Università degli Studi di Udine

Facoltà di Scienze Matematiche, Fisiche e Naturali

Laurea Specialistica in Informatica

Titolo: ***Embedded WEB Controller per monitoraggio ambientale***

Relatore: ***Prof. Carlo Tasso***

Candidato: ***Andrea Sbrana***

Anno Accademico 2005/06

Indice

Finalità

Un embedded Webserver è un server web integrato all'interno di un sistema embedded caratterizzato da risorse di calcolo limitate ma comunque capace di gestire documenti ed applicazioni web. L'applicazione della tecnologia Web ad un sistema embedded permette la creazione di interfacce utente mediante il linguaggio HTML. I vantaggi che ne derivano permettono di ottenere un'interfaccia user friendly ed a basso costo.

Il progetto del Webserver nasce dall'esigenza di interfacciare linee di Input/Output attraverso Internet per mezzo di un browser garantendo così la multipiattaforma per i client ed un costo di accesso molto limitato.

L'utilizzatore dispone così delle pagine WEB con cui poter interrogare lo stato delle linee di ingresso sia analogiche che digitali (8 digitali e 2 analogiche) collegate al Webserver ed al tempo stesso interagire con le linee di uscita digitali (8 in tutto) e con un display LCD alfanumerico composto da due righe di 16 caratteri ciascuna.

Il livello di sicurezza per l'esecuzione dei comandi che modificano lo stato di una qualunque uscita è garantito da un sistema di autenticazione che prevede l'inserimento di uno "user" correlato ad una "password".

La realizzazione del progetto è stata prevalentemente di tipo software, ma si è resa necessaria anche la stesura di un hardware dedicato.

Il linguaggio di programmazione impiegato è il "C" ANSI, integrato in piccolissima parte da subroutine in assembler Microchip.

Per la creazione delle pagine WEB invece è stato utilizzato il linguaggio HTML.

La fase di debug sia hardware che software è stata effettuata con un emulatore real-time per microcontrollori PIC della famiglia 18 (ICE-PIC 2000).

I campi applicativi del Webserver sono in generale molteplici e diversi tra di loro. All'interno della struttura C.I.S.I.F., dove è stato realizzato il prototipo, verrà utilizzato da postazione remota per:

- monitorare la temperatura e lo stato di avanzamento di esperimenti in ambito chimico con la possibilità di variare alcuni parametri degli esperimenti stessi, come ad esempio la quantità di liquido di perfusione;
- controllare lo stato dell'impianto di illuminazione del Centro di calcolo;
- monitorare la chiusura di porte e finestre ed attivare serrature elettroniche;
- visualizzare tramite display messaggi per l'utenza.

In funzione delle applicazioni richieste, dovranno essere preparate apposite interfacce di Input/Output di tipo dedicato e preferibilmente optoisolate che esulano da questa progettazione.

La struttura del software

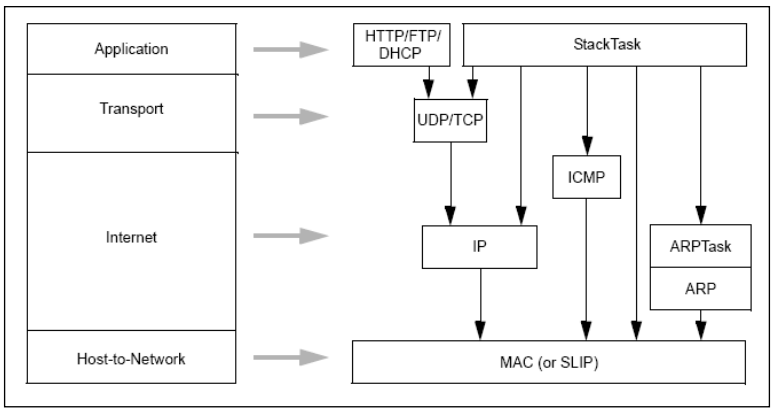
Generalità di implementazione del software

Il software di gestione dell'apparato Webserver è stato implementato con diversi moduli, ad ognuno dei quali è stata affidata una specifica funzione, in modo tale da renderne più comprensibile la lettura ed il debug. Nella tabella sottostante sono elencati i moduli impiegati. Il sistema operativo realizzato è di tipo RTOS (Real Time Operatine System) ed è stato inglobato nel modulo "Stack Manager".

Modulo	Descrizione
MAC(Ethernet)	Gestore della scheda NIC
ARP	Gestore di ARP
IP	Gestore di IP
ICMP	Gestore di ICMP
TCP	Gestore del TCP
HTTP	Gestore dell'HTTP
FTP	Gestore dell'FTP
DHCP (Client)	Client DHCP
MPFS	Gestore del File system
Stack Manager	Gestore dello stack TCP/IP
Webserver	Applicazione specifica del Webserver

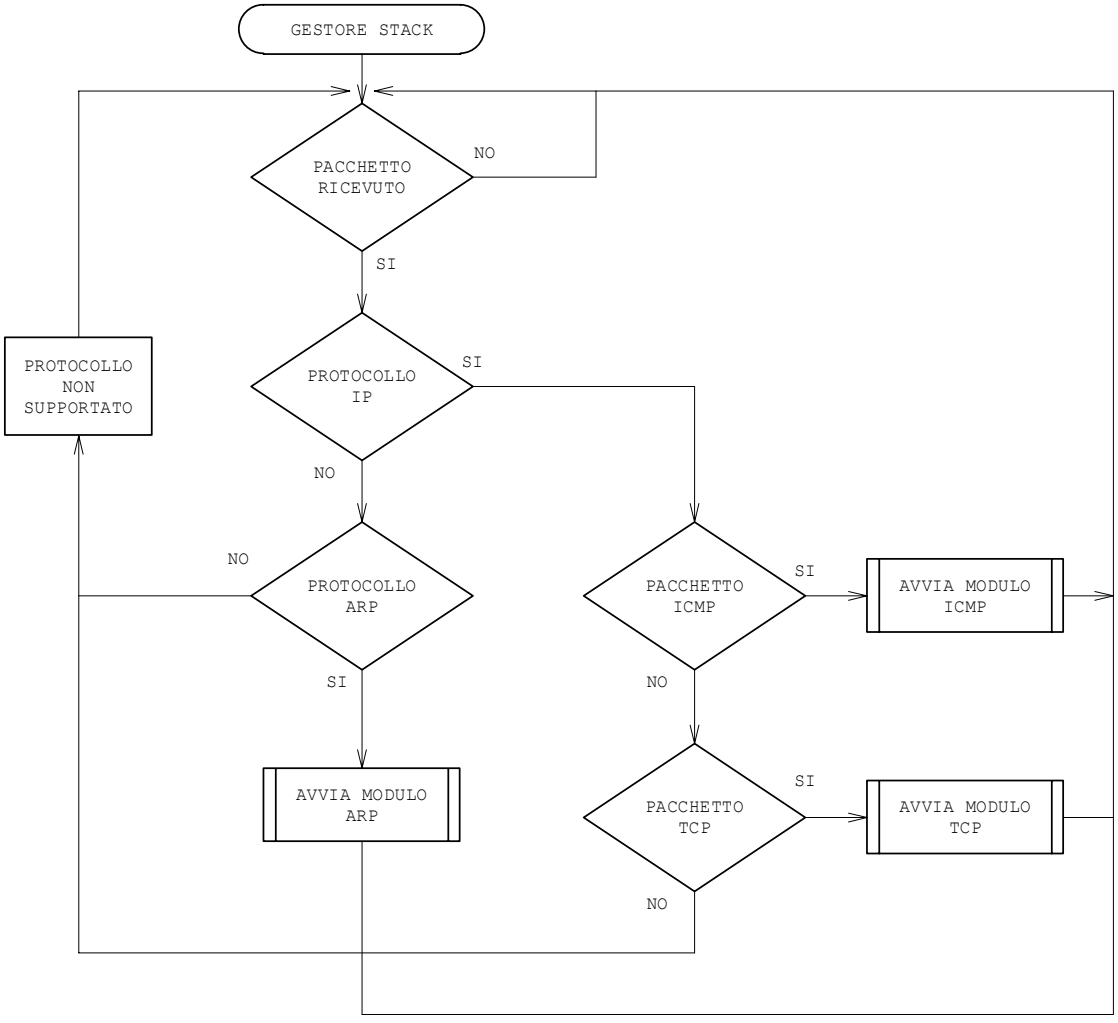
In base alla priorità delle richieste, il gestore dello stack decide quale modulo (o parte di modulo) deve essere avviato e/o concluso. Le richieste avvengono generalmente su interrupt di tipo hardware (ad esempio la periferica NIC avvisa del completamento di ricezione o trasmissione di un frame, oppure la memoria EEPROM seriale esterna segnala la fine di una fase di scrittura) ma successivamente vengono prese in considerazione anche segnalazioni di tipo software (ad esempio l'arrivo di un modulo in uno stato particolare della macchina a stati finiti con cui è stato implementato).

Nella figura sottostante, è riportato a sinistra lo schema a blocchi dei livelli OSI mentre a destra il modello applicato al Webserver. Si noti come, in particolare, sia presente il modulo Stack Task che organizza tutte le operazione dello stack TCP/IP.



Gestione dello stack

Il gestore dello stack lavora seguendo un ben preciso algoritmo che quando giunge un pacchetto va ad identificare il tipo di protocollo del pacchetto stesso e se questo è di tipo IP sale di livello per capire se il pacchetto è di tipo ICMP o TCP. Se il tipo di protocollo invece è ARP, allora viene subito avviato il modulo che gestisce tale protocollo. Nelle pagine seguenti è riportato il codice sorgente del modulo gestore dello stack.



Algoritmo di gestione dello stack

Codice sorgente del modulo gestore dello stack

```
/*********************************************************************************
 *                                    Stacktsk module
*********************************************************************************/
#define STACK_INCLUDE
#include "stacktsk.h"
#include "arptsk.h"
#include "mac.h"
#include "ip.h"

#define MAX_ICMP_DATA_LEN   (32)

typedef enum _SM_STACK
{
    SM_STACK_IDLE,
    SM_STACK_MAC,
    SM_STACK_IP,
    SM_STACK_ICMP,
    SM_STACK_ICMP_REPLY,
    SM_STACK_ARP,
    SM_STACK_TCP,
    SM_STACK_UDP
} SM_STACK;
static SM_STACK smStack;
/*****************************************
void StackInit(void)
{
    smStack = SM_STACK_IDLE;
    MACInit();
    ARPInit();
    UDPInit();
    TCPInit();
}
/*****************************************
void StackTask(void)
{
    static NODE_INFO remoteNode;
    static WORD dataCount;
    static BYTE data[MAX_ICMP_DATA_LEN];
    static WORD ICMPId;
    static WORD ICMPSeq;
    IP_ADDR tempLocalIP;

    union
    {
        BYTE MACFrameType;
        BYTE IPFrameType;
        ICMP_CODE ICMPCode;
    } type;

    BOOL lbContinue;

    lbContinue = TRUE;
    while( lbContinue )
    {
       lbContinue = FALSE;
        switch(smStack)
        {
        case SM_STACK_IDLE:
        case SM_STACK_MAC:
            if ( !MACGetHeader(&remoteNode.MACAddr, &type.MACFrameType) )
            {
                    if ( !MACIsLinked() )
                    {
                        MY_IP_BYTE1 = 0;
                        MY_IP_BYTE2 = 0;
                        MY_IP_BYTE3 = 0;
                        MY_IP_BYTE4 = 0;

                        stackFlags.bits.bInConfigMode = TRUE;
                        DHCPReset();
                    }
                break;
            }

            lbContinue = TRUE;
```

```
            if ( type.MACFrameType == MAC_IP )
                smStack = SM_STACK_IP;
            else if ( type.MACFrameType == MAC_ARP )
                smStack = SM_STACK_ARP;
            else
                MACDiscardRx();
            break;

        case SM_STACK_ARP:
            lbContinue = FALSE;
            if ( ARPProcess() )
                smStack = SM_STACK_IDLE;
            break;

        case SM_STACK_IP:
            if ( IPGetHeader(&tempLocalIP,
                             &remoteNode,
                             &type.IPFrameType,
                             &dataCount) )
            {
                lbContinue = TRUE;
                if ( type.IPFrameType == IP_PROT_ICMP )
                {
                    smStack = SM_STACK_ICMP;
                            DHCPAbort();
                        }
                    }
                }
                else if ( type.IPFrameType == IP_PROT_TCP )
                    smStack = SM_STACK_TCP;
                else if ( type.IPFrameType == IP_PROT_UDP )
                    smStack = SM_STACK_UDP;
                else
                {
                    lbContinue = FALSE;
                    MACDiscardRx();
                    smStack = SM_STACK_IDLE;
                }
            }
            else
            {
                MACDiscardRx();
                smStack = SM_STACK_IDLE;
            }
            break;

        case SM_STACK_UDP:
            if ( UDPProcess(&remoteNode, dataCount) )
                smStack = SM_STACK_IDLE;
            lbContinue = FALSE;
            break;
        case SM_STACK_TCP:
            if ( TCPProcess(&remoteNode, dataCount) )
                smStack = SM_STACK_IDLE;
            lbContinue = FALSE;
            break;
        case SM_STACK_ICMP:
            smStack = SM_STACK_IDLE;

            if ( dataCount <= (MAX_ICMP_DATA_LEN+9) )
            {
                if ( ICMPGet(&type.ICMPCode,
                             data,
                             (BYTE*)&dataCount,
                             &ICMPId,
                             &ICMPSeq) )
                {
                    if ( type.ICMPCode == ICMP_ECHO_REQUEST )
                    {
                        lbContinue = TRUE;
                        smStack = SM_STACK_ICMP_REPLY;
                    }
                    else
                    {
                        smStack = SM_STACK_IDLE;
                    }
                }
                else
```

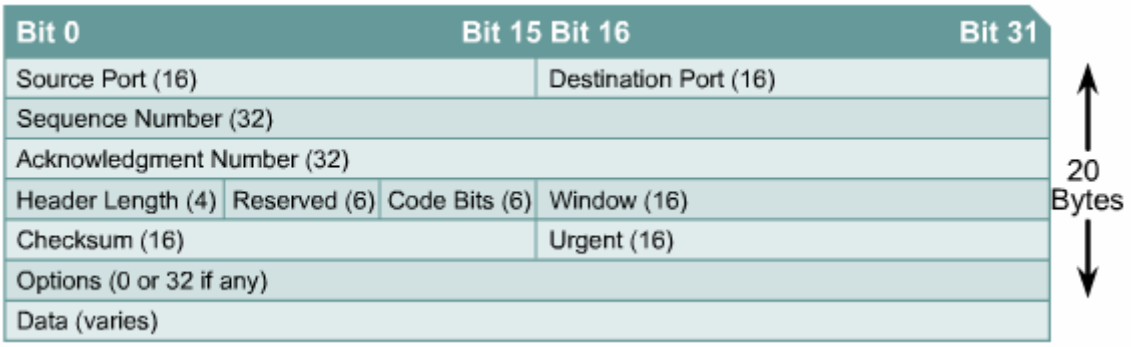
```
                {
                    smStack = SM_STACK_IDLE;
                }
            }
            MACDiscardRx();
            break;

        case SM_STACK_ICMP_REPLY:
            if ( ICMPIsTxReady() )
            {
                ICMPPut(&remoteNode,
                        ICMP_ECHO_REPLY,
                        data,
                        (BYTE)dataCount,
                        ICMPId,
                        ICMPSeq);

                smStack = SM_STACK_IDLE;
            }
            break;
        }
    }

    TCPTick();
    DHCPTask();

    if ( DHCPIsBound() )
        stackFlags.bits.bInConfigMode = FALSE;
}
```

Codice sorgente del modulo stacktsk.c

Il modulo TCP

Generalità del protocollo TCP

Il protocollo TCP ha il compito di fornire alle applicazioni un servizio affidabile per il trasferimento dei dati attraverso la rete. Questo protocollo offre un servizio orientato alla connessione (*connection-oriented*) e garantisce la consegna e l'ordinamento corretto dei dati grazie all'utilizzo di sequence number e conferme di consegna. Tra gli host impegnati nella comunicazione viene simulato un colloquio diretto attraverso un canale che consente lo scambio interattivo delle informazioni (full-duplex). I dati vengono presentati e ricevuti dal TCP ai protocolli superiori come un'unica sequenza (*byte-stream*). In questo modo è il TCP ad occuparsi di segmentarli lasciando ai protocolli superiori solo il compito di prepararli. Le informazioni contenute in un segmento sono suddivise in due parti: l'intestazione (*header*) e i dati (*data*).



Formato di un segmento TCP

L'intestazione di un pacchetto TCP è formata dai seguenti campi:

Source Port: campo di 16 bit che contiene il numero porta utilizzata dall'host mittente

Destination Port: campo di 16 bit che contiene il numero della porta utilizzata dall'host destinatario

Sequence Number: campo di 32 bit che definisce l'ordine in cui i segmenti devono essere riassemblati. E' utilizzato anche nella fase di connessione (*handshake*)

Acknowledgment Number: campo di 16 bit che contiene il prossimo numero di sequenza che l'host destinatario si aspetta di ricevere dall'host mittente. Esprime il numero di segmenti ricevuti correttamente fino a quel momento

Header Length: campo di 4 bit che definisce la lunghezza in parole a 32 bit dell'intestazione TCP. Indica dove inziano i dati

Reserved: campo di 6 bit riservato per futuri utilizzi

Code Bits: campo di 6 bit che contiene a sua volta 6 flag booleani:

- **URG** se è attivo indica che il campo Urgent Pointer è significativo e deve essere letto
- **ACK** se attivo indica che il campo Acknowledgement Number è significativo è deve essere letto
- **PSH** se attivo significa che il pacchetto deve essere inviato immediatamente, invece di attendere il riempimento del buffer
- **RST** viene utilizzato per indicare che la connessione deve essere reinizializzata, solitamente a seguito di problemi
- **SYN** viene utilizzato per stabilire una sessione, indica al destinatario di leggere il campo Sequence number e sincronizzare il proprio con esso

-FIN indica che l'host mittente non ha più dati da spedire, e vuole terminare la connessione

Window: campo di 16 bit che contiene la dimensione del buffer di dati che il mittente può accettare

Checksum: campo di 16 bit che stabilisce la correttezza delle informazioni (Intestazione + Dati)

Urgent: campo di 16 bit che indica quale porzione dati è urgente

Options: campo di dimensione varabile che contiene le opzioni per la comunicazione

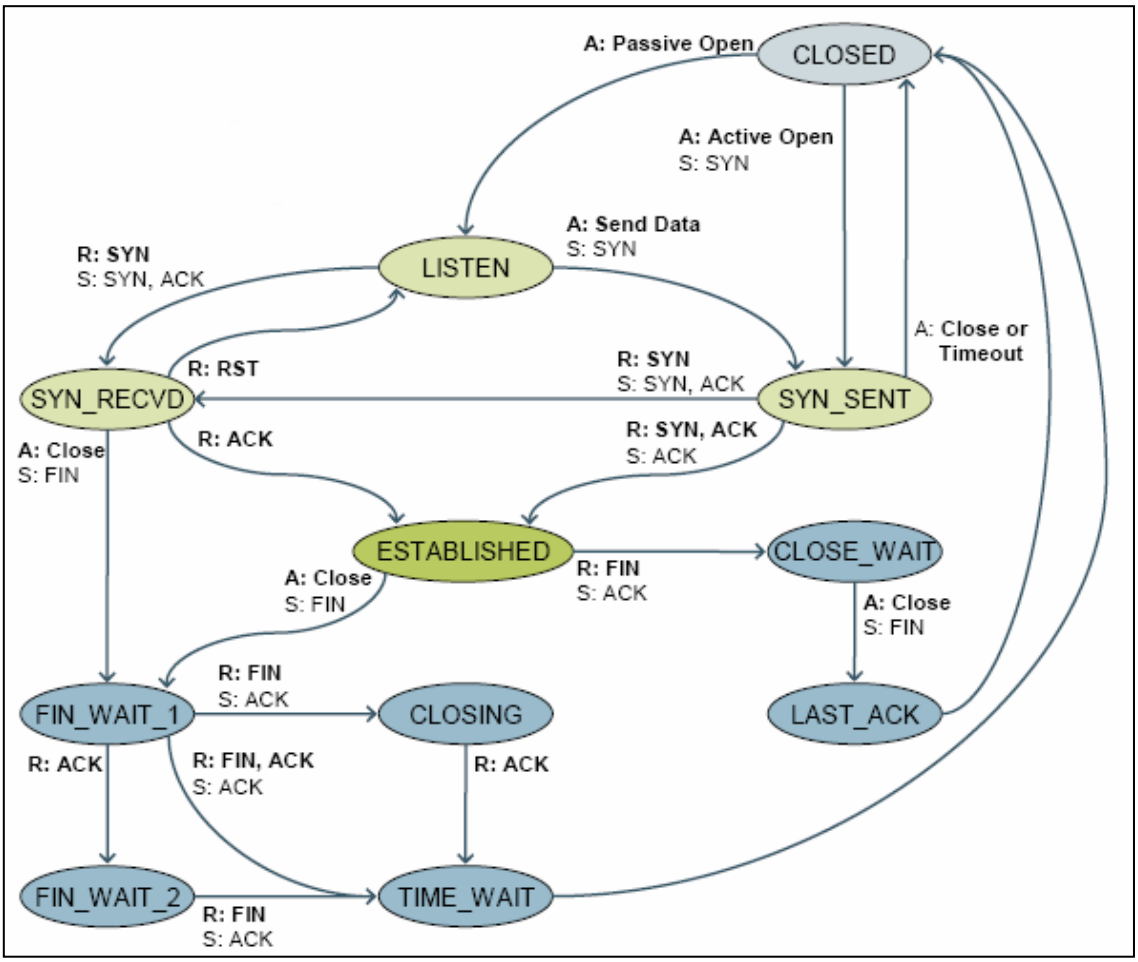
Data: i dati trasportati dal protocollo

Una sessione TCP attraversa diversi stati in seguito al verificarsi di determinati eventi:

- **LISTEN**: host in attesa di connessione
- **SYN-SENT**: host che ha inviato una richiesta di connessione ed è in attesa di risposta
- **SYN-RECEIVED**: host in attesa di conferma per la richiesta di connessione dopo aver ricevuto ed inviato una richiesta di conferma
- **ESTABLISHED**: host con una connessione aperta durante la quale i dati sono inviati e ricevuti
- **FIN-WAIT1**: host in attesa di una richiesta di termine della sessione o di conferma di richiesta di termine della connessione
- **FIN-WAIT2**: host in attesa di una richiesta di termine della sessione da parte di un host remoto
- **CLOSE-WAIT**: host in attesa di terminare la sessione

- **CLOSING**: host in attesa della conferma della richiesta di termine di connessione
- **LAST-ACK**: host in attesa della conferma dellle richiesta di termine della connessione già inviata all'host remoto
- **TIME-WAIT**: host in attesa (per un determinato lasso di tempo) per garantire che l'host remoto abbia ricevuto la conferma della richiesta di termine della connessione
- **CLOSED**: non esiste connessione tra host

Di seguito è riportato il diagramma della macchina a stati che implementa il protocollo TCP.



Macchina a stati finiti per il protocollo TCP

Nelle pagine successive viene mostrato il codice sorgente che implementa il protocollo TCP.

