

Mensile di elettronica applicata, attualità scientifica, novità tecnologiche.

€ 5,00

Elettronica In

www.elettronica.in.it

oltre l'elettronica

Anno XV - n. 143
Dicembre 2009 /
Gennaio 2010

Alla scoperta delle FUEL CELL

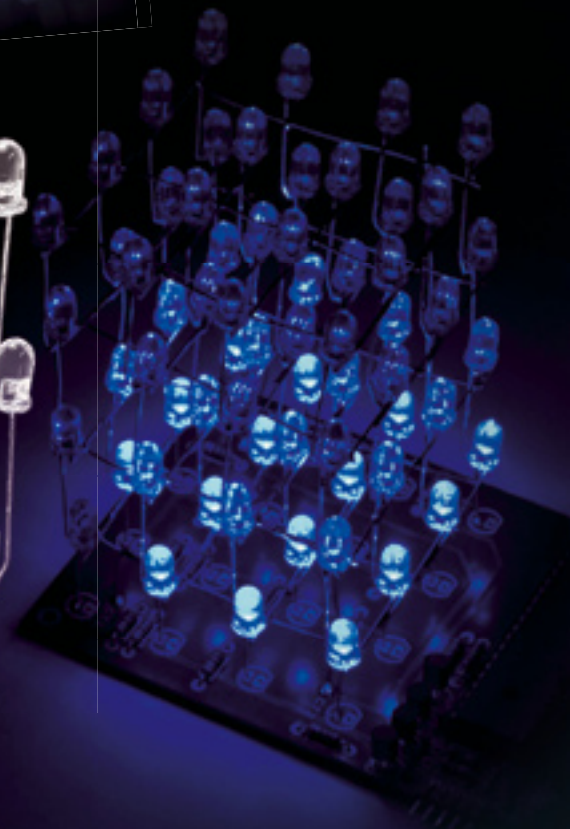


SPIDERIN costruiamo un Robot Ragno a 4 zampe

- Misuratore di consumo ZigBee: il software
- Cubo LED 3D
- Ricevitore EnOcean con uscite digitali e TRIAC
- RFID, il firmware
- Doppio lampeggiante con LED di potenza
- Corso FPGA

ESCLUSIVO!
Corso Wireless CPU

Realizza svariati giochi di luce tridimensionali selezionabili mediante pulsanti, che esegue ad una velocità impostabile dall'utente.

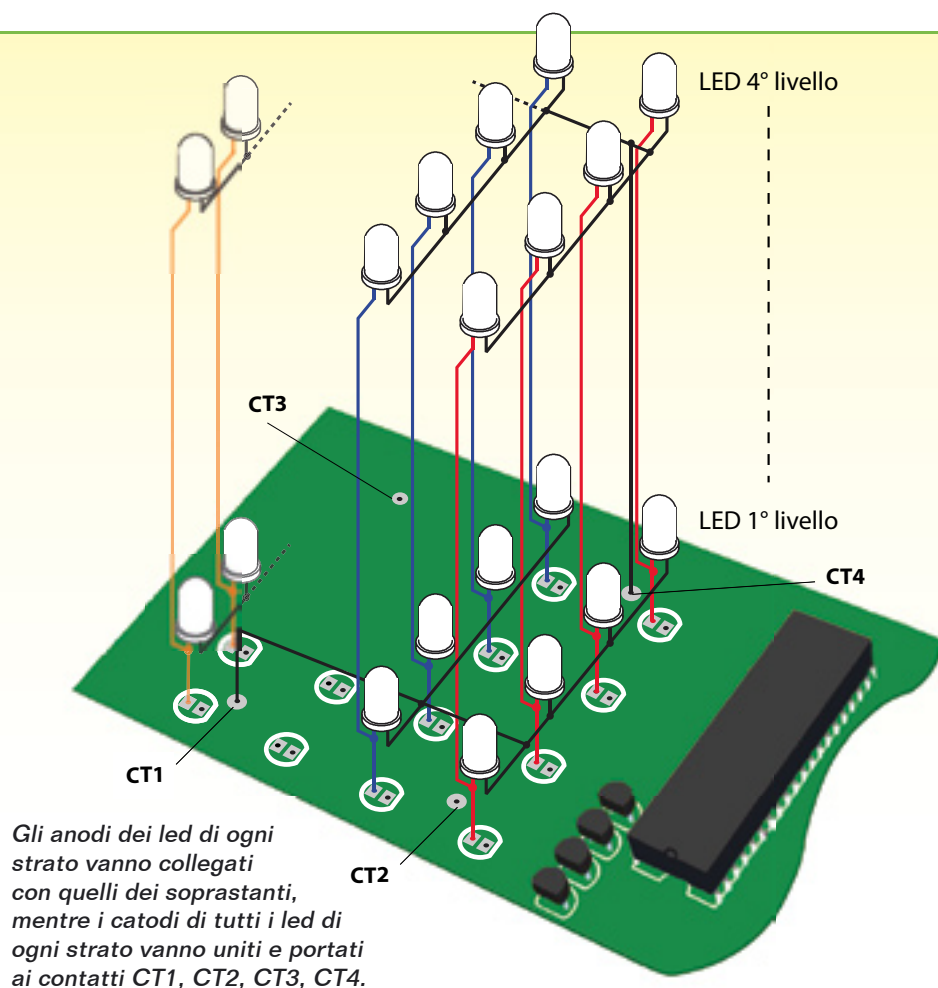


UN CUBO FATTO DI...LED

..... di ALESSANDRO SOTTOCORNOLA

Le festività di fine d'anno sono già arrivate e ancora una volta vi hanno preso di sorpresa? Cercate un effetto, un gadget luminoso da usare per animare una serata, un locale o la vetrina di un negozio? Guardate allora il progetto descritto in queste pagine: la sua originalità rispetto ad altri giochi di luce proposti in passato non mancherà di colpirvi. Infatti dopo tanti gadget luminosi composti da lampadine o led disposti su un circuito stampato a realizzare disegni e scritte, questa

volta abbiamo pensato a qualcosa di un po' più particolare; ci siamo dunque spostati nel tridimensionale, realizzando un gioco di luci formato da diodi luminosi disposti a cubo, capace quindi di comporre animazioni ad effetto spaziale, solido, invece che piano come le classiche barre, frecce o altra forma a led. Sotto l'aspetto circuitale non è una particolare novità, in quanto la sezione che controlla i diodi luminosi non è molto diversa da quella di qualsiasi gadget bidimensionale; la dif-



Gli anodi dei led di ogni strato vanno collegati con quelli dei soprastanti, mentre i catodi di tutti i led di ogni strato vanno uniti e portati ai contatti CT1, CT2, CT3, CT4.

ferenza sta nel montaggio, nella struttura fisica, che prevede la dislocazione di un certo numero di file di led nelle tre dimensioni dello spazio solido. Ma vediamo subito di cosa si tratta, andando a vedere per prima cosa lo schema elettrico.

IL CIRCUITO

Il nostro gadget è formato essenzialmente da un gruppo di diodi luminosi organizzati a matrice, che vengono fatti illuminare mediante un'unità di controllo, la quale fornisce gli impulsi di comando secondo un'opportuna sequenza ripetuta ciclicamente. L'unità di controllo è un microcontrollore Microchip programmato per eseguire diverse sequenze di comando, ad una velocità che possiamo regolare manualmente mediante i pulsanti P1 e P2, i quali per-

mettono anche di passare da un gioco di luci all'altro. Il tutto viene alimentato mediante un alimentatore stabilizzato basato sul regolatore integrato 7805, che, partendo dalla tensione applicata ai contatti + e - PWR (ovvero al plug di alimentazione) e passata dal diodo di protezione dall'inversione di polarità D1, ricava 5 volt ben stabilizzati con cui alimenta il micro.

Ma scendiamo un attimo nel dettaglio e vediamo come funziona il blocco visualizzatore, il quale, come appena accennato, è sostanzialmente una matrice di led: il micro è stato programmato in modo da accendere ogni diodo luminoso portando a livello alto due I/O, uno dei quali è una riga, mentre l'altro rappresenta la colonna. Le righe sono 16 e fanno capo alle linee di I/O RC0, RC1, RC2, RC3, RC4, RC5, RC6, RC7



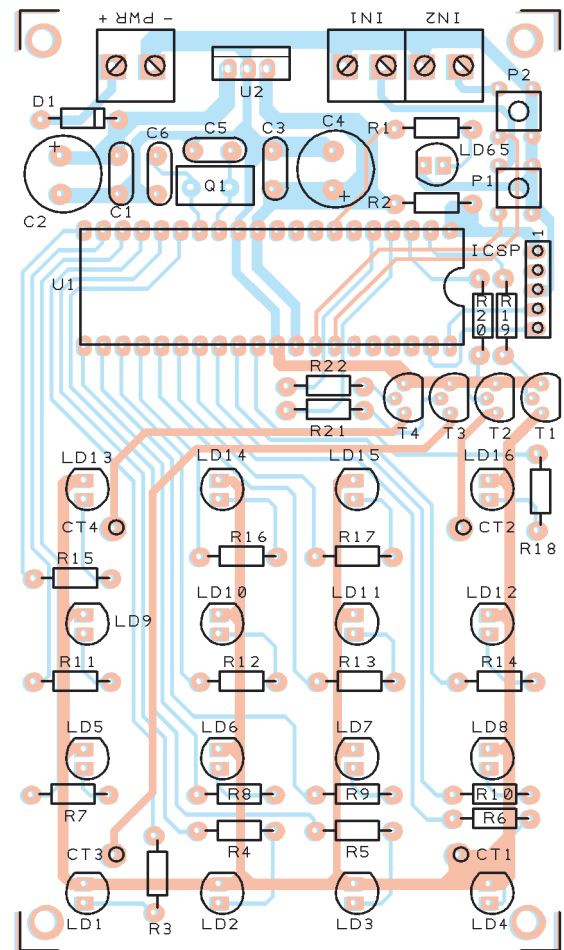
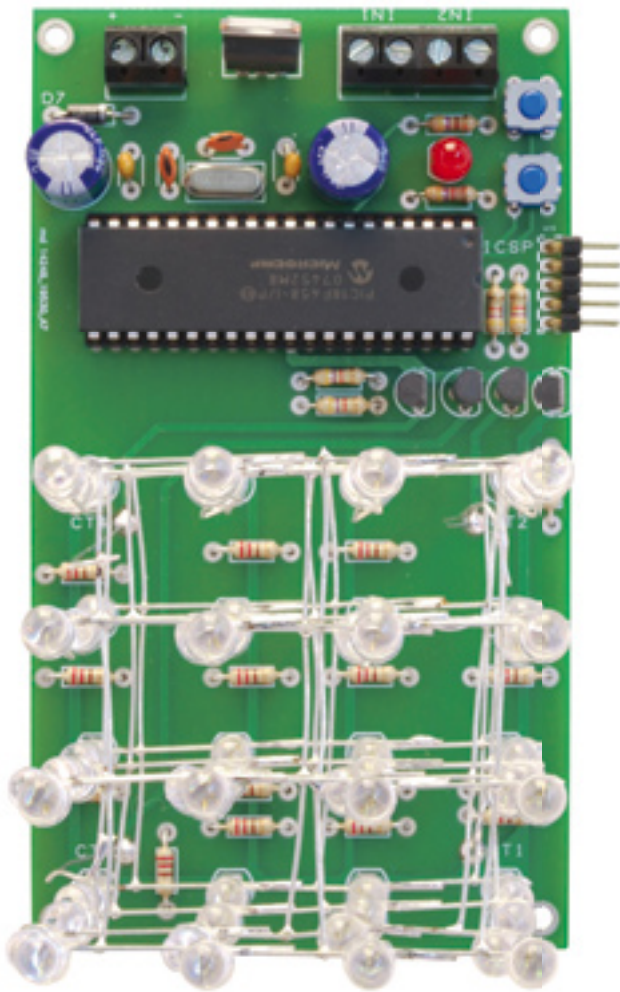
ed RD0, RD1, RD2, RD3, RD4, RD5, RD6 ed RD7; impiegano, insomma, le porte C e D. Quanto alle colonne, sono quattro e corrispondono ad RA0, RA1, RA2, RA3.

Notate che mentre gli I/O corrispondenti alle linee alimentano direttamente i led, quelli delle colonne si servono di un transistor ognuna, montato ad emettitore comune e usato per amplificare in corrente. Questo accorgimento si è reso necessario perché, vista la bassa tensione di alimentazione del micro e quindi la necessità di alimentare i led con livelli logici TTL, la caduta su ogni uscita potrebbe non permettere la corretta accensione dei diodi; in altre parole, sebbene possa sia erogare (modo "source" ossia a livello logico alto) sia assorbire (modo sink, cioè a zero logico) anche 25 mA di corrente, ciascun I/O determina su di sé una caduta che può superare anche 1,5 volt. Se decidiamo di alimentare un led inserendolo tra due linee di I/O del microcontrollore e perdiamo un volt e mezzo su ciascuna di esse, abbiamo appena 2 volt, che bastano appena a polarizzare direttamente un led e a fargli emettere luce.

Per limitare la caduta e riuscire ad applicare ai led una tensione utile più alta, abbiamo optato per dotare almeno una linea di un dispositivo di commutazione che in conduzione presenti una resistenza minore di quella delle uscite del micro; questo dispositivo è un transistor bipolare NPN, che, collegato ad emettitore

abbiamo deciso di utilizzare per il cubo dei led blu, i quali per accendersi richiedono almeno 3 volt; se avessimo usato led rossi o arancioni o gialli, questo problema non ci sarebbe stato ed avremmo potuto comandarli





Elenco Componenti:

R1: 470 ohm
R2: 4,7 kohm
R3÷R18: 220 ohm
R19÷R22: 4,7 kohm
C1, C3: 100 nF multistrato
C2, C4: 470 µF 25 VL elettrolitico
C5, C6: 10 pF ceramico

Q1: Quarzo 20 MHz
U1: PIC18F458-I/P (MF828)
U2: 7805
D1: 1N4007
T1÷T4: BC547
P1, P2: Microswitch
LD1..64: Led 5 mm blu

LD65: led 5 mm rosso

Varie:
- Morsetteria 2 poli (3 pz.)
- Zoccolo 20+20
- Pin strip maschio 5 poli
- Circuito stampato

direttamente con uno logico mediante una riga e chiudendoli a massa mediante una colonna. Bene, chiarito questo aspetto completiamo l'analisi del circuito con le parti del microcontrollore finora trascurate: partiamo dalle linee RB0 ed RB1, inizializzate come input con pull-up interno e usate per leggere i pulsanti di comando dell'unità. Le stesse linee

possono essere utilizzate anche per comandare dall'esterno le funzioni del circuito, mediante livelli logici o transistor montati in open-collector. La programmazione del micro avviene in-circuit mediante le linee PGC (piedino 39) e PGD (40). Infine, la RA5 viene utilizzata per comandare il led di segnalazione dello stato di funzionamento

del circuito. Il microcontrollore funziona con il generatore di clock interno, che si avvale, esternamente, del quarzo Q1 e dei condensatori di avvio C5 e C6.

COME FUNZIONA

Il firmware che gira nel microcontrollore prevede la realizzazione di 11 giochi di luce composti sul cubo formato dai

led; ognuno prevede una sequenza ripetuta ciclicamente. Alla prima accensione si parte con il primo dei giochi; premendo i pulsanti è possibile variare la velocità di esecuzione o spostarsi da un gioco all'altro. Dopo l'inizializzazione delle linee di I/O e l'avvio del primo gioco, la cui sequenza viene "pescata" dalla EEPROM interna al micro, il firmware testa ciclicamente le linee RB0 ed RB1 per verificare se viene premuto un tasto, ovvero attivata una delle linee (l'attivazione si ottiene con lo zero logico). Quando avviene una transizione 1/0 logico su dette linee, il firmware controlla la durata del livello basso: se dura meno di 0,5 s, provoca l'aumento o la diminuzione della velocità di esecuzione del gioco di luce correntemente riprodotto, mentre laddove la pressione del tasto sia prolungata oltre 0,5 s, il firmware passa al gioco successivo. Più esattamente, P1, quando è premuto brevemente, permette di velocizzare l'esecuzione del gioco in corso, mentre se viene premuto a lungo fa riprodurre il gioco successivo; se tale pulsante viene azionato durante l'esecuzione dell'ultimo gioco, il circuito inizia ad eseguire il primo. P2, premuto brevemente permette di rallentare l'esecuzione di un gioco, mentre premuto a lungo fa tornare indietro al gioco precedente; se azionato quando è in esecuzione il primo gioco, il circuito passa ad eseguire l'ultimo dei giochi di luce. Se si preme e si rilascia subito un pulsante, perché si desidera modificare la velocità di esecuzione di un gioco, il led LD65 conferma visivamente l'accettazione della richiesta emettendo un lampeggio; premendo e lasciando premuto un tasto per passare da un gioco al successivo o al precedente, bisogna invece

L'aperitivo più cool? **È a base di LED...**

L'evoluzione tecnologica nel settore dei diodi luminosi ha toccato ormai tutti i settori ed è approdata anche nella ristorazione: girando qua e là abbiamo scoperto magnifici e caratteristici bicchieri e secchielli per champagne realizzati in vetro e polycarbonato, che si illuminano di luce colorata. In particolare, i bicchieri dispongono di led colorati posti alla base, comandati da uno speciale circuito molto semplice e che connette loro una pila a bottone quando si versa una



attendere che tale led si accenda e, solo allora, rilasciare il tasto. Nella variazione della velocità, quando si arriva alla massima rapidità di esecuzione prevista, il led LD65 lo segnala emettendo una rapidissima serie di lampeggi; premendo ulteriormente P1, il solito led emette un lungo lampeggio (si accende per circa un secondo e poi si spegne) quin-

di la velocità passa al minimo. Premendo brevemente P2, LD65 emette ancora un lampeggio di 1 secondo e la velocità torna al massimo. Ogni volta che si priva il circuito dell'alimentazione, alla successiva accensione il firmware riparte dal primo gioco di luci.

REALIZZAZIONE PRATICA
Bene, spiegato come è fatto e in



Guarda gli effetti del cubo LED su YouTube!



Vuoi vedere in funzione il nostro cubo magico? Scoprire tutti gli effetti di cui è capace? Nulla di più semplice: collegati alla pagina <http://www.youtube.com/watch?v=lxJ4gWIoG0s>. Potrai anche vedere i video di moltissimi altri nostri progetti proposti in passato.

che modo funziona il cubo a led, bisogna pensare a realizzarlo in pratica; allo scopo dovete per prima cosa preparare il circuito stampato, che è a doppia faccia e richiede un minimo di attenzione. Le tracce del caso le potete scaricare dal nostro sito internet www.elettronica.in.it, quindi stampare su carta da lucido o foglio di acetato per ottenere le pellicole. Trattandosi di una basetta a doppia ramatura, impressionate nel bromografo prima una faccia dello stampato, poi fate un paio di fori per centrare l'altra traccia ed impressionate la seconda faccia; togliete quindi le pellicole e sviluppate, poi procedete all'incisione nella soluzione acida. Completata l'incisione forate lo stampato e iniziate a disporre i componenti lasciando per ultimi i led che comporranno il cubo: sistemate nell'ordine resistenze e diodo D1, zoccolo per il microcontrollore, quarzo, pulsanti e condensatori (dando la precedenza a quelli non polarizzati) quindi i transistor, da orientare tutti con il lato piatto rivolto all'esterno della basetta (dal lato lungo di quest'ultima). Poi montate il led rosso LD65, te-

nendone il lato smussato rivolto all'elettrolitico C4; ora inserite nelle rispettive piazzole il pin-strip a passo 2,54 mm da 5 poli per la programmazione in-circuit e sistemate le morsettiere a passo 5 mm per l'alimentazione e gli eventuali comandi a livello di tensione dall'esterno. A questo punto non vi resta che costruire la matrice di led che dovrà risultare a cubo; nel farlo, ricordate che ogni fila è composta da quattro diodi luminosi, ognuno dei quali ha l'anodo su una riga e il catodo su una colonna, ovvero su uno dei punti siglati CT1, CT2, CT3 e CT4.

Per realizzare la struttura procedete a strati, saldando un led in ognuna delle 16 posizioni previste sullo stampato; per l'esattezza saldate l'anodo di ogni led nella piazzola dello stampato corrispondente all'anodo, ovvero in quella opposta alla piazzola che nel disegno di disposizione componenti è contrassegnata dallo smusso.

Tutti i catodi, piegati a 90°, del primo strato di 16 led, dovete unirli usando i loro stessi terminali e saldarli in modo da realizzare un insieme solido; in un punto, collegherete questi catodi alla piazzola CT1, ovvero alla prima colonna. Fatto ciò iniziate a preparare il secondo strato di led, che dovrà essere realizzato similmente al primo: sull'anodo di ciascuno dei led del primo strato collegate l'anodo di un nuovo led, piegando di 90° il catodo di quest'ultimo. Tutti i catodi del secondo strato dovranno essere collegati insieme e portati con un filo in rame alla piazzola CT2 dello stampato.

Adesso preparate il terzo strato, saldando l'anodo di un led all'anodo di quello che gli si trova sotto e unendo tutti i catodi (piegati ad angolo retto) che poi

collegherete con un unico conduttore alla piazzola siglata CT3. Infine, procedendo in maniera analoga, realizzate l'ultimo strato di 16 led, collegando i catodi alla piazzola CT4.

Riepilogando, ad ogni anodo dello stampato dovete collegare, sviluppati verticalmente, per l'anodo quattro led; orizzontalmente, tutti i catodi di uno strato dovranno essere uniti e collegati ad una piazzola di colonna. Il tutto, stando attenti a non far toccare i fili dei catodi dei vari livelli. Fatto questo non vi resta che verificare se il vostro cubo a led funzionerà: alimentate il circuito e collegare il connettore per l'ICSP e caricate il firmware nel microcontrollore, quindi togliete l'alimentazione e sconnettete il connettore del programmatore. Adesso alimentate nuovamente il circuito: nel giro di pochi istanti dovrebbe avviarsi l'esecuzione del primo gioco di luci.

Se qualcosa non va, staccate l'alimentatore e verificate i collegamenti dei led e in particolare modo quelli dei catodi che vanno alle piazzole CT1, CT2, CT3, CT4. ■



per il MATERIALE

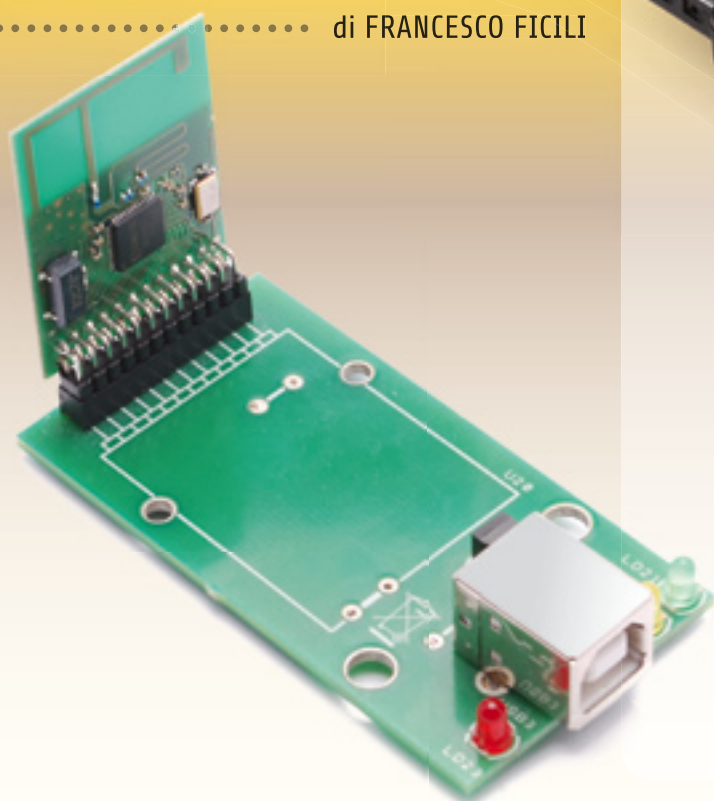
Tutti i componenti utilizzati in questo progetto sono di facile reperibilità. Il master del circuito stampato può essere scaricato dal sito della rivista così come il firmware utilizzato per programmare il microcontrollore PIC18F458. Quest'ultimo è anche disponibile già programmato (cod. MF828) al prezzo di 15,00 Euro, IVA compresa.

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)
Tel: 0331-799775 - Fax: 0331-792287
<http://www.futurashop.it>

Sistema modulare formato da unità periferiche inserite in una rete ZigBee, in grado di rilevare i consumi degli utilizzatori elettrici e trasmetterne i dati ad un concentratore connesso ad un computer via USB. Seconda puntata.



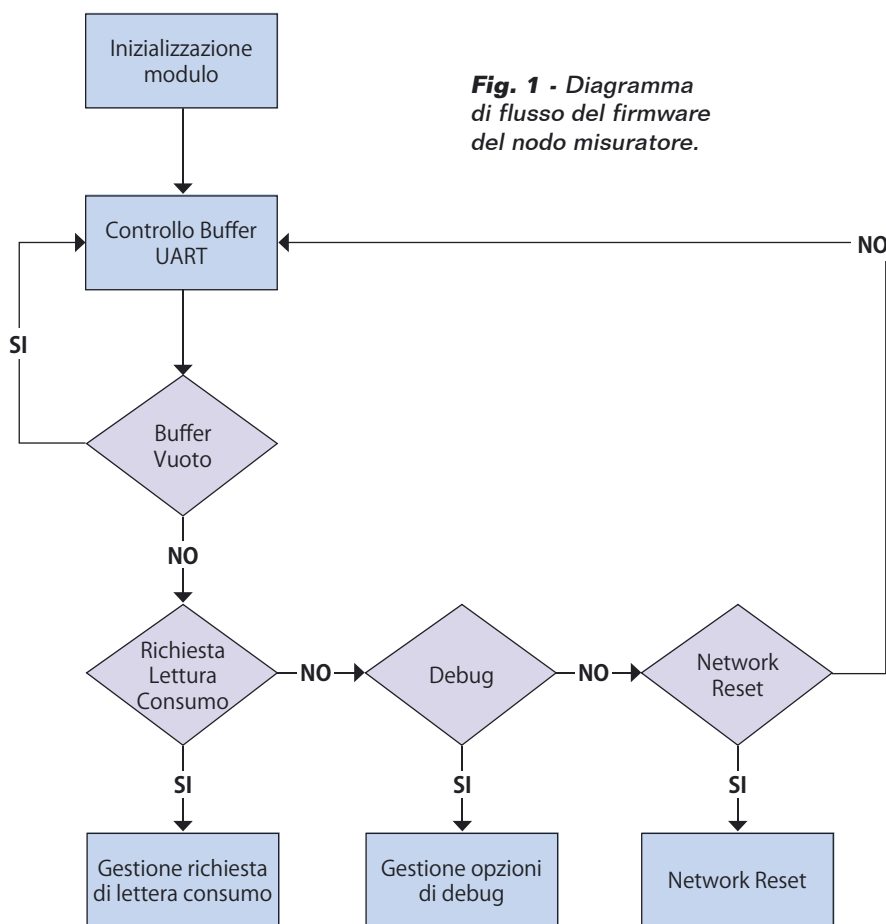
..... di FRANCESCO FICILI



MISURATORE DI CONSUMO ZIGBEE, IL SOFTWARE

Nell'ultimo fascicolo della rivista abbiamo introdotto il primo progetto pratico facente uso dei moduli radio ZigBee della Aurel: un misuratore di consumo di elettricità gestito interamente da dispositivi wireless disposti in modo da realizzare una rete Mesh ZigBee. Nella prima puntata ci siamo preoccupati di spiegare il principio di

funzionamento del sistema e descrivere l'hardware con il quale sono stati implementati i nodi misuratori e l'interfaccia tra nodo Coordinator e PC. In questa puntata conclusiva ci occuperemo di analizzare il firmware con cui sono stati programmati i microcontrollori PIC18F2550 presenti nei nodi misuratori, ed il software di interfaccia e gestione



della rete, da installare nel PC dal quale leggeremo e memorizzeremo i dati sul consumo di elettricità dei singoli utilizzatori. Il programma per PC, in particolare, è stato da noi realizzato in Visual Basic 6, facendo uso del componente OCX messo a disposizione dall'Aurel. Ma andiamo subito a vedere il firmware.

IL FIRMWARE

Il firmware che analizzeremo di seguito è relativo al solo microcontrollore PIC18F2550 utilizzato per gestire i nodi misuratori e ciò perché nel modulo Coordinator non abbiamo previsto alcun microcontrollore; in realtà l'FT232RL è una sorta di micro, però il programma che serve a farlo funzionare da convertitore Seriale-USB è già al suo interno. Tale prerogativa ci affranca dall'onere di scrivere un

firmware per gestire la conversione dei dati in arrivo dal Coordinatore ZigBee Aurel in informazioni in formato USB.

Per la programmazione del PIC18F2550, data l'abbondanza di memoria flash a disposizione, si è optato per l'uso di un compilatore C: in particolare, il compilatore C sviluppato dalla Microchip Technology per la programmazione dei PIC18, ossia il C18.

La naturale scelta per l'ambiente di sviluppo è stata MPLab IDE 8.10. La versione del compilatore utilizzata è la *Student Edition*, liberamente scaricabile dal sito web della Microchip. Il compilatore C18 in versione per studenti è praticamente identico a quello in versione commerciale, dal quale differisce solo per l'assenza di alcuni livelli di ottimizzazione del codice. In particolare, per

progetti non eccessivamente complessi tale versione è praticamente equivalente a quella commerciale.

Le funzioni alle quali deve assolvere il microcontrollore (e quindi il firmware) sono essenzialmente la gestione del protocollo di comunicazione di alto livello con il modulo ZigBee Aurel, la conversione del dato relativo al consumo proveniente dal NETBSEM3, più altre funzioni di servizio (come ad esempio la gestione dei led di debug o del link test). In particolare, il firmware dovrà contenere un ciclo principale all'interno del quale viene continuamente letto il buffer d'ingresso della porta seriale, ed in base ai comandi che vengono decodificati, intraprendere le relative procedure.

La struttura di alto livello del codice è stata schematizzata nella **Fig. 1**: come potete vedere, il firmware prevede lo svolgimento di un ciclo infinito all'interno del quale esegue una serie di istruzioni condizionali in base ai dati ricevuti sulla porta seriale (che costituiscono, nel loro insieme, il

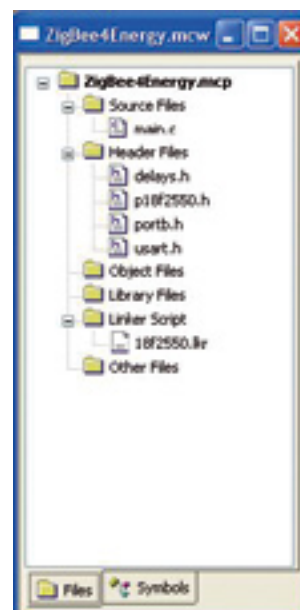


Fig. 2
Project manager di MPLab IDE relativo al progetto del firmware del nodo misuratore.

Listato 1

```
* Indirizzo dispositivo (MY_ADD) : da 1 a 255
*
*****/

#define MY_ADD          1                // Indirizzo dispositivo

/** L E D *****/
#define mInitLED()      TRISBbits.TRISB2 = 0; TRISBbits.TRISB3 = 0;

#define LED_1           PORTBbits.RB2
#define LED_2           PORTBbits.RB3

/** F L O W   C O N T R O L *****/
#define RTS             PORTCbits.RC0
#define CTS             PORTCbits.RC1

/** E N E R G Y   M E T E R *****/
#define CF              PORTAbits.RA0
```

Definizioni degli I/O del microcontrollore.

protocollo di comunicazione). Iniziamo adesso ad analizzare il codice sviluppato per l'implementazione del firmware. Nella **Fig. 2** è rappresentata la finestra del project manager di MPLab IDE, relativa al progetto del nodo misuratore. Oltre al file sorgente *main.c*, che costituisce il nocciolo del firmware, sono state utilizzate alcune librerie standard del C18, come la *delays.h* (per la generazione di ritardi software), la *usart.h* (per la gestione della porta USART) e la *portb.h* (solo per la gestione della rete di pull-up della porta B), oltre naturalmente al file delle definizioni specifiche del micro, *p18f2550.h*.

Andiamo adesso ad analizzare le porzioni più importanti del codice, partendo dalle definizioni iniziali. Per prima cosa sono stati definiti come costanti (utilizzando nomi mnemonici) gli I/O del micro utilizzati per le varie periferiche hardware.

La definizione di queste costanti, a fronte di un piccolo sforzo iniziale, contribuisce notevolmente a rendere più leggibile e strutturato il codice; ciò va a beneficio

del programmatore, in particolar modo quando deve operare su progetti destinati a subire, nel tempo, molte revisioni.

Gli I/O utilizzati sono relativi ai 2 led di debug (RB2 ed RB3), ai pin RTS e CTS (RC0 ed RC1) per il controllo di flusso hardware ed al pin CF (RA0) che viene collegato all'uscita del misuratore di consumo. Il codice utilizzato è visibile nel **Listato 1**.

Viene inoltre definita un'ulteriore costante, denominata MY_ADD, che rappresenta l'indirizzo del PIC del nodo misuratore. La sua funzione è quella di semplificare il riconoscimento del dispositivo via software. Comprendete ciò sapendo che nello standard ZigBee ogni dispositivo è naturalmente associato a due indirizzi che lo identificano univocamente all'interno di una rete: il MAC address e lo short address. L'indirizzo MAC nelle reti ZigBee Aurel riveste scarsa importanza, in quanto gli instradamenti e l'identificazione dei dispositivi vengono entrambi gestiti tramite l'indirizzo short. Il problema sta nel fatto che l'indirizzo short

Listato 2

```
void GetPOWER(void)
{
    unsigned int j;
    unsigned char old_CF;
    unsigned char length,i;

    energy_counter = 1;

    old_CF = CF;
    Delay100TCYx(2);

    LED_1 = 1;

    // Effettua la misura su un periodo di 4 secondi
    for (j=0;j<20000;j++)
    {
        if (CF != old_CF)
        {
            energy_counter++;

            old_CF = CF;
            Delay100TCYx(2);
        }

        LED_1 = 0;

        energy_counter = energy_counter/4;

        /* Set Buffer for trasmission */
        length = 14;
        OutputBuffer[0] = 'S';
        OutputBuffer[1] = 'N';
        OutputBuffer[2] = 'T';
        OutputBuffer[3] = '.';
        OutputBuffer[4] = '0';
        OutputBuffer[5] = '0';
        OutputBuffer[6] = '0';
        OutputBuffer[7] = '0';
        OutputBuffer[8] = '.';
        OutputBuffer[9] = '0';
        OutputBuffer[10] = '2';
        OutputBuffer[11] = '.';
        OutputBuffer[12] = MY_ADD;
        OutputBuffer[13] = energy_counter;
        OutputBuffer[14] = '.';

        for (i=0;i<(length+1);i++)
        {
            CTS = 0;
            while (RTS == 1);
            WriteUSART(OutputBuffer[i]);
            while (BusyUSART());
        }
        CTS = 1;
    }
}
```

La funzione GetPOWER.

è assegnato ai vari router dal coordinator in maniera abbastanza arbitraria e i valori non sono assolutamente progressivi, ma seguono un algoritmo proprio del protocollo. Dover identificare in questa maniera un dispositivo all'interno della rete è un'operazione sicuramente possibile, ma richiederebbe procedimenti software molto complessi. Se però etichettiamo, inserendo come costanti all'interno del codice, i singoli PIC dei nodi e facciamo viaggiare queste "etichette" insieme ai pacchetti dati, sapremo sempre con sicurezza da dove provengono le informazioni. Per fare un esempio pratico,

Il frigo consuma il 30 % in meno

Due ingegneri londinesi hanno sviluppato eCube, un dispositivo per abbattere i consumi energetici di frigoriferi e congelatori, sia domestici che industriali. Applicandolo al termostato, si possono ridurre i consumi di questi elettrodomestici, che, a livello mondiale, richiedono dal 20 al 33% del consumo di elettricità; conseguentemente, ciò comporterebbe una proporzionale riduzione delle emissioni di CO₂. Secondo i calcoli, se ognuno degli 87

milioni di refrigeratori del Regno Unito fosse dotato di un dispositivo eCube, le emissioni di gas serra sarebbero

ridotte di almeno 2 milioni di tonnellate per anno. Il nostro cubo "magico" misura meno di 8 cm di lato, costa 25 sterline (meno di 37 euro) è adattabile a quasi tutte le tipologie di apparecchi, diminuisce la frequenza dei guasti e quindi riduce la manutenzione ed allunga la vita del frigorifero, riducendo nel contempo la rumorosità nell'arco della giornata. Tutto questo è reso possibile dal principio d'azione di eCube, che si

comprende considerando che il sensore del termostato misura la temperatura dell'aria circolante nel frigorifero; con questo sistema, in un'ora si compiono, di norma, dai 12 ai 20 cicli di refrigerazione (attacco del compressore quando la temperatura supera la soglia massima e distacco al di sotto della soglia minima). Il fatto è che in realtà i cibi contenuti nel frigorifero si scaldano o si raffreddano più lentamente dell'aria, cioè hanno maggiore inerzia termica, quindi non servirebbe attaccare in continuazione il compressore. Per



Protocollo di comunicazione

Il protocollo di alto livello utilizzato per la comunicazione tra l'interfaccia software ed i nodi misuratori è molto semplice e costituito da pochi comandi simbolici con codifica ASCII su due byte. Come accennato in precedenza la

struttura utilizzata è per sua natura facilmente espandibile, in modo da rendere semplice l'eventuale aggiunta di nuove funzionalità. Nella tabella sono elencati tutti i comandi del protocollo, ognuno con la relativa descrizione.

Descrizione del protocollo di comunicazione.

Comando	Significato	Effetto del comando sul nodo remoto
LT	LED Test on/off (test 1)	Effettua il cambio di stato (toggle) sul LED1 della scheda.
LS	LED Test Lampeggio (test 2)	Effettua il lampeggio del LED2 della scheda.
ET	Eco Test	Effettua l'eco test, ritornando verso il coordinator un pacchetto dati con Payload "EE".
PM	Power Measure	Effettua la misura del consumo energetico e ritorna verso il coordinator un pacchetto dati contenente il MY_ADD e il dato sul consumo.
NR	Network Reset	Effettua il full-reset via software del radiomodem del nodo misuratore al quale viene inviato il pacchetto.
BT	Battery Test	Effettua il controllo del livello batteria e ritorna il dato verso il coordinator.
FW	Firmware	Ritorna verso il coordinator la major e minor revisione del firmware installato nel nodo misuratore.

supponendo di avere dieci lampade di cui siamo interessati a monitorare i consumi, dovremo generare dieci firmware che verranno etichettati con MY_ADD da 1 a 10. Possiamo quindi etichettare fisicamente i dieci dispositivi con gli stessi numeri e poi associare a livello software l'etichetta (che viaggerà insieme ai dati) al nome simbolico che vogliamo associare all'oggetto del monitoraggio (ad esempio 1 corrisponderà a "lampada soggiorno", 2 a "lampada bagno" e via di seguito). La decisione di intraprendere questa strada impone la scelta di determinate strutture dati e di un determinato protocollo, e costituisce un ottimo esempio di come effettuare con chiarezza determinate scelte iniziali contribuisca ad accorciare i tempi di sviluppo.

Analizziamo adesso come sono state implementate a livello firmware le varie funzioni che i

avvicinare il funzionamento del frigo o congelatore alle esigenze dei cibi, bisognerebbe che il sensore del termostato fosse posto in un materiale che si adegua alla temperatura come i cibi: la cera, per esempio. Il dispositivo eCube è infatti un cubo cavo di cera da applicare dove si infila il sensore del termostato, in modo che quest'ultimo senta variare la temperatura più lentamente, come varierebbe nei cibi. Ciò permette una riduzione dei cicli fino all'80 %, anche se ognuno dura più a lungo che in condizioni normali; ciò basta a garantire un certo risparmio e un'usura minore. Nei test condotti il sistema si è dimostrato efficace soprattutto nei grandi frigoriferi da ristorazione e catering e nei banchi industriali sempre aperti dei supermercati, ma anche in quello di casa consente risparmi interessanti.

nodi misuratori devono eseguire, partendo dalla GetPOWER, utilizzata per la lettura del consumo elettrico. Come illustrato già nella prima puntata, il misuratore di consumo NETBSEM3 agisce come un convertitore potenza/frequenza, restituendo una forma d'onda la cui frequenza è direttamente proporzionale (con fattore di proporzionalità pari ad 1) al consumo di potenza istantaneo in watt. Naturalmente per ottenere dati più precisi è possibile effettuare la lettura su un periodo di tempo più esteso e poi calcolarne la media.

Nella funzione GetPOWER è stato utilizzato un periodo di 4 secondi, con un periodo di campionamento di 200 microsecondi (Fs pari a 5 kHz) e poi il risultato viene diviso per 4, in modo da ottenere la potenza istantanea in watt.

La funzione è costituita da due parti: la prima serve ad effettua-

Listato 3

```
while(1)
{
    while(!DataRdyUSART()); // Resta in attesa dei dati dal PC

    getsUSART(data,7); // Legge 7 byte dal buffer di ricezione
    CTS = 1; // Flow control: disabilita trasmissione dati

    /* Link Test_1 */
    if ((data[4] == 'L') && (data[5] == 'T'))
    {
        LED_1 = !LED_1; // Sequenza Led 1
    }

    /* Link Test_2 */
    if ((data[4] == 'L') && (data[5] == 'S'))
    {
        LEDSeq1(); // Sequenza Led 2
    }

    /* Network Reset */
    if ((data[4] == 'N') && (data[5] == 'R'))
    {
        NetworkReset();
        Delay10KTCYx(20);
        LEDSeq4(); // Sequenza Led 4
        _asm RESET _endasm // Inline assembly
    }

    /* Echo Test */
    if ((data[4] == 'E') && (data[5] == 'T'))
    {
        EchoTest();
        CTS = 0; // Flow control: abilita trasmissione dati
        while(!DataRdyUSART()); // Resta in attesa dell'ack del modem
        getsUSART(data,4); // Legge la risposta del modem
        CTS = 1; // Flow control: disabilita trasmissione dati
        LEDSeq2(); // Sequenza Led 2
    }

    /* Power Measure */
    if ((data[4] == 'P') && (data[5] == 'M'))
    {
        GetPOWER();
        CTS = 0; // Flow control: abilita trasmissione dati
        while(!DataRdyUSART()); // Resta in attesa dell'ack del modem
        getsUSART(data,4); // Legge la risposta del modem
        CTS = 1; // Flow control: disabilita trasmissione dati
        LEDSeq3(); // Sequenza Led 2
    }

    /* Firmware Version */
    if ((data[4] == 'F') && (data[5] == 'W'))
    {
        SendFW();
        CTS = 0; // Flow control: abilita trasmissione dati
        while(!DataRdyUSART()); // Resta in attesa dell'ack del modem
        getsUSART(data,4); // Legge la risposta del modem
        CTS = 1; // Flow control: disabilita trasmissione dati
        LEDSeq5(); // Sequenza Led 2
    }

    /* Battery test */
    if ((data[4] == 'B') && (data[5] == 'T'))
    {
        ReadBattery(); // Interroga il modem per conoscere il livello batteria
        CTS = 0; // Flow control: abilita trasmissione dati
        while(!DataRdyUSART()); // Resta in attesa dell'ack del modem
        getsUSART(data,1); // Legge la risposta del modem
        CTS = 1; // Flow control: disabilita trasmissione dati

        BatteryTest(); // Invia la lettura batteria al PC
        CTS = 0; // Flow control: abilita trasmissione dati
        while(!DataRdyUSART()); // Resta in attesa dell'ack del modem
        getsUSART(data,4); // Legge la risposta del modem
        CTS = 1; // Flow control: disabilita trasmissione dati
        LEDSeq3(); // Sequenza Led 3
    }

    CTS = 0;
} // End while principale
```

Ciclo principale del firmware.

Il software



Fig. 3 - Schermata principale del software di gestione da installare nel PC.

Per la gestione della rete di misuratori di consumo è stato sviluppato un software di gestione ad hoc (la cui schermata principale è riportata nella **Fig. 3**) in grado di interrogare i nodi, verificare la comunicazione, effettuare le associazioni tra i nodi misuratori ed il carico posto sotto monitoraggio energetico e naturalmente salvare i dati per una successiva fase di post processing. Gli strumenti utilizzati per lo sviluppo del software sono il linguaggio di programmazione Visual Basic 6 e l'OCX fornito dall'Aurel. In quest'articolo non ci addentreremo nella descrizione del codice utilizzato per la realizzazione dell'interfaccia, ma ci limiteremo alla descrizione operativa del software. Chi fosse interessato ad una

descrizione più esaustiva sulla realizzazione di software per la gestione di reti tramite l'OCX Aurel può fare riferimento al corso sul protocollo ZigBee pubblicato nei numeri dal 133 al 136 della nostra rivista. Per le istruzioni di installazione e di configurazione del software riferitevi all'apposito riquadro che trovate in queste pagine. Come si vede dalla **Fig. 3**, la schermata principale è divisa in alcune sezioni. Attraverso la sezione Network Management è possibile attivare o disattivare la rete e selezionare la porta COM relativa alla scheda di interfaccia PC-Coordinator. La sezione PAN Control contiene alcune informazioni sulla rete creata (Indirizzo MAC del coordinator, PAN ID, canale RF in uso e indicazione del numero di

dispositivi che compongono la rete, quest'ultima fornita dall'utente) e consente di effettuare lo scan dei nodi misuratori (richiesta della lettura degli energy meter). Le due sezioni LEDs e Link Test permettono di effettuare alcune operazioni di debug, ed alla loro destra è presente una finestra di dialogo che viene utilizzata per visualizzare i dati relativi ai consumi energetici. Oltre alla finestra di dialogo è possibile attivare, tramite il menu a tendina Options/Monitor, una finestra che riporta tutto il traffico dati che avviene sulla seriale emulata, molto utile per comprendere che tipo di messaggi i vari router scambiano con il coordinator e per operazioni di debug. Nella **Fig. 4** è visibile un esempio di ciò. Vediamo adesso come è possibile assegnare dei nomi simbolici ai nodi, che rendano più agevole l'individuazione del carico a cui sono collegati e come sia possibile effettuare uno scan della rete per ottenere le letture dei vari nodi. Supponiamo di voler misurare il consumo di due lampade (che indicheremo genericamente con carico_1 e carico_2) e di disporre dei due nodi misuratori con indirizzi fisici 1 e 2 (i MY_ADD descritti nel paragrafo relativo al firmware). L'interfaccia software permette di effettuare l'associazione tra indirizzi fisici e nomi simbolici; l'associazione si effettua selezionando l'opzione del menu a tendina Options/Update Device Name: comparirà la finestra di dialogo illustrata nella **Fig. 5**, nella quale è possibile inserire l'indirizzo fisico ed il nome simbolico che si desidera associare al nodo in questione (nel-

l'esempio al nodo 1 è stato associato carico_1 ed al nodo 2 carico_2). Per rendere effettiva l'associazione bisogna premere il pulsante OK. Notate che è possibile modificare dinamicamente in ogni momento tutte queste associazioni. Una volta terminata questa fase, i nomi simbolici risulteranno associati agli indirizzi fisici e i vari dati di consumo saranno facilmente riconducibili ai dispositivi reali. A questo punto per effettuare una scansione è sufficiente premere il tasto Scan Network presente all'interno del box PAN Control. Il software inizierà la scansione della rete come illustrato nella **Fig. 6**. Al termine della scansione la finestra di dialogo riporterà i risultati relativi al consumo, insieme all'indicazione della data e dell'ora in cui è stata effettuata la misura, come visibile nella **Fig. 7**. È inoltre possibile effettuare la scansione automatica della rete, con periodo impostabile via soft-



Fig. 4 - Net Monitor.



