

160 PAGINE!

Mensile di progettazione elettronica, attualità scientifica, novità tecnologiche.

Elettronica In

www.elettronica.in.it

oltre l'elettronica

€ 6,00

Anno XIX - n. 174
Marzo 2013

Robot Area

Piattaforma
Robotica
OmniWheels

News & Eventi



- Analizzatore di spettro su display LCD
- Micrologio: mini orologio-sveglia a microcontrollore
- Key Tel: la chiave telefonica
- Più guadagno e meno rumore con i mixer passivi
- Arduino LCD Shield
- myKIT, esperimenti con LabVIEW e myDAQ

Allarme GSM multifunzione



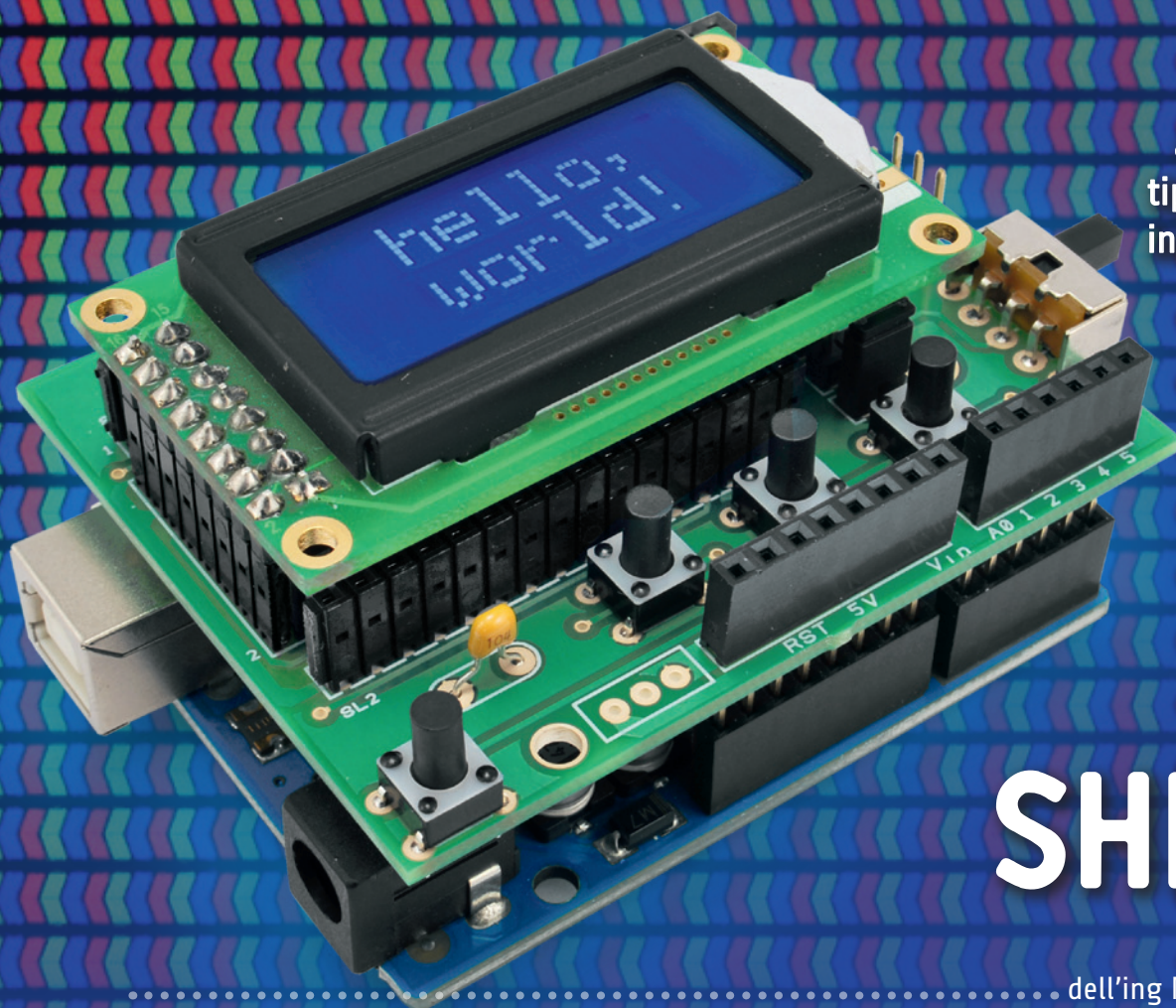
Realizziamo una lavagna interattiva



- Shield di espansione ADC per RaspberryPi compatibile con Arduino
- OpenWheels, montiamo l'elettronica



Costruiamo un visualizzatore in grado di adattarsi a tanti tipi di display con interfaccia basata su controller HD44780 o compatibile.



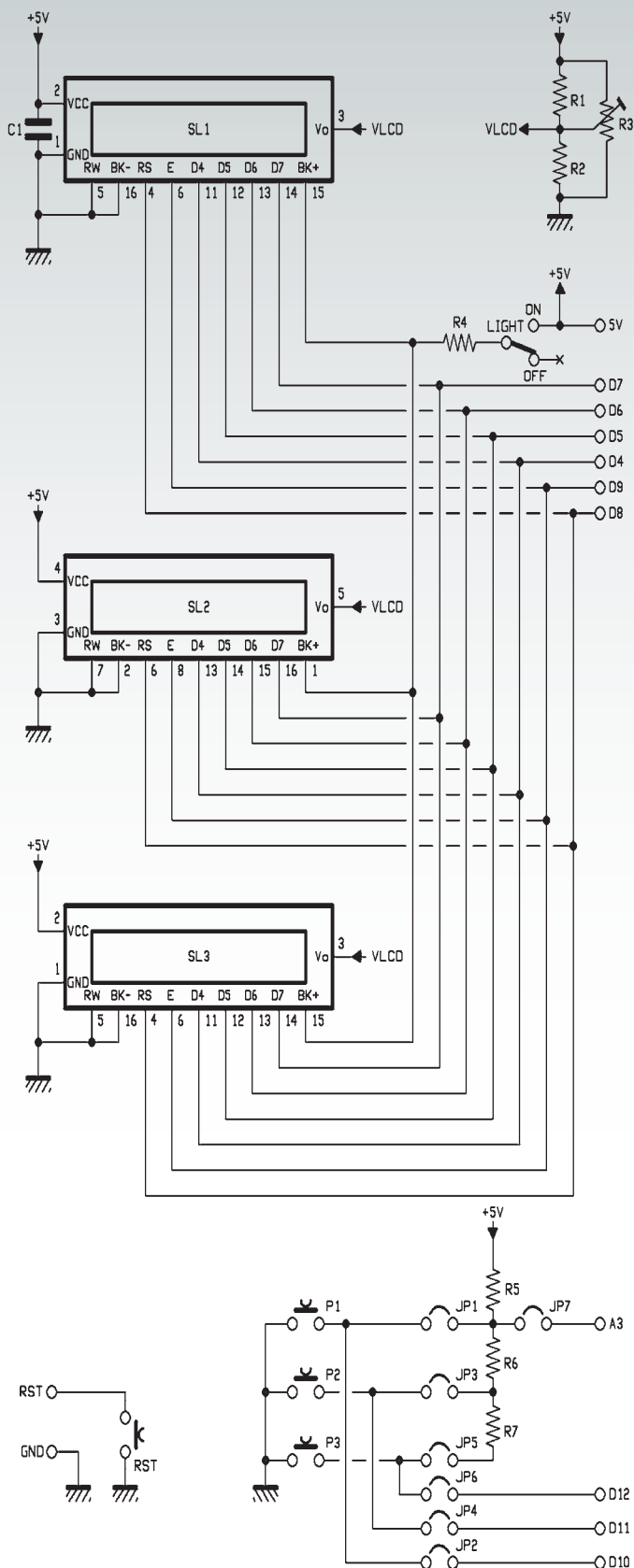
LCD SHIELD

dell'ing MIRCO SEGATELLO

Sovente nelle prime esperienze con Arduino si utilizza la funzione Serial Monitor, disponibile nell'IDE ufficiale, che permette di visualizzare dei valori a video. La funzione è nativamente supportata dal linguaggio di programmazione tramite il comando `serial.Print` e permette di visualizzare sul monitor del PC sia valori numerici che stringhe, secondo necessità. Se però vogliamo visualizzare in locale, direttamente sulla scheda Arduino, valori o stringhe, ci farebbe molto comodo avere un piccolo display e qualche pulsante utile ad esempio per impostare dei valori; l'ideale sarebbe avere tutto ciò su uno shield appositamente realizzato. Ecco perché in queste pagine vi proponiamo un LCD shield, nato per montare display LCD testuali, facilmente utilizzabile e completo di tre pulsanti per uso

generico. Per la massima compatibilità abbiamo pensato di realizzare uno shield sul quale si possano installare diversi modelli di display LCD, così da non vincolarvi ad un tipo preciso ma consentirvi di scegliere anche display reperibili come surplus. In generale possiamo dire che la compatibilità vale con la quasi totalità dei display realizzati con il chip Hitachi HD44780 (o compatibile) che sono la stragrande maggioranza in commercio. Si tratta di display a gestione parallela di 8 o 16 caratteri disposti su due righe, facilmente gestibili con un microcontrollore. La comunicazione con essi avviene tramite un bus di otto linee, ma opportunamente impostati possono colloquiare con solo quattro linee di dati alle quali occorre aggiungere una linea per il comando Register Select (RS) ed una per l'Enable

[schema **ELETTRICO**]



(E). In tutto ci si può accontentare di sei linee digitali.

Per gestire un display di questo tipo è sufficiente inviare sul bus il codice ASCII relativo al carattere da visualizzare ed agire opportunamente sulle linee RS ed E; il chip interno provvederà alla gestione fisica dei pixel affinché quel carattere sia visualizzato in modo corretto.

Per quanto ci riguarda, abbiamo testato lo shield con i seguenti display:

- LMB0820 (alfanumerico 8x2);
- ADM1602K (alfanumerico 16x2);
- DISPLAYTECH162B (alfanumerico 8x2);
- ACM1602B (alfanumerico 16x2);
- TM162AD (alfanumerico 16x2).

I primi quattro si possono acquistare da Futura Elettronica, www.futurashop.it.

Indipendentemente dal tipo di connettore adottato dal costruttore e da come sono disposti i pin, tutti i display basati su controller HD44780 o compatibile presentano la medesima interfaccia, composta da otto bit di dati, tre linee di controllo, l'alimentazione +Vcc e GND ed una linea per la regolazione del contrasto; tuttavia la disposizione delle linee cambia da modello a modello, ragion per cui prima di usare un display occorre consultarne il data-sheet. In alcuni modelli è previsto un retroilluminatore a LED, alimentato tramite due contatti del connettore.

In **Tabella 1** sono riportate le connessioni tra un generico display basato sul chip HD44780 e la scheda Arduino.

La linea R/W ha una funzione speciale e viene abilitata durante la lettura di alcuni particolari parametri contenuti nella memoria del display; nel nostro shield R/W sarà sempre posta a zero, il che abilita la sola scrittura nel display. Non dovete comunque preoccuparvi di alcunché, in quanto per controllare i display del genere non vi serve conoscere protocolli dati e roba simile, dato che la gestione di tutte le linee viene demandata alla libreria disponibile nell'ambiente di sviluppo (IDE) di Arduino.

Lo shield dispone di tre diversi tipi di connettori a seconda del tipo di display che vorrete connettere. I connettori sono posizionati in modo da lasciare spazio a tre pulsanti generici ed al pulsante di reset; i contatti della scheda Arduino sono sempre disponibili dal lato di alimentazione, mentre la disponibilità dei contatti digitali (da 0 a 13) è assicurata solo per i display più piccoli. Come potete vedere dalle foto del prototipo, il

miglior risultato in fatto di estetica e fruibilità dei contatti si ottiene con il display 8x2 caratteri, che tra l'altro è molto economico, ha i caratteri bianchi su sfondo blu e dispone di retroilluminazione. Per funzionare correttamente, ciascun display LCD necessita di una piccola tensione per la regolazione del contrasto, la quale normalmente viene ricavata tramite un trimmer i cui estremi sono alimentati dagli stessi 5 volt dell'alimentazione principale.

I display da noi testati hanno tutti una tensione di contrasto molto simile e nota, per cui è possibile omettere il trimmer e saldare le sole resistenze R1 e R2 che formano un partitore calibrato per ottenere la giusta tensione di contrasto.

Nel prototipo da noi collaudato è stata preferita questa seconda modalità, ma, nel caso utilizzate il display ADM1602K (scritte bianche su sfondo nero) suggeriamo l'utilizzo del trimmer per regolare il contrasto con più precisione.

Nel momento in cui inserirete lo shield sulla scheda Arduino, tutte le linee del display saranno connesse ai rispettivi pin di Arduino e la scheda sarà già pronta per funzionare. Con la scelta dei collegamenti da noi proposta rimangono liberi tutti i pin analogici (tranne A3) comprese le linee SDA e SCL per il bus I²C e tutte le linee per il bus SPI.

Ricordate che, mentre il display vero e proprio assorbe pochi milliamper, la sua retroilluminazione può raggiungere un assorbimento di 50÷100 mA, il che nelle applicazioni con alimentazione a batteria risulta decisamente troppo; pertanto abbiamo previsto un interruttore che vi dà la possibilità di inserire o disinserire la retroilluminazione.

Il retroilluminatore consiste semplicemente in un gruppo di LED connessi in serie e parallelo che normalmente si accendono alla massima intensità se si fornisce un'alimentazione di 5 volt; per altri display, invece, la tensione deve essere più bassa. Ad esempio, nel display LMB0820 la retroilluminazione si attiva con una tensione di 4,05 volt (l'assorbimento è di 60 mA) mentre nel modello TM162 sono necessari 4,2 volt (l'assorbimento è 110 mA).

Per evitare che i LED della retroilluminazione si guastino, occorre limitarne la corrente; allo scopo abbiamo previsto una resistenza in serie (R4) del valore di 22 ohm, adatta alla maggior parte dei display. Se alimentate Arduino con un alimentatore (usando il connettore di alimentazione esterna) che eroga una tensione superiore agli 8

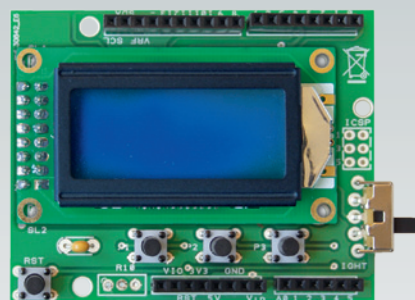


Fig. 1
LCD shield
con display
LMB0820.

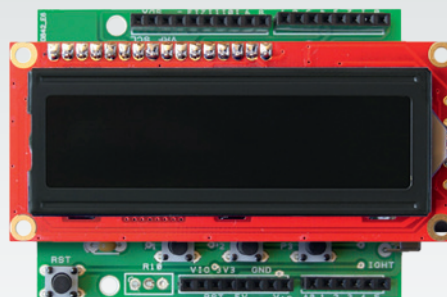


Fig. 2
LCD shield
con display
ADM1602K.

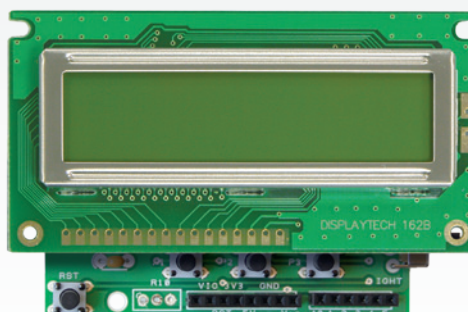


Fig. 3
LCD shield
con display
DISPLAYTECH162B.

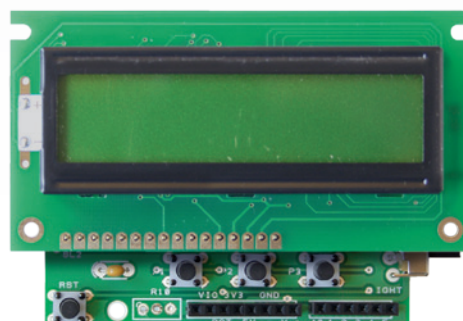


Fig. 4
LCD shield
con display
ACM1602B.

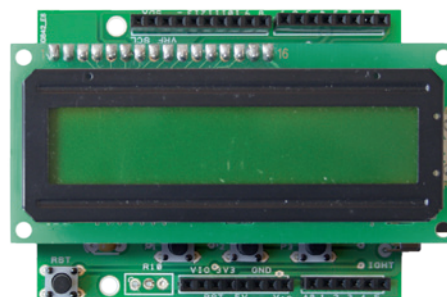
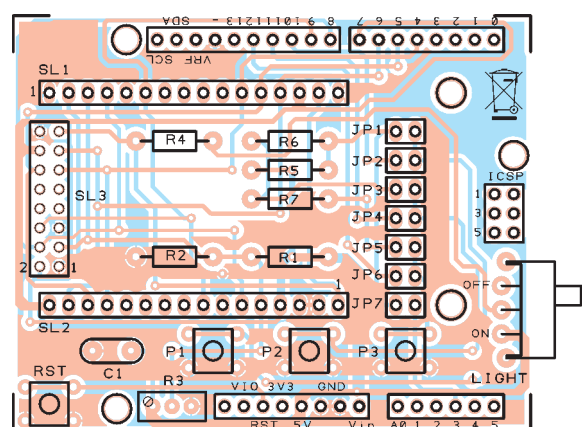
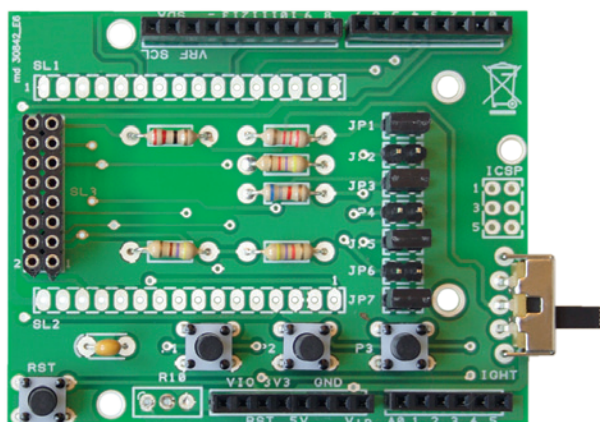


Fig. 5
LCD shield
con display
TM162AD.