Codice sorgente del modulo gestore del TCP

```
/*********************************************************************************************
 *
 *                                        Modulo TCP
 *
*********************************************************************************************/
#define THIS_IS_TCP

#include <string.h>
#include "stacktsk.h"
#include "helpers.h"
#include "ip.h"
#include "mac.h"
#include "tick.h"
#include "tcp.h"

#define MAX_TCP_DATA_LEN    (MAC_TX_BUFFER_SIZE - 54)
#define TCP_START_TIMEOUT_VAL   ((TICK)TICK_SECOND * (TICK)60)
#define FIN     (0x01)              // TCP flag
#define SYN     (0x02)
#define RST     (0x04)
#define PSH     (0x08)
#define ACK     (0x10)
#define URG     (0x20)

typedef struct _TCP_HEADER
{
    WORD    SourcePort;
    WORD    DestPort;
    DWORD   SeqNumber;
    DWORD   AckNumber;

    struct
    {
        unsigned int Reserved3      : 4;
        unsigned int Val            : 4;
    } DataOffset;

    union
    {
        struct
        {
            unsigned int flagFIN    : 1;
            unsigned int flagSYN    : 1;
            unsigned int flagRST    : 1;
            unsigned int flagPSH    : 1;
            unsigned int flagACK    : 1;
            unsigned int flagURG    : 1;
            unsigned int Reserved2  : 2;
        } bits;
        BYTE byte;
    } Flags;

    WORD    Window;
    WORD    Checksum;
    WORD    UrgentPointer;

} TCP_HEADER;

#define TCP_OPTIONS_END_OF_LIST     (0x00)
#define TCP_OPTIONS_NO_OP           (0x01)
#define TCP_OPTIONS_MAX_SEG_SIZE    (0x02)
typedef struct _TCP_OPTIONS
{
    BYTE        Kind;
    BYTE        Length;
    WORD_VAL    MaxSegSize;
} TCP_OPTIONS;

#define SwapPseudoTCPHeader(h)  (h.TCPLength = swaps(h.TCPLength))

typedef struct _PSEUDO_HEADER
{
    IP_ADDR SourceAddress;
    IP_ADDR DestAddress;
    BYTE Zero;
```

```
    BYTE Protocol;
    WORD TCPLength;
} PSEUDO_HEADER;

SOCKET_INFO TCB[MAX_SOCKETS];
static WORD _NextPort;
static DWORD ISS;

static  void    HandleTCPSeg(TCP_SOCKET s,
                             NODE_INFO *remote,
                             TCP_HEADER *h,
                             WORD len);

static void TransmitTCP(NODE_INFO *remote,
                        TCP_PORT localPort,
                        TCP_PORT remotePort,
                        DWORD seq,
                        DWORD ack,
                        BYTE flags,
                        BUFFER buffer,
                        WORD len);

static  TCP_SOCKET FindMatchingSocket(TCP_HEADER *h,
                                      NODE_INFO *remote);
static  void    SwapTCPHeader(TCP_HEADER* header);
static WORD CalcTCPChecksum(WORD len);
static void CloseSocket(SOCKET_INFO* ps);

#define SendTCP(remote, localPort, remotePort, seq, ack, flags)     \
        TransmitTCP(remote, localPort, remotePort, seq, ack, flags, \
                    INVALID_BUFFER, 0)

#define LOCAL_PORT_START_NUMBER (1024)
#define LOCAL_PORT_END_NUMBER   (5000)

void TCPInit(void)
{
    TCP_SOCKET s;
    SOCKET_INFO* ps;

    for ( s = 0; s < MAX_SOCKETS; s++ )
    {
        ps = &TCB[s];
        ps->smState             = TCP_CLOSED;
        ps->Flags.bServer       = FALSE;
        ps->Flags.bIsPutReady   = TRUE;
        ps->Flags.bFirstRead    = TRUE;
        ps->Flags.bIsTxInProgress = FALSE;
        ps->Flags.bIsGetReady   = FALSE;
        ps->TxBuffer            = INVALID_BUFFER;
        ps->TimeOut             = TCP_START_TIMEOUT_VAL;
    }
    _NextPort = LOCAL_PORT_START_NUMBER;
    ISS = 0;
}

TCP_SOCKET TCPListen(TCP_PORT port)
{
    TCP_SOCKET s;
    SOCKET_INFO* ps;

    for ( s = 0; s < MAX_SOCKETS; s++ )
    {
        ps = &TCB[s];
        if ( ps->smState == TCP_CLOSED )
        {
            ps->smState             = TCP_LISTEN;
            ps->localPort           = port;
            ps->remotePort          = 0;
            ps->remote.IPAddr.Val   = 0x00;
            ps->Flags.bServer       = TRUE;
            ps->Flags.bIsGetReady   = FALSE;
            ps->TxBuffer            = INVALID_BUFFER;
            ps->Flags.bIsPutReady   = TRUE;

            return s;
        }
    }
```

```
    return INVALID_SOCKET;
}



#ifdef STACK_CLIENT_MODE
TCP_SOCKET TCPConnect(NODE_INFO *remote, TCP_PORT remotePort)
{
    TCP_SOCKET s;
    SOCKET_INFO* ps;
    BOOL lbFound;
    lbFound = FALSE;

    for ( s = 0; s < MAX_SOCKETS; s++ )
    {
        ps = &TCB[s];
        if ( ps->smState == TCP_CLOSED )
        {
            lbFound = TRUE;
            break;
        }
    }

    if ( lbFound == FALSE )
        return INVALID_SOCKET;

    ps->localPort = ++_NextPort;
    if ( _NextPort > LOCAL_PORT_END_NUMBER )
        _NextPort = LOCAL_PORT_START_NUMBER;

    ps->Flags.bServer = FALSE;
    ps->remotePort = remotePort;
    ps->SND_SEQ = ++ISS;
    ps->SND_ACK = 0;
    memcpy((BYTE*)&ps->remote, (const void*)remote, sizeof(ps->remote));
    SendTCP(&ps->remote,
            ps->localPort,
            ps->remotePort,
            ps->SND_SEQ,
            ps->SND_ACK,
            SYN);
    ps->smState = TCP_SYN_SENT;
    ps->SND_SEQ++;
    return s;
}
#endif

BOOL TCPIsConnected(TCP_SOCKET s)
    return ( TCB[s].smState == TCP_EST );

void TCPDisconnect(TCP_SOCKET s)
{
    SOCKET_INFO *ps;

    ps = &TCB[s];
    if ( ps->smState != TCP_EST )
    {
        CloseSocket(ps);
        return;
    }

    TCPDiscard(s);

    SendTCP(&ps->remote,
            ps->localPort,
            ps->remotePort,
            ps->SND_SEQ,
            ps->SND_ACK,
            FIN | ACK);

        ps->SND_SEQ++;

    ps->smState = TCP_FIN_WAIT_1;
    return;
}

BOOL TCPFlush(TCP_SOCKET s)
{
```

```
    SOCKET_INFO *ps;

    ps = &TCB[s];

    if ( ps->TxBuffer == INVALID_BUFFER )
        return FALSE;

    if ( ps->Flags.bIsPutReady == FALSE )
        return FALSE;

    TransmitTCP(&ps->remote,
                ps->localPort,
                ps->remotePort,
                ps->SND_SEQ,
                ps->SND_ACK,
                ACK,
                ps->TxBuffer,
                ps->TxCount);
    ps->SND_SEQ += (DWORD)ps->TxCount;
    ps->Flags.bIsPutReady       = FALSE;
    ps->Flags.bIsTxInProgress   = FALSE;

    return TRUE;
}

BOOL TCPIsPutReady(TCP_SOCKET s)
{
    if ( TCB[s].TxBuffer == INVALID_BUFFER )
        return IPIsTxReady();
    else
        return TCB[s].Flags.bIsPutReady;
}

BOOL TCPPut(TCP_SOCKET s, BYTE byte)
{
    WORD tempTxCount;       // This is a fix for HITECH bug
    SOCKET_INFO* ps;

    ps = &TCB[s];

    if ( ps->TxBuffer == INVALID_BUFFER )
    {
        ps->TxBuffer = MACGetTxBuffer();
        MACReserveTxBuffer(ps->TxBuffer);
        ps->TxCount = 0;
        IPSetTxBuffer(ps->TxBuffer, sizeof(TCP_HEADER));
    }

    tempTxCount = ps->TxCount;
    if ( tempTxCount >= MAX_TCP_DATA_LEN )
        return FALSE;

    ps->Flags.bIsTxInProgress = TRUE;
    MACPut(byte);
    tempTxCount++;
    ps->TxCount = tempTxCount;
    if ( tempTxCount >= MAX_TCP_DATA_LEN )
        TCPFlush(s);
    return TRUE;
}

BOOL TCPDiscard(TCP_SOCKET s)
{
    SOCKET_INFO* ps;

    ps = &TCB[s];
    if ( !ps->Flags.bIsGetReady )
        return FALSE;

    MACDiscardRx();
    ps->Flags.bIsGetReady = FALSE;
    return TRUE;
}

WORD TCPGetArray(TCP_SOCKET s, BYTE *buffer, WORD count)
{
    SOCKET_INFO *ps;
```

```
    ps = &TCB[s];

    if ( ps->Flags.bIsGetReady )
    {
        if ( ps->Flags.bFirstRead )
        {
            IPSetRxBuffer(sizeof(TCP_HEADER));
            ps->Flags.bFirstRead = FALSE;
        }
        ps->Flags.bIsTxInProgress = TRUE;
        return MACGetArray(buffer, count);
    }
    else
        return 0;
}

BOOL TCPGet(TCP_SOCKET s, BYTE *byte)
{
    SOCKET_INFO* ps;

    ps = &TCB[s];

    if ( ps->Flags.bIsGetReady )
    {
        if ( ps->Flags.bFirstRead )
        {
            IPSetRxBuffer(sizeof(TCP_HEADER));
            ps->Flags.bFirstRead = FALSE;
        }
        if ( ps->RxCount == 0 )
        {
            MACDiscardRx();
            ps->Flags.bIsGetReady = FALSE;
            return FALSE;
        }
        ps->RxCount--;
        *byte = MACGet();
        return TRUE;
    }
    return FALSE;
}

BOOL TCPIsGetReady(TCP_SOCKET s)
{
    return (TCB[s].Flags.bIsGetReady );
}

void TCPTick(void)
{
    TCP_SOCKET s;
    TICK diffTicks;
    TICK tick;
    SOCKET_INFO* ps;
    DWORD seq;
    BYTE flags;

    flags = 0x00;
    for ( s = 0; s < MAX_SOCKETS; s++ )
    {
        ps = &TCB[s];
        if ( ps->Flags.bIsGetReady || ps->Flags.bIsTxInProgress )
            continue;
        if ( (ps->smState == TCP_CLOSED) ||
             (ps->smState == TCP_LISTEN &&
              ps->Flags.bServer == TRUE) )
            continue;
        tick = TickGet();
        diffTicks = TickGetDiff(tick, ps->startTick);
        if ( diffTicks <= ps->TimeOut )
            continue;
        if ( !IPIsTxReady() )
            return;
        ps->startTick = TickGet();
        ps->TimeOut++;
        ps->RetryCount++;
        switch(ps->smState)
        {
        case TCP_SYN_SENT:
```

```
            flags = SYN;
            break;

        case TCP_SYN_RCVD:
            if ( ps->RetryCount < MAX_RETRY_COUNTS )
                flags = SYN | ACK;
            else
                CloseSocket(ps);
            break;

        case TCP_EST:
            if ( ps->RetryCount <= MAX_RETRY_COUNTS )
            {
                if ( ps->TxBuffer != INVALID_BUFFER )
                {
                    MACSetTxBuffer(ps->TxBuffer, 0);
                    MACFlush();
                }
                else
                    flags = ACK;
            }
            else
            {
                if ( ps->TxBuffer != INVALID_BUFFER )
                    MACDiscardTx(ps->TxBuffer);
                ps->TxBuffer = INVALID_BUFFER;
                flags = FIN | ACK;
                ps->smState = TCP_FIN_WAIT_1;
            }
            break;

        case TCP_FIN_WAIT_1:
        case TCP_LAST_ACK:
            if ( ps->RetryCount > MAX_RETRY_COUNTS )
                CloseSocket(ps);
            else
                flags = FIN;
            break;

        case TCP_CLOSING:
        case TCP_TIMED_WAIT:
            if ( ps->RetryCount > MAX_RETRY_COUNTS )
                CloseSocket(ps);
            else
                flags = ACK;
            break;
        }
        if ( flags > 0x00 )
        {
            if ( flags != ACK )
                seq = ps->SND_SEQ++;
            else
                seq = ps->SND_SEQ;

            SendTCP(&ps->remote,
                    ps->localPort,
                    ps->remotePort,
                    seq,
                    ps->SND_ACK,
                    flags);
        }
    }
    return;
}

BOOL TCPProcess(NODE_INFO *remote, WORD len)
{
    TCP_HEADER      TCPHeader;
    PSEUDO_HEADER   pseudoHeader;
    TCP_SOCKET      socket;
    WORD_VAL        checksum;
    BYTE            optionsSize;

    MACGetArray((BYTE*)&TCPHeader, sizeof(TCPHeader));
    SwapTCPHeader(&TCPHeader);
    pseudoHeader.SourceAddress      = remote->IPAddr;
    pseudoHeader.DestAddress.v[0]   = MY_IP_BYTE1;
    pseudoHeader.DestAddress.v[1]   = MY_IP_BYTE2;
```

```
    pseudoHeader.DestAddress.v[2]   = MY_IP_BYTE3;
    pseudoHeader.DestAddress.v[3]   = MY_IP_BYTE4;
    pseudoHeader.Zero               = 0x0;
    pseudoHeader.Protocol           = IP_PROT_TCP;
    pseudoHeader.TCPLength          = len;
    SwapPseudoTCPHeader(pseudoHeader);
    checksum.Val = ~CalcIPChecksum((BYTE*)&pseudoHeader,
                                   sizeof(pseudoHeader));
    IPSetRxBuffer(16);
    MACPut(checksum.v[0]);
    MACPut(checksum.v[1]);
    IPSetRxBuffer(0);
    checksum.Val = CalcTCPChecksum(len);

    if ( checksum.Val != TCPHeader.Checksum )
    {
        MACDiscardRx();
        return TRUE;
    }
    optionsSize = (BYTE)((TCPHeader.DataOffset.Val << 2)-
                         sizeof(TCPHeader));
    len = len - optionsSize - sizeof(TCPHeader);
    IPSetRxBuffer((TCPHeader.DataOffset.Val << 2));
    socket = FindMatchingSocket(&TCPHeader, remote);
    if ( socket < INVALID_SOCKET )
        HandleTCPSeg(socket, remote, &TCPHeader, len);
    else
    {
        MACDiscardRx();
        if ( socket == UNKNOWN_SOCKET )
        {
            TCPHeader.AckNumber += len;
            if ( TCPHeader.Flags.bits.flagSYN ||
                 TCPHeader.Flags.bits.flagFIN )
                TCPHeader.AckNumber++;

            SendTCP(remote,
                    TCPHeader.DestPort,
                    TCPHeader.SourcePort,
                    TCPHeader.AckNumber,
                    TCPHeader.SeqNumber,
                    RST);
        }
    }
    return TRUE;
}

static void TransmitTCP(NODE_INFO *remote,
                        TCP_PORT localPort,
                        TCP_PORT remotePort,
                        DWORD tseq,
                        DWORD tack,
                        BYTE flags,
                        BUFFER buffer,
                        WORD len)
{
    WORD_VAL        checkSum;
    TCP_HEADER      header;
    TCP_OPTIONS     options;
    PSEUDO_HEADER   pseudoHeader;

    while( !IPIsTxReady() );
    header.SourcePort           = localPort;
    header.DestPort             = remotePort;
    header.SeqNumber            = tseq;
    header.AckNumber            = tack;
    header.Flags.bits.Reserved2 = 0;
    header.DataOffset.Reserved3 = 0;
    header.Flags.byte           = flags;
    header.Window               = MACGetFreeRxSize();
    if ( header.Window >= MAC_RX_BUFFER_SIZE )
        header.Window = MAC_RX_BUFFER_SIZE;

    else if ( header.Window > 54 )
    {
        header.Window -= 54;
    }
    else
```

```
        header.Window = 0;
    if ( header.Window > 40 )
        header.Window -= 40;
    else
        header.Window = 0;

    header.Checksum             = 0;
    header.UrgentPointer        = 0;
    SwapTCPHeader(&header);
    len += sizeof(header);

    if ( flags & SYN )
    {
        len += sizeof(options);
        options.Kind = TCP_OPTIONS_MAX_SEG_SIZE;
        options.Length = 0x04;
        options.MaxSegSize.v[0]  = (MAC_RX_BUFFER_SIZE >> 8); // 0x05;
        options.MaxSegSize.v[1]  = (MAC_RX_BUFFER_SIZE & 0xff); // 0xb4;
        header.DataOffset.Val   = (sizeof(header) + sizeof(options)) >> 2;
    }
    else
        header.DataOffset.Val   = sizeof(header) >> 2;

    pseudoHeader.SourceAddress.v[0] = MY_IP_BYTE1;
    pseudoHeader.SourceAddress.v[1] = MY_IP_BYTE2;
    pseudoHeader.SourceAddress.v[2] = MY_IP_BYTE3;
    pseudoHeader.SourceAddress.v[3] = MY_IP_BYTE4;
    pseudoHeader.DestAddress    = remote->IPAddr;
    pseudoHeader.Zero           = 0x0;
    pseudoHeader.Protocol       = IP_PROT_TCP;
    pseudoHeader.TCPLength      = len;
    SwapPseudoTCPHeader(pseudoHeader);
    header.Checksum = ~CalcIPChecksum((BYTE*)&pseudoHeader,
                        sizeof(pseudoHeader));
    checkSum.Val = header.Checksum;
    if ( buffer == INVALID_BUFFER )
        buffer = MACGetTxBuffer();
    IPSetTxBuffer(buffer, 0);
    IPPutHeader(remote, IP_PROT_TCP, len);
    IPPutArray((BYTE*)&header, sizeof(header));
    if ( flags & SYN )
        IPPutArray((BYTE*)&options, sizeof(options));
    IPSetTxBuffer(buffer, 0);
    checkSum.Val = CalcTCPChecksum(len);
    IPSetTxBuffer(buffer, 16);
    MACPut(checkSum.v[1]);
    MACPut(checkSum.v[0]);
    MACSetTxBuffer(buffer, 0);
    MACFlush();
}

static WORD CalcTCPChecksum(WORD len)
{
    BOOL lbMSB;
    WORD_VAL checkSum;
    BYTE Checkbyte;

    lbMSB = TRUE;
    checkSum.Val = 0;

    while( len-- )
    {
        Checkbyte = MACGet();
        if ( !lbMSB )
        {
            if ( (checkSum.v[0] = Checkbyte+checkSum.v[0]) < Checkbyte)
            {
                if ( ++checkSum.v[1] == 0 )
                    checkSum.v[0]++;
            }
        }
        else
        {
            if ( (checkSum.v[1] = Checkbyte+checkSum.v[1]) < Checkbyte)
            {
                if ( ++checkSum.v[0] == 0 )
                    checkSum.v[1]++;
            }
```

```
        }

        lbMSB = !lbMSB;
    }
    checkSum.v[1] = ~checkSum.v[1];
    checkSum.v[0] = ~checkSum.v[0];
    return checkSum.Val;
}

static TCP_SOCKET FindMatchingSocket(TCP_HEADER *h, NODE_INFO *remote)
{
    SOCKET_INFO *ps;
    TCP_SOCKET s;
    TCP_SOCKET partialMatch;

    partialMatch = INVALID_SOCKET;

    for ( s = 0; s < MAX_SOCKETS; s++ )
    {
        ps = &TCB[s];
        if ( ps->smState != TCP_CLOSED )
        {
            if ( ps->localPort == h->DestPort )
            {
                if ( ps->smState == TCP_LISTEN )
                    partialMatch = s;

                if ( ps->remotePort == h->SourcePort &&
                     ps->remote.IPAddr.Val == remote->IPAddr.Val )
                        return s;
            }
        }
    }

    ps = &TCB[partialMatch];
    if ( partialMatch != INVALID_SOCKET &&
         ps->smState == TCP_LISTEN )
    {
        memcpy((void*)&ps->remote, (void*)remote, sizeof(*remote));
        ps->localPort           = h->DestPort;
        ps->remotePort          = h->SourcePort;
        ps->Flags.bIsGetReady   = FALSE;
        ps->TxBuffer            = INVALID_BUFFER;
        ps->Flags.bIsPutReady   = TRUE;
        return partialMatch;
    }
    if ( partialMatch == INVALID_SOCKET )
        return UNKNOWN_SOCKET;
    else
        return INVALID_SOCKET;
}

static void SwapTCPHeader(TCP_HEADER* header)
{
    header->SourcePort      = swaps(header->SourcePort);
    header->DestPort        = swaps(header->DestPort);
    header->SeqNumber       = swapl(header->SeqNumber);
    header->AckNumber       = swapl(header->AckNumber);
    header->Window          = swaps(header->Window);
    header->Checksum        = swaps(header->Checksum);
    header->UrgentPointer   = swaps(header->UrgentPointer);
}

static void CloseSocket(SOCKET_INFO* ps)
{
    if ( ps->TxBuffer != INVALID_BUFFER )
    {
        MACDiscardTx(ps->TxBuffer);
        ps->TxBuffer            = INVALID_BUFFER;
        ps->Flags.bIsPutReady   = TRUE;
    }
    ps->remote.IPAddr.Val = 0x00;
    ps->remotePort = 0x00;
    if ( ps->Flags.bIsGetReady )
        MACDiscardRx();
    ps->Flags.bIsGetReady       = FALSE;
    ps->TimeOut                 = TCP_START_TIMEOUT_VAL;
    ps->Flags.bIsTxInProgress   = FALSE;
```

```
    if ( ps->Flags.bServer )
        ps->smState = TCP_LISTEN;
    else
        ps->smState = TCP_CLOSED;
    return;
}

static void HandleTCPSeg(TCP_SOCKET s,
                         NODE_INFO *remote,
                         TCP_HEADER *h,
                         WORD len)
{
    DWORD ack;
    DWORD seq;
    DWORD prevAck, prevSeq;
    SOCKET_INFO *ps;
    BYTE flags;

    flags = 0x00;
    ps = &TCB[s];
    prevAck = ps->SND_ACK;
    prevSeq = ps->SND_SEQ;
    ack = h->SeqNumber ;
    ack += (DWORD)len;
    seq = ps->SND_SEQ;
    ps->RetryCount  = 0;
    ps->startTick   = TickGet();
    ps->TimeOut = TCP_START_TIMEOUT_VAL;

    if ( ps->smState == TCP_LISTEN )
    {
        MACDiscardRx();

        ps->SND_SEQ     = ++ISS;
        ps->SND_ACK     = ++ack;
        seq             = ps->SND_SEQ;
        ++ps->SND_SEQ;
        if ( h->Flags.bits.flagSYN )
        {
            memcpy((void*)&ps->remote, (const void*)remote, sizeof(*remote));
            ps->remotePort  = h->SourcePort;
            flags           = SYN | ACK;
            ps->smState     = TCP_SYN_RCVD;
        }
        else
        {
            flags               = RST;
            seq                 = ack;
            ack                 = h->SeqNumber;
            ps->remote.IPAddr.Val = 0x00;
        }
    }
    else
    {
        if ( h->Flags.bits.flagRST )
        {
            MACDiscardRx();
            CloseSocket(ps);
            return;
        }
        else if ( seq == h->AckNumber )
        {
            if ( ps->smState == TCP_SYN_RCVD )
            {
                if ( h->Flags.bits.flagACK )
                {
                    ps->SND_ACK = ack;
                    ps->RetryCount = 0;
                    ps->startTick = TickGet();
                    ps->smState = TCP_EST;

                    if ( len > 0 )
                    {
                        ps->Flags.bIsGetReady   = TRUE;
                        ps->RxCount             = len;
                        ps->Flags.bFirstRead    = TRUE;
                    }
                    else
```

```
                MACDiscardRx();
            }
            else
                MACDiscardRx();
        }
        else if ( ps->smState == TCP_SYN_SENT )
        {
            if ( h->Flags.bits.flagSYN )
            {
                ps->SND_ACK = ++ack;
                if ( h->Flags.bits.flagACK )
                {
                    flags = ACK;
                    ps->smState = TCP_EST;
                }
                else
                {
                    flags = SYN | ACK;
                    ps->smState = TCP_SYN_RCVD;
                    ps->SND_SEQ = ++seq;
                }

                if ( len > 0 )
                {
                    ps->Flags.bIsGetReady   = TRUE;
                    ps->RxCount             = len;
                    ps->Flags.bFirstRead    = TRUE;
                }
                else
                    MACDiscardRx();
            }
            else
                MACDiscardRx();
        }
        else
        {
            if ( h->SeqNumber != ps->SND_ACK )
            {
                MACDiscardRx();
                return;
            }

            ps->SND_ACK = ack;

            if ( ps->smState == TCP_EST )
            {
                if ( h->Flags.bits.flagACK )
                {
                    if ( ps->TxBuffer != INVALID_BUFFER )
                    {
                        MACDiscardTx(ps->TxBuffer);
                        ps->TxBuffer            = INVALID_BUFFER;
                        ps->Flags.bIsPutReady   = TRUE;
                    }
                }

                if ( h->Flags.bits.flagFIN )
                {
                    flags = FIN | ACK;
                    seq = ps->SND_SEQ++;
                    ack = ++ps->SND_ACK;
                    ps->smState = TCP_LAST_ACK;
                }

                if ( len > 0 )
                {
                    if ( !ps->Flags.bIsGetReady )
                    {
                        ps->Flags.bIsGetReady   = TRUE;
                        ps->RxCount             = len;
                        ps->Flags.bFirstRead    = TRUE;
                        flags = ACK;
                    }
                    else
                    {
                        flags = 0x00;
                        ps->SND_SEQ = prevSeq;
                        ps->SND_ACK = prevAck;
```

```
                        MACDiscardRx();
                    }
                }
                else
                    MACDiscardRx();
            }
            else if ( ps->smState == TCP_LAST_ACK )
            {
                MACDiscardRx();
                if ( h->Flags.bits.flagACK )
                    CloseSocket(ps);
            }
            else if ( ps->smState == TCP_FIN_WAIT_1 )
            {
                MACDiscardRx();
                if ( h->Flags.bits.flagFIN )
                {
                    flags = ACK;
                    ack = ++ps->SND_ACK;
                    if ( h->Flags.bits.flagACK )
                        CloseSocket(ps);
                    else
                        ps->smState = TCP_CLOSING;
                }
            }
            else if ( ps->smState == TCP_CLOSING )
            {
                MACDiscardRx();

                if ( h->Flags.bits.flagACK )
                    CloseSocket(ps);
            }
        }
    }
    else
        MACDiscardRx();
}

if ( flags > 0x00 )
{
    SendTCP(remote,
            h->DestPort,
            h->SourcePort,
            seq,
            ack,
            flags);
}
}
```

Codice sorgente del modulo tcp.c

Analizziamo adesso le funzioni necessarie al modulo TCP:

TCPInit

Questa funzione inizializza la macchina a stati del modulo TCP.