Fig. 5 - Finestra di associazione indirizzo fisico-nome simbolico.

re la misura della frequenza in ingresso sul pin CF, in modo da ricavare il valore digitale del consumo energetico, mentre nella

seconda parte vengono preparati i dati per l'invio al modulo radio che si occupa di spedirli verso il coordinator.

Per determinare il numero di transizioni (e quindi il dato riguardante la frequenza) vengono utilizzate le due variabili CF e



Fig. 6 - Finestra di dialogo della Scansione della rete.

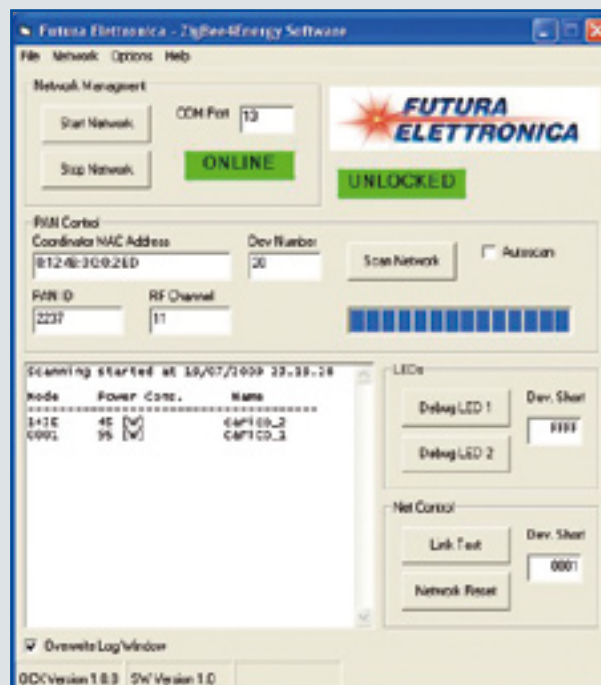


Fig. 7 - Risultato della scansione della rete.

ware. Per impostare il periodo della scansione selezionate, nella solita finestra di dialogo della **Fig. 6**, dal menu a tendina Options/Autoscan Period il periodo desiderato tra 1, 10, 30 e 60 minuti (come illustrato nella **Fig. 8**) e poi attivare la checkbox Autoscan presente all'interno del frame PAN Control. Verrà attivato un timer software contenente una routine che effettua la scansione dell'intera rete all'intervallo prefissato. Il software offre inoltre alcune opzioni di debug, che possono essere utili per verificare il corretto collegamento tra il coordinator ed i router della rete. Le opzioni di debug disponibili sono il controllo dei led di debug dei nodi misuratori (pulsanti Debug LED 1 e Debug LED 2) ed il classico eco test (pulsante Link Test) che richiede l'eco del pacchetto dati inviato al nodo misuratore. Il nodo destinazione di queste operazioni di debug viene selezionato tramite il suo indirizzo short (inserendolo nella textbox Dev. Short); l'operazione può

anche essere eseguita in broadcast, selezionando come indirizzo FFFF. Infine viene offerta la possibilità di visualizzare la lista dei nodi che compongono la rete. L'opzione si trova sotto il menu Options>Show Device list. La **Fig. 9** illustra il caso di una rete formata da cinque dispositivi generici. Installato in un Personal Computer, il software descritto permette di gestire efficacemente una rete di nodi misuratori, consentendo di verificare lo stato dei consumi sia on-demand che in maniera automatica (con salvataggio dei dati), il che permette così anche un fase successiva di post-processing. Una possibile evoluzione del sistema descritto in queste puntate consiste nel modificare il front-end in modo che non sia un eseguibile SW installato su un PC, ma del codice firmware caricato su una scheda embedded con interfaccia ethernet, in modo da avere l'output dei dati, già processati in forma grafica, sulla rete ethernet o in Internet.

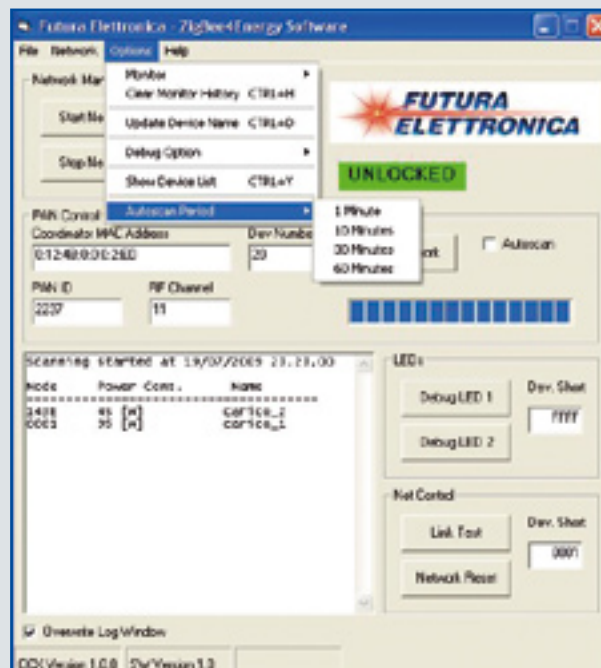


Fig. 8 - Selezione del periodo di scansione automatica della rete.

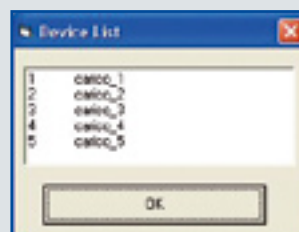


Fig. 9 - Lista dei dispositivi presenti nella rete.

old_CF. Ad ogni ciclo si confronta il valore del pin CF con il valore memorizzato al ciclo precedente (old_CF): se i due valori

sono differenti c'è stata una transizione e viene incrementato il contatore *energy_counter*. Il LED 1 viene acceso e poi spento per

segnalare inizio e fine del ciclo di misura, che dura circa 4 secondi. Al termine del ciclo il valore della variabile *energy_counter* viene

Istruzioni per l'installazione

Prima di installare il software per il monitoraggio dei consumi bisogna preventivamente installare il driver della porta COM virtuale necessaria per l'interfacciamento della scheda Coordinator. I Driver più aggiornati sono presenti sul sito web della FTDI (www.ftdichip.com). Il software da noi proposto è stato testato con successo con la versione VCP 2.04.16, ma anche le versioni più recenti dovrebbero funzionare correttamente. Una volta installato il driver è possibile collegare la scheda coordinator e il dispositivo dovrebbe essere visto sotto le risorse di sistema come USB serial port, con assegnato un dato numero identificativo, secondo quanto illustrato nella **Fig. 10**.

È importante ricordare il numero della porta COM assegnata, in quanto servirà successivamente. A questo punto è possibile installare

il software, utilizzando il pacchetto di installazione scaricabile dal nostro sito www.elettronica.in. Una volta avviato il software ricordatevi di selezionare la porta COM virtuale corrispondente alla USB serial port (nell'esempio COM13), o il software non riuscirà ad interloquire con il coordinator. Infine, per avviare il sistema di monitoraggio, l'ultimo step consiste nel posizionare la rete di nodi misuratori sulle utenze che si desidera monitorare. Utilizzando i nodi misuratori dovete prestare la massima attenzione, in quanto si tratta comunque di dispositivi collegati alla rete elettrica; perciò evitate di toccare il modulo router e curatevi di inserirlo in un contenitore isolato. Nella **Fig. 11** è riportata un'immagine di un nodo completo (misuratore di consumo collegato ad una scheda router).



Fig. 10 - Finestra Gestione Periferiche con evidenziata la COM virtuale aperta per il coordinator.

diviso per 4 in modo da ricavare il valor medio della potenza. La seconda parte serve ad impacchettare il dato ottenuto all'interno del buffer di trasmissione e di inviare i dati al coordinator attraverso la porta seriale; per l'invio dei dati si utilizza il comando SNT previsto dai moduli Aurel, la cui sintassi è la seguente:

SNT:[indirizzo short destinatario]:[numero di byte]:[dati];

Per indirizzare il coordinator si usa l'indirizzo short 0000 (short address di default dei coordinator in tutte le reti ZigBee). Il campo dati è costituito dai byte 12 e 13 e contiene il MY_ADD (cioè l'indirizzo dei nodi misuratori) ed il dato relativo al consumo (energy_counter). Una volta riempito il buffer si procede alla trasmissione utilizzando un ciclo while. È stato implementato anche il controllo di flusso hardware, quindi all'inizio di ogni trasmissione si pone a zero logico la linea CTS (Clear To Send) per segnalare al ricevitore

Fig. 11 - Misuratore di consumo completo di scheda router.

l'intenzione di inviare dei dati e poi si attende che quest'ultimo metta a zero anche la linea RTS (Ready To Send) per indicare la possibilità di effettuare la trasmissione.

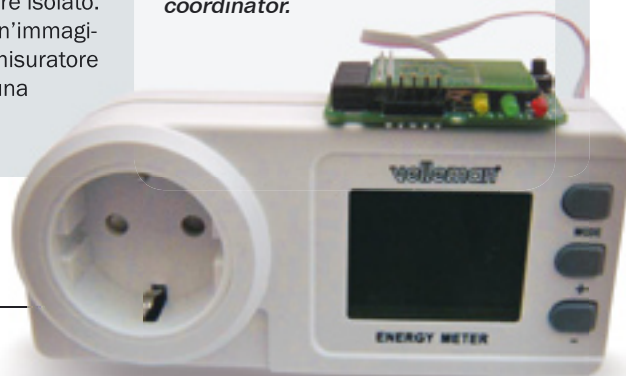
Analizziamo adesso il ciclo principale del firmware, il cui codice completo è riportato nel **Listato 3**, in modo da avere un'idea chiara di come si svolgono la ricezione e la trasmissione dei dati. Come si può notare osservando il listato, la struttura è costituita da un ciclo while infinito. La prima istruzione del ciclo serve ad effettuare una lettura in polling del buffer di ricezione della porta USART (*while(!DataRdyUSART()*). Appena vengono ricevuti dei dati sul buffer, il programma esce dal while e prosegue con la lettura di sette byte, che vengono prelevati

e spostati sul buffer data. Anche per inviare dati dal coordinator verso i router viene utilizzato il comando SNT. L'output di questo comando sulla seriale del nodo remoto indirizzato segue la sintassi illustrata nell'esempio qui di seguito:

MsgType --> 1 byte
• 0X02 --> Generic Radio Msg
Address sorgente --> 2 byte
Lunghezza dei dati --> 1 byte
Dati ricevuti --> #byte

punto e virgola --> 1 byte

Quindi il PIC riceverà sulla sua seriale un pacchetto di 7 byte, dove i dati (nel caso di payload di 2 byte, come previsto dal nostro protocollo di alto livello) si trovano in posizione 4 e 5 (nel

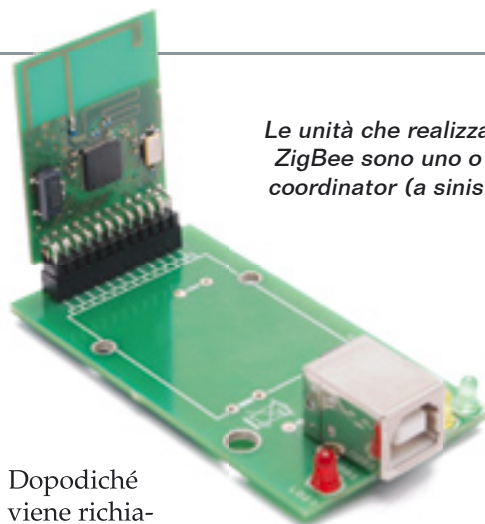


caso il buffer parta dall'indice 0, come nel caso della convenzione usata per l'ANSI C).

A questo punto decodificare il comando ricevuto è un'operazione abbastanza semplice: basta andare ad osservare, con una serie di istruzioni condizionali, il contenuto degli elementi **data[4]** e **data[5]** (payload) del buffer data. Quanto descritto in precedenza è ciò che viene fatto nella porzione di firmware successiva con una serie di istruzioni condizionali *if*. In base al tipo di istruzione decodificata, il firmware intraprende le seguenti azioni:

- debug LED1 (cambia di stato il pin corrispondente al LED1);
- debug LED2 (effettua tre lampeggi del LED2);
- network Reset (invia il comando di reset al modem e resetta il firmware);
- power Measure (effettua una misura di consumo e invia il dato al coordinator);
- battery Test (richiede al modem una lettura del livello batteria e invia il dato al coordinator);
- echo test (invia un pacchetto dati contenente i caratteri "EE" al coordinator);
- firmware (invia la Major e Minor revision del firmware).

Per comprendere meglio come sono eseguite queste operazioni, esaminiamo una campione: prendiamo in considerazione l'istruzione *if* che effettua la lettura del consumo e l'invio del dato al coordinator (vedere il **Listato 3**). A parte le istruzioni di debug dei led, che sono relativamente semplici da capire, tutte le altre seguono più o meno la stessa logica. Per prima cosa viene decodificata l'istruzione "PM" (Power Measure), verificando per mezzo dell'operatore AND che **data[4]** e **data[5]** contengano rispettivamente i caratteri "P" ed "M".



Le unità che realizzano il sistema di comunicazione ZigBee sono uno o più un router (a destra) ed un coordinator (a sinistra) dotato di interfaccia USB.



Dopodiché viene richiamata la funzione **GetPOWER()**, già descritta nella prima parte del paragrafo. Come detto in precedenza, questa funzione esegue una lettura del dato relativo al consumo ed invia il risultato via radio al coordinator. Dopo questa istruzione, che contiene un comando SNT, il radiomodem risponde sulla seriale con un totale di 4 byte. Per prima cosa, se il dato viene correttamente ricevuto e non vengono rilevati errori nei parametri, risponderà con il codice di terminazione "no_error" (0x30), seguito dal punto e virgola (0x3B). Inoltre, se il dato viene correttamente ricevuto dal coordinator, il radiomodem invierà sulla seriale altri due byte, corrispondenti al codice di terminazione "ack_received" (0x38), seguito anch'esso dal punto e virgola (0x3B). Ecco perché a seguito della chiamata alla funzione **GetPOWER()**, il firmware si preoccupa di leggere questi altri 4 byte. Se ciò non avvenisse, questi dati non letti andrebbero ad accumularsi sul buffer di ricezione della USART, andando a corrompere i dati successivi. Dopo la lettura dei byte viene chiamata la funzione **LedSeq3()**, che esegue una sequenza di lampeggio specifica, per evidenziare il successo dell'operazione di lettura. Una volta decodificato il comando, il ciclo riparte dall'inizio, andando nuovamente ad effettuare la lettura in polling del

buffer di ricezione della USART. Questa struttura, in verità molto semplice, permette comunque anche un facile upgrade del codice, in quanto per inserire una nuova funzione del protocollo basta introdurre una nuova istruzione condizionale *if* all'interno del ciclo while principale. Bene, ora che avete visto il firmware, non dovete fare altro che caricarlo nel micro dei router, collegare le unità, installare il software nel PC e mettere in funzione il sistema. ■



per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono di facile reperibilità. Il master del circuito stampato, il firmware per programmare il microcontrollore PIC18F2550 ed il software di gestione per il PC sono scaricabili gratuitamente dal sito della rivista. Il misuratore NETB-SEM3 costa 17,50 Euro mentre i due dispositivi ZigBee dell'Aurel costano entrambi 35,00 Euro, router (XTR-ZB1-RLI) e coordinator (XTR-ZB1-CLI). Il convertitore FT232RL costa 7,00 Euro. I prezzi si intendono IVA compresa.

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287
<http://www.futurashop.it>



Pubblichiamo la soluzione vincente del nostro quiz "Scrivi il firmware", che invitava a progettare il firmware per il lettore Rfid proposto nel fascicolo n° 139: ecco descrizione e listati del programma per il micro.

..... a cura della REDAZIONE

Il nostro concorso iniziato nel fascicolo 140 di Elettronica In è terminato il mese scorso e delle soluzioni ricevute, pubblichiamo la migliore e più tempestiva; dal momento che il lettore che l'ha proposta - il Sig. Mauro Diotti di Cesano Maderno (MI) - ci ha fornito anche una spiegazione molto esauriente, riteniamo giusto non confinare la soluzione in un riquadro della rubrica "Elettronica

Insieme", ma lasciare ampio spazio al vincitore del quiz, affinché ci esponga il modo in cui ha risolto il problema da noi posto. Ricordiamo che il quesito proponeva la stesura di un firmware che realizzasse almeno le funzioni principali del lettore di transponder descritto nel fascicolo n° 139 della nostra rivista. Il sig. Diotti ha centrato l'obiettivo; a lui vanno i complimenti della Redazione e, natural-

I TAG RFID

Il nostro progetto e di conseguenza il firmware scritto dal nostro lettore, sono stati pensati per leggere e gestire tag Rfid con tipo di codifica ASCII a 64 bit, la cui memoria è organizzata come indicato nella tabella qui accanto. I transponder di questo tipo (nel nostro progetto si usano dei Sokymat Unique) sono composti, oltre che dalla memoria che contiene il codice, da una logica per leggerne il con-

alimentata: a ciò provvede un semplice alimentatore basato su una bobina ai capi della quale il campo elettromagnetico del lettore genera una tensione e da un raddrizzatore con filtro capacitivo. Dunque, ogni volta che il tag viene esposto a un campo magnetico a 125 kHz, estrae dalla memoria il proprio codice, composto da 64 bit che vengono emessi serialmente (e vanno a cortocircuitare la bobina ogni volta che un dato vale 1 logico); ogni stringa inizia con un header di 9 bit tutti ad 1 logico, che servono a sincronizzare la comunicazione con il lettore e non fa parte dei

tenuto ogni volta che il dispositivo viene eccitato dal campo elettromagnetico prodotto dal lettore, campo che deve essere a 125 kHz. Affinché la logica funzioni, deve essere



1	1	1	1	1	1	1	1	1	1	9 header bits
8 version bits or customer ID				D00	D01	D02	D03	P0		
				D10	D11	D12	D13	P1		
32 data bits				D20	D21	D22	D23	P2		
				D30	D31	D32	D33	P3		
				D40	D41	D42	D43	P4		
				D50	D51	D52	D53	P5		
				D60	D61	D62	D63	P6		
				D70	D71	D72	D73	P7		
				D80	D81	D82	D83	P8		
				D90	D91	D92	D93	P9		10 line parity bits
4 column parity bits				C0	C1	C2	C3	Stop		

CARATTERISTICHE TECNICHE

- **Codifica a 64 bit**
- **Codice univoco memorizzato in ROM (EM 4102)**
- **Frequenza di funzionamento: 125 kHz**
- **Tipo di codifica personalizzabile dal costruttore a scelta tra:**
 - **Data Rate: 2, 4 o 8 kbps**
 - **Tipo di modulazione: Manchester, Biphase or PSK**
 - **Configurazione Sokymat (Manchester, 2 kbps)**

dati. Seguono 40 bit di dati corrispondenti a 10 righe di memoria per 4 bit, più 10 bit di parità delle righe e 4 per le colonne. Completa la stringa il bit di stop, che vale sempre 1 logico. Il contenuto della memoria non è uguale per tutti i tag, altrimenti ognuno potrebbe

aprire la serratura cui è abbinato un altro; il codice è univoco e quindi diverso da un tag all'altro. Ciò permette comunque l'esistenza di numerosissimi tag Rfid dello stesso tipo, dato che i bit di codifica sono sufficienti a formare addirittura 100 miliardi di combinazioni!

mente, l'oscilloscopio Velleman messo in palio. Lasciamo dunque la parola al Sig. Diotti, che ci racconta come è arrivato alla soluzione.

"...essendo un vostro assiduo lettore di vecchia data, ho seguito con grande interesse la vostra iniziativa; malgrado gli impegni, sono riuscito comunque a cimentarmi in questa avvincente sfida. Il risultato finale è stato un firmware, perfettamente fun-

zionante, ma che sicuramente è suscettibile di tante migliorie e soprattutto ottimizzazioni, che il poco tempo a disposizione mi ha impedito di fare. Essendo, inoltre, un fiero assertore della teoria secondo cui il sapere va condiviso con tutti per la crescita di tutti, autorizzo, anzi prego, chiunque voglia cimentarsi, a migliorare il mio software, e a condividere gli affinamenti che ne deriveranno con tutti i lettori. Inizio dunque con l'analisi progettuale.

ANALISI DEL PROGETTO

Partendo da un hardware esistente, il kit codice MK179 distribuito da Futura Elettronica, viene chiesto di scrivere il firmware per il microprocessore PIC12F629 che svolga tutte le funzioni descritte nell'articolo sulla rivista n. 139 di Elettronica In.

Per la stesura del codice ho preferito usare un compilatore "c": in particolare il CCS, regolarmente registrato dalla mia

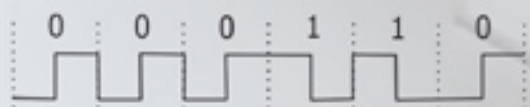


Fig. 1

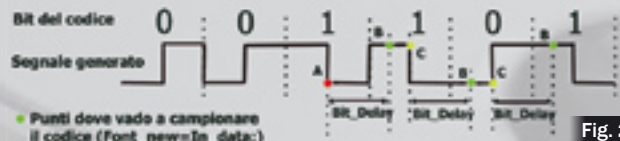


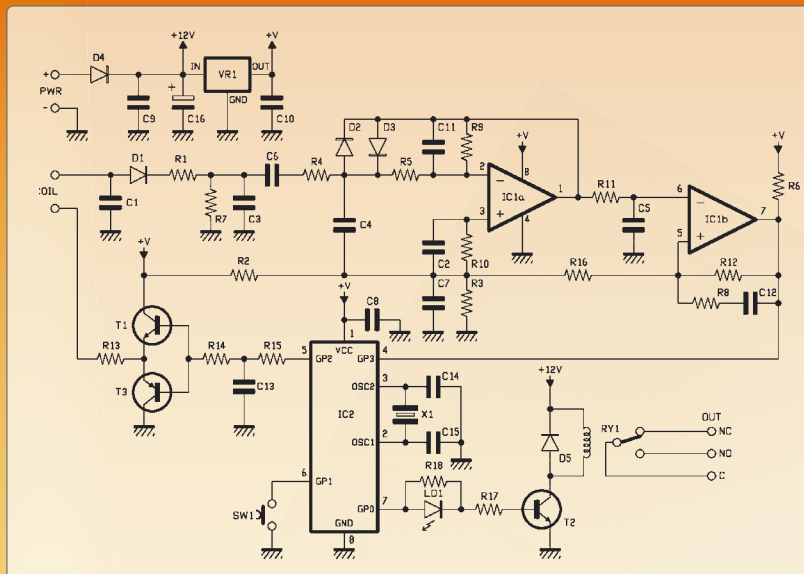
Fig. 2

azienda, in quanto una scrittura in assembler sarebbe stata a dir poco ostica per chiunque avesse voluto metterci le mani per migliorarlo, me per primo. Il software è comunque facilmente trasferibile su altri compilatori reperibili anche in versione freeware su Internet, oppure partendo dal file .lst che allego, che è l'out in assembler del compilatore CCS. È anche possibile (con qualche modifica) importarlo in MpLAB per poi modificarlo ed assemblarlo nuovamente in codice macchina.

La documentazione in mio possesso relativa al tag utilizzato e la lettura dei fascicoli arretrati di Elettronica In mi hanno permesso di mettere a fuoco il punto più problematico del progetto, che ritengo sia la generazione della portante a 125 kHz per eccitare il transponder e la contemporanea analisi dei dati da lui trasmessi. Il piccolo PIC di casa Microchip deve poter lavorare in multitasking: da una parte, sul pin GP2, deve generare la portante a 125 kHz necessaria ad eccitare il transponder, dall'altra, sul pin GP3, deve prendersi carico del treno di dati generati dal TAG, analizzarli e discriminarli, per dar modo alla terza parte del software (che può permettersi di lavorare più rilassata) di gestire il Pin GP0 per l'attivazione del relé e del Pin GP1 per l'analisi del pulsante.

Per queste operazioni ci viene in aiuto il solo fatto di disporre di un clock per il PIC di 20 MHz, che riesce a garantire una "sufficiente" velocità di calcolo. Infatti 20 MHz danno come clock di esecuzione delle istruzioni del PIC una frequenza di 5 MHz, pari ad 1/4, cioè una velocità di esecuzione di una istruzione di 0,2 μ s. Siccome devo generare

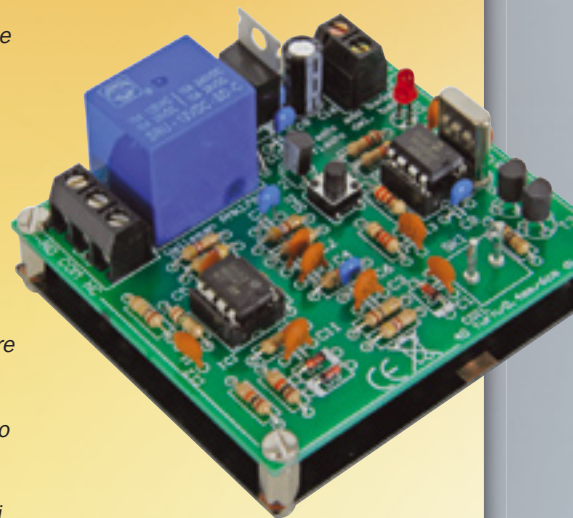
L'hardware del lettore RFID



La chiave RFID è basata su un semplice doppio operazionale per generare il campo elettromagnetico eccitatore e rilevare le commutazioni del chip dei tag standard codificati a 64 bit, oltre che su un piccolo microcontrollore di casa Microchip che decifra i dati, memorizza fino a 25 diversi codici (e altrettanti tag) e, riconoscendoli in lettura, attiva un relé (con scambio capace di commutare 2 A a 220 V) a tempo o a permanenza, a seconda dell'impostazione che si fa una volta sola per tutti i tag subito dopo l'apprendimento del primo di essi. Si tratta perciò di un lettore di tag RFID stand-alone per il comando di elettroserrature, tornelli, cancelli motorizzati ed altro di simile, in grado di riconoscere tag a badge e portachiavi a una distanza di circa 8 cm, il che significa che non serve appoggiare il tag come invece è richiesto da molti lettori di transponder per RFID riservati al controllo accessi.

La nostra chiave si alimenta a 12 Vcc (assorbe circa 100 mA) e dispone di un pulsante per avviare l'apprendimento o cancellare la memoria dei tag abbinati, oltre che di un led che fornisce tutte le segnalazioni, nelle procedure di

programmazione e ripete la condizione del relé nel normale utilizzo. I tag si apprendono uno alla volta, ma per rimuoverne uno occorre cancellare l'intera memoria. Quando il micro è stato appena programmato e acceso per la prima volta o a seguito della cancellazione totale, il circuito riconosce tutti i transponder standard con codifica a 64 bit e il relé viene comandato in modo impulsivo con tempo di attivazione di 1 secondo.



Listato 1

```
////////////////////////////////////////////////////////////////////  
// 21/11/2009 //  
// Software di gestione per KIT MK179 della Futura Elettronica //  
// Compilato con CCS Pic C Compiler //  
// Per processore Pic 12F629 //  
// Autore:Mauro Diotti //  
// //  
//////////////////////////////////////////////////////////////////  
  
#include "C:\PROGETTI\Rfid\Firmware\main.h"  
int Cicles,Index,Bit_index,Byte_index,retray,Bit_Delay,x,Byte_Temp;  
int Seq[16];  
int Code[5];  
int Tot_Code,Time_ON,EEprom_Offset;  
short count,Front_New,Valid_Code,Bistabile,Ok_Code;  
#rom 0x2100= {0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,0}  
              // inserisco nel file HEX la eeprom completamente vuota.  
  
Activate_Rele() // routine finale che attiva il rele per il tempo necessario  
{  
    int Sec;  
    Sec=Time_ON;  
    if(Bistabile)  
        Out=~Out; // in modalita bistabile inverte solo l'uscita  
    else // altrimenti  
    {  
        Out=1;  
        if(sec)  
        {  
            while(Sec) // la attiva solo per il tempo necessario  
            {  
                delay_ms(1000);  
                Sec--;  
            }  
        }  
        else // o solo per 0.5 secondi se Time_on è a 0  
            delay_ms(500);  
        Out=0;  
    }  
}  
  
void EEprom_Store(int offset) // memorizza in eeprom , nella posizione corretta  
{ // data da offset i 5 byte che compongono il codice  
    Byte_Temp=0; // di un trasponder. in 128 byte di eeprom ci stanno  
quindi while(Byte_Temp<5) // 25 codici diversi e avanzano 3 byte che uso per memorizzare  
{ // il tempo di attivazione del rele e il numero di codici  
    presenti.  
    Write_eeprom(Offset,Code[Byte_temp]);  
    delay_ms(2);  
    Offset++;  
    Byte_temp++;  
}  
}  
  
void rele(int x) // fa battere x volte il rele per conferma delle varie operazioni  
{  
    while(x)  
    {  
        Out=1;  
        delay_ms(200);  
        Out=0;  
        delay_ms(400);  
        x--;  
    }  
}  
  
void Init_Var() // routine di inizializzazione delle variabili  
{  
    index=16;  
    while(index)  
    {  
        index--;  
        Seq[index]=0;  
    }  
    cicles=0;  
    Index=128;  
    Bit_index=0;  
    Byte_index=0;  
    retray=200;  
    count=false;  
}  
  
short Check_Bit(int index) // routine che mi serve per calcolare quale bit di qual
```

Per semplificarmi la vita nella fase di funzionamento in multi tasking, ho pensato di fare solo una mezza analisi dei dati generati dal transponder, lasciando il compito della restante analisi alla terza parte di software, senza avere l'impegno di generare la portante a 125 kHz.

Sappiamo che il codice univoco del transponder TAG è composto da 64 bit, i primi dei quali sono posti fissi a "1" e servono per il sincronismo con l'hardware di lettura; i seguenti 54 comprendono i dati veri e propri del codice (40 bit) e i bit di controllo della parità sulle linee e sulle colonne (10 per le linee e 4 per le colonne). L'ultimo bit, quello di stop, è fisso a "0" ed è indispensabile anch'esso per la sincronizzazione con l'hardware nel caso si preferisca usare un sistema di decodifica diverso da quello usato da me.

Ho fatto quindi in modo da sincronizzarmi con il protocollo del TAG, generando sul pin GP2 la portante a 125 kHz e leggendo in contemporanea i dati che mi arrivavano sul pin GP3, dopodiché, senza fare alcuna analisi sul contenuto, ho memorizzato

in un buffer di 16 byte (**Seq[16]**) i 128 bit comprendenti il codice (16 Byte x 8bit = 128 bit); in questo modo sono sicuro che da qualche parte in quei 128 bit ritrovo la mia sequenza di 64 bit completi del codice. Per sincronizzarmi con il Tag ed avere la certezza che questo sia sufficientemente vicino all'antenna trasmittente/ricevente, in modo che non abbia incertezze di funzionamento, ho usato un metodo molto semplice.

Nella Fig. 1 ho inserito uno stralcio di codice dei TAG con codifica "manchester": come si vede, ad ogni passaggio da "0" a "1" la portante resta allo stato logico alto per un tempo doppio rispetto a quello che si ha per il passaggio da "0" a "0" o da "1" a "1". Nel caso, invece, del passaggio da "1" a "0", la portante resta per lo stesso tempo ma a valore zero.

Per la precisione, sapendo che il bit-rate del tag è di 2 kbaud, cioè un bit è lungo $1/2000 = 500 \mu s$, nel bit "0" avremo $250 \mu s$ a zero logico seguiti da $250 \mu s$ a livello alto. Nel bit "1" avremo il contrario, ma quello che interessa di più è che nel passaggio da "0" a "1" ci sarà un gradino a livello alto della durata di $500 \mu s$. Nella prima parte della routine "Main" genero la portante a 125 kHz e vado a cercarmi, per sincronizzarmi, questo gradino largo $500 \mu s$; per la precisione, continuo a cercarlo finché non ne ho trovati 200. A questo punto, dopo aver letto 200 gradini larghi $500 \mu s$, ho la certezza che sto ricevendo dei dati validi, alla giusta velocità, quindi sicuramente il transponder è posizionato correttamente vicino all'antenna. Usare questo gradino come sincronismo ha inoltre un altro vantaggio: oltre a sapere che il transponder è posizionato cor-

```
byte
{
    // andare a leggere avendo come indice index
    Bit_index= (0b10000000 >> (index & 0b00000111));
    Byte_index=index>>3;
    if(Seq[Byte_Index] & Bit_Index)
        return (true); // se il bit è a 1 restituisco un true
    else
        return(false); // altrimenti un false
}
short Find_Code() // routine di verifica della presenza di un codice valido
{
    int Index,Byte_Code,Bit_Sincro,x,y;
    short Bit_Code,Bit_parity,Start_Code;
    short Bit1,Bit2,Bit3,Bit4,Bit5,Bit6,Bit7,Bit8;
    Index=128;
    Bit_Sincro=0;
    Start_Code=false;
    while(!Start_Code) // finche non trovo 9 bit a 1
    {
        index--;
        Bit_index= (0b10000000 >> (index & 0b00000111)); // mi calcolo il byte e il bit del
byte da andare a testare
        Byte_index=index>>3;
        if(Seq[Byte_Index] & Bit_Index) // leggo il bit che ho indicizzato, e se è a 1
        {
            Bit_Sincro++; // incremento Bit_Sincro
            if(Bit_Sincro == 9) // e se arrivo a 9
                Start_Code=True; // sono all'inizio del codice vero e proprio
        }
        else // se invece il bit letto è a 0
        {
            Bit_Sincro=0; // rimetto a zero il contatore dei 9 Bit a 1
            if(Index<59) // se index arriva ad essere minore di 59 non ho piu speranze
                // di trovare un codice valido nei restanti bit
                Return false; // quindi esco dalla routine restituendo un false al main
        } // quando esco da qui vuol dire che ho trovato i 9 bit del sincronismo del trasponder

        x=0;
        while(x<5) // per 5 volte ripeto questo ciclo che mi serve per memorizzare in 5 byte
i bit del codice
        {
            Byte_Code=0;
            index--; // index tiene sempre il conto del bit che sono arrivato a leggere nella
matrice Seq[16]
            Bit1=Check_Bit(index); // chiamo la check_bit che mi legge quel bit all'interno
della matrice
            index--;
            Bit2=Check_Bit(index);
            index--;
            Bit3=Check_Bit(index);
            index--;
            Bit4=Check_Bit(index);
            Bit_Parity= Bit1 ^ Bit2 ^ Bit3 ^ Bit4; // faccio xor per sapere il bit di parita
            index--;
            Bit_Code= Check_Bit(index);
            if(Bit_Parity != Bit_Code) // e lo vado a confrontare con quello in Seq[16]
                return(false); // se è sbagliato annullo tutto e torno nel main con un false
            index--; // altrimenti proseguo con altri 4 bit
            Bit5=Check_Bit(index);
            index--;
            Bit6=Check_Bit(index);
            index--;
            Bit7=Check_Bit(index);
            index--;
            Bit8=Check_Bit(index);
            Bit_Parity= Bit5 ^ Bit6 ^ Bit7 ^ Bit8;
            index--;
            Bit_Code= Check_Bit(index);
            if(Bit_Parity != Bit_Code) // e ne controllo la parita
                return(false);
            if(Bit1)
                Bit_set(Byte_Code,7);
            if(Bit2)
                Bit_set(Byte_Code,6);
            if(Bit3)
                Bit_set(Byte_Code,5);
            if(Bit4)
                Bit_set(Byte_Code,4);
            if(Bit5)
                Bit_set(Byte_Code,3);
            if(Bit6)
                Bit_set(Byte_Code,2);
            if(Bit7)
                Bit_set(Byte_Code,1);
            if(Bit8)
                Bit_set(Byte_Code,0);
            Code[x]=Byte_Code; // se le parita erano ok scrivo il byte risultante nella matrice
Code[5]
            x++; // incremento x per passare ai successivi 8 bit del byte successivo
        }
        return true; // torno nel main con un true, avendo quindi un codice valido nella ma-
trice Code
    }
}
```

```

} // potro continuare nel main con le verifiche del codice
void Clear_EEPROM() // routine che azzerla la memoria EEPROM
{
    x=128;
    while(x)
    {
        x--;
        write_eeprom(x,0);
        delay_ms(2); // ritardo consigliato per essere certi che la scrittura in eeprom sia
// finita, senza usare interrupt
    }
    rele(3); // 3 battiti per conferma
}
void main() /////////////////////////////////////////////////// INIZIO PROGRAMMA
{
    //port_a_pullups(TRUE); // Attivo le pull up sulla porta A
    #ASM // ma siccome questa funzione del compilatore
    MOVWL 0x02 // me la attiva solo sul pin GP0 ho dovuto
    BSF 0x03.5 // riscriverla in assembler per poterla attivare
    MOVWF 0x15 // sul pin GP1 del pulsante
    BCF 0x01.7
    #ENDASM
    setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32); // configurazioni di default
    setup_timer_1(T1_DISABLED); // configurazioni di default
    setup_comparator(NC_NC); // configurazioni di default
    setup_vref(FALSE); // configurazioni di default
    set_tris_a(0b00001010); // configuro GP1 e GP3 come in e il resto come out
    if(!puls) // se all'accensione P1 è premuto
        Clear_eeprom(); // chiamo la routine di cancellazione della EEPROM
    while(!Puls) // aspetto che venga rilasciato il pulsante
        delay_ms(100); // semplice delay di antirimbato ( non necessario)
    Tot_Code=read_eeprom(127); // inizializzo la variabile TotCode e Time On
    Time_ON =read_eeprom(126); // salvate nelle ultime 2 locazioni di EEPROM
    Bistabile=false; // inizializzo la variabile della modalita bistabile
    if(Time_ON>240) // se il valore in EEPROM è maggiore di 240
        Bistabile=True; // considero il funzionamento in bistabile
    while(true) // inizio del ciclo infinito
    {
        Init_Var(); // chiamo la routine di inizializzazione delle variabili usate
        while(retray) // finche non ho trovato 200 impulsi da 500uS eseguo questa
        {
            // parte di codice
            Onda=1; // metto a livello 1 l'uscita che genera la portante a 125 KHz
            delay_cycles(15); // inserisco un ritardo di 15 cicli istruzione
            Front_new=In_data; // leggo il valore su GP3 e lo assegno a front_new
            cycles++; // incremento di 1 la variabile cicli che tiene il conto delle
            // onde complete a 125khz generate, per avere un tempo di circa
            // 500 uS devo generare 500uS/8uS = 62,5 cicli
            Onda=0; // metto a 0 l'uscita GP2
            if(Front_new) // se prima avevo letto uno stato logico 1
            {
                count=true; // allora comincio a contare i numeri di cycles dell'onda
                delay_cycles(9); // ritardo calcolato per avere 125KHz
            }
            else // se invece avevo letto uno stato logico 0
            {
                if(count) // se la variabile count è true vuol dire che in precedenza
                // avevo letto uno stato 1 in GP3 quindi adesso che è tornata
                // a 0 vado a leggere se l'ingresso è stato 1 per circa 500uS
                {
                    if((cycles > 60) && (cycles < 70)) // 500uS sono circa dai 60 ai 70
                    {
                        cycles=0; // cycles=0 per predisporla al prossimo conteggio
                        retray--; // decremento la variabile che conta i 200 impulsi a 500uS
                    }
                    count=false; // se invece l'impulso letto era fuori dai parametri
                }
                // rimetto a false la variabile count
            }
            else // se count è false genero un delay di 7 cicli istruzione
            {
                // per avere i 125 khz precisi sulla portante
                delay_cycles(7); // e setto cycles a 0
                cycles=0;
            }
        }
    }
    /////////////////////////////////////////////////// Chiude il while(retray), a questo punto sono arrivato a
    // contare 200 impulsi da circa 500uS, ho la certezza che il
    // trasponder sta funzionando bene.
    while(index) // inizio quindi un ciclo che ripeterò per 128 volte nel quale
    {
        // campiono i bit del codice del trasponder
        Bit_Delay=38; //
        index--;
        while(Bit_Delay) // per 38 volte genero la portante a 125KHz che mi da uno
        {
            // shift di posizionamento sul bit di circa 300uS
            Onda=1;
            delay_cycles(19);
            Onda=0;
            delay_cycles(13);
            Bit_Delay--;
        }
        /////////////////////////////////////////////////// Chiude while(Bit_Delay) a questo punto sono circa a
        // 1/4 del bit, il punto giusto per campionarlo
        Onda=1; // ...nel frattempo devo costantemente settare l'uscita
        // onda(GP2) ad intervalli regolari per avere i 125 KHz
        Bit_index= (0b10000000 >> (index & 0b00000111)); // dal valore di index

```

rettamente, so anche che in quel momento, quando leggo questo gradino da 500 μ s mi trovo al centro di 2 bit; quello precedente è uno "0" e quello successivo un "1". Se avessi usato come sincronismo i gradini stretti di 250 μ s non avrei potuto sapere se appartenevano alla seconda metà di un bit "0" o alla prima metà di un bit "1".

Tornando al firmware, una volta verificata la presenza di 200 gradini da 500 μ s parte la fase vera e propria della cattura dei 128 bit nel buffer **Seq[16]**.

Appena finisco di acquisire il duecentesimo gradino da 500 μ s, mi trovo nel punto rosso **A** indicato nella Fig. 2; non mi interessa sapere se sono all'inizio del codice, alla fine o a metà, perché quella analisi la farò in un secondo tempo. Adesso la cosa più importante è sapere che sono esattamente al centro di un bit "1". A quel punto comincia la parte di codice racchiusa in un ciclo di while (**while(index)**) che viene ripetuta per 128 volte e mi permette di catturare i 128 bit all'interno dei quali troverò la sequenza di 64 bit del codice del transponder.