Elenco Componenti:

- | | |
|--|---|
| R1: 4,7 kohm | - Strip Maschio/Femmina 6 poli (1 pz.) |
| R2: 470 ohm | - Strip Maschio/Femmina 8 poli (2 pz.) |
| R3: Trimmer multigiri 10 kohm | - Strip Maschio/Femmina 10 poli (1 pz.) |
| R4: 22 ohm | - Strip Maschio 2 poli (7 pz.) |
| R5: 4,7 kohm | - Strip femmina 8 poli (2 pz.) |
| R6: 2,2 kohm | - Strip Femmina 16 poli (2 pz.) |
| R7: 6,8 kohm | - Jumper (4 pz.) |
| C1: 100 nF ceramico | - Circuito stampato |
| P1 ÷ P3: Pulsante N.A. da c.s. | |
| RST: Pulsante N.A. da c.s. | |
| LIGHT: Deviatore a slitta 90° | |
| Varie: | |
| - Strip Maschio/Femmina 3 poli (2 pz.) | |

volt, consigliamo di elevare il valore della R4 a 47 ohm, per evitare che lo stabilizzatore di tensione a 5 volt interno si surriscaldi. Normalmente chi ha bisogno di un display lo fa per interagire con Arduino, quindi ci è sembrato logico prevedere anche dispositivi di input, che nel nostro caso sono i tre pulsanti. Le funzioni che di solito ven-

Tabella 1 - Assegnazione pin display LCD ad Arduino

PIN DISPLAY	FUNZIONE	PIN DI ARDUINO
VDD	Alimentazione +5volt	+5V
Vss	Alimentazione GND	GND
Vo	Tensione per il contrasto	Trimmer
RS	Selezione scrittura di dati o comandi	8
R/W	Abilita la lettura o la scrittura all'interno del display	GND
Enable	Linea di abilitazione	9
DB0	Linea dati 0	Non usato
DB1	Linea dati 1	Non usato
DB2	Linea dati 2	Non usato
DB3	Linea dati 3	Non usato
DB4	Linea dati 4	4
DB5	Linea dati 5	5
DB6	Linea dati 6	6
DB7	Linea dati 7	7
BL+	Retroilluminazione LED+	+5V
BL-	Retroilluminazione LED-	GND

gono implementate sono l'incremento di una variabile (+), il decremento di una variabile (-) e la funzione *Enter* per confermare il dato. Siccome il nostro intento è offrirvi uno shield molto flessibile, abbiamo predisposto due modalità di collegamento dei pulsanti: la prima è quella classica che fa uso di tre linee digitali di Arduino (nello specifico i pin 10, 11, 12) occupando di fatto la porta SPI e lasciando libero il pin 13, che corrisponde al LED di sistema; in questo modo le uniche linee digitali disponibili sono la 0, 1, 2, 3 e la 13. Per attivare questa modalità dovete disporre i jumper JP2, JP4, JP6, JP7, come visibile in **Fig. 6**. Non abbiamo previsto le classiche resistenze di pull-UP in quanto già implementate nel microcontrollore a bordo di Arduino e facilmente attivabili via software tramite la funzione `pinMode(buttonPin, INPUT_PULLUP)`; la funzione dei pulsanti è quindi in logica negativa: premendo, il valore è zero, mentre a riposo il valore è uno.

La seconda modalità prevede l'utilizzo di una sola linea analogica, ovvero la 3, che permette di lasciare disponibili ben otto linee digitali e le

Tabella 2 - Contatti dello shield assegnati ai pulsanti.

PULSANTE	PIN DI ARDUINO (MODALITÀ DIGITALE)	PIN DI ARDUINO (MODALITÀ ANALOGICA)
P1	10	AN3
P2	11	AN3
P3	12	AN3

Tabella 3 - Modalità analogica di lettura dei pulsanti

PULSANTE PREMUTO	TENSIONE AN3	VALORE ADC
Nessuno	5v	1024
P1	0v	0
P2	1,6v	328
P3	3,3v	676

Tabella 4 - Configurazione motorshiled_FE

JUMPER MOTORSHIELD_FE	PIN DI ARDUINO
PWMA	3
PWMB	11
DIRA	2
DIRB	12

due linee analogiche 4 e 5, utilizzate dal bus I²C; per attivare questa modalità dovete chiudere i jumper JP1, JP3, JP5, JP7, come visibile in Fig. 7. Con questa modalità, tranne la linea 3, tutte le altre possono essere usate come linee analogiche e se necessario si possono usare le linee 4 e 5 per il collegamento di periferiche I²C.

Per capire come sia possibile utilizzare una sola linea analogica per gestire i tre pulsanti bisogna riferirsi, nello schema elettrico, alla rete formata dalle resistenze R5, R6 ed R7; queste, poste in serie, formano un partitore di tensione a scala. Quando nessun pulsante è premuto, la linea AN3 si trova alla tensione di 5 volt perché nessuna corrente fluisce verso massa; se premiamo il pulsante P1, la linea AN3 viene cortocircuitata a massa e la sua tensione si annulla (0 volt). Se premiamo il pulsante P2, il nodo posto tra la resistenza R6 ed R7 viene posto a massa, cosicché la linea AN3, connessa al nodo tra R5 ed R6, raggiunge la tensione di $5 \cdot R6 / (R5 + R6) = 1,6$ volt.

Se, invece, premiamo il pulsante P3, la resistenza R7 viene cortocircuitata a massa e la linea AN3 assume il potenziale di $5 \cdot (R6 + R7) / (R5 + R6 + R7) = 3,3$ V. Questo significa che per ciascun pulsante premuto, la linea AN3 assume un potenziale diverso corrispondente ad un diverso valore della lettura del convertitore ADC di Arduino. La funzione `analogRead(A3)` fornirà un valore numerico per ciascuno dei pulsanti premuti, secondo quanto riportato nella Tabella 3.

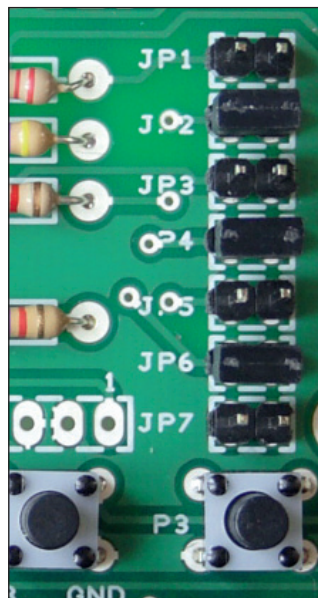


Fig. 6 - Posizione dei jumper per la gestione digitale dei pulsanti.

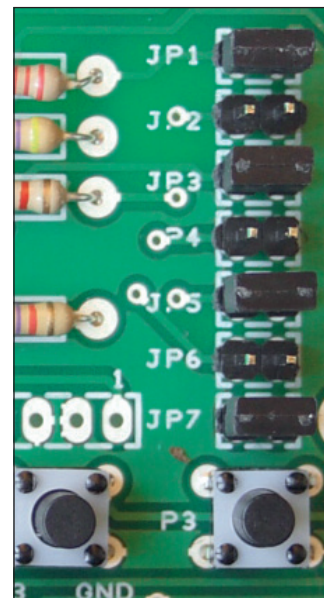


Fig. 7 - Posizione dei jumper per pulsanti connessi al pin analogico.

Questo sistema consente di leggere lo stato di tre pulsanti usando una sola linea analogica, però non è in grado di discriminare se più pulsanti sono premuti contemporaneamente; se ad esempio premiamo P1 la tensione su AN3 sarà di zero volt e non sarà diversa anche se contemporaneamente dovessimo premere gli altri due pulsanti. Concludiamo la descrizione dello schema elettrico con il pulsante RST, che replica il pulsante di reset presente sulla scheda Arduino; lo abbiamo voluto perché montato lo shield, tale pulsante non è più raggiungibile.

Nella scelta dei pin utilizzati dallo shield abbiamo tenuto conto della compatibilità con altri shield; in particolare, è possibile

liberare i pin relativi al bus SPI utilizzati ad esempio dallo shield Wi-Fi, dall'SD-shield e i pin del bus I²C (AN4 e AN5) utilizzati da molti sensori, come accelerometri e giroscopi.

Abbiamo assicurato la compatibilità anche con la MOTORshield_FE, grazie anche al fatto che di quest'ultima è possibile configurare i pin da usare. In questo caso è necessario utilizzare la linea AN3 per rendere operativi i pulsanti della LCD shield e

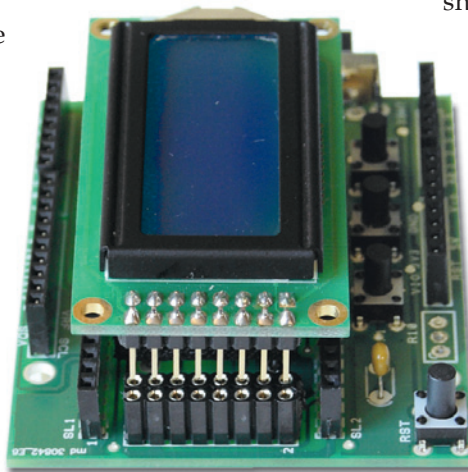


Fig. 8 - Particolare dei connettori del display.

Listato 1

```
/*
  LCDshieldFE_1 Esempio1
  Pulsanti connessi ai pin digitali
*/

// include la libreria:
#include <LiquidCrystal.h>
#define P1 10
#define P2 11
#define P3 12
// inizializza il display assegnando un nome
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup() {
  // Imposta gli ingressi dei pulsanti
  pinMode(P1, INPUT_PULLUP);
  pinMode(P2, INPUT_PULLUP);
  pinMode(P3, INPUT_PULLUP);
  // Imposta il numero di righe e colonne del display
  lcd.begin(16, 2);
  // Visualizza il messaggio
  lcd.print("Hello!");
}

void loop() {
  // si posiziona sulla riga sottostante
  lcd.setCursor(0, 1);
  // visualizza il pulsante premuto
  if ( digitalRead(P1)==0 ) lcd.print("P1 ");
  if ( digitalRead(P2)==0 ) lcd.print("P2 ");
  if ( digitalRead(P3)==0 ) lcd.print("P3");
}
```

Listato 2

```
/*
  LCDshieldFE_2 Esempio2
  Pulsanti connessi alla linea analogica
*/

// include la libreria:
#include <LiquidCrystal.h>
#define pushPin 3
// inizializza il display assegnando un nome
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup() {
  // Imposta il numero di righe e colonne del display
  lcd.begin(16, 2);
  // Visualizza il messaggio
  lcd.print("Hello!");
}

void loop() {
  // si posiziona sulla riga sottostante
  lcd.setCursor(0, 1);

  // visualizza il pulsante premuto
  int pushValue = analogRead(pushPin);
  if ( pushValue < 164 )
    lcd.print("P1");
  else if ( pushValue < 502 )
    lcd.print("P2");
  else if ( pushValue < 850 )
    lcd.print("P3");
  else
    lcd.print(" ");
}
```

configurare i pin della motorshield_FE come nella **Tabella 4**.

È perciò possibile utilizzare contemporaneamente sia la motorschild_FE che la LCDshield_FE. Nella motorshield_FE il controllo delle linee PWM avviene con i pin PWM 3 e 11, utilizzando entrambi il Timer 2, lasciando liberi gli altri timer; ciò è importante perché se fosse necessario modificare la frequenza del PWM, sarebbe sufficiente agire sui registri del solo timer 2, lasciando inalterati i registri degli altri timer di solito usati da Arduino per funzioni importanti come *millis* e *delay*.

REALIZZAZIONE PRATICA

La costruzione dello shield non presenta difficoltà: come al solito saldate per prime le resistenze e via via i componenti a più alto profilo. Per connettere il display allo shield consigliamo di saldare sul display un connettore maschio e sulla scheda un connettore femmina, come visibile in **Fig. 8**. Non è necessario saldare tutti i connettori per il display, ma solo quello che interessa; come già specificato, per la regolazione del contrasto saldate il trimmer o le resistenze R1 ed R2, ma non entrambi. Lo shield è compatibile con Arduino UNO rev3; nel caso di versioni precedenti è assicurata la compatibilità anche se gli ultimi due pin a sinistra non troveranno il connettore di Arduino e rimarranno volanti.

Per quanto riguarda la programmazione attraverso l'IDE di Arduino, non ci sono particolari difficoltà: non è necessaria una specifica libreria in quanto quella di sistema, denominata *liquid Crystal*, comprende già tutte le necessarie funzioni; vediamo di seguito, istruzione per istruzione.

- **LiquidCrystal lcd(rs, enable, d4, d5, d6, d7);** inizializza il display e lo dichiara come oggetto di nome "lcd" usato successivamente per la chiamata alle funzioni della libreria. Devono essere specificati i pin utilizzati da Arduino per comandare il display, quattro linee di dati e due linee di controllo.
- **lcd.begin(cols, rows);** imposta per l'oggetto "lcd" il numero di colonne (cols) ed il numero di righe (rows). Per il display LMB0820 bisogna specificare i valori 8 e 2, mentre per tutti gli altri, i valori 16 e 2.
- **lcd.clear();** cancella il contenuto del display.
- **lcd.home();** posiziona il cursore all'inizio della prima riga e prima colonna.
- **lcd.setCursor(thisRow, thisCol);** posiziona il cursore nella riga *thisRow* (0 è la prima riga) e nella colonna *thisCol* (0 è la

Tabella 5 - Esempi disponibili nella libreria *liquidCrystal*

AUTOSCROLL	DIMOSTRAZIONE DELLE FUNZIONI AUTOSCROLL() E NOAUTOSCROLL() DA APPLICARE PER LE SCRITTE SCORREVOLI
Blink	Esempio nel quale si abilita e disabilita il lampeggio del cursore
Cursor	Esempio nel quale si abilita e disabilita la visualizzazione del cursore
CustomCharacter	Esempio di utilizzo di caratteri personali
Display	Dimostrazione delle funzioni display() e noDisplay() per attivare o disabilitare la visualizzazione.
HelloWorld	Visualizza la scritta Hello World ed il valore di un contatore
Scroll	Visualizza la scritta "Hello World!" ed utilizza le funzioni scrollDisplayLeft() e scrollDisplayRight() per farla scorrere.
SerialDisplay	Visualizza i caratteri inviati da PC ad esempio con SerialMonitor
SetCursor	Come impostare la posizione del cursore sul display
TextDirection	Dimostrazione delle funzioni leftToRight() and rightToLeft() per muovere il cursore.

prima colonna). Ad esempio, per posizionare il cursore a metà della seconda riga, in un display 16x2, si scriverà **lcd.setCursor(1,8);**.

- **lcd.write(data);** scrive un carattere sul display fornendone il relativo codice ASCII.
- **lcd.print("testo");** scrive un testo sul display a partire dalla posizione attuale del cursore.
- **lcd.print(var);** scrive il valore della variabile "var" sul display a partire dalla posizione attuale del cursore. Se la variabile è un float, verranno scritte tutta la parte intera e due sole cifre della parte decimale.
- **lcd.cursor();** attiva la visualizzazione del cursore sulla posizione attuale.
- **lcd.noCursor();** disattiva la visualizzazione del cursore sulla posizione attuale.
- **lcd.blink();** attiva il lampeggio del cursore (se visualizzato).
- **lcd.noBlink();** disattiva il lampeggio del cursore (se questo è visualizzato).
- **lcd.display();** attiva la visualizzazione sul display.
- **lcd.noDisplay();** disattiva la visualizzazione sul display.
- **lcd.scrollDisplayRight();** trasla il testo del display di una posizione a destra.
- **lcd.scrollDisplayLeft();** trasla il testo del display di una posizione a sinistra.
- **lcd.autoscroll();** abilita lo spostamento automatico del cursore ad ogni inserimento di un nuovo carattere.
- **lcd.noAutoscroll();** disabilita lo spostamento automatico del cursore ad ogni inserimento di un nuovo carattere.
- **lcd.leftToRight();** imposta la direzione nella quale sarà inserito il prossimo carattere. Per impostazione predefinita il carattere sarà aggiunto a destra dell'ultimo carattere inserito.
- **lcd.rightToLeft();** imposta la direzione nella quale sarà inserito il prossimo carattere. Il carattere sarà aggiunto a sinistra dell'ultimo carattere inserito.