Sintassi

void TCPInit()

Note

Viene chiamata una sola volta al bootstrap (o reset) del sistema.

TCPListen

Abilita uno dei socket disponibili all'ascolto su una porta TCP.

Sintassi

TCP_SOCKET TCPListen(TCP_PORT *port*)

Parametri

port [in]

Numero di porta TCP su cui ascoltare

Valori di ritorno

Un identificatore di socket valido se esiste.
INVALID_SOCKET se non esiste.

TCPConnect

Inizia una richiesta di connessione su un host remoto e una porta remota.

Sintassi

TCP_SOCKET TCPConnect(NODE_INFO **remote*, TCP_PORT *port*)

Parametri

remote [in]

Host remoto che necessita di essere connesso

port [in]

Numero di porta TCP dell'host remoto

Valori di ritorno

Un identificatore di socket valido se esiste.
INVALID_SOCKET se non esiste.

TCPIsConnected

Verifica se un socket è connesso ad un host oppure no.

Sintassi

BOOL TCPIsConnected(TCP_SOCKET *socket*)

Parametri

socket [in]

Identificatore del socket da testare.

Valori di ritorno

TRUE: Se il socket è connesso all'host remoto.
FALSE: Se il socket non è connesso.

TCPDisconnect

Questa funzione richiede all'host remoto la disconnessione.

Sintassi

void TCPDisconnect(TCP_SOCKET *socket*)

Parametri

socket [in]

Identificatore del socket da sconnettere

TCPIsPutReady

Questa funzione determina quando un socket è pronto a trasmettere, ovvero quando è connesso ad un host remoto ed il suo buffer di trasmissione è vuoto.

Sintassi

BOOL TCPIsPutReady(TCP_SOCKET *socket*)

Parametri

socket [in]

Identificatore del socket da testare.

Valori di ritorno

TRUE: Se il socket is pronto a trasmettere.
FALSE: Se il socket non è connesso o il buffer non è pronto.

TCPPut

Carica un byte nel buffer di trasmissione di un socket.

Sintassi

BOOL TCPPut(TCP_SOCKET *socket*, BYTE *byte*)

Parametri

socket [in]

Identificatore del socket

byte [in]

Byte da caricare

Valori di ritorno

TRUE: Se il byte è stato caricato e c'è ancora posto nel buffer.
FALSE: Se il byte è stato caricato e non c'è più posto nel buffer.

Precondizioni

TCPIsPutReady == TRUE

Note

Quando il socket è caricato attraverso questa funzione, la trasmissione avviene solamente in funzione del numero di byte caricati, ovvero quando il buffer è stato completamente riempito. Se il buffer non viene riempito del tutto, è necessario utilizzare la funzione TCPFlush.

TCPFlush

Marca il socket come pronto per la trasmissione

Sintassi

void TCPFlush(TCP_SOCKET *socket*)

Parametri

socket [in]

Identificatore del socket che deve essere trasmesso.

TCPIsGetReady

Determina se un socket contiene dati ricevuti

Sintassi

BOOL TCPIsGetReady(TCP_SOCKET *socket*)

Parametri

socket [in]

Identificatore del socket di ricezione.

Valori di ritorno

TRUE: Se il socket contiene dati.
FALSE: Se il socket non contiene dati.

TCPGet

Recupera un byte dal buffer di un dato socket.

Sintassi

BOOL TCPGet(TCP_SOCKET *socket*, BYTE **byte*)

Parametri

socket [in]

Identificatore del socket.

byte [out]

Byte da recuperare

Valori di ritorno

TRUE: Se un byte è stato letto.

FALSE: Se non sono stati letti byte.

Precondizioni

TCPIsGetReady == TRUE

TCPGetArray

Recupera un array di dati da un socket.

Sintassi

WORD TCPGetArray(TCP_SOCKET *socket*, BYTE **byte*, WORD *count*)

Parametri

socket [in]

Identificatore del socket

byte [out]

Array di dati letto

count [out]

Numero totale di dati da leggere.

Valori di ritorno

Numero totale di dati letti.

Precondizioni

TCPIsGetReady == TRUE

TCPDiscard

Rilascia il buffer di ricezione abbinato ad un socket.

Sintassi

BOOL TCPDiscard(TCP_SOCKET *socket*)

Parametri

socket [in]

Identificatore del socket

Valori di ritorno

TRUE: Se il buffer di ricezione è stato rilasciato.

FALSE: Se il buffer di ricezione era già stato rilasciato.

Il modulo FTP

Generalità del protocollo FTP

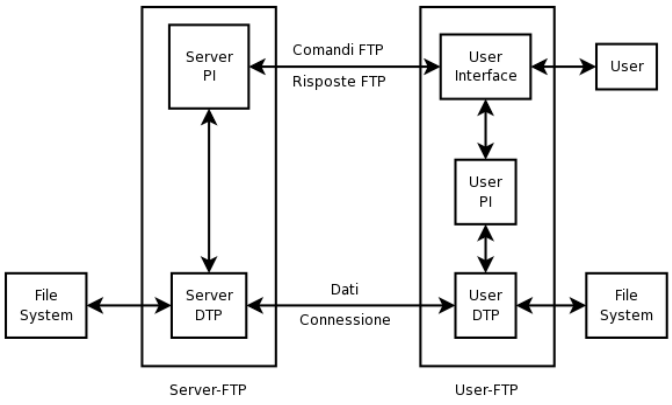
Il protocollo FTP (File Transfer Protocol) è il protocollo generalmente utilizzato per trasferire file di dati binari o di testo tra due host (upload da client a server e download da server a client) e viene descritto nella RFC 959. L'obiettivo per cui è stato sviluppato è il trasferimento affidabile ed efficiente dei dati e per questo motivo si basa sul protocollo TCP.

Un FTP Server rimane in attesa di connessioni sulla porta 21. I comandi utilizzati sono di tipo *case-insensistive,* possono essere seguiti da argomenti e sono terminati da un CRLF (Invio).

Il protocollo FTP utilizza due processi distinti per svolgere il proprio compito:

- PI (*Protocol Interpreter*) attraverso cui il client invia i comandi e riceve le risposte dal server
- DTP (*Data Transfer Process*) attraverso il quale il client ed il server si scambiano il file di dati che può essere di tipo binario oppure ASCII. Sul Webserver è stato implementato il trasferimento binario.

Il Data Transfer Process può essere di due tipi: Active MODE (*default*) o Passive MODE. Nella modalità Active Mode il client contatta il server il quale dà inizio alla connessione (sulla porta 20) per trasmettere i dati con il client. In Passive MODE è prerogativa del client dare il via alla connessione per il trasferimento dei dati. Il Webserver utilizza l'Active Mode.



Le fasi di una sessione FTP sono:

FASE 1	Il client contatta il server sulla porta 21 utilizzando il processo PI
FASE 2	Il server autentica il client (tramite user e password)
FASE 3	Trasferimento dati attraverso il DTP
FASE 4	Termine della sessione FTP (e conseguentemente TCP)

Ad ogni comando inviato, il server risponde inviando un codice che identifica la riuscita o meno dell'operazione richiesta. I codici di risposta sono numerici e tutti composti da tre caratteri **xyz**, ognuno dei quali identifica lo stato delle operazioni:

1yz	Risposta preliminare positiva. Indica che il comando è stato accettato e che si avrà un ulteriore risposta prima del comando successivo (ad esempio durante il trasferimento dei dati verrà notificato il trasferimento stesso in corso)
2yz	Comando terminato con successo
3yz	Risposta intermedia positiva. Comando eseguito correttamente e in attesa di ulteriori informazioni per completare l'operazione (ad esempio se durante l'autenticazione si passa solo lo user, poi viene richiesta la password per completare l'operazione di autenticazione)
4yz	Il comando non è stato eseguito correttamente
5yz	Comando che il server non ha potuto eseguire (ad esempio perché il comando non è stato implementato oppure è richiesta prima l'autenticazione)

Il secondo carattere specifica la natura della risposta con una maggiore granularità:

x**0**z	Informazione sulla sintassi
x**1**z	Informazioni di stato o di help
x**2**z	Informazioni sulla connessione
x**3**z	Autenticazione e accounting
x**4**z	Non ancora specificato
x**5**z	Indicazioni sul file system

Le risposte implementate nel Webserver sono le seguenti:

"220 Ready"
"331 Password required"
"230 Logged in"
"221 Bye"
"500 "
"502 Not implemented"
"530 Login required"
"150 Transferring data..."
"125 Done."
"226 Transfer Complete"
"200 ok"

Di seguito è proposto il codice sorgente del modulo ftp.c

Codice sorgente del protocollo FTP

```
/*****************************************************************************
 *
 *                                    Modulo FTP
 *
*****************************************************************************/
#define THIS_IS_FTP

#include <string.h>
#include <stdlib.h>
#include "ftp.h"
#include "tcp.h"
#include "tick.h"
#include "mpfs.h"

#define FTP_COMMAND_PORT                (21)
#define FTP_DATA_PORT                   (20)
#define FTP_TIMEOUT                     (TICK)((TICK)180 * TICK_SECOND)
#define MAX_FTP_ARGS                    (7)
#define MAX_FTP_CMD_STRING_LEN          (31)

typedef enum _SM_FTP
{
    SM_FTP_NOT_CONNECTED,
    SM_FTP_CONNECTED,
    SM_FTP_USER_NAME,
    SM_FTP_USER_PASS,
    SM_FTP_RESPOND
} SM_FTP;

typedef enum _SM_FTP_CMD
{
    SM_FTP_CMD_IDLE,
    SM_FTP_CMD_WAIT,
    SM_FTP_CMD_RECEIVE,
    SM_FTP_CMD_WAIT_FOR_DISCONNECT
} SM_FTP_CMD;

typedef enum _FTP_COMMAND
{
    FTP_CMD_USER,
    FTP_CMD_PASS,
    FTP_CMD_QUIT,
    FTP_CMD_STOR,
    FTP_CMD_PORT,
    FTP_CMD_ABORT,
    FTP_CMD_UNKNOWN,
    FTP_CMD_NONE,
} FTP_COMMAND;

ROM char *FTPCommandString[] =
{
    { "USER" },
    { "PASS" },
    { "QUIT" },
    { "STOR" },
    { "PORT" },
    { "ABOR" }
};
#define FTP_COMMAND_TABLE_SIZE  ( sizeof(FTPCommandString)/sizeof(FTPCommandString[0]) )

typedef enum _FTP_RESPONSE
{
    FTP_RESP_BANNER,
    FTP_RESP_USER_OK,
    FTP_RESP_PASS_OK,
    FTP_RESP_QUIT_OK,
    FTP_RESP_STOR_OK,
    FTP_RESP_UNKNOWN,
    FTP_RESP_LOGIN,
    FTP_RESP_DATA_OPEN,
    FTP_RESP_DATA_READY,
    FTP_RESP_DATA_CLOSE,
    FTP_RESP_OK,

    FTP_RESP_NONE
```

```
} FTP_RESPONSE;

ROM char *FTPResponseString[] =
{
    "220 Ready\r\n",
    "331 Password required\r\n",
    "230 Logged in\r\n",
    "221 Bye\r\n",
    "500 \r\n",
    "502 Not implemented\r\n",
    "530 Login required\r\n",
    "150 Transferring data...\r\n",
    "125 Done.\r\n",
    "226 Transfer Complete\r\n",
    "200 ok\r\n"
};

static union
{
    struct
    {
        unsigned int bUserSupplied : 1;
        unsigned int bLoggedIn: 1;
    } Bits;
    BYTE Val;
} FTPFlags;


static TCP_SOCKET       FTPSocket;
static TCP_SOCKET       FTPDataSocket;
static WORD_VAL         FTPDataPort;
static SM_FTP           smFTP;
static SM_FTP_CMD       smFTPCommand;
static FTP_COMMAND      FTPCommand;
static FTP_RESPONSE     FTPResponse;
static char             FTPUser[FTP_USER_NAME_LEN];
static char             FTPString[MAX_FTP_CMD_STRING_LEN+2];
static BYTE             FTPStringLen;
static char             *FTP_argv[MAX_FTP_ARGS];
static BYTE             FTP_argc;
static TICK             lastActivity;
static MPFS             FTPFileHandle;
static void ParseFTPString(void);
static FTP_COMMAND ParseFTPCommand(char *cmd);
static void ParseFTPString(void);
static BOOL ExecuteFTPCommand(FTP_COMMAND cmd);
static BOOL PutFile(void);
static BOOL Quit(void);

void FTPInit(void)
{
    FTPSocket       = TCPListen(FTP_COMMAND_PORT);
    smFTP           = SM_FTP_NOT_CONNECTED;
    FTPStringLen    = 0;
    FTPFlags.Val    = 0;
    FTPDataPort.Val = FTP_DATA_PORT;
}

BOOL FTPServer(void)
{
    BYTE v;
    TICK currentTick;

    if ( !TCPIsConnected(FTPSocket) )
    {
        FTPStringLen    = 0;
        FTPCommand      = FTP_CMD_NONE;
        smFTP           = SM_FTP_NOT_CONNECTED;
        FTPFlags.Val    = 0;
        smFTPCommand    = SM_FTP_CMD_IDLE;
        return TRUE;
    }

    if ( TCPIsGetReady(FTPSocket) )
    {
        lastActivity    = TickGet();

        while( TCPGet(FTPSocket, &v ) )
```

```
        {
            USARTPut(v);
            FTPString[FTPStringLen++]   = v;
            if ( FTPStringLen == MAX_FTP_CMD_STRING_LEN )
                FTPStringLen            = 0;
        }
        TCPDiscard(FTPSocket);

        if ( v == '\n' )
        {
            FTPString[FTPStringLen]     = '\0';
            FTPStringLen                = 0;
            ParseFTPString();
            FTPCommand                  = ParseFTPCommand(FTP_argv[0]);
        }
    }
    else if ( smFTP != SM_FTP_NOT_CONNECTED )
    {
        currentTick = TickGet();
        currentTick = TickGetDiff(currentTick, lastActivity);
        if ( currentTick >= FTP_TIMEOUT )
        {
            lastActivity                = TickGet();
            FTPCommand                  = FTP_CMD_QUIT;
            smFTP                       = SM_FTP_CONNECTED;
        }
    }

    switch(smFTP)
    {
    case SM_FTP_NOT_CONNECTED:
        FTPResponse = FTP_RESP_BANNER;
        lastActivity = TickGet();

    case SM_FTP_RESPOND:
         SM_FTP_RESPOND_Label:
        while( !TCPIsPutReady(FTPSocket) );
        if ( TCPIsPutReady(FTPSocket) )
        {
            ROM char* pMsg;
            pMsg = FTPResponseString[FTPResponse];
            while( (v = *pMsg++) )
            {
                USARTPut(v);
                TCPPut(FTPSocket, v);
            }
            TCPFlush(FTPSocket);
            FTPResponse = FTP_RESP_NONE;
            smFTP = SM_FTP_CONNECTED;
        }

    case SM_FTP_CONNECTED:
        if ( FTPCommand != FTP_CMD_NONE )
        {
            if ( ExecuteFTPCommand(FTPCommand) )
            {
                if ( FTPResponse != FTP_RESP_NONE )
                    smFTP = SM_FTP_RESPOND;
                else if ( FTPCommand == FTP_CMD_QUIT )
                    smFTP = SM_FTP_NOT_CONNECTED;

                FTPCommand = FTP_CMD_NONE;
                smFTPCommand = SM_FTP_CMD_IDLE;
            }
            else if ( FTPResponse != FTP_RESP_NONE )
            {
                smFTP = SM_FTP_RESPOND;
                goto SM_FTP_RESPOND_Label;
            }
        }
        break;
    }
    return TRUE;
}

static BOOL ExecuteFTPCommand(FTP_COMMAND cmd)
{
    switch(cmd)
```

```
    {
    case FTP_CMD_USER:
        FTPFlags.Bits.bUserSupplied = TRUE;
        FTPFlags.Bits.bLoggedIn = FALSE;
        FTPResponse = FTP_RESP_USER_OK;
        strncpy(FTPUser, FTP_argv[1], sizeof(FTPUser));
        break;

    case FTP_CMD_PASS:
        if ( !FTPFlags.Bits.bUserSupplied )
            FTPResponse = FTP_RESP_LOGIN;
        else
        {
            if ( FTPVerify(FTPUser, FTP_argv[1]) )
            {
                FTPFlags.Bits.bLoggedIn = TRUE;
                FTPResponse = FTP_RESP_PASS_OK;
            }
            else
                FTPResponse = FTP_RESP_LOGIN;
        }
        break;

    case FTP_CMD_QUIT:
        return Quit();

    case FTP_CMD_PORT:
        FTPDataPort.v[1] = (BYTE)atoi(FTP_argv[5]);
        FTPDataPort.v[0] = (BYTE)atoi(FTP_argv[6]);
        FTPResponse = FTP_RESP_OK;
        break;

    case FTP_CMD_STOR:
        return PutFile();

    case FTP_CMD_ABORT:
        FTPResponse = FTP_RESP_OK;
        if ( FTPDataSocket != INVALID_SOCKET )
            TCPDisconnect(FTPDataSocket);
        break;

    default:
        FTPResponse = FTP_RESP_UNKNOWN;
        break;
    }
    return TRUE;
}

static BOOL Quit(void)
{
    switch(smFTPCommand)
    {
    case SM_FTP_CMD_IDLE:
        if ( smFTPCommand == SM_FTP_CMD_RECEIVE )
            MPFSClose();

        if ( FTPDataSocket != INVALID_SOCKET )
        {
            MPFSClose();
            TCPDisconnect(FTPDataSocket);
            smFTPCommand = SM_FTP_CMD_WAIT;
        }
        else
            goto Quit_Done;
        break;

    case SM_FTP_CMD_WAIT:
        if ( !TCPIsConnected(FTPDataSocket) )
        {
Quit_Done:
            FTPResponse = FTP_RESP_QUIT_OK;
            smFTPCommand = SM_FTP_CMD_WAIT_FOR_DISCONNECT;
        }
        break;

    case SM_FTP_CMD_WAIT_FOR_DISCONNECT:
        if ( TCPIsPutReady(FTPSocket) )
        {
```

```
            if ( TCPIsConnected(FTPSocket) )
                TCPDisconnect(FTPSocket);
        }
        break;
    }
    return FALSE;
}

static BOOL PutFile(void)
{
    BYTE v;

    switch(smFTPCommand)
    {
    case SM_FTP_CMD_IDLE:
        if ( !FTPFlags.Bits.bLoggedIn )
        {
            FTPResponse     = FTP_RESP_LOGIN;
            return TRUE;
        }
        else
        {
            FTPResponse     = FTP_RESP_DATA_OPEN;
            FTPDataSocket   = TCPConnect(&REMOTE_HOST(FTPSocket), FTPDataPort.Val);
            smFTPCommand    = SM_FTP_CMD_WAIT;
        }
        break;

    case SM_FTP_CMD_WAIT:
        if ( TCPIsConnected(FTPDataSocket) )
        {
            if ( !MPFSIsInUse() )
            {
                FTPFileHandle   = MPFSFormat();
                smFTPCommand    = SM_FTP_CMD_RECEIVE;
            }
        }
        break;

    case SM_FTP_CMD_RECEIVE:
        if ( TCPIsGetReady(FTPDataSocket) )
        {
            lastActivity    = TickGet();
            MPFSPutBegin(FTPFileHandle);
            while( TCPGet(FTPDataSocket, &v) )
            {
                USARTPut(v);
                MPFSPut(v);
            }
            FTPFileHandle = MPFSPutEnd();
            TCPDiscard(FTPDataSocket);
        }
        else if ( !TCPIsConnected(FTPDataSocket) )
        {
            MPFSPutEnd();
            MPFSClose();
            TCPDisconnect(FTPDataSocket);
            FTPDataSocket   = INVALID_SOCKET;
            FTPResponse     = FTP_RESP_DATA_CLOSE;
            return TRUE;
        }
    }
    return FALSE;
}

static FTP_COMMAND ParseFTPCommand(char *cmd)
{
    FTP_COMMAND i;

    for ( i = 0; i < (FTP_COMMAND)FTP_COMMAND_TABLE_SIZE; i++ )
    {
    if ( !memcmppgm2ram((void*)cmd, (ROM void*)FTPCommandString[i], 4) )
            return i;
    }
    return FTP_CMD_UNKNOWN;
}

static void ParseFTPString(void)
```

```
{
    BYTE *p;
    BYTE v;
    enum { SM_FTP_PARSE_PARAM, SM_FTP_PARSE_SPACE } smParseFTP;

    smParseFTP  = SM_FTP_PARSE_PARAM;
    p           = (BYTE*)&FTPString[0];

    while( *p == ' ' )
        p++;

    FTP_argv[0]  = (char*)p;
    FTP_argc     = 1;

    while( (v = *p) )
    {
        switch(smParseFTP)
        {
        case SM_FTP_PARSE_PARAM:
            if ( v == ' ' || v == ',' )
            {
                *p = '\0';
                smParseFTP = SM_FTP_PARSE_SPACE;
            }
            else if ( v == '\r' || v == '\n' )
                *p = '\0';
            break;

        case SM_FTP_PARSE_SPACE:
            if ( v != ' ' )
            {
                FTP_argv[FTP_argc++] = (char*)p;
                smParseFTP = SM_FTP_PARSE_PARAM;
            }
            break;
        }
        p++;
    }
}
```

Codice sorgente di ftp.c

Funzioni principali del modulo gestore dell'FTP

FTPInit

Questa funzione inizializza la macchina a stati del modulo FTP.

Sintassi

void FTPInit()

FTPServer

Questa funzione implementa la macchina a stati del modulo FTP.

Sintassi

bool FTPServer()

Valori di ritorno

TRUE: Se non ci sono errori.

FALSE: Se sono presenti errori.

ExecuteFTPCommand

Questa funzione esegue un comando di tipo FTP.

Sintassi

bool FTPServer(FTP_COMMAND cmd)

Parametri

cmd

Comando da eseguire

Valori di ritorno

TRUE: Se non ci sono errori.
FALSE: Se sono presenti errori.

Note

I comandi implementati sono: USER, PASS, QUIT, STOR, PORT, ABOR.

ParseFTPString

Questa funzione esegue il parsing della stringa passata dal terminale.

Sintassi

void ParseFTPString()

ParseFTPCommand

Questa funzione esegue il parsing del comando ricevuto.

Sintassi

FTPCommand ParseFTPCommand(char *cmd)

Parametri

cmd [in]

comando.

Valori di ritorno

USER, PASS, QUIT, STOR, PORT, ABOR, UNKNOWN, NONE.

PutFile

Questa funzione implementa la macchina a stati del comando ftp PUT.

Sintassi

bool PutFile()

Valori di ritorno

TRUE: Se l'operazione si chiude senza errori.
FALSE: Se l'operazione non termina con la memorizzazione dei file.

FTPQuit

Chiude la sessione FTP.

Sintassi

bool FTPQuit()

Valori di ritorno

TRUE: Se non ci sono errori.
FALSE: Se sono presenti errori.

FTPVerify

Questa funzione viene richiamata dal server FTP quando riceve una richiesta di connessione da uno user remoto e serve per autenticarlo.

Sintassi

BOOL FTPVerify(char **login*, char **password*)

Parametri

login [in]

Stringa di caratteri che contiene lo user name

password [in]

Stringa di caratteri che contiene la password

Valori di ritorno

TRUE: Se *login e password* coincidono con quelli definiti in Webserver.c

FALSE: Se *login* o *password* non coincidono

L'FTP Server usa il valore di ritorno di questa funzione per permettere o negare l'accesso allo user FTP remoto.

Note

La lunghezza dello user è definita da FTP_USER_NAME_LEN nell'header file "ftp.h".

La massima lunghezza della password e quella totale per il comando FTP è definita da MAX_FTP_CMD_STRING_LEN in "ftp.c".

Il modulo HTTP

Generalità del protocollo HTTP

L'HTTP è un protocollo di livello 5 e funziona su un meccanismo di richiesta e risposta: il client esegue una richiesta ed il server restituisce la risposta. Nell'uso comune il client corrisponde al browser ed il server al sito web. Il protocollo HTTP prevede quindi due tipi di messaggi: di richiesta e di risposta.

A differenza del protocollo FTP, anch'esso di livello 5, nell'HTTP le connessioni vengono generalmente chiuse una volta che una particolare richiesta (o una serie di richieste correlate) è stata soddisfatta (si dice infatti che questo protocollo è stateless). Ciò rende il protocollo HTTP ideale nei siti dove le pagine molto spesso contengono dei collegamenti (*link*) a pagine ospitate da altri server ma pone problemi agli sviluppatori di contenuti web, perché costringe ad utilizzare dei metodi alternativi per conservare lo stato dell'utente (ad esempio con l'impiego di cookies).

