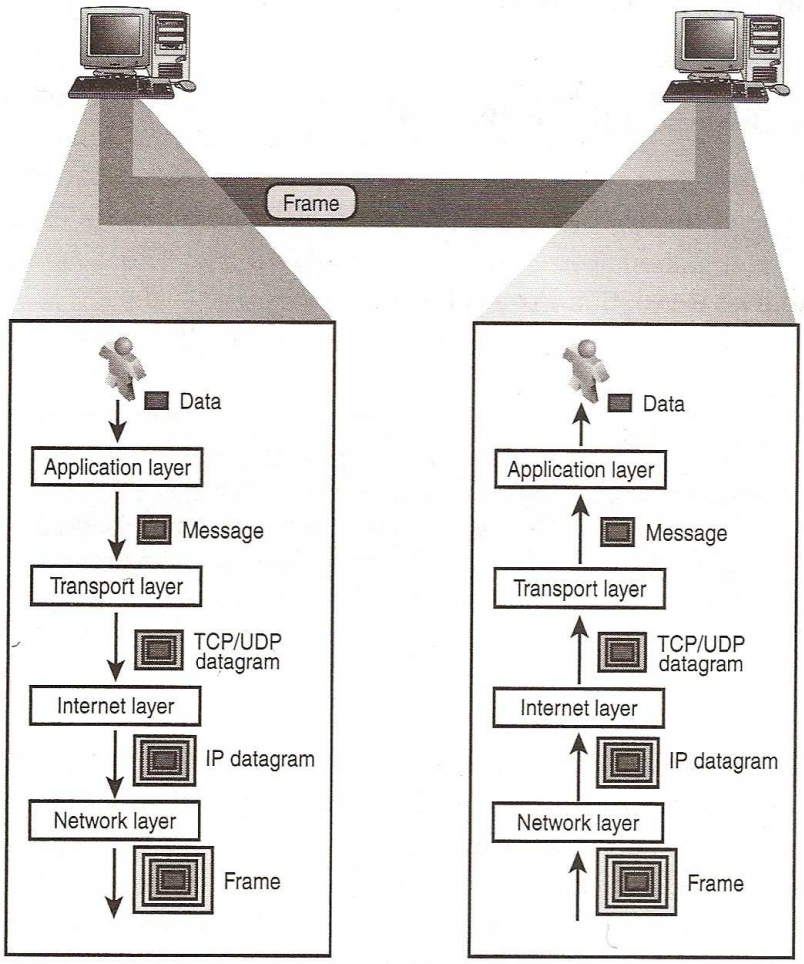


Figura 4.4: incapsulamento dei dati nei vari livelli dello stack

destinatario. I numeri di porta, o porte, sono identificatori numerici utilizzati dai protocolli TCP e UDP allo scopo di identificare più endpoint, capaci di ricevere e trasmettere dati, all'interno di uno stesso dispositivo. Un endpoint (punto terminale) a livello pratico non è altro che la combinazione dell'indirizzo IP e del numero della porta. Quando viene dato inizio ad una nuova comunicazione, il numero di porta del mittente viene deciso dal mittente stesso, mentre numero di porta del destinatario è fissato in base al tipo di servizio dei livelli superiori che si va ad usare. Per esempio, per utilizzare il protocollo HTTP l'endpoint del destinatario deve necessariamente avere come numero di porta 80. Se un'applicazione vuole comunicare con più di un dispositivo, essa deve utilizzare un endpoint condiviso tra gli endpoint degli altri interlocutori.

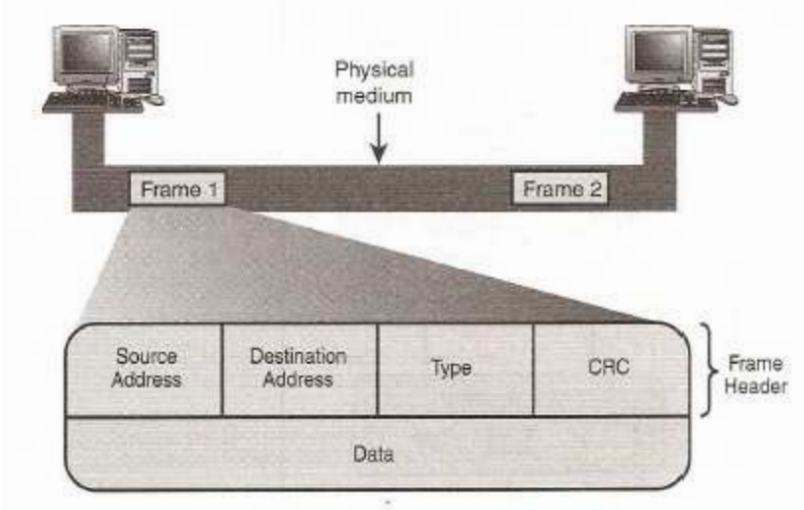


Figura 4.5: frame del livello rete

Una volta che la connessione è stata realizzata, può avere inizio la trasmissione dei dati per mezzo di appositi servizi. Il loro scopo è quello di garantire un trasferimento sicuro ed efficiente delle informazioni tra due interlocutori, evitando che il messaggio ricomposto dal destinatario sia deficitario di alcuni segmenti, o viceversa, presenti segmenti duplicati. Ogni volta che un dispositivo invia un segmento di dati, esso aspetta di ricevere la conferma della ricezione da parte del destinatario prima di inviarne uno di nuovo. Questo meccanismo viene chiamato *Acknowledgement*. Per gestire l'eventuale perdita di un segmento, il mittente, una volta effettuata la trasmissione, imposta un timer che indica il tempo entro il quale l'acknowledgement dovrebbe arrivare. Se tale timeout scade, allora il dispositivo ritrasmette il segmento. Potrebbe verificarsi, tuttavia, la situazione in cui l'acknowledgement, per svariati motivi, arrivi al trasmettitore con un ritardo superiore al tempo di timeout. Se non fosse previsto un ulteriore strumento di controllo, il destinatario riceverebbe un duplicato del segmento che è stato inviato due volte e di conseguenza invierebbe un altro segmento di acknowledgement. Per evitare questo inconveniente, ad ogni segmento corrisponde un segment number (riportato nell'intestazione del segmento stesso) che identifica la posizione del segmento nella serie di segmenti che sta per essere trasmessa. L'acknowledgement number, anch'esso appartenente al header, riporta invece il segment number del segmento che il dispositivo si aspetta di ricevere alla prossima trasmissione. In questo modo se si verifica una ritrasmissione errata il destinatario che si vede recapitare un segmento con lo stesso segment number di quello precedente,

elimina la seconda copia e non invia alcun ACK. In figura 4.6 è riportato l'esempio della trasmissione di 3 segmenti, il primo dei quali arriva subito al destinatario, il secondo viene smarrito e correttamente ritrasmesso mentre il terzo mostra un ACK che arriva dopo una ritrasmissione errata.

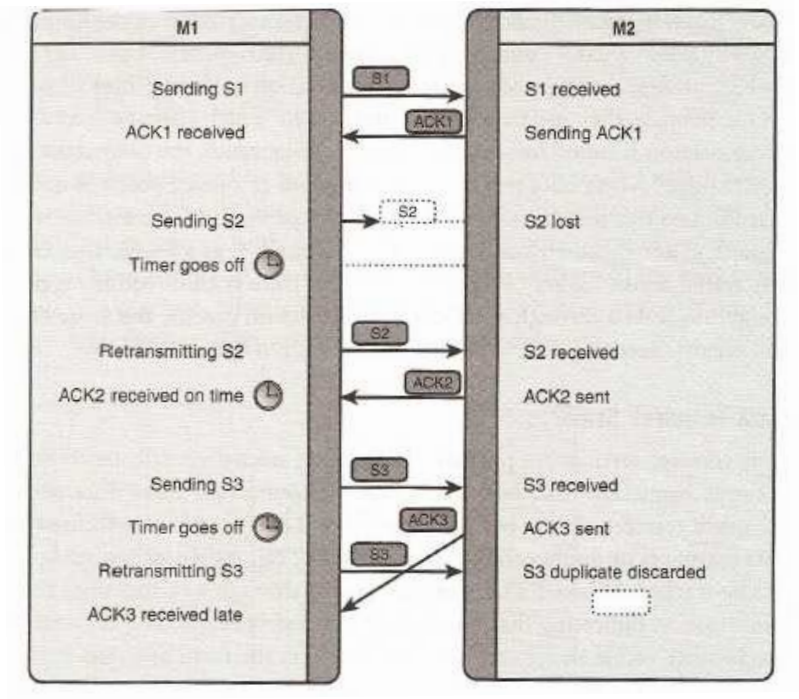


Figura 4.6: utilizzo del sequence number nel riconoscimento di duplicati

Per ottimizzare l'utilizzo del canale in una trasmissione bidirezionale, se il ricevente ha dei dati da inviare, questi possono essere inseriti nello stesso segmento che svolge il ruolo di acknowledgement.

L'applicativo che utilizza il protocollo TCP, al fine di minimizzare l'utilizzo della rete per le trasmissioni, può prevedere un buffer, ossia una certa quantità di memoria dove accumulare più dati da inviare in un solo colpo. Un processo simile può essere implementato anche al ricevitore che accumula una certa quantità di dati prima di renderla disponibile all'applicazione.

Struttura di un segmento TCP

Una rappresentazione schematica di un segmento TCP è riportata in figura 4.7.