- **lcd.createChar();** permette di creare un carattere speciale non incluso nel set base dei caratteri del display. È possibile creare sino ad otto caratteri speciali (da 0 a 7) con una risoluzione di 8x5 pixel. Il nuovo carattere viene specificato con un vettore di byte contenente i valori dei pixel che dovranno essere visualizzati. La funzione *write* sarà usata per la visualizzazione dei caratteri personali (indice da 0 a 7).

Tutti gli esempi disponibili con la libreria sono compatibili con il nostro LCDshield, però è necessario specificare con attenzione i pin utilizzati quando si dichiara l'oggetto *LiquidCrystal(rs,*

Listato 3

```

/*
  LCDshieldFE_3 Esempio3
  Visualizza carattere personale
*/

#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

// crea il carattere speciale "smiley":

byte smiley[8] = {
  0b00000,
  0b00000,
  0b01010,
  0b00000,
  0b00000,
  0b10001,
  0b01110,
  0b00000
};

void setup() {
  // crea il nuovo carattere nella posizione 0
  lcd.createChar(0, smiley);
  lcd.begin(16, 2);
  lcd.print("Hello! ");
  // visualizza il carattere speciale
  lcd.write((byte)0);
}

void loop() {
}

```


enable, d4, d5, d6, d7):

rs → 8
enable → 9
d4 → 4
d5 → 5
d6 → 6
d7 → 7

La corretta sintassi per creare l'oggetto "lcd" è: `LiquidCrystal lcd(8, 9, 4, 5, 6, 7)`; modificate quindi questa riga per rendere compatibili gli esempi della libreria con il nostro shield.

È necessario specificare anche il tipo di display utilizzato tramite la riga `lcd.begin(numCaratteri, numRighe)`; `numCaratteri` indica il numero di caratteri del display (di solito 8, 16 o 20) mentre `numRighe` il numero di righe (che di solito è 2 o 4). Per l'utilizzo pratico della libreria abbiamo pensato di descrivere alcuni semplici sketch direttamente derivati dagli esempi disponibili nella libreria stessa. Il primo esempio semplicemente visualizza la classica scritta "Hello!" e, nella riga

sottostante, mostra il nome del pulsante premuto; i pulsanti sono gestiti in modalità digitale. Il secondo esempio visualizza la scritta "Hello!" e, nella riga sottostante, il nome del pulsante premuto; i pulsanti funzionano, in questo caso, nella modalità analogica.

Ogni display consente la creazione di un massimo di otto caratteri personali definiti a piacere; nel terzo esempio vediamo come poterne creare uno e visualizzarlo. Ciascun carattere viene definito tramite una matrice di 8x5 pixel che, nel codice per Arduino, viene specificata con un vettore contenente otto byte per ciascun carattere, riferito ad una riga. Per ciascuno dei cinque pixel della riga viene riportato uno "zero" se il pixel è spento e un "uno" se il pixel è acceso. È sufficiente richiamare la funzione `lcd.createChar(numCarattere, nomeCarattere)` per creare il carattere nella memoria del display ed usare il comando `lcd.write(numCarattere)` per visualizzarlo; nei **Listato 1**, **Listato 2** e **Listato 3** trovate degli esempi di quanto detto. ■



Non perdere questa strepitosa offerta!

Volume "Primi passi con ARDUINO" e scheda Arduino UNO R3

solo €29,90, risparmi €4,50!

cod. ARDUKITBOOK1

FUTURA ELETTRONICA

Via Adige, 11 - 21013 Gallarate (VA)
Tel. 0331/799775 • Fax. 0331/792287

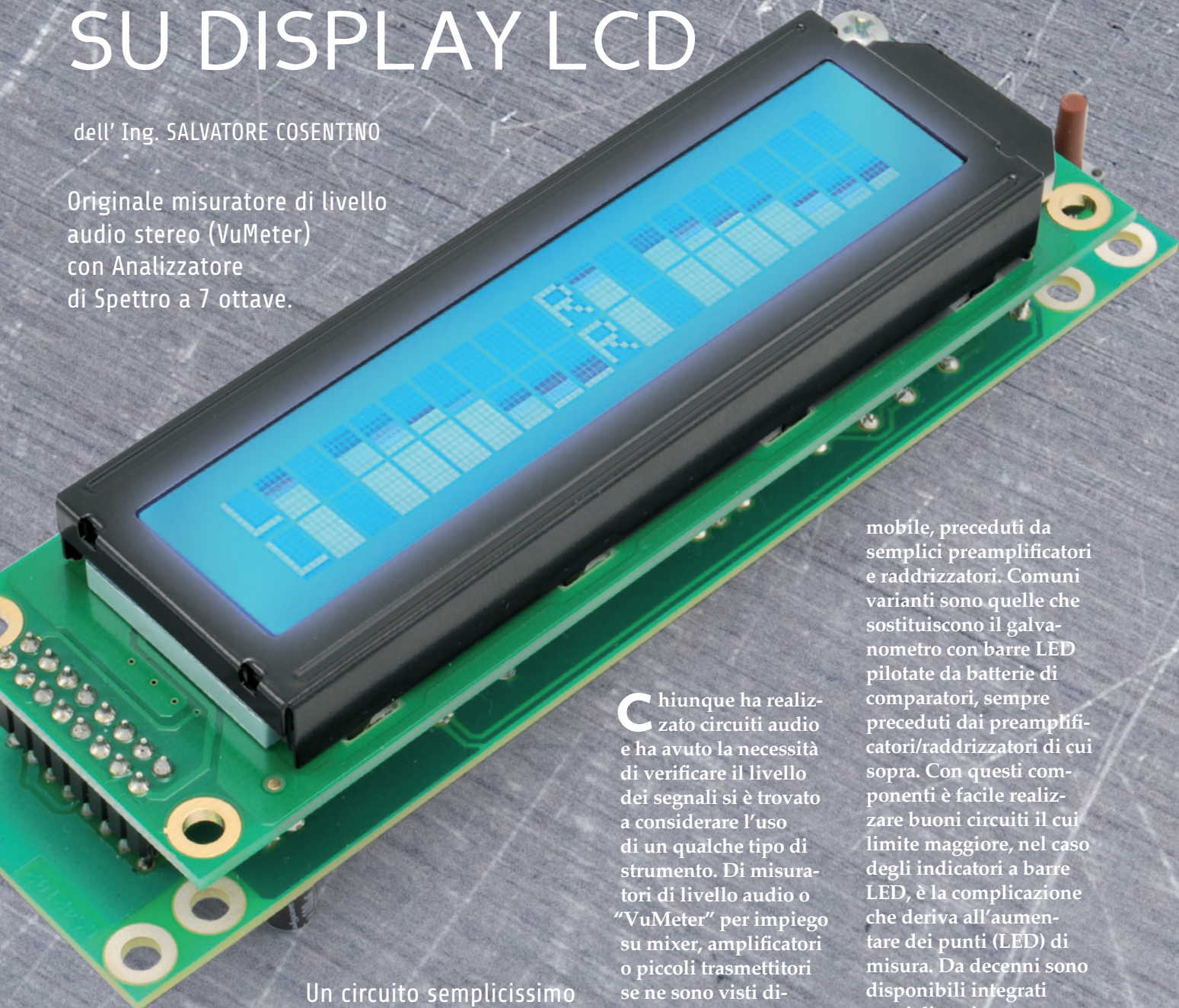
Maggiori informazioni e acquisti on-line sul sito www.futurashop.it

Prezzo IVA incluso

ANALIZZATORE DI SPETTRO AUDIO FFT SU DISPLAY LCD

dell' Ing. SALVATORE COSENTINO

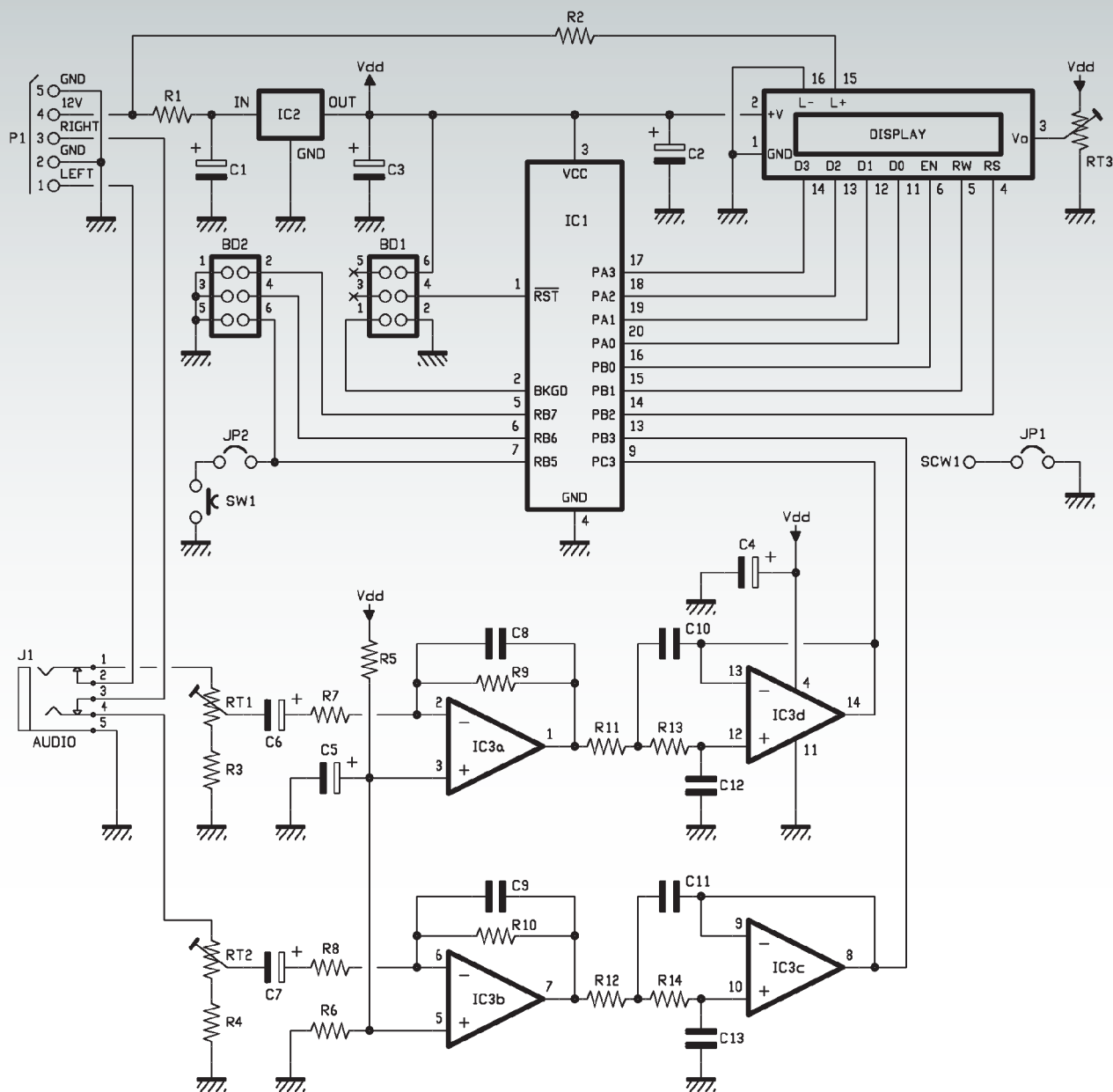
Originale misuratore di livello
audio stereo (VuMeter)
con Analizzatore
di Spettro a 7 ottave.



Un circuito semplicissimo
dalle prestazioni intriganti e
di facile uso che ci avvicina alla teoria
matematica del campionamento e delle FFT.

Chiunque ha realizzato circuiti audio e ha avuto la necessità di verificare il livello dei segnali si è trovato a considerare l'uso di un qualche tipo di strumento. Di misuratori di livello audio o "VuMeter" per impiego su mixer, amplificatori o piccoli trasmettitori se ne sono visti diversi e gli oggetti più comuni e tradizionali sono basati sui classici galvanometri a bobina

mobile, preceduti da semplici preamplificatori e raddrizzatori. Comuni varianti sono quelle che sostituiscono il galvanometro con barre LED pilotate da batterie di comparatori, sempre preceduti dai preamplificatori/raddrizzatori di cui sopra. Con questi componenti è facile realizzare buoni circuiti il cui limite maggiore, nel caso degli indicatori a barre LED, è la complicazione che deriva all'aumentare dei punti (LED) di misura. Da decenni sono disponibili integrati specializzati per questo scopo, ma sono oramai di difficile reperibilità e comunque costosi.



Su apparati di tipo consumer, anche di costo non elevato, sono diventati frequenti display elettronici a LED o a plasma che visualizzano, oltre al livello audio generale, quello in varie bande di frequenza. Chi ha provato a cercare gli schemi dei circuiti impiegati allo scopo ha in genere scoperto che sono basati sull'uso di circuiti integrati specializzati analogici che integrano batterie di filtri Passa

Banda per separare il segnale audio su più bande che sono poi separatamente rettificate, processate ed inviate ad un opportuno display a barre, in genere di tipo proprietario. Questi componenti non sono facilmente reperibili per un hobbista ed il costo non sarebbe comunque trascurabile. Un esempio immediatamente accessibile ai lettori di Elettronica In è quello presentato sul numero 90 del 2004 che si può

facilmente scaricare dal sito della rivista. È interessante notare che in questo circuito, realizzato su una scheda di dimensioni non proprio contenute, la maggior parte della circuiteria era di tipo "discreto" con integrati di tipo comune a parte i comparatori di livello/driver dei LED. Proprio questo, a parte il display a LED, ne ha condizionato le dimensioni. Da molti decenni è nota la tecnologia FFT (Fast Fourier

Transform) basata sulla stessa teoria matematica che permette di ricavare informazioni sulla frequenza dei singoli componenti di un qualsivoglia inviluppo complesso di segnale analogico. In pratica quello che un Analizzatore di Spettro ricava per via puramente analogica (L'Analizzatore di Spettro convenzionale è fondamentalmente un ricevitore a banda stretta con sintonia variabile che scansiona sistematicamente un intervallo di frequenze), tramite FFT è ottenuto con metodi di analisi e processo digitale a partire da una stringa di campioni di segnale di lunghezza opportuna. Per molto tempo le FFT sono state utilizzate per l'analisi di segnali a frequenza molto bassa ove è impraticabile l'analisi analogica classica e, comunque, non è richiesta una elevata velocità di elaborazione. Da anni il progredire della tecnica ha reso possibile l'elaborazione in tempo reale di segnali, in banda audio e oltre, su comuni PC che utilizzano questa tecnica per gli scopi più disparati e all'interno dei programmi di analisi e compressione audio. Con opportune limitazioni, gli algoritmi FFT sono oramai anche alla portata dei microprocessori più piccoli, persino a 8 bit. In questo caso le limitazioni sono ancora evidenti: sebbene esistano microcontrollori (di seguito μC) molto potenti e di costo limitato a maggior numero di bit (vedi il proliferare di ARM a 16/32 bit, del costo di pochi euro), il quantitativo limitato di memoria RAM e la scarsa potenza specifica delle istruzioni a 8 bit, unitamente alla velocità non troppo elevata, rendono difficile o impossibile l'analisi con FFT in campo audio in tempo reale da parte dei piccoli microcontrollori. Lo scrivente si è interessato da qualche tempo allo studio di

Gli schermi flessibili di Samsung



Qualche anno fa erano stati presentati degli schermi di tipo "e-paper" flessibili: adatti all'uso per degli ebook, ma poco interessanti per altre applicazioni a causa della loro natura monocromatica e non retroilluminata. A livello di prototipo erano stati realizzati degli schermi da arrotolare, ma non avevano poi riscosso particolare interesse.

Con lo sviluppo degli schermi YOUM, Samsung propone invece la qualità dei suoi attuali schermi AMOLED in versione flessibile. A differenza degli LCD, gli AMOLED producono direttamente la luce per ciascun pixel grazie alla natura foto emettitrice della sostanza organica utilizzata. Ora gli ingegneri di Samsung hanno risolto i problemi per realizzare il pilotaggio dei singoli pixel RGB su un supporto flessibile, contenente tutta l'elettronica necessaria ad ottenere la matrice attiva. Da questo successo nasce quindi uno schermo flessibile sottile e di elevata qualità, tanto definito e luminoso, quanto gli attuali AMOLED basati su vetro. I prototipi mostrati al CES e al MWC includevano – fra gli altri – uno smartphone con lo schermo

riplegato anche sui bordi e un telefono da aprire come un libro per trasformarlo in un tablet. La presentazione di questa tecnologia, denominata YOUM, ha lasciato intendere che potrà essere la prossima arma per sbaragliare la concorrenza e almeno in termini di qualità non c'è che essere stupiti. I più informati parlano già di fine 2013 o inizio 2014 per un prossimo Galaxy in versione doppio schermo, così come mostrato nel video di presentazione.