La prima operazione che faccio è quella di generare un ritardo (**Bit_Delay**) pari a 375 μ s, in modo da arrivare al punto verde **B**, cioè nella prima metà del bit che dovrò acquisire; a quel punto vado a leggere l'ingresso GP3 e saprò se sto catturando un bit 0 o un bit 1. Il dato corrispondente lo vado a memorizzare nel buffer **Seq[16]** e resto in attesa di leggere dal pin GP3 un fronte di tensione, al verificarsi del quale mi troverò nel punto giallo **C** e ricomincerò daccapo il ciclo del while, generando il ritardo di **Bit_delay**, andando a campionare l'ingresso, memorizzando il bit e aspettando

il fronte. Così, per 128 volte; il tutto ovviamente continuando a generare la portante quanto più possibile precisa di 125 kHz. Dopo di ciò chiamo la routine **Find_Code()** che si occupa di leggere in sequenza i 128 bit che ho memorizzato per trovare il codice del transponder. Durante tutto questo processo (terza fase del programma) posso evitare di generare la portante a 125 kHz, in quanto ormai i dati contenuti nel codice sono al sicuro nella RAM del PIC, il che semplifica di molto la programmazione. In questa routine non faccio altro che andarmi a rileggere uno alla volta i bit memorizzati fino a trovarne 9 consecutivi a "1"; da quel punto so che parte il codice vero e proprio del transponder. Vado e leggere gruppi di 4 bit alla volta, li metto in OR esclusivo per ricavarne il bit di parità e confronto quest'ultimo con il bit seguente della sequenza: se il check della parità dà esito positivo proseguo con un altro gruppo di 4 bit su cui faccio lo stesso test; una volta convalidati 8 bit li memorizzo nel buffer **Code[5]**, poi vado avanti con i restanti bit fino a riempire il buffer con i 40 bit veri e propri del codice

```
// compreso tra 0 e 127 mi calcolo il rispettivo Byte
// e bit del byte da settare se trovo un bit 1 del codice
Byte_index=index>>3;
Front_new=In_data; // qui leggo l'ingresso GP3 che mi da il valore del bit
// del codice che poi andro a memorizzare nella variabile
// Seq[16] nella posizione indicizzata da Byte_index
// e Bit_index

Delay_cycles(1);
Onda=0;
if(Front_new) // se il bit del codice è a 1 metto a 1 il relativo bit
// della variabile Seq[16]
{
    delay_cycles(17);
    Onda=1;
    Seq[Byte_index]=(Seq[Byte_index] | Bit_index);
    Onda=0;
    delay_cycles(9);
}
else // altrimenti se il bit è 0 non scrivo nulla e lascio lo 0
// già presente dopo l'inizializzazione delle variabili
    delay_cycles(8); // e mi occupo solo dei soliti ritardi per il 125 Khz
while(In_data == Front_new) // ora resto in attesa di rilevare il cambiamento
{
    // di fronte del segnale del trasponder
    Onda=1;
    delay_cycles(19);
    Onda=0;
    delay_cycles(9);
} //////////////// Chiude while(In_data == Front_new) Una volta individuato il
fronte
} //////////////// Chiude while(index) ricomincio il ciclo dall'inizio per altre
127 volte
if(Find_Code()) // una volta catturati i 128 bit chiamo la routine find_code
// che mi restituirà TRUE se il codice è valido, FALSE se non trova
i 9 bit
// di sincronismo o se trova delle parità sbagliate.
{
    if(!Puls) // se find_code da un true e se è stato premuto il pulsante
    {
        if(!Tot_Code) // se è il primo TAG da memorizzare
        {
            Eeprom_Store(0); // chiamo la routine che memorizza il codice in Eeprom
            Tot_Code=1;
            write_eeprom(127,1); // e setto a 1 l'ultimo byte in eeprom
            rele(2); // chiamo la funzione rele che farà battere il rele per conferma 2
volte
            delay_ms(2000); //attendo 2 secondi
            Time_ON=0; // per default inizializzo time_on a 0
            while(!Puls && (Time_ON < 241)) // se dopo 2 secondi ho ancora il pulsante
schiacciato
            {
                rele(1); // battito del rele
                Time_ON++; // incremento i secondi nella variabile Time_ON
                delay_ms(1000); // delay
            }
            if(!Puls) // se dopo 240 secondi non ho lasciato il pulsante
            {
                Bistabile=true; //abilito il funzionamento in modo bistabile
                write_eeprom(126,241); //e scrivo 241 nella cella 126
            }
            else // se invece prima dello scadere dei 240 secondi lascio il pulsante
            {
                Bistabile=false; // non attivo la modalità bistabile
                write_eeprom(126,Time_ON); // e scivo nella cella 126 il tempo in cui
// dovrà restare attivo il rele
                rele(2); // 2 battiti del rele per conferma
            }
        } // chiudo if(!Tot_Code)
        else // se invece non è il primo TAG da memorizzare
        {
            if(Tot_Code<25) // se ho in memoria meno di 25 codici
            {
                Eeprom_Offset=Tot_Code *5; // calcolo l'offset per memorizzare il codice in
eeprom
                Eeprom_Store(Eeprom_Offset); // e chiamo la routine che lo memorizza
                Tot_Code++; // incremento la variabile che tiene conto di quanti codici ha
in memoria
                write_eeprom(127,Tot_Code); // e lo scrive in eeprom
                rele(2); // 2 battiti per conferma
                if(Tot_Code == 25) // se sono al 25esimo codice
                {
                    rele(3); // 3 battiti per avviso di memoria esaurita
                }
            }
            while(!Puls); // prima di proseguire aspetto che venga rilasciato il pulsante
            delay_ms(100);
        }
        else // Se invece NON è stato premuto il pulsante dopo il true
della find_Code
        {
            // verifico se il codice aquisito è in memoria e attivo
            l'uscita.
            x=0;
            while(x<Tot_Code) // ripeto questo codice per il numero di codici che ho in eeprom
```

Continuazione Listato 1

```
{
    Ok_Code=true;
    EEPROM_Offset=x*5;
    Byte_Temp=0;
    while(Byte_Temp<5)
    {
        if(read_eeprom(EEPROM_Offset+Byte_Temp) != Code[Byte_Temp])
            Ok_Code=false; // se trovo un solo byte sbagliato metto a false Ok_Code
        Byte_Temp++; // verifico il byte successivo
    }
    x++; // incremento x per leggere il codice successivo
    if(Ok_Code)// ma se nel frattempo il codice letto era presente in eeprom
    {
        Activate_Rele(); // chiamo la routine che attiva il rele
        x=Total_code; // questa ultima parte di codice è MOLTO sporca, metto x=Total_
    } // solo per uscire dal ciclo di while in modo da non testare i
    codici // seguenti presenti in eeprom...sarebbe decisamente da siste-
    mare
} // Chiude il while(true)
} // Chiude il main
```

transponder. A quel punto vado a controllare lo stato del pulsante: se è premuto vuol dire che quel codice lo si vuole memorizzare in EEPROM, eseguo quell'operazione chiamando la routine **Eeprom_Store**, passandogli come parametro l'offset dato dall'indicizzazione del codice, andando poi a memorizzare quell'indice nell'ultima locazione di memoria disponibile: la 127. Verifico inoltre che, se si tratta del primo codice memorizzato, eseguo la parte di codice che si occupa di acquisire il tempo di attivazione del relé e memorizzo in EEPROM questo valore (nella posizione 126) con valori che vanno da 0 a 240 (0=0,5 s.). Il codice 241 significa funzionamento bistabile. Poi riparto dall'inizio del ciclo **while(true)** aggiornando nuovamente le variabili di programma e generando ancora una volta la portante di 125 kHz, in attesa di un nuovo treno di bit dall'ingresso GP3. Nel caso invece in

cui dopo aver riempito il mio buffer **Code[5]** non rilevo alcuna chiusura del pulsante, vado a rileggermi in EEPROM i codici precedentemente memorizzati, fino a trovarne uno che coincida con **Code[5]**; a quel punto chiamo la routine **Activate_rele()** e mediante essa attivo il relé per il tempo richiesto, dopodiché torno all'inizio del ciclo **while(true)** e ricomincio tutto daccapo. A questo punto la spiegazione del firmware è finita. Ripeto ancora che questo progetto l'ho "buttato giù" con parecchia fretta. Inoltre, non essendo un softwarista di mestiere, ma solo un autodidatta a tempo perso, ritengo che tutte le routine possano essere migliorate ed ottimizzate. Ho sicuramente usato un numero esagerato di variabili che rendono il software un po' troppo macchinoso da rileggere; inoltre il sistema da me usato di campionare il bit a 1/4 della lunghezza non è detto che sia il migliore. Forse analizzando i

fronti del segnale, da alto a basso e viceversa, si poteva creare una routine migliore, magari sfruttando l'interrupt di variazione di stato dei pin; probabilmente andava gestita meglio la RAM usando puntatori anziché matrici e andava inoltre attivato il watch-dog per affrontare possibili situazioni di stallo.

Chissà, col tempo proverò altre soluzioni; nel frattempo resterò in attesa di spiare da chi vincerà questo concorso quali soluzioni avrà adottato, più efficaci delle mie...

Appreziamo l'umiltà del nostro lettore, che certo non avrebbe immaginato di vincere lui questo concorso. A lui facciamo i migliori auguri, invitandolo a seguirci e a concorrere alle nostre iniziative future.

I listati delle routine cui il Sig. Mauro si riferisce sono illustrati in queste pagine, nei listati Listato 1 e Listato 2: il primo contiene la porzione di firmware chiamata *main.c* e il secondo la parte *main.h*. ■

Listato 2

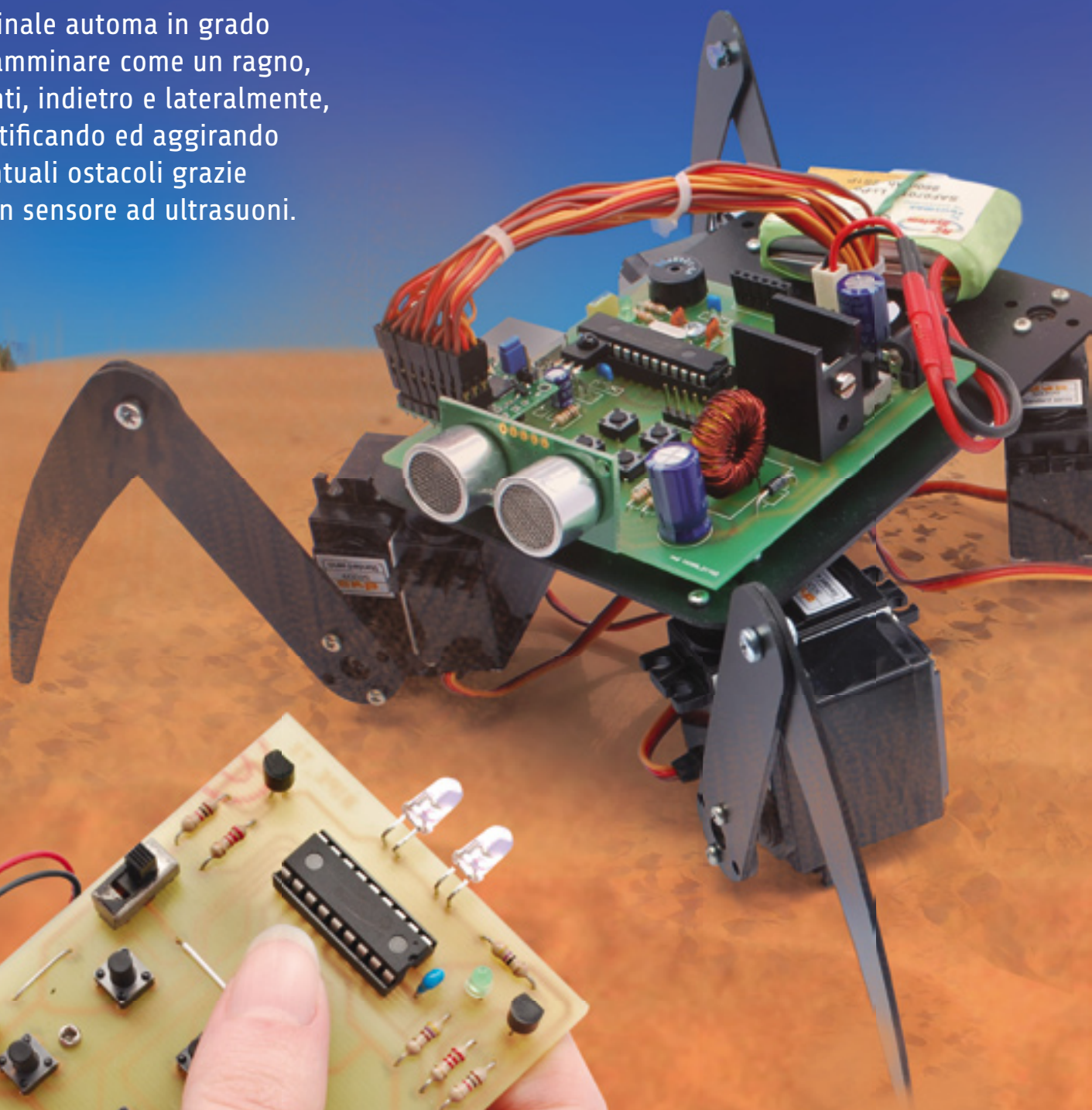
```
#include <12F629.h>
#FUSES NOWDT, HS, NOCPD, NOPROTECT, NOMCLR, NOPUT, BROWNOUT
#use delay(clock=2000000)
#byte Port_a =0x05

#bit Out =0x05.0 // A0 uscita rele
#bit Puls =0x05.1 // A1 ingresso pulsante
#bit Onda =0x05.2 // A2 usato per creare l'onda quadra a 125 KHz.
#bit In_Data =0x05.3 // A3 ingresso dati
```



Il signor Mauro Diotti di Cesano Maderno (MI) ha proposto la soluzione firmware per il lettore RFid.

Originale automa in grado di camminare come un ragno, avanti, indietro e lateralmente, identificando ed aggirando eventuali ostacoli grazie ad un sensore ad ultrasuoni.



dell'ing. MIRCO SEGATELLO

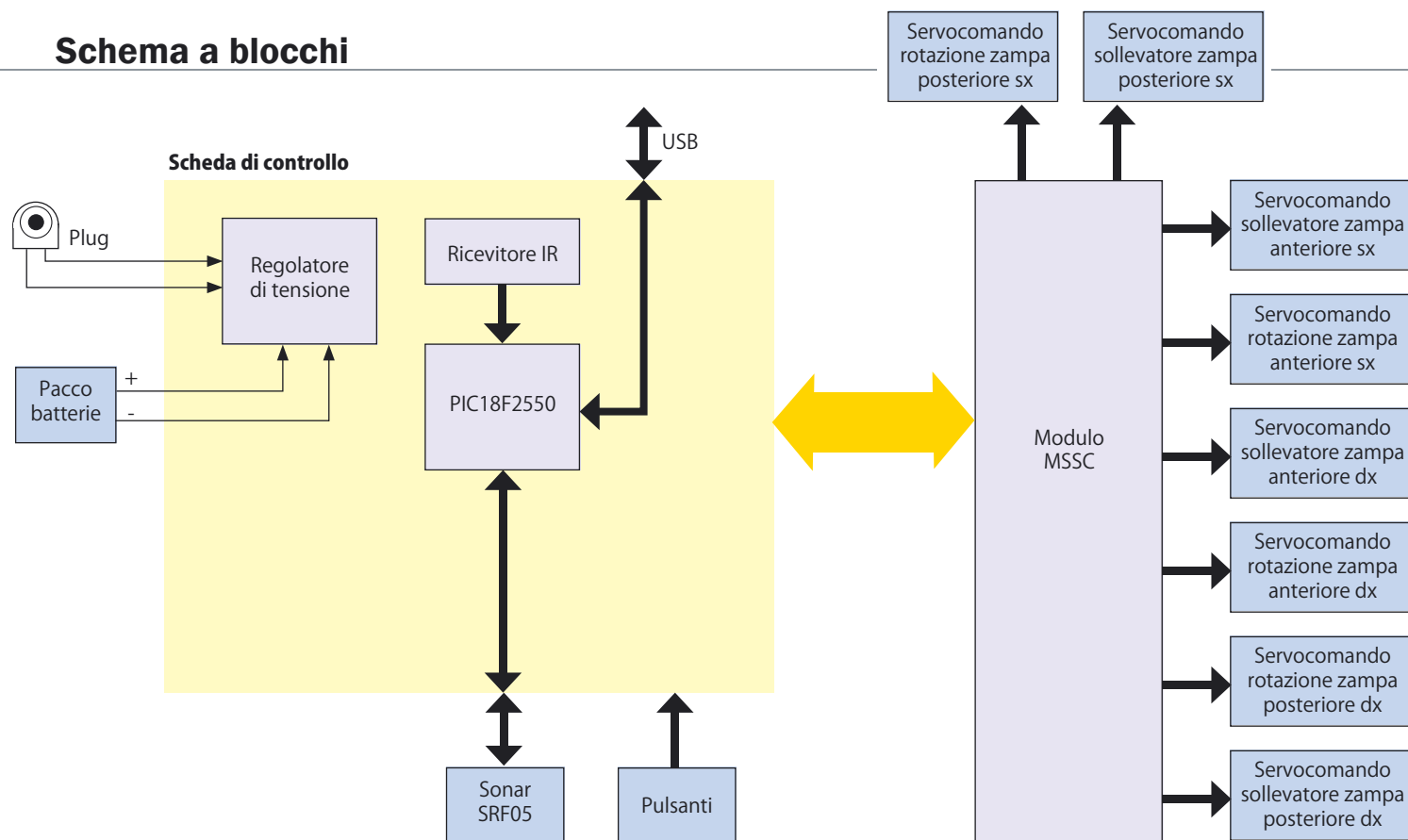


SPIDERIN ROBOT RAGNO A 4 ZAMPE

Visto il grande successo del robot bipede, ad un anno di distanza abbiamo deciso di realizzare un nuovo e suggestivo automa, questa volta non con le sembianze umane, bensì dall'aspetto di un grosso ragno. Ma non vi preoccupate: non fa impressione ma è una sorta di simpatico giocattolo che vi darà la possibilità di comprendere le problematiche connesse alla progettazione di un robot con arti ancora più snodati di quelli del bipede. Anche se ad un primo pensiero potremmo immaginare una complessità maggiore del bipede, in realtà, come vedremo, siamo riusciti ad ottenere tutti i movimenti con una struttura abbastanza semplice. In linea di principio più gambe ha un robot, maggiori sono i punti di appoggio e migliore sarà la sua stabilità; per questo, in un primo studio preliminare avevamo pensato di dotare il nostro robot di 8 zampe, come un vero ragno. Per ogni zampa sarebbero stati necessari tre gradi di libertà, al fine di simulare il reale movimento delle zampe di un ragno, ma facendo questo ci saremmo ritro-

vati con ben 24 gradi di libertà, non proprio facilissimi da gestire e richiedenti un hardware dal costo poco abbordabile. Essendo il nostro obiettivo la divulgazione della robotica con progetti di complessità e costi accettabili, è stato necessario riformulare gli obiettivi e i termini di progetto. Ci siamo chiesti quali fossero gli elementi base per poter proporre un robot con zampe in grado di compiere i movimenti base di camminata e rotazione, senza necessariamente venir meno ad un progetto interessante e di sicuro effetto che non trascurasse gli elementi fondamentali della rappresentazione robotica di un ragno. Accettando alcuni compromessi ci è stato possibile ridurre di molto le risorse necessarie, passando a quattro zampe, riducendo anche i gradi di libertà per zampa. Nella stesura del progetto era necessario tenere in considerazione anche la complessità della parte meccanica, che in alcuni progetti supera decisamente il livello medio di preparazione di un lettore di una rivista di elettronica; senza

Schema a blocchi



parlare dei costi. L'unico punto fermo di partenza era quello di prevedere per i vari movimenti l'utilizzo di servocomandi da modellismo dello stesso tipo di quelli usati nel bipede, in modo da permettere un riutilizzo degli stessi ed una semplificazione generale dal punto di vista elettromeccanico. Altro punto di partenza era prevedere un controller dedicato alla gestione dei servo, in modo da svincolare il microcontrollore principale dal compito più gravoso, con il vantaggio di ottenere un firmware più snello ed una gestione ottimale delle periferiche. Uno degli aspetti più importanti per un progetto open-source come questo è di prevedere più di una piattaforma di sviluppo che un prodotto finito, dato che a questo pensano già molte aziende cinesi che stanno inondando il mercato di robot per impieghi ludici, dalle forme più svariate. Per tale ragione era nostro interesse predisporre degli input come pulsanti (o comando remoto) facilmente

riutilizzabili per altre applicazioni; anche la scheda di controllo può essere riutilizzata per altri robot. Pensando al nostro ragno come ad un sistema di sviluppo, era nostra intenzione potervi mettere nelle condizioni di sviluppare a vostro piacimento il programma di gestione, prevedendo un sistema molto semplice ed immediato per la programmazione del microcontrollore. Come vedremo in seguito, non serve un programmatore in quanto un semplice cavetto USB ed un software sono sufficienti per consentire la programmazione del microcontrollore della scheda di controllo.

IL ROBOT

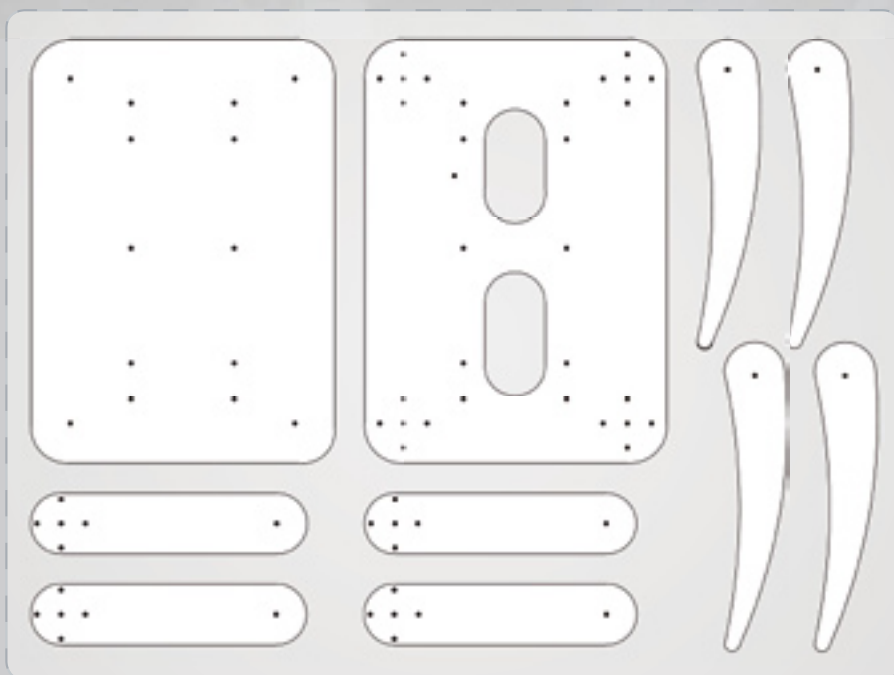
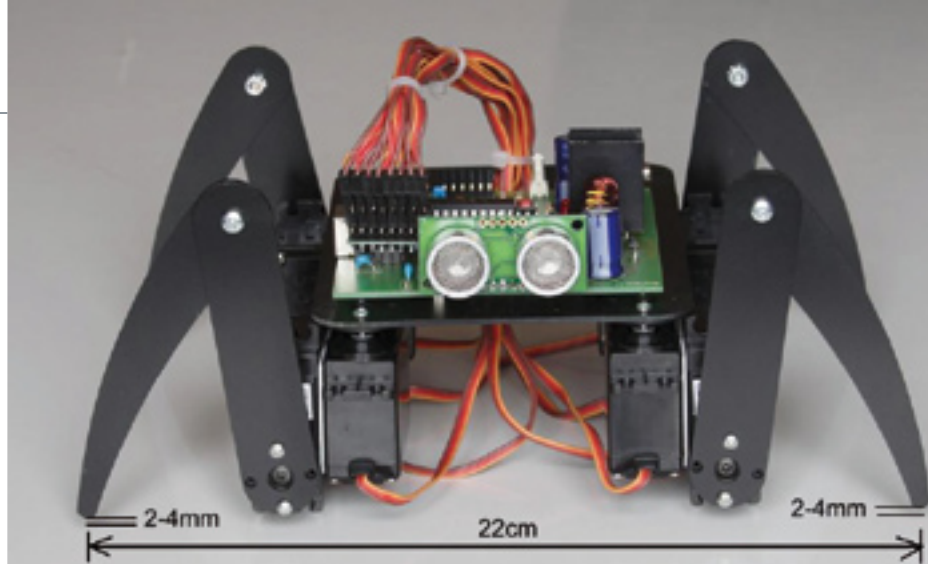
Adesso che finalmente abbiamo chiarito le premesse, entriamo nei dettagli del nostro progetto. La nostra prima scelta è stata l'utilizzo di quattro gambe con due gradi di libertà di movimento, per un totale di otto gradi di libertà; questo è il minimo per poter realizzare una forma

simile ad un ragno ed assicurare dei movimenti accettabili. Tale scelta impone l'utilizzo di un controller per otto servocomandi; allo scopo abbiamo scelto un modello reperibile nel catalogo della Futura Elettronica: il modulo MSSC (Micro Serial Servo Controller), che prevede tutta una serie di funzioni quali il controllo della velocità e della posizione indipendente per ogni servo. A questo punto siamo partiti con il disegno della parte meccanica, la quale si è ispirata al robot bipede e quindi si realizza con la stessa tecnica utilizzata per gli stampati dei circuiti elettronici. Con un attento uso delle forme siamo riusciti ad evitare disastuose saldature o incollature, e, come vedremo, per l'assemblaggio è sufficiente del semplice biadesivo. Nella sostanza, la meccanica è formata da un corpo base sul quale saranno fissati i servo adibiti ai movimenti e dai quali si dipartono le zampe, queste ultime realizzate in due pezzi, ciascuna con una forma

molto simile a quella dei ragni. Un pezzo supplementare è stato pensato per essere posto al disotto di tutta la struttura allo scopo di irrobustirla, in modo da eliminare la tendenza alla flessione tipica della vetronite. Alla fine ci siamo accorti che questo pezzo non è del tutto fondamentale e può essere omesso a patto di accettare una lieve deformazione della struttura durante il funzionamento. Consigliamo in questo caso di realizzare la struttura con della vetronite doppia faccia, in modo che la ramatura rimanga e contribuisca al suo rafforzamento. In alternativa può andar bene qualsiasi altro materiale purché leggero e rigido, come alluminio (spesso 2 mm) o plexiglass dello spessore di 4 mm. Una bella verniciata di colore nero opaco consentirà di ottenere un aspetto molto simile a quello dei ragni veri, anche se nulla vieta di costruire una "carrozzeria" a piacimento: ad esempio con della spugna colorata. Il disegno della parte meccanica è riportato in queste pagine, ma consigliamo di scaricare i relativi file dal nostro sito. Tutti i fori riportati in essi fanno ovviamente riferimento ai servocomandi S209 utilizzati in questo progetto.

ASSEMBLAGGIO MECCANICO

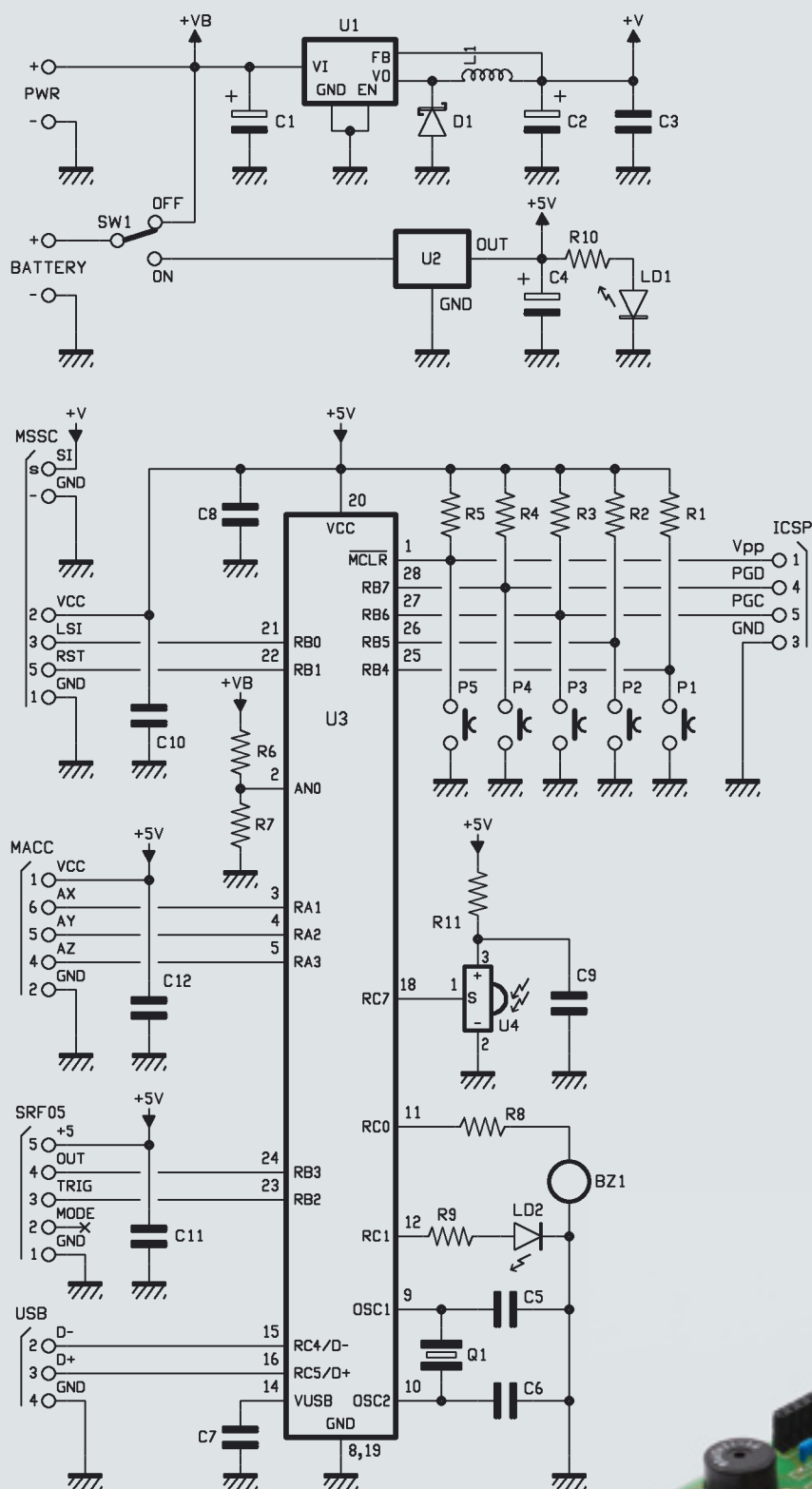
Non appena in possesso dei pezzi meccanici, potete procedere al loro assemblaggio, facendo riferimento ai disegni ed alle foto pubblicate. Per ogni zampa dovete accoppiare due servocomandi che serviranno uno per il movimento di rotazione, ed uno per alzare o abbassare la gamba. L'unione tra i due può essere ottenuta con del biadesivo, che dalle prove da noi effettuate ha retto bene gli sforzi meccanici e consente un eventuale recupero dei servocomandi. L'incollaggio



Qui sopra è illustrata la dima per la realizzazione dei particolari meccanici componenti la base e le zampe del robot ragno; il disegno è in scala 1:2,5 cioè ridotto al 40 %. In alto, le dimensioni effettive del robot.

è senz'altro la soluzione più performante, ma assicuratevi di utilizzare della buona colla per impieghi specifici su plastiche. Non abbiamo previsto complicati settaggi software per posizionare i servo in modo ottimale, in quanto se seguirete la procedura qui descritta non avrete alcun problema. In pratica bisogna portare tutti i servocomandi in posizione di neutro e per fare questo potete utilizzare un apposito tester per servocomandi oppure avvalervi dello stesso circuito

di controllo, il quale, all'accensione, è stato programmato per portare tutti i servo in posizione di neutro. La squadretta di ogni servocomando (quella tonda) va fissata alla struttura base tramite due piccole viti, in modo che i servo risultino allineati. Sulla squadretta ci sono svariati fori; utilizzate quelli più adatti a raggiungere lo scopo. Assemblate le zampe unendo i due pezzi con una vite da 3 mm e relativo dado (o incollati) ed allineatele in modo che tutte e quattro abbiano

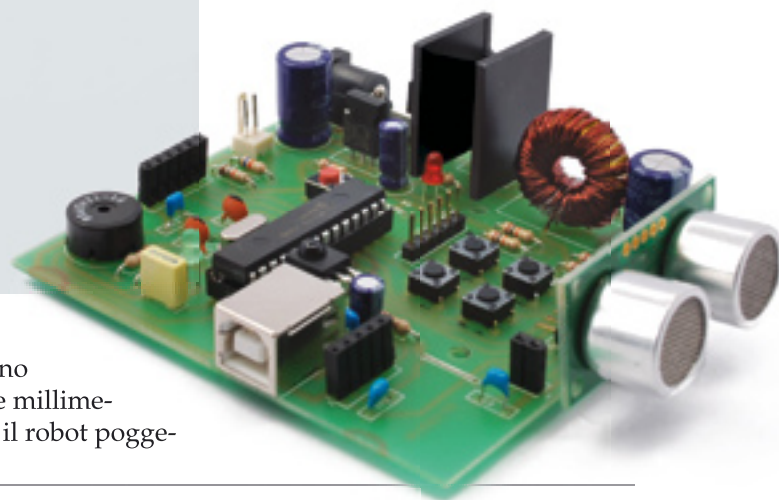


rà sui quattro servo che hanno il compito di alzare o abbassare le zampe. Abbiamo scelto questa posizione di neutro perché permetteva di non mettere sotto sforzo i servo e quindi contenere gli assorbimenti di corrente da fermo.

Guardando il ragno frontalmente, la distanza tra i punti di appoggio delle zampe dovrà risultare di circa 22 cm e ogni zampa dovrà essere ortogonale con la struttura meccanica. Utilizzate dei distanziatori e i fori predisposti per fissare lo stampato sopra al robot e delle fascette per radunare ordinatamente i cavi dei servocomandi. Frontalmente si dovrà inserire il sensore ad ultrasuoni, che, anche se non strettamente necessario, aggiunge un tocco di veridicità simulando la presenza degli occhi.

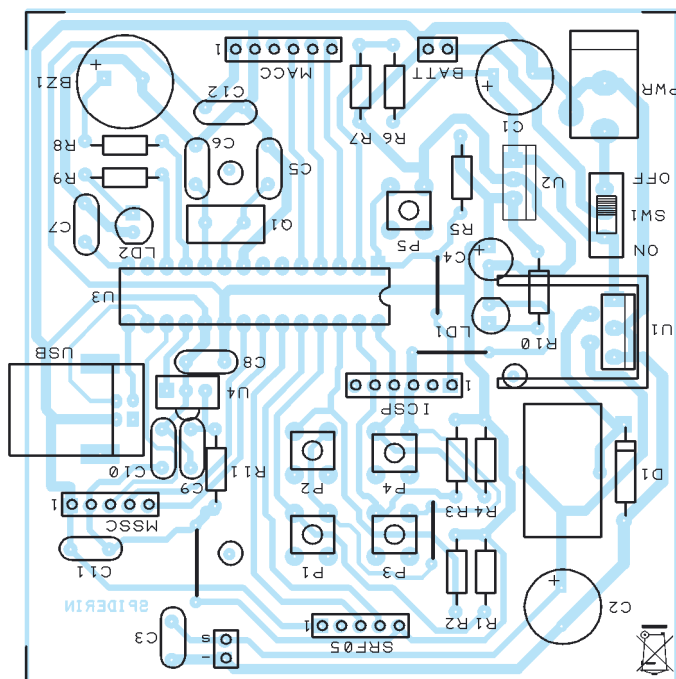
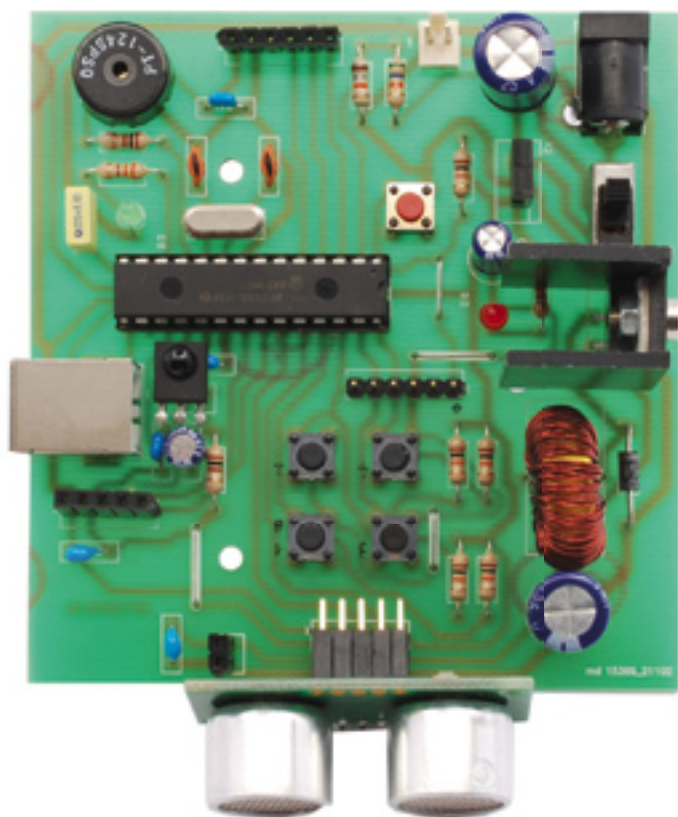
IL CIRCUITO DI CONTROLLO

Come potete vedere, lo schema elettrico del circuito che governa il robot-ragno non è affatto complicato ed è tutto incentrato sull'utilizzo di un microcontrollore PIC18F2550. La scelta è caduta su questo PIC per svariati motivi: in primo luogo la possibilità di farlo funzionare con un clock di 48 MHz; poi per la disponibilità del modulo USB interno, che ci permette, tramite un applicativo denominato



la stessa angolazione. A lavoro ultimato, appoggiando il robot su un tavolo, i quattro punti d'appoggio corrispondenti alla parte

terminale delle zampe si dovranno trovare a qualche millimetro dal tavolo ed il robot pogge-



Elenco Componenti:

R1 ÷ R5: 10 kohm
R6: 56 kohm
R7: 18 kohm
R8, R11: 100 ohm
R9: 330 ohm
R10: 330 ohm

C1, C2: 1000 μ F 25 VL
elettrolitico

C3, C4: 100 nF multistrato

C5: 10 μ F 25 VL elettrolitico

C6, C7: 22 pF ceramico
C8 ÷ C11: 100 nF multistrato
Q1: Quarzo 20 MHz
U1: LM2576-5
U2: LM2940
U3: PIC18F2550-I/SP (MF852)
U4: PNA4602
MSSC: Driver Motori MSSC
SRF05: Modulo ultrasuoni
SRF05
MACC: Modulo accelerome-

tro M-ACC

D1: 1N5819

LD1: Led 3 mm rosso

LD2: Led 3 mm verde

L1: Induttanza 400 μ H

BZ1: Buzzer senza elettronica

SW1: Deviatore a slitta MV

P1 ÷ P5: Microswitch

Varie:

- Plug alimentazione

- Zoccolo 14+14

- Strip femmina 2 poli
- Strip femmina 5 poli
- Strip femmina 6 poli
- Strip femmina 5 poli 90°
- Strip maschio 6 poli
- Vite 10 mm 3 MA
- Dado 3 MA
- Dissipatore
- Connettore USB-B
- Servomotore SERVO209 (8 pz.)
- Circuito stampato

BootLoader, di programmare direttamente il PIC da PC senza l'ausilio di alcun programmatore. Abbiamo comunque previsto il connettore J6 compatibile Microchip PICKIT2, da usare per la programmazione classica. L'alimentazione è sdoppiata e prevede un'alimentazione con regolatore lineare per la parte di controllo e quella che alimenta il PIC, ed una sezione switching per l'alimentazione di potenza. Per la regolazione

lineare abbiamo indicato l'utilizzo di un integrato tipo LM2937 o LM2940, perché del tipo Low DropOut, ovvero a bassa caduta uscita/ingresso e perciò in grado di stabilizzare anche con basse tensioni di ingresso. Senza molte preoccupazioni potrete adottare un LM7805, a patto di usare un'alimentazione superiore agli 8 volt. Questo primo stabilizzatore non è però in grado di fornire tutta la corrente necessaria ai servocomandi che ad un primo

calcolo abbiamo valutato in circa 3 A. Ovviamente il circuito non assorbe continuamente questa corrente, ma durante le varie fasi di manovra si possono verificare molte oscillazioni nell'assorbimento. Un alimentatore di tipo lineare non avrebbe consentito un buon rendimento e, sapendo che il robot sarà alimentato da una batteria, si è deciso di adottare un alimentatore di tipo switching Step-Down, realizzato tramite un LM2576-5. Questo ci permette di

Le connessioni dei servocomandi

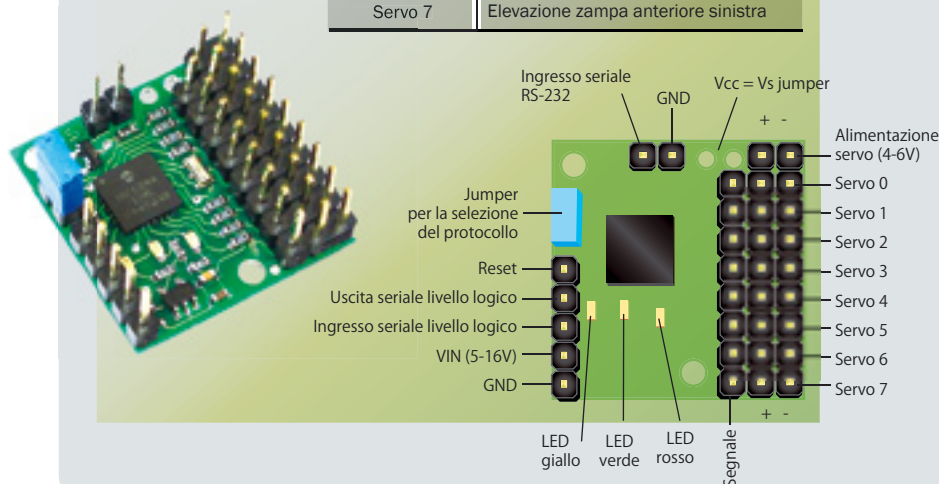
Il robot ragno necessita in tutto di otto servocomandi, per la totalità gestiti da un controller MSSC dedicato. I connettori a tre contatti di ciascuno dei servo vanno inseriti direttamente nelle punte previste nel modulo MSSC, che corrispondono alle rispettive

uscite; allo scopo, il disegno in basso a destra in questo riquadro illustra i punti dove

collegare ciascuno. La **Tabella 1** riassume le funzioni delle singole uscite dell'MSSC.

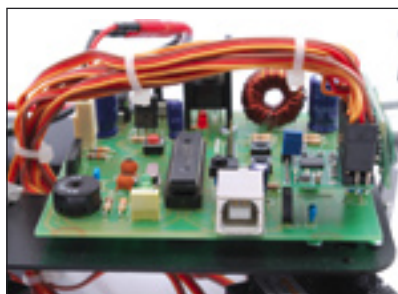
Tabella 1

Connettore di MSSC	Ubicazione
Servo 0	Angolazione zampa anteriore destra
Servo 1	Elevazione zampa anteriore destra
Servo 2	Angolazione zampa posteriore destra
Servo 3	Elevazione zampa posteriore destra
Servo 4	Angolazione zampa posteriore sinistra
Servo 5	Elevazione zampa posteriore sinistra
Servo 6	Angolazione zampa anteriore sinistra
Servo 7	Elevazione zampa anteriore sinistra



alimentare il robot con un pacco batteria che possa fornire un range di tensioni molto ampio, compreso tra un minimo di 7 ed un massimo di 12 volt. È quindi possibile utilizzare pacchi batteria Li-Po da 2 o 3 celle oppure pacchi batteria Ni-Cd o Ni-MH con 8÷10 celle. Consigliamo comunque di limitare il peso delle batterie, quindi se optate per le batterie Ni-Cd o Ni-MH è meglio usare formato AAA, mentre per le batterie agli ioni di litio o più performanti, vanno bene 2 o 3 celle da 1000 mA/h. Giusto per la cronaca, il nostro prototipo è stato collaudato sia con un pacco batteria da 8 celle nichel-cadmio da 800 mA/h, sia con un pacco di Li-Po formato da due celle da 860 mA/h. La scheda è equipaggiata con pulsanti, led e cicalino utilizzabili a piacere; inoltre troverete il connettore J5 previsto

per accogliere un accelerometro tipo MMA7260QT. Attualmente questo componente non viene gestito dal firmware, ma in futuro sarà utile per sviluppare alcuni esperimenti di camminata in ambienti accidentati. Per rilevare gli ostacoli durante l'avanzamento abbiamo utilizzato un sensore ad ultrasuoni tipo SRF05, scelto anche per un fatto estetico, ovvero per la sua forma che



Il circuito di controllo del robot si interfaccia direttamente ad un PC mediante una connessione USB ed è predisposto per ospitare un SONAR.

assomiglia molto a quella di due occhi allineati. Come sul bipede, abbiamo previsto un telecomando ad infrarossi facente utilizzo del sensore PNA4602; per la precisione, diciamo che il firmware previsto gestisce esattamente lo stesso telecomando del bipede, quindi se volete usare questa funzione dovrete realizzare il telecomando descritto sull'articolo del Robot Bipede pubblicato sul numero 133 della rivista. Le resistenze R6 ed R7 rappresentano un partitore di tensione utilizzato per leggere tramite il canale analogico AN0 del PIC il valore della tensione di alimentazione. La massima tensione ammessa è di 20,48 volt il che permette, utilizzando il convertitore ADC del PIC con risoluzione a 10 bit, di misurarla con una precisione di 20 mV. Abbiamo fissato la minima tensione di alimentazione in 6,5 V, al di sotto dei quali il robot non si muove e porta nello stato di OFF tutti i servocomandi per evitare di scaricare eccessivamente la batteria. Questo valore è l'ideale se si alimenta il robot con due celle Li-Po oppure con 8 celle Ni-Cd; se invece utilizzate una tensione più alta, sarebbe necessario reimpostare questo valore a seconda del numero di celle presenti. Il tutto vi permetterà di ottenere l'arresto del funzionamento prima di aver scaricato eccessivamente le batterie (al



Ogni zampa è fissata ad un servocomando, a sua volta fissato al corpo di un secondo servo, il cui albero è attaccato alla base; ciò permette i movimenti di rotazione e sollevamento.

riguardo, vedete la descrizione del firmware).

IL CABLAGGIO

Per l'assemblaggio della parte elettrica non ci sono particolari indicazioni; lo stampato può essere realizzato partendo dai file scaricabili dal nostro sito www.elettronica.in.it. Tutto il materiale è di facile reperibilità ed il firmware occorrente per la programmazione del micro lo trovate anch'esso nel nostro sito Internet. Una volta in possesso della basetta ramata, come al solito procedete alla saldatura dei componenti a più basso profilo e successivamente quelli più alti per finire con i condensatori da 1.000 µF e l'integrato regolatore di tensione. Particolare attenzione va messa nella saldatura del ricevitore ad infrarossi U4, il quale dovrà essere piegato in modo che la parte bombata sia rivolta verso l'alto; questo per favorire la ricezione dei comandi. Per quanto riguarda l'induttanza essa dovrà essere realizzata partendo da un nucleo tipo poliferro (di solito di colore giallo) delle dimensioni di 17,8 x 9,6 x 6 mm sul quale avvolgerete circa 80 spire di filo di rame del diametro di 0,6 mm di diametro, corrispondenti a due strati di spire. All'integrato U1 dovrà essere collegato un dissipatore al fine di smaltire il calore prodotto; non è necessario utilizzare alcun isolatore in quanto il dissipatore non entra in contatto con altri componenti. Per quanto riguarda il connettore per la batteria consigliamo di usarne uno che prevenga l'inversione di polarità, in quanto la connessione errata dell'alimentazione comporta il danneggiamento della scheda. Il modulo controllo servocomandi può essere saldato direttamente, ma consigliamo di utilizzare dei semplici strip femmina in modo

da poterlo disinserire all'occorrenza, oppure semplicemente per poterlo riutilizzare in un altro progetto. Una raccomandazione riguarda le resistenze R6 ed R7, che essendo adibite alla lettura della tensione della batteria dovranno essere molto precise, quindi utilizzate resistenze con precisione dell'1 % oppure se fate ricorso a resistori al 5 % misurate il valore con un buon tester scegliendo quelle più precise (quelle il cui valore più si avvicina a quello dichiarato).

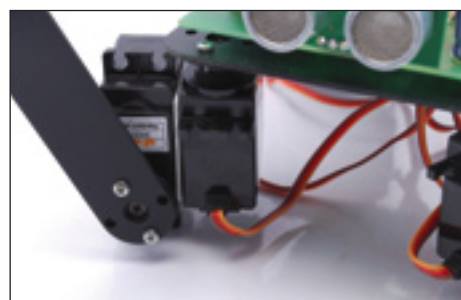
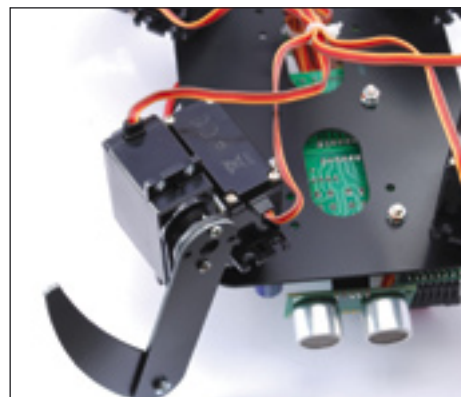
Quando collegherete i servocomandi dovrete fare attenzione alla loro posizione seguendo la **Tabella 1**.