Qui di seguito viene invece riportata una breve descrizione di ogni campo:

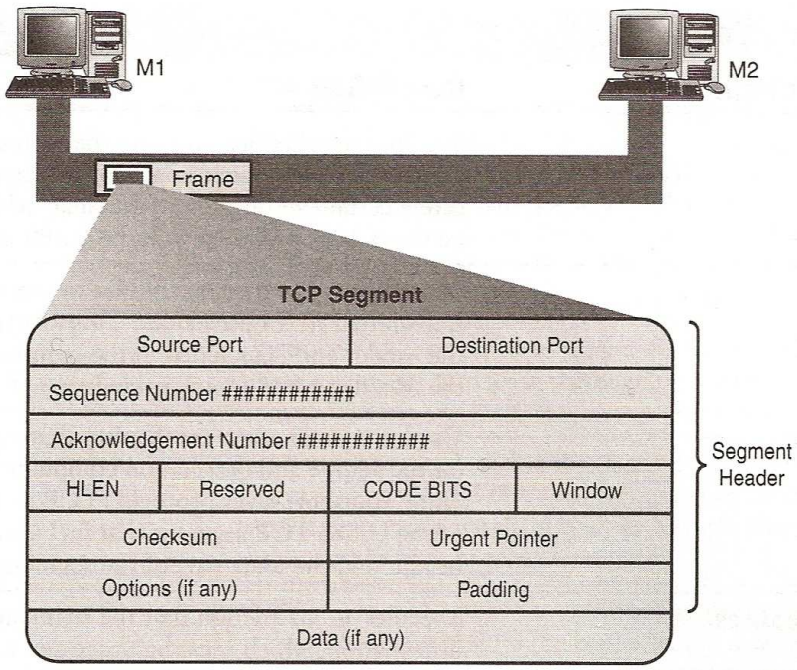


Figura 4.7: struttura di un segmento TCP

Source Port: il numero di porta di chi invia;

Destination Port: il numero di porta del ricevitore;

Segment Number: numero di sequenza del segmento;

Acknowledgement Number: indica il numero di sequenza del prossimo segmento che deve ricevere ed è dato dal Sequence Number dell'ultimo segmento ricevuto incrementato di uno;

HLEN: lunghezza dell'header;

Reserved: momentaneamente non utilizzato;

Code Bits: contiene 6 flag. SYN è usato per stabilire una connessione, ACK è usato per indicare al ricevitore che l'acknowledgement number contiene un valore valido, PSH indica al ricevitore di passare dubito il segmento all'applicazione senza metterlo nel buffer, RST rilascia immediatamente una connessione, URG indica che il segmento è da inviare subito all'applicazione e FIN indica che il segmento è l'ultimo della sequenza e che si intende terminare la sessione;

Window: specifica la massima dimensione che deve avere il prossimo pacchetto inviato;

Checksum: campo di controllo utilizzato per la verifica della validità del segmento;

Urgent Pointer: specifica che il segmento contiene informazioni urgenti che devono essere inviate subito all'applicazione. Viene utilizzato insieme al flag URG.

Option: specifica alcune opzioni accessorie.

Apertura e chiusura di una nuova connessione

Una nuova connessione viene stabilita mediante un processo che viene chiamato *Three-way handshake*, nel quale due dispositivi scambiano sequence number e acknowledgement number allo scopo di effettuare la sincronizzazione.

Tutto inizia quando il mittente chiede al suo applicativo di predisporre un endpoint al fine di creare una connessione. Questa fase prende il nome di *active open*. Il mittente invia quindi un segmento con il flag SYN attivato, ad indicare che è il primo segmento della serie. Il destinatario, a sua volta, quando riceve il segmento di sincronizzazione chiede al proprio applicativo di creare un nuovo endpoint. Questa fase prende il nome di *passive open*. Sfruttando questo endpoint il ricevitore manda un segmento di acknowledgement, con attivati i flag ACK e SYN, verso l'altro interlocutore. Il valore del campo acknowledgement number è calcolato dal valore del segment number del primo segmento di sincronizzazione incrementato di uno. Per terminare l'handshake il mittente invia un nuovo segmento di ACK. Quest'ultimo passaggio, al fine dell'inizializzazione di una nuova connessione, non sarebbe strettamente necessario. Tuttavia esso ha lo scopo di permettere ad entrambi gli interlocutori di avere una stima del tempo di timeout iniziale. Quanto appena descritto è riportato schematicamente in figura 4.8.

La chiusura di una connessione funziona in modo simile all'handshake iniziale. Tutto inizia da uno dei due interlocutori che non avendo più dati da trasferire decide di inviare un segmento di chiusura con il flag FIN attivato ad indicare che non saranno più inviati segmenti di dati. Contemporaneamente l'applicativo viene informato della volontà di chiudere la connessione. Alla ricezione del FIN segment il ricevitore avvisa il proprio applicativo che non seguiranno altri segmenti di dati ed invia un segmento di ACK. A questo punto per terminare completamente la comunicazione,

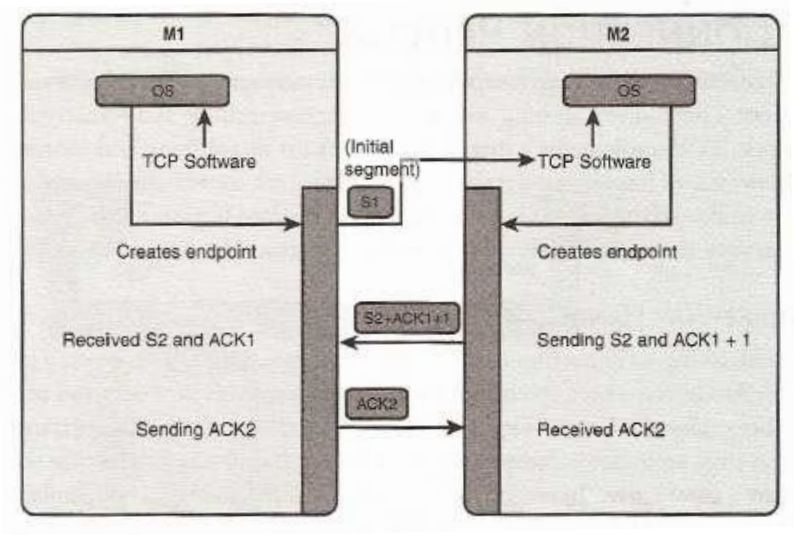


Figura 4.8: passi di un Three-way handshake

il ricevitore informa il proprio applicativo di chiudere la connessione e invia un FIN segment all'interlocutore. Quest'ultimo informa a sua volta il proprio applicativo che non ci sono più segmenti di dati in attesa e risponde con un ulteriore ACK. Alla fine di questa procedura gli applicativi di entrambi i dispositivi distruggono gli endpoint e rilasciano le risorse allocate per quella comunicazione. Quanto appena descritto è riportato schematicamente in figura 4.9.

4.2.2 Il protocollo IP

Il protocollo IP (Internet Protocol) svolge la sua funzione nel livello Internet del modello di riferimento TCP/IP. Il suo compito è essenzialmente quello di trasportare dal mittente al destinatario i segmenti provenienti dagli strati superiori. Le principali caratteristiche di questo protocollo sono qui elencate:

- IP realizza sempre trasmissioni di tipo senza connessione. Ogni datagramma è quindi un'unità di dati indipendente e libera di raggiungere il mittente seguendo una strada diversa dagli altri datagrammi;
- IP consente il trasferimento di dati tra dispositivi appartenenti alla stessa rete (local delivery) oppure a reti diverse (remote delivery).
- il mittente non è a conoscenza del percorso seguito dal messaggio durante la trasmissione. Il datagramma viene rimbalzato da un rou-

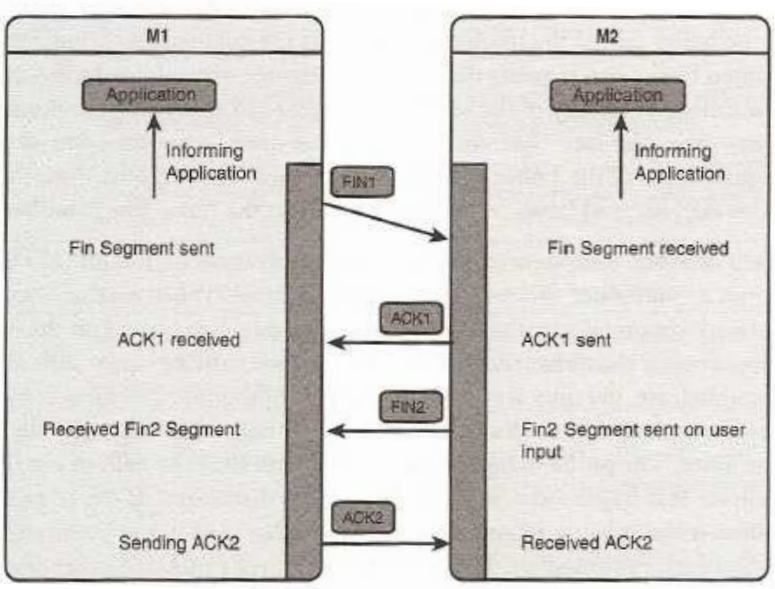


Figura 4.9: passi della chiusura di una connessione

ter (instradatore) ad un altro e giunge a destinazione seguendo il percorso più breve. Questo concetto è simile quello utilizzato dal servizio postale il quale trasferisce un pacco da un ufficio all'altro per raggiungere la destinazione nel più breve tempo possibile.

- ogni datagramma per essere inviato viene passato al livello sottostante il livello internet, dove viene incapsulato all'interno di un frame

- questo protocollo non gestisce la perdita dei datagrammi. IP lavora assieme al protocollo ICMP, appartenente allo stesso livello del modello di riferimento, il quale è responsabile di generare messaggi di errore qualora si verifichino problemi di trasmissione.