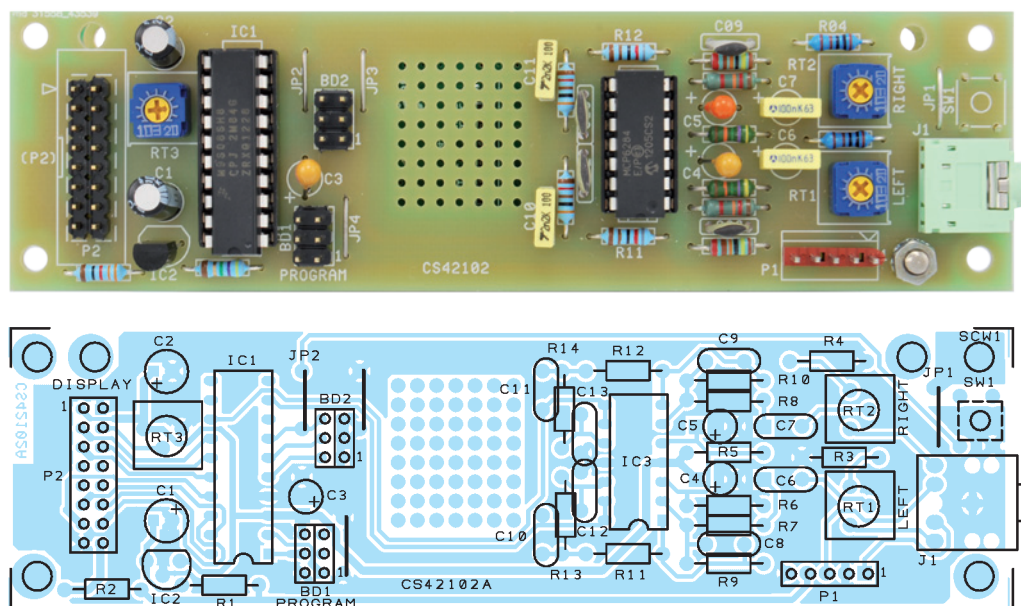
Elenco Componenti:

R1: 47 ohm
 R2: 330 ohm
 R3: 1 kohm
 R4: 1 kohm
 R5: 470 kohm 1%
 R6: 470 kohm 1%
 R7: 12 kohm
 R8: 12 kohm
 R9: 140 kohm 1%
 R10: 140 kohm 1%
 R11, R12: 8,2 kohm 1%
 R13, R14: 30 kohm 1%
 RT1÷RT3: Trimmer 10 kohm
 C1: 100 μ F 25 VL elettrolitico
 C2: 100 μ F 25 VL elettrolitico
 C3, C4: 4,7 μ F 16 VL tantalio
 C5: 1 μ F 35 VL tantalio
 C6, C7: 100 nF 63 VL poliestere
 C8, C9: 100 pF ceramico
 C10, C11: 2,2 nF 63 VL poliestere

C12, C13: 220 pF ceramico
 IC1: MC9S08SH8CPJ (MF1056)
 IC2: 78L05
 IC3: MCP6284
 SW1: Microswitch

DISPLAY:
 LCD Alfanumerico
 2x20 116x37mm
 Varie:
 - Zoccolo 7+7
 - Zoccolo 10+10

- Strip 3 vie maschio (4 pz.)
 - Strip 5 vie maschio
 - Strip 8 vie maschio (2 pz.)
 - Strip 8 vie femmina (2 pz.)
 - Presa jack 3,5mm da CS
 - Circuito stampato



questi argomenti ed ha sviluppato algoritmi che possono essere utilizzati con questi componenti sia per l'uso pratico, sia per il confronto di prestazioni tra varie classi e famiglie di microcontrollori. Lo scopo di questo articolo è proprio quello di verificare sul campo reale queste limitazioni e il loro impatto sull'uso pratico. Allo scopo si è utilizzato un μ C a 8 bit di costo inferiore a 1 euro per un centinaio di pezzi e meno di due euro in quantità singola, se acquistati al posto giusto. Questo articolo vi propone un'interessante utilizzazione di questo piccolo dispositivo nella realizzazione di un "VuMeter" di tipo completamente digitale, sia con visualizzazione del livello di picco classico tramite barre pseudo-analogiche, sia nel dominio della frequenza con più bande singole all'interno dei canali

stereo Destro e Sinistro. Il display utilizzato per la visualizzazione è un comune modulo LCD alfanumerico da 2x20 caratteri con retroilluminazione a basso consumo. Gli unici componenti utilizzati in aggiunta al display e al μ C sono un quadruplo amplificatore operazionale e la circuiteria associata per realizzare una coppia di circuiti preamplificatore / filtro Passa Basso, uno per ciascuno dei canali Destro e Sinistro. Cercheremo di ridurre al minimo gli argomenti più tecnici, limitandoci a quanto è necessario per realizzare l'oggetto proposto per stare in un'unica puntata. Ovviamente l'argomento matematico è ben più complesso anche tralasciando i dettagli e si presta ad uno sviluppo più ampio: alcuni altri sviluppi ed una trattazione un po' più complessa saranno oggetto di altre puntate

assieme a possibili circuiti complementari.

IL MICROCONTROLORE UTILIZZATO

Il microcontrollore utilizzato è un Freescale 9S08SH8CPJ a 20 piedini di tipo DIP (Dual-in Line Plastic) a 8 bit, con 8kB di memoria flash di programma e 512B di RAM, alimentazione a 5V e frequenza massima del clock istruzioni pari a 20MHz. Tra le altre periferiche incorpora un ADC converter a 12 canali, 10bit con ciclo di campionamento minimo pari a 2,5 μ s. Poiché il μ C utilizzato non è il solito PIC di MICROCHIP ma un altro dispositivo molto usato industrialmente - ma meno conosciuto al pubblico hobbistico - è opportuno acquisire delle informazioni di base su questi dispositivi e sulle tecniche di

programmazione in-circuit della loro memoria flash; per evitare di appesantire questo articolo si rimanda chi fosse interessato al precedente progetto dell'Analizzatore di Rete dello stesso autore, presentato sui numeri 158, 159 e 160 di EI nel luglio-ottobre del 2012. In questo caso si utilizzava un processore Freescale simile, un MC9S08SE8CWL a 28 pin, ma meno veloce (10MHz max vs. 20MHz); sono utilizzabili gli stessi strumenti di programmazione descritti in conclusione nella puntata finale dell'ottobre 2012 (EI 160). Al fine di rendere questo progetto alla portata di tutti, si è volutamente utilizzato il più piccolo dispositivo in involucro classico DIP disponibile con la massima velocità di clock. Sebbene la stessa famiglia preveda un μC identico a 16pin, che sarebbero stati sufficienti, quest'ultimo è solo in involucro SMD: si è adottato quindi il dispositivo a 20pin che è montabile su zoccolo. Questa caratteristica lo rende particolarmente semplice da programmare esternamente e da utilizzare in assenza di ausili di programmazione in loco. Il programma sorgente, ampiamente commentato, è disponibile per il download sul sito della rivista.

LO SCHEMA

Arriviamo così allo schema. Di seguito lo esaminiamo nelle poche parti che lo costituiscono. Il microcontrollore è direttamente interfacciato al display LCD tramite un bus dati a 4 bit e 3 linee di controllo. Una quarta linea su quest'ultimo regola il contrasto del display. La corrente di retroilluminazione (20mA tipici) è ricavata dall'alimentazione principale a 12V tramite una resistenza di caduta e limitazione (R02) da $\frac{1}{4}$ W. La tensione di alimentazione

di tutto il circuito è regolata da un economico stabilizzatore standard a 5V. Il consumo di tutto l'apparato, al netto della corrente di retroilluminazione del display, è circa 15 mA.

Il segnale audio è applicabile sia sul connettore principale di alimentazione e I/O, sia al jack stereo J1. Da qui, allo stesso modo per il canale Destro e quello Sinistro, è inviato a un trimmer di regolazione con campo di variabilità tra 1 e 10 e quindi a un preamplificatore e filtro Passa-Basso costituito da due amplificatori operazionali all'interno di un unico chip quadruplo. Questo circuito amplifica per 11,7 volte il segnale ad esso applicato e lo filtra a poco più di 15kHz @ -3dB con pendenza di 60dB/decade. I due condensatori di ingresso C6 e C7 accoppiano il segnale in alternata all'ingresso del primo amplificatore operazionale polarizzato in corrente continua a metà della tensione di alimentazione (+5V) e costituiscono un blando filtro Passa-Alto con $F_t(-3dB) = 125Hz$. L'uscita dei filtri di canale è applicata direttamente a due dei canali dell'ADC del

microcontrollore.

Considerando che la dinamica dell'ADC è pari alla tensione di alimentazione (5V), il segnale d'ingresso a piena dinamica può essere regolato tra 430 e 4700mVpp (152-1662mVeff) con i componenti nominali a schema, un campo di ingresso valido per il collegamento a un'ampia gamma di dispositivi audio: dall'uscita lineare di un sistema HiFi, a quella di un Ipod o di un PC. Se necessario, riducendo il valore di R7/8 il guadagno può essere facilmente aumentato anche di parecchio.

Il connettore di programmazione BD1 permette la programmazione e il debug "in circuit" del μC mentre al connettore BD2 può essere collegato un commutatore di programma o semplicemente il pulsante SW1 con la stessa funzione.

IL FILTRO DI CANALE

Chiunque abbia studiato teoria del campionamento sa bene che il segnale da campionare deve essere limitato in frequenza a (poco) meno della metà della frequenza di campionamento

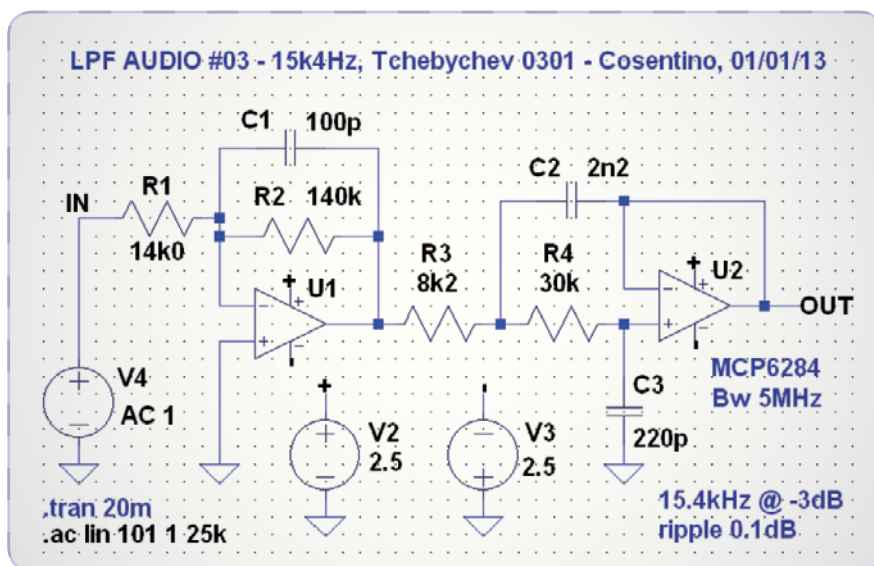


Fig. 1 - Filtro PB, schema di simulazione.

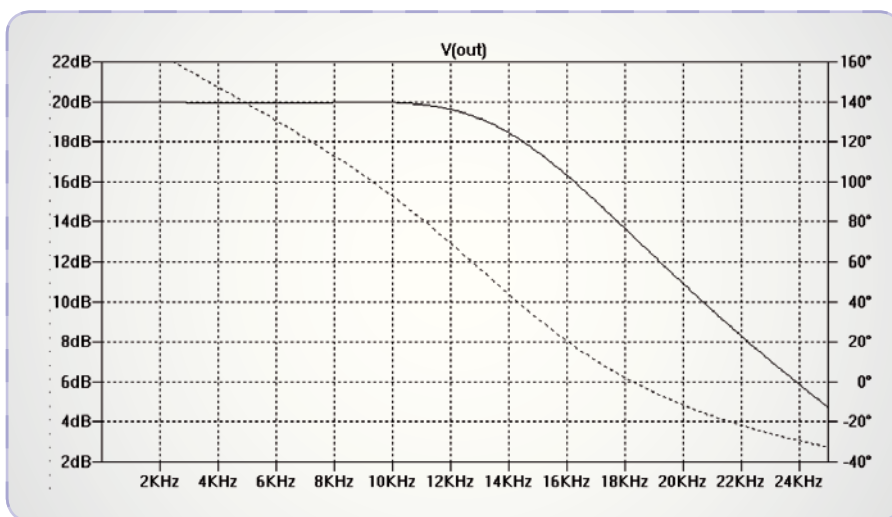


Fig. 2 - Filtro PB, funzione di trasferimento.

(teorema di Shannon), pena l'insorgere di artefatti dovuti alla conversione in banda utile delle frequenze superiori. Più è ripida la pendenza del filtro Passa-Basso dedicato a questo scopo, maggiore è la vicinanza con cui il segnale utile di ingresso può arrivare a questa frequenza teorica massima. Nel nostro caso la frequenza di campionamento è impostata a 32kHz e la banda superiore del segnale utile in ingresso deve essere limitata tra poco più di 10kHz e poco meno di 16kHz. Sebbene il filtro che utilizziamo sia un semplice filtro di Tchebichev di 3° ordine, la frequenza di taglio è stata posta abbastanza vicino a quella massima ed esattamente a 15,4kHz a -3dB. Nel nostro caso è possibile perché la lieve tracimazione delle frequenze oltre i 16kHz avviene comunque solo in prossimità dell'ultima banda (ottava) di frequenza e non costituisce difetto particolare per il nostro circuito. Appena oltre, il segnale viene attenuato ragionevolmente e non costituisce problema per le bande inferiori.

Il filtro da noi utilizzato è facilmente simulabile a scopo di studio con un semplice simulatore pSpice reperibile gratuitamente in rete, l'ottimo LTSpice

di Linear Technology. In Fig. 1 lo schema e in Fig. 2 la funzione di trasferimento ricavata, che è perfettamente congruente con quella reale, a patto di utilizzare componenti di buona qualità e ragionevole tolleranza. Il file in formato LTSpice (LPF AUDIO #03 - 15kHz Tchев 0301.asc) è disponibile per il download dal sito della rivista assieme agli altri file collegati all'articolo: può essere interessante confrontarne le prestazioni con altri simili. Si noti l'eccellente piattezza in

banda ottenuta con componenti sostanzialmente standard. C'è poco da dire sui componenti che non sia già nella descrizione. La tolleranza suggerita all'1% per le resistenze determina la precisione dei filtri assieme ai condensatori. Se l'1% non è reperibile va bene anche il 5%. Un paio di valori sono però della serie E24 e quindi un po' meno reperibili degli altri in serie E12. R05 ed R06 devono essere uguali tra di loro; nello specifico possono essere preselezionate opportunamente se al 5%: il valore assoluto è meno importante dell'accoppiamento tra di esse. L'Op-amp è un componente con dinamica di I/O che include la tensione di alimentazione, banda passante di 5MHz e basso consumo. Il display è un componente standard, il colore suggerito non è fondamentale (vanno bene anche i comuni giallo/verdi) è però importante la bassa corrente di retroilluminazione (20mA tipici) perché quelli più comuni consumano molto di più (250-320mA) e la resistenza di limitazione corrispondente

Fig. 3

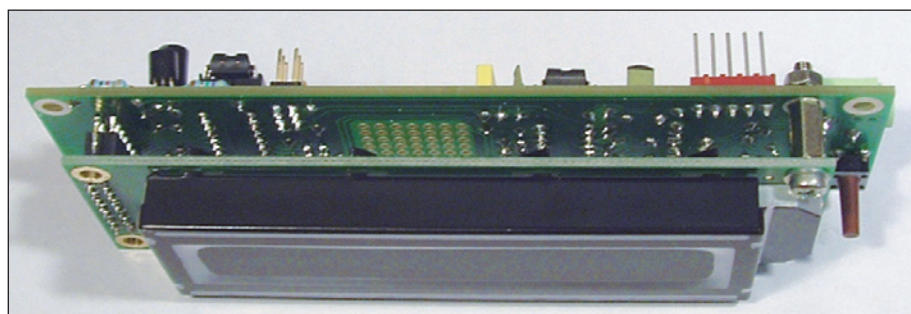
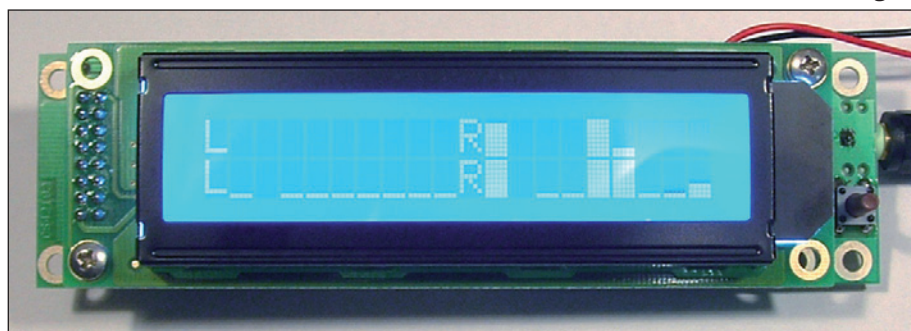


Fig. 4

Come programmare i microcontrollori Freescale

(R02) dissiperebbe (e scalderebbe) moltissimo, anche un paio di watt, se modificata per questa corrente.

I due condensatori di ingresso C6 e C7 sono in poliestere non polarizzati, sebbene lo schema li prevedesse inizialmente polarizzati, al tantalio, di capacità superiore. Gli altri condensatori polarizzati non sono molto critici: sono preferiti quelli al tantalio per le capacità inferiori soprattutto in prossimità di IC1: a parte questo, gli altri potrebbero essere di tipo convenzionale.