COLLAUDO

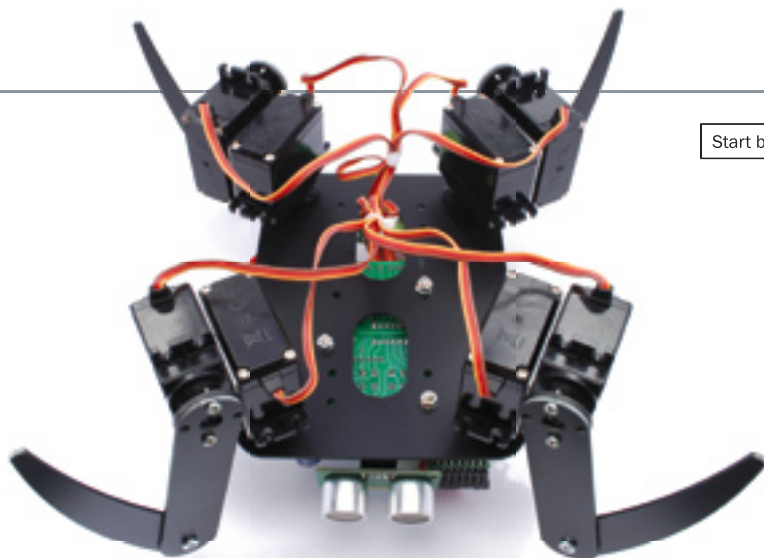
Per un primo collaudo consigliamo di disinserire il controller dei servi e di non inserire il PIC: ponete semplicemente una resistenza del valore di 100 ohm tra i terminali + e - del connettore J2, corrispondenti all'alimentazione di potenza dei servo. Fornite tensione alla scheda e controllate con un tester la presenza dei 5 volt sia sul lato PIC che sull'alimentatore switching; il LED D2 dovrà essere acceso. Verificato questo siete pronti per inserire il PIC opportunamente programmato ed il controller dei servo. All'accensione, dopo 1 secondo (trascorso il warm-up dell'alimentatore switching), tutti i servo saranno posizionati nel punto di neutro. Potrebbe capitare che qualche servo vibri leggermente, questo è dovuto al meccanismo interno che non ha ancora trovato il punto di neutro. Alle volte è sufficiente un piccolo colpo al servo per sistemare le cose.

IL FIRMWARE

Il programma del PIC (sviluppato con il compilatore PICBASIC-PRO) si occupa essenzialmente

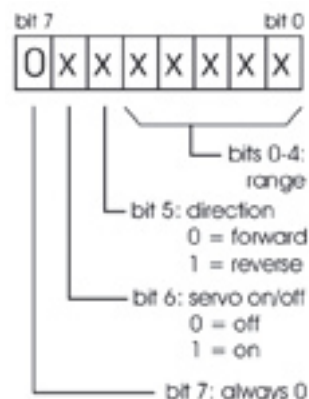


di controllare lo stato delle periferiche e di inviare i corretti comandi al controller dei servo. In linea di massima non è molto complicato però è necessario calcolare con estrema precisione la posizione di ogni singolo servo a seconda del movimento che si vuole ottenere, rispettando tutte le tempistiche in modo esatto. Ma iniziamo con ordine: all'avvio come di consueto configuriamo tutte i registri interni del PIC ed in modo particolare il modulo USART che useremo per ricevere i comandi via telecomando ad infrarossi. La ricezione viene gestita tramite interrupt ed è stata configurata per una velocità di 1.200 baud, per questo motivo non possiamo utilizzare il TX del modulo USART per gestire il modulo MSSC che invece sarà comandato a 9.600 baud con il comando software SEROUT del



Start byte=0x80	Device ID=0x01	Command	Servo Num	Data1	Data2
-----------------	----------------	---------	-----------	-------	-------

Struttura di una stringa di comando (in alto) e del byte contenente effettivamente il comando (a destra).



compilatore. Prima di addentrarci nei dettagli del firmware andiamo a descrivere il modulo MSSC usato per il controllo dei servo e una prima domanda da porsi è il perché del suo utilizzo. Come sappiamo i servo si comandano con degli impulsi ripetuti circa 50 volte al secondo,

la cui durata (tecnica del PWM) consente di definire la posizione da raggiungere. Con 8 servo è necessario generare otto impulsi ogni 20 ms, la cui durata deve essere calcolata di volta in volta; è chiaro che non rimane molto tempo per gestire anche pulsanti, interfaccia in-

frarossi, buzzer ed altro. Risulta più immediato far fare il lavoro "sporco", cioè la generazione degli impulsi, ad un altro PIC, ed occuparci solamente di calcolare le coordinate relative alla posizione. Il modulo da noi adottato ha anche una particolarità: possiamo fornirgli, oltre alla

Il robot calciatore



Indubbiamente il robot che più affascina i costruttori è quello umanoide, perché da sempre il sogno dell'uomo è costruire automi in grado di svolgere al suo posto le operazioni quotidiane. Nel settore è apparso un nuovo, interessante protagonista: si tratta di NAO, prodotto dalla *Aldebaran Robotics*, un umanoide completamente programmabile che

già nel 2010 potrà essere venduto al pubblico. Il robot, alto circa 50 cm, è un piccolo mostro di tecnologia completamente programmabile, governato da una CPU x86 AMD Geocode da 500 MHz, 1Gb di memoria flash e 256Mb di SDRAM. Il sistema operativo è Linux e, oltre alle varie funzioni di movimento, implementa una connessione Wi-Fi ed Ethernet, due altoparlanti, un sintetizzatore vocale e due webcam che usa come occhi. La struttura permette ben 25 gradi di libertà DOF, che gli



rendono possibili numerosi movimenti, tra cui la capacità di sedersi e rialzarsi autonomamente. Il comportamento del robot può essere completamente personalizzato grazie al programma d'interfaccia chiamato *Choregraphe*; ciò significa che NAO potrà fare qualsiasi cosa...anche giocare a calcio. Non a caso, si è fatto conoscere partecipando al celebre torneo di calcio per robot *RoboCup*, che ha visto diversi automi sfidarsi all'ultimo goal. NAO si basa sulla piattaforma Linux Gostai Urbi, molto utilizzata per lo sviluppo di robot ed è rivolto a tutti gli amanti della robotica; peccato per il prezzo (circa 10.000 euro) che non sia proprio per tutte le tasche...

posizione da far raggiungere al servo, anche la sua velocità; questo indipendentemente per ogni servo. In questo modo potremo ad esempio spostare l'angolazione della gamba ad una velocità, ma alzarla ad una velocità diversa. Abbiamo sfruttato proprio questa funzione per creare delle andature a velocità differenti. Questa funzione più avanzata (Pololu Mode) è disponibile qualora non venga cortocircuitato il "protocol selection jumper"; altrimenti sarà attivata la modalità più limitata (Mini SSC II Mode) che prevede solamente il posizionamento immediato del servo. L'invio dei comandi avviene in modalità RS-232, facilmente gestibile dal PIC con un apposito protocollo descritto di seguito. Nella modalità "Pololu Mode" si hanno svariate opzioni di controllo del servo. Ogni servo è comandato secondo il valore di una variabile interna, che può essere compreso tra 500 e 5.500, con una risoluzione di mezzo microsecondo. La posizione può essere specificata in modo assoluto o relativo rispetto ad un punto di neutro; è quindi possibile impostare il valore di neutro, la direzione di comando e la scala usata per definire la variabile di controllo. Il baudrate per la comunicazione viene automaticamente riconosciuto dal microcontrollore di bordo e deve essere compreso tra 2.000 e 40.000 baud. Il protocollo di comunicazione prevede l'invio di 5 o 6 byte: il primo serve per la sincronizzazione e deve valere 80H (128D) ed il secondo è, invece, l'indirizzo del dispositivo, che vale 01H. I rimanenti byte definiscono quale comando è stato inviato e il suo valore; in ogni caso il primo bit di ogni byte deve valere zero. Nella pagina accanto è illustrata la struttura di

una tipica stringa di dati. Vediamo ora in dettaglio i vari comandi disponibili, tutti impartibili tramite l'istruzione SE-RUOT del compilatore.

COMMAND0

Con questo comando si definisce la scala con la quale sarà fornita la posizione da raggiungere, la direzione di movimento e si imposta se il servo è attivo o disattivo. Il comando consta di 8 bit: i primi 4 bit definiscono il range con il quale sarà possibile stabilire la posizione del servo; nel nostro caso lavoreremo in un range compreso tra 0 e 127, mantenendo il punto di neutro a 63. Quindi questo valore sarà impostato al suo massimo, ovvero 15. In questo modo gestiremo una variabile a 7 bit che corrisponderà ad un movimento di 90°. La direzione di ogni singolo servo è stata impostata in modo che ad un valore superiore a 63 (valore di neutro) corrisponda un movimento della gamba in avanti ed un movimento di angolazione tale da alzare il robot, e ciò indipendentemente da come sia stato installato il servo. La funzione servo OFF spegne il servo

corrispondente e quindi l'assorbimento di corrente si porta quasi a zero; questa funzione la usiamo a batteria scarica per limitare la corrente assorbita dal robot se la tensione di alimentazione è insufficiente. Il bit più significativo è sempre a zero. Il comando da inviare sarà quindi:

serout LSO, 2, [\$80,\$01,\$00, 0, \$6F]
da ripetersi per ogni servo sostituendo di volta in volta il penultimo byte con il valore del servo da impostare.

COMMAND1

Con questo comando è possibile impostare la velocità di movimento del servo. Impostando a zero questo valore il servo sarà posizionato immediatamente, altrimenti il servo si muoverà gradualmente in un tempo valu-

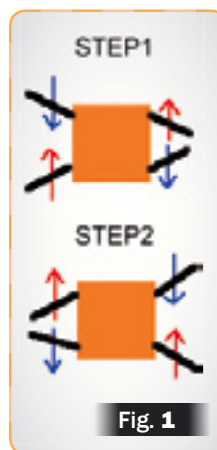


Fig. 1

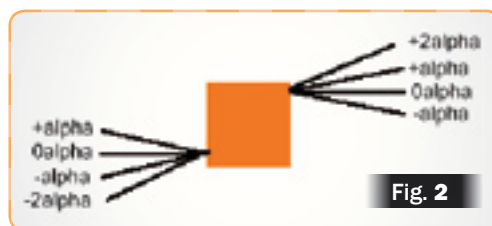


Fig. 2

Tabella 2

Angolo di spostamento	Valore del comando per il modulo MSSC
+2alpha	103
+alpha	83
0alpha	63
-alpha	43
-2alpha	23

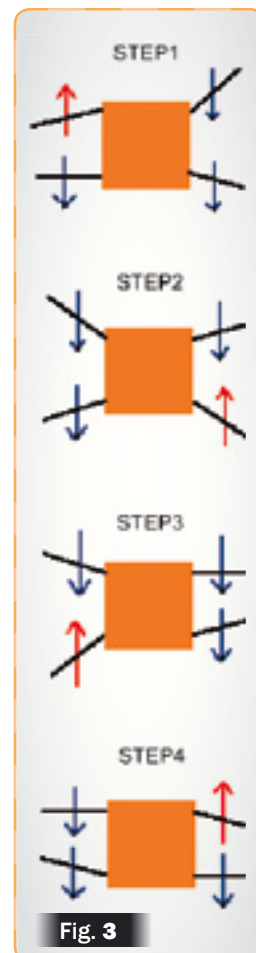


Fig. 3

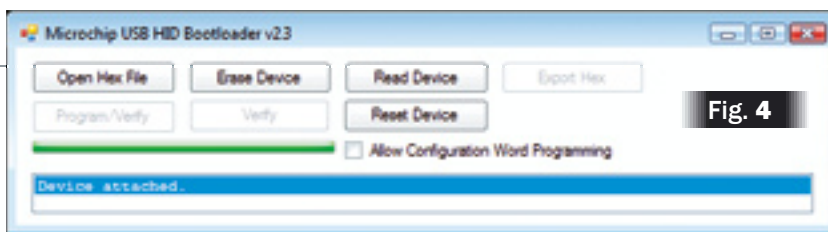


Fig. 4

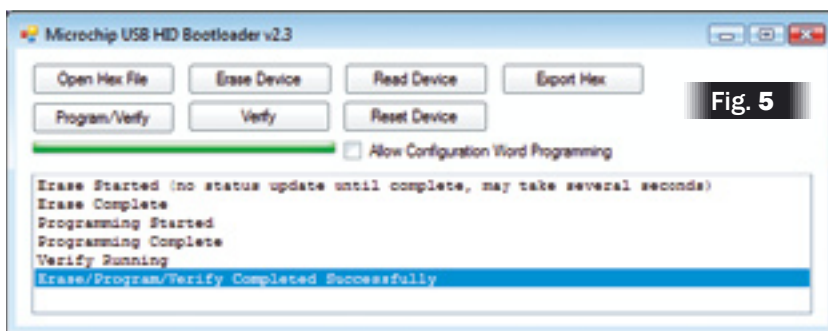


Fig. 5

tabile in circa $T_{move} = (\text{pos_partenza} - \text{pos_arrivo}) * 150 / \text{Speed}$ [msec] dove Speed è il valore impostato nella variabile e dovrà essere compreso tra 1 e 127. Il comando da inviare sarà: `serout LSO, 2, [$80,$01,$01, 0, speed]` e vedrà la variabile speed impostata di volta in volta a seconda della velocità desiderata. Il comando in questione sarà ripetuto per ogni servo.

COMMAND2

Questo è il comando che useremo più spesso, perché ci permetterà di posizionare i servocomandi nella posizione voluta. Invieremo quindi: `serout LSO, 2, [$80,$01,$02, 0, Pos]` dove con il valore della variabile Pos definiremo la posizione che il

servocomando dovrà raggiungere, rispettando la scala impostata precedentemente. Il valore 63 corrisponde a posizionare il servo al centro, mentre con i valori estremi superiore di 127 e inferiore di 0, il servo si posizionerà rispettivamente a $+45^\circ$ e a -45° .

I MOVIMENTI DEI SERVO

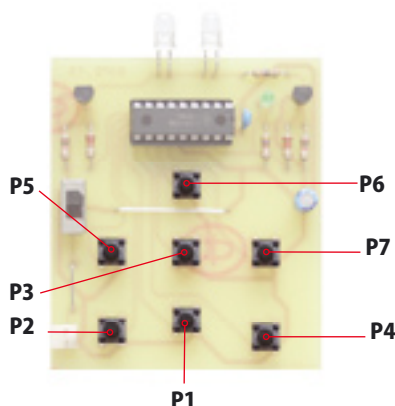
Spieghiamo ora quale ragionamento abbiamo fatto per definire i movimenti dei servo in modo da ottenere un movimento in avanti. Per spiegarci meglio, useremo dei semplici disegni stilizzati che rappresentano il robot visto dall'alto con indicati i movimenti delle quattro zampe. In linea generale possiamo affermare che in ogni istante ci dovranno essere delle zampe che toccano terra e spingono in avanti il robot mentre, le rimanenti, sollevate, si riposizioneranno. Possiamo quindi ipotizzare un primo caso

Tabella 3 - Funzioni dei pulsanti sul robot

Pulsante	Funzione
P1	Stop
P2	Gira DX
P3	Avanza (mode2)
P4	Avanza (mode1)

Tabella 4

Pulsante	Variabile CMN	Funzione
P1	0	Non usato
P2	1	Avanza (mode2)
P3	2	Gira SX
P4	3	Avanza Slow
P5	4	Stop
P6	5	Avanza (mode1)
P7	6	Gira DX



in cui due zampe toccano terra e spingono in avanti il robot e le altre due si stanno riposizionando; nello step successivo i movimenti si invertiranno. In soli due step siamo in grado di far avanzare il robot, ma va da sé che se ci sono solo due punti di appoggio la stabilità statica non è garantita; però se consideriamo un po' di inerzia della struttura ed un baricentro ben equilibrato al centro, il tutto funziona, seppure esteticamente non sia il massimo. Nella Fig. 1 sono riportati i movimenti in cui con le frecce blu si indicano i movimenti delle zampe basse (appoggiate al terreno) e in rosso i movimenti a zampe alzate (non appoggiate a terra).

C'è un secondo approccio al problema che prevede di lasciare sempre tre zampe a terra e di muoverne una sola senza appoggio a terra. Partiamo dal presupposto che per eseguire un passo occorre compiere quattro fasi, ma avere tre zampe a terra permette una maggiore stabilità. Il massimo lo si otterrebbe se la struttura fosse assolutamente rigida e non si deformasse con una zampa sollevata, e se il baricentro risultasse sempre all'interno del triangolo formato dai punti di appoggio delle rimanenti zampe. Per migliorare la stabilità facciamo in modo che le zampe non rimangano mai troppo chiuse, ma piuttosto un po' aperte. Per ogni singola fase di un passo, ciascun servo si muove di un certo numero di gradi (alpha) e quindi i possibili movimenti sono riportati nella Fig. 2. All'interno del firmware, a questi spostamenti corrisponde il posizionamento dei servo nei punti riepilogati nella Tabella 2. I movimenti corrispondenti sono meglio descritti dalla Fig. 3. In questa modalità, disponendo di soli due gradi di libertà per zampa, ogni movimento descrive

Il trasmettitore IR

Il trasmettitore del telecomando a infrarossi è lo stesso che avete già visto impiegato nel robot bipede descritto nel fascicolo n° 133: è basato su un microcontrollore che genera una portante a 38 kHz che modula in base al tasto premuto, pilotando poi due led IR.



un angolo di cerchio; per questo motivo le zampe a terra non rimangono ferme sul punto di appoggio ma subiscono un piccolo spostamento. Questo comporta inevitabilmente una certa frizione delle zampe con il pavimento e quindi il tutto funziona solo su superfici che permettano un certo slittamento delle gambe. Il sensore ad ultrasuoni viene utilizzato per rilevare la presenza di ostacoli ad una distanza inferiore a 30 cm, nel qual caso blocca automaticamente l'avanzata del robot; non viene, invece, utilizzato nelle manovre di rotazione. Il sensore può essere omesso ed il firmware girerà comunque, tuttavia noterete come usare il sonar migliori nettamente l'autonomia di movimento del robot.

LA PROGRAMMAZIONE

Era un nostro preciso obiettivo poter mettere tutti nelle condizioni di sperimentare e sviluppare questo nuovo progetto e per questo abbiamo previsto la possibilità di programmare il micro a bordo del circuito semplicemente accedendo alla scheda via USB. Il PIC deve essere preprogrammato con un piccolo applicativo residente nella parte alta della memoria, in grado di gestire la comunicazione USB e di trasferire i dati in arrivo direttamente nell'area memoria del PIC. Questo applicativo si chiama BootLoader e lo trovate di solito attivo in cellulari, navigatori e in tutte quelle apparecchiature che permettano di aggiornare il firmware senza l'impiego di programmatori esterni. Il PIC previsto per questo progetto è già programmato con il bootloader che consente successivamente di caricare il firmware del robot tramite porta USB. Se vi procurate l'ambiente di sviluppo PICBASICSIPRO sarete anche in grado

di editare i sorgenti del progetto e ricompilarli in modo da personalizzare le funzioni svolte o semplicemente per farci degli studi. Nulla vieta, ovviamente, di utilizzare qualsiasi altro compilatore a vostro piacimento; quel che conta è utilizzare del firmware compilato.

È importante specificare all'interno del programma l'utilizzo del bootloader: in questo modo andrete a sovrascrivere il bootloader stesso durante la programmazione. In PICBASIC questo viene indicato con le seguenti due righe di programma:

```
DEFINE LOADER_USED 1  
DEFINE RESET_ORG 1000h
```

A questo punto installate il software HID Booloader.exe fornito sempre assieme ai file di questo progetto ed avviatelo. Questo semplicissimo programma e il bootloader per il PIC (HID Bootloader - PIC18F2550)

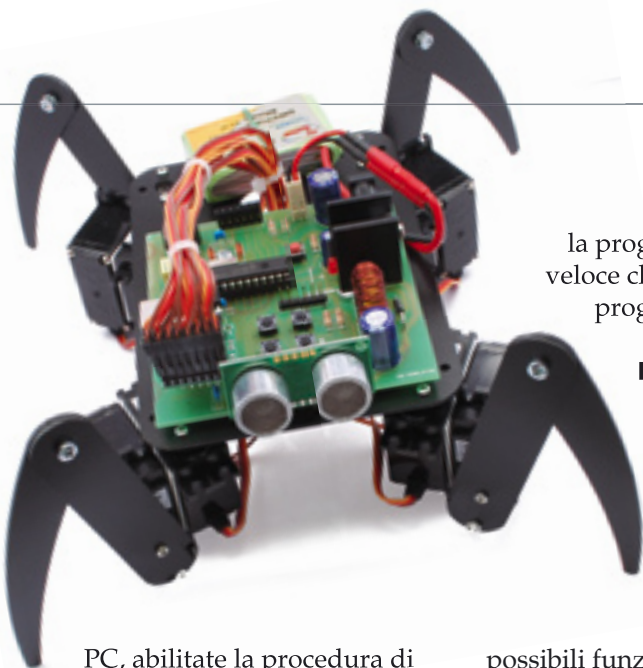
sono scaricabili gratuitamente dal sito della Microchip. Con il robot acceso e connesso tramite apposito cavo alla USB del



per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono facilmente reperibili in commercio. Il firmware ed il master del circuito stampato sono scaricabili gratuitamente dal sito della rivista (www.elettronica.in.it). E' anche disponibile la scatola di montaggio completa del robot ragno (cod. FT852SET) al prezzo di 179,00 Euro. Il kit comprende tutti i particolari elettronici e meccanici, i servocontrolli, i circuiti stampati, il sensore, il telecomando ad infrarossi ed il microcontrollore già programmato. Quest'ultimo è disponibile anche separatamente al prezzo di 14,00 Euro (cod. MF852). Tutti i prezzi si intendono IVA compresa.

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11, 21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287
<http://www.futurashop.it>



la programmazione è più veloce che con un normale programmatore.

IL TELECOMANDO

Il robot può essere gestito almeno nelle funzioni essenziali, direttamente con i pulsanti presenti sulla sua scheda principale; le

possibili funzioni sono riepilogate nella **Tabella 2**. Riferendoci ad essa diciamo che con *mode1* abbiamo indicato la camminata con tre punti di appoggio e con *mode2* la camminata con due punti di appoggio; la rotazione avviene comunque con due punti di appoggio.

Abbiamo previsto, comunque, di poter gestire il robot anche a distanza tramite un telecomando a raggi infrarossi provvisto di sette pulsanti. In questo caso riutilizziamo lo stesso telecomando già utilizzato per il robot bipede che prevedeva già sette pulsanti per altrettante funzioni che andiamo a ridefinire per il nuovo robot e che sono riepilogate nella **Tabella 3**.

Il sensore utilizzato per la ricezione dei comandi ed installato

PC, abilitate la procedura di programmazione nel modo seguente: premete il pulsante P1(boot), a seguire premete il pulsante di reset, quindi rilasciate il pulsante di reset e successivamente rilasciate il pulsante P1. Il sistema operativo (in questo caso Windows) emetterà una nota acustica a conferma dell'attivazione di una nuova periferica ed il software evidenzierà la presenza della nuova periferica (**Fig. 4**). Non dovete far altro che aprire il file sorgente compilato (estensione HEX) e caricarlo sul PIC con il comando programma, al termine cliccate su Reset del software o premete il pulsante di reset sulla scheda, il nuovo programma è già pronto a funzionare. Come potete constatare la procedura è decisamente semplice e

Guarda Spiderin in azione su YouTube!



Vuoi vedere in azione il nostro Spiderin? Camminare come un ragno aggirando gli ostacoli con un SONAR? Nulla di più semplice: collegati alla pagina <http://www.youtube.com/watch?v=ei1KpBuoYUc>. Potrai anche vedere i video di moltissimi altri nostri progetti proposti in passato.

a bordo della scheda si chiama PNA4602 ed è predisposto per funzionare ad una frequenza di 38 kHz, segnale che dovrà essere modulato con il segnale da inviare. Nel nostro caso è sufficiente inviare un byte contenente il valore numerico del comando con una velocità di 1.200 baud. Ad ogni ricezione di segnale un piccolo beep viene generato dal buzzer e se il comando è riconosciuto esso viene eseguito immediatamente. Per il momento è tutto, lasciamo a voi lettori il piacere di divertirvi con questo nuovo robot.

RM ELETTRONICA

Forniture per hobbisti, scuole e industria

Distributore autorizzato:

FUTURA
ELETTRONICA

ELSE
Kit

ELETTRONICA

Il tuo punto
di riferimento
per l'elettronica
a ROMA

Via Val Sillaro 38 • 00141 Roma • Tel: 06-8104753



Realizziamo un SoC utilizzando il microcontrollore Microblaze della Xilinx e l'ambiente di sviluppo EDK.

di
MARIANO SEVERI

Fino ad alcuni anni fa le FPGA erano prevalentemente utilizzate per realizzare logica di interconnessione ("glue logic") tra altri dispositivi (ad esempio, per implementare la decodifica di indirizzo del bus di un microprocessore in accesso alle periferiche), in moduli di acquisizione dati (ad esempio, per controllare convertitori analogico/digitali esterni) o in sistemi di comunicazione a più elevato data-rate. Col tempo, però, la crescente complessità dei dispositivi ne ha favorito la diffusione in ambiti di applicazione diversi, dall'elaborazione

numerica dei segnali, ai sistemi di trasmissione dati, agli apparati di modulazione e demodulazione. Grazie alla elevata capacità logica, infatti, si è riusciti ad integrare sempre maggiori funzionalità all'interno dei dispositivi fino a realizzare dei veri e propri System-On-Chip (SoC). Adottando la definizione proposta dalla Dataquest nell'ormai lontano 1995, un SoC, come mostrato nella **Fig. 1**, è un dispositivo che integra un'unità di elaborazione dati (sia essa un microprocessore, un microcontrollore o un DSP), un'unità di me-

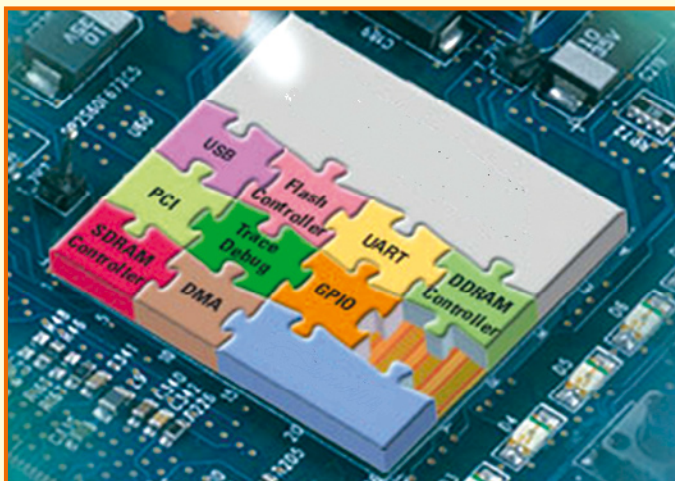


Fig. 1 - Un intero sistema in un singolo componente: i System-On-Chip.

moria on-chip e fino a 100.000 gate di logica configurabile in funzione dell'esigenza dell'utente. Le FPGA sono quindi piattaforme ideali per la realizzazione di SoC, grazie anche alla sempre maggiore diffusione di core IP. Il termine IP è acronimo per Intellectual Property, ovvero proprietà intellettuale. Un IP core è un circuito logico – descritto sotto forma di netlist o mediante linguaggi di descrizione hardware come il VHDL – a sé stante ed in grado di implementare una particolare funzione (ad esempio una porta di comunicazione UART) che, in alternativa, sarebbe realizzata da un componente dedicato. All'interno di una FPGA possono essere istanziati e connessi diversi IP, ognuno con diverse funzionalità, allo stesso modo in cui fino ad ora si sono integrati su uno stesso circuito stampato diversi componenti. Esistono dunque IP core per funzioni di vario tipo, che vanno da semplici interfacce di comunicazione (come UART, I²C, SPI), a controller di rete (Gigabit Ethernet), fino a moduli per la compressione delle immagini (encoder H264, ad esempio) o più in generale per l'elaborazioni numerica dei segnali. E vi sono anche core IP di microcontrollori più o meno complessi. Alcuni sono disponibili gratuitamente sul web, mentre altri sono soggetti a licenza. Actel, ad esempio, rilascia gratuitamente una versione del microcontrollore 8051 per le proprie FPGA e distribuisce una versione delle nuove ProAsic3 in cui una parte del dispositivo è già programmata per implementare una CPU ARM Cortex-M1. Xilinx ed Altera hanno invece sviluppato soluzioni proprietarie, i microcontrollori MicroBlaze e Nios-II. Ed è proprio del MicroBlaze che ci occuperemo in questa puntata, presentando una breve introduzione alla architettura del processore ed un

esempio di System-on-Chip basato su di esso. Utilizzeremo per questo la scheda di sviluppo per FPGA Spartan-3AN che abbiamo descritto nelle precedenti puntate del corso.

IL MICROCONTROLLORE MICROBLAZE DI XILINX

La **Fig. 2** mostra uno schema di principio dell'architettura del processore MicroBlaze, un sistema RISC (Reduced Instruction Set CPU) a 32 bit con architettura Harvard caratterizzata da spazi di indirizzamento separati per le memorie dati e programma. Ogni area ha una dimensione massima di 4 GByte; il processore vi supporta accessi a word, half-word e byte. Le due aree possono essere sovrapposte all'interno di una stessa memoria, così da consentire il debug del software applicativo. Il Register File consiste in 32 registri a 32 bit di tipo "general purpose"; sono poi previsti fino a 18 registri speciali.

L'intera architettura del processore può essere configurata in fase di sintesi mediante parametri generici, scegliendo le funzionalità necessarie alla propria applicazione e considerando quelle che sono le esigenze dell'utente in termini di prestazioni ed utilizzo di risorse interne; possono essere create fino a 70 diverse configurazioni!

La CPU, ad esempio, ha una architettura pipelined il cui numero di stadi può essere scelto tra 3 e 5. La più semplice configurazione in 3 stadi prevede le fasi di Fetch, Decode, ed Execute e assicura una capacità di calcolo dell'ordine di 0,95 DMIPs/MHz; è usata tipicamente per le FPGA di più piccole dimensioni, come i dispositivi della serie Spartan 3A DSP, dove raggiunge una massima frequenza di clock di 106 MHz. L'architettura in 5 stadi di pipeline utilizza invece i passi di Fetch, Decode, Execute, Access Memory e Writeback; nei dispositivi della serie Virtex-5 FXT raggiunge una frequenza di clock di 235 MHz. La capacità di calcolo è di 1,19 DMIPs/MHz.

La maggior parte delle istruzioni termina l'esecuzione in un singolo ciclo di clock; nel caso di istruzioni multi-ciclo la pipeline entra in stallo, come ad esempio durante l'accesso a memorie lente. In questo caso, per ridurre la penalità il processore implementa un buffer di prefetch delle istruzioni. Alla ripresa da una condizione di stallo, la successiva istruzione

da eseguire viene letta da tale buffer.

Il MicroBlaze ha un insieme di istruzioni ortogonale, nel senso che ogni istruzione è in grado di operare su dati accessibili mediante uno qualunque dei modi di indirizzamento supportati; supporta istruzioni di tipo A, con due operandi sorgente ed uno destinazione di tipo registro, e di tipo B con un operando sorgente di tipo registro, l'altro di tipo immediato ed un singolo operando destinazione di tipo registro. L'insieme di istruzioni include operazioni di tipo aritmetico, logico, salto e load/store, oltre ad istruzioni speciali. Non sono previste, invece, istruzioni di Input/Output; infatti il processore usa uno schema di I/O di tipo memory-mapped. Dispone, come mostrato in Fig. 2, di tre interfacce dedicate per l'accesso alla memoria :

- Local Memory Bus (LMB), che è un bus sincrono per l'accesso alle memorie on-chip disponibili all'interno della FPGA con latenza di un singolo ciclo di clock;
- Processor Local Bus (PLB), ossia un bus sincrono non multiplexato, multimaster, con protocollo di tipo pipelined, come definito all'interno dell'architettura CoreConnect di IBM ed impiegato per la connessione al processore di periferiche veloci ed a bassa latenza;
- Xilinx Cache Link (XCL), che è un bus sincrono punto-punto di tipo single-master single-slave impiegato per la connessione a controller di dispositivi di memoria esterna (SDRAM, DDR, DDR2). Tale interfaccia è tipicamente impiegata per gestire i trasferimenti tra la memoria esterna e la cache interna del processore (ove questa sia stata abilitata in fase di configurazione del sistema).

Il processore supporta modalità di accesso in memoria di tipo reale o virtuale. In modalità reale è usato l'indirizzo effettivo per accedere direttamente alla memoria; in modalità virtuale invece, l'indirizzo effettivo viene traslato in un indirizzo fisico mediante l'unità MMU (Memory Management Unit). La

modalità virtuale consente di rilocare dinamicamente in sistemi multi-task programmi e dati (il che permette di eseguire, ad esempio, programmi che richiedano quantità di memoria maggiore dello spazio indirizzabile dal processore) oltre a proteggere l'accesso alla memoria creando regioni private. La MMU del processore MicroBlaze (anche questa, qualora non sia necessaria, può essere disabilitata durante la fase di configurazione per ridurre l'occupazione di risorse del sistema) supporta dimensioni di pagina fino a 16 MByte e controllo indipendente per la traslazione e la protezione degli indirizzi per le memorie istruzioni e dati; la strategia di sostituzione della pagina è definita dal software di gestione ed è quindi dipendente dalla particolare applicazione.

Il MicroBlaze dispone inoltre di memoria cache separata per istruzioni e dati; la dimensione di tali memorie può essere configurata, al solito, mediante parametri generici. L'utilizzo di una cache istruzioni consente di migliorare le prestazioni nell'esecuzione di software applicativi residenti nello spazio di memoria esterno alla regione accessibile mediante bus LMB. La cache istruzioni è di tipo direct mapped (1-way associativa), con 4 o 8 word per linea. Cache con dimensioni di linea maggiori migliorano tipicamente le prestazioni nel caso di accessi sequenziali in memoria in quanto, in questo caso, nella cache viene precaricato

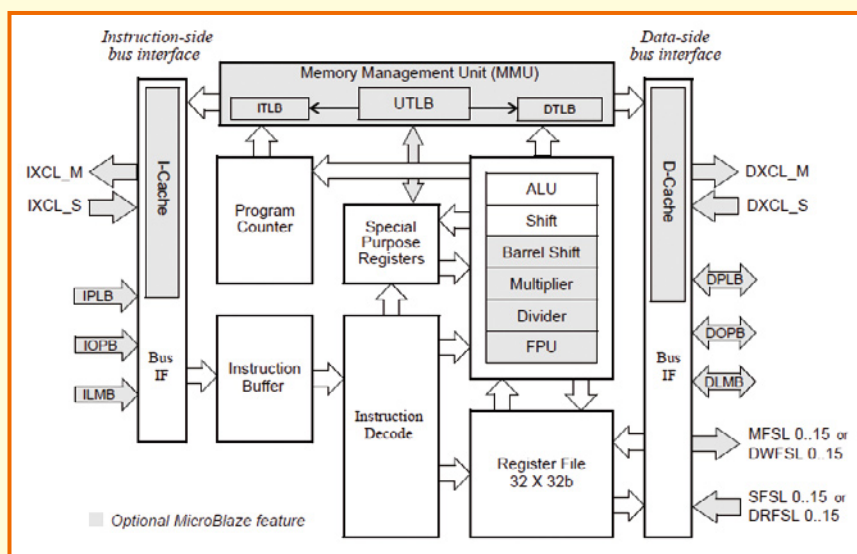


Fig. 2 - Architettura del processore MicroBlaze.

un numero maggiore di locazioni. Nel caso invece di applicativi che implementino uno schema piuttosto casuale di accesso alla memoria, l'impiego di cache con dimensioni di linea maggiore determina una frequenza di cache "miss" maggiore e quindi, probabilmente, un degrado della velocità di esecuzione del programma. È prevista la possibilità di abilitare dinamicamente la cache mediante accesso ad uno dei registri di configurazione della CPU, oltre alla disponibilità di istruzioni WIC per invalidare linee della memoria. La cache dati, analogamente a quella istruzioni, è di tipo direct-mapped (1-way associativa), 4 o 8 word per linea, di tipo write-through o write-back; possono essere associate alla cache solo aree di memoria non appartenenti allo spazio di indirizzamento riservato al bus LMB. Configurabili sono pure un Barrel Shifter ed un moltiplicatore in grado di eseguire le operazioni in un singolo periodo di clock, un divisore tra interi con latenza di 32 cicli ed una FPU (Floating Point Unit) per i calcoli in virgola mobile. Tale FPU è basata sullo standard IEEE 754, usa un formato in singola precisione (incluse le definizioni di infinito, not-a-number e zero) ed implementa la mo-

dalità round-to-nearest; supporta operazioni di addizione, sottrazione, moltiplicazione, divisione, confronto, conversione e radice quadrata. La catena di compilazione inclusa nell'ambiente di sviluppo EDK descritto in seguito, supporta nativamente tali operazioni in singola precisione, che sono quindi tradotte in istruzioni FPU; nella realizzazione di filtri FIR, ad esempio, si ottiene una accelerazione dell'esecuzione dei programmi di un fattore fino a 42. Le operazioni in virgola mobile in doppia precisione sono invece emulate via software. Oltre ai bus per l'accesso in memoria descritti in precedenza il processore MicroBlaze dispone di fino a 16 porte FSL (Fast Simplex Link), che possono essere abilitate in fase di sintesi. Si tratta di interfacce punto-punto unidirezionali per comunicazioni di tipo data streaming. Il processore accede a tali interfacce mediante le semplici operazioni put e get, che istruiscono il trasferimento dati da un registro sorgente alla porta stessa e viceversa. Le istruzioni possono essere eseguite nelle diverse modalità blocking data, non-blocking data, blocking control, and non-blocking control. Le porte FSL sono tipicamente impiegate per connettere acceleratori hardware dedicati al processore.

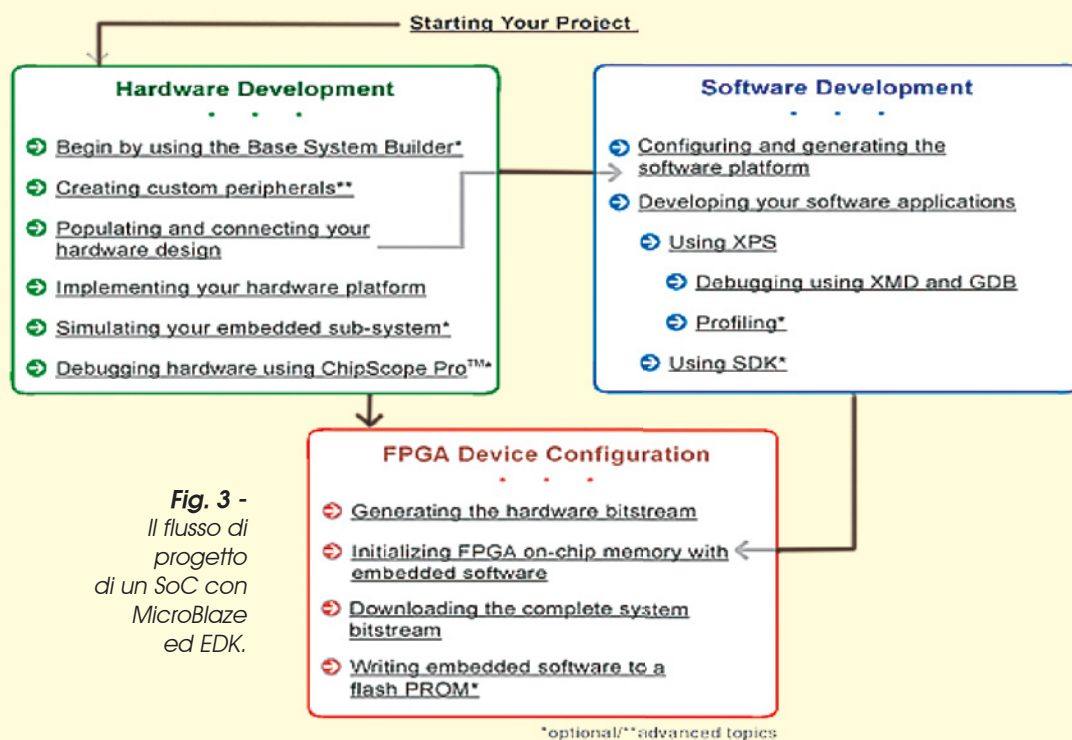


Fig. 3 -
Il flusso di
progetto
di un SoC con
MicroBlaze
ed EDK.

Sono inoltre disponibili interfacce dedicate per scopi di debug e trace. L'interfaccia di debug, in particolare, si connette alla periferica MDM (MicroBlaze Debug Module) che consente l'accesso alla porta JTAG del dispositivo FPGA per la comunicazione con un comune tool BDM (Background Debug Module) residente su una macchina host; la periferica MDM è in grado di supportare architetture multi-processore. Tra le capacità di debug rese disponibili vi sono la configurazione di un numero limitato di breakpoint e watchpoint hardware e di un numero illimitato di breakpoint software, il controllo step-by-step dell'esecuzione del processore, l'accesso in lettura e scrittura alla memoria, alla memoria cache, ai registri generici e speciali (seppure con alcune eccezioni). L'interfaccia di Trace consente invece di esportare una serie di segnali di stato interni per scopi di monitoring ed analisi delle prestazioni.

L'AMBIENTE DI SVILUPPO EDK

La Fig. 3 mostra uno schema di principio del flusso di progetto necessario per creare un SoC basato su microprocessore MicroBlaze. EDK (Embedded Development Kit) è l'ambiente di sviluppo fornito per gestire tale flusso. È contenuto nella distribuzione ISE Design Suite Embedded Edition; una versione di valutazione completamente funzionante per un periodo di prova di 30 giorni può essere scaricata previa registrazione dal seguente sito web: http://www.xilinx.com/ise_eval/index.htm. L'EDK contiene:

- XPS (Xilinx Platform Studio), l'ambiente grafico che permette di generare la configurazione hardware del nostro SoC selezionando da una ampia libreria di core IP le funzionalità necessarie alla propria applicazione;
- SDK (Software Development Kit), l'ambiente di sviluppo integrato con funzionalità avanzate e basato sulla piattaforma Eclipse per la creazione, la compilazione ed il debug del software applicativo.

Come al solito, il modo migliore per imparare ad usare un nuovo ambiente di sviluppo è forse quello di metterlo alla prova; senza indugi, vediamo quindi come realizzare la più classica delle applicazioni "Hello World!". Useremo come banco di prova la piattaforma di sviluppo per FPGA Spartan-3AN che abbiamo

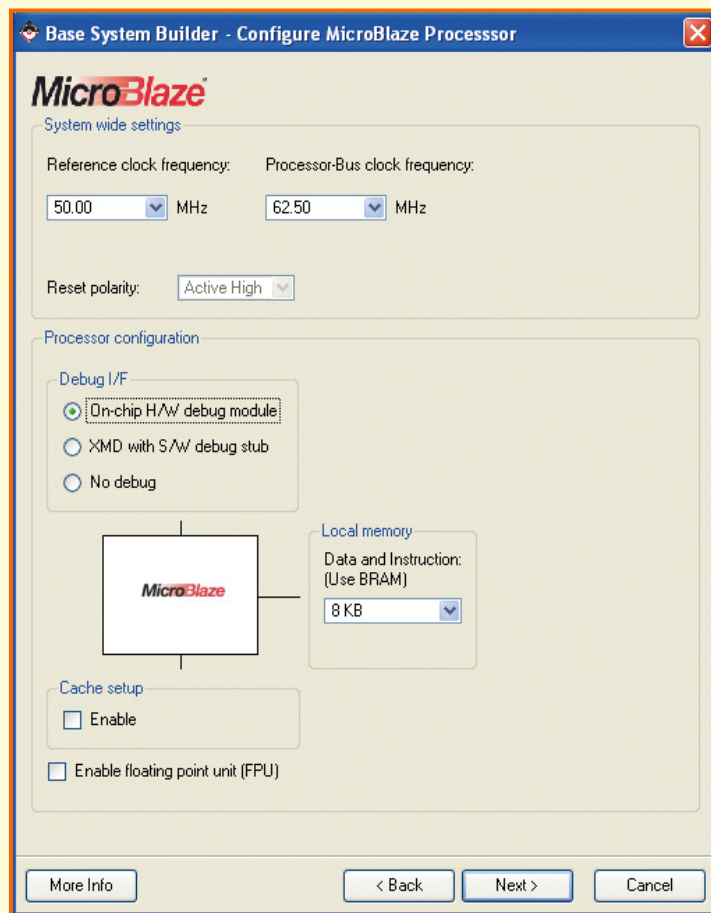


Fig. 4 - La finestra del Base System Builder per la configurazione del MicroBlaze.

presentato nelle precedenti puntate del corso con la versione 10.1 dell'EDK. Per brevità non potremo ovviamente vedere in dettaglio tutti gli aspetti del problema, ma le indicazioni riportate dovrebbero essere sufficienti a portare a termine in pochi passi il nostro progetto. Per una descrizione più dettagliata potrete consultare i manuali forniti unitamente all'ambiente di sviluppo, oltre alle diverse Application Notes che sono rese disponibili sul sito web di Xilinx (www.xilinx.com).

UN SEMPLICE ESEMPIO: HELLO WORLD!