Il messaggio di richiesta è composto di tre parti:

Linea di richiesta	Sezione Header	Body

La linea di richiesta è composta dal metodo, URI e versione del protocollo. Il metodo di richiesta può essere uno dei seguenti:

- GET
- POST
- HEADER
- PUT
- DELETE
- TRACE
- CONNECT

L'URI sta per **U**niform **R**esource **I**dentifier ed indica l'oggetto della richiesta (ad esempio la pagina web che si intende ottenere).

I metodi HTTP più comuni sono GET e POST. Il metodo GET è usato per ottenere il contenuto della risorsa indicata come URI (come può essere il contenuto di una pagina HTML). Una richiesta con metodo GET non prevede l'uso del body. Nel progetto Webserver è stato implementato esclusivamente il metodo GET per semplicità di programmazione. Il metodo POST è usato di norma per inviare informazioni al server

(ad esempio i dati di un form). In questo caso l'URI indica che cosa si sta inviando e il body ne indica il contenuto.

Il messaggio di risposta è composto dalle seguenti tre parti:

Linea di stato	Sezione Header	Body

La linea di stato riporta un codice a tre cifre catalogato nel seguento modo:

1XX - Informativo

2XX – Richiesta del client andata a buon fine

3XX – Richiesta del client ridiretta

4XX – Richiesta del client incompleta

5XX – Errore del server

Nel caso più comune il server risponde con un codice 200 (OK) e fornisce il contenuto nella sezione body. Per semplicità, sono state implementate solamente le 3 risposte obbligatoriamente necessarie per il dialogo con un browser:

"HTTP/1.0 200 OK Content-type: "

"HTTP/1.0 404 Not found"

"HTTP/1.0 503 Service Unavailable"

Il modulo HTTP implementato per il Webserver è relativamente semplice e, per questo motivo, incorpora un numero limitato di caratteristiche basilari per il funzionamento di un embedded WEB server. Di seguito vengono elencate le più significative:

- Supporta connessioni multiple HTTP
- Supporta pagine web memorizzate in EEPROM interna e/o esterna
- Supporta esclusivamente il metodo “GET” (non il “POST”)
- Supporta una CGI (Common Gateway Interface) per invocare funzioni predefinite con il browser remoto
- Supporta la creazione di pagine web dinamiche
- Supporta i seguenti formati di file:
  - TXT
  - HTML
  - GIF
  - CGI
  - JPG
  - JAVA
  - WAV

Codice sorgente del modulo gestore dell'HTTP

```
/*****************************************************************************
*                                    Modulo HTTP
*****************************************************************************/
#define THIS_IS_HTTP_SERVER
#include <string.h>
#include "stacktsk.h"
#include "http.h"
#include "mpfs.h"
#include "tcp.h"

#define HTTP_VAR_ESC_CHAR       '%'
#define HTTP_DYNAMIC_FILE_TYPE  (HTTP_CGI)
#define HTTP_TXT        (0)
#define HTTP_HTML       (1)
#define HTTP_GIF        (2)
#define HTTP_CGI        (3)
#define HTTP_JPG        (4)
#define HTTP_JAVA       (5)
#define HTTP_WAV        (6)
#define HTTP_UNKNOWN    (7)
#define FILE_EXT_LEN    (3)
typedef struct _FILE_TYPES
{
    char fileExt[FILE_EXT_LEN+1];
} FILE_TYPES;

static ROM FILE_TYPES httpFiles[] =
{
    { "TXT" },          // HTTP_TXT
    { "HTM" },          // HTTP_HTML
    { "GIF" },          // HTTP_GIF
    { "CGI" },          // HTTP_CGI
    { "JPG" },          // HTTP_JPG
    { "CLA" },          // HTTP_JAVA
    { "WAV" }           // HTTP_WAV
};
#define TOTAL_FILE_TYPES        ( sizeof(httpFiles)/sizeof(httpFiles[0]) )

typedef struct _HTTP_CONTENT
{
    ROM char typeString[20];
} HTTP_CONTENT;

static ROM HTTP_CONTENT httpContents[] =
{
    { "text/plain" },           // HTTP_TXT
    { "text/html" },            // HTTP_HTML
    { "image/gif" },            // HTTP_GIF
    { "text/html" },            // HTTP_CGI
    { "image/jpeg" },           // HTTP_JPG
    { "application/java-vm" },  // HTTP_JAVA
    { "audio/x-wave" }          // HTTP_WAV
};
#define TOTAL_HTTP_CONTENTS     ( sizeof(httpContents)/sizeof(httpConetents[0]) )

typedef enum _SM_HTTP
{
    SM_HTTP_IDLE,
    SM_HTTP_GET,
    SM_HTTP_NOT_FOUND,
    SM_HTTP_GET_READ,
    SM_HTTP_GET_PASS,
    SM_HTTP_GET_DLE,
    SM_HTTP_GET_HANDLE,
    SM_HTTP_GET_HANDLE_NEXT,
    SM_HTTP_GET_VAR,
    SM_HTTP_DISCONNECT,
    SM_HTTP_DISCONNECT_WAIT,
    SM_HTTP_HEADER,
    SM_HTTP_DISCARD
} SM_HTTP;


typedef enum _HTTP_COMMAND
```

```
{
    HTTP_GET,
    HTTP_POST,
    HTTP_NOT_SUPPORTED,
    HTTP_INVALID_COMMAND
} HTTP_COMMAND;

typedef struct _HTTP_INFO
{
    TCP_SOCKET socket;
    MPFS file;
    SM_HTTP smHTTP;
    BYTE smHTTPGet;
    WORD VarRef;
    BYTE bProcess;
    BYTE Variable;
    BYTE fileType;
} HTTP_INFO;
typedef BYTE HTTP_HANDLE;

typedef enum
{
    HTTP_NOT_FOUND,
    HTTP_OK,
    HTTP_HEADER_END,
    HTTP_NOT_AVAILABLE
} HTTP_MESSAGES;

static ROM char *HTTPMessages[] =
{
    "HTTP/1.0 404 Not found\r\n\r\nNot found.\r\n",
    "HTTP/1.0 200 OK\r\n\Content-type: ",
    "\r\n\r\n",
    "HTTP/1.0 503 \r\n\r\nService Unavailable\r\n"
};

ROM BYTE HTTP_OK_STRING[]                       =           \
                        "HTTP/1.0 200 OK\r\nContent-type: ";
#define HTTP_OK_STRING_LEN                                  \
                        (sizeof(HTTP_OK_STRING)-1)

ROM BYTE HTTP_HEADER_END_STRING[]               =           \
                        "\r\n\r\n";
#define HTTP_HEADER_END_STRING_LEN                          \
                        (sizeof(HTTP_HEADER_END_STRING)-1)

ROM BYTE HTTP_GET_STRING[]                      =           \
                        "GET";
#define HTTP_GET_STRING_LEN                                 \
                        (sizeof(HTTP_GET_STRING)-1)

ROM BYTE HTTP_DEFAULT_FILE_STRING[]             =           \
                        "INDEX.HTM";
#define HTTP_DEFAULT_FILE_STRING_LEN                        \
                        (sizeof(HTTP_DEFAULT_FILE_STRING)-1)

#define MAX_HTTP_ARGS       (20)
#define MAX_HTML_CMD_LEN    (80)
static HTTP_INFO HCB[MAX_HTTP_CONNECTIONS];
static void HTTPProcess(HTTP_HANDLE h);
static HTTP_COMMAND HTTPParse(BYTE *string,
                              BYTE** arg,
                              BYTE* argc,
                              BYTE* type);
static BOOL SendFile(HTTP_INFO* ph);

void HTTPInit(void)
{
    BYTE i;

    for ( i = 0; i <  MAX_HTTP_CONNECTIONS; i++ )
    {
        HCB[i].socket = TCPListen(HTTP_PORT);
        HCB[i].smHTTP = SM_HTTP_IDLE;
    }
}

void HTTPServer(void)
```

```
{
    BYTE conn;

    for ( conn = 0;  conn < MAX_HTTP_CONNECTIONS; conn++ )
        HTTPProcess(conn);
}

static void HTTPProcess(HTTP_HANDLE h)
{
    BYTE httpData[MAX_HTML_CMD_LEN+1];
    HTTP_COMMAND httpCommand;
    WORD httpLength;
    BOOL lbContinue;
    BYTE *arg[MAX_HTTP_ARGS];
    BYTE argc;
    BYTE i;
    HTTP_INFO* ph;
    ROM char* romString;

    ph = &HCB[h];

    lbContinue = TRUE;
    while( lbContinue )
    {
        lbContinue = FALSE;
        if ( !TCPIsConnected(ph->socket) )
        {
            ph->smHTTP = SM_HTTP_IDLE;
            break;
        }
        switch(ph->smHTTP)
        {
        case SM_HTTP_IDLE:
            if ( TCPIsGetReady(ph->socket) )
            {
                lbContinue = TRUE;
                httpLength = 0;
                while( httpLength < MAX_HTML_CMD_LEN &&
                       TCPGet(ph->socket, &httpData[httpLength++]) );
                httpData[httpLength] = '\0';
                TCPDiscard(ph->socket);
                ph->smHTTP = SM_HTTP_NOT_FOUND;
                argc = MAX_HTTP_ARGS;
                httpCommand = HTTPParse(httpData, arg, &argc, &ph->fileType);
                if ( httpCommand == HTTP_GET )
                {
                    if ( argc > 1 )
                    {
                        HTTPExecCmd(&arg[0], argc);
                        ph->fileType = HTTP_CGI;
                    }

                    ph->file = MPFSOpen(arg[0]);
                    if ( ph->file == MPFS_INVALID )
                    {
                        ph->Variable = HTTP_NOT_FOUND;
                        ph->smHTTP = SM_HTTP_NOT_FOUND;
                    }
                    else if ( ph->file == MPFS_NOT_AVAILABLE )
                    {
                        ph->Variable = HTTP_NOT_AVAILABLE;
                        ph->smHTTP = SM_HTTP_NOT_FOUND;
                    }
                    else
                        ph->smHTTP = SM_HTTP_HEADER;
                }
            }
            break;

        case SM_HTTP_NOT_FOUND:
            if ( TCPIsPutReady(ph->socket) )
            {
                romString = HTTPMessages[ph->Variable];

                while( (i = *romString++) )
                    TCPPut(ph->socket, i);

                TCPFlush(ph->socket);
```

```
                ph->smHTTP = SM_HTTP_DISCONNECT;
            }
            break;

        case SM_HTTP_HEADER:
            if ( TCPIsPutReady(ph->socket) )
            {
                lbContinue = TRUE;
                for ( i = 0; i < HTTP_OK_STRING_LEN; i++ )
                    TCPPut(ph->socket, HTTP_OK_STRING[i]);

                romString = httpContents[ph->fileType].typeString;
                while( (i = *romString++) )
                    TCPPut(ph->socket, i);

                for ( i = 0; i < HTTP_HEADER_END_STRING_LEN; i++ )
                    TCPPut(ph->socket, HTTP_HEADER_END_STRING[i]);

                if ( ph->fileType == HTTP_DYNAMIC_FILE_TYPE )
                    ph->bProcess = TRUE;
                else
                    ph->bProcess = FALSE;

                ph->smHTTPGet = SM_HTTP_GET_READ;
                ph->smHTTP = SM_HTTP_GET;
            }
            break;

        case SM_HTTP_GET:
            if ( TCPIsGetReady(ph->socket) )
                TCPDiscard(ph->socket);

            if ( SendFile(ph) )
            {
                MPFSClose();
                ph->smHTTP = SM_HTTP_DISCONNECT;
            }
            break;

        case SM_HTTP_DISCONNECT:
            if ( TCPIsConnected(ph->socket) )
            {
                if ( TCPIsPutReady(ph->socket) )
                {
                    TCPDisconnect(ph->socket);
                    ph->smHTTP = SM_HTTP_DISCONNECT_WAIT;
                }
            }
            break;
        }
    }
}

static BOOL SendFile(HTTP_INFO* ph)
{
    BOOL lbTransmit;
    BYTE c;

    MPFSGetBegin(ph->file);

    while( TCPIsPutReady(ph->socket) )
    {
        lbTransmit = FALSE;
        if ( ph->smHTTPGet != SM_HTTP_GET_VAR )
        {
            c = MPFSGet();
            if ( MPFSIsEOF() )
            {
                MPFSGetEnd();
                TCPFlush(ph->socket);
                return TRUE;
            }
        }

        if ( ph->bProcess )
        {
            switch(ph->smHTTPGet)
            {
```

```
            case SM_HTTP_GET_READ:
                if ( c == HTTP_VAR_ESC_CHAR )
                    ph->smHTTPGet = SM_HTTP_GET_DLE;
                else
                    lbTransmit = TRUE;
                break;

            case SM_HTTP_GET_DLE:
                if ( c == HTTP_VAR_ESC_CHAR )
                {
                    lbTransmit = TRUE;
                    ph->smHTTPGet = SM_HTTP_GET_READ;
                }
                else
                {
                    ph->Variable = (c - '0') << 4;
                    ph->smHTTPGet = SM_HTTP_GET_HANDLE;
                }
                break;

            case SM_HTTP_GET_HANDLE:
                ph->Variable |= (c - '0');
                ph->smHTTPGet = SM_HTTP_GET_VAR;
                ph->VarRef = HTTP_START_OF_VAR;
                break;

            case SM_HTTP_GET_VAR:
                ph->VarRef = HTTPGetVar(ph->Variable, ph->VarRef, &c);
                lbTransmit = TRUE;
                if ( ph->VarRef == HTTP_END_OF_VAR )
                    ph->smHTTPGet = SM_HTTP_GET_READ;
                break;

            default:
                while(1);
            }
            if ( lbTransmit )
                TCPPut(ph->socket, c);
        }
        else
            TCPPut(ph->socket, c);
    }

    ph->file = MPFSGetEnd();
    return FALSE;
}

static HTTP_COMMAND HTTPParse(BYTE *string,
                              BYTE** arg,
                              BYTE* argc,
                              BYTE* type)
{   BYTE i;
    BYTE smParse;
    HTTP_COMMAND cmd;
    BYTE *ext;
    BYTE c;
    ROM char* fileType;

    enum
    {   SM_PARSE_IDLE,
        SM_PARSE_ARG,
        SM_PARSE_ARG_FORMAT
    };

    smParse = SM_PARSE_IDLE;
    ext = NULL;
    i = 0;

    if ( !memcmppgm2ram(string, (ROM void*) HTTP_GET_STRING, HTTP_GET_STRING_LEN) )
    {   string += (HTTP_GET_STRING_LEN + 1);
        cmd = HTTP_GET;
    }
    else
        return HTTP_NOT_SUPPORTED;
    while( *string == ' ' )
        string++;
    c = *string;
    while ( c != ' ' &&  c != '\0' && c != '\r' && c != '\n' )
```

```
    {  if ( i >= *argc )
            break;

       switch(smParse)
       {case SM_PARSE_IDLE:
            arg[i] = string;
            c = *string;
            if ( c == '/' || c == '\\' )
                smParse = SM_PARSE_ARG;
            break;

        case SM_PARSE_ARG:
            arg[i++] = string;
            smParse = SM_PARSE_ARG_FORMAT;

        case SM_PARSE_ARG_FORMAT:
            c = *string;
            if ( c == '?' || c == '&' )
            {
                *string = '\0';
                smParse = SM_PARSE_ARG;
            }
            else
            {
                if ( c == '+' )
                    *string = ' ';
                else if ( c == '.' && i == 1 )
                    ext = ++string;
                else if ( c == '=' )
                {
                    *string = '\0';
                    smParse = SM_PARSE_ARG;
                }

                else if ( c == '/' || c == '\\' )
                    arg[i-1] = string+1;
            }
            break;
        }
        string++;
        c = *string;
    }
    *string = '\0';

    *type = HTTP_UNKNOWN;
    if ( ext != NULL )
    {
        ext = (BYTE*)strupr((char*)ext);

        fileType = httpFiles[0].fileExt;
        for ( c = 0; c < TOTAL_FILE_TYPES; c++ )
        {
            if ( !memcmppgm2ram((void*)ext, (ROM void*)fileType, FILE_EXT_LEN) )
            {
                *type = c;
                break;
            }
            fileType += sizeof(FILE_TYPES);
        }
    }

    if ( i == 0 )
    {
        memcpypgm2ram(arg[0], (ROM void*)HTTP_DEFAULT_FILE_STRING,
                                    HTTP_DEFAULT_FILE_STRING_LEN);
        arg[0][HTTP_DEFAULT_FILE_STRING_LEN] = '\0';
        *type = HTTP_HTML;
        i++;
    }
    *argc = i;
    return cmd;
}
```

Codice sorgente del file http.c

Funzioni principali del modulo gestore dell'HTTP

HTTPInit

Questa funzione inizializza la macchina a stati del modulo HTTP.

Sintassi

void HTTPInit()

HTTPServer

Questa funzione attiva il server http per massimo MAX_HTTP_CONNECTIONS connessioni.

Sintassi

void HTTPServer()

Note

Questa funzione viene avviata al bootstrap e deve essere preceduta dalla HTTPInit.

HTTPProcess

È la funzione che implementa la macchina a stati del modulo HTTP.

Sintassi

void HTTPProcess(HTTP_HANDLE h)

Parametri

h [in]

Identificatore della connessione HTTP.

Note

Per ogni connessione http possibile, viene avviata questa funzione in modo indipendente. Deve essere chiamata dopo la HTTPServer.

HTTPGetVar

Questa funzione è richiamata dal modulo server http quando il parser trova una stringa del tipo '%xx' in una pagina CGI.

Sintassi

WORD HTTPGetVar(BYTE *var*, WORD *ref*, BYTE **val*)

Parametri

var [in]

Variabile di ingresso da convertire

ref [in]

Il modulo HTTP sfrutta il valore di ritorno di HTTPGetVar per determinare se chiamare di nuovo questa funzione per ulteriori dati: poiché viene restituito solo un byte alla volta, il valore di *ref* permette all'applicazione principale di tener traccia del trasferimento dei dati essendo impiegato come indice per l'array di dati da restituire. Il valore che indica la fine del trasferimento è HTTP_END_OF_VAR.

val [out]

Carattere di ritorno

Valori di ritorno

HTTP_START_OF_VAR
Carattere
HTTP_END_OF_VAR

Note

Poiché il valore di ritorno è di 16 bit, si potranno trasferire fino a 64 Kbytes di dati con una singola variabile.
Per trasferirne di più si possono inserire due o più variabili consecutive.

HTTPExecCmd

Questa funzione viene chiamata quando il server http riceve un metodo GET con più di un parametro. Decodifica il codice del metodo ed intraprende l'azione correlata come rintracciare e pubblicare la pagina di risposta e/o eseguire operazioni di input/output.

Sintassi

void HTTPExecCmd(BYTE ***argv*, BYTE *argc*)

Parametri

argv [in]

Lista degli argomenti stringa. La prima stringa (*argv*[0]) rappresenta l'azione del form, mentre le restanti (*argv*[1..n]) sono parametri di comando.

argc [in]

Numero totale dei parametri compresa l'azione del form.

Valori di ritorno

Eventualmente può essere passato il riferimento alla pagina di risposta da pubblicare

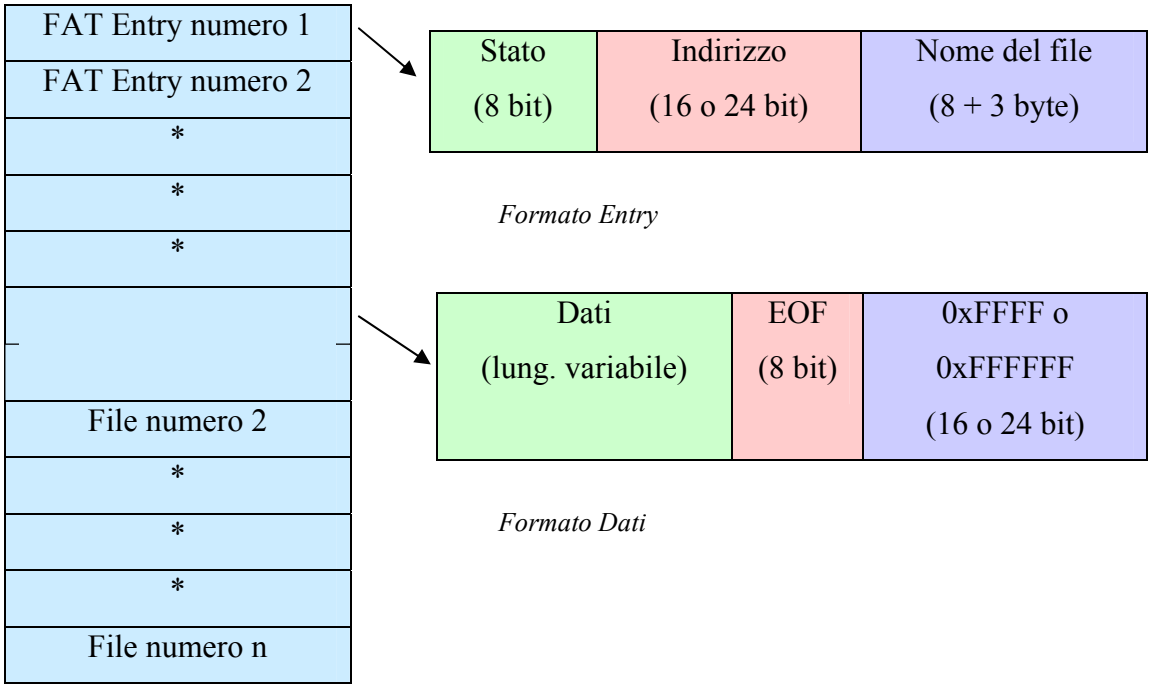
Note

Il numero di argomenti e la lunghezza totale della stringa passata dal browser sono definiti da MAX_HTTP_ARGS e MAX_HTML_CMD_LEN in "http.c".

Il modulo del FILE SYSTEM

Generalità del file system implementato

La memorizzazione delle pagine WEB sul supporto dedicato (memoria EEPROM di tipo seriale) deve essere eseguita in modo tale da poter recuperare velocemente le informazioni corrette al momento in cui servono, ovvero durante la creazione delle pagine dinamiche e della successiva pubblicazione.
È stato implementato quindi un "mini" file system in un formato molto semplice ma robusto il cui formato è illustrato nelle tabelle che seguono.



Formato della FAT

Formato Entry

Formato Dati

Il file system è composto da una FAT che contiene le informazioni sullo stato del file (in uso, cancellato, ultimo), dall'indirizzo (indirizzo fisico della cella di memoria da cui inizia il file) e dal nome del file (in notazione "short" cioè massimo 8 caratteri più 3 di estensione).
Il file viene inserito a partire quindi dalla cella di indirizzo XXXX (16 o 24 bit a seconda della capienza della memoria impiegata), è seguito da un carattere di EOF (End Of File) e dalla sequenza 0xFFFF (0xFFFFFF per l'indirizzo a 24 bit).

Se all'interno del file è presente un carattere EOF, questo è sostituito con un carattere "stuff" detto DLE (Data link Escape).
Questo tipo di file system è stato implementato per ottenere la massima compressione (intesa non come riduzione di occupazione del file ma come ottimizzazione degli spazi di memoria) possibile in supporti di memoria di bassa capacità: non sono presenti quindi tutte quelle caratteristiche tipiche di un file system che invece viene caricato su supporti di memoria di più grande capacità come ad esempio gli hard disk.
Allo stesso tempo anche la gestione del file system da parte del sistema operativo è risultata molto semplice e non si è previsto di poter cancellare, modificare o aggiungere file dopo che l'immagine del file system è stata creata. Ciò implica che per eseguire una qualsiasi modifica si deve necessariamente ricaricare in memoria l'immagine completa del file system attraverso il protocollo ftp.
Per la creazione dell'immagine del file system, si è impiegato un eseguibile fornito con il kit di valutazione di Microchip che legge tutti i file contenuti in una directory specificata nel path passato su riga di comando, li "comprime" togliendo tutti gli spazi ed i caratteri inutili compresi i carriage-return ed i line-feed, ne calcola la lunghezza in byte e scrive per ciascuno una FAT Entry. Al termine di tutte le FAT Entry, vengono scritti i file "alleggeriti". Il formato del file di uscita è di tipo binario.
Il tipo di supporto scelto per contenere il file system è di tipo EEPROM seriale con capacità di 256Kbit oppure 512Kbit. Questo tipo di supporto implica un certo ritardo nella fase di scrittura, sia per l'invio di dati in forma seriale (nel Webserver si dialoga con questa periferica a 400Khz), sia per il tempo necessario alla memorizzazione dei dati su celle EEPROM: il tempo per memorizzare una "pagina" di 64 byte è di 5ms e quindi per 256Kbit si ha un tempo richiesto di circa 500*5=2.500mS oltre al tempo necessario al trasferimento dei dati.
Poiché durante il normale impiego del Webserver le pagine WEB verranno prevalentemente "lette" dal supporto seriale (eliminando così il tempo necessario alla memorizzazione), non si è ritenuto idoneo utilizzare supporti con interfaccia parallela in cui per la lettura di un dato si deve comunque sempre passare prima un indirizzo mentre nel caso della periferica seriale è sufficiente inviare l'indirizzo della prima cella per ricevere il contenuto delle celle successive in modo automatico.