IL CIRCUITO STAMPATO E IL MONTAGGIO

Il circuito stampato è solo di poco più largo del display e alto quanto quest'ultimo per permettere il montaggio a sandwich o "piggy-back" dell'uno sull'altro, per un successivo fissaggio sul pannello frontale di un contenitore. Data la semplicità del circuito, questo è stato realizzato con componentistica tradizionale, i circuiti integrati - montati su zoccolo - su circuito stampato monofaccia, facile da produrre. I master possono essere scaricati come al solito dal sito della rivista. Le dimensioni del circuito sono 36,8x132mm. All'interno del circuito stampato, restando disponibile un po' di spazio senza componenti e collegamenti, è stata realizzata una mini area "millefori" sulla quale è possibile montare qualche piccola circuiteria accessoria.

Il montaggio è alla portata di tutti e non vale la pena soffermarsi più di tanto. I quattro ponticelli in filo stagnato con piegatura a passo 7,5mm sono ricavati da altrettanti reofori tagliati in eccesso dalle resistenze. Attenzione alla polarità dei condensatori elettrolitici ed al tantalio. L'unica difficoltà consiste nel montaggio

Sono diverse le interfacce di programmazione della famiglia di microcontrollori Freescale a 8 bit che è possibile utilizzare per lo sviluppo e per le prove. Alcune sono di tipo industriale, altre contenute in schede dimostrative, altre ancora nate da progetti amatoriali (il cosiddetto progetto "Open BDM"). Di seguito illustreremo brevemente ciascuno dei tipi citati, il cui costo è ragionevolmente alla portata anche dell'hobbista che voglia portare avanti qualche esperimento.

La prima interfaccia BDM è quella proposta dallo sviluppatore ufficiale che affianca Freescale a questo scopo: la P&E. Il modello più economico è l'USB-ML-12 rev.C Multilink Adapter, che può essere acquistato in diversi siti in Italia ed all'estero ad un prezzo compreso tra i 75 e i 90 € + IVA. Per questo importo o poco più, le spese di spedizione sono spesso gratuite (anche dagli U.S.A.) ma è bene controllare e comparare le offerte. Il dispositivo in questione è altamente professionale e valido per tutta la famiglia Freescale 9S08 e l'economica RS08 a 8 bit, nonché la HC(S)12 a 16 bit e la recente ColdFire V1 a 32 bit, hardware compatibile con alcuni microcontrollori ad 8 bit. Per una descrizione dettagliata si veda il sito Freescale (www.freescale.com) o quello P&E (www.pemicro.com).

La seconda interfaccia è quella contenuta nella scheda dimostrativa e di valutazione del microcontrollore 9S08SE8, quello utilizzato nel nostro circuito. Infatti questa scheda, costruita da AXIOM per Freescale, oltre a permettere di sperimentare il μC in questione riguardo a molte delle sue funzioni hardware più generiche (linee di I/O, pilotaggio di LED, ingresso da switch e da ADC, porta RS232) con semplici programmi, può essere facilmente usata per programmare e "debuggare" lo stesso microcontrollore montato su schede esterne. Una buona scelta, quindi, per chi volesse fare esperimenti hardware e software prima di montare un circuito applicativo. Anche il prezzo

è competitivo; infatti, nonostante l'uso non sia limitato alla semplice interfaccia, questa scheda costa meno della proposta P&E: la si può trovare intorno ai 60 € + IVA negli stessi posti e con le stesse modalità della USB-ML-12 rev.C Multilink Adapter.

L'unico neo è che, sebbene non ci siano reali limitazioni hardware, l'uso per la programmazione esterna potrebbe essere limitato via software alla sola famiglia 9S08SE.

La scheda si chiama DEMO9S08SE8 e se ne può facilmente scaricare la documentazione, compresi schemi e guide, sul sito Internet della Freescale.

L'ultima scheda è figlia del progetto open-source USBDM, pur essendo montata industrialmente dalla Wytek. Si chiama USBDM-1-JS16 ed è sostanzialmente equivalente all'interfaccia della P&E,

pur costando circa 1/5. Per un pezzo singolo la spedizione dagli USA costa un po' più della scheda, ma il totale è comunque estremamente conveniente.

Purtroppo la prova sul campo non è stata altrettanto soddisfacente: l'uso di driver non del tutto standard e una certa macchinosità di installazione non hanno consentito di conseguire un funzionamento corretto, per cui non ci sentiamo di consigliarla come primo dispositivo a neofiti che non abbiano mezzi per verificarne il buon funzionamento e non godano di una certa abilità nello "smanettare" con driver e configurazioni.

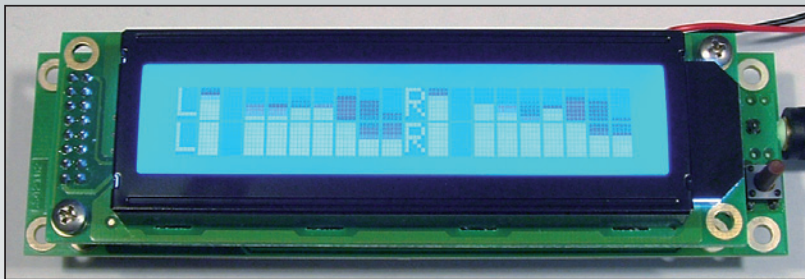
Informazioni su questa scheda e su numerosi altri prodotti Wytek a basso prezzo basati sulla famiglia 9S08, possono essere reperite sul sito www.evbplus.com, ove potete anche acquistare la scheda di programmazione. Tutte le schede descritte necessitano dell'IDE CodeWarrior V6.3 o superiore, "special edition".

Per le prime due schede descritte, questo software è in genere fornito su disco, mentre per la terza va scaricato dal sito della Freescale.

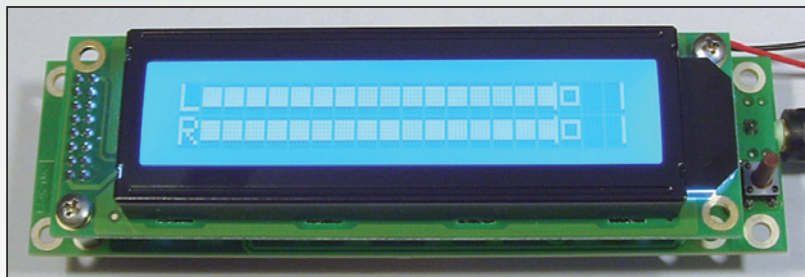


Fig. 5

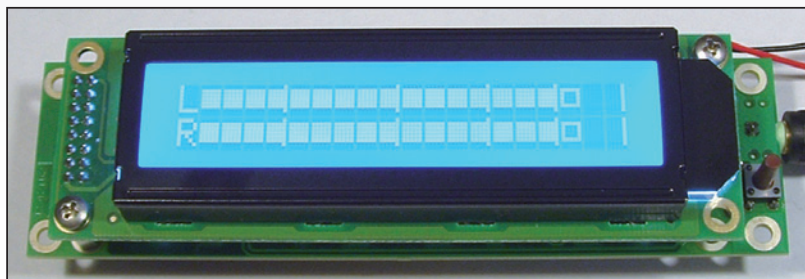
- 1) FFT stereo su 7 ottave tra 250 e 15000 Hz.



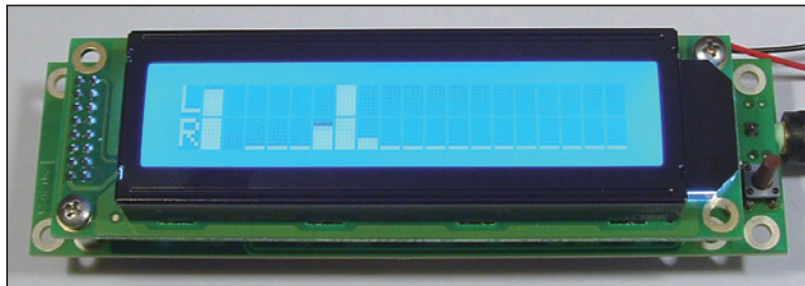
- 2) VuMeter di picco, scala semilogaritmica.
Mark a 0dB (fondoscala) e -3dB.



- 3) VuMeter di picco, scala semilogaritmica.
Mark a 0dB, -3, -6, -10 e -20dB.



- 4) FFT mono (d+s) su 17 barre tra 250 e 16000 Hz.



della strip maschio P2, montata dall'alto e saldata dal basso sul lato rame: questa strip andrà infatti inserita nella corrispondente strip femmina montata in corrispondenza sul retro dell'LCD. Eseguire la saldatura tra il corpo di ciascun pin e la piazzola

corrispondente con poco stagno (per evitare la risalita sul pin), uniformemente senza lasciare bolle tra il pin e la piazzola e velocemente per evitare di deformare la plastica che sorregge il pin dall'altro lato o di scollare la pista. È indispensabile un salda-

tore con punta molto sottile, alla temperatura adeguata, da usare con filo di stagno sottile per SMD (0,5mm). Il pulsante SW1 è l'unico componente che va saldato se necessario (vedi funzione successiva) dal lato rame, per potere essere generalmente accessibile da pannello frontale.

Le illustrazioni di Fig. 3 e Fig. 4 mostrano il montaggio dei componenti e l'assemblaggio del display sul circuito stampato, tenuto meccanicamente tramite i connettori di collegamento a 16 poli maschio e femmina (quello femmina sul retro del display) e da 2 o 3 colonnine distanziali e di rinforzo meccanico da 10mm.

COLLEGAMENTI ED UTILIZZO

I collegamenti all'alimentazione e al segnale di ingresso sono immediati: la massa dell'alimentazione ed il positivo vanno rispettivamente al piedino 5 e 4 di P1. Attenzione a non sbagliare: non ci sono diodi di protezione! L'alimentazione deve essere in corrente continua, ma non serve sia particolarmente stabilizzata: va bene anche quella ottenuta con un trasformatore/ponte diodi/condensatore di filtro, mediamente sui 12V $\pm 15\%$. Tensioni minori o maggiori, comprese tra 7,5 e 20V, possono essere accettate pur di variare in accordo R02 per fornire una ventina di milliampere alla retroilluminazione del display. Il segnale audio D/S può essere collegato alternativamente a P1 (piedini 3/1, massa sul 2) o a J1 tramite jack da 3,5mm.

Il pulsante montato sul retro del circuito stampato e azionabile dal lato del display permette di impostare in successione 4 differenti modi di funzionamento che descriveremo di seguito; all'atto dell'accensione il circuito partirà sempre dal 1° di questi modi.

Fig. 6

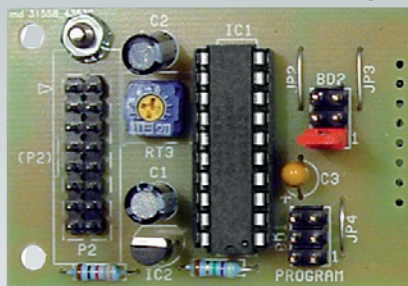


Tabella 1

Pin 3-4	Pin 5-6	Funzionamento
OFF	OFF	Modo 1 (FFT stereo)
OFF	ON	VuMeter senza riferim.
ON	OFF	VuMeter con riferimenti
ON	ON	FFT mono 17 bande

In alternativa al pulsante è possibile montare dei ponticelli fissi su BD2 e/o un commutatore 2 vie/4 posizioni che permetterà di impostare qualsiasi degli altri modi, sia come default, sia durante il funzionamento. I 4 modi di funzionamento sono visibili in Fig. 5.

Nel funzionamento come VuMeter la barra ha un andamento "semilogaritmico". Ogni carattere del display è diviso in due barrette verticali per un totale di 40 punti di misura: le prime 2 posizioni sono occupate però dalle lettere L (Sinistro) e R (Destro). Tra 0 e -12dB ogni carattere vale 1dB, per cui ogni punto di misura vale 1/2 dB. Tra -12 e -20dB, ogni punto vale 1dB, successivamente, sotto i -20dB, ogni punto di misura vale 2,5dB (5dB per carattere). Il segnale comincia a essere visualizzato quando maggiore di -30dB. Un carattere speciale, un quadratino aperto, marca la posizione del picco massimo nell'ultimo secondo.

Nel funzionamento come Analizzatore di Spettro, sono rilevate 63 bande di frequenza, ciascuna ampia 250Hz tra 250Hz e 16kHz. La prima banda, fino a poco meno di 250Hz è poco significativa e si è deciso di scartarla. Sul display FFT queste bande sono raggruppate in quantità via via crescenti per raggruppare l'energia in modo logaritmico, secondo la naturale sensibilità dell'orecchio umano. Le prime 2 bande, 250 e 500 Hz sono barre singole, la terza barra raggruppa 750 e 1000Hz, abbiamo poi 1250-2000Hz, 2k-4kHz, 4-8kHz, 8-16kHz. Manca purtroppo la rappresentazione dell'energia poco sotto i 250Hz ma non è possibile fare di meglio con i 128 punti di campionamento per ciascuno dei due canali. Nella rappresentazione FFT a 17 barre, le singole bande di ciascuno dei 2 canali vengono sommate

in potenza e sono rappresentate insieme come un segnale monofonico. Queste bande vengono poi raggruppate sempre in maniera crescente, ma meno densamente, in modo visualizzare l'andamento con la frequenza in modo "pseudo" logaritmico. In questo caso le prime 4 barre sono singole, l'ultima raggruppa ben 10 sotto-bande (13,5-16kHz). In ambedue i quadri FFT, la prima barra subito dopo la/le lettere di canale sono proporzionali al picco istantaneo del segnale. Sebbene durante il funzionamento il segnale appaia sostanzialmente continuo, non è esattamente così: il display viene rinfrescato all'incirca 40 volte al secondo, con un periodo di misura pari o poco superiore ai 25ms. Questo tempo è così ripartito:

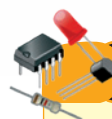
- 4ms: campionamento e preprocesso di 128+128 campioni di segnale destro e sinistro
- 10,6ms: FFT canale sinistro
- 10,8ms: scambio canale di processo ed FFT canale destro
- 1,0ms: raggruppamento e processo display
- 1,6ms: trasferimento dati al display

Nel funzionamento come VuMeter, il campionamento è molto

più costante durante il periodo di quadro e vengono effettuati 6 cicli da 128+128 campioni s+d per un periodo totale di 24ms, alla fine del quale vengono estratti il picco del segnale per ciascun canale. In questo caso il processo del segnale è inferiore all'1% del periodo di quadro mentre l'aggiornamento del display incide ancora per circa il 6%. Non è possibile ridurre quest'ultimo periodo per la relativa lentezza del display: il μ C potrebbe andare anche 10 volte più veloce!

PONTICELLI DI PREDISPOSIZIONE

Per cambiare il modo di funzionamento tramite pulsante non occorre inserire alcun ponticello. Se invece si preferisce un modo di funzionamento variabile tramite commutatore, occorre inserire un ponticello di cortocircuito fisso tra i piedini 1 e 2 di BD2 (vedi ponticello rosso sul particolare di Fig. 6). le altre 2 coppie di piedini su BD2 determinano il modo di funzionamento secondo la Tabella 1. ■



per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono facilmente reperibili in commercio. Il master del circuito stampato ed il firmware col quale è programmato il micro sono scaricabili gratuitamente dal sito della rivista (www.elettronica.in.it). Il microcontrollore Freescale è anche disponibile già programmato (Cod. MF1056) al prezzo di 10,00 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>

OpenWheels

MONTIAMO L'ELETTRONICA

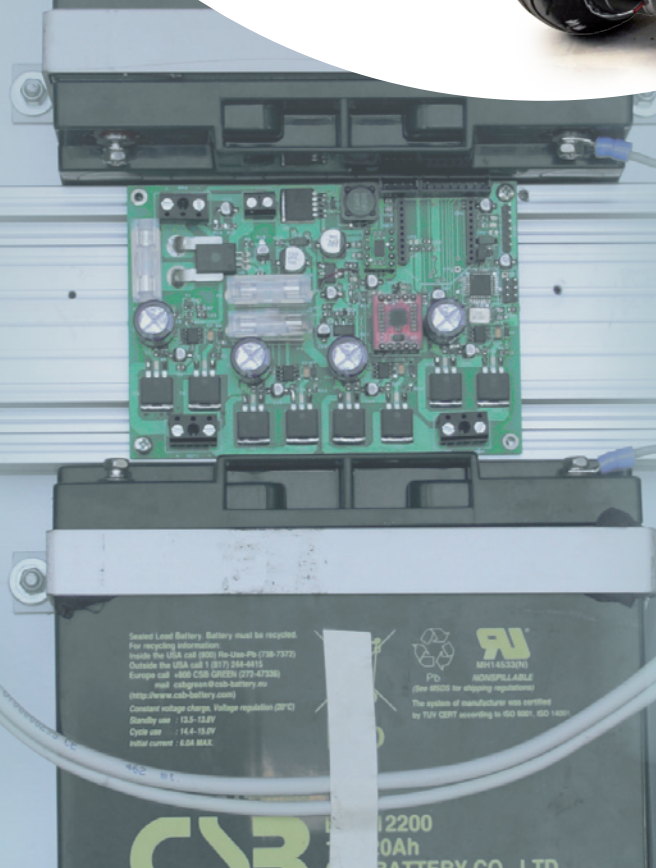
Dopo la descrizione della struttura meccanica, del principio di funzionamento e dello schema della scheda di controllo, in questa puntata ci occupiamo del cablaggio e del collaudo di quest'ultima.