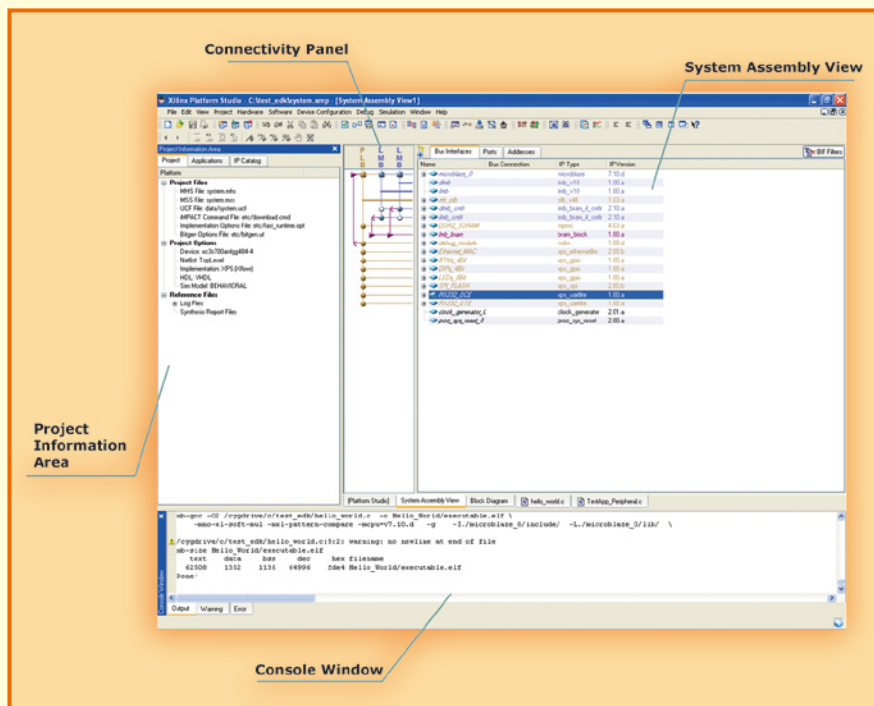
Configurare un SoC per FPGA Xilinx basato su processore MicroBlaze e creare un applicativo di esempio è estremamente semplice ed immediato se si usa la funzionalità di Base System Builder. Dopo aver lanciato l'ambiente di sviluppo EDK, compare un pannello iniziale che ci richiede se vogliamo creare un progetto nuovo o aprirne uno esistente. Selezioniamo la voce Base System Builder Wizard (recommended) quindi indichiamo il nome

Fig. 5 - Il pannello principale dell'ambiente XPS.

del progetto ed eventualmente la directory di destinazione sul disco rigido del nostro computer. Si aprirà una finestra di benvenuto; richiederemo ancora una volta la creazione di un nuovo progetto. Quindi selezioniamo la nostra scheda scegliendo:

- Board Vendor : Xilinx;
- Board Name : Spartan-3AN Kit;
- Board Revision : D.

Dopo aver confermato la scelta, si aprirà una serie di finestre (del tipo mostrato in Fig. 4) che ci permettono di volta in volta di configurare le caratteristiche della nostra CPU e delle periferiche ad essa associate. È possibile abilitare, ad esempio, la cache del processore o la generazione della FPU; possiamo aggiungere al nostro SoC fino a due porte di comunicazione UART, un controller ethernet o una porta SPI per l'accesso alla memoria flash seriale presente sulla scheda di sviluppo. Nel caso del nostro semplice esempio, basterà in realtà lasciare inalterate le selezioni predefinite che ci vengono proposte. Dopo un po', dovremmo finalmente ritrovarci all'interno dell'ambiente XPS (Xilinx Platform Studio) come mostrato nella Fig. 5, con il nostro progetto correttamente creato e configurato. Nell'area System Assembly View, sulla destra del pannello, appare la lista dei moduli e delle periferiche che sono presenti nel nostro SoC; l'area immediatamente alla sua sinistra, indicata come Connectivity Panel, mostra invece i bus di connessione che sono stati previsti. Si noti, ad esempio, la connessione delle porte UART indicate come RS232DCE ed RS232DTE al bus PLB del processore. Cliccando sulla tab Addresses della System Assembly View, il contenuto cambia; viene mostrato lo spazio di indirizzamento associato ad ogni periferica. Nella Project Information Area presente sulla sinistra del pannello di XPS, sono invece riportate le informazioni di progetto. Cliccando ad esempio sulla tab IP catalog, comparirà la lista di tutti i core IP



che sono disponibili e che potrebbero essere aggiunti al nostro progetto, laddove ve ne fosse la necessità. Per fare questo è sufficiente selezionare la periferica desiderata con il mouse e trascinarla sull'area System Assembly View. Sarà quindi necessario connetterla al bus del processore (si può fare facilmente utilizzando il menu a tendina che si apre cliccando sull'istanza del modulo all'interno della System Assembly View) e configurarne le caratteristiche (mediante l'apposita finestra che si apre facendo doppio click su questa stessa istanza). Una volta definita la configurazione hardware del nostro SoC, dobbiamo generare il bit-stream di configurazione della FPGA e creare l'applicativo. Selezioniamo quindi Hardware -> Generate Bitstream. Se il processo termina correttamente vedremo un messaggio **Done!** nella Console Window che si trova in basso all'interno del nostro ambiente. Tenete presente che questa fase di sintesi e place&route può durare anche qualche decina di minuti, a seconda della complessità del sistema che abbiamo configurato e delle prestazioni del nostro PC; abbiate quindi pazienza. Una volta creata la piattaforma hardware dobbiamo passare a creare il nostro applicativo. Per farlo possiamo usare le utility fornite all'interno dello stesso EDK oppure esportare il progetto verso SDK, l'ambiente di sviluppo basato su Eclipse cui si è accennato in precedenza. Per semplicità e facilità di uso adotteremo la prima soluzione. EDK include

un ambiente di compilazione per il processore MicroBlaze basato sulla catena GNU e che supporta codice scritto in linguaggio C e C++; include le librerie standard C, Math, GCC e C++. Selezioniamo dapprima Software -> Generate Libraries and BSPs per generare le librerie per la gestione delle periferiche ed il Board Support Package del nostro SoC. Quindi nella Project Information Area selezioniamo la tab Application. Troveremo già alcuni progetti software creati automaticamente dal Base System Builder per verificare il corretto funzionamento delle memorie del processore (TestApp_Memory) o delle periferiche accessibili (TestApp_Peripheral). Per creare una nostra applicazione è sufficiente selezionare Add Software Application Project. Si aprirà un popup dove inseriremo il nome del nostro progetto. Dopo aver confermato, il progetto apparirà tra gli altri all'interno della lista. Cliccando con il pulsante destro del mouse sulla voce **Sources**, aggiungiamo un nuovo file al progetto. Espandiamo quindi il menu a tendina sotto la stessa voce e clicchiamo due volte per aprire questo nuovo file. Editiamo il testo aggiungendo le poche righe di codice di seguito riportate:

```
#include "stdio.h"
```

```
int main (void) {
    printf("Hello world");
}
```

Infine salviamo il file. Quindi, dal menu di popup che si apre cliccando con il pulsante destro del mouse sul nome del nostro progetto all'interno della Project Information Area, selezioniamo "Mark to Initialize BRAMs". Infatti il MicroBlaze inizia ad eseguire dalla locazione 0x00 dopo il reset e nella configurazione creata automaticamente dal Base System Builder tale area è associata alla memoria embedded del processore. Selezionando "Mark to Initialize BRAMs" istruiamo il compilatore ad allocare in tale memoria le diverse sezioni del software compilato. Dallo stesso popup menu, selezioniamo infine la voce **Build Project** per compilare il nostro applicativo e generare il file *elf*. Se non ci sono errori, nella finestra di Output comparirà il messaggio **Done!** con le indicazioni proprio delle dimen-

sioni delle diverse sezioni (text, data, bss, dec) risultato della compilazione. Nel nostro caso non si pone alcun problema; nel caso di applicativi più complessi potremmo invece ricevere un messaggio di errore che ci indica che la memoria interna non è sufficiente a contenere tutte le sezioni. Tenerlo presente nel caso si sviluppino applicazioni proprie!

Avendo scelto di far eseguire il codice dalla memoria interna del processore, dobbiamo modificare infine il bitstream di configurazione della FPGA in modo che proprio questa memoria sia correttamente inizializzata. Selezioniamo dalla barra dei menu principali di XPS la voce Device Configuration -> Update Bitstream. A questo punto non ci resta che provare che il tutto funzioni. Connettiamo il nostro kit di sviluppo al PC mediante il cavo USB ed un cavo seriale; alimentiamo la scheda e apriamo sul PC un qualunque programma di gestione della porta seriale come Hyperterminal. Usiamo la seguente configurazione per la comunicazione : 9600 8-N-1. Da XPS selezioniamo infine dal menu principale Device Configuration -> Download Bitstream. L'FPGA viene configurata attraverso la porta USB. Se non si verificano errori, al termine del processo il nostro sistema inizierà a funzionare e dovremmo vedere sul nostro terminale apparire la scritta "Hello world!".

IL DEBUG DELL'APPLICATIVO

L'esempio che abbiamo visto è estremamente semplice. In una applicazione reale il nostro applicativo sarà più complesso e non è detto che funzioni correttamente da subito. Abbiamo bisogno quindi di poter fare il debug del codice che abbiamo scritto. La **Fig. 6** chiarisce le diverse configurazioni di debug che sono supportate. EDK include al suo interno il software XMD (Xilinx Microprocessor Debugger). Come si può vedere dalla figura, XMD controlla il processore target in modalità "hardware" oppure in modalità "software". Nel primo caso comunica con la periferica MDM descritta in precedenza e che si connette direttamente alla porta di debug del MicroBlaze; nel secondo, invece, è richiesto che il processore esegua il programma dedicato xmdstub che si può trovare nella lista delle applicazioni importate automaticamente in ogni nuovo progetto. In entrambi i casi la

comunicazione avviene su linea seriale o cavo JTAG. XMD rende poi disponibile una interfaccia TCL (Tool Command Language) che può essere usata per controllare il debug da riga di comando eventualmente mediante script. In alternativa supporta il controllo da remoto mediante il protocollo TCP definito da GDB, il debugger GNU. In questo secondo caso, per i debugger può essere usata un'interfaccia grafica. Il debugger può risiedere sulla stessa macchina dove è eseguito XMD (questa macchina è fisicamente connessa alla scheda target) oppure su una macchina remota. Nel nostro esempio, il Base System Builder, come possiamo vedere dalla finestra di System Assembly View del progetto, ha creato una configurazione del sistema che impiega la periferica MDM; per il debug useremo quindi la modalità hardware. Selezioniamo dalla barra dei menu il comando Debug -> XMD Debug Option... e verifichiamo che sia selezionato il tipo di connessione corrispondente. Nella Project Information Area, all'interno della sezione Applications, con il pulsante destro del mouse selezioniamo l'applicazione microblaze_0_bootloop e scegliamo l'opzione Mark to Initialize BRAMs. Quindi eseguiamo i comandi Device Configuration -> Update Bitstream e (avendo verificato che la scheda target sia accesa e connessa al PC mediante il cavo di configurazione USB) Device Configuration -> Download Bitstream. In questo modo l'FPGA verrà configurata ed il processore eseguirà al power-up un programma che lo forza in uno stato busy, evitando che si verifichino eccezioni che potrebbero precludere la corretta comunicazione con il debugger. Lanciamo quindi XMD eseguendo il comando Debug -> Launch XMD... Si aprirà una finestra DOS con la shell di XMD; controlliamo che non siano riportati messaggi di errore che indichino l'impossibilità di stabilire la connessione con il target. Come debugger useremo quello integrato in EDK. Eseguiamo il comando Debug

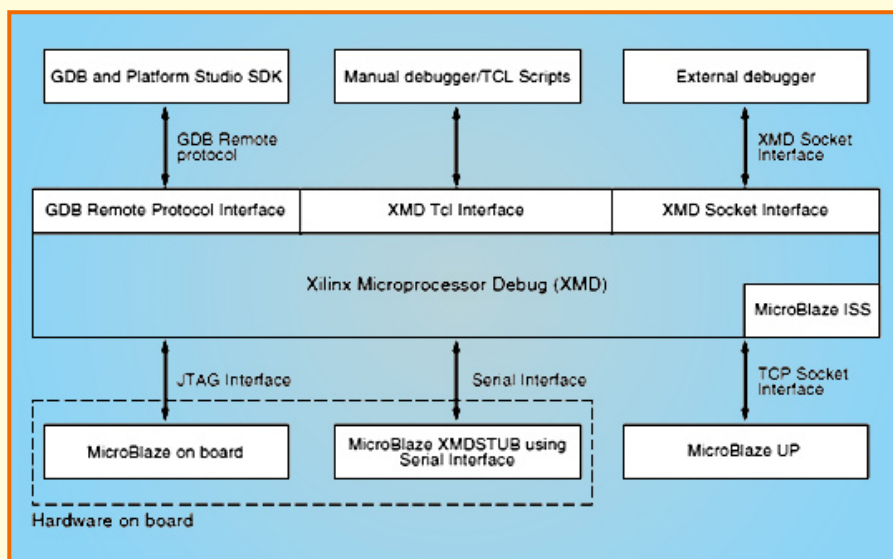


Fig. 6 - Diverse configurazioni di debug per processore MicroBlaze.

-> Launch Software Debug e selezioniamo la nostra applicazione dal menu a tendina che ci verrà presentato. Dopo aver confermato la selezione si aprirà infine la finestra principale. Dalla barra dei menu selezioniamo Run -> Connect to target e quindi Run -> Download. Il debugger si conatterà ad XMD e scaricherà il nostro applicativo nella memoria esterna del processore. Nella finestra principale sarà inoltre visualizzato il codice; come in ogni classico debugger è possibile introdurre dei breakpoint software in corrispondenza della esecuzione di una particolare istruzione. È possibile controllare quindi l'esecuzione del nostro applicativo passo-passo e visualizzare il contenuto dello spazio di memoria del processore e dei registri interni.

CONCLUSIONI

In questa puntata abbiamo mostrato come realizzare un SoC usando FPGA Xilinx. Utilizzando l'ambiente di sviluppo EDK e la libreria di core IP fornita con esso possiamo creare il microcontrollore che più si adatta alle nostre esigenze. La soluzione è corredata da un ambiente di sviluppo e debug del software applicativo del tutto autoconsistente. Si chiude così il nostro corso introduttivo alle logiche programmabili. Ovviamente non abbiamo potuto trattare tutti gli aspetti in dettaglio ma speriamo che le informazioni fornite e gli esempi proposti abbiano stimolato la curiosità del lettore verso un nuovo mondo dalla pressoché infinite possibilità. ■

FUEL-CELL ELETTRICITA' DALL'IDROGENO

di ARSENIO SPADONI

Auto elettriche capaci di percorrere centinaia di km, centrali poco inquinanti e silenziose da mettere in cantina, linee d'alta tensione e tralicci ridotti al minimo: questo ed altro possono darcelo le fuel-cell, una scoperta che ha quasi due secoli ma che è ancora nell'età dello sviluppo.

Da diversi anni, ed in particolare tra il pubblico più vasto da quando Beppe Grillo si fece riprendere in TV ad annusare il tubo di scappamento di un'auto dalla quale usciva solo vapore, si fa un gran parlare delle fuel-cell (o *pile a combustibile*, *celle a combustibile* o *pile a idrogeno* che dir si voglia) e di come potrebbero essere usate per alimentare automobili elettriche capaci di farci circolare senza inquinare, per centinaia di km, liberi dal peso e dall'ingombro della batterie e dall'incomodo di effettuare frequenti e noiose ricariche. In realtà le fuel-cell sono storia vecchia, perché

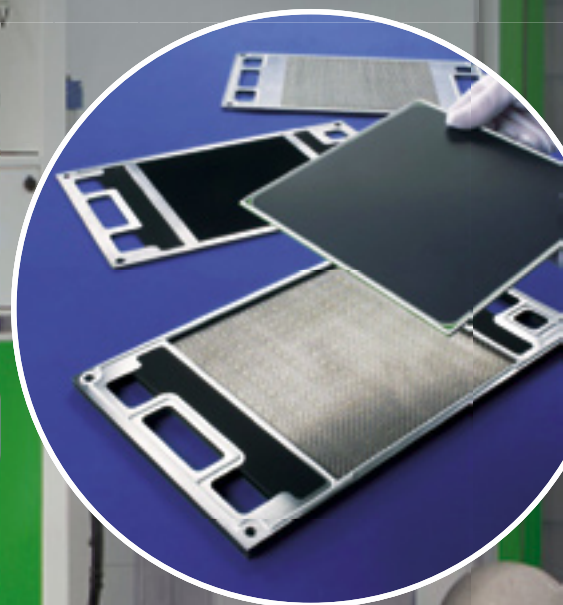
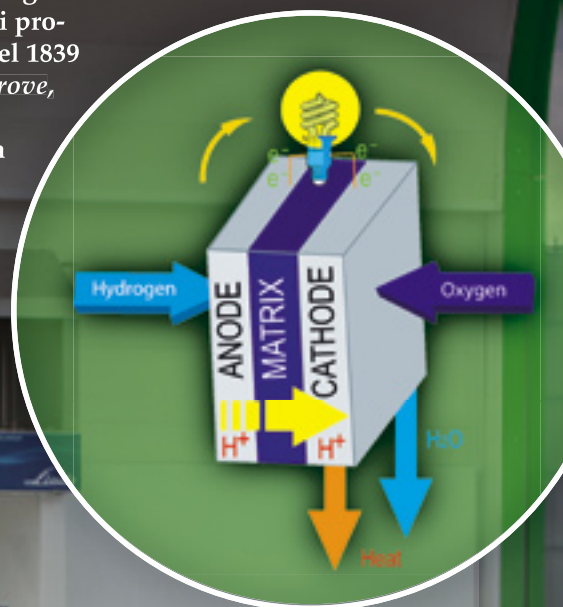
furono già usate nelle prime missioni spaziali e ancora si usano negli Space Shuttle. Attorno alle pile a combustibile si è creato un dibattito acceso: quello che più si sente dire è che sono la soluzione per cambiare radicalmente le auto e liberare le



grandi città dalla morsa dell'inquinamento prodotto dagli scarichi delle vetture a benzina o diesel. La realtà delle fuel-cell è che oltre ad essere ancora difficilmente realizzabili su larga scala e costose da produrre, non è vero che non inquinano: sicuramente non producono direttamente delle sostanze inquinanti, tuttavia per garantire ciò debbono essere alimentate con idrogeno puro, che è un gas non reperibile in natura e la cui produzione comporta inquinamento atmosferico e il dispendio di una gran quantità di energia. L'unico modo per ottenere idrogeno senza inquinare

consiste nel produrlo per elettrolisi, partendo dall'elettricità prodotta da fonti rinnovabili; ma in questo caso, che senso ha usare elettricità per produrre idrogeno dal quale ricavare elettricità? Questo ed altro ancora frenano la diffusione su vasta scala delle pile a combustibile. In queste pagine vedremo di analizzare

le varie tecnologie e ne valuteremo gli aspetti positivi e quelli negativi sulla base dei progressi fatti da quel 1839 in cui *William Grove*, giurista inglese con l'hobby della Fisica, scoprì per caso la pila a combustibile durante un esperimento di elettrolisi



Tipo di cella	Ioni mobili	Temperatura di utilizzo	Applicazioni e note
AFC	OH ⁻	50÷200 °C	Veicoli spaziali, sommergibili.
PEMFC	H ⁺	30÷100 °C	Autotrazione e applicazioni mobili
DMFC	H ⁺	20÷90 °C	Piccoli generatori portatili e auto.
PAFC	H ⁺	70÷220 °C	Sistemi di produzione combinata di potenza e calore (CHP) fino a 200 kW
MCFC	CO ₃ ²⁻	600÷650 °C	Sistemi CHP fino a qualche MW di potenza.
SOFC	O ²⁻	500÷1.000 °C	Sistemi CHP da qualche kW fino a centinaia di kW.

Caratteristiche delle fuel-cell a confronto.

dell'acqua, allorché si accorse che sospendendo l'erogazione di elettricità l'acqua si riformava e che tra gli elettrodi si verificava una differenza di potenziale. La pila così inventata fu battezzata "batteria voltaica a gas". La scoperta fu accantonata per oltre un secolo, perché poteva funzionare solo a condizione di alimentarla con idrogeno ed ossigeno puri, che a quel tempo era tutt'altro che facile e produrre e gestire. Il primo vero impiego delle fuel-cell fu nell'ambito della missione spaziale *Gemini*; il termine *fuel-cell* fu coniato proprio in occasione del varo del progetto *Gemini*. Anche le missioni *Apollo* e *Shuttle Orbiter* utilizzarono celle a combustibile. Negli ultimi decenni, visti i problemi di inquinamento atmosferico e in considerazione del fatto che le fonti fossili prima o poi finiranno, la fuel-cell viene vista come una possibile alternativa a carbone, gas naturale e petrolio da cui l'Europa entro il 2020 - se non si porrà rimedio agli attuali ritmi di crescita - ricaverà l'85 % dell'energia consumata (oggi siamo al 75 %). La fuel-cell è un generatore chimico di corrente elettrica continua alimentato da almeno due gas: un combustibile (idrogeno) e un ossidante (ossigeno o aria); nella fuel-cell entrano idrogeno e ossigeno ed escono corrente, acqua e calore. Di fuel-cell ne esistono

di vario tipo, basate su diverse reazioni chimiche. Come le normali pile, dispongono sempre di un *anodo* e di un *catodo* immersi o posti a contatto con un *elettrolito*; per facilitare la reazione chimica, l'anodo e il catodo possono essere rivestiti da un materiale catalizzatore. Negli anni, gli studiosi hanno provato ad usare vari elettroliti; sono così nate le fuel-cell ad elettrolito alcalino (AFC) ad acido solforico (SAFC) ad acido fosforico (PAFC) a carbonati fusi (PAFC) a membrana a scambio di protoni (PEMFC) e ad elettrolito basato su ossidi solidi (SOFC).

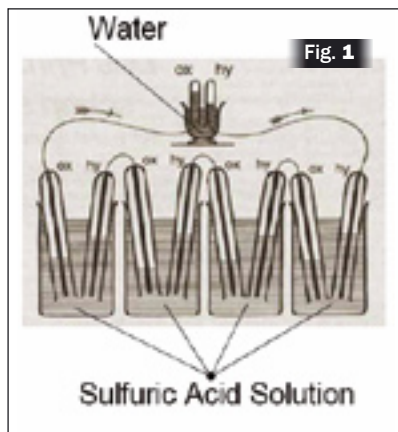
Per ottenere la corrente elettrica, almeno nelle fuel-cell con membrana a scambio di protoni, si fa entrare dell'idrogeno dalla parte dell'anodo; per effetto del catalizzatore che riveste questo elettrodo, l'idrogeno viene scisso in due atomi e passa dalla forma molecolare (H₂) a quella ionica (H⁺) scoprendo una carica elettrica libera. Di ogni atomo d'idrogeno, mentre il nucleo (composto dal solo protone) passa nell'elettrolito, l'elettrone deve prendere un'altra strada; realizzando parte dell'elettrodo in metallo, gli elettroni si addensano su di esso (tipicamente il catalizzatore è un metallo, quindi è su di sé che si addensano gli elettroni che non possono attraversare l'elettrolito). I protoni arrivano dall'altro lato

della pila ed entrano in contatto con il catodo, dove il catalizzatore qui presente ha scomposto le molecole biatomiche di ossigeno in ioni ossigeno O⁻; qui dovrebbe innescarsi la reazione chimica tra idrogeno e ossigeno, solo che non avviene perché all'idrogeno mancano gli elettroni. Questi ultimi possono arrivare collegando con un filo elettrico la parte metallica dell'anodo a quella del catodo, eventualmente interponendo un utilizzatore; a questo punto può avvenire la reazione e gli elettroni che vengono richiamati dal catodo determinano il flusso che forma la corrente. La prima fuel-cell, ossia quella di *Grove*, era realizzata da una soluzione di acido solforico che faceva da elettrolito e conteneva parzialmente immersi due elettrodi di platino, ognuno dei quali risultava chiuso superiormente da un cilindro di vetro aperto nella parte inferiore, la quale era immersa sotto la superficie dell'elettrolito (**Fig. 1**); in un cilindro veniva introdotto idrogeno e nell'altro ossigeno. Una cella siffatta è di tipo SAFC, ossia *Sulfuric Acid Fuel Cell* (cella a combustibile ad acido solforico). Al suo posto sono state sperimentate e realizzate altre tipologie; quelle che oggi rivestono maggiore interesse sono cinque, ognuna delle quali presenta proprie caratteristiche e un diverso grado di sviluppo: PAFC (*Phosphoric Acid Fuel-Cell*), MCFC (*Molten Carbonate Fuel-Cell*), SOFC (*Solid Oxide Fuel-Cell*), PEMFC (*Proton Exchange Membrane Fuel-Cell*) e AFC (*Alkaline Fuel Cell*). Una distinzione si può fare per temperatura di funzionamento, in dispositivi a bassa/media e alta temperatura: della prima categoria fanno parte la PEM e le sue varianti (ad esempio la DMFC) oltre alle PAFC e AFC; delle fuel-cell ad

alta temperatura fanno parte la SOFC e la MCFC. Vediamo per prime le celle a bassa temperatura, partendo dalla PEM, che utilizzano come elettrolito una speciale membrana sintetica o una lamina di polimero, che permette solo il passaggio dei protoni. Le PEM sono le celle più indicate per l'autotrazione e quelle che fanno maggiormente sperare, seppure presentano ancora molti problemi pratici derivanti innanzitutto dal costo della membrana (il quale si può abbattere usando materiali polimerici, che però hanno minore durata) e poi dalla delicatezza dell'insieme, causata sia dall'acqua che si forma abbondantemente nella zona di catodo e che può allagare la membrana impedendo il transito degli ioni, sia dal fatto che l'eventuale presenza di anidride carbonica e monossido di carbonio avvelenano (contaminano) il catalizzatore in platino. Per questo motivo le PEM necessitano di idrogeno purissimo. Una variante della PEM è la cella DM (*Direct Methanol*) che funziona a metanolo e ricava internamente l'idrogeno che le occorre. Le celle a bassa temperatura funzionano a $30\div 100\text{ }^{\circ}\text{C}$ (le DMFC, intorno ai $50\text{ }^{\circ}\text{C}$) e sono adatte a realizzare generatori di piccolissime dimensioni.

Quanto alle AFC, sono basate su un elettrolito alcalino (idrato di potassio, KOH) trattenuto in una matrice posta fra i due elettrodi, che vengono addizionati con platino; durano a lungo perché l'elettrolito non corrode i metalli di anodo e catodo. Nella AFC il funzionamento si basa sul passaggio di ioni OH^- nell'elettrolito; la temperatura di funzionamento è tra $50\text{ e }200\text{ }^{\circ}\text{C}$. Sviluppata negli anni '60 del secolo scorso per le missioni spaziali, è adatta a produrre basse potenze. Oggi,

causa il costo del platino e la necessità di essere alimentata da idrogeno purissimo e aria depurata dalla CO_2 , non si usa quasi più. Un'altra fuel-cell funzionante a media temperatura è la PAFC: viene alimentata con gas ricco di idrogeno (metano, GPL, gas di carbone) oppure metanolo e aria (in questo caso si parla di cella DM-PAFC) e funziona a temperature che oscillano tra $150\text{ e }230\text{ }^{\circ}\text{C}$; anche nelle PAFC, l'idrogeno in arrivo all'anodo viene scomposto in forma ionica e gli ioni positivi H^+ migrano attraverso l'acido fosforico per arrivare al catodo, dove incontrano gli elettroni passati dal collegamento elettrico. La PAFC è adatta per generatori a basse e medie potenze. Una delle pile a combustibile ad alta temperatura più note è la MCFC, che si può alimentare con aria e gas più scadenti del metano e anche meno puri, e riesce a tollerare la presenza di ossido di carbonio prodotto dalla combustione degli idrocarburi e presente nell'aria. Come elettrolito impiega carbonati di litio e potassio e funziona intorno ai $600\text{ }^{\circ}\text{C}$. È la pila più adatta a realizzare generatori di alta potenza,



quindi per costituire vere e proprie centrali elettriche. La MCFC può ricavare da sé l'idrogeno che le serve; inoltre, il catalizzatore può essere di materiale molto economico (nichel) grazie al fatto che le alte temperature agevolano la dissociazione dell'idrogeno molecolare.

L'ultima cella di cui ci occupiamo è la SOFC, ossia quella che impiega come elettrolito un ossido solido; diversamente da tutte le altre non richiede catalizzatore, grazie all'elevatissima temperatura di funzionamento, che arriva anche a $1.000\text{ }^{\circ}\text{C}$. La SOFC è adatta a realizzare impianti di medie dimensioni e accetta all'ingresso gas metano o miscele contenenti idrogeno e ossidi di carbonio. La cella base di qualsiasi pila a combustibile ha una struttura piatta a tre strati, di cui quello centrale, compreso fra il catodo e l'anodo, costituisce o contiene l'elettrolito. Alcune celle funzionano in orizzontale, altre in verticale. Le superfici affacciate devono avere un'area sufficiente per ottenere intensità di corrente adeguate alle esigenze applicative; si può così arrivare a superfici dell'ordine del metro quadrato

Celle a bassa e media temperatura: **PEM, AFC, DMFC**

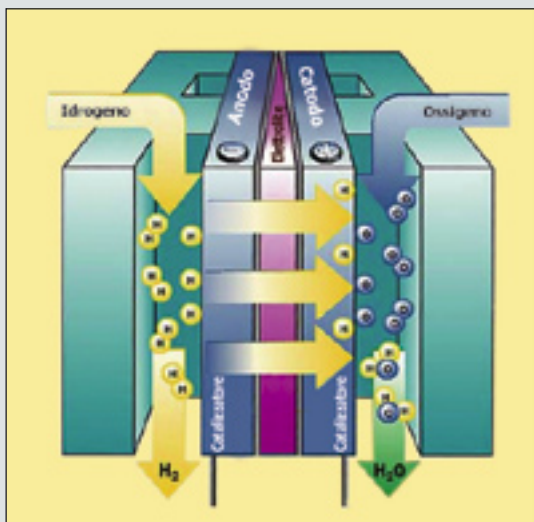


Le PEMFC (*Proton Exchange Membrane Fuel Cell*) anche dette SPFC (*Solid Polymer Fuel Cell*, ossia celle ad elettrolito polimerico solido) sono dotate di una membrana detta "a scambio di protoni" perché di ogni atomo lascia passare i protoni; proprio questa

caratteristica fa sì che dalla membrana passino gli ioni idrogeno e che gli ioni ossigeno restino, invece, dalla loro parte: infatti, ogni ione ossigeno è formato da protoni e neutroni. La PEM è realizzata da una membrana di *Nafion* (un composto brevettato dalla Dupont) sulle superfici della quale sono depositi gli elettrodi, realizzati dal catalizzatore in platino; gli elettrodi devono essere porosi (gas-diffusion) per lasciar penetrare l'idrogeno da un lato e l'ossigeno dall'altro. La membrana deve essere ben idratata; a ciò provvede l'acqua che si forma durante il funzionamento, che in parte va drenata, altrimenti allaga

la membrana impedendo il passaggio dei gas. Dato che le temperature di funzionamento sono vicine a quella ambiente, per disperdere il calore occorrono impianti di raffreddamento che consumano parte dell'elettricità prodotta. Il tempo di avviamento di una PEM è compreso tra circa 2 e 3 minuti. La fuel-cell ad elettrolito polimerico è quella sulla quale viene investito il maggior sforzo in termini di ricerca; ciò perché offre una densità di potenza maggiore rispetto alle altre fuel-cell (con l'unica eccezione dell'alcalina AFC, che però è troppo delicata). Le fuel-cell PEM rispondono bene ai cambiamenti di carico elettrico ed è perciò che risultano adatte all'utilizzo nell'ambito dei trasporti leggeri, per la generazione di energia elettrica per utilizzi domestici e nell'ambito della navigazione spaziale. Oltre al costo del *Nafion*, per realizzare le PEM bisogna fare i conti con il prezzo del platino, che ha spinto l'industria a cercare nuovi materiali, quali i polimeri solidi con cui si costruiscono le SPFC e composti basati su politetrafluoroetilene (PTFE); le membrane così realizzate integrano il

materiale catalizzatore e quindi non necessitano di platino. Il miglioramento delle tecnologie produttive ha comunque permesso di ridurre il contenuto di platino: qualche anno fa servivano 16,8 mg di catalizzatore per kW di potenza elettrica prodotta; oggi ne bastano 0,98 mg per kW, che si traducono in 25 € al kW. Ciò significa che per produrre una fuel-cell da 30 kW, il costo del solo catalizzatore si aggirerebbe sui 750 euro; adottando i polimeri solidi si può scendere a una media di 140 euro. In alternativa al platino, nelle PEM possono essere usati catalizzatori di metallo non nobile (ad esempio il nichel) sottoponendo a pirolisi (scissione ad opera del calore) a 900 °C l'acetato di ferro in una miscela di gas argento/idrogeno/ammoniaca. L'unico combustibile con cui le PEMFC possono funzionare è l'idrogeno puro: anche una piccola quantità di CO può danneggiare irreparabilmente il catalizzatore dell'idrogeno il quale deve anche essere privo di ossigeno; infatti se ossida si riduce la sua capacità catalitica e si verifica una perdita di rendimento. Le DMFC sono l'ideale per l'uso automobilistico e



per ogni cella elementare. Ogni cella sviluppa una tensione di valore compreso tra mezzo volt e un volt, ma tipicamente fornisce 0,7 V. Per ottenere tensioni maggiori si realizzano strutture chiamate *stack* (catasta) dove più celle vengono sovrapposte una all'altra, interponendo uno strato isolante e collegandole in serie (mediante elettrodi chiamati "piatti bipolari"). Quindi per fuel-cell si intende la cella elementare, mentre uno *stack* è un sistema formato da più

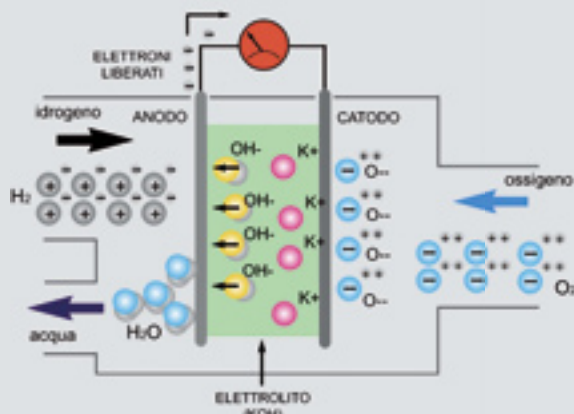
celle uguali. Per quanto siano propagate come generatori ecologici, le fuel-cell inquinano, anche se non tutte, pur se meno dei tradizionali sistemi di produzione dell'energia e delle automobili.

Le uniche a non sviluppare inquinanti sono le PEM e le AFC, ma l'idrogeno che serve loro deve essere ottenuto con processi che comportano l'uso di idrocarburi, calore e sviluppo di inquinanti quali CO e CO₂. Le altre celle rilasciano anidride carbonica.

EFFICIENZA DELLE FUEL-CELL

Per valutare l'uso delle pile a combustibile in luogo dei tradizionali sistemi di produzione dell'elettricità, bisogna avere un'idea di quanto idrogeno occorra per ricavare una certa corrente. Questa dipende da quanti elettroni si riescono a liberare dall'idrogeno per metterli in circolo e dirigerli verso l'anodo, ovvero dalla quantità di idrogeno introdotta nell'anodo e dalla capacità del catalizzatore di scindere il gas. Inoltre, an-

per i piccoli generatori portatili e fissi, anche in virtù della buona durata (40.000 ore). Una variante della PEM è la DMFC (*Direct Methanol Fuel Cell*) il cui elettrolito è un polimero rivestito dal lato anodico da un catalizzatore che riesce ad estrarre l'idrogeno dal metanolo. Le piccole dimensioni delle DMFC, spesso simili a una scatola di fiammiferi, le rendono adatte a essere utilizzate come batterie per strumenti portatili; non a caso sono stati di recente sviluppati moduli per PC notebook e simili. La tecnologia delle celle ad uso diretto di metanolo è in continuo sviluppo e si sta tentando la reazione con altri tipi di alcol. L'efficienza teorica della cella DMFC è del 40 %, ma le attuali applicazioni pratiche non riescono ad assicurare più del 20 %. Per ora non è chiaro se il metanolo in eccesso, uscito dalla fuel-cell perché non ha reagito, possa essere riciclato; in questo caso l'efficienza della pila aumenterebbe. Oltre alle PEM, esistono PAFC in grado di riformare il metanolo e alimentarsi con l'idrogeno che ne deriva. Le PEM DM sarebbero l'ideale per l'auto, dato



Schema di funzionamento della PAFC.

che consentirebbero il rifornimento con un liquido, sicuramente più facile da trattare dell'idrogeno. Quanto alle AFC, si tratta di celle a media temperatura, prime ad avere un'applicazione pratica, dato che circa 50 anni fa hanno trovato impiego a bordo delle prime navicelle spaziali; oggi vengono usate nella tecnologia spaziale e nei motori di sottomarini. La durata di vita è limitata, in quanto si manifesta un rapido decadimento delle prestazioni: ogni 1.000 ore di funzionamento la tensione di ciascun elemento cala di 15÷50 mV. L'elettrolito è una soluzione di idrossido di potassio (KOH) più o meno concentrata, trattenuta in una matrice solida. La fuel-cell alcalina usa idrogeno ed aria necessariamente depurata da anidride carbonica. Sebbene usando soluzioni

di KOH concentrate (85%) sia possibile raggiungere temperature di lavoro superiori a 250 °C, attualmente le celle AFC lavorano intorno ai 100 °C, adottando soluzioni a basse concentrazioni di KOH (35÷50 %). L'efficienza delle AFC può spingersi fino al 60 % (70 % usando il calore prodotto) quindi è molto alta, più di quella delle PEM. Il tempo di avvio è molto ridotto (un paio di minuti). La fuel-cell di tipo AFC funziona così: l'idrogeno entra dalla parte dell'anodo e l'ossigeno raggiunge l'area del catodo; il catalizzatore decompone le molecole d'idrogeno in due ioni H+ e gli elettroni vanno dall'anodo al catodo generando una corrente elettrica che può fluire chiudendo gli elettrodi con un utilizzatore elettrico. Perché



ci sia corrente, sul catodo deve avvenire una reazione chimica, che in questo caso coinvolge ioni OH- contenuti nell'elettrolito; per l'esattezza, gli ioni di ossigeno che si sono formati per effetto del catalizzatore reagiscono con l'acqua dell'elettrolito formando ossidrili (OH-) i quali attraversano l'elettrolito e migrano all'anodo. Qui, gli ioni idrossido reagiscono con gli ioni idrogeno formando acqua; una parte dell'acqua ritorna al catodo dove è disponibile per la successiva reazione. Le celle AFC usano come combustibile l'idrogeno, ma in alcuni casi anche idrazina, che è un combustibile per missili a base di idrogeno, quindi molto leggero a parità di energia rilasciabile.

che liberando tutti gli elettroni, questi potranno raggiungere il catodo in funzione delle proprietà capillari degli elettrodi e della conducibilità dell'elettrolito. La corrente è funzione della carica elettrica formata dagli elettroni liberati nella reazione chimica (Q , espressa in coulomb) e fatta circolare in un conduttore nell'unità di tempo t (espresso in secondi):

$$I = Q / t$$

La carica è direttamente propor-

zionale alla quantità di idrogeno nella zona anodica, anche se bisogna tener conto della quantità di gas che si riesce a introdurre e a dissociare.

La carica dell'elettrone è $1,6 \times 10^{-19} \text{C}$ e il numero di elettroni su cui si può contare è $6,02293 \times 10^{23}$ ogni mole (22,17 l) di idrogeno; la corrente teoricamente ottenibile da una pila a combustibile nella quale viene introdotta una mole di idrogeno al secondo è pari a 96.487 A. Nella pratica non è così, anche perché la tensione sotto

carico scende a meno di 0,7 V (Figura 2).

IMPIEGHI DELLE FUEL-CELL

Ad oggi con le fuel-cell sono state realizzate piccole centrali elettriche e gruppi di cogenerazione; ciò vale in special modo per le celle operanti ad alte temperature, dalle quali è possibile ricavare vapor d'acqua da utilizzare per azionare turbine collegate ad alternatori e quindi ricavare ancora elettricità: in questo caso è stato possibile ottenere rendimenti

dell'ordine del 70÷75 %, certamente superiori a quelli di un generatore elettrico con turbina a gas (che non supera il 40 %) o motore alternativo a combustione interna. Un aspetto tra i più interessanti del generatore di energia elettrica basato su celle a combustibile è che può essere installato, grazie al fatto che è silenzioso e non inquinante, nelle aree urbane e quindi vicino all'utenza, risparmiando non poco sul costo di trasporto dell'energia elettrica (che incide per almeno il 5 %).

Nella realizzazione di piccole centrali elettriche è stato necessario diversificare le tecnologie, in base allo scopo dell'applicazione: per i piccoli impianti (centrali da usare nelle case e per la produzione locale di potenze da pochi kW a un centinaio di kW) si è visto che risultano convenienti le PAFC e le SOFC; per potenze da alcune centinaia di kW ad alcuni MW, le MCFC hanno dimostrato risultati interessanti. Un altro settore dove molto si è speso per la ricerca è quel-



Le SOFC funzionano ad $800 \div 1.000^{\circ}\text{C}$ e come elettrolito impiegano ossidi solidi, che sono stabili e non corrodono gli elettrodi, il che si traduce in una lunga durata delle celle SOFC. Queste fuel-cell sono considerate le più sofisticate tra quelle di media potenza (fino a poche centinaia di kW) perché risultano di semplice costruzione, sono efficienti e non richiedono idrogeno puro (la presenza di CO e CO₂ non è un problema). L'elettrolito è formato da ossido di zirconio stabilizzato con ossido di ittrio; il catodo è invece costituito da manganito di lantanio opportunamente trattato, mentre l'anodo è realizzato con un materiale ceramico chiamato cermet, a base di nichel-ossido di zirconio. Una peculiarità molto importante delle celle solid-oxide è che non necessitano l'uso né del catalizzatore, né del circuito di raffreddamento indispensabile negli altri tipi ad asportare il calore prodotto, che, almeno nei piccoli sistemi, serve per incentivare la reazione di reforming del combustibile

necessaria ad estrarre l'idrogeno. Come la cella MCFC, la SOFC presenta un'elevata efficienza: oltre il 60%. Tuttavia la necessità di trattenere il calore per effettuare il reforming, riducono le possibilità di cogenerazione e di recupero del calore per creare elettricità mediante turbine a vapore. Il principale inconveniente di questo tipo di fuel-cell è la grande dimensione, il che designa la SOFC per la realizzazione di centrali elettriche. Inoltre per fare entrare il sistema in temperatura (almeno fino a 650°C , cosa ottenuta utilizzando un bruciatore) sono richiesti diversi minuti, il che limita l'utilizzo di tali sistemi alla realizzazione di centrali elettriche di piccole dimensioni. Va però detto che la SOFC non soffre della lentezza delle reazioni chimiche che affligge altri tipi di fuel-cell, in quanto qui la riduzione dell'ossigeno non richiede l'uso di catalizzatori; ciò significa che segue prontamente le variazioni di carico elettrico e si presta a realizzare generatori fatti per lavorare singolarmente

negli edifici. Il principio di funzionamento delle SOFC, che si basa sulla migrazione degli ioni ossigeno (prodotti sul catodo per scissione dell'ossigeno molecolare ad opera del caldo) attraverso le lacune presenti nella struttura cristallina che contraddistingue gli ossidi solidi, verso l'anodo. Qui gli ioni ossigeno arrivati dal catodo si combinano con quelli idrogeno formati sull'anodo e producono acqua (subito vaporizzata) ed elettroni liberi, che passano dall'elettrodo e, tramite i collegamenti e l'utilizzatore, giungono al catodo, dove concorrono alla creazione di nuovi ioni. Le fuel-cell ad ossidi solidi vengono costruite in modo da sfruttare le alte temperature e il fatto che l'acqua si sviluppa non sul catodo ma sull'anodo, per effettuare il reforming di idrocarburi e gas contenenti idrogeno e ossidi di carbonio; ciò fa delle SOFC celle altamente economiche perché possono essere abbinate ad impianti chimici che producono idrogeno impuro o dove vi sia la rete del metano. Chiaramente le SOFC producono CO e CO₂, ma la quantità è, a parità di elettricità sviluppata, di gran lunga minore di quella emessa dai tradizionali generatori (ad esempio le turbine a gas) anche comprendendo l'inquinamento prodotto dal bruciatore che serve all'avvio. Nelle SOFC l'idrogeno si ricava per steam-reforming o water-shift; nel primo caso il CO viene trasformato in idrogeno e CO₂ con l'aiuto del vapore acqueo formato

Le celle ad alta temperatura: SOFC ed MCFC

sul catodo. Con il metano, invece, si realizza lo steam reforming. In base alla loro struttura, le SOFC si possono suddividere in *substrate-supported* e *self-supported*. La geometria adottata per le SOFC varia da un costruttore all'altro: c'è quella *planare* e quella *tubolare*, sviluppata dalla Westinghouse, che è senza dubbio quella migliore. Planare e tubolare sono le soluzioni più semplici; esistono poi configurazioni più avanzate come quella a piatti monolitici e a single-chamber, per ora in fase di studio. Costruttori di SOFC sono anche la Hexis svizzera, l'americana Versa Power System e l'australiana CFCL.