Codice sorgente del modulo gestore del file system

```
/*****************************************************************************
 *                                  Modulo MPFS
*****************************************************************************/
#define THIS_IS_MPFS

#include <string.h>
#include <stdlib.h>
#include "mpfs.h"
#include "xeeprom.h"

#define MAX_FILE_NAME_LEN   (12)
#define MPFS_DATA           (0x00)
#define MPFS_DELETED        (0x01)
#define MPFS_DLE            (0x03)
#define MPFS_ETX            (0x04)

typedef struct  _MPFS_ENTRY
{
    BYTE Flag;

    MPFS Address;
    BYTE Name[MAX_FILE_NAME_LEN];           // 8 + '.' + 3
} MPFS_ENTRY;

static union
{
    struct
    {
        unsigned int bNotAvailable : 1;
    } bits;
    BYTE Val;
} mpfsFlags;

BYTE mpfsOpenCount;

#define MPFS_Start     MPFS_RESERVE_BLOCK

MPFS _currentHandle;
MPFS _currentFile;
BYTE _currentCount;

BOOL MPFSInit(void)
{
    mpfsOpenCount = 0;
    mpfsFlags.Val = 0;
    XEEInit(EE_BAUD(CLOCK_FREQ, 400000));
    return TRUE;
}

MPFS MPFSOpen(BYTE* file)
{
    MPFS_ENTRY entry;
    MPFS FAT;
    BYTE fileNameLen;

    if ( mpfsFlags.bits.bNotAvailable )
        return MPFS_NOT_AVAILABLE;

    if ( *file == '\0' )
        return MPFS_INVALID;

    file = (BYTE*)strupr((char*)file);

    while(1)
    {
        XEEReadArray(EEPROM_CONTROL, FAT, (unsigned char*)&entry, sizeof(entry));

        if ( entry.Flag == MPFS_DATA )
        {
            fileNameLen = strlen((char*)file);
            if ( fileNameLen > MAX_FILE_NAME_LEN )
                fileNameLen = MAX_FILE_NAME_LEN;

            if( !strncmp((char*)file, (char*)entry.Name, fileNameLen) )
```

```
            {
                _currentFile = (MPFS)entry.Address;
                mpfsOpenCount++;
                return entry.Address;
            }
            FAT += sizeof(entry);
        }
        else if ( entry.Flag == MPFS_ETX )
        {
            if ( entry.Address != (MPFS)MPFS_INVALID )
                FAT = (MPFS)entry.Address;
            else
                break;
        }
    else
        return (MPFS)MPFS_INVALID;
    }
    return (MPFS)MPFS_INVALID;
}

void MPFSClose(void)
{
    _currentCount = 0;
    mpfsFlags.bits.bNotAvailable = FALSE;
    if ( mpfsOpenCount )
        mpfsOpenCount--;
}

BOOL MPFSGetBegin(MPFS handle)
{
    _currentHandle = handle;
    return (XEEBeginRead(EEPROM_CONTROL, handle) == XEE_SUCCESS);
}

BYTE MPFSGet(void)
{
    BYTE t;

    t = XEERead();
    _currentHandle++;

    if ( t == MPFS_DLE )
    {
        t = XEERead();
        _currentHandle++;
    }
    else if ( t == MPFS_ETX )
        _currentHandle = MPFS_INVALID;

    return t;
}

MPFS MPFSGetEnd(void)
{
    XEEEndRead();
    return _currentHandle;
}

MPFS MPFSFormat(void)
{
    mpfsFlags.bits.bNotAvailable = TRUE;
    return (MPFS)MPFS_RESERVE_BLOCK;
}

BOOL MPFSPutBegin(MPFS handle)
{
    _currentHandle = handle;
    _currentCount = (BYTE)handle;
    _currentCount &= (MPFS_WRITE_PAGE_SIZE-1);
    return (XEEBeginWrite(EEPROM_CONTROL, handle) == XEE_SUCCESS);
}

BOOL MPFSPut(BYTE b)
{
    if ( XEEWrite(b) )
        return FALSE;

    _currentCount++;
```

```
    _currentHandle++;
    if ( _currentCount >= MPFS_WRITE_PAGE_SIZE )
    {
        MPFSPutEnd();
        XEEBeginWrite(EEPROM_CONTROL, _currentHandle);
    }
    return TRUE;
}

MPFS MPFSPutEnd(void)
{
    _currentCount = 0;
    XEEEndWrite();
    while( XEEIsBusy(EEPROM_CONTROL) );

    return _currentHandle;
}

MPFS MPFSSeek(MPFS_OFFSET offset)
{
    WORD i;

    MPFSGetBegin(_currentFile);

    i = 0;
    while(i++ != offset)
        MPFSGet();

    MPFSGetEnd();

    return _currentHandle;
}
```

Codice sorgente del modulo mpfs.c

Funzioni principali del modulo gestore del file system

MPFSInit

Questa funzione inizializza la macchina a stati del modulo del file system.

Sintassi

BOOL MPFSInit()

Note

Viene chiamata una sola volta al bootstrap (o reset) del sistema.

Valori di ritorno

TRUE: Se viene trovato il file system in EEPROM.
FALSE: Se non viene trovato il file system.

MPFSOpen(BYTE* file)

Questa funzione apre il file indicato.

Sintassi

MPFS MPFSOpen(BYTE* file)

Valori di ritorno

Viene restituito l'identificatore del file aperto.

MPFSClose(void)

Questa funzione chiude il file attualmente aperto.

Sintassi

void MPFSClose(void)

Note

Viene chiamata per ogni file aperto in precedenza.

MPFSGet(void)

Questa funzione restituisce un byte dal file attualmente aperto.

Sintassi

BYTE MPFSGet(void)

Valori di ritorno

Viene restituito il byte all'indirizzo corrente del file aperto.

MPFSPut(BYTE b)

Questa funzione scrive un byte sul file attualmente aperto in EEPROM.

Sintassi

BOOL MPFSPut(BYTE b)

Parametri

b [in]

Byte da memorizzare

Valori di ritorno

TRUE: Se il byte viene memorizzato correttamente in EEPROM.

FALSE: Se il byte non viene memorizzato correttamente in EEPROM.

Il modulo di gestione dell'interfaccia WEB

Generalità del modulo di interfaccia

Il modulo che coordina tutti gli altri e che ha all'interno la procedura "main" è il Webserver. Nella fase di bootstrap del sistema, vengono caricati tutti i moduli visti precedentemente e successivamente vengono inizializzati con le opportune variabili e poi avviati.

Contestualmente sono inizializzate tutte le periferiche hardware connesse al microcontrollore come il display LCD, la memoria EEPROM seriale esterna, l'interfaccia NIC, i buffer dei canali digitali di input e di output.

Creazione delle pagine WEB dinamiche

La parte più interessante e significativa di questo modulo è la creazione dinamica delle pagine HTML e conseguentemente lo scambio di dati con le periferiche relative.

Durante la creazione della singola pagina web in cui il Webserver deve pubblicare dei dati (ad esempio lo stato dei canali digitali di ingresso) viene eseguito un parsing sul codice HTML passato dal programmatore e visibile nella descrizione delle pagine Web e vengono cercate stringhe di tipo "%xx" dove xx può variare tra 00 e 99 mentre il carattere "%" ha la funzione di codice di controllo (detto anche carattere di "escape"). Quando il parser trova tale carattere all'interno della stringa di testo, lo rimuove e chiama la funzione "HTTPGetVar che sostituirà ai caratteri xx il valore attuale della variabile corrispondente. Per visualizzare il carattere "%" è necessario ripeterlo per due volte (ad esempio per visualizzare 38% in una pagina, si scrive "38%%").

```
Codice della pagina html che visualizza lo stato dell'ingresso digitale 1
                .
                .
                <td><img src=LED%00.gif></td>
                <td>Ingresso 1</td>
                .
                .
-------------------------------------------------------------------------------
Codice della funzione HTTPGetVar che restituisce il valore "0" o "1" a seconda
dell'ingresso digitale 1

WORD HTTPGetVar(BYTE var, WORD ref, BYTE* val)
{
    switch(var)
    {
        case INP I1:
         if ( DIGITAL INPUT&0b00000001)
             *val = '1';
         else
             *val = '0';
         break;
                .
}
```

Esempio di sostituzione di una variabile ad una cifra

Nell'esempio di codice riportato, si vede che nella pagina HTML si pubblica un'immagine che

ha nome "LED%00.gif". Quando la funzione HTTPGetVar riportata di seguito riceve la chiamata, in "var" viene inserito il valore "00" che segue il carattere "%". Viene eseguito uno "switch – case" per identificare la variabile e, dopo aver analizzato lo stato dell'ingresso numero 1, restituisce il byte di valore zero oppure uno. Tale cifra viene quindi sostituita al posto di "%00" e così la stringa di chiamata dell'immagine viene ad essere modificata dinamicamente in "<img src=LED0.gif>" oppure "<img src=LED1.gif>" a seconda dello stato dell'ingresso numero 1. A questo punto è sufficiente avere due immagini diverse e la rappresentazione della variabile è completata.

Nel caso in cui invece di un'immagine si debba andare a pubblicare una stringa (ad esempio il risultato della conversione di un canale analogico composto da più cifre) si adotta una tecnica di chiamate ripetute come si vede nell'esempio successivo.

```
Codice della pagina html che visualizza il valore dell'ingresso analogico 1
                .
                .
                .
                .
                <td>Ingresso 1</td>

                <td><%16></td>
                .
                .
                .
                .
-------------------------------------------------------------------------------------
Codice della funzione HTTPGetVar che restituisce una stringa con il valore dell'ingresso
analogico 1

WORD HTTPGetVar(BYTE var, WORD ref, BYTE* val)
{
    switch(var)
    {
       case INP_A1:

                if ( ref == HTTP START OF VAR )
                       ref = (BYTE)0;

                *val = AN0String[(BYTE)ref];

                if ( AN0String[(BYTE)ref] == '\0' )
                       return HTTP END OF VAR;

                (BYTE)ref++;
                return ref;
                break;
                .
                .
                .
                .
}
```

Esempio di sostituzione di una variabile a più cifre

La seconda funzione richiesta al Webserver è quella di modificare lo stato di canali di uscita digitali oppure la visualizzazione sul display LCD in base a quanto inviato dal browser. La funzione che implementa questa operatività è la HTTPExecCmd.
Nel riquadro seguente è riportato l'esempio di scrittura sul display LCD:

```
Codice della pagina html che invia dati al display
                        .
                        .
                        .
        <tr>
                <td><input type="text" name="D1" size="20" maxlength=16></td>
        </tr>
        <tr>
                <td><input type="text" name="D2" size="20" maxlength=16></td>
        </tr>
                        .
                        .
                        .

-------------------------------------------------------------------------------
Codice della funzione che scrive sul display

void HTTPExecCmd(BYTE** argv, BYTE argc)
{
    BYTE command;
    BYTE var;

    command = argv[0][0] - '0';

    switch(command)
    {
        case CGI CMD LCDOUT:
                if (SuperUser == TRUE)
                {
                        XLCDGoto(0, 0);
                        XLCDPutROMString(blankLCDLine);
                        XLCDGoto(1, 0);
                        XLCDPutROMString(blankLCDLine);
                        XLCDGoto(0, 0);
                        XLCDPutString(argv[2]);
                        XLCDGoto(1, 0);
                        XLCDPutString(argv[4]);
                }
                        .
                        .
                        .
        }
}
```

Esempio di scrittura sul display

I caratteri da inviare al display, vengono passati al Webserver per mezzo del metodo "GET". La funzione HTTPExecCmd li riceve già analizzati e divisi dal parser e pronti per essere manipolati. Per la fase di scrittura vera e propria, si ricorre a funzioni predisposte per il particolare hardware come la XLCDGoto, la XLCDPutROMString e la XLCDPutString.
In modo analogo, viene modificato lo stato dei singoli canali di uscita:

```
                                .
                                .
                                .
                <td><input type=submit name=0 value="<>"></td>
                <td><img src=LED%08.gif></td>
                <td>Uscita 1</td>
                                .
                                .
                                .
                <td><input type=submit name=7 value="<>"></td>
                <td><img src=LED%15.gif></td>
                <td>Uscita 8</td>
                                .
                                .
                                .
Codice della funzione che modifica lo stato dei canali digitali di uscita

 void HTTPExecCmd(BYTE** argv, BYTE argc)
{
    BYTE command;
    BYTE var;

    command = argv[0][0] - '0';

    case CGI_CMD_DIGOUT:
        var = argv[1][0] - '0';
               if (SuperUser == TRUE)
                switch(var)
               {
                       case VAR_L1:
                              DIGITAL_OUTPUT ^= 0b00000001;
                              ToggleOut();
                              break;
                       case VAR L2:
                              DIGITAL OUTPUT ^= 0b00000010;
                              ToggleOut();
                              break;
                                      .
                                      .
                                      .

                       case VAR_L8:
                              DIGITAL_OUTPUT ^= 0b10000000;
                              ToggleOut();
                              break;
```

Esempio di modifica dello stato dei canali digitali di uscita

La funzione che si occupa di invertire o "togglare" lo stato dei canali di uscita del Webserver agisce su di un registro esterno ad 8 bit che acquisisce il dato in parallelo dal bus dei dati nel momento in cui riceve un clock sull'ingresso dedicato.

Per invertire lo stato di un bit senza la conoscenza di quale sia il suo stato precedente, viene sfruttata l'operazione di XOR che restituisce il valore booleano "0" quando i due bit di ingresso sono uguali e il valore booleano "1" quando questi sono diversi. Supponendo quindi di voler modificare l'uscita numero 2 (bit Z) del registro delle uscite rappresentato in binario come "xxxxxZxx" è sufficiente eseguire la funzione XOR tra tale registro ed il byte "00000100".