La struttura di un datagramma IP è riportata in figura 4.10. Tra gli elementi che compongono un datagramma vale la pena menzionare il campo *Protocol* il quale riporta il protocollo che ha generato l'informazione contenuta nel campo Data e *Time to Live* che riporta, in secondi, il tempo per il quale il datagramma può essere mantenuto in circolazione per la rete. Quest'ultimo parametro è necessario per assicurarsi che un pacchetto di dati non continui a rimbalzare tra un router e un altro, in cerca della destinazione, per un tempo indefinito. Il Time to Live è impostato dalla

sorgente ad un valore legato al tipo di servizio che il pacchetto deve svolgere e viene decrementato da ogni router incontrato nel percorso verso la destinazione.

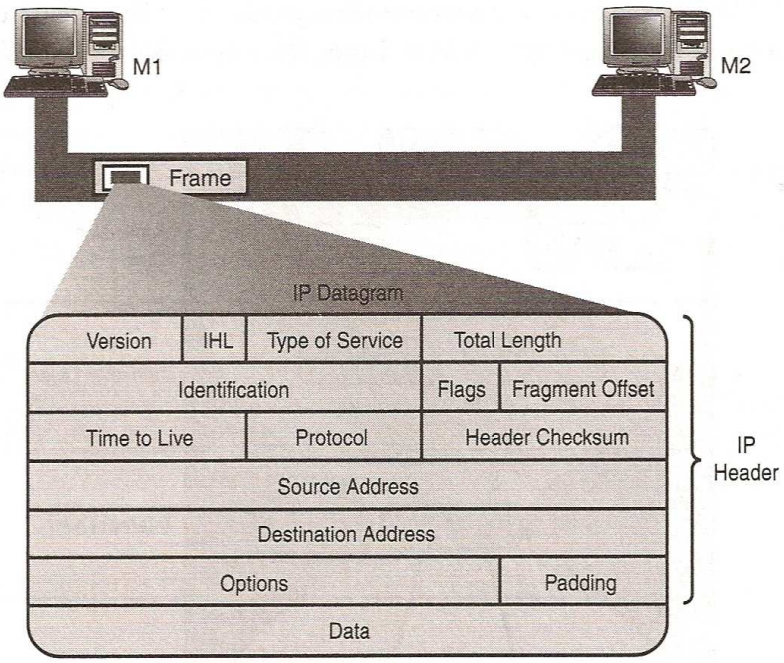


Figura 4.10: datagramma IP

Per consentire il corretto indirizzamento di un datagramma, il protocollo IP identifica in modo univoco ciascuna macchina collegata ad una rete basata sullo stack TCP/IP con un numero. Nella versione 4 del protocollo, questo numero è a 32 bit, mentre nella versione 6, nata allo scopo di aumentare il numero di macchine collegabili alla rete internet, per ogni indirizzo si utilizzano 48 bit. Per rappresentare questi numeri in un formato leggibile e pratico, si fa uso della notazione puntata, ossia con ciascun ottetto separato da un punto e convertito in numero decimale (es. 192.168.1.122). L'indirizzo IP può essere visto come composto da due parti: i bit più significativi indicano la rete alla quale il dispositivo è connesso e prendono il nome di *prefisso di rete*, mentre i bit meno significativo identificano il singolo dispositivo all'interno di una rete e prendono il nome di *numero dell'host*. Gli host possono comunicare tra loro, da punto a punto, solo se appartengono alla stessa rete, cioè se hanno lo stesso prefisso. Per mettere in comunicazione due host appartenenti a reti diverse si rende necessario l'utilizzo di un router (instradatore). Questo dispositivo è dotato di diverse interfacce di rete, ognuna delle quali è collegata ad una rete diversa

e provvede a reindirizzare i pacchetti che riceve da ciascuna scheda verso la rete di destinazione identificata dal prefisso di rete del destinatario di ciascun datagramma.

Per aumentare la flessibilità di costruzione delle reti, il numero di bit destinati al prefisso non è fissato. Esistono quindi reti di classe A, B e C che dedicano al numero di rete rispettivamente 1, 2 e 3 ottetti e lasciano 3, 2 e 1 ottetti per il numero di host. Togliendo i bit impiegati per identificare la classe e i bit riservati, si possono ottenere 126 reti di classe A ciascuna con 16777214 possibili host, 16384 reti di classe B, ciascuna con 65534 possibili host e 2097152 reti di classe C, ciascuna con 254 possibili host. Esistono inoltre reti di classe D, destinate a comunicazioni multicasting, e reti di classe E dedicate ad usi sperimentali. Nonostante esista questa suddivisione, da diversi anni per aumentare l'efficienza nell'assegnazione degli indirizzi IP non si utilizzano più le classi di rete ma la divisione tra il numero di rete e quello di host può avvenire in un bit qualsiasi dell'indirizzo.

4.2.3 Il protocollo HTTP

Il protocollo HTTP, che agisce al livello application dello stack di protocolli TCP/IP, è responsabile dell'effettivo trasferimento dei dati relativi ai contenuti web. HTTP è un protocollo di tipo client - server nel quale il client invia una richiesta e il server restituisce una risposta. Questo meccanismo è illustrato chiaramente nella figura 4.11.

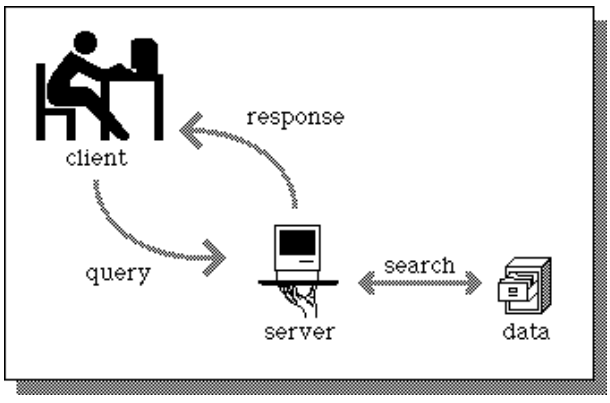


Figura 4.11: sistema client-server

L'ultima versione di questo protocollo presenta un'importante evoluzione rispetto a quella precedente, nella quale la richiesta di un'informazione e la sua risposta venivano trasmesse in sessioni di comunicazione diver-

se. L'effettivo miglioramento deriva proprio dal fatto che la richiesta di una risorsa e la risposta sono trasmesse su una connessione persistente che rimane attiva fino a che il client o il server la termina volontariamente. Tutto ciò aumenta notevolmente l'efficienza in termini di tempo per effettuare una trasmissione e di quantità di pacchetti scambiati tra il client e il server per la continua apertura e chiusura di connessioni TCP.

La gestione del protocollo HTTP prevede l'utilizzo di un software diverso per il lato server e il lato client. Nel primo caso, il programma rimane in ascolto di connessioni in arrivo ed è in grado rispondere ad eventuali richieste dei dati che ha a disposizione da parte dei client. Questi ultimi, a loro volta, accedono al servizio HTTP solitamente per mezzo del browser web il quale, in modo completamente invisibile all'utente del computer, utilizza i comandi previsti dal protocollo per richiedere di inviare dati al server. Le informazoni scambiate in una comunicazione HTTP sono di solito codificate in un linguaggio chiamato Hypertext Markup Language (HTML), che permette di definire la struttura e l'organizzazione dei contenuti in una pagina web, fornendo istruzioni interpretate dal browser stesso. Negli ultimi anni, tuttavia, si è vista una rapida evoluzione della tipologia di contenuti presentati sulle pagine web e di conseguenza i browser di ultima generazione sono in grado di interpretare messaggi contenenti informazioni relative a svariati tipi di file. Per consentire ciò, dato che il protocollo HTTP supporta solamente il trasferimento di informazione in formato ASCII, qualunque file di formato diverso (jpeg, mp3, swf, ecc.) è codificato dal server e decodificato dal browser secondo le specifiche del relativo MIME type (Multipurpose Internet Mail Extensions). Nei seguenti paragrafi sarà descritta la struttura di una richiesta e della relativa risposta HTTP.

Richiesta HTTP

Lo scopo di una rihiesta HTTP è quello di recuperare i contenuti di una pagina web da un server. Per fare ciò, il messaggio di richiesta è composto da quattro campi: *request line*, *header*, *blank line* e *body*. Il campo body è tipicamente lasciato vuoto, ma può anche contenere informazioni di codifica o dati da inviare al server. La blank line ha solo lo scopo di dividere l'header dal body ed è realizzata tramite i due caratteri ASCII CR (Carriage Return) e LF (Line Feed). L'header contiene invece una serie di informazioni aggiuntive che vengono utilizzate dal browser. Queste informazioni si possono riferire al nome del server legato all'indirizzo URL, al formato dei dati che devono essere inviati dal server, ai permessi del client ad accedere alla risorsa richiesta, al numero di porte del client, alla versio-

ne del browser che ha inviato la richiesta e ad alcuni criteri con i quali il server deve processare la richiesta medesima (ad esempio, richiesta solo di pagine create dopo una certa data). La request line è senza dubbio il campo più importante in un messaggio di richiesta; essa contiene informazioni relative al tipo di richiesta, all'URL (Uniform Resource Locator) di destinazione (cioè l'indirizzo del webserver) e la versione del protocollo HTTP implementata. Il tipo di richiesta, che è spesso chiamata *metodo*, contiene istruzioni riguardanti le azioni che il server deve eseguire per soddisfare la richiesta del client. I metodi più utilizzati sono:

- **GET:** indica che il messaggio di risposta del server dovrà contenere nel campo body il documento indicato dall'URL;
- **POST:** indica che il server dovrà accettare i dati contenuti nel body del messaggio di richiesta come input proveniente dal client;
- **HEAD:** richiede le informazioni contenute nell'header del documento indicato nell'URL; può essere utilizzato, per esempio, per verificare la validità di un hyperlink e per vedere se è stato modificato recentemente;
- **PUT:** indica che il documento contenuto nel body del messaggio deve essere memorizzato nel server all'indirizzo indicato dall'URL; se il documento è già esistente, la versione precedente verrà sovrascritta.