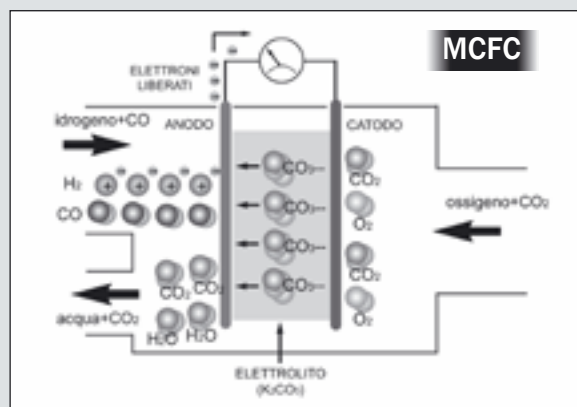
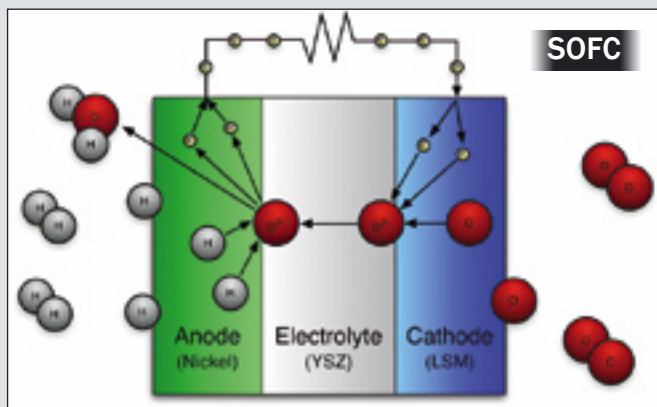
Oltre alle SOFC, rientrano nella categoria delle celle ad alta temperatura le MCFC, anche dette celle a carbonati fusi perché usano come elettrolito una miscela di carbonati alcalini (tipicamente di litio e di potassio, ma anche di sodio) trattenuta in una matrice porosa costituita da alluminato di litio; funzionano grazie alla migrazione di ioni carbonato. Una volta riscaldati ad una temperatura intorno a 650 °C, i sali (carbonati) si fondono e diventano elettricamente conduttori: gli ioni carbonato (CO_3^-) scendono dal catodo all'anodo, dove si uniscono con l'idrogeno

formando acqua, anidride carbonica ed elettroni, che vengono indirizzati tramite un circuito esterno di nuovo al catodo, generando corrente elettrica e calore. Il pregio delle MCFC è che possono essere realizzate con materiali economici e reperibili in grandi quantità, quindi si prestano alla realizzazione di grandi centrali elettriche. Gli elettrodi di una MCFC sono entrambi a base di nichel: il catodo impiega ossido di nichel litato (è nichel poroso con 1÷2 % di litio) mentre l'anodo usa nichel con piccole percentuali (fino al 10 %) di cromo. Escludendo la manutenzione, il costo d'esercizio è bassissimo, dato che il combustibile richiesto può essere qualsiasi derivato del petrolio o il metano, in quanto le alte temperature degli elettrodi permettono l'estrazione (riformazione) diretta dell'idrogeno.

La reazione chimica che porta alla liberazione degli ioni carbonato si innesca con gli ossidi di carbonio: l'idrogeno scomposto dall'azione del catalizzatore in ioni H^+ reagisce con gli ioni carbonato che partendo dall'elettrolito sono approdati all'anodo stesso; la reazione apporta due atomi d'idrogeno ed uno ione carbonato e determina, per ogni molecola d'idrogeno scissa ed ogni ione carbo-

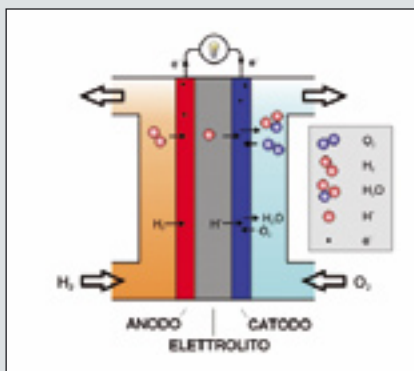
nato, una molecola d'acqua, una di anidride carbonica, ma libera due elettroni, che tramite l'elettrodo finiranno (attraverso l'utilizzatore elettrico) al catodo. Qui giungono ossigeno e anidride carbonica; il primo viene ridotto dal catalizzatore e si combina con la CO_2 , riformando gli ioni carbonato che sono migrati verso l'anodo, grazie agli elettroni in arrivo dall'anodo. Questo vale per le celle alimentate a idrogeno da un lato e ad ossigeno e anidride carbonica dall'altro. Nelle MCFC si può usare anche metano, grazie al reforming effettuato con l'aiuto del catalizzatore in nichel: le molecole H_2 si scindono in ioni H^+ ed elettroni. All'anodo avviene l'ossidazione del monossido di carbonio, che forma ulteriore combustibile (rilascia idrogeno e CO_2). L'acqua prodotta sull'anodo viene reinviata al catodo per favorire le reazioni precedenti. La corrosività dell'elettrolito rappresenta uno dei principali problemi delle MCFC, compensati però, oltre che dalla possibilità di impiegare idrocarburi invece dell'idrogeno puro, l'efficienza ottenibile, che supera il 60 %, dato che si può recuperare il calore prodotto in impianti di cogenerazione, dove, oltre all'elettricità si genera liquido caldo che può essere utilizzato per il teleriscal-

damento, ma anche per la produzione di ulteriore elettricità mediante alternatori mossi da turbine a vapore. A causa della lentezza delle reazioni in gioco, le MCFC si avviano e rispondono lentamente; perciò sono adatte a realizzare centrali elettriche dotate di sistemi di ripartizione del carico. Ultimamente la ricerca ha sviluppato nuovi elettroliti, il che ha permesso un allungamento della vita a 15.000 ore. Pur richiedendo affinamenti tecnologici, le MCFC sono le più interessanti per la produzione di grandi quantità di elettricità; allo scopo sono oggetto di importanti programmi di sviluppo, perseguiti negli Stati Uniti da ERC ed MC Power. In Giappone è stato avviato il NEDO e in Europa abbiamo ARGE, MOLCARE, ECN. In particolare, il MOLCARE (MOLten CARbonate Europe) guidato da Ansaldo Ricerche, ha portato allo sviluppo di un generatore modulare da 500 kW. Ansaldo Fuel Cells è la più importante azienda europea che realizza MCFC; nel resto del mondo troviamo FuelCell Energy e GenCell Corporation (U.S.A.) CFC Solutions (Germania) Ishikawajima-Harima Heavy Industries (Giappone) POSCO/KEPCO consortium and Doosan Heavy Industries (Corea).



La cella di tipo **PAFC**

Sono fuel-cell a media temperatura di funzionamento, che usano come elettrolito una soluzione di acido ortofosforico di cui è imbevuta una matrice di carburo di silicio posta fra due elettrodi porosi di grafite. Sono state le prime fuel-cell prodotte in grandi quantità e ad aver trovato un diffuso impiego in impianti fissi per la produzione simultanea di elettricità e calore (cogeneratori) da 200 kW, in Giappone, USA ed Europa. Le PAFC lavorano ad una temperatura compresa tra i 150 e i 220 °C ed hanno una buona efficienza: 40-45 %. Il catalizzatore in questo caso è il solito platino. Dato che l'acido fosforico è abbastanza corrosivo, il materiale con cui realizzare gli elettrodi ed il catalizzatore va scelto con cura. La possibilità di usare idrogeno di scarsa qualità rende le fuel-cell ad acido fosforico adatte a realizzare piccole centrali elettriche nelle industrie chimiche dove è disponibile idrogeno impuro. Il principio di funzionamento delle PAFC è il seguente: l'idrogeno o una miscela di gas ricchi di idrogeno fluisce verso l'anodo, dove gli ioni idrogeno rilasciati sono in grado di migrare attraverso l'elettrolito (che è un buon conduttore di ioni) verso il catodo; gli elettroni, invece, migrano attraverso un circuito esterno, passano nell'utilizzatore elettrico e raggiungono il catodo. In corrispondenza del catodo, dove viene fatta fluire aria, le molecole di ossigeno acquistano gli elettroni provenienti dal circuito esterno e si combinano con i protoni. La temperatura di lavoro di una PAFC deriva da un compromesso tra la conduttività



dell'elettrolito (che aumenta con la temperatura) e il tempo di vita della cella (che, invece, diminuisce all'aumentare della temperatura). Sebbene la maggior parte degli impianti installati operi a pressione atmosferica, è possibile spingersi ad 8 atm; al crescere della pressione si verifica un aumento delle prestazioni, dovuto non solo all'aumento della reversibilità del potenziale, ma anche alla riduzione della diffusione di polarizzazione del catodo e alla riduzione della perdita ohmica. Tuttavia, un sistema che lavora con idrogeno introdotto a pressione maggiore di quella atmosferica contrappone al ridotto consumo del combustibile l'aumento del costo di costruzione e di manutenzione. Attualmente le fuel-cell ad acido fosforico sono tra le più usate: dal 1970, anno dell'installazione del primo impianto, se ne contano più di 500. In Europa, l'impianto più potente è quello da 1,3 MW, realizzato a Milano nel polo universitario della Bicocca con celle della IFC (International Fuel Cells) da Ansaldo Ricerche, AEM ed ENEA. Anche il Times Square Building di New York è alimentato con fuel cell di tipo PAFC. Fra gli sviluppatori di PAFC, spicca la CERC (USA).



lo dell'autotrazione, dove la tecnologia migliore è la PEM; i risultati sono interessanti, soprattutto con le nuove membrane polimeriche, che sostituiscono il platino e quindi abbassano i costi a livelli compatibili con la produzione in serie. Impieghi di nicchia riguardano i generatori portatili di elettricità, il primo dei quali fu proposto dalla Bell Laboratories, ideatrice di una micro-cella (pochi centimetri di lato) per alimentare apparecchi telefonici portatili; esistono anche unità usate dall'esercito statunitense e costruite da ERC (Energy Research Corporation) Engelhard e Westinghouse per alimentare ricetrasmittenti militari da campo. Ultimamente è stato immesso sul mercato un generatore tascabile funzionante con la tecnica *Direct Methanol* e ci sono anche piccoli generatori nei quali l'idrogeno è stoccato in idruri metallici. Ancora, la Siemens ha realizzato un impianto di produzione per un sommergibile a propulsione elettrica, basata su fuel-cell AFC. Oltre che nei mezzi terrestri e marini, le fuel-cell possono trovare impiego anche negli aerei. Un esempio è *Antares*, decollato nel luglio 2009 dallo scalo aereo di Amburgo con un pilota a bordo. Il volo è durato solo dieci minuti ma il collaudo è stato molto positivo: si sta già lavorando per aumentare le prestazioni in fatto di velocità da circa 170 a 300 km/h. L'aereo a idrogeno è frutto della collaborazione tra il Centro Aerospaziale Tedesco (DLR) di Colonia, la Lange Aviation, la BASF fuel-cell di Francoforte, la Serenergy e il Consorzio Airbus. E non si tratta dell'unico esperimento riguardante l'applicazione delle fuel-cell all'aviazione: SkySpark, il primo aereo italiano con motore elettrico alimentato da fuel-cell, a giugno 2009 ha

battuto il record di percorrenza, volando alla velocità di 250 km/h.

IL FUTURO DELLE FUEL-CELL

Le pile a combustibile potrebbero costituire centrali elettriche collocate in vicinanza delle utenze domestiche e commerciali: in un futuro non troppo lontano sarebbe possibile eliminare le reti elettriche periferiche e limitare i tradizionali metodi di produzione dell'elettricità ai soli grandi complessi industriali, perché per case, uffici, piccole ditte e negozi, la corrente potrebbe essere prodotta localmente. Integrando centrali a fuel-cell con impianti fotovoltaici di grande dimensioni, non è da escludere che le città potranno essere alimentate solo con queste due tecnologie; certo, si porrebbe il problema di quale tensione adottare: se la continua o l'alterna-



Skyspark: l'aereo a fuel-cell.

ta. Attualmente le reti elettriche sono in alternata, però quel che funziona in alternata può andare anche in continua, anzi, in certi casi quest'ultima è meglio: ad esempio, nelle ferrovie. Inoltre, i motori elettrici ora usati negli elettrodomestici, negli utensili e negli ascensori, in tram e filobus, possono essere in continua; quanto a TV, computer ecc., i loro alimentatori funzionerebbero in modo più efficiente con la continua. D'altra parte le linee elettriche sono state realizzate in alternata per ridurre le perdite di trasferimento dalle centrali, che si trovano lontane dagli abitati e per far funzionare i motori a

gabbia, che fino a pochi anni fa erano impiegati in macchine utensili, lavatrici ed elettrodomestici in generale. La riduzione delle perdite a parità di potenza trasportata si ottiene elevando al massimo possibile la tensione delle linee, in modo da trasportare basse correnti ed avere quindi minime dissipazioni nei cavi; prima dell'avvento dell'elettronica e dei dispositivi di commutazione allo stato solido, l'unico modo per elevare una tensione era usare un trasformatore, il quale in continua non funziona. Potendo disporre di centrali sul posto del consumo, operanti già a bassa tensione, non si pone più il problema dell'elevamento e della riduzione sul posto di consumo, quindi non serve lavorare in alternata. Ogni edificio potrebbe avere l'elettricità da una pila a

combustibile, che riceverebbe il gas (idrogeno più o meno puro o gas naturale o di città) dalle tubazioni già esistenti. L'impiego di centrali di medie e grandi dimensioni potrebbe soddisfare un quartiere; considerato che le fuel-cell per questi impieghi sviluppano anche una certa quantità di calore, mediante opportuni scambiatori liquido/liquido si potrebbe convogliare il calore di raffreddamento delle celle verso le abitazioni, realizzando impianti di teleriscaldamento che farebbero risparmiare gas ed altri combustibili e l'uso di pompe di calore, almeno per l'inverno.

Quanto al settore dell'autotrazione, si può prospettare l'impiego di celle PEM, tuttavia siamo ancora lontani dalla produzione di massa di autovetture a idrogeno, e ciò a causa del costo

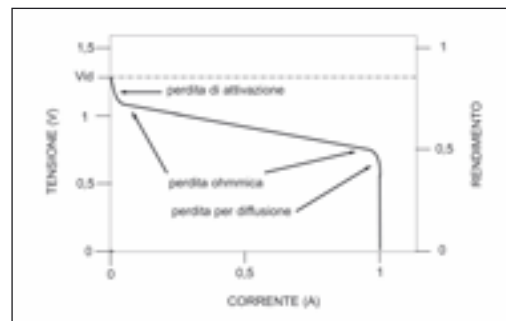


Fig. 2 - Tensione ai capi di una fuel-cell al crescere della corrente erogata.

dell'idrogeno purissimo richiesto e del catalizzatore in platino (attualmente non c'è al mondo abbastanza platino da costruirsi fuel-cell per tutti i veicoli a motore). Bisogna poi fare i conti con la lentezza di avvio della reazione che determina la produzione di elettricità: per far partire un'auto a fuel-cell bisogna aspettare alcuni minuti o utilizzare batterie che suppliscano intanto che le celle non siano in grado di fornire corrente, il che aumenterebbe il peso dei veicoli. Nelle PEM, c'è poi da affrontare il problema dell'acqua, che si forma sul catodo e che arriverebbe ad allagare la membrana impedendo il passaggio dei gas attraverso gli elettrodi e quindi il funzionamento. Certo, basterebbe lasciar scolare l'acqua, però, a parte il fatto che d'inverno nei paesi mitteleuropei e nordeuropei ciò significherebbe creare patine di ghiaccio sulle strade, l'acqua non si può scaricare del tutto perché nelle PEM serve a idratare la membrana a scambio di protoni. Peraltro ciò espone, sempre d'inverno, al rischio di congelamento, che significherebbe danneggiare la membrana. Comunque Toyota e Honda dicono di aver risolto i problemi, tanto che le vetture possono funzionare anche a -25 °.

A parte ciò, bisogna ridurre i costi di produzione dell'idrogeno e realizzare una rete di distribuzione che inizialmente, come per le pompe di benzina, avrà dei

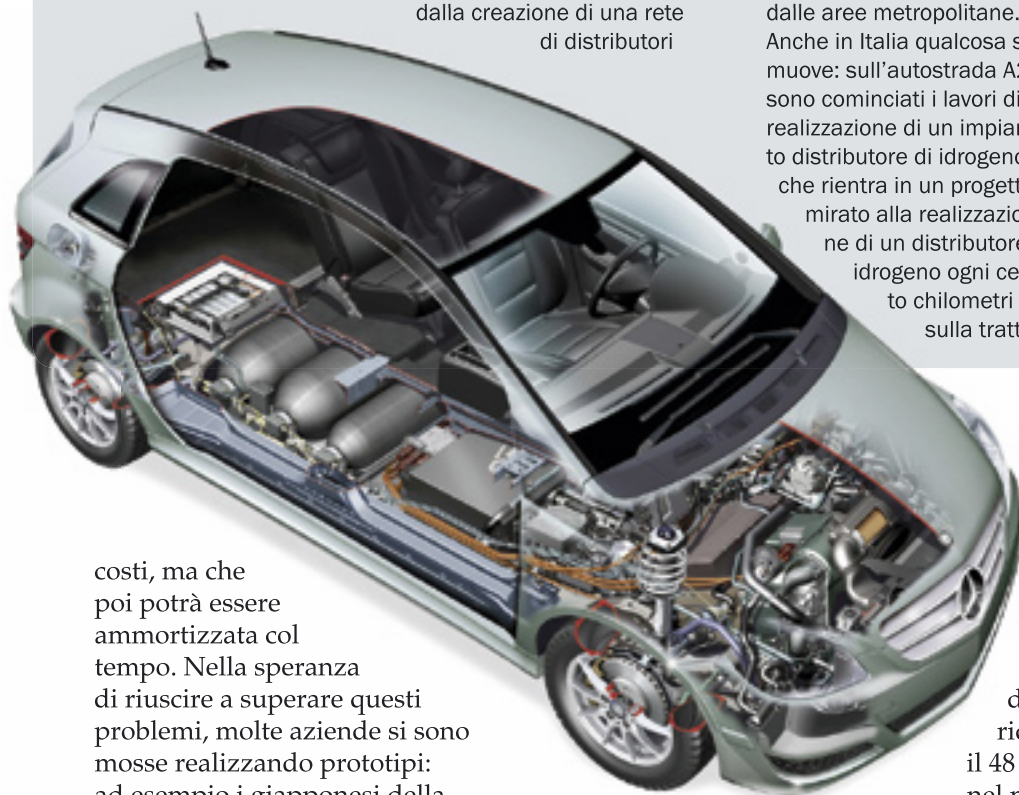
L'automobile incontra la fuel-cell

AG, Ford, General Motors, Honda, Hyundai, Kia, Renault, Nissan e Toyota, hanno sottoscritto con i maggiori produttori di piccole fuel-cell un accordo per lo sviluppo e l'introdu-

zione sul mercato, a partire dal 2015, di vetture di serie a trazione elettrica alimentata da fuel-cell. La disponibilità della clientela ad acquistare un'automobile a fuel-cell dipenderà, però, dalla creazione di una rete di distributori

di idrogeno, eventualmente abbinati a quelli di benzina e gasolio esistenti. Questa rete sarà operativa, grazie all'impegno di società petrolifere come la tedesca OMV, entro il 2015 a partire dalle aree metropolitane. Anche in Italia qualcosa si muove: sull'autostrada A22 sono cominciati i lavori di realizzazione di un impianto distributore di idrogeno, che rientra in un progetto mirato alla realizzazione di un distributore a idrogeno ogni cento chilometri sulla tratta

autostradale da Modena a Monaco (Germania). L'impianto sarà ultimato nel 2010: produrrà mediante elettrolisi ottenuta da fotovoltaico e distribuirà circa 2 milioni di mc di idrogeno all'anno, equivalenti a circa 650mila litri di benzina o 550mila litri di gasolio. Qualche casa automobilistica ha già avviato la produzione e tra le vetture apparse al recente salone di Tokyo troviamo la Honda Clarity (per ora limitata agli U.S.A.) la cui commercializzazione, causa il prezzo (200mila \$)



costi, ma che poi potrà essere ammortizzata col tempo. Nella speranza di riuscire a superare questi problemi, molte aziende si sono mosse realizzando prototipi: ad esempio i giapponesi della Fuji hanno allestito una serie di minibus elettrici per il mercato americano e in Europa sono stati e sono in sperimentazione bus a fuel-cell prodotti da Alstom, Siemens e dalla belga Elenco. La società canadese Ballard Power System, leader nel settore delle fuel-cell, iniziò a studiare e sviluppare la tecnologia mirata al settore automotive già nel 1983. Le piastre che compongono le tipiche fuel-cell Ballard sono larghe circa 20 cm, spesse meno di mezzo millimetro, e producono ciascuna circa 0,7 V. Nell'agosto 1997 l'azienda ha stipulato con Daimler-Benz una joint-venture creando la società Fuel Cell Engines, che ha portato alla realizzazione di una Mercedes Classe A elettrica e più tardi della Classe B elettrica. Già nel 1995 la cella a combustibile riusciva a produrre

poco meno di 1 kW per litro di volume (oggi 1,5 kW/l): un buon risultato, dato che per un'auto-vettura a 5 posti occorrono dai 30 a 50 kW (da 40 a 67 CV). Un significativo esempio di applicazione delle fuel-cell nel settore del trasporto pubblico lo sta dando Mercedes-Benz Buses, che di recente ha presentato Citaro FuelCELL Hybrid, il suo primo autobus ibrido basato su pile a combustibile, erede del NE-BUS, il primo autobus al mondo alimentato da uno stack di pile a combustibile da 205 kW.

PRODUZIONE DELL'IDROGENO

L'idrogeno difficilmente si trova in natura in forma ionica, ma bisogna estrarlo dai composti che lo contengono in abbondanza: ad esempio acqua, combustibili

fossili, sostanze minerali e organismi vegetali. Perché le fuel-cell diventino una realtà di tutti i giorni, occorre affinare le metodiche di produzione dell'idrogeno e incrementare i volumi prodotti, sperando che le economie di scala consentano una riduzione del prezzo. Oggi il 48 % dell'idrogeno prodotto nel mondo (500 miliardi di m³) deriva da gas naturale e frazioni leggere di petrolio attraverso lo steam reforming (trasformazione con vapore). L'idrogeno viene prodotto, immagazzinato e utilizzato prevalentemente nell'industria petrolchimica. Lo steam reforming attualmente è il principale metodo di produzione e viene realizzato in un impianto detto convertitore catalitico, dove sono introdotti vapor d'acqua alla temperatura di 800 °C e idrocarburi leggeri (metano, benzina); gli impianti di reforming (*reformer* o *riformatori*) a vapore producono anche più di 100.000 metri cubi di idrogeno all'ora. Dal processo di trasformazione si ricavano idrogeno e biossido di carbonio, nonché metano e CO; con l'impiego di vapore acqueo si realizza la reazione shift, grazie alla quale la maggior parte del

avviene con la formula leasing e la Suzuki Fuel Cell Concept (una sportiva con stack da 80 kW). In Europa, mentre BMW (che aveva prodotto la Hydrogen 7, auto a motore endotermico alimentato a idrogeno) ha abbandonato la sperimentazione, Mercedes annuncia l'avvio della produzione in serie della Classe B a fuel-cell, che sarà sul mercato nella primavera del 2010. Ma la fuel-cell non si ferma all'auto: sempre Mercedes, ha realizzato una flotta di bus (per l'uso cittadino) Citaro da 205 kW di potenza. E che dire della Suzuki? Visto che produce anche motociclette, ha pensato di realizzare anche il celebre scooter Burgman in versione fuel-cell.



Un'auto realizzata in Germania con un generatore a fuel-cell PEM a metanolo che carica le batterie con cui viene azionato il motore elettrico (www.efoy.de).

monossido viene trasformata in CO₂ (poi eliminato) e idrogeno. Nonostante sia un processo altamente ottimizzato, lo steam-reforming non risolve il problema della dipendenza dai combustibili fossili e l'idrogeno prodotto costa più del metano: 5÷8 \$ per kg di gas compresso. Un'alternativa vantaggiosa è l'integrazione del processo di produzione dell'idrogeno con la generazione di elettricità; un esempio è il Consorzio Hydrogen Park di Porto Marghera, dove già vengono prodotte circa 5.000 tonnellate/anno di idrogeno dallo scarto del cracking dell'etilene della Polimeri Europa (ENI), della Ineos e della Syndial che l'Enel intende usare per una centrale a idrogeno da 16 MW, che utilizzerà anche idrogeno estratto dal carbone. La produzione di idrogeno da fonti fossili causa il rilascio di grandi quantità di CO₂, gas notoriamente ad effetto serra, ma le pile a combustibile operano con un'efficienza 2 o 3 volte maggiore, a parità di energia prodotta, rispetto a generatori che bruciano il metano; quindi determinano emissioni inquinanti complessivamente inferiori. Al momento si mira al confinamento della CO₂ prodotta

insieme all'idrogeno mediante iniezione nel sottosuolo, nei giacimenti esauriti di metano o petrolio, sotto gli oceani o nei giacimenti acquiferi. Alcune società petrolifere sono interessate a utilizzare la CO₂ per iniettarla nei giacimenti al fine di estrarre maggiori quantità di petrolio. Un altro metodo per ricavare l'idrogeno è l'ossidazione parziale, che consiste nella trasformazione termica di idrocarburi pesanti (residui di oli pesanti dell'industria petrolchimica) con l'ausilio di ossigeno e di vapore acqueo. Il metodo dell'ossidazione parziale funziona anche con il carbone, che viene miscelato con acqua per ottenere una sospensione con un contenuto solido del 50÷70 %. Anche l'ossidazione parziale di gas naturale, petrolio o carbone produce emissioni di CO₂. L'idrogeno può essere estratto anche dal carbone minerale, mediante gassificazione, che si effettua portando il carbone alla temperatura di 900÷1.000 °C; si ricava gas di sintesi contenente idrogeno e CO₂. Tutti i gassificatori producono sostanze inquinanti (ceneri, ossidi di zolfo e ossidi di azoto) che devono essere eliminate dal gas prodotto. Recentemente è stata valutata la possibilità di estrarre l'idrogeno dalle biomasse, mediante gassificazione, pirolisi e successivo reforming della frazione liquida

prodotta, produzione di alcol etilico e successivo reforming, produzione biologica attraverso processi basati su fenomeni di fotosintesi o fermentazione. Allo scopo di abbattere i costi di produzione e l'impatto sull'ambiente delle fonti energetiche, Italia e Giappone stanno studiando, per la produzione d'idrogeno, tecniche di fotosintesi che sfruttano alghe, microrganismi e rifiuti organici. In questo campo stanno facendo molto i laboratori dall'ENI Ricerche, grazie agli studi effettuati sul batterio *Pyrococcus furiosus*. Negli U.S.A. e, precisamente alla Penn State University (Pennsylvania) è stato messo a punto un processo di fotolisi che permette l'estrazione di idrogeno utilizzando acqua, diodi in titanio e l'intero spettro della luce solare. Ma la tecnica di produzione migliore è l'elettrolisi, ovvero la scissione dell'acqua mediante l'elettricità; questo processo, noto da circa due secoli, consente di ottenere idrogeno praticamente puro, ma per quanto sia semplice da attuare risulta economicamente sconsigliato, perché richiede 4÷5 kW/h di energia elettrica per ogni m³ di idrogeno prodotto. Potrà diventare economicamente accettabile solo se sarà possibile abbassare il costo dell'elettricità, ovvero ottenere corrente elettrica gratuita (eccezion fatta per l'ammortamento degli impianti) da fotovoltaico o eolico. La BMW, a Dubai, ha dato inizio a una progetto per costruire un grande impianto di elettrolisi a energia solare per la produzione di idrogeno, da trasportare con condotti simili a quelli impiegati per il trasporto del petrolio. Un interessante sistema di produzione è anche il PLAM, che ad una temperatura di 1.600 °C consente la decomposizione di

Centrali elettriche a **fuel-cell**

Le prime centrali per la produzione di energia elettrica da celle a combustibile installate a scopo sperimentale in varie parti del mondo sono state PAFC di produzione americana. Attualmente

è operativa da più di cinque anni, a Manhattan, una centrale da 5 megawatt (quella del Time Square Building). In Giappone, patria dell'elettronica, un accordo tra Toshiba, Hitachi, Mitsubishi, Fuji Electric e Sanyo Electric ha permesso lo sviluppo di importanti progetti; ad esempio Tokyo ha una centrale da 11 MW realizzata dalla Toshiba. In Italia, la prima centrale che sfrutta celle PAFC è stata progettata e costruita dall'Ansaldo per



l'AEM (ora A2A) di Milano tra il 1995 e il 1998: produce 1,3 MW e si trova nella zona della Bicocca. Altre due piccole centrali da 25 kW cadauna, con celle a combustibile della Fuji, sono

installate all'Acoser di Bologna. Da poco si è conclusa la realizzazione, presso Ansaldo Ricerche, di uno stack di elementi MCFC da 100 kW complessivi, costituito da due moduli (con celle di area 0,75 m²). Ansaldo Fuel Cells ed Enel hanno anche firmato un accordo di collaborazione per realizzare e testare entro l'anno un sistema trigenerativo ad MCFC da 0,5 MW presso l'Area Sperimentale Enel di Livorno.

idrocarburi in carbonio puro e idrogeno; il processo non causa emissioni nocive e richiede, oltre all'idrocarburo, solo energia elettrica e acqua per il raffreddamento. Fin dall'aprile 1992, un impianto pilota produce circa 500 kg di carbonio e 2.000 m³ di idrogeno all'ora, con l'impiego di 1.000 m² di gas naturale e 2.100 kW di elettricità. Dal procedimento si sviluppa anche vapore, con una potenza di circa 1.000 kW. Tra gas, carbone e vapore, l'impianto ha un rendimento del 98 %.

Per le applicazioni veicolari si realizzano piccoli reformer a vapore o ad ossidazione parziale, destinati a ricavare l'idrogeno da benzina, gasolio o alcol etilico e metilico; ciò allo scopo di rispar-

miare sulla produzione dell'idrogeno, utilizzare le stazioni di servizio esistenti e stare al riparo dai pericoli del trasporto di idrogeno. In quest'ottica si guarda con interesse al reattore a metanolo sperimentato all'Università del Minnesota, che trasforma alcol etilico in idrogeno: grazie al ridotto tempo necessario alla catalisi (1/100 di secondo) il reattore produce grandi quantità di idrogeno per unità di tempo. Questo reformer va ancora perfezionato, perché la CO₂ che sviluppa come residuo non è tollerata dalle celle PEM. Oltre che nelle fuel-cell per autoveicoli, il reforming integrato trova impiego nelle MCFC e nelle SOFC, dove avviene direttamente sull'anodo: per effetto dell'elevata temperatura, l'acqua prodotta dalla cella viene tramutata in vapore e con il CO avviene la reazione di shift.

STOCCAGGIO DELL'IDROGENO

L'idrogeno il combustibile che presenta la maggiore densità energetica: 1 kg contiene la stessa energia di 2,1 kg di metano o di 2,8 kg di benzina. Ma in rapporto al volume la densità energetica dell'idrogeno liquido è circa 1/3 di quella del gas naturale; ciò

comporta una resa a parità di volume decisamente bassa, il che obbliga, per avere la stessa densità, a comprimerlo a pressioni maggiori di quella del GPL o del metano. Per portare l'idrogeno a bordo di veicoli occorre comprimerlo anche a più di 200 bar. Questo comporta tutta una serie di problemi, che vanno dalla corrosione dei serbatoi e delle tubature, al pericolo di esplosione. Attualmente i metodi per immagazzinare l'idrogeno sono tre: allo stato gassoso sotto pressione, allo stato liquido o assorbito da materiali solidi. L'immagazzinamento a pressione è prassi negli impianti industriali e nei veicoli; si utilizzano bombole cilindriche in acciaio che resistono fino a 200 bar. Di recente sono state sviluppate bombole leggere a struttura composita, capaci di resistere anche a più di 700 bar, con le quali si potranno raggiungere densità dell'idrogeno di 36 kg a m³. Per aumentarne la sicurezza, le bombole ad altissima pressione saranno costituite da tre strati: uno interno polimerico, uno intermedio in fibra di carbonio capace di sopportare elevate trazioni ed uno esterno in grado di proteggere dagli urti e dalla corrosione. Lo stoccaggio allo stato liquido avviene in speciali serbatoi "criotecnici" dove è possibile mantenere la temperatura compresa tra -253 e -239 °C, oltre cui l'idrogeno torna gassoso. Ciò che ha limitato questa tecnica sono i costi di liquefazione (legati alla grande energia richiesta) e l'isolamento termico che devono avere i contenitori. Attualmente le tecnologie di conservazione dell'idrogeno liquido sono ancora troppo lontane dalla produzione in serie. Uno dei più interessanti metodi di immagazzinamento dell'idrogeno è l'idrogenazione dei metalli; quando il metallo assorbe idro-



geno (idrogenazione) si raffredda, mentre per estrarre l'idrogeno (deidrogenazione) basta riscaldare l'idruro. In rapporto al volume, la capacità immagazzinabile dai serbatoi a idruri è molto elevata; purtroppo i serbatoi sono molto pesanti e costosi e le attuali tecniche non consentono la produzione su larga scala. I serbatoi ad assorbimento più moderni sono realizzati con nanotubi di carbonio, ma di recente alcuni scienziati americani dell'Università del Delaware (U.S.A.) hanno scoperto che le piume del pollame possono, se carbonizzate, produrre una sostanza in grado di immagazzinare, rispetto ai nanotubi di carbonio, a parità di massa, una quantità di idrogeno 1,7 volte maggiore. Ciò pare sia dovuto alle proprietà della cheratina di formare, se scaldata, una struttura fortemente assorbente e leggera; un serbatoio per auto fatto di questo materiale potrebbe costare meno di 200 dollari U.S.A. In alcune fuel-cell portatili viene precaricata una certa quantità di idrogeno mediante idrogenazione, usando il boroidruro di sodio (NaBH_4).

LA DISTRIBUZIONE DELL'IDROGENO

Il trasporto dell'idrogeno avviene usando le stesse tecniche collaudate per il metano e il GPL (su carri ferroviari o automezzi, ma anche su navi). È stata proposta anche la costruzione di idrogenodotti, tramite i quali, in futuro, l'idrogeno potrebbe essere fornito ad ogni edificio. Idrogenodotti esistono già in alcune regioni della Germania e, in tutto il mondo, sono in uso oltre 1.000 km di tubazioni. Proprio in Germania, il 10 settembre scorso, nell'ambito dell'iniziativa "H2 Mobility", è stato stipulato un accordo tra Linde, Daimler, EnBW, NOW, OMV, Shell, Total e Vattenfall per realizzare una rete di stazioni di servizio per le migliaia di autovetture che saranno costruite entro il 2015. Il trasporto di idrogeno liquido può anche avvenire per mare, in navi simili di quelle con le quali oggi viene trasportato il gas naturale, ma sono state progettate anche delle navi speciali; tuttavia ciò potrà avere un senso quando la domanda mondiale crescerà, perché per le quantità di idrogeno che oggi vengono trasportate via mare è più consono l'uso di contenitori speciali di dimensioni standard, che possono essere trasportati in tutto il mondo via mare. L'idrogeno è esplosivo, infiammabile ed estremamente volatile, tuttavia quando brucia si consuma molto rapidamente e, data la sua volatilità, non resta a terra ma sviluppa fiamme dirette verso l'alto, al contrario di benzina e GPL, che rimangono a terra e possono produrre danni molto consistenti. Nelle automobili, avere idrogeno nel serbatoio comporta più o meno i pericoli di quelli derivanti dalla benzina o dal GPL.

The poster is for the 31st Regional Electronics Show in Scandiano (RE). It features a blue background with a circuit-like pattern. At the top left is the logo of the Comune di Scandiano. At the top right is a logo for 'MOSTRA ELETTRONICA' with a stylized green lightning bolt. The main title '31ª MOSTRA REGIONALE ELETTRONICA SCANDIANO (RE)' is in large, bold, black letters. Below it, the dates '20/21 FEBBRAIO 2010' are in large, bold, black letters. The opening hours 'Orari: Sabato 9-18,30 Domenica 9-18' are listed. Ticket prices 'Biglietti: Intero euro 7,50 - Ridotto euro 5,00' and 'Ragazzi fino a 12 anni ingresso gratuito' are provided. Contact information 'Info: 0522/764302' and website 'www.fierasandiano.it' are included. A list of topics to be covered is shown in green boxes: telefonia, componentistica, computer, hi-fi car, radiantismo cb/om, videoregistrazione, mercatino delle pulci, and radioamatoriali. On the right side, there is a close-up image of a person's face with circuit lines overlaid.

DOPPIO LAMPEGGIATORE

A

LED

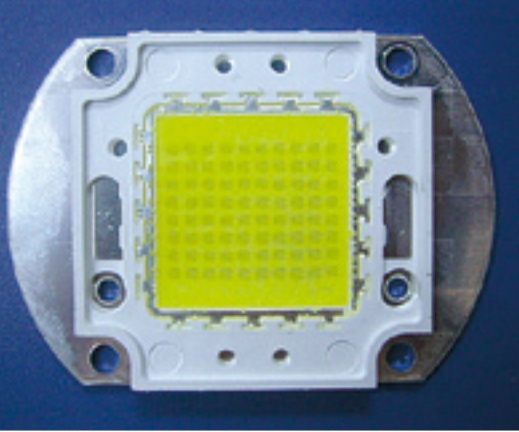
di DAVIDE SCULLINO

Utilizza due gruppi di quattro LED ad alta luminosità che lampeggiano alternativamente a una frequenza impostabile mediante un apposito trimmer; ideale per realizzare segnalatori di ostacoli sulla strada, indicatori di passaggi, lampeggiatori per veicoli di soccorso o per sistemi d'allarme, ma anche per evidenziare oggetti in vetrina.

La tecnologia dei led sta facendo passi da gigante, tanto che ormai in molti dispositivi in cui venivano impiegate le lampadine oggi possiamo trovare diodi luminosi; così anche i lampeggiatori e la segnaletica stradale sono allo stato solido, come pure i segnalatori d'allarme e tanto altro. Proprio la disponibilità di led ad alta efficienza, capaci di sostituire le tradizionali lampadine, ha ispirato la realizzazione del progetto descritto in queste pagine: un lampeggiatore a due canali in grado di comandare due gruppi di quattro led ciascuno, da collocare ad

esempio ai lati del tetto di un veicolo di soccorso, ai fianchi di una porta o alle estremità di un passaggio per segnalarne le pareti, ai lati di un carrello elevatore per renderlo più evidente agli operatori, ma anche in tante altre situazioni pratiche. Il progetto di cui parliamo è un lampeggiatore a due canali che non possiamo definire banale: infatti realizza sette diverse segnalazioni,





Efficienza luminosa e potenza dei led sembrano crescere in modo inarrestabile. Di recente è stato annunciata dalla casa cinese Youtech (<http://en.youtech-cn.com>) la realizzazione e disponibilità di led multigiunzione della potenza di ben 100 watt; questi moduli sono

Megaled da 100 watt

formati da 80 diodi (organizzati in 8 file di 10 led in parallelo) capaci di sviluppare un flusso luminoso di circa 6.000 lumen, emettendo luce bianca fredda (temperatura di colore di 6.500 °K) con un angolo di 130°, ad una tensione di alimentazione di 35÷38 volt. La durata dichiarata è di 100.000 ore. Ma non finisce qui: sembra che tra breve arriverà un nuovo modulo da 300 watt!

facilmente selezionabili da chi lo usa, mediante la pressione di un pulsante che consente di scorrere in sequenza tra le varie possibilità offerte; per ogni gioco di luce possiamo regolare manualmente la velocità di esecuzione, mediante un trimmer. Ma non solo: il circuito è stato realizzato con l'ausilio di led a luce bianca fredda ad alta efficienza, da ben 20 candele ciascuno; questo significa che ciascuno dei due gruppi è in grado di produrre un'emissione luminosa dell'intensità di 80 candele, equivalenti alla luce sviluppata da una lampadina a incandescenza per auto da una ventina di watt. La grande direzionalità che caratterizza i led ad alta efficienza permette di irradiare la luce in una precisa direzione, quindi di realizzare lampeggianti molto efficaci, che non sono certo pensati per radiomodelli ma possono trovare tranquillamente

applicazione in autoveicoli veri e propri per vari scopi.

SCHEMA ELETTRICO

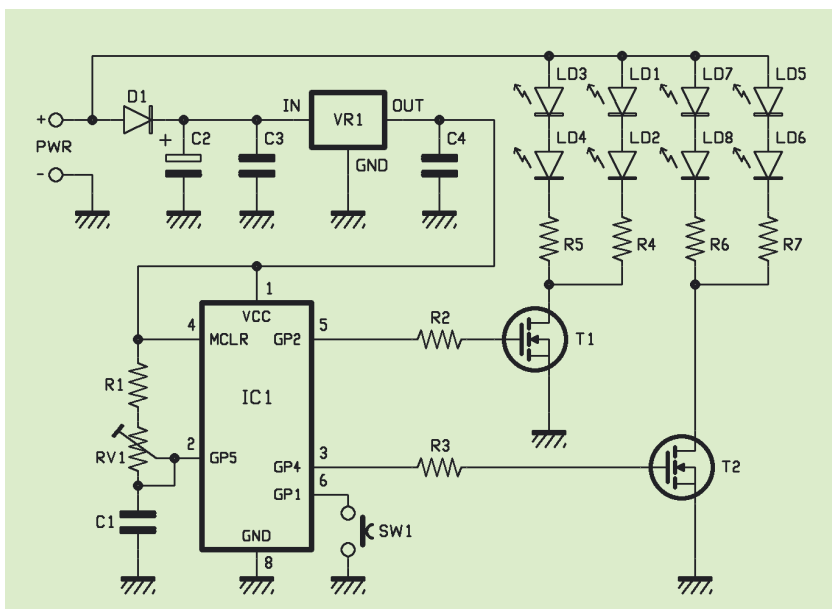
Il circuito che realizza il lampeggiatore è molto semplice, vista la funzione che deve fare; tuttavia, le possibili combinazioni luminose e la regolazione della velocità di esecuzione o frequenza di lampeggio richiederebbero una logica un po' complessa; ecco perché abbiamo affidato la gestione del lampeggiatore ad un microcontrollore: il PIC12F629 della Microchip. Il programma prevede l'esecuzione di sette giochi di luce selezionabili sequenzialmente mediante la pressione di un semplice pulsante; ogni gioco può essere eseguito a una velocità che dipende dall'impostazione del trimmer presente nel circuito. Dopo il power-on reset, il microcontrollore inizializza le proprie linee di I/O impostando GP2 e GP4 come uscite destinate al comando dei led mediante i mosfet, GP1 come ingresso con pull-up interno dedicato alla lettura del pulsante di selezione dei giochi di luce e GP5 ancora come ingresso, ma destinato alla lettura del trimmer; a riguardo va detto che l'RV1 viene letto mediante l'istruzione POT del

PICBasic, ovvero scaricando periodicamente il condensatore C1 mediante la linea GP5 (usata, in tal caso, come uscita che commuta da open-drain a zero logico) e rilasciandola, così da verificare gli intervalli di carica e scarica e risalire da essi alla resistenza inserita dal cursore del trimmer. Inizializzati gli I/O, il programma inizia a produrre gli impulsi corrispondenti al primo tipo di segnalazione, ovvero fa pulsare alternativamente le linee GP2 e GP4, generando da una un treno di impulsi a livello logico alto quando l'altra è a zero e viceversa. I sette tipi di segnalazione luminosa sono i seguenti: 1) lampeggio tremolante alternato dei due gruppi di led; 2) lampeggio tremolante alternato dei due gruppi di led, con intervallo in cui entrambi sono spenti nel passaggio dal lampeggio dell'uno a quello dell'altro; 3) lampeggio tremolante simultaneo dei due gruppi di led con pausa in cui tutti i diodi vengono spenti; 4) lampeggio simultaneo continuativo dei due gruppi di led; 5) variazione di luminosità continua di tutti e due i gruppi di led simultaneamente; 6) ciclo di due lampeggi simultanei dei led intervallati da una pausa; 7) lampeggio in dissolvenza con passaggio da un gruppo di led all'altro (dal gruppo di destra a quello di sinistra).

Nel circuito certo noterete che per alimentare i led ci siamo serviti di due mosfet di potenza BUK9535, che sono ciascuno in grado di commutare correnti di ben 36 ampere, quindi apparentemente "sprecati" per questa applicazione; in realtà questi mosfet sono stati usati pensando a possibili sviluppi del circuito e, nello specifico, per darvi la possibilità di aumentare a piacimento il numero dei led im-

CARATTERISTICHE TECNICHE

- Tensione di alimentazione: 12 Vcc
- Assorbimento: 100 mA circa
- Corrente max per uscita: 3 A
- Intensità luminosa per canale: 80 cd
- 7 effetti luminosi a velocità regolabile



piegati laddove voleste realizzare lampeggianti di grande potenza luminosa. Nell'attuale configurazione, mostrata dallo schema elettrico, ogni gruppo ottico consta di due serie di led ognuna polarizzata mediante una propria resistenza limitatrice di corrente (R4 per la serie LD1/LD2, R5 per LD3/LD4, R6 per LD7/LD8 ed R7 per LD5/LD6) poste tra loro in parallelo; ma nulla vi impedisce di connettere in parallelo anche dieci o venti serie di led (ovviamente ognuna con una sua resistenza). Infatti ammettendo di polarizzare ciascun led con una corrente di 25 milliampere, ognuno dei mosfet, anche senza dissipatore, potrebbe tranquillamente alimentare un centinaio di serie e quindi 200 led, prelevando dal drain di ciascun mosfet una corrente non superiore ai 2,5 ampere! Inoltre, l'uso dei mosfet permette a chi volesse realizzare un lampeggiatore a lampadine comuni, di farlo senza problemi: infatti al posto dei diodi luminosi e delle rispettive resistenze R4, R5, R6, R7, potete collegare delle lampadine da 12 volt 21 o 55 watt, senza paura di sovraccaricare i mosfet. Comunque, in linea di massima il circuito è stato progettato per

fornire in uscita mediamente 3 ampere a canale e ciò non tanto per limiti dei mosfet, quanto perché lo stampato l'abbiamo disegnato con piste calcolate per far scorrere tale corrente; chi volesse realizzare applicazioni in cui serve qualcosa di più, dovrà semplicemente aumentare lo spessore delle piste, ad esempio stagnando abbondantemente la superficie di queste ultime, cosa che permette di aumentare cospicuamente la sezione utile. L'intero nostro circuito va alimentato a 12 volt in continua e richiede una corrente che dipende da quanti led si usano; nell'attuale configurazione, ad esempio, assorbe non più di 65 milliampere. Il regolatore integrato VR1 che vedete contornato dai condensatori C2, C3 e C4 (che servono da filtro per l'alimentazione, in ingresso i primi due e in uscita il terzo...) serve esclusivamente a stabilizzare a 5 volt la tensione che alimenta il microcontrollore PIC12F629. La tensione che alimenta i circuiti di uscita, quindi led e mosfet, viene presa direttamente dalla morsettiere PWR; quella che va al regolatore integrato passa, invece, dal diodo di protezione D1, che serve ad evitare di danneggiare il

78L05 laddove per errore applicaste l'alimentazione a + e - PWR con la polarità invertita.

Notate che per le uscite non serve alcun diodo in quanto i led possono tranquillamente sopportare tensioni inverse di 12 volt e i mosfet T1 e T2 sono provvisti di un diodo di protezione interno con l'anodo sul source e il catodo sul drain.