Codice sorgente del modulo di interfaccia

```
/********************************************************************************
*                                    Modulo Webserver
********************************************************************************/
#define THIS_IS_STACK_APPLICATION
#define BAUD_RATE       (19200)
#define USART_USE_BRGH_LOW

#include <string.h>
#include "stacktsk.h"
#include "tick.h"
#include "mpfs.h"
#include "xlcd.h"
#include "xeeprom.h"
#include "delay.h"

#if defined(STACK_USE_DHCP)
#include "dhcp.h"
#endif

#define STARTUP_MSG "CISIF-Web Server"
ROM char StartupMsg[] = STARTUP_MSG;
ROM char SetupMsg[] = "Setup utente...";
APP_CONFIG AppConfig;

BYTE myDHCPBindCount = 0;

BYTE DIGITAL_OUTPUT = 0;
static void ToggleOut(void);

BYTE DIGITAL_INPUT = 0;
static void Read_input(void);

BOOL SuperUser = FALSE;
BOOL ADMINVerify(char *login, char *password);

static void InitAppConfig(void);
static void InitializeBoard(void);
static void ProcessIO(void);
void NotifyRemoteUser(void);
static void DisplayIPValue(IP_ADDR *IPVal, BOOL bToLCD);
static void SetConfig(void);
static BOOL DownloadMPFS(void);
static void SaveAppConfig(void);

#pragma interrupt HighISR save=section(".tmpdata")
void HighISR(void)
void interrupt HighISR(void)
{
        TickUpdate();
}
#pragma code highVector=0x08
void HighVector (void)
{
    _asm goto HighISR _endasm
}
#pragma code

static void USARTPut(BYTE c)
{
    while( !TXSTA_TRMT);
    TXREG = c;
}

static void USARTPutString(BYTE *s)
{
    BYTE c;

    while( (c = *s++) )
        USARTPut(c);
}

#define USARTIsGetReady()    (PIR1_RCIF)
#define USARTGet()           (RCREG)
```

```
void main(void)
{
    static TICK t = 0;
    InitializeBoard();
    TickInit();
    MPFSInit();
    InitAppConfig();
    if ( PORTB_RB5 == 0 )
    {
        XLCDGoto(1, 0);
        XLCDPutROMString(SetupMsg);
        SetConfig();
    }
    StackInit();
    HTTPInit();
    FTPInit();

#if defined(STACK_USE_DHCP) || defined(STACK_USE_IP_GLEANING)
    if ( AppConfig.Flags.bIsDHCPEnabled )
    {
        XLCDGoto(1, 0);
        XLCDPutROMString(DHCPMsg);
    }
    else
    {
        myDHCPBindCount = 1;
#if defined(STACK_USE_DHCP)
        DHCPDisable();
#endif
    }
#endif

    while(1)
    {
        if ( TickGetDiff(TickGet(), t) >= TICK_SECOND/2 )
        {
            t = TickGet();
            LATA4 ^= 1;
        }
        StackTask();
        HTTPServer();
        FTPServer();
        ProcessIO();
        if ( DHCPBindCount != myDHCPBindCount )
        {
            DisplayIPValue(&AppConfig.MyIPAddr, TRUE);
            myDHCPBindCount = DHCPBindCount;
            if ( AppConfig.Flags.bIsDHCPEnabled )
            {
                XLCDGoto(1, 14);
                if ( myDHCPBindCount < 0x0a )
                    XLCDPut(myDHCPBindCount + '0');
                else
                    XLCDPut(myDHCPBindCount + 'A');
            }
        }
    }
}

ROM char blankLCDLine[] = "                ";

static void DisplayIPValue(IP_ADDR *IPVal, BOOL bToLCD)
{
    char IPDigit[8];

    if ( bToLCD )
    {
        XLCDGoto(1, 0);
        XLCDPutROMString(blankLCDLine);
    }
    XLCDGoto(1, 0);

    itoa(IPVal->v[0], IPDigit);
    if ( bToLCD )
    {
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
```

```
    else
    {
        USARTPutString((BYTE*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[1], IPDigit);
    if ( bToLCD )
    {
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else
    {
        USARTPutString((BYTE*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[2], IPDigit);
    if ( bToLCD )
    {
        XLCDPutString(IPDigit);
        XLCDPut('.');
    }
    else
    {
        USARTPutString((BYTE*)IPDigit);
        USARTPut('.');
    }

    itoa(IPVal->v[3], IPDigit);
    if ( bToLCD )
        XLCDPutString(IPDigit);
    else
        USARTPutString((BYTE*)IPDigit);
}

static char AN0String[8];
static char AN1String[8];

static void ProcessIO(void)
{
    WORD_VAL ADCResult;


    ADCON0      = 0b10000001;
    ADCResult.v[0] = 100;
    while( ADCResult.v[0]-- );
    ADCON0_GO   = 1;
    while( ADCON0_GO );
    ADCResult.v[0] = ADRESL;
    ADCResult.v[1] = ADRESH;
    itoa(ADCResult.Val, AN0String);

    ADCON1      = 0b10000100;
    ADCON0      = 0b10001001;
    ADCResult.v[0] = 100;
    while( ADCResult.v[0]-- );
    ADCON0_GO   = 1;
    while( ADCON0_GO );
    ADCResult.v[0] = ADRESL;
    ADCResult.v[1] = ADRESH;
    itoa(ADCResult.Val, AN1String);

    ADCON1      = 0b10001110;
    Read_input();
}

#define CGI_CMD_DIGOUT      (0)
#define CGI_CMD_LCDOUT      (1)
#define CGI_CMD_LOGIN       (2)
#define VAR_L1                      (0)
#define VAR_L2                      (1)
#define VAR_L3                      (2)
#define VAR_L4                      (3)
#define VAR_L5                      (4)
#define VAR_L6                      (5)
#define VAR_L7                      (6)
```

```
#define VAR_L8                      (7)

#define INP_I1                      (0)
#define INP_I2                      (1)
#define INP_I3                      (2)
#define INP_I4                      (3)
#define INP_I5                      (4)
#define INP_I6                      (5)
#define INP_I7                      (6)
#define INP_I8                      (7)
#define INP_O1                      (8)
#define INP_O2                      (9)
#define INP_O3                      (16)
#define INP_O4                      (17)
#define INP_O5                      (18)
#define INP_O6                      (19)
#define INP_O7                      (20)
#define INP_O8                      (21)
#define INP_A1                      (22)
#define INP_A2                      (23)
#define USER_LOG                    (33)
#define VAR_STROUT_LCD              (34)

ROM char COMMANDS_OK_PAGE[] = "OUTDIG.CGI";
#define COMMANDS_OK_PAGE_LEN  (sizeof(COMMANDS_OK_PAGE))
ROM char CMD_UNKNOWN_PAGE[] = "HOME.HTM";
#define CMD_UNKNOWN_PAGE_LEN    (sizeof(CMD_UNKNOWN_PAGE))
ROM char ADMIN_USER[]    = "admin";
#define ADMIN_USER_LEN   (sizeof(ADMIN_USER)-1)
ROM char ADMIN_PASS[]    = "microchip";
#define ADMIN_PASS_LEN   (sizeof(ADMIN_PASS)-1)

void HTTPExecCmd(BYTE** argv, BYTE argc)
{
    BYTE command;
    BYTE var;

    command = argv[0][0] - '0';

    switch(command)
    {
    case CGI_CMD_DIGOUT:
        var = argv[1][0] - '0';
              if (SuperUser == TRUE)
              {
        switch(var)
        {
              case VAR_L1:
                            DIGITAL_OUTPUT ^= 0b00000001;
                            ToggleOut();
              break;
              case VAR_L2:
                            DIGITAL_OUTPUT ^= 0b00000010;
                            ToggleOut();
              break;
              case VAR_L3:
                            DIGITAL_OUTPUT ^= 0b00000100;
                            ToggleOut();
              break;
              case VAR_L4:
                            DIGITAL_OUTPUT ^= 0b00001000;
                            ToggleOut();
              break;
              case VAR_L5:
                            DIGITAL_OUTPUT ^= 0b00010000;
                            ToggleOut();
              break;
              case VAR_L6:
                            DIGITAL_OUTPUT ^= 0b00100000;
                            ToggleOut();
              break;
              case VAR_L7:
                            DIGITAL_OUTPUT ^= 0b01000000;
                            ToggleOut();
              break;
              case VAR_L8:
                            DIGITAL_OUTPUT ^= 0b10000000;
                            ToggleOut();
```

```
                break;
                }
                }
                memcpypgm2ram(argv[0],
               (const ROM char*)COMMANDS_OK_PAGE, COMMANDS_OK_PAGE_LEN);
        break;

    case CGI_CMD_LCDOUT:
                if (SuperUser == TRUE)
                {
                        XLCDGoto(0, 0);
                        XLCDPutROMString(blankLCDLine);
                        XLCDGoto(1, 0);
                        XLCDPutROMString(blankLCDLine);
                        XLCDGoto(0, 0);
                        XLCDPutString(argv[2]);
                        XLCDGoto(1, 0);
                        XLCDPutString(argv[4]);
                }
                 memcpypgm2ram((unsigned char*)argv[0],
                    (const ROM char*)CMD_UNKNOWN_PAGE, CMD_UNKNOWN_PAGE_LEN);
        break;

    case CGI_CMD_LOGIN:
        if ( ADMINVerify(argv[2], argv[4]) )
                {       SuperUser = TRUE;}
        else
        {       SuperUser = FALSE;}
        memcpypgm2ram((unsigned char*)argv[0],
                (const ROM char*)CMD_UNKNOWN_PAGE, CMD_UNKNOWN_PAGE_LEN);
        break;

    default:
        memcpypgm2ram((unsigned char*)argv[0],
               (const ROM char*)CMD_UNKNOWN_PAGE, CMD_UNKNOWN_PAGE_LEN);
        break;
    }
}
#endif

WORD HTTPGetVar(BYTE var, WORD ref, BYTE* val)
{
    switch(var)
    {
    case USER_LOG:
                if (SuperUser == TRUE)
                        *val = '1';
                else
                        *val = '0';
                break;

       case INP_I1:
        if ( DIGITAL_INPUT&0b00000001)
            *val = '1';
        else
            *val = '0';
        break;
    case INP_I2:
        if ( DIGITAL_INPUT&0b00000010 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_I3:
        if ( DIGITAL_INPUT&0b00000100 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_I4:
        if ( DIGITAL_INPUT&0b00001000 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_I5:
        if ( DIGITAL_INPUT&0b00010000 )
            *val = '1';
```

```
        else
            *val = '0';
        break;
    case INP_I6:
        if ( DIGITAL_INPUT&0b00100000 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_I7:
        if ( DIGITAL_INPUT&0b01000000 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_I8:
        if ( DIGITAL_INPUT&0b10000000 )
            *val = '1';
        else
            *val = '0';
        break;

    case INP_O1:
        if ( DIGITAL_OUTPUT&0b00000001)
            *val = '1';
        else
            *val = '0';
        break;
    case INP_O2:
        if ( DIGITAL_OUTPUT&0b00000010)
            *val = '1';
        else
            *val = '0';
        break;
    case INP_O3:
        if ( DIGITAL_OUTPUT&0b00000100 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_O4:
        if ( DIGITAL_OUTPUT&0b00001000 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_O5:
        if ( DIGITAL_OUTPUT&0b00010000 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_O6:
        if ( DIGITAL_OUTPUT&0b00100000 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_O7:
        if ( DIGITAL_OUTPUT&0b01000000 )
            *val = '1';
        else
            *val = '0';
        break;
    case INP_O8:
        if ( DIGITAL_OUTPUT&0b10000000 )
            *val = '1';
        else
            *val = '0';
        break;

    case INP_A1:
        if ( ref == HTTP_START_OF_VAR )
        {
            ref = (BYTE)0;
        }
        *val = AN0String[(BYTE)ref];
        if ( AN0String[(BYTE)ref] == '\0' )
```

```
            return HTTP_END_OF_VAR;

        (BYTE)ref++;
        return ref;
       break;

    case INP_A2:
        if ( ref == HTTP_START_OF_VAR )
        {
            ref = (BYTE)0;
        }
        *val = AN1String[(BYTE)ref];
        if ( AN1String[(BYTE)ref] == '\0' )
            return HTTP_END_OF_VAR;

        (BYTE)ref++;
        return ref;
       break;
    }
    return HTTP_END_OF_VAR;
}

ROM char FTP_USER_NAME[]    = "ftp";
#define FTP_USER_NAME_LEN   (sizeof(FTP_USER_NAME)-1)
ROM char FTP_USER_PASS[]    = "microchip";
#define FTP_USER_PASS_LEN   (sizeof(FTP_USER_PASS)-1)

BOOL FTPVerify(char *login, char *password)
{
    if ( !memcmppgm2ram(login, (ROM void*)FTP_USER_NAME, FTP_USER_NAME_LEN) )
    {
        if ( !memcmppgm2ram(password, (ROM void*)FTP_USER_PASS, FTP_USER_PASS_LEN) )
            return TRUE;
    }
    return FALSE;
}
#endif

BOOL ADMINVerify(char *login, char *password)
{
    if ( !memcmppgm2ram(login, ADMIN_USER, ADMIN_USER_LEN) )
    {
        if ( !memcmppgm2ram(password, ADMIN_PASS, ADMIN_PASS_LEN) )
            return TRUE;
    }
    return FALSE;
}

static void InitializeBoard(void)
{
        ADCON1  = 0b10001110;
        TRISA   = 0x03;
        PORTA_RA5 = 0;
        PORTC_RC0=0;
        TRISC_RC0=0;
        ToggleOut();
        PORTC_RC1=0;
        TRISC_RC1=0;
        PORTC_RC5=0;
        TRISC_RC5=0;
        INTCON2_RBPU = 0;

    XLCDInit();
    XLCDGoto(0, 0);
    XLCDPutROMString(StartupMsg);

    TXSTA = 0b00100000;
    RCSTA = 0b10010000;
    SPBRG = SPBRG_VAL;
    T0CON = 0;
    INTCON_GIEH = 1;
    INTCON_GIEL = 1;
}

static void InitAppConfig(void)
{
    BYTE c;
    BYTE *p;
```

```
    AppConfig.MyIPAddr.v[0]     = MY_DEFAULT_IP_ADDR_BYTE1;
    AppConfig.MyIPAddr.v[1]     = MY_DEFAULT_IP_ADDR_BYTE2;
    AppConfig.MyIPAddr.v[2]     = MY_DEFAULT_IP_ADDR_BYTE3;
    AppConfig.MyIPAddr.v[3]     = MY_DEFAULT_IP_ADDR_BYTE4;

    AppConfig.MyMask.v[0]       = MY_DEFAULT_MASK_BYTE1;
    AppConfig.MyMask.v[1]       = MY_DEFAULT_MASK_BYTE2;
    AppConfig.MyMask.v[2]       = MY_DEFAULT_MASK_BYTE3;
    AppConfig.MyMask.v[3]       = MY_DEFAULT_MASK_BYTE4;

    AppConfig.MyGateway.v[0]    = MY_DEFAULT_GATE_BYTE1;
    AppConfig.MyGateway.v[1]    = MY_DEFAULT_GATE_BYTE2;
    AppConfig.MyGateway.v[2]    = MY_DEFAULT_GATE_BYTE3;
    AppConfig.MyGateway.v[3]    = MY_DEFAULT_GATE_BYTE4;

    AppConfig.MyMACAddr.v[0]    = MY_DEFAULT_MAC_BYTE1;
    AppConfig.MyMACAddr.v[1]    = MY_DEFAULT_MAC_BYTE2;
    AppConfig.MyMACAddr.v[2]    = MY_DEFAULT_MAC_BYTE3;
    AppConfig.MyMACAddr.v[3]    = MY_DEFAULT_MAC_BYTE4;
    AppConfig.MyMACAddr.v[4]    = MY_DEFAULT_MAC_BYTE5;
    AppConfig.MyMACAddr.v[5]    = MY_DEFAULT_MAC_BYTE6;

    AppConfig.Flags.bIsDHCPEnabled = TRUE;

    p = (BYTE*)&AppConfig;

    XEEBeginRead(EEPROM_CONTROL, 0x00);
    c = XEERead();
    XEEEndRead();

    if ( c == 0x55 )
    {
        XEEBeginRead(EEPROM_CONTROL, 0x01);
        for ( c = 0; c < sizeof(AppConfig); c++ )
            *p++ = XEERead();
        XEEEndRead();
    }
    else
        SaveAppConfig();
}

static void SaveAppConfig(void)
{
    BYTE c;
    BYTE *p;

    p = (BYTE*)&AppConfig;
    XEEBeginWrite(EEPROM_CONTROL, 0x00);
    XEEWrite(0x55);
    for ( c = 0; c < sizeof(AppConfig); c++ )
        XEEWrite(*p++);
    XEEEndWrite();
}

ROM char menu[] =
    "\r\n\r\n\r\Web-Server v1.0 ("STARTUP_MSG", " __DATE__ ")\r\n\r\n"

    "\t1: Modifica numero seriale.\r\n"
    "\t2: Modifica default IP address.\r\n"
    "\t3: Modifica default gateway address.\r\n"
    "\t4: Modifica default subnet mask.\r\n"
    "\t5: Abilita DHCP & IP Gleaning.\r\n"
    "\t6: Disabilita DHCP & IP Gleaning.\r\n"
    "\t7: Download MPFS image.\r\n"
    "\t8: Salva & Esci.\r\n"
    "\r\n"
    "Fai una scelta (1-8): ";

typedef enum _MENU_CMD
{
    MENU_CMD_SERIAL_NUMBER          = '1',
    MENU_CMD_IP_ADDRESS,
    MENU_CMD_GATEWAY_ADDRESS,
    MENU_CMD_SUBNET_MASK,
    MENU_CMD_ENABLE_AUTO_CONFIG,
    MENU_CMD_DISABLE_AUTO_CONFIG,
    MENU_CMD_DOWNLOAD_MPFS,
```

```
    MENU_CMD_QUIT,
    MENU_CMD_INVALID
} MENU_CMD;

ROM char* menuCommandPrompt[] =
{
    "\r\nNumero seriale (",
    "\r\nDefault IP Address (",
    "\r\nDefault Gateway Address (",
    "\r\nDefault Subnet Mask (",
    "\r\nDHCP abilitato.\r\n",
    "\r\nDHCP disabilitato.\r\n",
    "\r\nPronto per ricevere l'immagine MPFS con protocollo Xmodem.\r\n",
    "\r\nApplicazione partita"
};

ROM char InvalidInputMsg[] = "\r\nDati immessi non validi e non memorizzati.\r\n"
                             "Premi un tasto per continuare...\r\n";

void USARTPutROMString(ROM char* str)
{
    BYTE v;

    while( v = *str++ )
        USARTPut(v);
}


BYTE USARTGetString(char *buffer, BYTE bufferLen)
{
    BYTE v;
    BYTE count;

    count = 0;
    do
    {
        while( !USARTIsGetReady() );

        v = USARTGet();

        if ( v == '\r' || v == '\n' )
            break;

        count++;
        *buffer++ = v;
        *buffer = '\0';
        if ( bufferLen-- == 0 )
            break;
    } while(1);
    return count;
}

BOOL StringToIPAddress(char *str, IP_ADDR *buffer)
{
    BYTE v;
    char *temp;
    BYTE byteIndex;

    temp = str;
    byteIndex = 0;

    while( v = *str )
    {
        if ( v == '.' )
        {
            *str++ = '\0';
            buffer->v[byteIndex++] = atoi(temp);
            temp = str;
        }
        else if ( v < '0' || v > '9' )
            return FALSE;

        str++;
    }

    buffer->v[byteIndex] = atoi(temp);
    return (byteIndex == 3);
}
```

```
MENU_CMD GetMenuChoice(void)
{
    BYTE c;

    while ( !USARTIsGetReady() );
    c = USARTGet();

    if ( c >= '1' && c < MENU_CMD_INVALID )
        return c;
    else
        return MENU_CMD_INVALID;
}

#define MAX_USER_RESPONSE_LEN   (20)
void ExecuteMenuChoice(MENU_CMD choice)
{
    char response[MAX_USER_RESPONSE_LEN];
    IP_ADDR tempIPValue;
    IP_ADDR *destIPValue;

    USARTPut('\r');
    USARTPut('\n');
    USARTPutROMString(menuCommandPrompt[choice-'0'-1]);

    switch(choice)
    {
    case MENU_CMD_SERIAL_NUMBER:
        itoa(AppConfig.SerialNumber.Val, response);
        USARTPutString((BYTE*)response);
        USARTPut(')');
        USARTPut(':');
        USARTPut(' ');

        if ( USARTGetString(response, sizeof(response)) )
        {
            AppConfig.SerialNumber.Val = atoi(response);
            AppConfig.MyMACAddr.v[4] = AppConfig.SerialNumber.v[1];
            AppConfig.MyMACAddr.v[5] = AppConfig.SerialNumber.v[0];
        }
        else
            goto HandleInvalidInput;
        break;

    case MENU_CMD_IP_ADDRESS:
        destIPValue = &AppConfig.MyIPAddr;
        goto ReadIPConfig;

    case MENU_CMD_GATEWAY_ADDRESS:
        destIPValue = &AppConfig.MyGateway;
        goto ReadIPConfig;

    case MENU_CMD_SUBNET_MASK:
        destIPValue = &AppConfig.MyMask;

    ReadIPConfig:
        DisplayIPValue(destIPValue, FALSE);
        USARTPut(')');
        USARTPut(':');
        USARTPut(' ');
        USARTGetString(response, sizeof(response));

        if ( !StringToIPAddress(response, &tempIPValue) )
        {
HandleInvalidInput:
            USARTPutROMString(InvalidInputMsg);
            while( !USARTIsGetReady() );
            USARTGet();
        }
        else
            destIPValue->Val = tempIPValue.Val;
        break;

    case MENU_CMD_ENABLE_AUTO_CONFIG:
        AppConfig.Flags.bIsDHCPEnabled = TRUE;
        break;
```

```
    case MENU_CMD_DISABLE_AUTO_CONFIG:
        AppConfig.Flags.bIsDHCPEnabled = FALSE;
        break;

    case MENU_CMD_DOWNLOAD_MPFS:
        DownloadMPFS();
        break;

    case MENU_CMD_QUIT:
        SaveAppConfig();
        break;
    }
}

static void SetConfig(void)
{
    MENU_CMD choice;

    do
    {
        USARTPutROMString(menu);
        choice = GetMenuChoice();
        if ( choice != MENU_CMD_INVALID )
            ExecuteMenuChoice(choice);
    } while(choice != MENU_CMD_QUIT);

}

#define XMODEM_SOH      0x01
#define XMODEM_EOT      0x04
#define XMODEM_ACK      0x06
#define XMODEM_NAK      0x15
#define XMODEM_CAN      0x18
#define XMODEM_BLOCK_LEN 128

static BOOL DownloadMPFS(void)
{
    enum SM_MPFS
    {
        SM_MPFS_SOH,
        SM_MPFS_BLOCK,
        SM_MPFS_BLOCK_CMP,
        SM_MPFS_DATA,
    } state;

    BYTE c;
    MPFS handle;
    BOOL lbDone;
    BYTE blockLen;
    BYTE lResult;
    BYTE tempData[XMODEM_BLOCK_LEN];
    TICK lastTick;
    TICK currentTick;

    state = SM_MPFS_SOH;
    lbDone = FALSE;

    handle = MPFSFormat();
    lastTick = TickGet();
    do
    {
        TickUpdate();

        currentTick = TickGet();
        if ( TickGetDiff(currentTick, lastTick) >= (TICK_SECOND/2) )
        {
            lastTick = TickGet();
            USARTPut(XMODEM_NAK);
            LATA2 ^= 1;
        }

    } while( !USARTIsGetReady() );

    while(!lbDone)
    {
        TickUpdate();
        if ( USARTIsGetReady() )
```

```
        {
            LATA2 ^= 1;
            c = USARTGet();
        }
        else
            continue;

        switch(state)
        {
        default:
            if ( c == XMODEM_SOH )
            {
                state = SM_MPFS_BLOCK;
            }
            else if ( c == XMODEM_EOT )
            {
                LATA2 = 1;
                MPFSClose();
                USARTPut(XMODEM_ACK);
                lbDone = TRUE;
            }
            else
                USARTPut(XMODEM_NAK);

            break;

        case SM_MPFS_BLOCK:
            lResult = XMODEM_ACK;
            blockLen = 0;
            state = SM_MPFS_BLOCK_CMP;
            break;

        case SM_MPFS_BLOCK_CMP:
            state = SM_MPFS_DATA;
            break;

        case SM_MPFS_DATA:
            tempData[blockLen++] = c;
            if ( blockLen > XMODEM_BLOCK_LEN )
            {
                MPFSPutBegin(handle);
                lResult = XMODEM_ACK;
                for ( c = 0; c < XMODEM_BLOCK_LEN; c++ )
                    MPFSPut(tempData[c]);
                handle = MPFSPutEnd();
                USARTPut(lResult);
                state = SM_MPFS_SOH;
            }
            break;
        }
    }
    return TRUE;
}

void XLCDDelay15ms(void)
{
    DelayMs(15);
}

void XLCDDelay4ms(void)
{
    DelayMs(4);
}

void XLCDDelay100us(void)
{
    INTCON_GIEH = 0;
    Delay10us(1);
    INTCON_GIEH = 1;
}

void ToggleOut(void)
{
        PORTC_RC0=0;
        TRISD=0x00;
        PORTD=DIGITAL_OUTPUT;
        Nop();
        Nop();
```

```
        Nop();
        Nop();
        PORTC_RC0=1;
        Nop();
        PORTC_RC0=0;
}

void Read_input(void)
{
        TRISD=0xff;
        PORTC_RC1=0;
        Nop();
        Nop();
        Nop();
        Nop();
        PORTC_RC1=1;
        Nop();
        Nop();
        Nop();
        Nop();
        PORTC_RC1=0;
        Nop();
        Nop();
        Nop();
        Nop();
        PORTC_RC5=1;
        Nop();
        Nop();
        DIGITAL_INPUT = PORTD;
        PORTC_RC1=0;
        PORTC_RC5=0;
}
```

Codice sorgente del modulo Webserver.c

Struttura delle pagine WEB

Generalità di implementazione delle pagine WEB

Nell'implementazione delle pagine WEB, è stato necessario ridurne la complessità e di conseguenza l'impatto grafico con l'utente per riuscire a contenerle nella memoria EEPROM seriale da 32KByte. La struttura delle pagine è di tipo tabellare con un IFRAME (In Line Frame) centrale che consente l'inserimento di un frame in modo dinamico.

Questo tag è correttamente supportato da tutti i browser moderni (Netscape lo riconosce dalla versione 5). Di seguito sono riportati la rappresentazione grafica della pagina principale ed il relativo codice.

Ciascuna pagina è implementata con una tabella di due righe e due colonne.

Nella prima riga le colonne sono state raggruppate in una colonna unica e si è inserita un'immagine (e.g. il logo dell'Università di Pisa) ed un titolo sempre presenti.

È stato inoltre utilizzato un foglio di stile per gestire il colore di sfondo, l'altezza, l'allineamento e la linea in basso in termini di spessore e colore.

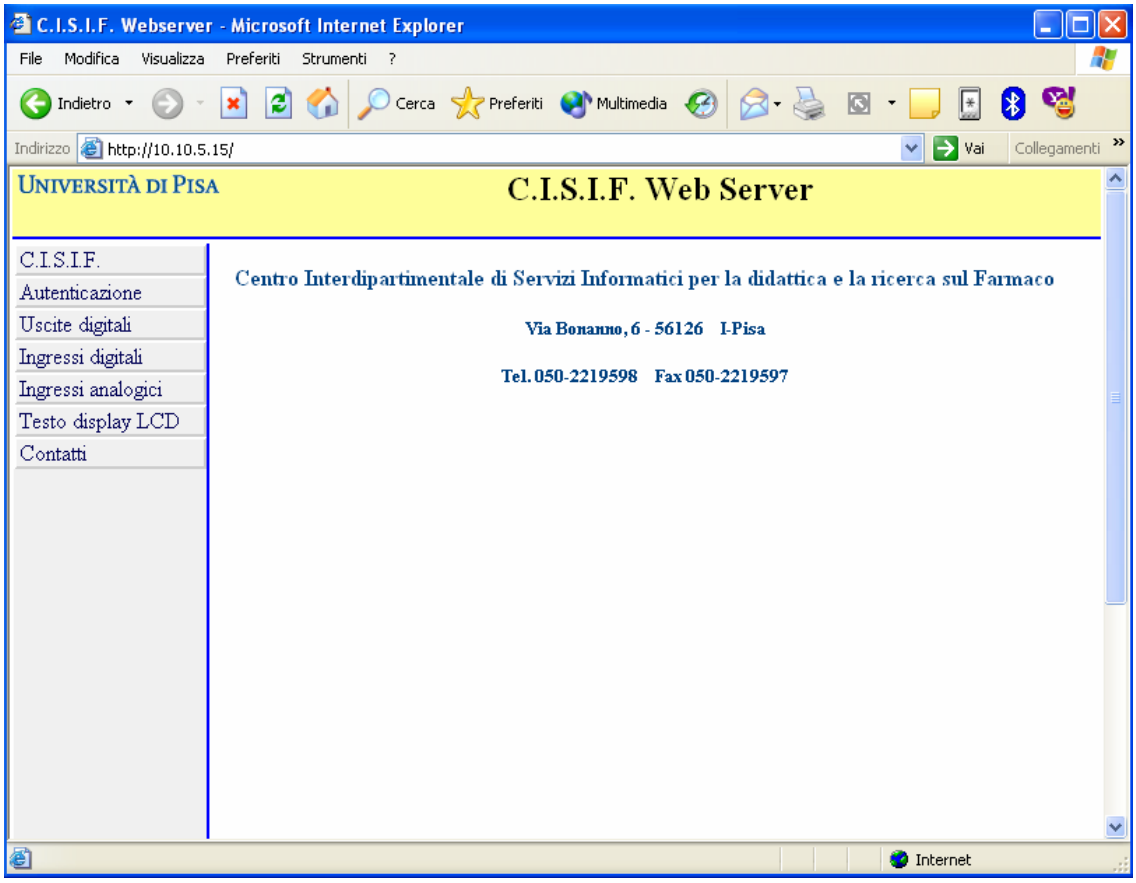
```
.pglog {
font-size: large;
color: #0000FF;
align:center;
}
.a {
color: #00ff00;
text-decoration: none;
}
.tbrd {
border: 2px solid #999999;
padding: 0px;
}
.pinbtn {
font-weight:bold;
width: 38px;
}
.pinlbl {
font-weight: bold;
}
```

Codice del foglio di stile

La Home Page

La Home Page del Webserver è mostrata nella figura sottostante. Nella colonna di sinistra è stata inclusa una sottotabella di sette righe e una colonna dove sono stati inglobati 7 pulsanti implementati con la tecnica dello stile senza impiego di tag di input oppure di immagini ad hoc.

Al passaggio del mouse sopra una voce del menù, quest'ultima cambia colore per evidenziare la presenza di un collegamento ad una pagina interna che verrà successivamente visualizzata nella cella in basso a destra (finestra grande principale) della tabella principale con il metodo degli Iframe visto in precedenza.



Home Page

```
<html>
<head>
<title>C.I.S.I.F. Webserver</title>
<style type="text/css">
.top {border-bottom: 2px solid #0000ff;
       background-color: #ffff99;
       width: 100%;
       height: 50px;
       vertical-align: middle;}
.menuBtn {
       border-top: 2px solid #FFFFFF;
       border-right: 2px solid #CCCCCC;
       border-bottom: 2px solid #CCCCCC;
       border-left: 2px solid #FFFFFF;
       text-indent: 2px;}
.left {border-right: 2px solid #0000FF; background-color: #F0F0F0;}
a {color: #000055; text-decoration: none;}
a:hover {color: #ff0000; text-decoration: none;}
</style>
</head>
<body scrolling="auto" marginwidth="0" marginheight="0" topmargin="0" bottommargin="0"
leftmargin="0" rightmargin="0">
<table width="100%" height="692">
<tr class="top"><td colspan="2" class="top" height="32">
<h2 align="center">
</h2>
<h2 align="center">
<img border="0" src="logo.gif" align="left" width="144" height="20">C.I.S.I.F. Web
Server</h2>
</td></tr>
<tr>
  <td height="667"><table height="100%" class="left" cellpadding="0" cellspacing="0"
width="140">
    <tr height="20"><td class="menuBtn"><a href="home.htm"
target="main">C.I.S.I.F.</a></td></tr>
    <tr height="20"><td class="menuBtn"><a href="login.cgi"
target="main">Autenticazione</a></td></tr>
    <tr height="20"><td class="menuBtn"><a href="outdig.cgi" target="main">Uscite
digitali</a></td></tr>
    <tr height="20"><td class="menuBtn"><a href="indig.cgi" target="main">Ingressi
digitali</a></td></tr>
    <tr height="20"><td class="menuBtn"><a href="inana.cgi" target="main">Ingressi
analogici</a></td></tr>
    <tr height="20"><td class="menuBtn"><a href="display.cgi" target="main">Testo
display LCD</a></td></tr>
    <tr height="20"><td class="menuBtn"><a href="contact.htm"
target="main">Contatti</a></td></tr>
    <tr><td> </td></tr>
  </table></td>
  <td width="100%" height="667">
  <iframe name="main" src="home.htm" width="100%" height="100%" scrolling="auto"
frameborder="0">
  <br>Main Page
  </td>
</tr>
</body>
</html>
```

Codice della Home Page

```
<html>
<head>
<title>C.I.S.I.F. Home</title></head>
<body bgcolor="#FFFFFF">
<table border="0" cellpadding="0" cellspacing="0" width="100%">
  <tr>
    <td valign="top" width="100%">
    <p align="center"><font face="Arial Rounded MT Bold" color="#004080">
    <strong>Centro Interdipartimentale di Servizi Informatici per la didattica e
    la ricerca sul Farmaco</strong></font></p>
    <p align="center"><font face="Arial Rounded MT Bold" color="#004080"><small>
    <strong>Via Bonanno, 6 - 56126    I-
Pisa</strong></small></font></p>
    <p align="center"><font face="Arial Rounded MT Bold" color="#004080"><small>
    <strong>Tel. 050-2219598    Fax 050-
2219597</strong></small></font></p>
    <p ALIGN="JUSTIFY"> </p>
    </td>
  </tr>
</table>
</body>
</html>
```

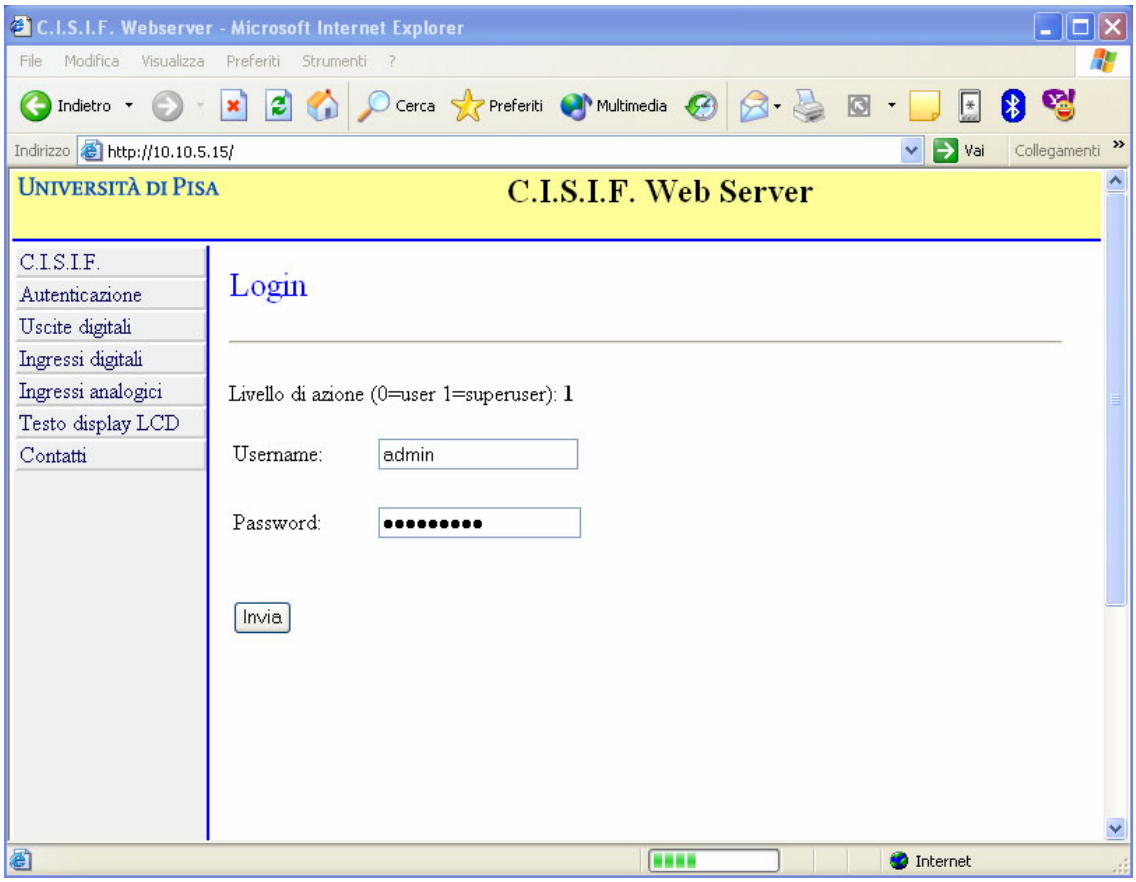
Codice della pagina principale inserita nell'Iframe

La pagina di autenticazione

L'accesso al Webserver è consentito a tutti coloro che ne conoscono l'indirizzo IP o il suo nome registrato in un DNS: per questo motivo, mentre è possibile per tutti leggere lo stato degli ingressi analogici e/o digitali e delle uscite digitali, non deve esserne permessa la modifica, come pure la scrittura sul display LCD.