di MIRCO SEGATELLO

Nella puntata precedente abbiamo descritto il principio di funzionamento del nostro veicolo personale soffermandoci sulle soluzioni studiate e poi adottate in sede di progettazione dello schema elettrico definitivo. Abbiamo descritto dettagliatamente tale schema analizzando le varie sezioni che lo compongono, in particolare la sezione di potenza (la più critica sotto molti punti di vista) ed il "cervello" - ovvero il microcontrollore - che, come il cervello umano, invia gli ordini ai dispositivi che controllano l'avanzamento (due motori) sulla base dei controlli impartiti tramite il manubrio nonché delle informazioni fornite dai vari sensori, principalmente dall'accelerometro. Abbiamo anche visto come il "cervello" del nostro OpenWheels sia un microcontrollore Atmel programmato con l'IDE di Arduino, precisamente un ATMEGA 328A-U. A questo proposito, come abbiamo già accennato

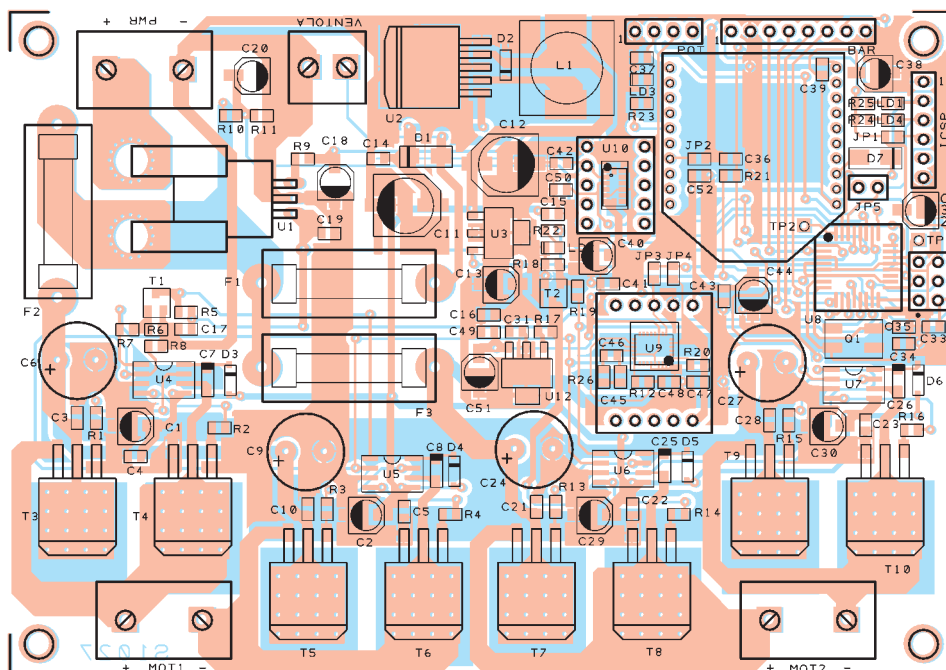
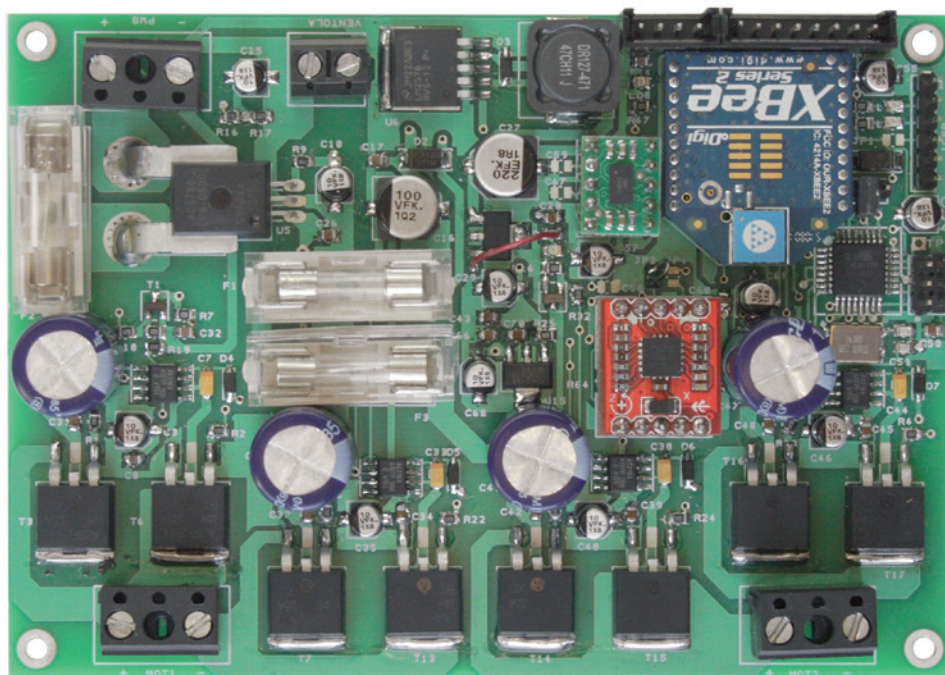
nella precedente puntata, qualche dubbio iniziale ha fatto capolino nella nostra mente considerando che la scheda Arduino non è proprio il massimo in fatto di velocità di elaborazione ed il calcolo matematico complesso non è il suo forte ma, con un sapiente utilizzo delle risorse disponibili, siamo riusciti ad ottenere ottimi risultati.

Per fugare sin dall'inizio qualsiasi dubbio, abbiamo realizzato una prima versione dell'elettronica



Elenco Componenti:

R1 ÷ R4: 4,7 ohm (0805)
 R5, R6: 1 kohm (0805)
 R7: 4,7 kohm (0805)
 R8, R9: 2,2 kohm (0805)
 R10: 47 kohm (0805)
 R11, R13 ÷ R16: 4,7 kohm (0805)
 R12: 33 kohm (0805)
 R17, R18: 1 kohm (0805)
 R19: 4,7 kohm (0805)
 R20: 33 kohm (0805)
 R21: 10 kohm (0805)
 R22 ÷ R25: 1 kohm (0805)
 R26: 10 kohm (0805)
 C1, C2: 10 μ F 35 VL elettrolitico (B)
 C3 ÷ C5: 100 nF ceramico (0805)
 C6: 1000 μ F 35 VL elettrolitico
 C7, C8: 10 μ F 25 VL tantalio (A)
 C9: 1000 μ F 35 VL elettrolitico
 C10: 100 nF ceramico (0805)
 C11: 100 μ F 35 VL elettrolitico (F)
 C12: 220 μ F 25 VL elettrolitico (F)
 C13: 10 μ F 35 VL elettrolitico (B)
 C14 ÷ C17: 100 nF ceramico (0805)
 C18: 10 μ F 50 VL elettrolitico (B)
 C19, C21 ÷ C23: 100 nF ceramico (0805)
 C20: 10 μ F 35 VL elettrolitico (B)
 C24, C27: 1000 μ F 35 VL elettrolitico
 C25, C26: 10 μ F 25 VL tantalio (A)
 C28: 100 nF ceramico (0805)
 C29, C30: 10 μ F 35 VL elettrolitico (B)
 C31: 100 nF ceramico (0805)
 C32: 10 μ F 35 VL elettrolitico (B)
 C33, C36: 100 nF ceramico (0805)
 C34, C35: 22 pF ceramico (0805)
 C37: 220 nF ceramico (0805)
 C38, C40: 10 μ F 35 VL elettrolitico (B)
 C39, C41: 100 nF ceramico (0805)
 C42: 3,3 nF ceramico (0805)
 C43, C47 ÷ C49: 100 nF ceramico (0805)
 C44: 10 μ F 35 VL elettrolitico (B)
 C45: 470 nF ceramico (0805)
 C46: 10 nF ceramico (0805)
 C50: 3,3 nF ceramico (0805)
 C51: 10 μ F 35 VL elettrolitico (B)
 C52: 100 nF ceramico (0805)
 D1: GF1M
 D2 ÷ D6: BAT42W
 D7: GF1M
 T1, T2: BC817
 T3 ÷ T10: AU1RF2804S
 Q1: Quarzo 16 MHz
 U1: ACS758LCB-050U-PFF
 U2: LM2576HVS-12/NOPB
 U3: LM2937IMP-5.0/NOPB
 U4 ÷ U7: IRS2184S
 U8: ATMEGA328-AU
 U9: LPR403 o WRL-10415
 U10: MMA7361
 U11: XBEE1MWCHIP
 U12: TC1262-3.3VDB



L1: Induttanza 470 μ H 850 mA
 F1: Fusibile 1A
 F2, F3: Fusibile 10A
 LD1, LD2: LED verde (0805)
 LD3, LD4: LED rosso (0805)

Varie:

- Porta fusibile da CS
 - Morsettiera 2 poli passo 10 mm (3 pz.)
 - Morsettiera 2 poli passo 5 mm (1 pz.)

- Strip maschio 2 poli
 - Strip maschio 3 poli (2 pz.)
 - Strip maschio 4 poli
 - Strip maschio 6 poli
 - Strip maschio 8 poli
 - Strip femmina 5 poli (4 pz.)
 - Strip femmina 10 poli passo 2 mm (2 pz.)
 - Jumper
 - Circuito stampato

di OpenWheels con una board Arduino Uno, una breadboard filata e tutti gli altri componenti collegati in maniera provvisoria per verificarne le prestazioni (e nel caso sostituirli facilmente); pur con collegamenti precari, questa prima soluzione ci ha permesso di acquisire una certa esperienza e ci ha consentito di verificare le prestazioni delle soluzioni adottate. Dalle numerose prove ed esperienze sul campo è scaturito lo schema elettrico definitivo descritto nella puntata precedente ed il PCB presentato in queste pagine. Questo mese ci occuperemo quindi della costruzione, della programmazione e del collaudo di tale scheda.

La basetta da noi messa a punto è molto compatta (misura appena 90 x 125 mm) tenendo conto che accoglie tutti i componenti, compresi gli elementi di potenza. Ovviamente si tratta di una basetta a doppia faccia con fori metallizzati la cui costruzione richiede un'attrezzatura (bagni galvanici) che un hobbista sicuramente non possiede. Ciò, tuttavia, non rappresenta un problema dal momento che il PCB, realizzato in maniera professionale, è disponibile presso Futura Elettronica (www.futurashop.it).

Per il montaggio dei vari componenti - inserimento e saldatura - valgono le solite raccomandazioni relative alla sequenza (prima i componenti a più basso profilo e, poi, via via tutti gli altri) e sull'impiego del saldatore (bisogna evitare di surriscaldare i componenti più delicati, tipicamente i semiconduttori).

A montaggio ultimato, prima di qualsiasi altra operazione, è necessario eseguire un controllo visivo dello stato delle saldature e della correttezza dei componenti montati.

Durante la prima fase del collaudo non inseriremo né il modulo



Fig. 1 - Come caricare il bootloader se viene utilizzato un micro Atmel vergine. In questo caso è necessario ricorrere a una board Arduino da usare come programmatore.

Xbee né i fusibili della sezione di potenza e neppure l'accelerometro ed il giroscopio; inseriremo invece il fusibile F1 da 1A per poter alimentare la sezione di controllo. Se utilizzate un microcontrollore vergine, ovvero senza bootloader, dovrete provvedere a caricarlo; la procedura è ben documentata in internet ma, per completezza, la riassumiamo di seguito. Per caricare il bootloader in un Atmel vergine vi servirà una scheda Arduino da utilizzare come programmatore

programming interface) del tutto uguale a quello presente sulla scheda Arduino: non dovremo fare altro che collegare questi due connettori, compreso il segnale di reset per l'Atmel di Openwheels (Fig. 1).

Potete crearvi un cavo ad hoc oppure usare dei cavetti intestati come quello *cod. Jumper 50* di Futura Elettronica (Fig. 2 e Fig. 3). Dopo aver realizzato il cablaggio elettrico potete iniziare la procedura per il caricamento del bootloader tramite l'IDE di Arduino. Per prima cosa aprite l'esempio denominato *ArduinoISP* e caricate-lo, con la normale procedura, sulla scheda Arduino: questo permetterà di trasformare Arduino in un

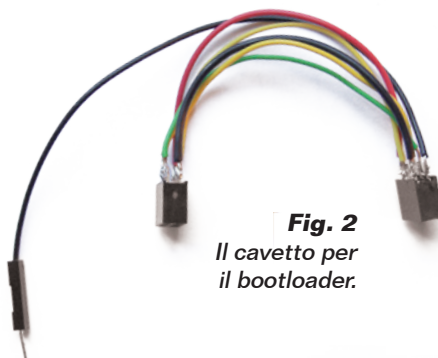


Fig. 2
Il cavetto per il bootloader.

nonché alcuni cavetti intestati. Non è necessario alimentare la scheda OpenWheels in quanto il circuito ricaverà l'alimentazione dalla scheda Arduino.

Sulla nostra basetta è presente un connettore denominato SPI (serial

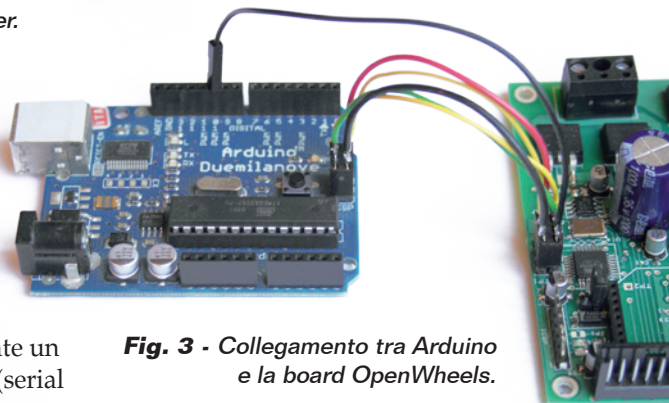


Fig. 3 - Collegamento tra Arduino e la board OpenWheels.

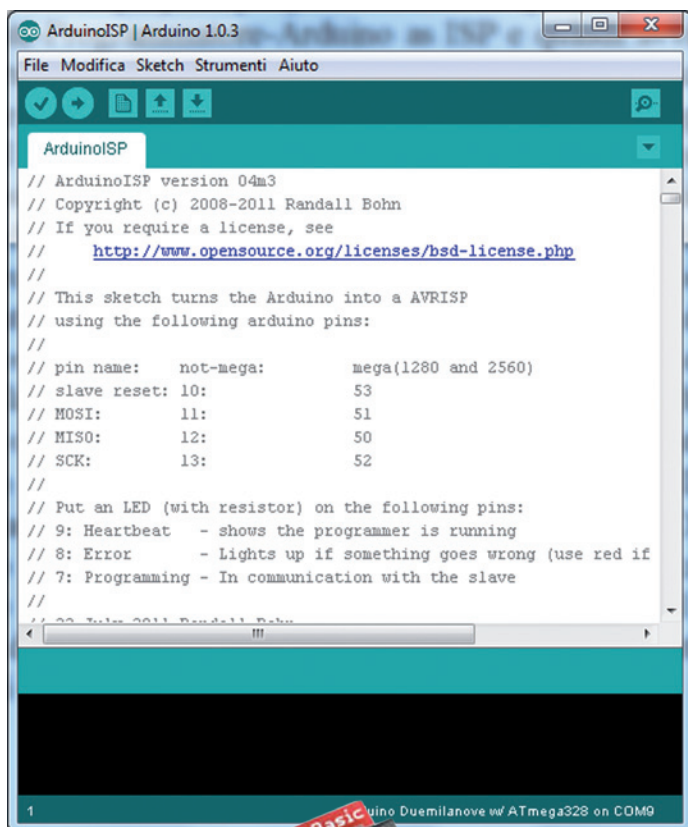


Fig. 4
Adattatore
USB/seriale.

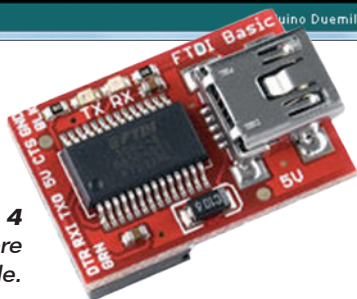


Fig. 5
Selezioniamo
dall'IDE
ArduinoISP.

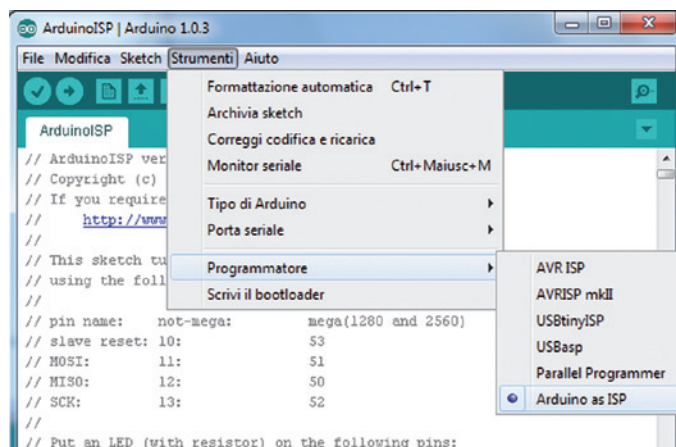


Fig. 6 - Scegliamo la risorsa ArduinoISP.