REALIZZAZIONE PRATICA

Bene, spiegato come funziona il doppio lampeggiatore possiamo passare a descrivere come realizzarlo; la prima nota riguarda il circuito stampato, che è stato disegnato prevedendo la possibilità di staccare le due parti laterali in modo da disporle, eventualmente, in dei proiettori o altri involucri per realizzare lampeggianti. La possibilità di dividere la basetta permette, inoltre, di collocare i led lateralmente, in modo da realizzare lampeggiatori in cui le luci si accendono sui lati opposti

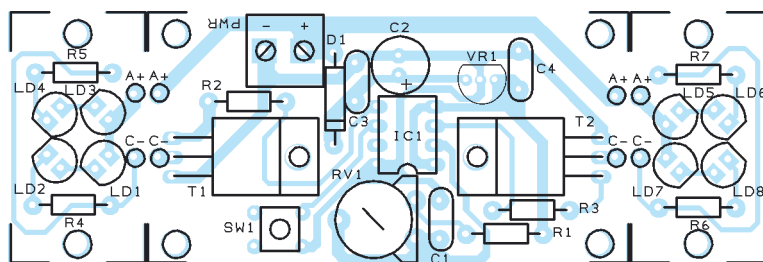


per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono facilmente reperibili in commercio. Di questo circuito è anche disponibile la scatola di montaggio di produzione Velleman distribuita nel nostro paese da Futura Elettronica. Il kit (cod. MK180) costa 17,00 Euro IVA compresa e comprende tutti i componenti, il micro già programmato, la basetta forata e serigrafata e gli otto LED ad alta luminosità.



Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287
<http://www.futurashop.it>



Elenco Componenti:

R1: 33 kohm	D1: 1N4007
R2÷R7: 330 ohm	LD1÷LD8: Led 5 mm bianco
RV1: Trimmer 22 kohm MO	T1,T2: BUK9535-55
C1: 100 pF ceramico	SW1: Microswitch
C2: 100 µF 25 VL elettrolitico	Varie:
C3, C4: 100 nF multistrato	- Morsettiere 2 poli
IC1: PIC12F629-I/P (VMK180)	- Zoccolo 4+4
VR1: 78L05	- Circuito stampato

del corpo. Per queste ragioni le piste che portano dai mosfet e dall'alimentazione positiva ai gruppi di led sono interrotte e vanno unite, durante il montaggio dei componenti, con due ponticelli se si intende mantenere un unico circuito stampato, ovvero con fili di adeguata lunghezza, se si desidera scorporare i gruppi di quattro led dal circuito di controllo.

Procediamo con ordine: la prima

cosa da fare per costruire il lampeggiatore è preparare lo stampato, tutto intero, la cui traccia lato rame potete scaricare dal nostro sito Internet (www.elettronica.in) in modo da ottenere (mediante una stampa laser su carta da lucido o acetato) la pellicola per procedere con la fotoincisione. L'alternativa alla realizzazione in proprio dello stampato è l'acquisto del kit (www.futurashop.it). A questo punto dovete decidere

se intendete tenere il circuito tutto insieme o separare i led; in quest'ultimo caso tagliate le estremità tra le piazzole di interconnessione (A+ e C-). In ogni caso, iniziate a disporre i componenti dalle resistenze e dal diodo D1 (la cui fascetta colorata, che indica il catodo, deve essere rivolta all'interno della basetta) quindi proseguite con lo zoccolo (4+4 piedini) del microcontrollore, il pulsante e i mosfet, che vanno disposti sdraiati dopo averne piegato ad angolo retto i tre terminali. Andate avanti sistemando i condensatori (prestando l'attenzione alla polarità degli elettrolitici) il trimmer, la morsettiere di alimentazione e i led, rammentando per questi ultimi di disporre il lato smussato come indicato nel disegno di montaggio. In ultimo, con degli spezzoni di terminali tagliati da resistenze o condensatori realizzate i ponticelli di connessione tra il corpo centrale e le parti laterali: ne servono due per ogni lato, in modo da collegare drain dei mosfet e positivo di alimentazione. Se avete separato gli stampati, collegate le piazzole A+ e C- alle rispettive delle basette dei led con della piattina bipolare. Volendo utilizzare lampadine a 12 volt o gruppi più numerosi di led, potete connetterli sempre con della piattina bipolare, ai punti A+ e C-.

Una volta ultimato il circuito, dovete inserire nello zoccolo il microcontrollore già programmato, mantenendolo con la tacca di riferimento rivolta al trimmer; potete quindi procedere al collaudo, collegando alla morsettiere un alimentatore in grado di fornire 12 Vcc ed una corrente pari a quella assorbita dai led o dalle lampadine, più una quindicina di milliampere per il microcontrollore.

Guarda la potenza del lampeggiante a LED su YouTube!



Vuoi vedere in funzione il nostro lampeggiante a LED? Scoprire la potenza dei 2 gruppi di LED? Nulla di più semplice: collegati alla pagina <http://www.youtube.com/watch?v=VWrhXUnzIKI>. Potrai anche vedere i video di moltissimi altri nostri progetti proposti in passato.

Ricevitore basato sull'RCM120 con due uscite digitali e due analogiche a TRIAC. Ad ognuno dei quattro canali di uscita si possono abbinare fino ad un massimo di 6 diversi pulsanti di un trasmettitore PTM200, rendendo possibili diversi scenari di utilizzo della scheda.

RICEVITORE ENOCEAN 2 CANALI ON/OFF e 2 USCITE TRIAC

..... di MATTEO DESTRO

Non è trascorso molto dall'ultima volta che abbiamo parlato di dispositivi basati sui moduli Enocean, però vogliamo proporre ancora un progetto con essi, dato che li riteniamo sicuramente interessanti. I prodotti Enocean sono sistemi wireless per radiocomando le cui unità trasmettenti, che possono essere basate su pulsanti o pensate per leggere segnali analogici, non necessitano di alimentazione; ciò è importante in special modo per i pulsanti, perché ci permette di posizionarli a

piacere per accendere una lampada o comandare porte e tapparelle motorizzate, senza curarci di tirare alcun filo, realizzare tracce o scendere a compromessi con vincoli riguardanti l'impianto elettrico dei locali. L'unica accortezza è, trattandosi di moduli RF, evitare di posizionarli dietro o contro piastre di metallo. Venendo al progetto descritto in queste pagine, si tratta di un sistema di controllo

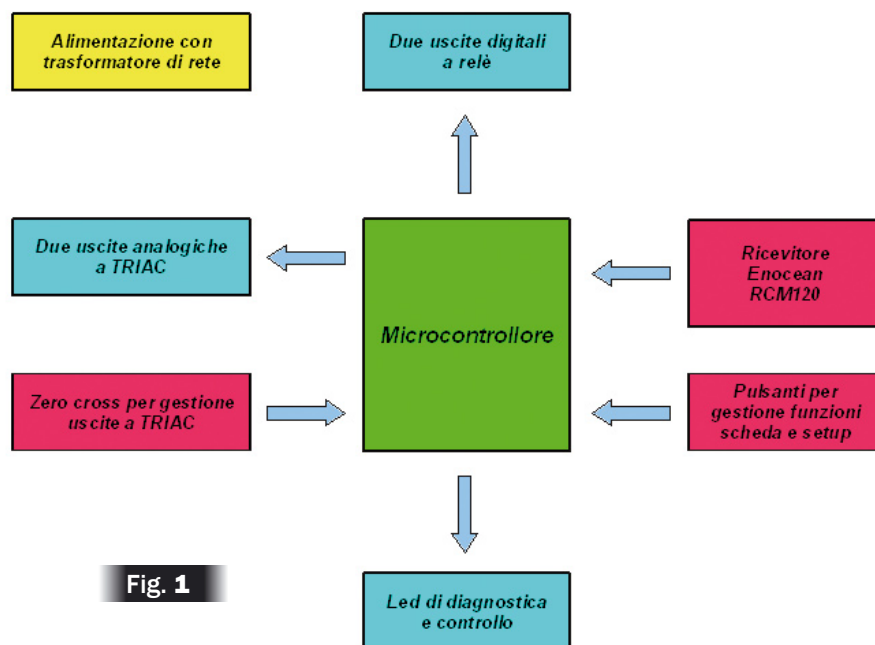


Fig. 1

per due uscite digitali, ON/OFF, e due uscite analogiche dimmerabili a TRIAC, utili per la regolazione dell'intensità luminosa di lampade che consentono la regolazione della loro intensità luminosa. Ciò che possiamo dimmerare sono le lampadine ad incandescenza e quelle ad alogeni, oppure i trasformatori elettronici ai quali sono collegate lampade dimmerabili a bassa tensione.

L'HARDWARE

Iniziamo la descrizione dell'hardware, partendo dallo schema a blocchi (Fig. 1) dove si possono identificare quattro aree distinte per colore. Il cuore del sistema è il microcontrollore PIC18F2620 che si occupa di gestire tutte le funzioni della scheda (colore verde); seguono le sezioni di

ingresso (colore rosso) nelle quali possiamo identificare:

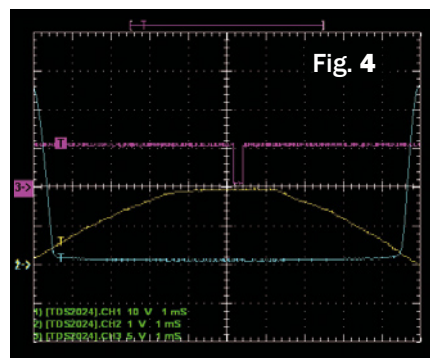
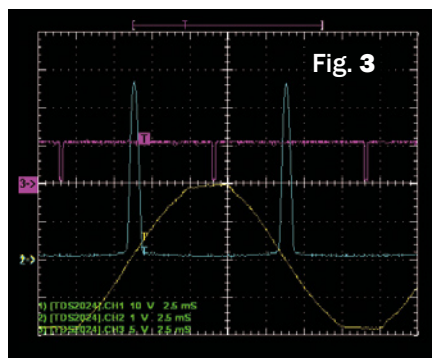
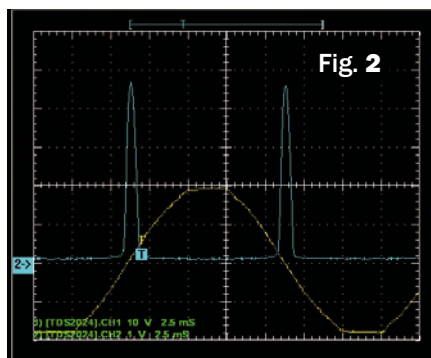
- la gestione dei due pulsanti di ingresso, necessari per gestire le funzioni di configurazione della scheda;
- la gestione del ricevitore RCM120 attraverso la connessione seriale;
- la gestione dello zero-crossing; per ognuna delle due uscite a TRIAC è prevista una sezione di gestione dello zero-crossing.

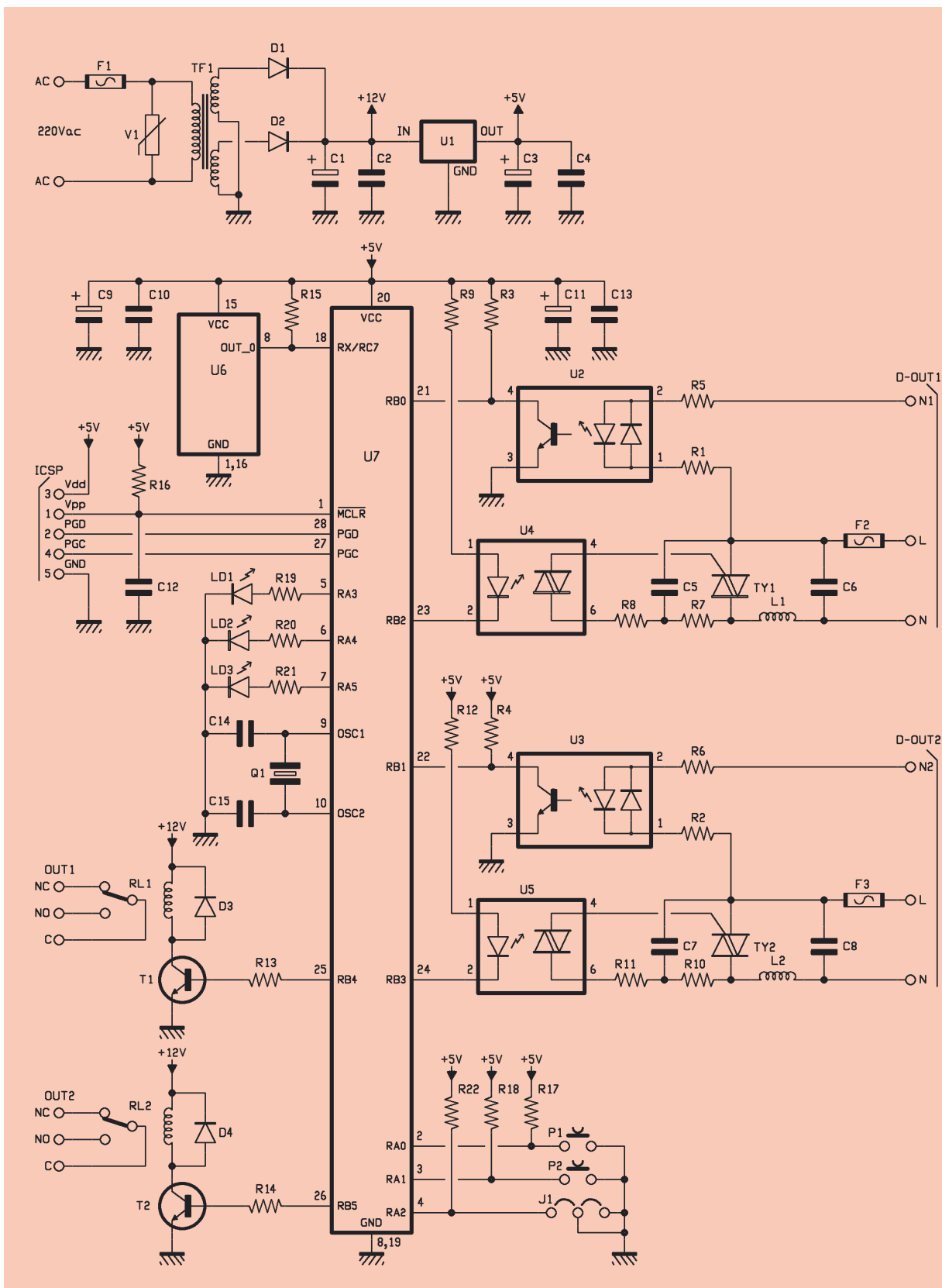
Poi abbiamo le sezioni di uscita (di colore azzurro); in questo caso si possono distinguere:

- gestione delle due uscite digitali a relé;
- gestione delle due uscite analogiche a TRIAC;
- gestione dei led di diagnostica e setup.

Infine, rimane la sezione di alimentazione (colore giallo).

Dallo schema a blocchi nasce lo schema elettrico che trovate nella pagina accanto. La sezione di alimentazione è stata realizzata grazie a un trasformatore con due secondari, dal quale otteniamo, mediante un raddrizzatore a doppia semionda, +12 V per alimentare le bobine dei relé e, grazie al regolatore di tensione LM7805, si ottiene la tensione stabilizzata +5 V. Sul primario del trasformatore è stata predisposta una protezione formata dal termofusibile F1 e dal varistore V1; arrivi una sovratensione, il varistore cortocircuiterà la linea facendo intervenire il termofusibile, aprendo così il circuito. Come di consueto, la sezione di ricezione dei telegrammi trasmessi dai TX PTM200 viene gestita dal modulo RCM120, il quale, grazie alla connessione seriale, trasferisce i dati al PIC18F2620, che si occuperà di interpretarli e di gestirne la parte che richiede l'eventuale comando delle uscite digitali e analogiche. La trasmissione seriale tra il PIC e il modulo RCM120 deve essere configurata per avere una velocità di trasmissione di 9.600 bit/s, un bit di start, un bit di stop e nessun bit di parità. Fondamentale è la resistenza di pull-up da 4,7 kohm (R15) in quanto l'uscita OUT_0, del modulo RCM120 è di tipo open-collector.

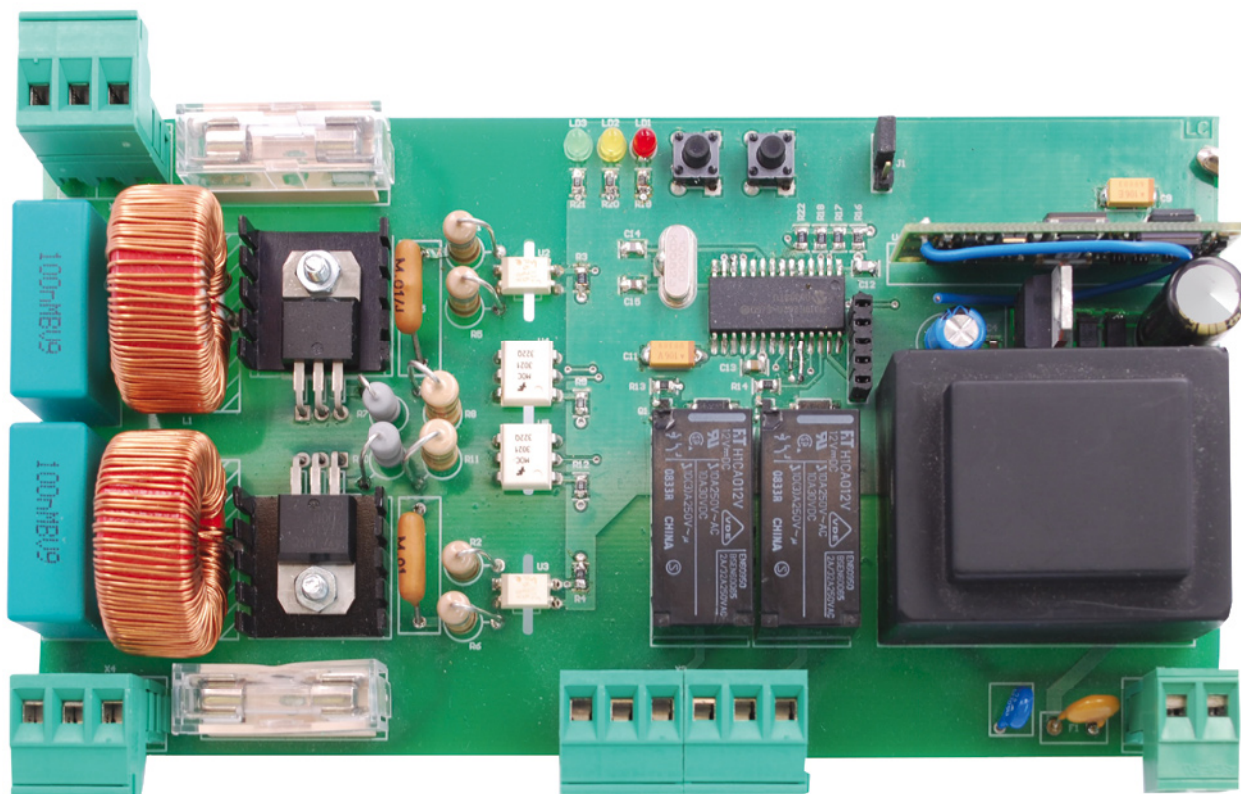




La scheda usa due pulsanti, P1 e P2, necessari durante le fasi di configurazione; il primo, chiama-

to LRN, serve per entrare in modalità di acquisizione dei moduli trasmettitori PTM200, mentre il

secondo, detto CLR, serve per resettare la memoria EEPROM nella quale sono memorizzati i



Elenco Componenti:

R1: shunt 0,001 ohm 1 W
 R2: 68 kohm 1W
 R3, R4: 2,2 kohm (0805)
 R5, R6: 68 kohm 1W
 R7: 56 ohm 1W
 R8, R11: 330 ohm 1W
 R9, R12: 330 ohm (0805)
 R10: 56 ohm 1W
 R13, R14: 1,2 kohm (0805)
 R15: 4,7 kohm (0805)

R16: 22 kohm (0805)
 R17, R18: 4,7 kohm (0805)
 R19, R20: 470 ohm (0805)
 R21: 470 ohm (0805)
 R22: 4,7 kohm (0805)
 RR1: 68 kohm 1W
 C1: 680 μ F 25 VL elettrolitico
 C2: 100 nF multistrato (0805)
 C3: 100 μ F 25 VL elettrolitico
 C4: 100 nF multistrato (0805)

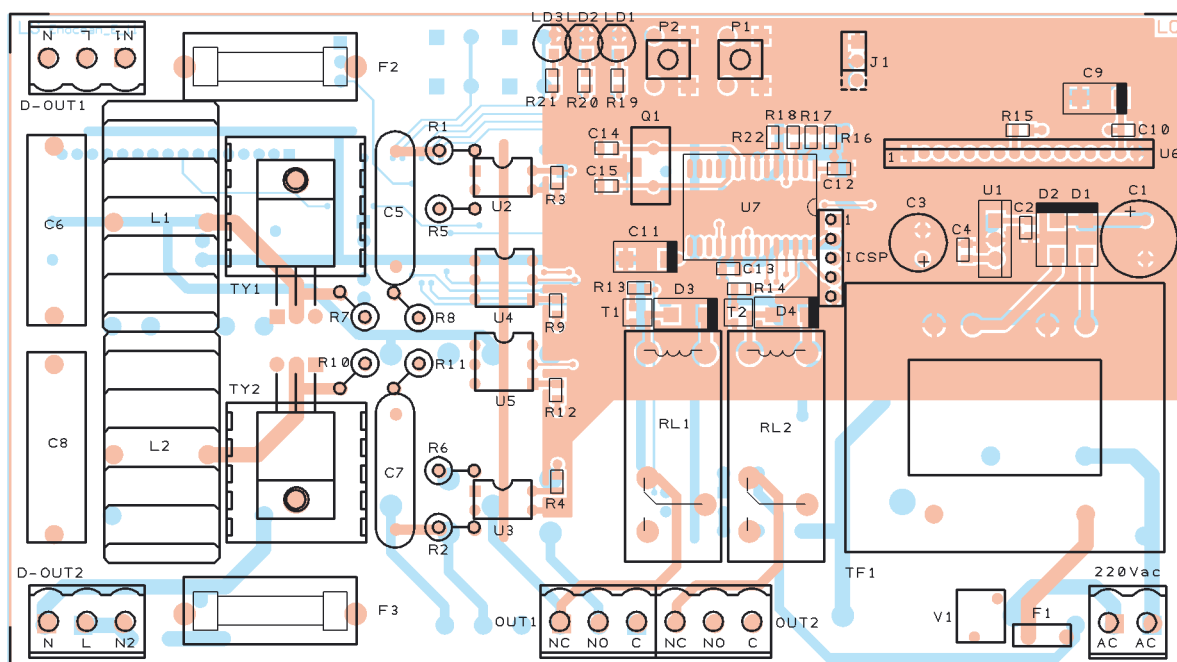
C5: 10 nF 275 V poliestere
 C6: 100 nF 300 V poliestere
 C7: 10 nF 275 V poliestere
 C8: 100 nF 300 V poliestere
 C9: 10 μ F 25 VL tantalio (C)
 C10, C13: 100 nF multistrato (0805)
 C11: 10 μ F 25 VL tantalio (C)
 C12: 220 pF ceramico (0805)
 C14, C15: 22 pF ceramico (0805)
 D1, D2, D3, D4: GF1M

moduli trasmettitori PTM200. Il jumper J1 serve per selezionare le due possibili modalità di gestione dell'innesco dei due TRIAC. Se il jumper non è inserito si ottiene una gestione dell'innesco idonea a pilotare carichi puramente resistivi (ad esempio lampade alogene) ma inadatto a pilotare dei carichi induttivi (ad esempio il motore di un ventilatore). In questo caso la larghezza dell'impulso di gate è pari a 200 μ s. Nel caso in cui il jumper sia inserito (RA2 a massa) la gestione del gate del TRIAC viene

fatta con un impulso a larghezza variabile, che permette di gestire in modo corretto carichi sia resistivi che induttivi. In questo caso la larghezza dell'impulso varia da un massimo di 8 ms a un minimo di 200 μ s. La larghezza dell'impulso viene determinata dal firmware e dipende dalla percentuale di regolazione del carico e dal passaggio per lo zero della tensione di rete.

I tre led LD1, LD2 e LD3 servono durante la procedura di acquisizione dei moduli trasmettitori PTM200; ne parleremo più

approfonditamente durante la descrizione della procedura di acquisizione dei trasmettitori. Le due uscite digitali vengono realizzate semplicemente con due relé (RL1 e RL2) i quali sono gestiti dal microcontrollore attraverso due transistor T1 e T2. Le uscite digitali sono connesse ai pin RB4 e RB5 del PIC18F2620. La sezione a TRIAC è un pochino più complicata; per la descrizione faremo riferimento al canale a TRIAC CH0, iniziando col dire che quando si realizza un dimmer a TRIAC è necessario in-



U1: 7805
U2: TLP626
U3: TLP626
U4, U5: MOC3023
U6: RCM120
U7: PIC18F2620-E/SO (MF851)
Q1: Quarzo 10 MHz
LD1: Led 3 mm rosso
LD2: Led 3 mm giallo
LD3: Led 3 mm verde

T1, T2: BC846B
TY1, TY2: BTA16-600B
RL1: Relé 12 V 1 scambio
RL2: Relé 12V 1 scambio
P1, P2: Microswitch
TF1: Trasformatore 2x9Vac/3VA
F1: Termofusibile 80 mA
F2, F3: Fusibile 1,5A Rapido
L1: Bobina di filtro Bourne 100 μ H
L2: Bobina di filtro Bourne 100 μ H

V1: Varistore S05/230V

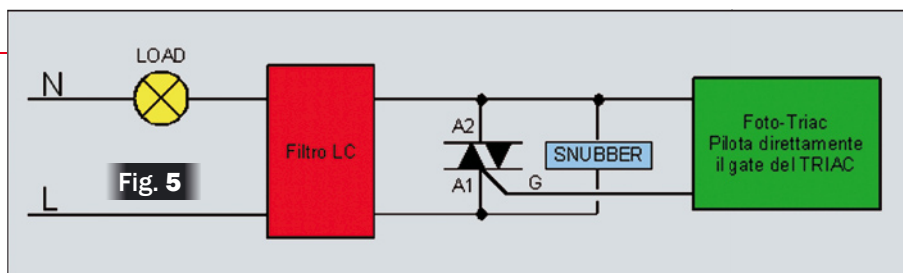
Varie:

- Strip femmina 5 poli
- Strip maschio 3 poli
- Jumper
- Portafusibile da c.s. (2 pz.)
- Morsettiera componibile 2 poli
- Morsettiera componibile 3 poli (4 pz.)
- Circuito stampato

tercettare il passaggio per lo zero della tensione di rete (il cosiddetto Zero-Crossing). Un circuito semplice per realizzare uno ZCD (Zero Crossing Detector) può essere realizzato utilizzando una resistenza e un condensatore, ossia una rete RC. Però in questo modo non si ha un isolamento galvanico tra la tensione di rete e la bassa tensione presente sulla nostra scheda. Per ottenere, invece, l'isolamento galvanico, si può utilizzare un circuito realizzato con un fotoaccoppiatore (integrato U2, TLP626). L'int-

grato presenta, dal lato dell'alta tensione, due led collegati in antiparallelo; riesce così ad attivarsi e rilevare il passaggio per lo zero sia nella semionda positiva, sia nella semionda negativa. Sul lato di bassa tensione, il componente ha un fototransistor in configurazione open-collector, al quale va collegata una resistenza di pull-up da 2,2 kohm. Per capire meglio è necessario guardare la Fig. 2, dove il segnale in giallo è la tensione di rete (230 Vca) mentre quello azzurro è il segnale presente sul collettore

del foto-transistor. Come si può osservare, tutte le volte che la tensione di rete è maggiore della tensione di innesco dei led (VF), sia per la semionda positiva che negativa, il fototransistor si trova nello stato di ON, di conseguenza sul pin RB0 del PIC18F2620 ritroveremo il valore logico "0". Quando invece i due LED non sono in conduzione, cioè quando la tensione di rete è minore della tensione VF, sull'uscita del fototransistor si ritrova il valore logico "1", come evidenziato dalla Fig. 2. L'impulso che si



vede, come detto prima, viene riportato sull'ingresso RB0 del PIC il quale è configurato come interrupt esterno (INT0). L'interrupt è attivo sul fronte di discesa dell'impulso. Tutte le volte che si genera l'interrupt, il microcontrollore carica un timer, scaduto il quale è possibile inviare l'impulso di accensione sul gate del TRIAC. Vedremo approfonditamente tutto ciò nel seguito di questo articolo. Si noti che la tensione di rete non viene portata direttamente sui pin 1 e 2 dell'integrato U2: entrambi sono preceduti da una resistenza da 68 kohm, altrimenti il componente si guasterebbe subito. In ogni semionda le resistenze sono in serie e determinano la necessaria caduta, prendendo su di loro la tensione che i diodi non riuscirebbero a sopportare. Alla circuitazione di gestione dello zero crossing va aggiunta l'elettronica di gestione dell'innescamento del TRIAC, la quale, per semplicità, viene riassunta con lo schema a blocchi di Fig. 5. Come si può osservare, il carico viene collegato a monte del filtro LC sul filo di neutro; questo filtro è formato da un'induttanza più un condensatore e serve per filtrare i disturbi che potrebbero essere captati dalla linea elettrica quando si utilizza un TRIAC per dimmerare un carico. Passato il filtro, il filo del neutro arriva sul pin A2 del TRIAC, mentre sul pin A1 arriva il filo della fase. Quando il TRIAC è acceso, la corrente fluisce dalla fase al neutro e quindi nel carico. Collegata in parallelo al TRIAC c'è una rete RC che prende il nome di *snubber*. Per ultimo

rimane il fototriac necessario per separare galvanicamente la bassa tensione dall'alta tensione; in questo modo il PIC18F2620 non pilota direttamente il gate del TRIAC, ma lo fa mediante un fotoaccoppiatore provvisto di TRIAC in uscita, che a sua volta interviene sul gate del TRIAC del circuito. Quest'ultimo è un BTA16BW (TY1) ed il fototriac è un MOC3023 (U4). La rete snubber in parallelo al TRIAC, formata dalla resistenza R7 e dal condensatore C5, serve perché quando il TRIAC passa dallo stato ON allo stato OFF, potrebbe autoinnescarsi malgrado il suo gate non sia più polarizzato. L'autoinnesco è dovuto al fatto che, quando la corrente passa per lo zero, la tensione viene riapplicata istantaneamente ai capi del TRIAC, il quale, siccome richiede un piccolo intervallo per spegnersi completamente, all'inversione di polarità resta in conduzione. Un autoinnesco può essere generato:

- dalla pendenza della corrente di/dt ; questo parametro determina la quantità di carica che rimane, quando la corrente passa per lo zero, e potrebbe essere iniettata nella area di gate del TRIAC;
- dalla pendenza della tensione dv/dt ; questo parametro definisce la corrente capacitiva la quale potrebbe essere iniettata attraverso il gate del TRIAC.

Lo snubber ha quindi lo scopo di limitare la velocità di variazione della tensione applicata al TRIAC; in altre parole, migliora il tempo di risposta del TRIAC e nel contempo diminuisce

l'influenza della variazione di tensione. Per dimensionare lo snubber si può procedere in questo modo: si determina, dal data-sheet MOC3023, la massima corrente ITSM, che vale 1A. Si determina quindi la resistenza R8 con la seguente formula:

$$R8 = \frac{230V \cdot \sqrt{2}}{1A} = 330$$

A questo punto si determina la tensione in ingresso al TRIAC, V_{inT} , la quale dipende dai seguenti parametri: V_{TM} , V_{GT} e I_{GT} , i quali sono tutti ricavabili dal datasheet del TRIAC utilizzato (BTA16BW). Quindi si può scrivere:

$$V_{inT} = R8 \cdot I_{GT} + V_{TM} + V_{GT}$$

La I_{GT} vale 50 mA, V_{TM} è pari a 1,55 V e $V_{GT} = 1,3$ V. Pertanto la V_{inT} vale:

$$V_{inT} = 330\Omega \cdot 50mA + 1,55V + 1,3V = 19,35V$$

Supponendo che la massima tensione di turn off che può presentarsi sull'uscita sia di 600V si può affermare che:

$$\frac{dV}{dt} = \frac{V_{Tmax}}{\tau} = \frac{600V}{(R7 \cdot C5)}$$

Da cui si può ricavare:

$$R7 \cdot C5 = \frac{V_{Tmax}}{\frac{dV}{dt}}$$

La tensione dv/dt , ricavata dal datasheet del TRIAC, è pari a 1000 V/ μ s e quindi:

$$R7 \cdot C5 = 600 \cdot e^{-9}s$$

Assumendo che la corrente I_{GT} del TRIAC sia di 50 mA, valore sempre ricavato dal data-sheet, possiamo scrivere che:

$$R7 + R8 = \frac{V_{inT}}{I_{GT}} = \frac{19,35V}{50mA} = 387\Omega$$

Di conseguenza R7 vale:

$$R7 = 387 \, \Omega - 330 \, \Omega = 57 \, \Omega$$

Per concludere, il valore del condensatore C5 sarà:

$$C5 = \frac{\tau}{R2} = \frac{600 \, e^{-9}s}{57} = 10,5 \, nF$$

Riassumendo, per entrambi i canali analogici, R8 ed R11 valgono $330 \, \Omega$, R7 e R10 sono da $56 \, \Omega$, C5 e C7 valgono $10nF$. Ad ognuno dei due TRIAC è necessario associare un dissipatore per smaltire il calore generato durante il funzionamento a regime del dispositivo.

Ora va detto che nei dispositivi a semiconduttore il passaggio della corrente elettrica provoca, per effetto joule, un innalzamento della temperatura. Tale temperatura può essere calcolata utilizzando la seguente formula:

$$Tw - Ta = Pd \cdot Rth$$

Dove Tw è la temperatura raggiunta dal dispositivo a causa della dissipazione della potenza applicata Pd, Ta è la temperatura ambiente ed Rth è la resistenza termica, che rappresenta la difficoltà di smaltimento del calore verso l'ambiente circostante. L'unità di misura per la resistenza termica è il $^{\circ}C/W$.

Nei datasheet del costruttore viene sempre indicata la temperatura massima di funzionamento del semiconduttore (Tmax) ed è quindi possibile calcolare la potenza massima dissipabile Pmax ad una data temperatura ambiente Tamax. A questo punto la formula matematica sopra esposta diventa:

$$Tmax - Tamax = Pmax \cdot Rth$$

Oltre alla temperatura massima di funzionamento ed alla resistenza termica tra semiconduttore e contenitore (o, sovente,



Arrivano le Smart Grid

Fornire elettricità non significa solo produrla, ma anche distribuirla attraverso una rete, fino ad arrivare all'utente finale. Comporta inoltre la capacità di gestire la produzione di elettricità di migliaia di piccoli generatori e convogliarla nelle aree dove c'è maggior richiesta. L'attuale rete di distribuzione è nata per trasportare e distribuire l'elettricità dalla centrale all'utente finale e in un futuro molto prossimo potrebbe avere non pochi problemi a gestire l'elettricità prodotta dai privati, che viene immessa in rete in modo irregolare e dai punti a bassa tensione. Gestire la generazione distribuita in BT e ottimizzare la distribuzione dell'elettricità prodotta dai vari soggetti privati sono i principali obiettivi delle Smart Grid, le reti elettriche intelligenti, gestite da potenti computer connessi via Internet che organizzeranno l'attività dei punti di distribuzione dell'elettricità nostrana e di quella importata. In U.S.A. è già partito il progetto Power Grid 2.0, che vedrà impiegati 40 milioni di contatori intelligenti e 3.000 chilometri di linee di trasmissione, per realizzare una Smart Grid in grado di dare una svolta alla rete elettrica, trasformandola in una moderna rete che utilizza microprocessori e software per lavorare in modo efficiente e attingere a fonti rinnovabili per la distribuzione di energia. L'idea di adottare questa tecnologia è nata proprio studiando ciò che accade negli U.S.A. e valutando i cambiamenti anche nel nostro sistema di distribuzione a seguito all'entrata di numerosi gestori privati che hanno portato all'immissione di molta elettricità ma senza il necessario coordinamento per dirigerla correttamente dove realmente serve.

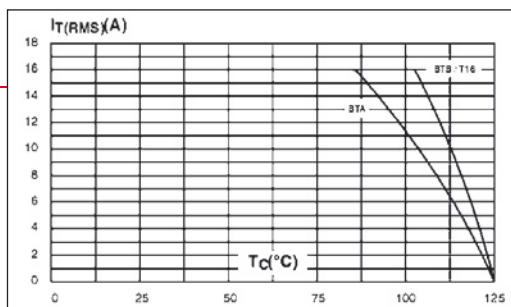


Fig. 6

che caratterizza l'efficienza di un dissipatore e che riassume tutti i parametri sopra elencati: prende il nome di resistenza termica tra dissipatore e ambiente (R_{thda}); tale valore viene sempre specificato dal costruttore del dissipatore. L'unità di misura è ancora il $^{\circ}\text{C}/\text{W}$. A questo punto possiamo riformulare la formula matematica esposta prima trasformandola nella seguente:

$$T_{jmax}-T_{amax}=P_{dmax}\cdot(R_{thjc}+R_{thcd}+R_{thda})$$

Dove T_{jmax} è la massima temperatura di giunzione del semiconduttore, T_{amax} è la massima temperatura ambiente raggiungibile, P_{dmax} è la potenza che si vuole dissipare, R_{thjc} è la resistenza termica tra giunzione e case del semiconduttore, R_{thcd} è la resistenza termica fra contenitore e dissipatore e R_{thda} è la resistenza termica dissipatore ambiente specificata dal costruttore del dissipatore. Quanto appena detto può essere riassunto con la Fig. 8.

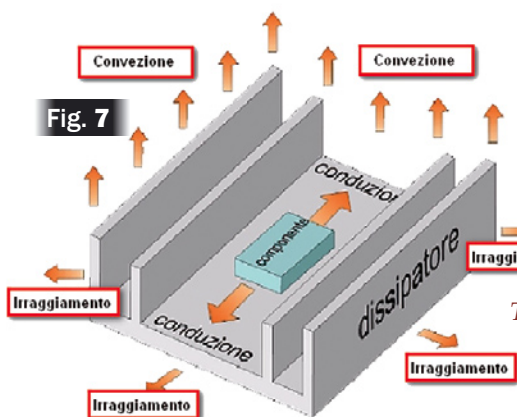
A questo punto possiamo calcolare la resistenza termica che deve avere il dissipatore per soddisfare le esigenze della nostra applicazione. Si stabilisce che ogni TRIAC debba gestire un carico massimo di 300 W, e che quindi la corrente I_T RMS sarà pari a 1,3 A. Dal grafico nella Fig. 9 si deduce la potenza che deve dissipare il TRIAC, la quale si attesta a circa 1,2 W (P_{dmax}). La resistenza termica R_{thjc} , ricavata dal data-sheet, è pari a $1,2^{\circ}\text{C}/\text{W}$, mentre la resistenza termica R_{thcd} è anch'essa pari a $1,2^{\circ}\text{C}/\text{W}$. La massima temperatura di giunzione è 125°C , ma per stare larghi imponiamo che non debba superare i 100°C . Impostiamo, infine, la temperatura ambiente T_{amax} a 50°C . Possiamo quindi riscrivere la formula come segue:

$$R_{thda} = \frac{T_{jmax}-T_{amax}}{P_{dmax}} - R_{thjc} - R_{thcd}$$

Sostituendo i valori sopra elencati abbiamo il valore della resistenza termica del dissipatore:

$$R_{thda} = \frac{100^{\circ}\text{C}-50^{\circ}\text{C}}{1,2\text{W}} - 1,2^{\circ}\text{C}/\text{W} - 1,2^{\circ}\text{C}/\text{W} = 39,2^{\circ}\text{C}/\text{W}$$

Quindi per dissipare il calore generato dal TRIAC durante il suo funzionamento è necessario scegliere un dissipatore che abbia una resistenza termica R_{thda} pari a $39^{\circ}\text{C}/\text{W}$. Ad esempio, il dissipatore 8201 della Pada soddisfa a questa condizione (la Fig. 10 ne mostra le specifiche). Nella nostra applicazione abbiamo utilizzato quello con altezza di 13 mm (8201/19/ST H13). Completiamo il discorso sul circuito di potenza dicendo che sono stati previsti due fusibili per ogni canale analogico, che nello schema appaiono siglati F2 e F3. I fusibili usati tollerano una corrente massima di 1,5 A e servono



tra semiconduttore e ambiente) nei data-sheet viene riportato un grafico nel quale si mette in relazione la potenza dissipata dal dispositivo con la temperatura massima raggiungibile dal case; tale curva si chiama "curva di derating". La Fig. 6 mostra la curva riguardante i TRIAC BTA16BW. Come si può osservare, fino ad una temperatura di 85°C è possibile dissipare la massima potenza resa disponibile dal TRIAC, mentre oltre gli 85°C è necessario abbassare la potenza e quindi la corrente commutabile. Ad esempio a 100°C la massima corrente gestibile si attesta a 11A, contro i 16 A alla massima potenza. Con queste informazioni è possibile scegliere un dissipatore adeguato alla propria applicazione. Si ricorda che un dissipatore può smaltire il calore per conduzione, convezione e irraggiamento, come mostra la Fig. 7. Va inoltre detto che l'efficienza di un dissipatore dipende da parametri quali il materiale utilizzato, le sue dimensioni, la forma, il colore, la posizione sulla scheda, le condizioni di ventilazione e infine la posizione del semiconduttore sul dissipatore stesso. Si può definire un parametro

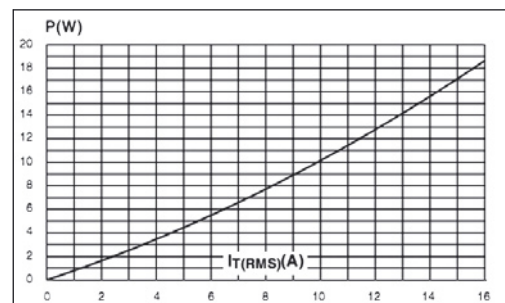
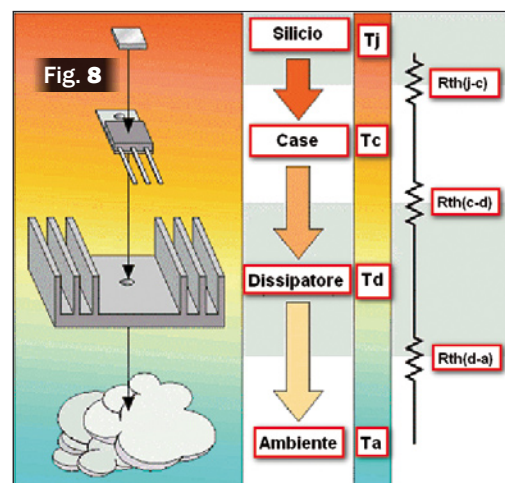


Fig. 9

Cod. d'ord.	Rth(°/w)	Lungh.(mm)	Cod. Descr.	Cod. Forat.
Order Code	Rth(°/w)	Length(mm)	Descr. Code	Hole Code
00003575	30	19	8201/19/ST H10	T0220
00003580	29	19	8201/19/ST H13	T0220

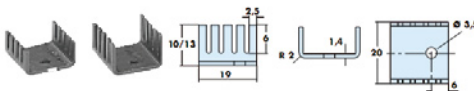


Fig. 10

per proteggere i TRIAC in caso di sovraccarico o corto circuito sull'utilizzatore.

DESCRIZIONE FIRMWARE:

Il cuore del sistema è il microcontrollore PIC18F2620, il quale gestisce tutte le funzioni messe a disposizione dalla scheda. Il firmware è stato scritto in linguaggio C, che agevola la realizzazione delle routine più complesse. La Fig. 11 mette in evidenza l'impostazione data al progetto, il quale suddivide le varie sezioni del programma in diversi file con estensione ".c" e i relativi file ".h". I file più significativi sono:

- rcm120_management.c
- output_management.c

Il primo si occupa di gestire i pacchetti dati inviati dal modulo RCM120. Ricordiamo che il modulo ricevitore RCM120 dialoga con il microcontrollore grazie all'interfaccia seriale di cui dispone; però del PIC viene utilizzato soltanto il pin RX, dato che la comunicazione è unidirezionale dall'RCM120 al microcontrollore. La gestione dell'interfaccia seriale prevede un buffer che permette di memorizzare fino a tre pacchetti dati, consecutivi, permettendo così di gestire più trasmettitori PTM200 che inviano i propri dati contemporaneamente (la funzionalità del buffer è già stata trattata nei precedenti articoli sui progetti Enocean). Nel file "rcm120_management.c" trovano spazio anche le routine per la memorizzazione degli ID dei moduli PTM200, durante la fase di acquisizione, e le rispettive routine per cancellare gli ID dalla memoria EEPROM. Il secondo file, "output_management.c" contiene tutte le routine di gestione delle uscite, sia digitali che analogiche, le quali dipendo-

no dagli ID dei moduli PTM200 memorizzati nella EEPROM del PIC18F2620.

Le uscite sono indipendenti le une dalle altre e ciò significa che è possibile pilotarne più d'una contemporaneamente e da trasmettitori PTM200 diversi.

La logica di funzionamento del firmware può essere generalizzata con il flow-chart disegnato nella Fig. 12, dove sono evidenziate due sezioni distinte: la prima si occupa della inizializzazione del microcontrollore (appare in azzurro) e la seconda della gestione di tutte le funzionalità della scheda. Questa seconda sezione prende il nome di "main" (è evidenziata in giallo).

Durante l'inizializzazione vengono configurate tutte le periferiche del PIC18F2620 (*Init. CPU*) e tutta la RAM, la quale viene configurata con valori noti (*Init. RAM*). Si ricorda che i microcontrollori allo startup non hanno la memoria RAM inizializzata, quindi è buona norma fissare ogni locazione di memoria a un valore noto. Per ultimo vengono inizializzate le macchine a stati che gestiscono le varie funzioni della scheda. La seconda sezione gestisce le funzioni messe a disposizione della scheda, in particolare:

- pulsanti e Jumper collegati agli ingressi della porta A (RA0, RA1 e RA2 configurati come ingressi digitali);
- led di diagnostica e controllo anch'essi collegati alla porta A (RA3, RA4 e RA5 configurati come uscite digitali);
- interfaccia seriale per gestione modulo RCM120.