Una delle pagine del menù realizza un'autenticazione di tipo software attraverso l'inserimento di due voci: l'account e la password di utente. Al tempo stesso visualizza lo stato dell'utente attualmente connesso che può essere di tipo user (con funzioni di sola lettura) o di tipo superuser (con funzioni aggiuntive di modifica).

Di seguito sono riportati la rappresentazione grafica della pagina di login ed il relativo codice.



Pagina di autenticazione

Da notare che la pagina di login, come altre pagine che verranno analizzate in seguito, ha l'estensione ".CGI" invece della standard ".htm": ogni pagina che contiene al suo interno delle variabili da visualizzare e/o pulsanti di comando per avviare un'azione, deve poter essere distinta dal Webserver e ciò viene implementato modificando

l'estensione del nome della pagina, in modo simile a quanto avviene per i moduli CGI presenti all'interno di un server WEB. La pagina di login contiene una variabile da visualizzare (il livello attuale di azione che identifica il tipo di utente connesso), due campi di input di cui il primo di tipo testo ed il secondo di tipo password ed un pulsante di comando che fa partire l'invio dei dati inseriti verso il Webserver.
Analizzando il sorgente della pagina di login, si nota la direttiva che indica al browser che il contenuto richiesto non deve essere messo in alcuna cache: questa direttiva è fondamentale per pagine dinamiche che devono essere aggiornate di frequente, come in questo caso.
Più in basso, dove viene descritto il livello di azione, da notare la variabile "%21" che, al momento della creazione dinamica della pagina, verrà sostituita con il valore presente all'interno della variabile corrispondente al numero esadecimale 0x21 nel Webbroser.
All'interno di questa pagina è stato creato un form con il metodo "GET" e la "action=2". Il metodo "GET" indica al browser di inviare i dati secondo la sintassi standard direttamente attraverso la URL, mentre il valore di "action" verrà sfruttato dal Webserver per identificare la pagina che ha eseguito una richiesta di modifica dello stato di funzionamento. I codici "action" implementati sono tre e sono visibili nella tabella sottostante.

METODO	PAGINA	DESCRIZIONE
0	command.cgi	Modifica output digitali
1	display.cgi	Scrive sulle due righe del display
2	login.cgi	Logga l'utente come utente o superuser

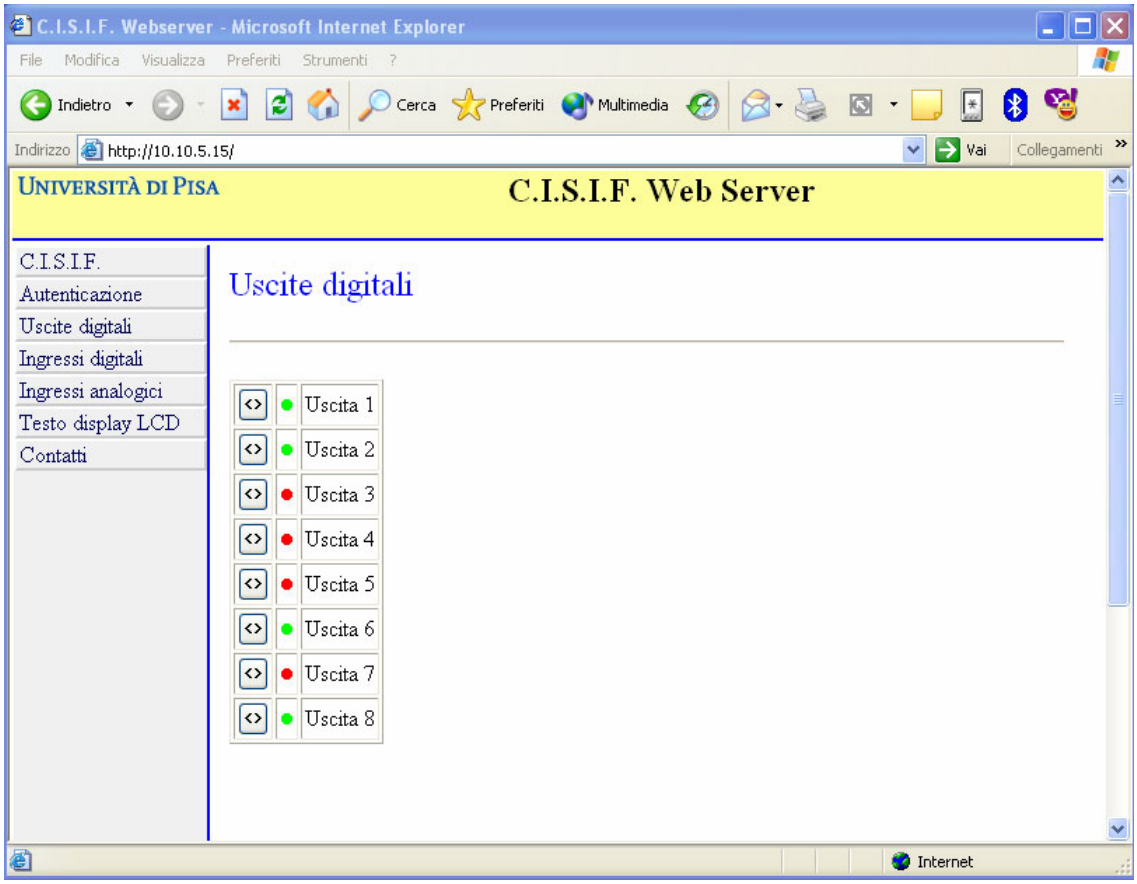
All'interno del form sono presenti due tag di input con l'impostazione della lunghezza massima, il nome ed il tipo.
La lunghezza massima fissata consente di controllare e limitare la stringa inviata dal browser al Webserver.
Il nome invece deve essere presente perché il Webserver lo identifica e lo impiega per discriminare il tag di input corrispondente.
Infine il tipo permette di visualizzare il testo in chiaro nel campo "Username" ma di mascherarlo per il campo "Password" come di consuetudine.

```
<html>
<head>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<link href="stili.css" rel="stylesheet" type="text/css">
</head>
<body>
<p class="pglog">Login</p>
<hr>
<br>Livello di azione (0=user 1=superuser): <b>%21<b>
<br>
<br>
<form method=GET action=2>
<table>
<tr><td width="100">Username:</td>
<td><input type=text size=20 maxlength=10 name=UT></td>
</table>
<br>
<table>
<tr><td width="100">Password:</td>
<td><input type=password size=20 maxlength=10 name=PW></td>
</table>
<br>
<br>
<table>
<tr><td><input type=submit value="Invia"></td>
</table>
</form>
</body>
</html>
```

Codice dalla pagina di autenticazione

La pagina delle uscite digitali

Come accennato in precedenza, il modulo Webserver gestisce 8 uscite digitali che possono essere impiegate con diversi tipi di interfacce hardware per il controllo di periferiche eterogenee (a bassa tensione ca e/o cc, ad alta tensione, per carichi induttivi e/o capacitivi, ecc). Deve essere possibile quindi sia visualizzare lo stato di queste uscite, sia poterlo modificare attraverso la finestra del browser: la pagina che consente tali operazioni è quella mostrata sotto.



Pagina delle uscite digitali

Tramite questa interfaccia vengono visualizzati contemporaneamente gli otto canali di uscita e, in funzione del livello di accesso dell'utente, tali stati possono essere modificati uno ad uno utilizzando gli specifici pulsanti di comando che effettuano un "toggle" di stato.

Non appena il comando viene inviato, il Webserver risponde visualizzando la pagina aggiornata con lo stato dell'uscita: i cerchietti verdi o rossi simulano la presenza di LED e segnalano lo stato di ogni singola uscita (led verde = uscita ON, led rosso = uscita OFF). Vedremo dall'analisi del sorgente della pagina che modificare il colore in

rapporto allo stato dell'uscita oppure anche l'immagine dei LED è sufficiente cambiare pochi parametri.

Anche questa pagina ha il nome con estensione ".CGI".

Da notare che questa pagina è accessibile da chiunque ne conosca l'URL, ma per poter modificare lo stato delle uscite è necessario essersi precedentemente autenticati attraverso la pagina di login: il Webserver ha infatti un flag che memorizza lo stato di accesso e di autenticazione di un utente. Al primo accesso, per default l'utente non ha i privilegi di superuser.

Analizzando il codice sorgente della pagina delle uscite, si nota che la struttura è di tipo tabellare ed in ogni riga sono presenti un pulsante, un immagine e del testo. Il pulsante di tipo submit viene identificato con un nome numerico compreso tra 0 e 7 che consente alla funzione di gestione sul microcontrollore di sapere quale pulsante è stato premuto e conseguentemente di togglare lo stato dell'uscita corrispondente.

L'immagine viene richiamata con il nome cui viene aggiunta una variabile di stato indicata con %08, %09,…,%15 che varrà "1" o "0" a seconda che la corrispondente uscita sia accesa o spenta. Questa distinzione visualizza l'immagine LED0 o l'immagine LED1.

```
<html>
<body bgcolor="#FFFFFF">
<body>
<link href="stili.css" rel="stylesheet" type="text/css">
<p class="pglog">Uscite digitali</p>
<hr>
<FORM METHOD=GET action=0>
<form>
   <table cellpadding="2" border="1">
        <tr>
               <td><input type=submit name=0 value="<>"></td>
               <td><img src=LED%08.gif></td>
               <td>Uscita 1</td>
        </tr>
        <tr>
               <td><input type=submit name=1 value="<>"></td>
               <td><img src=LED%09.gif></td>
               <td>Uscita 2</td>
        </tr>
        <tr>
               <td><input type=submit name=2 value="<>"></td>
               <td><img src=LED%10.gif></td>
               <td>Uscita 3</td>
        </tr>
        <tr>
               <td><input type=submit name=3 value="<>"></td>
               <td><img src=LED%11.gif></td>
               <td>Uscita 4</td>
        </tr>
        <tr>
               <td><input type=submit name=4 value="<>"></td>
               <td><img src=LED%12.gif></td>
               <td>Uscita 5</td>
        </tr>
        <tr>
               <td><input type=submit name=5 value="<>"></td>
               <td><img src=LED%13.gif></td>
               <td>Uscita 6</td>
        </tr>
        <tr>
               <td><input type=submit name=6 value="<>"></td>
               <td><img src=LED%14.gif></td>
               <td>Uscita 7</td>
        </tr>
        <tr>
               <td><input type=submit name=7 value="<>"></td>
               <td><img src=LED%15.gif></td>
               <td>Uscita 8</td>
        </tr>
   </table>
</form>
</body>
</html>
```

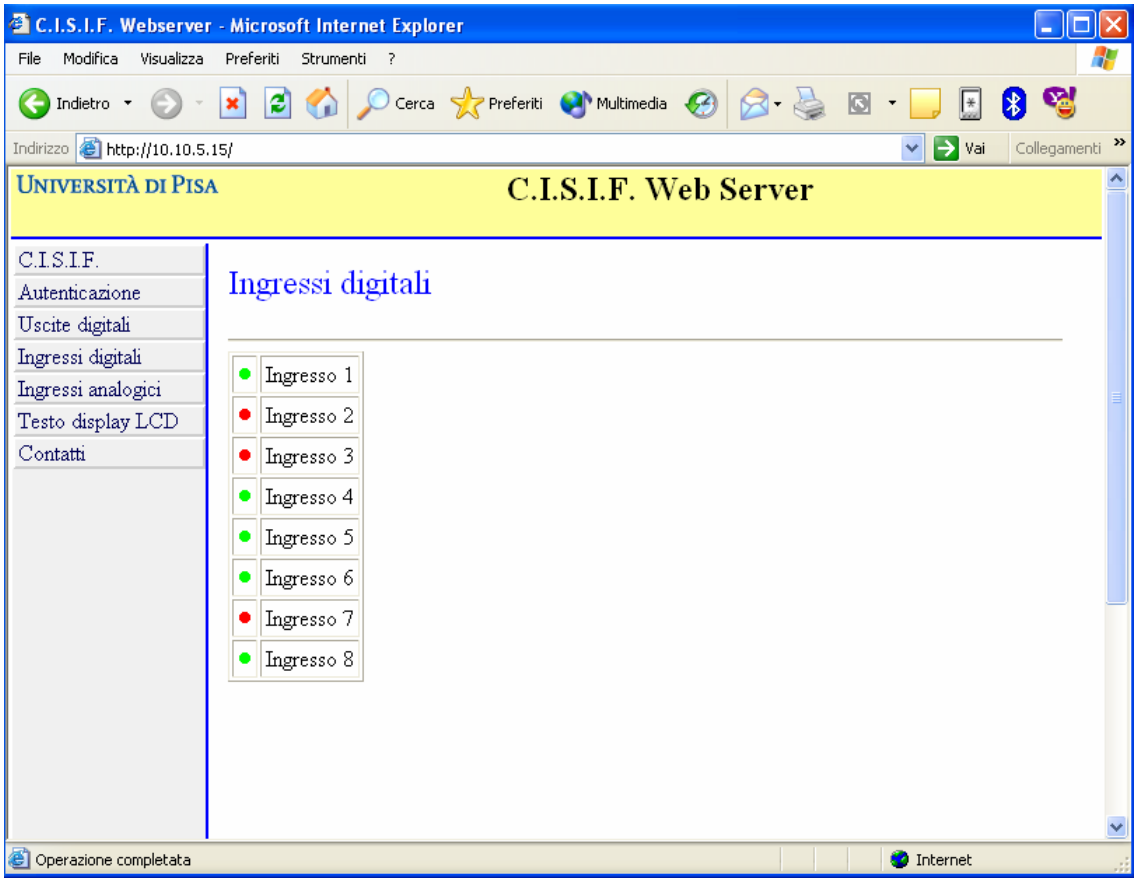
Codice della pagina delle uscite digitali

La pagina degli ingressi digitali

Il Webserver controlla lo stato di otto ingressi digitali e lo visualizza attraverso una pagina dedicata e visibile di seguito.

Questa pagina, dovendo rappresentare lo stato degli ingressi in tempo reale (o quasi) viene creata in modo dinamico ed aggiornata ogni x secondi, dove x è impostabile a programma tra 1 e 10 secondi.

Come per la pagina delle uscite digitali, la rappresentazione dell'ingresso viene effettuata con delle immagini: i cerchietti verdi o rossi simulano la presenza di LED e segnalano lo stato di ogni singolo ingresso (led verde = ingresso ad alto livello, led rosso = ingresso a basso livello). Modificare il colore in rapporto allo stato dell'ingresso oppure sostituire l'immagine dei LED con altra diversa non costituisce problematiche particolari.



Pagina degli ingressi digitali

Dato che chiunque acceda a questa pagina non può alterare lo stato degli ingressi o comunque di qualsiasi altro parametro di funzionamento del Webserver, non è obbligatorio possedere i privilegi di superuser per monitorarla da remoto.

Le label relative ai vari canali sono tutte modificabili da sorgente e nella visualizzazione seguente sono indicative del numero dell'ingresso che si va a visualizzare.
Da un'analisi generale del codice sorgente, si nota subito la direttiva "refresh" impartita al browser, che indica che ogni x secondi (dove x è specificato con l'attributo "content") la pagina dovrà essere riletta e quindi aggiornata in modo dinamico.
I dati sono contenuti in una tabella dove nella colonna di sinistra vengono visualizzati i LED, mentre nella colonna di destra è presente una label che il programmatore può modificare in fase di realizzazione delle pagine web.
Come per la pagina delle uscite digitali, ogni ingresso viene rappresentato da un'immagine che simula la presenza di un LED: le due immagini hanno nome LED0.gif e LED1.gif e corrispondono al LED rosso ed al LED verde. Per inserirle dinamicamente, nella pagina chiamata INDIG.CGI i loro nomi sono seguiti da %00, %01,…,%07: in fase di pubblicazione, il programma sostituisce a queste variabili il numero "0" oppure il numero "1" in funzione dello stato dell'ingresso corrispondente.
Per modificare la rappresentazione dell'ingresso (variazione di colore o di dimensioni e forma dei LED) è sufficiente creare altre immagini ".GIF" e nominarle xxxx0.gif e xxxx1.gif dopo averle inserite nella directory di tutte le pagine web.

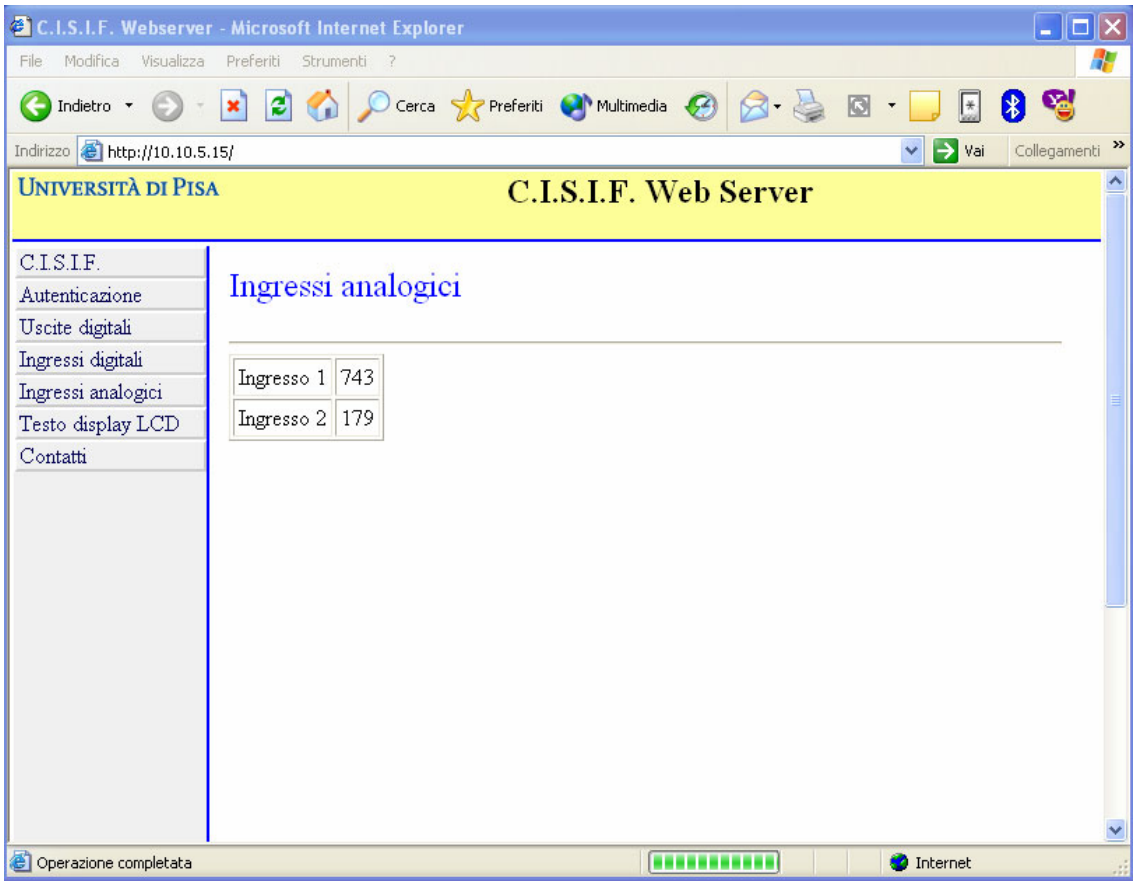
```
<html>
<meta http-equiv="refresh" content="3">
<link href="stili.css" rel="stylesheet" type="text/css">
<body>
<p class="pglog">Ingressi digitali</p>
<hr>
<table cellpadding="3" border="1">
	<tr>
		<td><img src=LED%00.gif></td>
		<td>Ingresso 1</td>
	</tr>
	<tr>
		<td><img src=LED%01.gif></td>
		<td>Ingresso 2</td>
	</tr>
	<tr>
		<td><img src=LED%02.gif></td>
		<td>Ingresso 3</td>
	</tr>
	<tr>
		<td><img src=LED%03.gif></td>
		<td>Ingresso 4</td>
	</tr>
	<tr>
		<td><img src=LED%04.gif></td>
		<td>Ingresso 5</td>
	</tr>
	<tr>
		<td><img src=LED%05.gif></td>
		<td>Ingresso 6</td>
	</tr>
	<tr>
		<td><img src=LED%06.gif></td>
		<td>Ingresso 7</td>
	</tr>
	<tr>
		<td><img src=LED%07.gif></td>
		<td>Ingresso 8</td>
	</tr>
</table>
</body>
</html>
```

Codice della pagina degli ingressi digitali

La pagina degli ingressi analogici

Il microcontrollore impiegato nel progetto del Webserver dispone di alcuni ingressi analogici e tra questi ne sono stati scelti due compatibilmente con le altre periferiche collegate (ad esempio la memoria EEPROM seriale, il display LCD, l'interfaccia RS232 e l'interfaccia di rete).

I due convertitori analogico/digitale impiegati hanno una risoluzione di 10 bit (1.024 valori possibili) e riferimento a 5 volt. Il valore visualizzato è quindi, senza operazioni di calcolo, compreso tra 0 e 1.023 in corrispondenza di un ingresso in tensione compreso tra 0 e 5 volt.



Pagina degli ingressi analogici

La pagina degli ingressi analogici proposta è molto semplice da realizzare, in quanto non effettua calcoli sul valore mostrato, ma è possibile inserire una piccola parte in codice Javascript per normalizzare il numero visualizzato nell'unità fisica voluta. Per esempio, è possibile riportare esattamente il valore della tensione di ingresso al

convertitore in passi da 20mV dividendo il valore passato dalla variabile per 200 e imponendo almeno due cifre decimali (verrà mostrato un valore numerico compreso tra 0,00 e 5,01 volt).

Le label “Ingresso 1” ed “Ingresso 2” sono modificabili direttamente sulla pagina html.

Come per la pagina degli ingressi digitali, anche in questa è richiesto un aggiornamento costante dei dati letti e visualizzati.

All’analisi del codice sorgente, si nota infatti che la prima direttiva per il browser è proprio il “refresh” con tempo di 1 secondo.