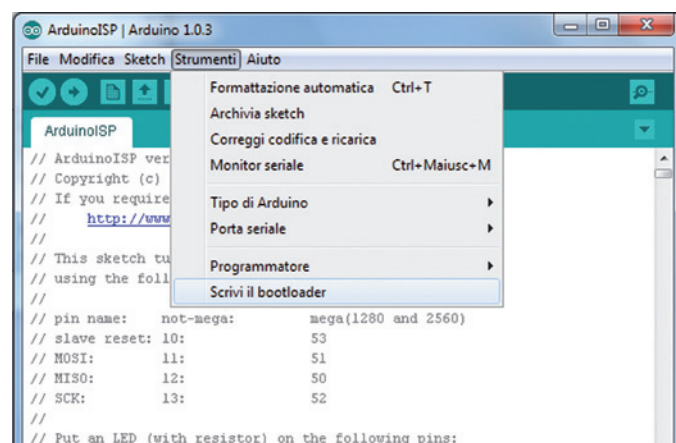


Fig. 7 - Scarichiamo il bootloader.

vero e proprio programmatore. A questo punto, sempre dall'IDE di Arduino, selezionate *Strumenti-Programmatore-Arduino as ISP* e quindi avviate la procedura per il bootloader selezionando *Strumenti-Scrivi il Bootloader*. Durante questa fase assicuratevi di aver selezionato come *Arduino board* la versione Duemilanove. La procedura richiede diversi secondi durante i quali i LED LD1 e LD4 della scheda OpenWheels dovranno lampeggiare. L'avvio della procedura di caricamento del bootloader è subordinata dalla lettura di un codice identificativo del chip da programmare, nel nostro caso utilizziamo una versione SMD che presenta un identificativo differente dal chip Atmel di solito installato sulla scheda Arduino, è

probabile quindi, si verifichi un errore del tipo: *Expected signature for ATMEGA328P is 1E 95 14*. Se si dovesse presentare questo errore dovrete cercare il file *avrdude.conf* nel percorso *arduinoxxx-hardware-tool-avr* ed aprirlo con un editor di testo; semmai, per ogni evenienza fate una copia di riserva. Cercate la riga con indicato *Atmega328P* e la riga che riporta il codice di *signature= 0x1e 0x95 0x0F*. Modificate l'ultimo parametro in modo che la riga diventi: *signature= 0x1e 0x95 0x14*. Salvate il file e riavviate l'IDE di Arduino, quindi ripetete la procedura di caricamento del bootloader. Fatto questo la scheda OpenWheels è a tutti gli effetti una scheda Arduino compatibile Duemilanove e le successive operazioni di programmazione avverranno tramite un convertitore USB/seriale basato sul chip FDTI come nella

versione Duemilanove di Arduino. Per le nostre operazioni abbiamo utilizzato il convertitore USB/seriale di Sparkfun con codice FDTI Basic impostato come di default per una tensione operativa di 5V e disponibile presso Futura Elettronica con codice 7300 FDTI5V (Fig. 4, Fig. 5, Fig. 6 e Fig. 7). Un primo collaudo al banco prevede semplicemente l'alimentazione della scheda tramite una sorgente esterna a 24Vdc, magari facendo ricorso ad un alimentatore stabilizzato con la funzione di limitazione in corrente. Con un semplice multimetro verificate che le tensioni di alimentazione relative ai +5V e +3,3V siano corrette; il LED LD2 di accensione deve risultare acceso. A questo punto siete pronti per cablare i vari componenti elettrici e rendere finalmente operativo



Fig. 8
Collegamento al potenziometro.

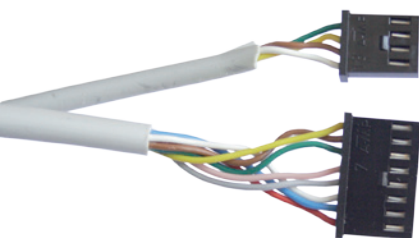


Fig. 9
Connettori per scheda Openwheels.

OpenWheels; ci sarà un po' di lavoro da fare e un po' di caveti da preparare.

Il primo step consiste nel cablare il potenziometro dello sterzo ed a questo proposito utilizzeremo del cavo commerciale 4x0,25 a quattro fili, anche se a noi ne servono solamente tre. Saldare i tre fili ai tre terminali del potenziometro ed all'altro estremo saldare l'apposito connettore facendo attenzione che al pin 1 venga saldato il conduttore collegato al pin del potenziometro rivolto verso la ruota di sinistra; ovviamente a quello centrale saldate il filo corrispondente al cursore del potenziometro e il terzo filo al terminale restante (Fig. 8). Conviene verificare la bontà del collegamento elettrico con un tester prima di collegare il cavo al relativo connettore presente sulla scheda OpenWheels. Verificate anche che il potenziometro sia centrato, ovvero che tra i pin 1 e 2 del connettore sia presente una resistenza di circa 5 kohm (non serve una grande precisione) così come tra i pin 2 e 3. Passiamo ora la cablaggio del manubrio, per il quale dobbiamo fare uso di un cavo 8x0,25 e, come già fatto precedentemente, dal "lato scheda" saldiamo un connettore ad otto poli, mentre all'altro lato cabliamo il display LCD ed i pulsanti secondo le indicazioni di Fig. 10.

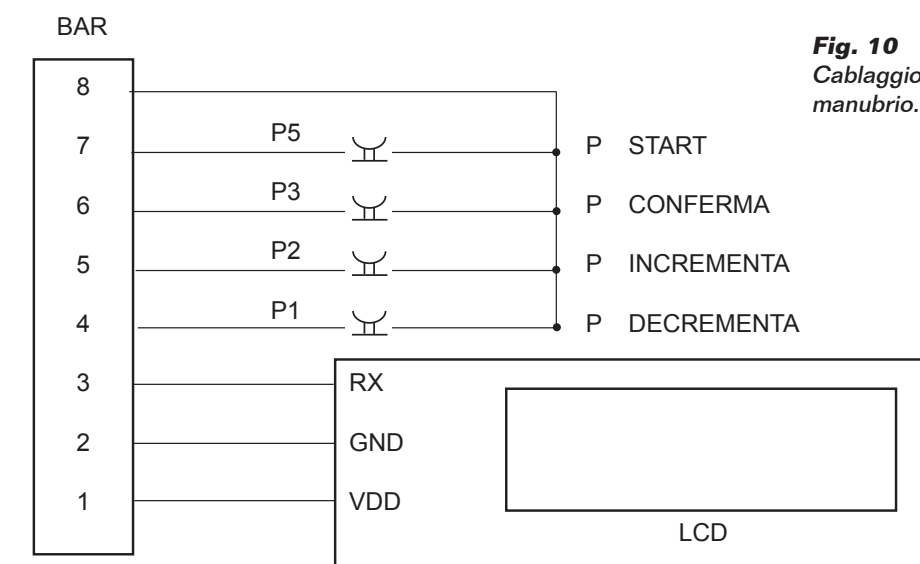


Fig. 10
Cablaggio manubrio.

Sull'impugnatura a portata di pollice destro (o sinistro) andrà posizionato il pulsante di start, mentre conviene montare il display LCD in posizione centrale, in modo che sia ben visibile durante la marcia. I pulsanti per l'impostazione dei parametri possono essere posti vicino al display. Risulta logico posizionare P1 a sinistra (funzione decrementa) e P2 a destra (funzione incrementa) mentre P3 può essere posto indifferentemente al centro oppure alla destra di tutti i pulsanti.

Sulla basetta principale sono presenti quattro fori in corrispondenza dei quattro angoli che, tramite delle viti e dei distanziatori in plastica, permettono il fissaggio sulla traversa principale, quella posta in asse alle ruote. Fate attenzione che la scheda non tocchi la barra di alluminio sottostante, pena un corto sulle piste di rame.

Le due batterie delimitano il posizionamento della basetta ed allo stesso stempo la proteggono da urti accidentali provenienti dal basso (Fig. 14). Particolare attenzione va posta al corretto orientamento della scheda pena un'errata lettura dei sensori; osservate attentamente le foto pubblicate nell'articolo; il connettore MOT1 si deve trovare sul lato destro di OpenWheels. I collegamenti elettrici sono decisa-

mente semplici grazie ai connettori già predisposti sulla scheda. Il primo collegamento da realizzare riguarda l'alimentazione delle batterie; i due elementi debbono essere connessi in serie facendo

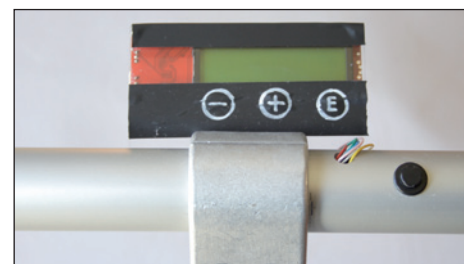


Fig. 11 - Display LCD e pulsante di Start.



Fig. 12 - Posizionamento dell'LCD.

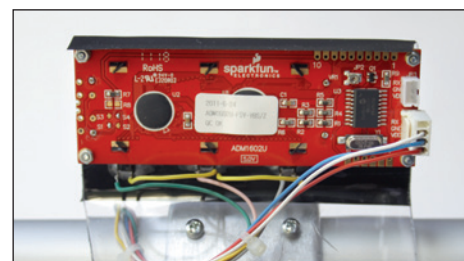


Fig. 13 - Cablaggio LCD e pulsanti.

```

/*
  OpenWheels Test Sensor Program
  by Mirco Segatello

  Hardware OpenWheels 1.1 Ottobre 2012
  board: compatibility to ARDUINO Duemilanove
  compiler: Arduino 1.0.2
  Copyright: Creative Commons BY-NC-SA
  */

#include <SoftwareSerial.h>

//ADC Inputs
#define Acc_ADC_pin 0
#define Gyro_ADC_pin 1
#define Steer_ADC_pin 2
#define VBat_ADC_pin 3
#define IBat_ADC_pin 4

//Driver pin
#define ShutDown_pin 3 // ShutDown for motor driver (LOW=SD)
#define PWMLeft_pin 9 // PWM for motor Left
#define PWMRight_pin 10 // PWM for motor Right

//Digital pin
#define SerLCD_TX 4 // software serial TX for Serial LCD
#define P1_pin 5 // P1 pushbutton + function
#define P2_pin 6 // P2 pushbutton - function
#define P3_pin 7 // P3 pushbutton Enter function
#define PStart_pin 8 // Start Contact

//Battery variables
float VBatLevel;
float VBatScale=0.036;
float IBatLevel;
float IBatScale=0.0538;

//motor variables
int dutyCycleLeft;
int dutyCycleRight;
int stallMotor; // value of PWM for motor stall
float drive=0;
int drivePWM;
int maxPWM;

// Sensor
int Acc_RAW;
int Gyro_RAW;
int Steer_RAW;
int IBat_Zero;

// LCD TX pin (RX not necessary)
SoftwareSerial LCD = SoftwareSerial(0, SerLCD_TX);

void setup() {
  Serial.begin(115200);
  analogReference(EXTERNAL);
  pinMode(ShutDown_pin, OUTPUT);
  digitalWrite(ShutDown_pin, LOW); //ShutDown driver at startUP
  pinMode(P1_pin, INPUT_PULLUP);
  pinMode(P2_pin, INPUT_PULLUP);
  pinMode(P3_pin, INPUT_PULLUP);
  pinMode(PStart_pin, INPUT_PULLUP);

  Serial.println("OpenWheels Test Sensor\n");
  delay(500);
  Serial.println("Init IBat offset...\n");
  initIBat();
  delay(500);
  Serial.println("Init LCD...\n");
  serLCDInit();
  delay(500);
}

void loop() {
  lcdPosition(1,0);
  Steer_RAW = analogRead(Steer_ADC_pin);
  Serial.print("Steer="); // ECO on SerialMonitor
  Serial.print(Steer_RAW);
  LCD.print("Str=");
  if (Steer_RAW<10) LCD.print(" ");
  if (Steer_RAW<100) LCD.print(" ");

```

(Continua)

uso di un conduttore che unisca tra loro il polo positivo di una batteria col negativo dell'altra. Approfittiamo di questo collegamento per inserire un fusibile a protezione delle batterie nel caso di accidentali corto circuiti. In questa prima fase ci occuperemo essenzialmente del test dei sensori pertanto non effettueremo i collegamenti ai motori. Un secondo cablaggio dovrà essere eseguito per portare alimentazione alla scheda; su questo collegamento inseriremo in serie anche l'interruttore generale. Non c'è un posto preciso in cui mettere questo interruttore, però consigliamo sia in una posizione facilmente accessibile e che minimizzi la lunghezza dei fili di collegamento, per ridurre le perdite elettriche sui conduttori. Siccome l'interruttore potrebbe essere inavvertitamente azionato, se volete disattivare completamente OpenWheels, la miglior soluzione è togliere il fusibile sul filo che connette in serie le batterie. Montate l'accelerometro ed il giroscopio, osservando scrupolosamente il verso di inserimento. Sempre senza fusibili sulla parte di potenza, potete alimentare la scheda e verificare che il LED di accensione rimanga correttamente acceso. A questo punto potete caricare lo sketch *OpenWheelsTestSens* che permette di inizializzare il display LCD e testare i sensori.

Aprirete Serial Monitor di Arduino ed impostate una comunicazione a 115 kbps: OpenWheels invierà una serie di dati relativi allo stato dei sensori e nel contempo aggiornerà il display LCD.

Si tratta di dati grezzi, non ancora elaborati, che ci fanno capire se tutto funziona in modo corretto. I dati provengono dal modulo ADC del micro e riguardano essenzialmente - in questa fase - la tensione e la corrente sulla batteria. Verificate che il valore della tensione sulla batteria sia corretta, la misura di

corrente sarà invece praticamente zero. Una verifica dell'accuratezza nella misura della corrente può essere effettuata inserendo un piccolo carico resistivo sul connettore della ventola.

La variabile relativa allo sterzo dovrà fornire un valore intermedio tra 0 e 1024 ovvero circa 512; inclinando il manubrio questo valore varierà proporzionalmente. L'accelerometro a riposo fornisce un valore di circa 1,65V ovvero circa 512 sul display, mentre il giroscopio fornisce a riposo un valore di 1,3V, ovvero circa 380 sul display. Entrambi questi valori variano a seconda dell'inclinazione della scheda (e del veicolo) per cui può essere una buona idea "giocare" con la scheda e vedere come reagiscono i sensori osservando il valore in uscita.

Verificate che i pulsanti funzionino correttamente, premeteli e controllate che sul display compaia l'indicazione relativa. Una prima verifica sommaria del funzionamento dei sensori può essere eseguita inclinando leggermente in avanti e indietro il manubrio, la lettura dell'accelerometro rimane pressoché stabile nella posizione raggiunta mentre il giroscopio varia la sua indicazione solo durante la rotazione del manubrio, riportandosi al valore di default quando l'inclinazione rimane costante. Questo spiega i presupposti descritti nell'articolo precedente quando abbiamo descritto la scelta dei sensori. Mentre il giroscopio fornisce un'indicazione sulla dinamica della rotazione, l'accelerometro fornisce nel lungo periodo un'indicazione dell'inclinazione assoluta di OpenWheels. Un altro punto su cui soffermarsi è la "pulizia" della misura: di solito il giroscopio fornisce una misura molto "pulita" mentre l'accelerometro risente del rumore interno dei suoi componenti ed è sensibile

```

if (Steer_RAW<1000)
  LCD.print(Steer_RAW);
else
  LCD.print("---");

lcdPosition(1,8);
VBatLevel = float(analogRead(VBat_ADC_pin))*VBatScale;
Serial.print("\tV Bat="); // ECO on SerialMonitor
Serial.print(VBatLevel);
LCD.print("Bat=");
LCD.print(VBatLevel);

IBatLevel = float(analogRead(IBat_ADC_pin)-IBat_Zero)*IBatScale;
Serial.print("\tI Bat="); // ECO on SerialMonitor
Serial.print(IBatLevel); // not OffSet calculated

lcdPosition(0,0);
Acc_RAW = analogRead(Acc_ADC_pin);
Serial.print("\tAcc="); // ECO on SerialMonitor
Serial.print(Acc_RAW);
LCD.print("Acc=");
if (Acc_RAW<10) LCD.print(" ");
if (Acc_RAW<100) LCD.print(" ");
if (Acc_RAW<1000)
  LCD.print(Acc_RAW);
else
  LCD.print("---");
LCD.print(" ");

lcdPosition(0,8);
Gyro_RAW = analogRead(Gyro_ADC_pin);
Serial.print("\tGyro="); // ECO on SerialMonitor
Serial.println(Gyro_RAW);
LCD.print("Gyr=");
if (Gyro_RAW<10) LCD.print(" ");
if (Gyro_RAW<100) LCD.print(" ");
if (Gyro_RAW<1000)
  LCD.print(Gyro_RAW);
else
  LCD.print("---");

Serial.println();

if (digitalRead(P1_pin)==0) {
  Serial.println("P1 pressed"); // ECO on SerialMonitor
  clearLCD();
  lcdPosition(0,0);
  LCD.print("P1 pressed");
}
if (digitalRead(P2_pin)==0) {
  Serial.println("P2 pressed"); // ECO on SerialMonitor
  clearLCD();
  lcdPosition(0,0);
  LCD.print("P2 pressed");
}
if (digitalRead(P3_pin)==0) {
  Serial.println("P3 pressed"); // ECO on SerialMonitor
  clearLCD();
  lcdPosition(0,0);
  LCD.print("P3 pressed");
}
if (digitalRead(PStart_pin)==0) {
  Serial.println("PStart pressed"); // ECO on SerialMonitor
  clearLCD();
  lcdPosition(0,0);
  LCD.print("PStart pressed");
}

delay(500);
}

void initIBat() {
  int i;
  IBat_Zero=0;
  for (i=0; i<50; i++)
  {
    IBat_Zero += analogRead(IBat_ADC_pin);
    delay(20);
  }
  IBat_Zero /= i;
}