Di quest'ultima va detto che l'intercettazione dei byte inviati dal modulo da parte del micro viene gestita sfruttando gli interrupt:

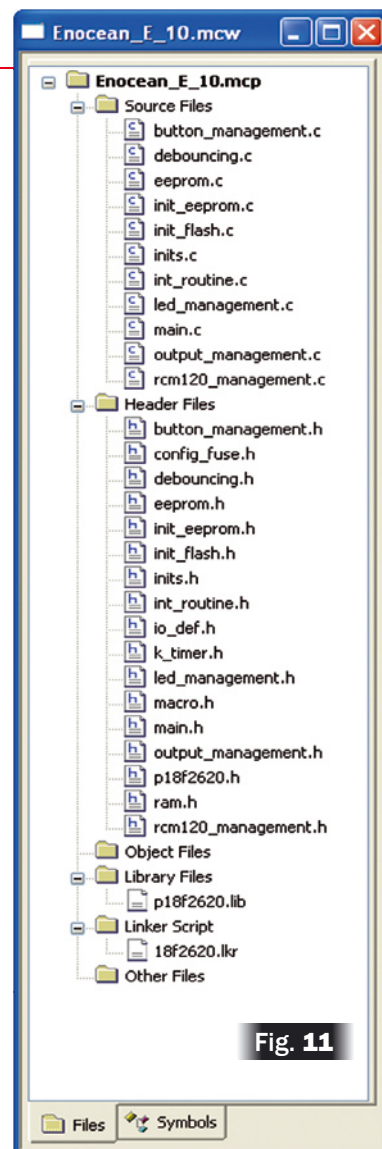


Fig. 11

la periferica ha un registro di ricezione, chiamato RSR, il quale riceve il dato comprensivo del bit di start e stop. Non appena viene ricevuto il bit di stop, il dato a 8 bit viene estrapolato e inviato al registro RCREG, generando un interrupt (Bit RCIF). A questo punto il codice che gestisce l'interrupt della periferica UART legge il registro RCREG e lo memorizza nel buffer di ricezione a scorrimento. Ad ogni byte ricevuto il puntatore al buffer viene incrementato di una unità, puntando così alla locazione successiva. Il buffer è stato dimensionato per memorizzare fino a 50 byte, permettendo così di gestire eventuali situazioni di congestione. Il firmware testa di continuo il buffer ed è quindi

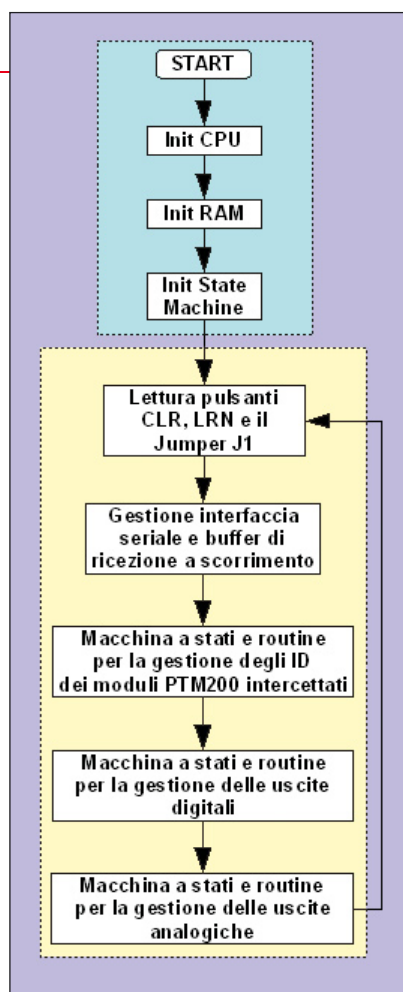


Fig. 12

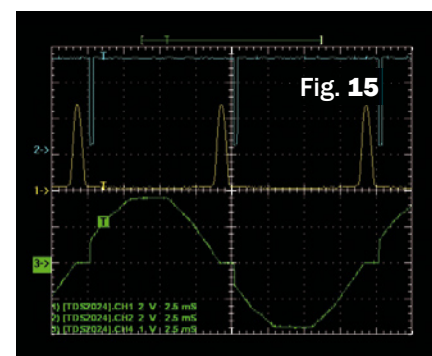
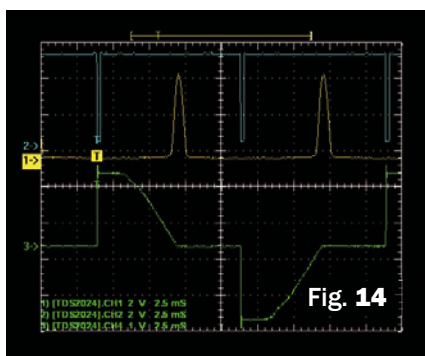
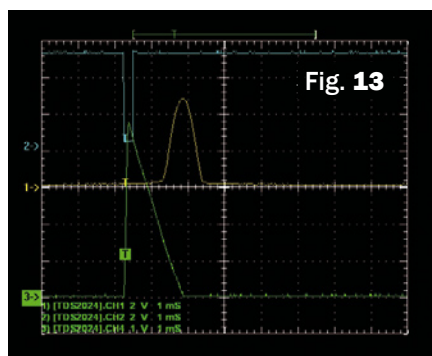
buffer non sarà totalmente svuotato dai dati ricevuti. Se i dati presenti nel buffer sono validi, verrà analizzato l'ID del modulo trasmesso e il relativo pulsante premuto, quindi il codice andrà a gestire le uscite corrispondenti. A riguardo va detto che nella gestione delle uscite sono coinvolti:

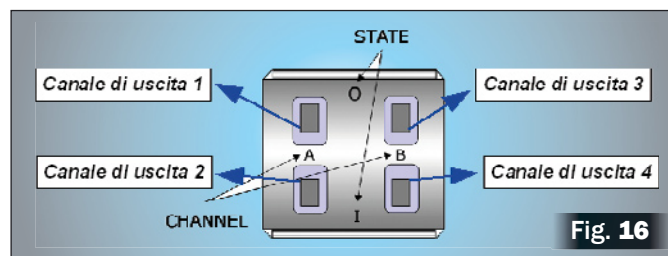
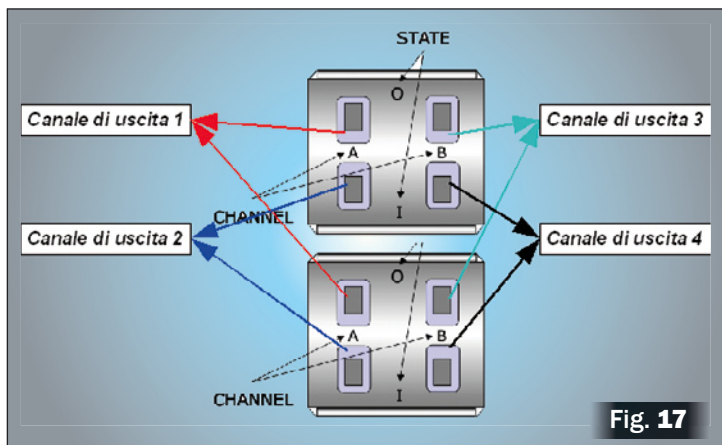
- la sezione di controllo canali di uscita a relé e a TRIAC (viene utilizzata la porta B e in particolare, di essa i pin RB2, RB3, RB4 e RB5 sono configurati come uscite digitali);
- la sezione di ingresso per il rilevamento dello zero-crossing (sono utilizzati i pin RB0 e RB1 della porta B); gli ingressi corrispondenti sono configurati come interrupt esterni.

Analizziamo ora il concetto che sta dietro alla gestione delle due uscite a TRIAC, fermo restando che la gestione dell'impulso di trigger del TRIAC è inteso quando il jumper non è inserito; il concetto esposto vale anche se il jumper è inserito, ma l'impulso generato avrà una durata diversa (come detto in precedenza). Per la corretta gestione delle uscite è necessario utilizzare le due periferiche hardware interne al PIC18F2620 denominate CCP1 module e CCP2 module, entrambe in configurazione comparatore (Compare). Il termine CCP significa Capture/Compare/PWM e nella configurazione *compare* la

periferica CCP si presenta come nella **Fig. 19**. Ad ognuna delle due periferiche CCP viene associato un contatore a 16 bit: per la periferica CCP1 viene usato il timer 1, mentre alla periferica CCP2 è associato il timer 3. Come detto prima, durante la descrizione dell'hardware, ad ogni intercettazione del passaggio per lo zero della tensione di rete viene generato un segnale di zero-crossing, il quale viene letto dal microcontrollore attraverso i due pin di ingresso RB0 (Canale analogico CH0) e RB1 (Canale analogico CH1) impostati come interrupt esterni attivi sul fronte di discesa del segnale in ingresso. Ragioniamo, ora, sul canale analogico CH0: ogni volta che viene intercettato lo zero-crossing viene caricata una costante di tempo nei registri CCPR1H e CCPR1L della periferica CCP1 ed è azzerato il timer associato a 16 bit TMR1 (vedere **Fig. 19**). Quando il valore raggiunto dal TMR1 uguaglia la costante di tempo caricata nel registro CCPR1, viene generato un interrupt, il quale viene intercettato per generare l'impulso di gate del TRIAC. La durata dell'impulso di gate si attesta sui 200 µs (con jumper inserito la durata dell'impulso varia da un minimo di 200µs a un massimo di 8ms). Le figure 13,14 e 15 mettono in relazione l'impulso di gate, associato al TRIAC del canale CH0,

in grado di accorgersi quando un pacchetto è stato ricevuto in modo completo; a questo punto inizia l'elaborazione dei dati ricevuti, in particolare del checksum, dei byte di SYNC e del byte di ORG. La nostra applicazione gestisce solo i moduli trasmettitori PTM200, quindi il byte di ORG deve sempre essere pari a 0x05. Se i risultati dei test sono negativi, il sistema sposterà verso l'alto il contenuto del buffer e ritenterà il test finché l'esito non sarà positivo, oppure fino a quando il





con il segnale di zero-crossing e la relativa corrente di TRIAC. In particolare, risultano in giallo il segnale di zero-crossing, in azzurro l'impulso dato al gate del TRIAC e in verde la corrente commutata dal TRIAC stesso. La Fig. 15 mostra la gestione del TRIAC, con luminosità al 100 %, mentre la Fig. 13 evidenzia lo spunto di corrente all'accensione. La sonda di corrente utilizzata per tracciare il grafico è impostata per fornire 100 mV per ogni ampere, quindi si deduce che la corrente di picco è 18 A, almeno per quel che riguarda i tracciati nelle figure 14 e 15. Nella Fig. 13 si nota che allo spunto la corrente raggiunge 48 A. Il carico applicato al TRIAC è 300 W.

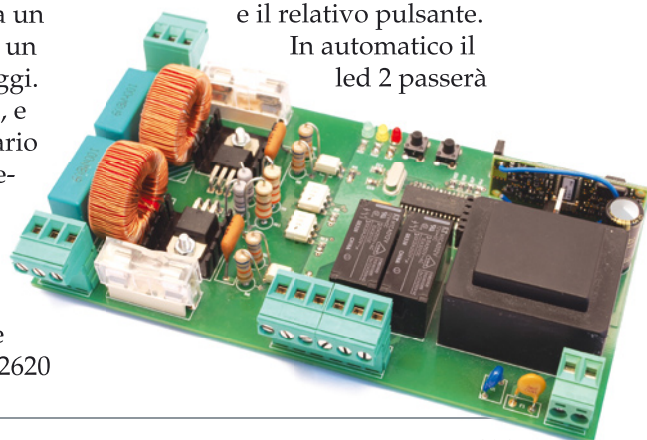
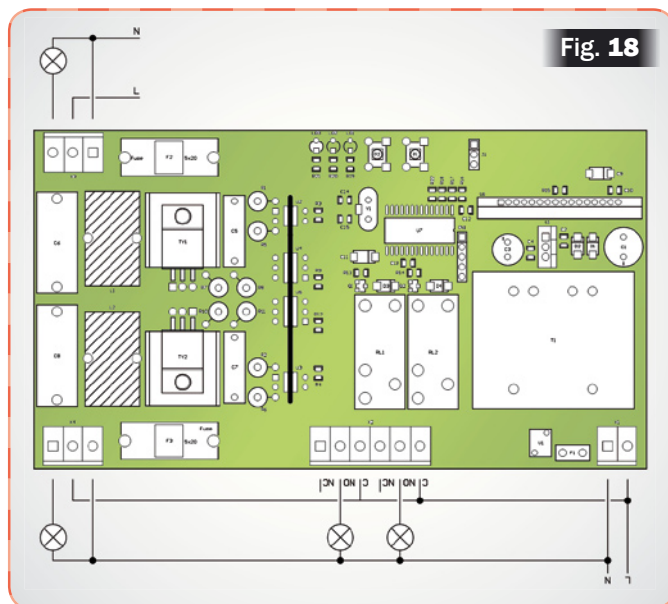
ACQUISIZIONE ID DEI MODULI PTM200

I due pulsanti LRN (P1) e CLR (P2) di cui la scheda dispone, permettono di richiamare le varie funzioni necessarie alle impostazioni di base; vediamo di seguito come utilizzarli. Premendo il pulsante CLR per più di cinque secondi vengono cancellati tutti gli ID memorizzati nella EEPROM del PIC18F2620. Il led 1 rimarrà acceso per tre secondi a indicare il buon esito dell'operazione. Premendo il pulsante LRN per più di cinque secondi verranno cancellati tutti gli ID come per il punto precedente, in più il sistema entrerà in modalità

di acquisizione. Durante la cancellazione degli ID, il led 1 rimarrà acceso per tre secondi. Passati i tre secondi il led 1 verrà spento mentre il led 2 e il led 3 inizieranno a lampeggiare con una cadenza ben precisa a indicare le varie fasi di acquisizione degli ID da associare alle quattro uscite della scheda (due uscite digitali e due uscite analogiche). Il led 2 indica il numero di ID che si sta memorizzando per il canale di uscita n-esimo. Siccome per ogni canale è possibile memorizzare fino a sei ID differenti si noterà che il led 2 emetterà da un minimo di un lampeggio a un massimo di sei lampeggi indicando così a quale numero di ID si è arrivati durante l'acquisizione. Il led 3 invece indica su quale canale si sta agendo e quindi farà da un minimo di un lampeggio a un massimo di quattro lampeggi. Per acquisire un nuovo ID, e relativo pulsante, è necessario premere per tre volte consecutive il pulsante desiderato del modulo PTM200. Se questa condizione viene verificata il sistema provvederà a memorizzare nella EEPROM del PIC18F2620

il codice ID del modulo PTM200 e il codice del pulsante stesso. Supponiamo di volere acquisire il primo ID da associare al canale 1 di uscita. Il led 2 eseguirà un lampeggio ogni quattro secondi evidenziando che si sta acquisendo il primo ID, mentre il led 3 eseguirà anch'esso un lampeggio ogni quattro secondi evidenziando che stiamo agendo sul canale di uscita 1. Premendo per tre volte consecutive sul pulsante del modulo trasmettitore verrà acquisito l'ID dello stesso e il relativo pulsante.

In automatico il led 2 passerà



da uno a due lampeggi ogni quattro secondi, indicando che il sistema è pronto ad acquisire il secondo ID per il canale di uscita 1. Questa procedura continuerà fino all'acquisizione del sesto ID. Una volta acquisito il sesto ID il sistema passerà in automatico al secondo canale di uscita. Di conseguenza il led 2 farà nuovamente un lampeggio ogni quattro secondi mentre il led 3 farà due lampeggi ogni quattro secondi evidenziando che si è passati al secondo canale di uscita. Ovviamente non si è obbligati ad acquisire per forza sei ID per ogni canale. Quando si vuole passare da un canale di uscita al successivo è sufficiente premere il pulsante LRN (P1). Se per un tempo superiore a cinque minuti il PIC18F2620 non riceve più ID dalla seriale, provvederà in automatico ad uscire dalla modalità di acquisizione. La Fig. 16 mostra una possibile configurazione che si può assegnare

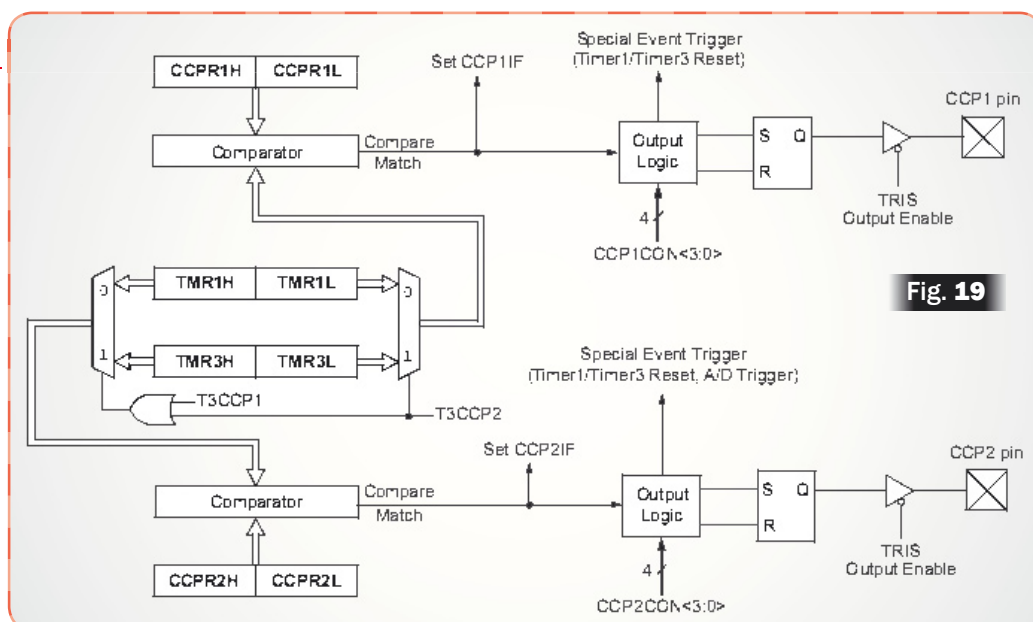


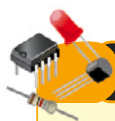
Fig. 19

alle uscite utilizzando un solo trasmettitore PTM200: ad ogni pulsante è stato assegnato un pulsante del TX. Per le uscite digitali della scheda, ogni pulsante ad esse associato funziona da toggle, cioè con ad ogni pressione inverte lo stato dell'uscita. Quanto alle uscite analogiche, ognuna viene associata ad un pulsante, premendo brevemente il quale si accende, ovvero si porta l'uscita stessa all'intensità luminosa precedentemente impostata prima dello spegnimento. Una nuova breve pressione del pulsante spegne l'uscita, salvando in memoria l'ultimo valore di luminosità raggiunto. Quando l'uscita analogica è attiva, una pressione continuativa sul pulsante ad essa associato farà aumentare o diminuire l'intensità luminosa della stessa, passando da un minimo al massimo e viceversa. La Fig. 17 mostra, invece, come associare più pulsanti per ogni canale, ricordiamo che per ogni canale si possono al massimo associare sei pulsanti differenti.

REALIZZAZIONE PRATICA

Per costruire l'unità qui descritta, una volta in possesso dello stampato (ottenibile per fotoincisione dalle tracce rame scaricabili dal nostro sito www.elettronica.in) vi conviene iniziare il montaggio

dai componenti SMD, in particolare dalle resistenze e dai condensatori in formato 0805. Proseguite con i diodi, i due transistor e il microcontrollore PIC18F2620. Successivamente montate i fotoaccoppiatori, il quarzo, i led e i pulsanti, compreso il jumper. A questo punto passate alla parte di potenza, quindi TRIAC con dissipatore, resistenze di potenza, condensatori e induttanze del filtro, portafusibili, relé e morsettiere. Infine montate la sezione di alimentazione e il trasformatore. Controllate di avere montato tutto in modo corretto, che non ci siano cortocircuiti tra le piste o tra i pin del microcontrollore; se tutto è ok potete alimentare la scheda e verificare che tutte le alimentazioni siano corrette. Per l'uso, riferitevi alla Fig. 18, che mostra come collegare alla scheda i relativi carichi. Ovviamente non è necessario collegare tutti carichi, ma basta connettere solo quelli necessari. L'uscita digitale CH0 viene riportata sulla morsettieria X2 (pin 1, 2 e 3) mentre l'uscita digitale CH1 viene riportata sempre sulla morsettieria X2 ma sui pin 4, 5 e 6. Per quanto riguarda le uscite analogiche, il canale CH0 viene riportato sulla morsettieria X3 mentre l'uscita analogica CH1 viene riportata sulla morsettieria denominata X4.



per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono facilmente reperibili in commercio. Il master del circuito stampato ed il firmware con cui è programmato il micro possono essere scaricati gratuitamente dal sito della rivista (www.elettronica.in). Il modulo RCM120 utilizzato in questo progetto costa 51,00 Euro, IVA compresa.

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287
<http://www.futurashop.it>

Corso Wireless CPU



I comandi AT

In questa puntata conosciamo ed impariamo ad usare i comandi AT speciali delle wireless CPU di Wavecom, utili per realizzare svariate funzioni.

dell'Ing.
NICOLA FRISO

Nelle puntate precedenti abbiamo preso familiarità con le wireless CPU serie WMP di Wavecom, abbiamo visto come configurare l'ambiente di sviluppo M2M Studio per la programmazione delle WCPU, ed abbiamo anche scritto la nostra prima applicazione utilizzando le librerie API "Open-AT" di Wavecom. L'applicazione mandava un SMS di allarme e faceva

lampeggiare un led in corrispondenza della pressione di un tasto; l'abbiamo potuta testare fisicamente grazie al modulo "Augusto" e alla demoboard EVKAUGUSTO di Shitek Technology, descritti nelle scorse puntate. Prima di addentrarci ulteriormente nello studio delle librerie API di Open AT è necessario conoscere almeno le basi del funzionamento dei comandi AT.

I comandi AT rappresentano il protocollo di comunicazione standard dei moduli GSM, ma anche di qualsiasi altro modem. Esistono dei comandi AT standard comuni a tutti i modem GSM (secondo la specifica ETSI GSM) ma anche dei comandi AT non standard "aggiuntivi" introdotti dal produttore del modulo GSM; in questo caso Wavecom. La grande novità introdotta da Open AT (da qui l'origine del nome) risiede nella possibilità, offerta al programmatore, di definire e creare nuovi comandi AT personalizzati; vedremo come fare nel corso di questa puntata. Nel seguito di questo articolo sono riportati i principali comandi AT di uso più comune. I comandi AT in generale hanno tutti una formattazione standard che è riportata nella

Tabella 1 - Formato dei comandi AT.

Comando	Effetto
AT+<cmd>=?	Visualizza i parametri ammessi per quel comando
AT+<cmd>?	Visualizza le impostazioni attuali per quel comando
AT+<cmd>	Esegue il comando AT esteso senza parametri
AT+<cmd>=<parameters>	Esegue il comando AT esteso con parametri
AT<cmd>	Esegue il comando AT senza parametri

Tabella 1. In tutti i comandi <cmd> è l'identificativo del comando e <parameters> la lista dei parametri previsti (separati dalla virgola

se è presente più di un parametro). Ovviamente i comandi AT sono molti di più di quelli riportati in questa puntata del corso ma elencarli tutti esula dagli scopi di questo articolo; chi volesse approfondire l'argomento potrà trovare tutti i comandi standard, ad esempio, nel manuale di cui al punto [1] della Bibliografia. Per poterli provare è sufficiente collegare la porta seriale del PC con la UART1 del modulo Augusto che è presente sulla demoboard EVK AUGUSTO. In alternativa è necessario costruirsi un'interfaccia RS232 come quella riportata nel data-sheet del modulo Augusto prodotto dalla Shitek Technology (www.shitek.eu). Nell'apposita sezione di questa puntata sono riportati anche alcuni comandi AT introdotti da Wavecom, che avevamo già descritto nella prima puntata del corso, comandi che sono utili per il download e l'avvio della propria applicazione Open AT all'interno del modulo.

Vi invitiamo a prendere familiarità con i comandi AT in quanto molto spesso occorrerà richiamarli all'interno delle nostre applicazioni Open AT. Le librerie ADL introdotte nella scorsa puntata infatti, permettono di fare molte cose ma non tutto, e spesso è necessario utilizzare proprio

Listato 1 - Richiamo del comando AT+CREG all'interno di un'applicazione.

```
void adl_main( adl_InitType_e InitType )
{
    adl_atCmdCreate("AT+CREG?",FALSE,(adl_atRspHandler_t)adl_at_CREG_response_handler,"*",NULL);
}
void adl_at_CREG_response_handler(adl_atResponse_t *at_str_pars)
{
    ascii res[]={0,0};
    if(at_str_pars->RspID == ADL_STR_OK)
        return;
    if(at_str_pars->RspID == ADL_STR_ERROR ||
       at_str_pars->RspID == ADL_STR_CME_ERROR ||
       at_str_pars->RspID == ADL_STR_CMS_ERROR)
    {
        TRACE((1,"RILEVATO ERRORE=%d",at_str_pars->RspID));
        return;
    }
    wm_strGetParameterString ( res, at_str_pars->StrData, 2 );
    // legge parametri di registrazione
    if(res[0]=='1')
    {
        TRACE((1,"Registrato presso operatore locale"));
    }
    else if (res[0]=='5')
    {
        TRACE((1,"Registrato in roaming verso altro operatore"));
    }
    else
    {
        TRACE((1,"Non ancora registrato. "));
    }
}
```

i comandi AT per eseguire certe particolari funzioni (si veda ad esempio la gestione della tastiera presentata nella scorsa puntata). Facciamo presente, inoltre, che le librerie Open AT rappresentano un layer software sottostante ai comandi AT, il che vuol dire che quando inviamo un comando AT, questo non fa altro che richiamare delle funzioni di libreria ADL oppure altre funzioni ancora a più basso livello. A tal proposito segnaliamo che è disponibile un'applicazione Open AT "Internet plug-in" scritta da Wavecom, dove sono stati definiti dei nuovi comandi AT che permettono di gestire lo stack TCP/IP ad alto livello. Con questi specifici comandi è possibile inviare/ricevere mail, creare una connessione FTP, HTTP, UDP e molto altro. Esempi d'uso di questo plug-in si possono trovare nel manuale "AT command user guide for Wavecom IP v4.01" scaricabile dal sito Wavecom. È sufficiente scaricare dal sito l'applicazione **gcc_WIPSoft_256KB.wpb.dwl**, fare il download con il comando AT+WDWL oppure con il software DWLWIN, e avviarla con il comando AT+WOPEN=1, per avere a disposizione questa serie aggiuntiva di comandi AT.

Nella scorsa puntata abbiamo già visto come richiamare un comando AT all'interno della nostra applicazione; ora ci limitiamo pertanto a fornire i passi essenziali. Con la funzione **adl_cmdCreate** si lancia il comando AT. Il prototipo della funzione è il seguente:

```
s8 adl_atCmdCreate ( ascii * atstr, u16
rsphflag, adl_atRspHandler_t rsphdl,
... )
```

I parametri corrispondenti hanno il significato descritto qui di seguito:
atstr = si tratta della stringa contenente il comando AT da inviare;
rsphflag = mettendo FALSE a questo flag si lascia come porta per l'invio

Listato 2 - Richiamo del comando AT+WIND e rilevazione dell'evento WIND 4.

```
void adl_main( adl_InitType_e InitType )
{
    // Send AT+WIND=255
    adl_atCmdCreate("AT+WIND=255",FALSE,(adl_atUnSoHandler_t)adl_at_
WIND255_Handler,ADL_STR_OK,NULL);
    // Subscribe WIND 4
    adl_atUnSoSubscribe("+WIND: 4",(adl_atUnSoHandler_t)Wind4_Handler);
}
bool adl_at_WIND255_Handler(adl_atUnsolicited_t *paras)
{
    TRACE((1,"WIND 255 OK"));
    return TRUE;
}
bool Wind4_Handler(adl_atUnsolicited_t *paras)
{
    adl_atUnSoUnSubscribe("+WIND: 4",(adl_atUnSoHandler_t)Wind4_
Handler);
    TRACE((1,"WIND 4 OK"));
    return TRUE;
}
```

del comando AT la porta di default (Open AT virtual port); una impostazione più avanzata, che esula dagli scopi di questo articolo, permette di impostare una porta diversa tramite la macro **ADL_AT_PORT_TYPE(_port,_flag)** (per ulteriori dettagli si consulti l'"ADL User Guide");

Tabella 2 - Opzioni della funzione **adl_atCmdSubscribe**.

Tipo	Valore	Descrizione
ADL_CMD_TYPE_PARA	0x0100	Ammessi comandi AT del tipo: AT+cmd=x,y Il numero di parametri ammessi deve essere messo in OR con questa maschera: ADL_CMD_TYPE_PARA 0xXY dove X è il numero massimo di parametri ammessi e Y il numero minimo di parametri ammessi.
ADL_CMD_TYPE_TEST	0x0200	Ammessi comandi AT del tipo: AT+cmd=?
ADL_CMD_TYPE_READ	0x0400	Ammessi comandi AT del tipo: 'AT+cmd?'
ADL_CMD_TYPE_ACT	0x0800	Ammessi comandi AT del tipo: 'AT+cmd'
ADL_CMD_TYPE_ROOT	0x1000	Sono ammessi tutti i comandi che iniziano con la stringa definita ma senza il carattere ; finale che serve per il parsing di comandi AT concatenati. Quindi se la stringa è "AT-" tutti i comandi AT del tipo "AT-cmd1", "AT-cmd2" sono ammessi.

Listato 3 – Applicazione AT+LED.

```
#include "adl_global.h"
const u16 wm_apmCustomStackSize = 1024;
#define NUM_GPIO 1 // number of GPIOs sets
static adl_ioDefs_t Gpio_Config[NUM_GPIO] = {
    {ADL_IO_GPIO | 0 | ADL_IO_DIR_OUT | ADL_IO_LEV_LOW}
};
// set GPIO0 as output low
};
static s32 myGpioOut_Handle; // handle to GPIO0 out
static adl_tmr_t* tim_handle; // timer handle for led
static u8 timer_led; // timer led 0 to 1000ms
/*****
/* Function : Led_TimerHandler */
*****/
void Led_TimerHandler ( u8 ID, void * Context )
{
    adl_ioDefs_t Gpio_to_write = ADL_IO_GPIO | 0 ;

    TRACE (( 1, "Led timer" ));

    // Read output and toggle led
    if(adl_ioReadSingle( myGpioOut_Handle,&Gpio_to_write))
        // Reset output
        adl_ioWriteSingle ( myGpioOut_Handle, &Gpio_to_write,
FALSE );
    else
        // Set output
        adl_ioWriteSingle ( myGpioOut_Handle, &Gpio_to_write ,
TRUE );
}
/*****
/* Function : cmdATLED_Handler */
*****/
void cmdATLED_Handler (adl_atCmdPreParser_t *param)
{
    u16 time_ms;
    timer_led = wm_atoi(ADL_GET_PARAM(param,0));
    TRACE (( 1, "AT+LED ricevuto: Stop led timer" ));
    adl_tmrUnSubscribe( tim_handle,Led_TimerHandler,ADL_TMR_TYPE_
100MS); // Stop timer led

    if(timer_led == 0 || timer_led > 255)
    {
        TRACE (( 1, "Stop led timer" ));
        return;
    }
    time_ms = timer_led * 10; // trasformo in centinaia di ms (unità
base del timer)
    TRACE (( 1, "AT+LED Start led timer:%d s",timer_led ));
    tim_handle = adl_tmrSubscribe( TRUE,time_ms,ADL_TMR_TYPE_
100MS,Led_TimerHandler );
}
/*****
/* Function : adl_main */
*****/
void adl_main( adl_InitType_e InitType )
{
    TRACE (( 1, "Subscribe GPIO0_0"));
    // Subscribe GPIO0 output
    myGpioOut_Handle = adl_ioSubscribe(NUM_GPIO,Gpio_Con-
fig,0,0,0);
    TRACE (( 1, "handler returns %d", myGpioOut_Handle ));
    TRACE (( 1, "AT+LED Subscribe" ));
    // Sottoscrive il comando AT+LED=<parametro>, la maschera
ADL_CMD_TYPE_PARA|0X0011 indica che è un comando del tipo AT+cmd=X
con un solo parametro ammesso
    adl_atCmdSubscribe("at+led",cmdATLED_Handler,ADL_CMD_
TYPE_PARA|0X0011);
}
```

rsphdl = funzione associata al comando AT; deve essere definita all'interno del codice ed è con questa funzione che si cattura la risposta al comando AT; ... = lista di parametri di tipo stringa che deve essere in grado di rilevare la funzione di risposta. La lista deve terminare con NULL; il carattere "*" all'interno della lista indica che la funzione definita in **rsphdl** riceve tutti i parametri di risposta (in alternativa si possono elencare le stringhe di risposta che la funzione definita in **rsphdl** è in grado di rilevare).

Sempre riguardo al parametro ... , notate che quando avviene la risposta al comando AT la funzione **rsphdl** può essere richiamata più volte, una per ogni stringa di risposta generata (normalmente è richiamata due volte, una volta per la risposta contenente i parametri e un'altra volta per ricevere l'OK). È allora possibile definire solo alcune stringhe di risposta ammesse e ignorare le altre.

Quella relativa al parametro ... è comunque una funzionalità avanzata; per i nostri scopi useremo "*" oppure ADL_STR_OK qualora ci interessasse rilevare solo l'OK. Nel **Listato 1** è riportato un esempio che richiama il comando AT+CREG e ne rileva la risposta.

Con questo pezzo di codice è possibile quindi determinare lo stato di registrazione alla rete del modulo GSM.

In modo analogo a quanto visto nel **Listato 1**, si può eseguire qualsiasi altro comando AT dall'interno delle proprie applicazioni Open AT.

Nel **Listato 2** è riportato un altro esempio di un frammento di codice utile per rilevare l'evento WIND 4. Questo evento ci dice che il modulo è registrato alla rete e che la SIM è pronta all'uso. Quindi solo dopo aver rilevato il WIND 4 sarà possibile leggere e scrivere in SIM, fare chiamate, inviare SMS ed altro ancora, attraverso la propria applicazione Open AT.

Veniamo infine alla questione più interessante: la possibilità di creare i propri comandi AT personalizzati. Si supponga ad esempio di voler aggiungere alla propria

applicazione un comando AT per far lampeggiare un led con periodo da 0 (spento) a 255 s. Il led è collegato al pin GPIO_0 del modulo Augusto configurato come uscita (vedi puntata precedente). Vogliamo creare un comando AT del tipo:

AT+LED=<tempo>

dove <tempo> è un numero, in secondi, da 0 a 255 che rappresenta il periodo in cui il led rimane acceso. Il duty-cycle è fisso al 50%. Il codice dell'applicazione è riportato nel **Lista-to 3**. Il codice fa uso della funzione di libreria **adl_atCmdSubscribe** per creare il comando AT, i parametri da passare a questa funzione sono nell'ordine:

- la stringa del comando AT;
- la funzione di risposta al comando AT;
- le opzioni che definiscono il tipo di comando AT secondo la formattazione standard descritta precedentemente e il numero di parametri ammessi (vedere **Tabella 2**).

Per quanto riguarda le funzioni che gestiscono i timer e i GPIO, avendole già introdotte nella scorsa puntata, abbiamo usato parte dello stesso codice dell'esempio "Allarme" visto nel numero precedente, solo che questa volta il tempo di lampeggio del led è un parametro settabile dall'utente con il comando AT+LED.

PRINCIPALI COMANDI AT

Bene, in conclusione di questa puntata, come accennato, descriviamo i principali comandi AT previsti nel protocollo GSM (comandi AT estesi) e quelli specifici delle wireless CPU, che sono tra i più utilizzati nelle varie applicazioni.

AT

Comando di interrogazione di base.

Esempio:

AT

OK

Comandi AT di configurazione

AT+IPR=<baudrate>

Imposta la velocità della porta seriale.

Esempio:

AT+IPR=115200

OK

AT+IPR?

+IPR: 115200

OK

ATE<parameter>

Attiva/Disattiva l'eco dei caratteri.

Esempio:

ATE0 Eco disattivo

OK

ATE1 Eco attivo

OK

AT+CFUN=<parameter>

Riavvio del modulo wireless CPU.

Esempio:

AT+CFUN=1

OK

ATI3

Legge la versione del firmware.

Esempio:

ATI3

R70_00gg.WMP100 2009124 012408 21:14

OK

AT&W

Salva le impostazioni correnti.

Esempio:

AT&W

OK

Comandi AT speciali introdotti da Wavecom

AT+WIND=<IndLevel>

Attivazione dei codici di "Wavecom Indication".

<IndLevel> è il codice dell'evento da attivare secondo la **Tabella 3** riportata a pagina seguente.

bit settato a 0: indication disattivata

bit settato a 1: indication attivata

Esempio:

AT+WIND=255 attiva i primi 8 bit di

<IndLevel>

AT+WIND?

+WIND: 255

OK

Quando si genera un evento viene inviata la stringa seguente :

Tabella 3 - Descrizione dei codici <IndLevel>.

0	no unsolicited "+WIND: <event>" will occur
1 (bit 0)	activate hardware SIM insert / remove indications or SIM presence after software reset
2 (bit 1)	activate calling party alert indication
4 (bit 2)	activate idle state indication
8 (bit 3)	activate end of Wireless CPU® initialization (after PIN1/CHV 1 code entered) indication
16 (bit 4)	activate new call identifier indication
32 (bit 5)	activate call release indication
64 (bit 6)	activate network service available indication
128 (bit 7)	activate network lost indication
256 (bit 8)	activate audio ON indication
512 (bit 9)	activate SIM phonebooks reload status
1024 (bit 10)	activate SIM phonebooks checksum indication
2048 (bit 11)	activate interruption indication
4096 (bit 12)	activate hardware Rack Open/Closed Indication
8192 (bit 13)	activate NITZ indication
16384 (bit 14)	activate SMS service ready indication

WIND: <event>

dove <event> è il codice dell'evento. I codici degli eventi e la loro descrizione sono riportati nella **Tabella 4**.

AT+WDWL

Wavecom downloading (comando per programmare il modulo).

Dopo questo comando il modulo risponde con dei caratteri NAK ed è in attesa del download della applicazione. Il download può avvenire via hyper-terminal, inviando il file **.dwl** o **.wpb** in formato 1K X-Modem (vedi prima puntata del corso).

AT+WOPEN=<parameter>

Controllo dell'avvio/cancellazione della applicazione.

Esempio:

AT+WOPEN=1 Attiva l'avvio automatico dell'applicazione

OK

AT+WOPEN=0 Disattiva l'avvio automatico dell'applicazione

OK

AT+WOPEN=3 Cancella la memoria flash dati

AT+WOPEN=4 Cancella la memoria flash programma

Comandi AT riguardanti la gestione della SIM

AT+CPIN

Imposta pin o verifica lo stato della SIM

Esempio:

AT+CPIN?

+CPIN: <code>

dove <code> è lo stato della SIM riportato nella **Tabella 5**.

Se il PIN della SIM viene disabilitato oppure è già stato introdotto correttamente la risposta è READY.

AT+CPBR=<parameter>

Legge la rubrica della SIM.

Esempio:

Legge la prima voce in rubrica

AT+CPBR=1

+CPBR: 1,"112",129,"SOS"

OK

Legge le voci da 1 a 3

AT+CPBR=1,3

+CPBR: 1,"34111223344",129,"Primo nome"

+CPBR: 2,"12345678595",129,"Secondo nome"

+CPBR: 3,"01564888888",129,"Terzo nome"

OK

AT+CPBW =<idx>,<number>,<type>,<text>

Scrive in rubrica nella posizione <idx> il numero <number> associato al nome <text>. <type> è un parametro 129 numero standard oppure 145 numero con il prefisso internazionale ('+' davanti).

Esempio:

Scrive alla posizione 5

AT+CPBW=5,"0491234567",129,"Nicola"

OK

Cancella la posizione 1

AT+CPBW=1
OK

AT+CGMR=<index>

Legge gli SMS ricevuti memorizzati in SIM. <index> è la posizione in SIM dell'SMS da leggere.

Esempio:

AT+CMGR=1
+CMGR: "REC UN-
READ","0146290800","98/10/01,18:22:11+00"
ABCdefGHI
OK

AT+CMGW=<number>

Scriva SMS in SIM avente il numero <number>. Dopo l'invio (<CR>) occorre scrivere il messaggio e quindi premere CTRL-Z.

Esempio:

AT+CMGW=" +39123456" <CR>
> Ciao a tutti <CTRL-Z>
+CMGW: 1
OK

AT+CMGS=<number>

Invia un SMS verso il numero <number>. Dopo l'invio (<CR>) occorre scrivere il messaggio e quindi premere CTRL-Z.

Esempio:

AT+CMGW=" +39123456" <CR>
> Ciao a tutti <CTRL-Z>
+CMGS: 1
OK

AT+CMGD=<index>

Cancella un SMS salvato nella posizione <index> dalla SIM.

Esempio:

AT+CMGD=1
OK

ATD <numero> effettua una chiamata dati

ATD <numero>; effettua una chiamata voce

ATH termina una chiamata

Comandi AT riguardanti la rete

AT+CSQ

Verifica la forza del segnale radio (valore RSSI del campo da 0 assente a 31 ottimo, 99 non rilevabile) e il bit error rate BER (da 0 a 7, 99 non rilevabile).

Tabella 4 - Descrizione codice eventi.

Codice evento	Evento
0	the SIM presence pin has detected a "SIM removed"
1	the SIM presence pin has detected a "SIM inserted"
2	the calling party is alerting
3	the product is ready to process AT commands (except phonebooks, AOC, SMS), at init or after AT+CFUN=1
4	the product is ready to process all AT commands, end of phonebook init or swap (FDN to ADN)
5	a call <idx> has been created (after ATD or +CCWA...)
6	a call <idx> has been released, after a NO CARRIER, a "+CSSU: 5" indication, or after the release of a call waiting
7	the network service is available for an emergency call
8	the network is lost
9	the audio channel is opened
10	reload status of each SIM phonebook after init phase (after power on or SIM insertion)
11	checksum of SIM phonebooks after initialization
12	an interruption has occurred
13	the rack has been detected as closed
14	the rack has been detected as opened
15	the Wireless CPU® has received a NITZ information message from the network
16	16 SMS and SMS CB services are initialized

Esempio:

AT+CSQ
+CSQ: 17,1
OK

AT+CREG?

Verifica stato di registrazione alla rete. Risposta del tipo:

+CREG: <mode>,<stat>

dove <mode> è l'impostazione di configurazione del comando fatta con AT+CREG=<mode>, il default è 0 mentre <stat> è lo stato di registrazione dove: 0=non registrato

Tabella 3 - Descrizione dei codici <IndLevel>.

READY	ME is not writing for any password
SIM PIN	PIN 1/CHV 1 is required
SIM PUK	PUK1 is required
SIM PIN2	PIN 2/CHV 2 is required
SIM PUK2	PUK2 is required
PH-SIM PIN	SIM lock (phone-to-SIM) is required
PH-NET PIN	network personalization is required
PH-NETSUB PIN	network subset is required
PH- SERVPROV PIN	service provider is required
PH- CORPORATE PIN	corporate is required

1=registrato

2=in fase di ricerca operatore

3=registrazione negata

4=non usato

5=registrato in roaming

Esempio:

AT+CREG?

+CREG: 0,1

OK

Per ulteriori dettagli sui comandi AT qui esposti e su altri comandi AT vi invitiamo a consultare i documenti richiamati nella bibliografia.

Bibliografia:

- [1] "AT Commands Interface Guide for Open AT® Firmware v7.4"
- [2] "AT command user guide for Wavecom IP v4.01"
- [3] "ADL User Guide for Open AT® OS v6.1"

Tutti questi documenti sono scaricabili previa registrazione dal sito Internet della Wavecom, www.wavecom.com.

Modulo GSM-GPRS con CPU integrata

Tutti i prezzi si intendono
IVA inclusa.



SWMAUGUSTO è un modulo embedded GSM/GPRS che integra una CPU Wireless Wavecom serie WMP con a bordo 11 GPIO, 1 ADC, 2 ingressi di interrupt, ingressi keyboard, 2 UART, 1 SPI, 1 USB, RTC.

Permette di eseguire un codice proprietario all'interno della CPU serie WMP. Può essere utilizzato senza microcontrollore esterno, risparmiando

anche sui costi legati allo sviluppo di librerie software per la gestione dei comandi AT e di tutte le periferiche a bordo del microcontrollore le quali risultano essere già integrate nella CPU serie WMP. L'ambiente di sviluppo OPEN-AT Software Suite 2.0 di Wavecom (gratuito) mette a disposizione un sistema operativo multitasking completo di tutte le librerie ed esempi già pronti per gestire connessioni dati GSM, GPRS, SMS, voce, vivavoce con cancellatore d'eco integrato, input/output, ADC, keyboard e tutte le periferiche della CPU serie WMP.

Principali caratteristiche:

CPU: Wavecom WMF50, microcontrollore ARM 9, 32 bit, 26 MHz; memoria: 4 MB Flash, 2 MB RAM; interfacce: 11 GPIO uso generico, 2 interrupt configurabili, 1 ADC 10 bit, 5x5 keypad con antirimbombo integrato, 2 canali audio separati con DTMF integrato; interfacce seriali: 2 UART, connettore programmazione/debug (con UART 1), 1 USB SLAVE, 1 SPI I2C, JTAG; aliment.: 3,4 ÷ 4,2 Vdc.

DEMOBOARD per modulo GSM-GPRS AUGUSTO

Permette di programmare facilmente il modulo SWMAUGUSTO e di testarne tutte le periferiche. Dispone di porte COM, USB, RJ45, ingresso audio e circuito di ricarica batteria. Sono presenti inoltre trimmer per testare gli ingressi analogici, led per testare gli I/O, due pulsanti di interrupt e una

tastiera a matrice 3x2 (costituita da 6 pulsanti). Tutti i pin del modulo AUGUSTO sono riportati all'esterno su comodi pin strip passo 2,54mm.

La confezione NON comprende il modulo SWMAUGUSTO.

DEMOBOARD SWM

€295,00



**FUTURA
ELETTRONICA**

Via Adige, 11 - 21013 GALLARATE (VA)
Tel. 0331/799775 - Fax 0331/778112
www.futurashop.it

Maggiori informazioni e caratteristiche tecniche sono disponibili sul sito www.futurashop.it tramite il quale è anche possibile effettuare acquisti on-line.