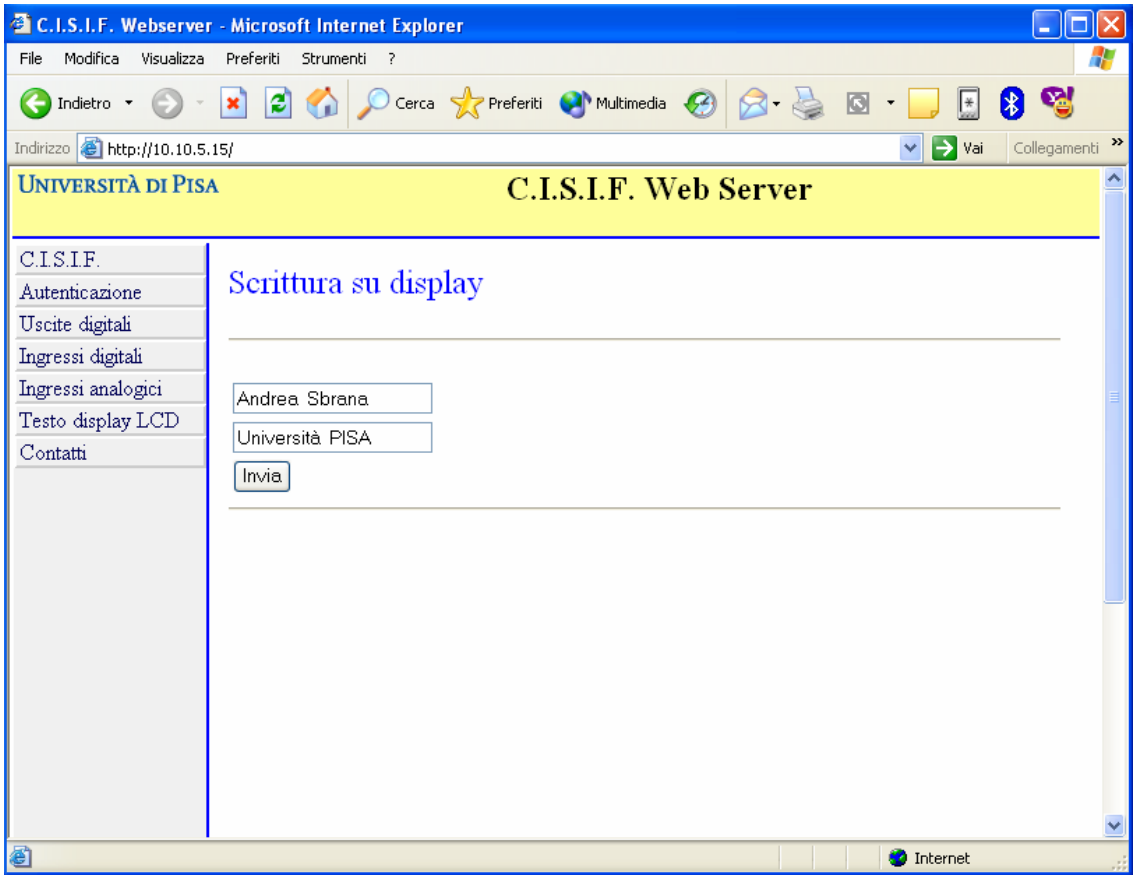
Per completezza, viene riportato di seguito anche il foglio di stile che è stato impiegato per tutte le pagine html presentate fino ad adesso.

```
<html>
<meta http-equiv="refresh" content="1">
<link href="stili.css" rel="stylesheet" type="text/css">
<body>
<p class="pglog">Ingressi analogici</p>
<hr>
<table cellpadding="3" border="1">
	<tr>
		<td>Ingresso 1</td>
		<td>%16</td>
	</tr>
	<tr>
		<td>Ingresso 2</td>
		<td>%17</td>
	</tr>
</table>
</body>
</html>
```

Codice della pagina degli ingressi analogici

La pagina di scrittura su display LCD

Il circuito Webserver prevede il collegamento ad un display LCD da 2 righe e 16 colonne. Tale display dà inizialmente dei messaggi di servizio come ad esempio se sta lavorando il DHCP oppure indica l'indirizzo IP impostato. Dopo la fase iniziale, il display è a disposizione del programmatore e/o dell'utente del sistema per visualizzare messaggi di testo (massimo 32 caratteri su due righe) inviati attraverso un'interfaccia web la cui grafica viene proposta di seguito.



Pagina di scrittura su display

Sono presenti due campi di input (uno per riga). In ogni campo è possibile inserire fino ad un massimo di 16 caratteri ASCII stampabili. Ogni volta che viene premuto il pulsante "Invia", il contenuto dei due input viene inviato al display che lo visualizza.
Poiché i caratteri sono 16 per riga, se la lunghezza dell'input immesso non è tale il firmware completa con caratteri "spazio" (carattere "0x20" in ASCII) fino a completare la riga.
Per scrivere sul display, è necessario possedere i diritti di superuser come per la modifica dei canali di output digitali.

Premendo il pulsante “Invia” senza avere immesso alcun carattere nei campi di input, il display si pulisce resettandosi.

L’estensione di questa pagina è “.CGI”.

Analizzando il codice sorgente necessario all’implementazione della pagina che consente la scrittura su display LCD, si nota la creazione di un form con il metodo “GET” e la “action=1”. Il metodo “GET” indica al browser di inviare i dati secondo la sintassi standard direttamente attraverso la URL, mentre il valore di “action” verrà sfruttato dal Webserver per identificare la pagina che ha eseguito una richiesta di modifica dello stato di funzionamento come illustrato per la pagina di autenticazione.

All’interno del form sono presenti due tag di input con l’impostazione della lunghezza massima, il nome ed il tipo.

La lunghezza massima fissata consente di controllare e limitare la stringa inviata dal browser al Webserver, dato che il display LCD ha un numero fisso di caratteri visualizzabili (16x2).

Il nome invece deve essere presente perché il Webserver lo identifica e lo impiega per discriminare il tag di input corrispondente (riga 1 oppure riga 2).

Il tipo utilizzato per questi due tag è quello di testo.

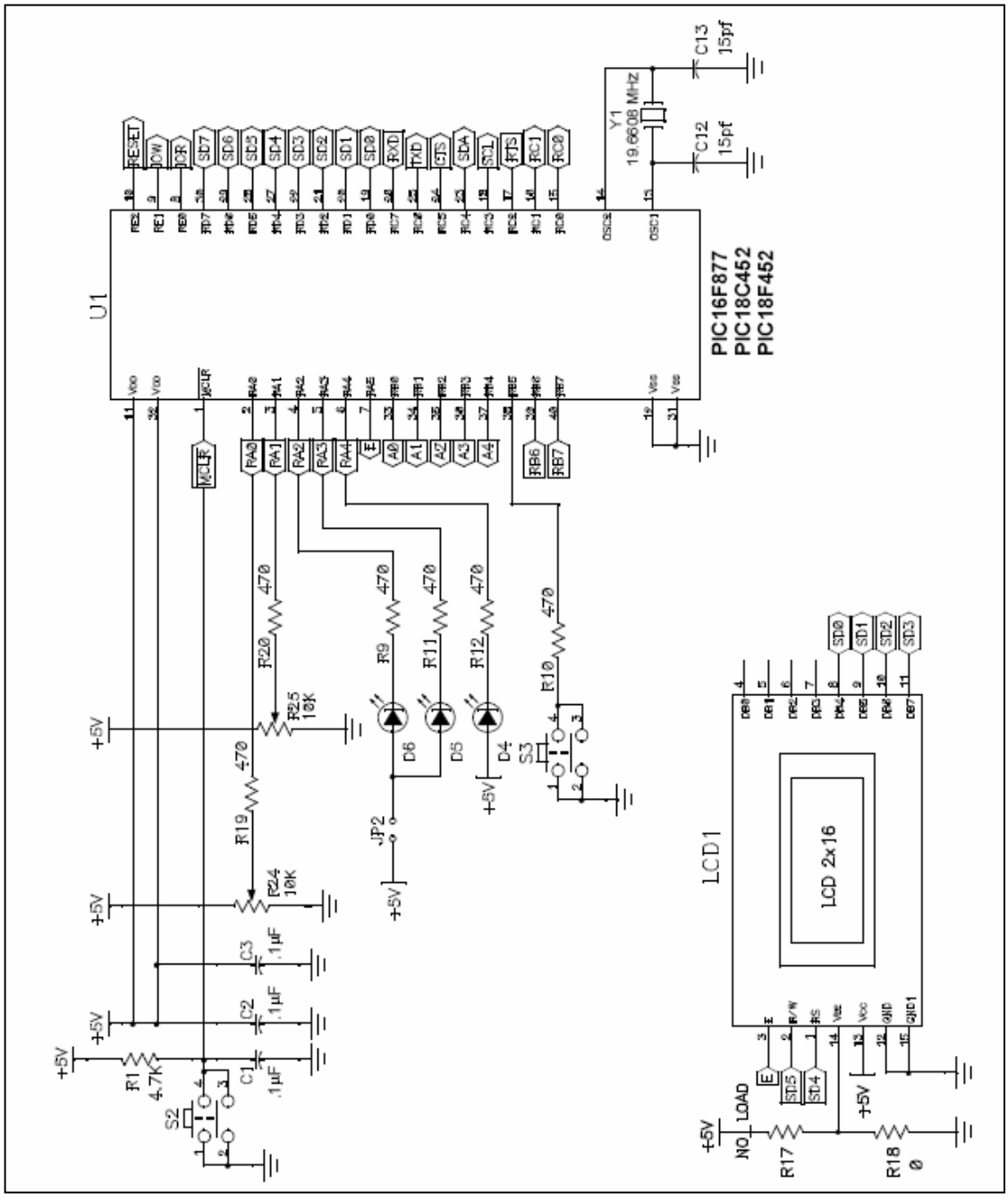
Infine è presente un terzo tag di input, di tipo “sottometti” che consente l’invio dei dati al Webserver.

```
<html>
<body bgcolor="#FFFFFF" >
<link href="stili.css" rel="stylesheet" type="text/css">
<p class="pglog">Scrittura su display</p>
<hr>
<form method=GET action=1>
   <table>
        <tr>
               <td><input type="text" name="D1" size="20" maxlength=16></td>
</tr>
        <tr>
               <td><input type="text" name="D2" size="20" maxlength=16></td>
</tr>
        <tr>
               <td><input type="submit" value="Invia" name="M1"></td>
</tr>
   </table>
<form><hr>
</body>
</html>
```

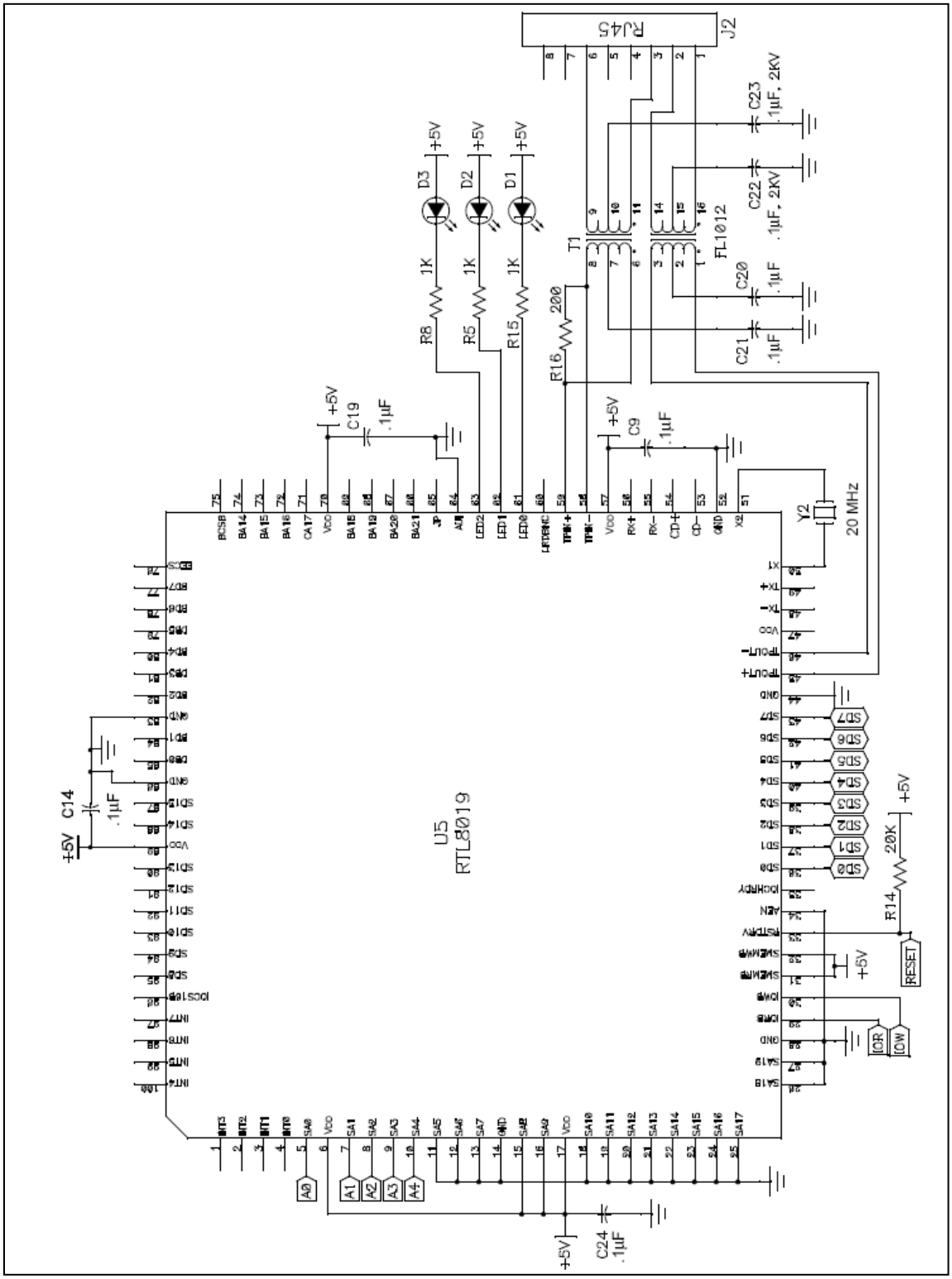
Codice della pagina di scrittura su display LCD

Appendice A: Schema elettrico del Webserver

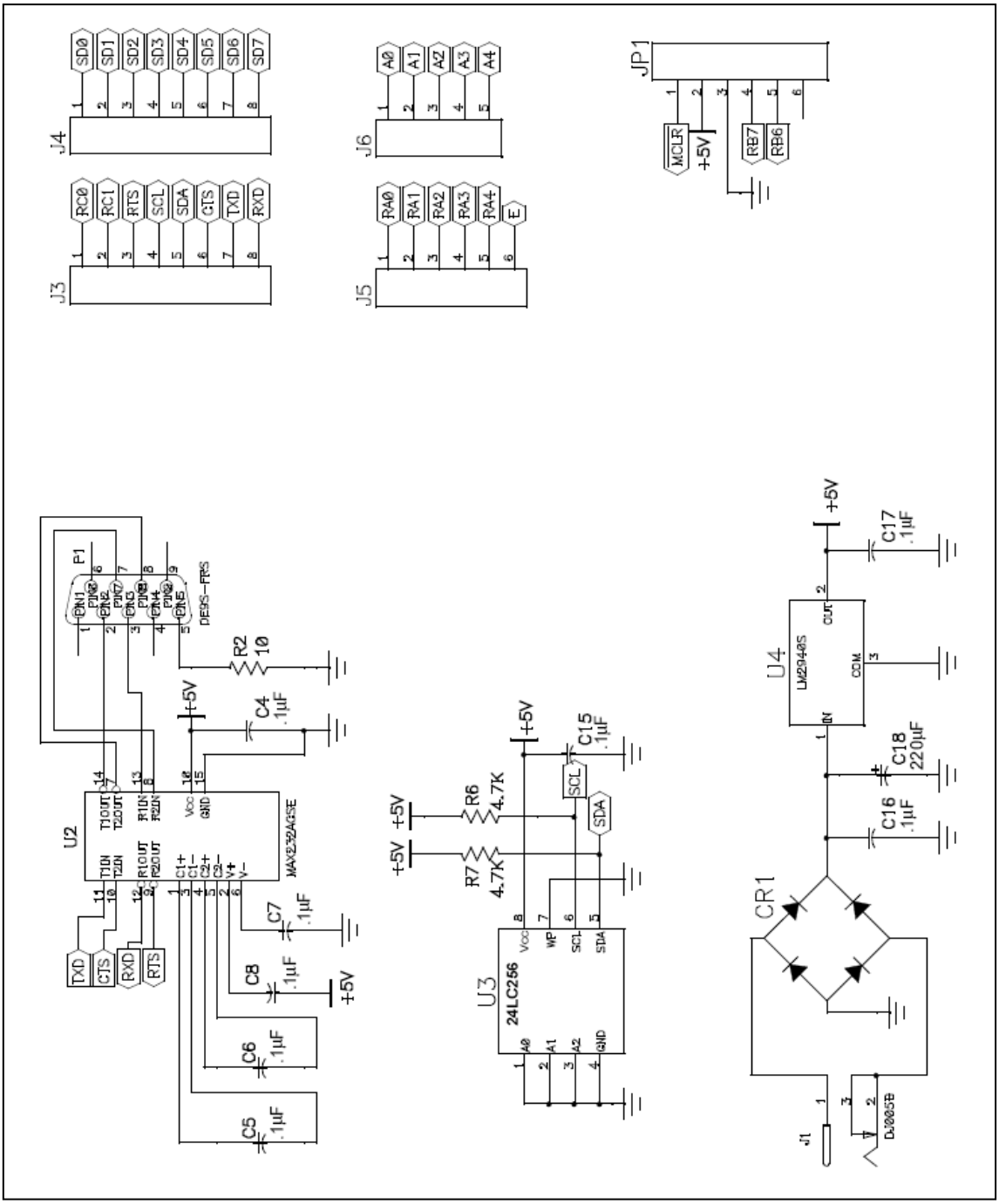
Modulo CPU e display



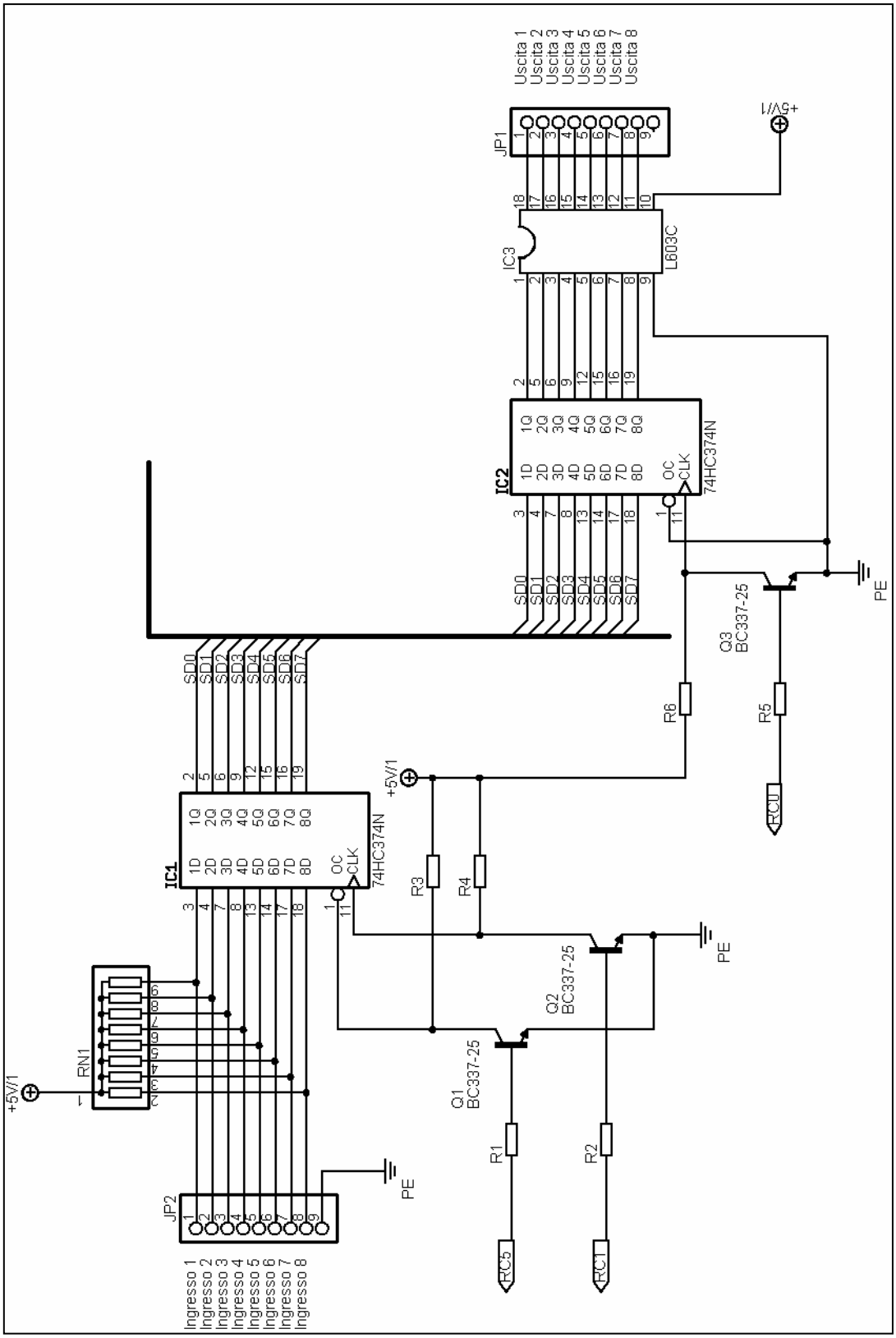
Modulo di interfaccia di rete



Modulo memoria, interfaccia RS232 e alimentazione



Modulo interfaccia di input/output digitale



Appendice B: Caratteristiche del microcontrollore PIC18F452

• Indirizzamento lineare di memoria di programma fino a 32 Kbytes

• Indirizzamento lineare di memoria dati fino a 1.5 Kbytes

• Numero di operazioni fino a 10 MIPs:

- 4 MHz - 10 MHz osc./clock input con PLL attivo

• Istruzioni a 16-bit, dati ad 8-bit

• Due livelli di priorità per gli interrupt

• Moltiplicatore hardware 8 x 8 con singolo ciclo

Caratteristiche delle periferiche:

• Corrente di sink/source 25 mA/25 mA

• 3 pin per interrupt esterni

• Timer0: 8-bit/16-bit timer/counter con prescaler programmabile a 8-bit

• Timer1: 16-bit timer/counter

• Timer2: 8-bit timer/counter con registro a 8-bit per il periodo (base dei tempi per PWM)

• Timer3: 16-bit timer/counter

• 2 moduli Capture/Compare/PWM così configurabili:

- Capture input: cattura a 16-bit, con massima risoluzione di 6.25 ns
- Compare input: compara a 16-bit, con massima risoluzione di 100 ns
- PWM output: PWM con risoluzione a 10-bit e max. frequenza PWM 39 kHz

• Modulo Master Synchronous Serial Port (MSSP), con 2 modalità operative:

- 3-wire SPI™ (supporta tutte e 4 le modalità SPI)
- I2C™ in modalità Master e Slave

• Modulo USART indirizzabile:

- Supporta RS-485 e RS-232

• Modulo porta parallela

Caratteristiche analogiche:

• Convertitore A/D 10-bit con:

- Veloce tempo di campionamento
- Conversione possible anche durante lo stato di SLEEP

- Linearità ≤ 1 LSb

• Low Voltage Detection programmabile

- Supporta interrupt da Low Voltage Detection

• Brown-out Reset (BOR) programmabile

Caratteristiche specifiche:

• 100,000 cicli di cancellazione/scrittura sulla memoria di programma FLASH

• 1,000,000 cicli di cancellazione/scrittura sulla memoria dei dati EEPROM

• Durata dei dati in memoria FLASH e EEPROM : > 40 anni

• Auto-riprogrammabile sotto controllo software

• Power-on Reset (POR), Power-up Timer (PWRT) e Oscillator Start-up Timer (OST)

• Watchdog Timer (WDT) con oscillatore RC interno

• Protezione del codice programmabile

• Modalità SLEEP per risparmio energetico

• Oscillatore selezionabile con le seguenti opzioni:

- 4X Phase Lock Loop
- Oscillatore secondario (32 kHz) per orologi

• In-Circuit Serial Programming™ (ICSP™) su due pin

• In-Circuit Debug (ICD) su due pin

Tecnologia CMOS:

• Basso consumo, alta velocità con tecnologia FLASH/EEPROM

• Ampio range di alimentazione (2.0V to 5.5V)

• Range di temperatura esteso per impieghi industriali

• Basso consumo:

- < 1.6 mA tipici @ 5V, 4 MHz
- 25 μA tipici @ 3V, 32 kHz
- < 0.2 μA corrente tipica di standby

Bibliografia

- Kernighan, Brian e Dennis Ritchie, “The C Programming Language”, II ed., Prentice Hall, 1988.
- A.S. Tanebaum, "The Modern Operating Systems", Prentice Hall, 1992
- Hitachi. “LIQUID CRYSTAL CHARACTER DISPLAY MODULES”, March 1994
- Microchip Technology Inc. “18FXX2” – DS39564B, 2002
- M. Avvenuti, G. Cecchetti, “HTML, CSS e Javascript”, SEU, 2003
- Microchip Technology Inc. “24AA256/24LC256/24FC256” - DS21203M, 2004
- L. Peterson, S. Davie, “RETI DI CALCOLATORI”, Apogeo, 2004
- Microchip Technology Inc. “24AA512/24LC512/24FC512” - DS21754F, 2005
- Microchip Technology Inc. “MPLAB® C18 C COMPILER LIBRARIES” - DS51297E, 2005
- RFC 826 Ethernet Address Resolution Protocol
- RFC 791 Internet Protocol
- RFC 792 Internet Control Message Protocol
- RFC 793 Transmission Control Protocol
- RFC 959 File Transfer Protocol
- RFC 1866 HyperText Markup Language
- RFC 2616 HyperText Transfer Protocol
- RFC 1541 Dynamic Host Configuration Protocol