Un esempio di richiesta GET è quella che si fa quando dal proprio browser si inserisce l'indirizzo di una pagina web per visualizzarla. Un esempio di richiesta POST è quella che si realizza quando si compila un form in una pagina web e si inviano i dati al server cliccando il pulsante *submit*. Anche in questo caso, la risorsa richiesta è indicata dall'URL contenuto nella request line. Qui di seguito viene riportato un esempio di richiesta GET:

```
GET /index.html/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0
(compatible; Konqueror/3.2; Linux) (KHTML, like Gecko)
Accept: text/html, image/jpeg, text/*, image/*
Accept-Encoding: x-gzip, x-deflate, gzip, deflate
Accept-Charset: iso-8859-1, utf-8;q=0.5, *;q=0.5
Accept-Language: en
Host: www.google.it
```

Una rappresentazione intuitiva dell'inoltro di una richiesta HTTP è riportata in figura 4.12.

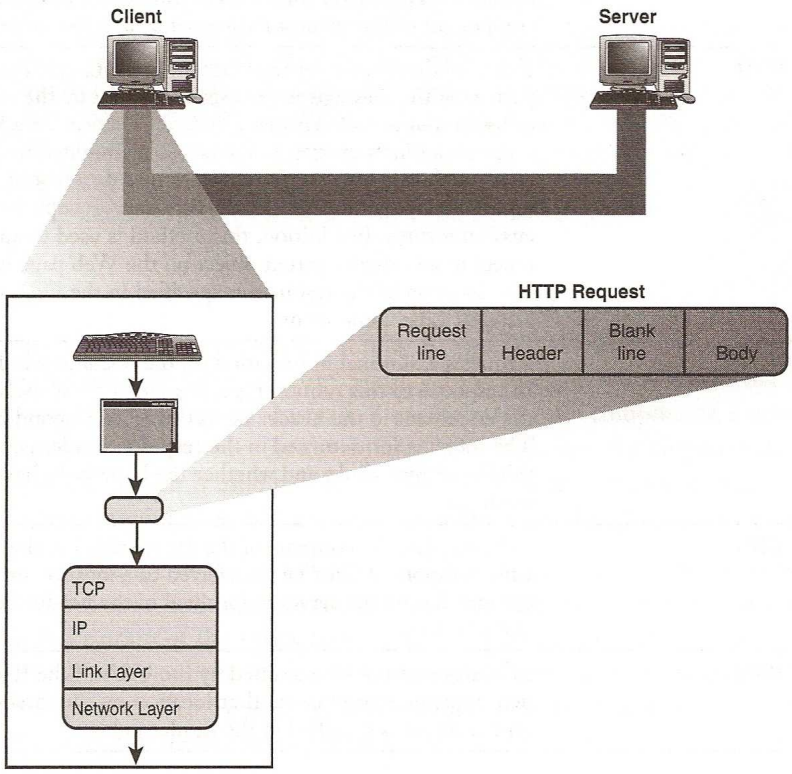


Figura 4.12: richiesta HTTP

Risposta HTTP

La struttura di una risposta HTTP è abbastanza simile a quella della richiesta. La differenza sostanziale sta nel contenuto del campo header e nel fatto che al posto della request line troviamo la status line. Come nella richiesta, è presente il campo body, che è utilizzato per trasferire informazioni di codifica o i dati richiesti dal client. La status line contiene informazioni relative al protocollo HTTP implementato sul server, un codice che indica lo stato della corrispondente richiesta e una descrizione di tale codice. Esistono ben 35 codici di stato divisi in base al tipo di informazione cui si riferiscono. I codici di stato più comuni sono:

- **200 OK:** il server ha fornito correttamente il contenuto nella sezione body;
- **400 Bad Request:** la risorsa richiesta non è comprensibile al server;
- **404 Not Found:** la risorsa richiesta non è stata trovata e non se ne conosce l'ubicazione; di solito avviene quando l'URL è stato indicato

in modo errato, oppure è stato rimosso il contenuto dal server;

- **500 Internal Server Error:** il server non è in grado di rispondere alla richiesta per un suo problema interno;
- **505 HTTP Version Not Supported:** la versione di http non è supportata.

Il campo header contiene informazioni riguardanti l'età della pagina web, i metodi supportati dal server, la versione del server stesso e la codifica dei dati restituiti (MIME type)

Qui di seguito viene riportato un esempio di risposta

```
HTTP/1.0 200 OK
Date: Mon, 28 Jun 2004 10:47:31 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4
X-Powered-By: PHP/4.3.4
Vary: Accept-Encoding,Cookie
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: it
Content-Type: text/html; charset=utf-8
Age: 7673
Connection: close
```

Una rappresentazione intuitiva dell'invio di una risposta del server verso il client è riportata in figura 4.13.

4.3 Realizzazione del webserver sul modulo Rabbit

La realizzazione del webserver è passata attraverso la configurazione dei protocolli dello stack TCP/IP. Per fare questo, il Dynamic C mette a disposizione numerose estensioni e librerie.

La prima configurazione che è necessario realizzare è quella relativa al protocollo IP, il quale assicura un metodo efficace per trasferire i dati da e verso i client. I parametri di rete che devono essere impostati sono l'indirizzo IP, il subnet mask, l'indirizzo del gateway ed eventualmente il nome del server. Essi vengono impostati all'inizio del codice tramite la definizione dei seguenti valori:

- _PRIMARY_STATIC_IP: definisce l'indirizzo IP del server;

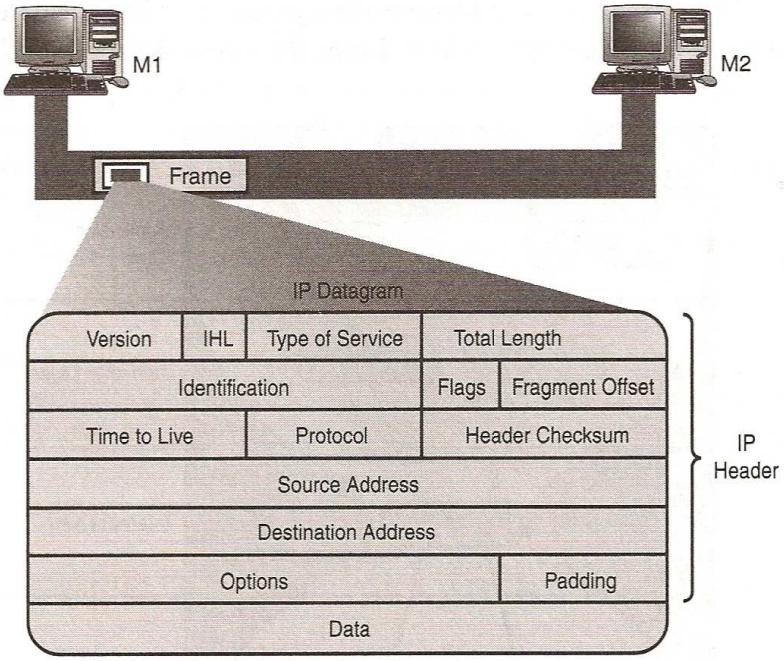


Figura 4.13: risposta HTTP

- _PRIMARY_NETMASK: è la maschera di sottorete che permette di definire quanti bit dell'indirizzo IP sono destinati ad identificare il server all'interno della rete;

- MY_GATEWAY: identifica l'indirizzo del router che permette l'interfacciamento della macchina verso altre reti;

- MY_NAMESERVER: identifica il nome del server.

Tutti i parametri precedenti devono essere impostati mediante la direttiva C `#define` con il rispettivo indirizzo espresso in notazione puntata e in forma stringa. È conveniente inoltre definire il parametro TCPCONFIG con il valore 1 per impostare l'indirizzo IP in modo statico. Nell'eventualità che si vogliano ricavare i parametri di rete mediante l'utilizzo del protocollo DHCP (Dynamic Host Configuration Protocol), è possibile impostare il parametro TCPCONFIG al valore 5. Questo tipo di configurazione viene realizzato in fase di compilazione del codice C; risulta quindi scomodo nel caso fosse necessario cambiare i parametri di rete. In questa eventualità, è possibile impiegare la funzione *ifconfig()* per andare a sovrascrivere in fase di esecuzione i parametri appena elencati.

Il secondo passo consiste nella configurazione e inizializzazione della libreria relativa al protocollo TCP. Questa è responsabile della realizzazione di una comunicazione robusta tra client e server. Anche in questo caso ci si avvale di alcune funzioni di interfacciamento. In particolare, la funzione *sock_ init()* permette di inizializzare le strutture dati interne (tra cui anche il buffer di ingresso e il buffer di uscita) e di attivare il chip responsabile della comunicazione ai livelli data link e fisico. Per il corretto funzionamento del webserver è inoltre conveniente definire i seguenti parametri: MAX_TCP_SOCKET_BUFFERS e TCP_BUF_SIZE. Il primo definisce il numero massimo di socket TCP che possono essere gestiti contemporaneamente; questo valore può influenzare la velocità di risposta del server HTTP: più è alto, maggiori saranno le risorse di sistema dedicate al protocollo TCP e di conseguenza maggiore sarà l'impiego di memoria RAM. Il secondo parametro, invece, imposta le dimensioni dei buffer di ingresso e di uscita e, in maniera simile al precedente, va ad influenzare la velocità del sistema e il consumo di risorse. Nel caso si preveda di progettare pagine web con molti elementi che richiedono autonomamente la connessione al webserver (come immagini o funzioni per l'aggiornamento di variabili), è conveniente utilizzare anche la funzione *tcp_ reserveporte(80)*, la quale implementa per ogni socket disponibile una coda delle richieste di connessione pendenti. Qualora questa funzione non fosse impiegata, una richiesta di connessione che non dovesse trovare un socket disponibile verrebbe semplicemente rifiutata. È sufficiente effettuare un'unica chiamata di tale funzione all'interno del main per attivare questa opzione. Un'altra funzione che è necessario utilizzare per fare effettivamente funzionare il protocollo TCP è *tcp_ tick()*, la quale deve essere richiamata all'interno del ciclo infinito del programma per processare i pacchetti in arrivo sul buffer di ingresso. È opportuno inserire questa funzione in un task diverso da quello che gestisce la comunicazione seriale al fine di poter trarre vantaggio dal multitasking cooperativo e garantire la corretta esecuzione del servizio HTTP anche in caso di malfunzionamento della seriale.

La realizzazione del webserver procede con la configurazione del protocollo HTTP. Anche in questo caso una funzione di inizializzazione, http _init(), provvede a resettare le librerie ad uno stato noto e ad allocare le risorse necessarie al funzionamento del server. È necessario che questa funzione venga richiamata all'interno del main immediatamente dopo la funzione sock_init() menzionata precedentemente. Al server HTTP è riservata un'apposita funzione, http_handler(), che svolge il ruolo di elaborazione dei pacchetti in arrivo sul buffer di ingresso e che può essere impiegata al posto della funzione tcp_tick() che è ad uso generico per

tutti i servizi che sfruttano il protocollo TCP. A questo punto, le librerie che implementano lo stack di protocolli TCP/IP sono adeguatamente configurate ed il loro funzionamento è autonomo.

Lo step successivo è quello di organizzare le risorse che devono far parte del webserver e di impostare i gruppi di utenti ed i relativi permessi di accesso alle risorse stesse. Anche in questo caso, il Dynamic C fornisce alcune librerie che facilitano queste operazioni. La libreria più importante è quella che implementa ciò che viene chiamato Zserver e che svolge appunto la funzione di resource manager. Il suo scopo principale è quello di recuperare le risorse immagazzinate nelle memorie flash utilizzate dal modulo e di organizzarle nel filesystem virtuale del webserver. La figura 4.14 evidenzia il ruolo centrale di questo componente nel funzionamento del server HTTP.

La configurazione del Zserver inizia importando i file che andranno a comporre il server nella memoria flash del modulo Rabbit. Questa operazione è resa possibile grazie all'utilizzo della direttiva del Dynami C `#ximport`. Di seguito è riportato un esempio di come sono stati importati alcuni file che caratterizzano il funzionamento del server:

```
#ximport "/index.html" index_html
#ximport "/grafico_1.swf" grafico_1_swf
#ximport "/images/logo.jpeg" logo_jpeg
```

Il valore riportato tra le virgolette rappresenta il percorso dove il file è memorizzato rispetto alla cartella del progetto contenente il file C che si sta scrivendo. Il secondo parametro è invece una label che identifica l'indirizzo della memoria nel quale viene copiato il file che si desidera importare. Una volta importati tutti i file è necessario configurare il filesystem virtuale del server andando a determinare la struttura delle cartelle e l'organizzazione dei file all'interno di esse. Per compiere questa operazione è sufficiente utilizzare delle macro messe a disposizione dalla libreria Zserver. Per una più facile comprensione, qui di seguito è riportato un esempio di utilizzo di queste macro:

```
SSPEC_RESOURCETABLE_START
  SSPEC_RESOURCE_XMEMFILE ("/index.html", index_html)
  SSPEC_RESOURCE_XMEMFILE ("/grafici/grafico1.swf",
  grafico_1_swf)

  SSPEC_RESOURCE_XMEMFILE ("/immagini/logo.jpeg",
  logo_jpeg)
SSPEC_RESOURCETABLE_END
```

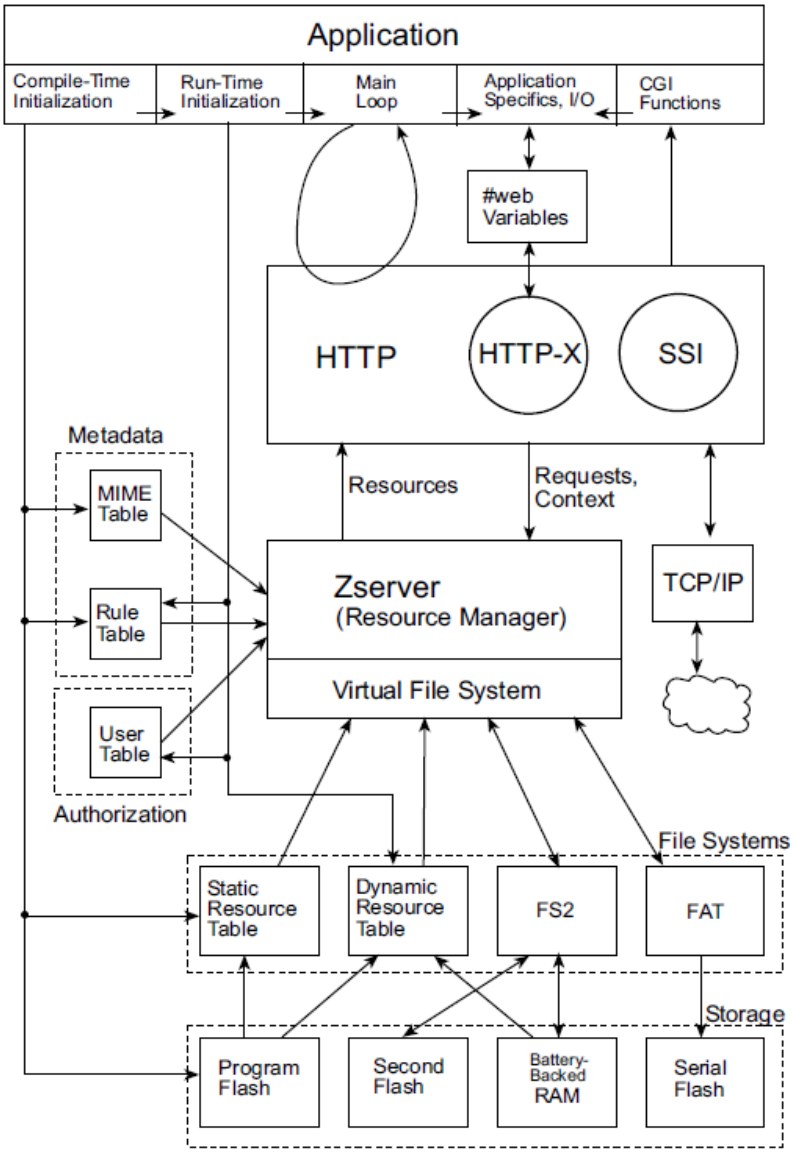


Figura 4.14: struttura del webserver

SSPEC_RESOURCE_XMEMFILE permette di inserire un nuovo file sulla tabella delle risorse statiche. Si noti che queste macro sono eseguite in fase di compilazione, perciò non consentono l'aggiunta di risorse al server web in fase di esecuzione. Per far questo è eventualmente possibile utilizzare delle apposite funzioni. Il primo parametro passato alla macro SSPEC_RESOURCE_XMEMFILE è la collocazione che si vuole dare al file all'interno del filesystem virtuale, mentre il secondo parametro è la

stessa label utilizzata in fase di importazione del file sulla memoria flash.

Altre risorse di fondamentale importanza alle quali è necessario dare una collocazione all'interno del server sono le funzioni CGI (Common Gateway Interface). Queste particolari funzioni non sono indispensabili per la realizzazione di semplici webserver, ma se si desidera ottenere elevati gradi di dinamicità delle pagine web esse diventano uno strumento molto potente e flessibile. Come vedremo nel seguente capitolo queste funzioni sono state impiegate per passare alle pagine web i valori delle variabili di stato della macchina che richiedono un aggiornamento asincrono rispetto al caricamento delle pagine stesse. Le funzioni CGI sono scritte direttamente all'interno del file C che caratterizza il funzionamento del server e vengono incluse all'interno della tabella delle risorse statiche precedentemente menzionata attraverso l'utilizzo della macro SSPEC_RESOURCE_FUNCTION. Qui di seguito è riportato un esempio di come viene utilizzata questa macro:

```
SSPEC_RESOURCETABLE_START
   SSPEC_RESOURCE_XMEMFILE ("/index.html", index_html)
   SSPEC_RESOURCE_XMEMFILE ("/grafici/grafico1.swf",
   grafico_1_swf)

   SSPEC_RESOURCE_XMEMFILE ("/immagini/logo.jpeg",
   logo_jpeg)
   ....
   SSPEC_RESOURCE_FUNCTION ("/function/getValue.cgi"
   get_value)
SSPEC_RESOURCETABLE_END
```

Nell'esempio riportato il valore get_value corrisponde al nome della funzione inclusa nel codice C, mentre un client può richiedere l'esecuzione di tale funzione semplicemente facendo una richiesta di tipo GET all'indirizzo http://[indirizzo_ip]/function/getValue.cgi.

Dopo aver costruito la struttura delle cartelle del server ed aver assegnato una collocazione a tutti i file, è possibile aggiungere dei gruppi di utenti ed assegnare ad ognuno di questi dei diritti di accesso alle risorse del server. Questo, in molte applicazioni, dove l'accesso alla rete è controllato, può risultare superfluo, ma nel caso del controllo che si vuole realizzare, l'accesso alle pagine web da parte di un utente non autorizzato può rappresentare un problema. Anche in questo caso, il Dynamic C mette a disposizione alcune funzioni e direttive per aggiungere le limitazioni di accesso desiderate. La prima cosa da fare è quella di indicare al server che tipo di autenticazione, basic o digest, si esige per accedere alle risorse protette. Questo viene realizzato defi-

nendo il parametro USE_HTTP_BASIC_AUTHENTICATION oppure USE_HTTP_DIGEST_AUTHENTICATION al valore 1 in base alle proprie necessità. In questo caso, si è deciso di utilizzare l'autenticazione base dato che l'autenticazione digest complica il passaggio di parametri attraverso la barra degli indirizzi del browser, tecnica, quest'ultima, in alcuni casi impiegata nella realizzazione di questo server. Il secondo passo è quello di indicare al Zserver quali gruppi di utenti si intende utilizzare. Per fare questo, è necessario usare la direttiva #web_groups seguita dai nomi indicativi dei gruppi di utenti. In questo progetto si è deciso di prevedere due gruppi di utenti: *normaluser*, il quale dà ai propri utenti il permesso di accedere solamente alle funzioni di base della gestione, e *admin*, il quale permette ai propri utenti di utilizzare tutte le funzionalità del server. Per impostare tali diritti si è fatto uso delle funzioni sauth_adduser(), sauth_usermask() e sspec_addrule(). La prima permette di creare un nuovo utente del server HTTP e di assegnargli una password. La seconda funzione, invece, consente di assegnare ad un utente precedentemente creato l'appartenenza ad uno dei gruppi di utenti previsti. L'ultima funzione ha lo scopo di impostare quali gruppi di utenti possono accedere ad una determinata cartella del filesystem virtuale. Qui di seguito è riportata la configurazione che è stata adottata in questo server:

```
sspec_addrule("/","Amministratore o utente semplice",
user,user, SERVER_HTTP,SERVER_AUTH_DIGEST, NULL);

sspec_addrule("/","Amministratore o utente semplice",
admin|user, admin|normaluser, SERVER_HTTP,
SERVER_AUTH_DIGEST, NULL);

sspec_addrule("/administrator", "Amministratore",
admin, admin, SERVER_HTTP, SERVER_AUTH_DIGEST, NULL);

userid = sauth_adduser("Pippo", "password",
SERVER_HTTP);

sauth_setusermask(userid, admin, NULL);

sauth_setusermask(sauth_adduser("Pluto", "PASSWORD",
SERVER_HTTP),user, NULL);
```

In questo caso vediamo che sia gli utenti appartenenti al gruppo *normaluser* che quelli appartenenti al gruppo *admin* hanno accesso a tutti i file (l'indirizzo indicato è infatti la root del filesystem), mentre solo gli

utenti appartenenti al secondo gruppo possono accedere ai contenuti della cartella administrator e alle relative sottocartelle.

L'ultima configurazione che rimane da fare per consentire il funzionamento del server HTTP con tutte le sue potenzialità è quella relativa ai MIME type (Multimedia Internet Mail Extension) i quali sono resposabili della corretta interpretazione dei file non ASCII scaricati dal server. I MIME vengono infatti indicati nell'intestazione della risposta HTTP del server affinché i client siano in grado di codificare contenuti come immagini e suoni comunque trasferiti come sequenza di caratteri ASCII. Per una facile comprensione delle modalità con le quali alle estensioni dei file possono essere associati i rispettivi MIME type all'interno del server Rabbit, di seguito viene riportato un esempio:

```
SSPEC_MIMETABLE_START
  SSPEC_MIME_FUNC( ".shtml", "text/html", shtml_handler),
  SSPEC_MIME( ".cgi", ""),
  SSPEC_MIME(".js","text/html"),
  SSPEC_MIME(".css","text/css"),
  SSPEC_MIME(".jpg","image/jpeg"),
  SSPEC_MIME(".jpeg","image/jpeg"),
  SSPEC_MIME( ".gif", "image/gif"),
  SSPEC_MIME_FUNC(".zhtml", "text/html", zhtml_handler),
  SSPEC_MIME(".html", "text/html"),
  SSPEC_MIME( ".swf", "application/x-shockwave-flash")
SSPEC_MIMETABLE_END
```

Capitolo 5

Dinamicità delle pagine web

5.1 Il linguaggio HTML

Alla fine degli anni Ottanta, presso il CERN di Ginevra, vennero sviluppati da Tim Berners-Lee il linguaggio HTML (HyperText Markup Language) e il protocollo HTTP che consente il trasferimento di documenti scritti in tale formato. Il massiccio impiego di queste tecnologie è tuttavia iniziato più di un decennio dopo, nel 1994 circa, quando si cominciarono ad avere i primi utilizzi a livello commerciale del web. L'HTML, come dice il nome stesso, è un linguaggio di marcatura che permette di indicare come devono essere disposti gli elementi di una pagina web. I web browser, come Internet Explorer e Mozilla Firefox, sono programmi che interpretano il contenuto di un documento HTML allo scopo di comporre la pagina web disponendo testi, immagini, suoni ed oggetti vari. In realtà HTML non ha le capacità di definire l'aspetto finale di una pagina web, ma ha solamente lo scopo di indicarne il contenuto logico. Questo significa essenzialmente che una pagina web scritta in linguaggio HTML può essere visualizzata in modo differente su dispositivi diversi. Va sottolineata la differenza tra i linguaggi di programmazione, dotati di variabili, strutture, funzioni e strutture di controllo, e i linguaggi di marcatura che invece, per così dire, non sono in grado di prendere decisioni o di cambiare il loro comportamento in funzione di eventi. Il componente principale della sintassi del linguaggio HTML è l'elemento, termine con il quale viene indicata la struttura di base che svolge la funzione di formattazione dei contenuti della pagina o che si occupa di passare delle informazioni al browser. Ogni elemento è racchiuso all'interno di marcature dette tag, costituite da una sequenza di caratteri racchiusa tra due parentesi angolari, cioè i segni minore e maggiore (esempio:
 è il tag che serve per indicare un ritorno a

capo). Una pagina HTML ha una particolare struttura che potremmo definire annidata. Un documento scritto in questo linguaggio inizia con l'indicazione del tipo di documento (Document Type Definition, DTD) la quale fornisce al browser l'indirizzo URL che contiene le specifiche HTML utilizzate per scrivere il documento. La parte rimanente del documento è racchiusa tra i tag <html> e contiene le informazioni riguardanti la struttura della pagina. In particolare, questa sezione del documento è suddivisa a sua volta in una sezione di intestazione, racchiusa tra i tag <head>, e nel corpo del documento, racchiuso tra i tag <body>. Di seguito è riportato un esempio minimale di pagina HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//
EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
        lang="en" xml:lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html;
        charset=ISO-8859-1" />
</head>

<body>
<p>Work in progress</p>
<a href="/index.html">Torna alla pagina iniziale</a>

</body>

</html>
```

5.2 Linguaggi di programmazione di pagine dinamiche

Il linguaggio HTML non è in grado di generare pagine web dinamiche dove la struttura e il contenuto del documento possano variare. Per fare ciò è necessario ricorrere all'utilizzo di linguaggi di programmazione lato client o lato server. I linguaggi lato server vengono eseguiti da un apposito interprete nel momento in cui una pagina web viene richiesta e in base al risultato dell'elaborazione restituiscono un documento con contenuti e strutture diversi. Esempi di linguaggi lato server molto utilizzati sono PHP e ASP. Diversamente, il codice dei linguaggi lato client può essere inserito all'interno del codice HTML di una pagina web tramite l'utilizzo

di appositi tag e viene eseguito dal web browser in fase di caricamento della pagina. Il linguaggio di programmazione lato client più diffuso è JavaScript, il quale è stato ampiamente utilizzato nella realizzazione delle pagine di gestione dell'autoclave. Di seguito è riportato un esempio di codice JavaScript inserito in una pagina HTML:

```
<html>
<body>

<h1>Esempio di pagina dinamica </h1>

<script type="text/javascript">
document.write("<p>" + Date() + "</p>");
</script>
<--! questa pagina visualizza la data -->
</body>
</html>
```

Nelle figure 5.1 e 5.2 sono invece riportati due esempi di pagine internet costruite per consentire il controllo della macchina. Si noti che l'impostazione della struttura delle pagine è stata realizzata mediante l'impiego dei fogli di stile (CSS) i quali permettono di ottenete quasi lo stesso effetto grafico indipendentemente dal tipo di browser utilizzato.

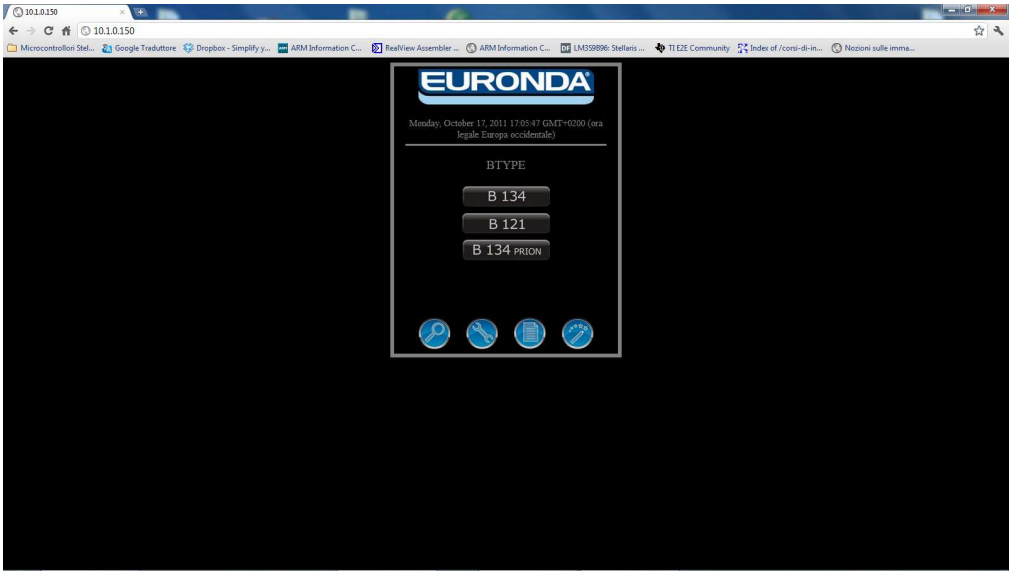


Figura 5.1: home page del webserver

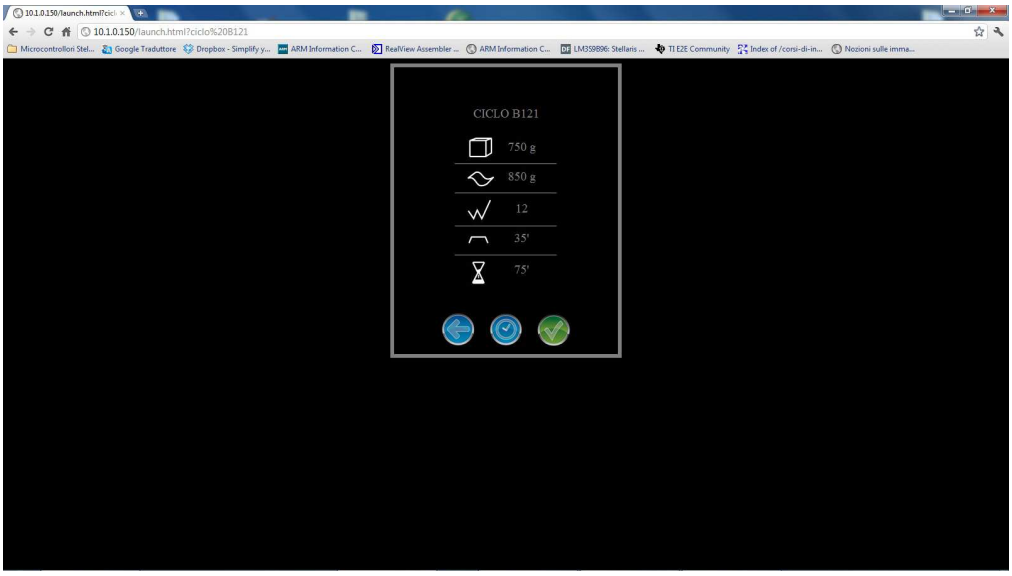


Figura 5.2: pagina di lancio del ciclo di sterilizzazione B121.

5.3 Realizzazione di pagine web dinamiche

La gestione via web dell'autoclave che si vuole realizzare richiede che alcuni valori vengano aggiornati sulle pagine web in modo automatico e senza generare fastidiosi effetti di sfarfallio. Tutto ciò richiede di risolvere il problema legato alla natura client-server del protocollo HTTP responsabile del trasferimento dei contenuti web. Come risulta chiaro dalla figura 4.11, nei sistemi di tipo client-server il dispositivo che svolge il ruolo di server effettua un trasferimento di dati solo in risposta ad una richiesta proveniente da un client. È dunque facile comprendere che il cambiamento di una variabile di stato appartenente al sistema lato server è percepito dai client solo se essi richiedono l'aggiornamento di tale valore. Il server non è quindi in grado di trasmettere volontariamente alcun tipo di informazione verso un dispositivo remoto.

Per risolvere il problema appena esposto, nell'implementazione del web server è stata utilizzata una tecnologia nota con il nome di AJAX (Asynchronous JavaScript and XML) allo scopo di generare richieste di aggiornamento asincrone del contenuto di alcuni elementi presenti nelle pagine web. Tali richieste vengono dette asincrone perché sono inoltrate in background e in momenti non relazionati al caricamento dell'intera pagina. I grandi vantaggi derivanti dall'utilizzo di AJAX sono l'elevato grado di dinamicità delle pagine nonostante un consumo molto limitato di banda. Diversi oggetti possono cambiare stato molto rapidamente utilizzando una frequenza

di aggiornamento elevata. La figura 5.3 illustra in uno schema a blocchi il funzionamento AJAX. Di seguito è riportato un esempio di come viene inoltrata una richiesta AJAX e di come il risultato ottenuto può essere elaborato.

```
var response;
var url= "/getMachineVariable.cgi";
var req= new XMLHttpRequest();
req.open ("GET",url ,true);
req.send();
req.onreadystatechange=function(){
if (req.readyState==4&&req.status==200)
{
 response=req.responseText;
}
}
```

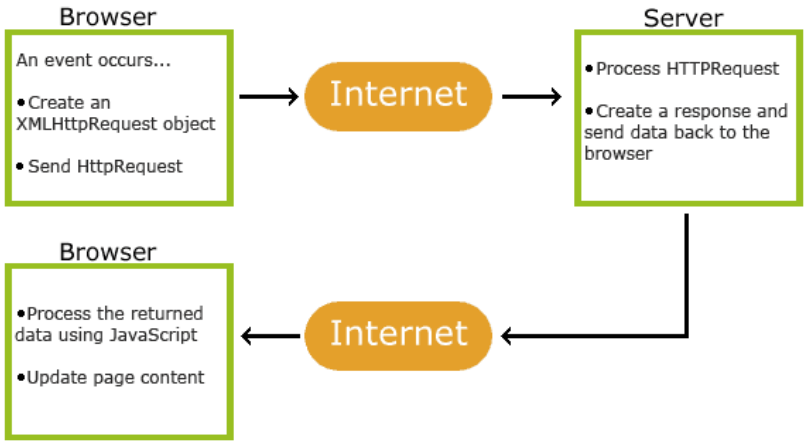


Figura 5.3: schema a blocchi del funzionamento di una richiesta AJAX

XMLHttpRequest è l'oggetto che permette lo scambio di dati tra il client e il server. I metodi open e send, applicati ad un oggetto di questo tipo, permettono di inoltrare una richiesta di tipo GET o eventualmente di tipo POST. onreadystatechange, invece, rappresenta una proprietà dell'oggetto alla quale viene assegnata la funzione responsabile di processare il risultato della richiesta. L'istruzione if (req.readyState == 4 && req.status==200) consente di processare il risultato solamente se la richiesta è andata a buon fine, cioè se è stata ricevuta una risposta corretta da parte del server. La variabile url contiene l'indirizzo della risorsa richiesta,

che in questo caso è una funzione CGI (Common Gateway Interface), la quale restituisce un messaggio HTTP contenente un testo nel campo body. Un esempio di funzione CGI implementata nel server HTTP è riportata qui di seguito:

```
getMachineVariable(HttpState* state)
{
  sprintf (_buffer, "HTTP/1.0 200 OK\r\n"\
   "Content_Type: text/html\r\n\r\n"\
   "%s",ptc1);
   cgi_sendstring(state,_buffer);
}
```

Tale funzione utilizza a sua volta la funzione cgi_sendstring(state,_buffer) per rispondere alla richiesta AJAX sopra riportata inviando in formato di testo il valore della temperatura misurata dal sensore ptc1.

5.4 Realizzazione di grafici sul web

Uno dei requisiti di progettazione di questo controllo remoto era quello di poter visualizzare in un grafico l'andamento di alcune variabili di stato dell'autoclave durante le fasi di sterilizzazione. Per realizzare ciò è stato necessario ricorrere all'utilizzo di oggetti flash, i quali permettono di sviluppare soluzioni grafiche molto potenti e accattivanti. In particolare, tra i vari oggetti disponibili si è scelto di impiegare OpenFlashChart, il quale consente di creare vari tipi di grafici. Tra i vantaggi presentati da questa libreria abbiamo che essa è disponibile in licenza LGPL, quindi il codice sorgente è modificabile in base alle esigenze dell'applicazione, e che grazie alle funzioni di interfacciamento con diversi linguaggi di programmazione consente un utilizzo e una configurazione molto flessibili. Di particolare rilevanza sono il fatto che nel caso di grafici X-Y la scala dell'asse X si dimensiona automaticamente in base al numero di elementi inseriti e che è possibile visualizzare il valore di una traccia in un determinato punto semplicemente muovendo sopra di essa il mouse. Per far fronte alle esigenze di aggiornamento dei valori del grafico con la modalità scelta per questo scopo, si è dovuto aggiungere la funzione di interfacciamento push() per poter inserire un valore alla volta tramite elaborazioni in linguaggio JavaScript. I codici HTML e JavaScript impiegati per la realizzazione dei grafici dinamici è riportata in appendice. In figura 5.4 sono invece riportati i grafici relativi all'acquisizione di quattro temperature durante l'esecuzione di un ciclo di sterilizzazione.

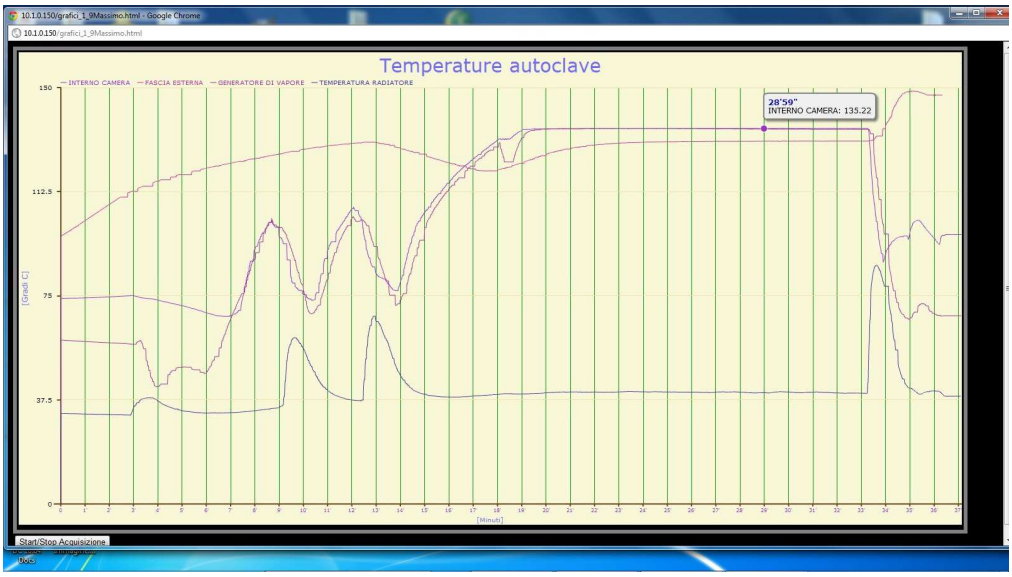


Figura 5.4: grafici relatici a 4 sensori di temperatura

Capitolo 6

Conclusioni e sviluppi futuri

Il risultato di questo lavoro di tesi è stato quello di realizzare un sistema di controllo remoto mediante l'utilizzo di pagine web. Particolare attenzione è stata rivolta al problema dell'ottenimento di pagine web dinamiche in grado di tenere costantemente aggiornati i valori corrispondenti alle variabili di stato della macchina. Per consentire un adeguato e intuitivo monitoraggio dell'andamento delle variabili di maggiore importanza durante le fasi di sterilizzazione, è stata utilizzata la libreria OpenFlashChart allo scopo di implementare grafici X-Y in grado di visualizzare contemporaneamente i dati di 4 sensori di temperatura. Un altro aspetto della progettazione che ha richiesto molta attenzione è stato quello della realizzazione dell'interfacciamento seriale con la macchina. In questa fase, sono state sfruttate le potenzialità messe a disposizione dal sistema di sviluppo al fine di ottenere un funzionamento in multitasking cooperativo e di evitare quindi che i tempi morti introdotti dalla gestione del protocollo seriale vadano a compromettere il funzionamento del server HTTP. Sono state gestite inoltre le situazioni in cui si verifichino anomalie nel collegamento seriale con la macchina, ottenendo un funzionamento molto robusto dell'interfacciamento con la macchina stessa.

Gli sviluppi futuri per questo progetto riguardano prevalentemente l'implementazione di tutte le altre funzionalità di controllo e di configurazione della macchina, le quali comunque richiedono l'utilizzo delle tecnologie già sfruttate in questa prima fase di progettazione. Inoltre è possibile pensare a molte altre funzionalità aggiuntive, come l'utilizzo dell'autenticazione DIGEST per l'accesso alle pagine web, l'aggiornamento da remoto del firmware del modulo o il download dal webserver di file contenenti informazioni sulla macchina e sui cicli di sterilizzazione effettuati.

Appendice A

Funzioni e protocollo seriale

A.1 Funzione per il calcolo del CRC

```
const int xorconst = 0xa001;
void
calcola_CRC
(char *ptr_CRC, char *ptr_data, char lunghezza)
{
 auto unsigned int ristemp;
 auto unsigned int temp;
 auto unsigned char firstbit;
 auto char i;
 auto char t;
 ristemp= 0xffff;
 for (i=0;i<lunghezza;i++)
 {
        temp=(int)*(ptr_data+i);
  temp= temp^ristemp;    //prima xor
  for (t=1; t<9; t++)
  {
   firstbit=((temp)&(0x0001));
   temp=temp>>1;
   if(firstbit==1)
   {
        temp= temp^xorconst;
   }
  }
  ristemp=temp;
```

```
  }
  sprintf(ptr_CRC,"%x",temp);
}
```

Appendice B

Funzioni per la realizzazione dei grafici

B.1 Funzione aggiunta nel file openFlashChart.as

Con le funzioni fornite dalla libreria di open-flash-chart, non è possibile aggiungere singoli valori ad una delle tracce del grafico senza andare ad aggiungere ogni volta una label sull'asse x che nel caso di questa applicazione è l'asse dei tempi. Inserendo nella libreria questa funzione, invece, se non si vuole aggiungere una label è sufficiente passare la stringa Nnl al posto delle label da aggiungere. Si noti che se il campo label viene lasciato vuoto, sull'asse delle x viene aggiunto un valore contentente una stringa vuota.

```
ExternalInterface.addCallback
("push_value", null, pushValue);

function pushValue (set:Number, val:String,
 label:String, mov:Boolean ):Void
{
 if( set<_root.chartValues.length() )
{
 _root.chartValues.styles[set].add( Number( val ), label );
 if (label!="Nnl"){
 _root._x_axis_labels.add(label);
}
 // tell the x axis where the grid lines are:
_root._x_axis.set_grid_count(_root.chartValues.length());
 if (mov==true){
```

```
    _root.move();
  }
  }
}
```

B.2 Configurazione del grafico

Esistono diverse modalità per configurare i vari parametri del grafico, una di queste è quella di inserirli nel codice Javascript che carica il file Flash sulla pagina HTML.

```
<head>
<script>
var params = {};
 params.allowscriptaccess = "always";
 params.allownetworking = "all";
 var flashvar={};
 flashvar.variables = "true";
 flashvar.x_label_style="8,#9933CC,0,60,#00A000";
 flashvar.title = "Temperature autoclave,
        {font-size: 30px;color:0x736AFF;}";
 flashvar.x_legend = "[Minuti],10,0x736AFF";
 flashvar.y_legend = "[Gradi C],12,0x736AFF";
 flashvar.y_ticks = "5,10,4";
 flashvar.line = "1,0x9933CC,PTC1,10";
 flashvar.line_2 = "1,0xCC3399,PTC2,10";
 flashvar.line_3 = "1,0xAA3399,PTC3,10";
 flashvar.line_4 = "1,0x443399,PTC4,10";
 flashvar.x_labels="0";
 flashvar.y_min = "0";
 flashvar.y_max = "100";
 flashvar.x_ticks="3";
 flashvar.tool_tip = "#x_label#<br>#key#: #val#";
 flashvar.x_axis_steps="60";
 flashvar.values="0";
 flashvar.values_2 ="0";
 flashvar.values_3 ="0";
 flashvar.values_4 ="0";
swfobject.embedSWF("open-flash-chartM2.swf",
        "myAlternativeContent", "1500px", "100%",
```

```
        "9.0.0","expressInstall.swf", flashvar, params);
</script>
</head>
<body>
<div id="myAlternativeContent">
<a href="http://www.adobe.com/go/getflashplayer">
<img src="http://www.adobe.com/images/shared/
download_buttons/get_flash_player.gif" alt="Get Adobe Flash player"
</a>
</body>
```

B.3 Funzione JavaScript per l'utilizzo dei grafici flash

Sono le funzioni che, inserite all'interno dei tag <script> si occupano di effettuare la richiesta AJAX al server per recuperare i valori dei 4 sensori di temperatura e di effettuare il push() sui grafici.

```
//inserisce un nuovo valore sul grafico.
//primo parametro: numero della traccia sul grafico
//secondo parametro: valore
//terzo parametro: label (mettere "Nnl"
//se non si vuole aggiungere una label)
//quarto parametro: boolean per aggiornamento
//istantaneo del grafico
function push(graph,value,update){
 var t_label;
 var tmp=findSWF("myAlternativeContent");
 if(graph==0){
  secondi=secondi+1;
  if (secondi>59){
        secondi=0;
        minuti++;
  }
}
 if (secondi<1){
        t_label= String(minuti)+"\'";
 }
else{
 t_label= String(minuti)+"\'"+String(secondi)+"\"";
```

```
 }
 if (graph!=0){
   t_label="Nnl";
 }
 x=tmp.push_value(graph,value,t_label,update);
}

function updateValue(){
var values = new Array();
var i;
var url= "../pages/getValues.cgi";
if (acquisizione==true){
 var req= new XMLHttpRequest();
 req.open ("GET",url ,true);
 req.send();
 req.onreadystatechange=function(){
 if (req.readyState==4 &&req.status==200)
 {
  if (req.responseText!=""){
   values = String(req.responseText).split(",");
  push(0,values[0],0);
  push(1,values[1],0);
  push(2,values[2],0);
  push(3,values[3],1);
  }
 }
 }
  }//if (acq)
}
```

Bibliografia

[1] R. S. R.Padmini and S.Nivedita, *Special Edition Using TCP/IP*. Que, 2nd ed., 2002.

[2] J. Campbell, *C Programmer's Guide to Serial Comunication*. Howard W.Sms & Company, 1988.

[3] D. Flanagan, *JavaScript: The Definitive Guide*. Apogeo, 2nd ed., 2006.

[4] *Dynamic C User's Manual*. http://ftp1.digi.com/support/documentation/019-0167_F.pdf, 2010.

[5] *Rabbit 5000 Microprocessor User's Manual*. http://ftp1.digi.com/support/documentation/019-0168_C.pdf, 2010.

[6] *Dynamic C TCP/IP User's Manual Vol.1*. http://ftp1.digi.com/support/documentation/0190143_h.pdf, 2007.

[7] *Dynamic C TCP/IP User's Manual Vol.1*. http://ftp1.digi.com/support/documentation/0190143_G.pdf, 2007.

[8] *Dynamic C Function Reference Manual*. http://ftp1.digi.com/support/documentation/90001215_A.pdf, 2007.

[9] *S29AL008D User's Manual*. http://www.spansion.com/Support/S29AL008D_00.pdf, 2005.

[10] *ICS1893 User's Manual*. http://www.datasheetcatalog.org/icst/ICS1893.pdf, 2000.

[11] G. Gobbi, *Rete Ethernet e TCP/IP*.

[12] *MODBUS over serial line Specification and implementation guide.* http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf, 2006.

[13] "Macromedia flash 5 guida di riferimento ad actionscript."

[14] *Protocollo di comunicazione E9Touch*, 2010.

Elenco delle figure