```

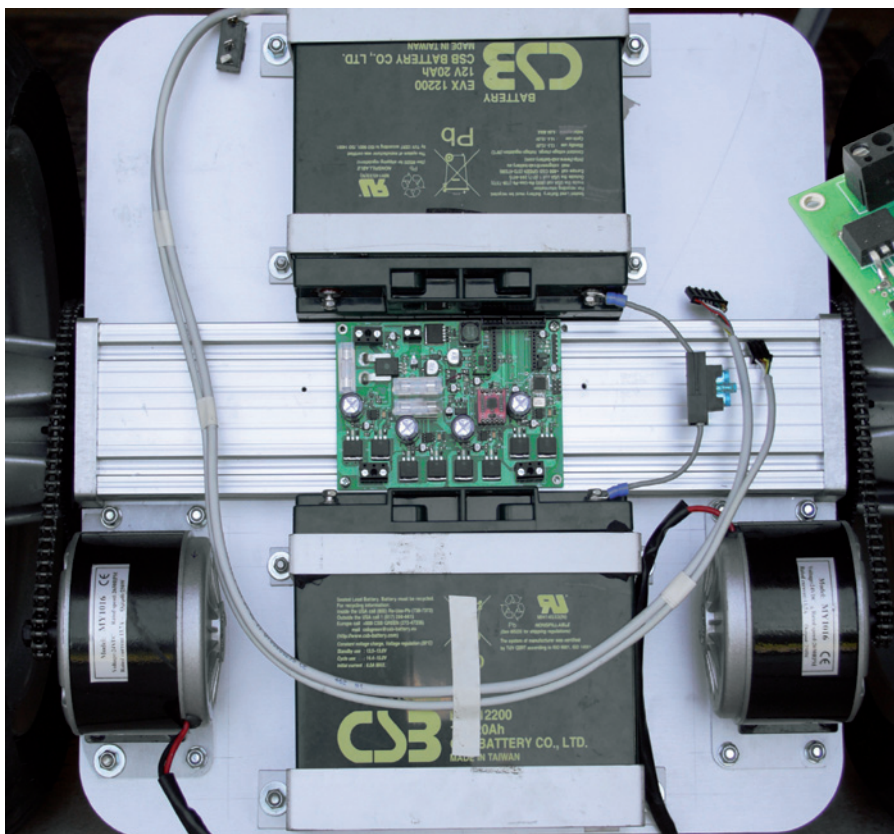



Fig. 15 - Posizionamento del PCB sotto la pedana del veicolo.

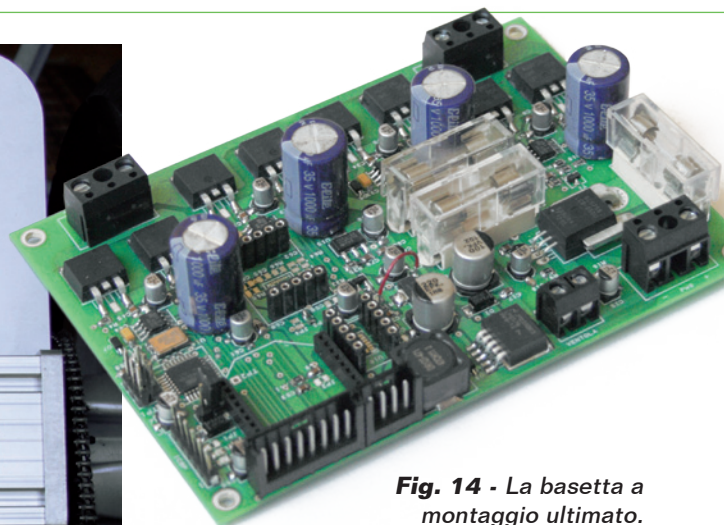


Fig. 14 - La basetta a montaggio ultimato.

ad ogni vibrazione esterna. Approfondiremo questi aspetti quando implementeremo la telemetria e i dati acquisiti dai sensori saranno disponibili in un apposito grafico. A questo punto è possibile iniziare il test dei motori di OpenWheels. Per questa operazione è necessario predisporre alcune misure di sicurezza la più importante delle quali è mantenere il veicolo sollevato da terra anche se probabilmente sarebbe meglio rimuovere le ruote lasciando i motori liberi di girare. Lo sketch denominato *OpenWheelsTestMotor* vi permette di testare i motori semplicemente agendo sul valore del duty-cycle impostabile tramite i pulsanti P+ e P-. In questi primi sketch non sono implementate le protezioni a livello software quindi ricordiamo di usare sempre la massima cautela. Il motore di destra va cablo sul connettore di destra denominato MOT1 e di conseguenza il motore di sinistra sul connettore MOT2; per assicurare il corretto senso di rotazione rispettate la serigrafia riportata sulla scheda. Durante questa prova è necessario inserire i due fusibili di potenza, magari riducendo il loro valore a soli 5A. Il valore del duty-cycle sarà visibile sia su serial monitor, sia sul display LCD. Il pulsante PStart, quello sul manubrio, viene usato per abilitare il driver di potenza; solo con questo pulsante premuto si ha la rotazione

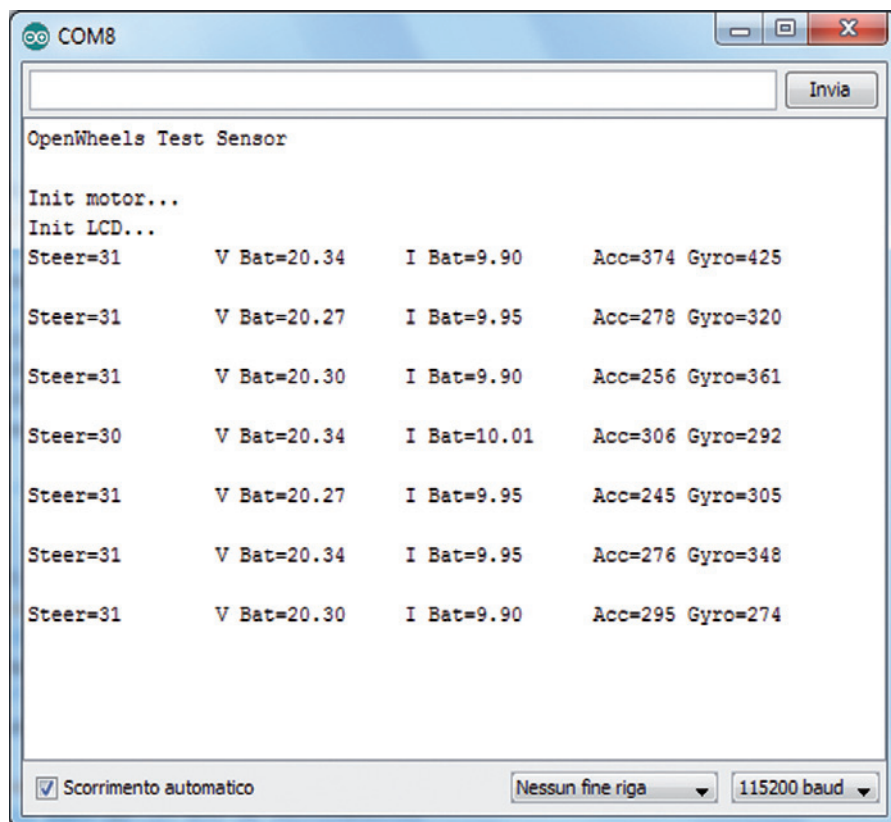


Fig. 16 - Arduino Serial Monitor per test sensori.

dei motori. Per evitare picchi di corrente potenzialmente pericolosi è opportuno premere e rilasciare questo pulsante solo quando il PWM è impostato a zero. Premendo il pulsante Enter il valore del PWM viene immediatamente impostato a zero. Potete seguire questa procedura anche al banco, prima di installare la scheda sulla meccanica, utilizzando un piccolo motore a 24 VDC. Come avrete modo di verificare, il pulsante di Start risulterà molto comodo per l'avvio di OpenWheels che, una volta alimentato, risulterà pienamente operativo ma in modalità di stand-by con assorbimenti bassissimi. In questa modalità tutte le funzioni sono implementate ed il display potrà fornire tutta una serie di indicazioni sulla lettura dei sensori, ma i motori non saranno azionati, potendo quindi ruotare liberamente come una specie di "folle" delle automobili; in questa modalità OpenWheels potrà facilmente essere spostato tirandolo dal manubrio.

Il prossimo mese descriveremo il sistema di telemetria wireless da noi messo a punto per controllare da remoto tutti i parametri operativi del veicolo e inizieremo l'analisi del firmware definitivo (ma suscettibile di miglioramenti).

Ricordiamo che per questo progetto abbiamo realizzato un minisito (<http://openwheels.elettronica.in.it>) nel quale vengono descritti in maniera molto più dettagliata i particolari costruttivi del nostro veicolo, le modifiche apportate nel tempo e quelle suggerite dai lettori, nonché i circuiti di controllo proposti dai lettori. Lo scopo ultimo è quello di realizzare un veicolo perfettamente funzionante ad un costo di gran lunga inferiore rispetto a quello di analoghi prodotti disponibili in commercio nonché quello di poter personalizzare il veicolo in relazione alle proprie esigenze.

Sketch TestMotor

```

/*
  OpenWheels Test Motor Program
  by Mirco Segatello

  Hardware OpenWheels 1.1 Ottobre 2012
  board: compatibility to ARDUINO Duemilanove
  compiler: Arduino 1.0.2
  Copyright: Creative Commons BY-NC-SA
*/

#include <SoftwareSerial.h>

//ADC Inputs
#define Acc_ADC_pin 0
#define Gyro_ADC_pin 1
#define Steer_ADC_pin 2
#define VBat_ADC_pin 3
#define IBat_ADC_pin 4

//Driver pin
#define ShutDown_pin 3 // ShutDown for motor driver (LOW=SD)
#define PWMLeft_pin 9 // PWM for motor Left
#define PWMRight_pin 10 // PWM for motor Right

//Digital pin
#define SerLCD_TX 4 // software serial TX for Serial LCD
#define P1_pin 5 // P1 pushbutton + function
#define P2_pin 6 // P2 pushbutton - function
#define P3_pin 7 // P3 pushbutton Enter function
#define PStart_pin 8 // Start Contact

//Battery variables
float VBatLevel;
float VBatScale=0.036;
float IBatLevel;
float IBatScale=0.0538;

//motor variables
int dutyCycleLeft;
int dutyCycleRight;
int stallMotor; // value of PWM for motor stall
float drive=0;
int drivePWM;
int maxPWM;
boolean status;

// Sensor
int Acc_RAW;
int Gyro_RAW;
int Steer_RAW;

// LCD TX pin (RX not necessary)
SoftwareSerial LCD = SoftwareSerial(0, SerLCD_TX);

void setup() {
  Serial.begin(115200);
  analogReference(EXTERNAL);

  pinMode(P1_pin, INPUT_PULLUP);
  pinMode(P2_pin, INPUT_PULLUP);
  pinMode(P3_pin, INPUT_PULLUP);
  pinMode(PStart_pin, INPUT_PULLUP);

  Serial.println("OpenWheels Test Motor\n");
  Serial.println("Init motor...");
  initMotor();
  delay(500);
  Serial.println("Init LCD...\n");
  serLCDInit();
  delay(500);
}

void loop() {
  status=digitalRead(PStart_pin);
  digitalWrite(ShutDown_pin, !status);

  if (digitalRead(P1_pin)==0) {
    if (drive>-stallMotor) drive-=1.0;
    SetMotor();
  }
  if (digitalRead(P2_pin)==0) {
    if (drive<stallMotor) drive+=1.0;
    SetMotor();
  }
  if (digitalRead(P3_pin)==0) {
    drive=0.0;
    SetMotor();
  }
  Serial.print("drive="); // ECO on SerialMonitor
  Serial.println(int(drive));
  clearLCD();
  lcdPosition(0,0);
  LCD.print("PWM=");
  LCD.print(int(drive));
  lcdPosition(1,0);
  if (status==0)
    LCD.print("RUN");
  else
    LCD.print("STOP");

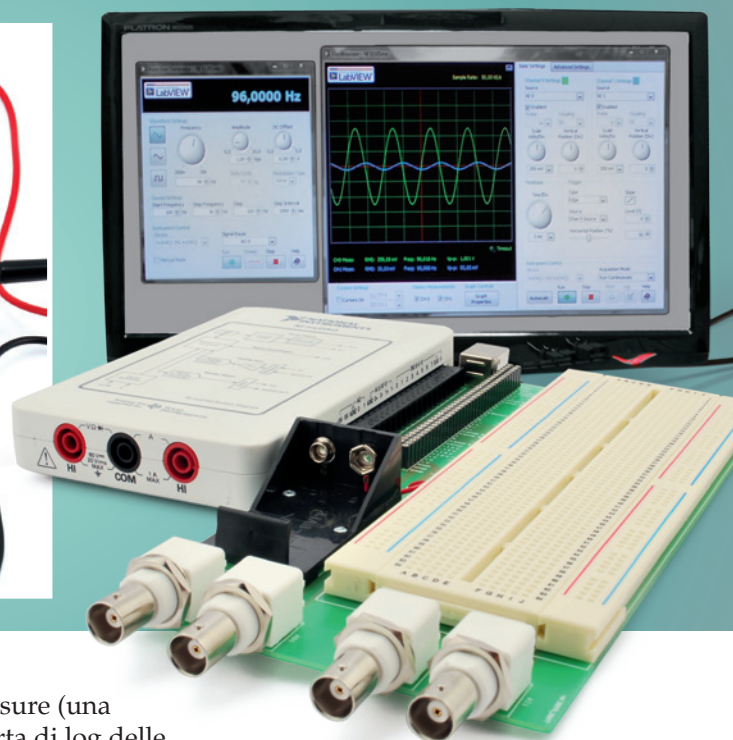
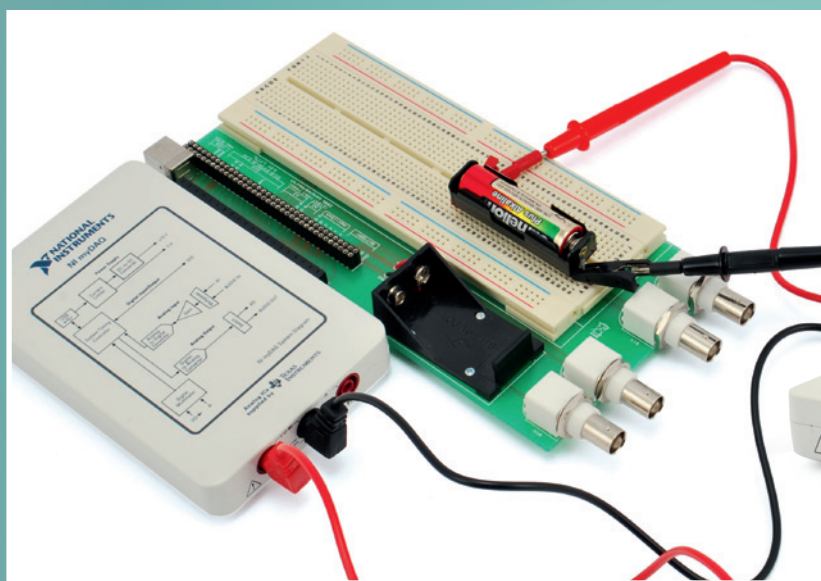
  delay(100);
}

```


myKIT, MISURA DI TENSIONE CON MYDAQ E COL MULTIMETRO DIGITALE

Acquisiamo una tensione sfruttando il DMM e visualizziamola con NI LabVIEW.

..... di DAVIDE SCULLINO



Prosegue la serie delle esercitazioni realizzate impiegando il set didattico myKIT e myDAQ di National Instruments; questo mese descriviamo un esperimento, rientrante nei 22 del programma didattico che corredda myKIT, il quale ci propone di utilizzare la scheda di acquisizione myDAQ per misurare una tensione e visualizzarne il valore in un Pannello Frontale in LabVIEW. La tensione viene acquisita utilizzando il canale DMM (Digital Multi Meter) di myDAQ e la relativa misura sarà visualizzata sia in forma numerica che mediante un grafico che riporta la successione storica delle

misure (una sorta di log delle tensioni lette dal DMM di NI myDAQ). Per l'esattezza, in questa esercitazione ci dedichiamo ad acquisire la tensione continua fornita da una pila mini stilo, ossia in formato AAA. Per quanto riguarda il cablaggio, è sufficiente collegare il terminale positivo dell'ingresso DMM di myDAQ al capo positivo della pila ed il terminale negativo DMM al polo negativo della batteria. Nella pratica, questo collegamento viene effettuato sfruttando la protoboard, che è la scheda

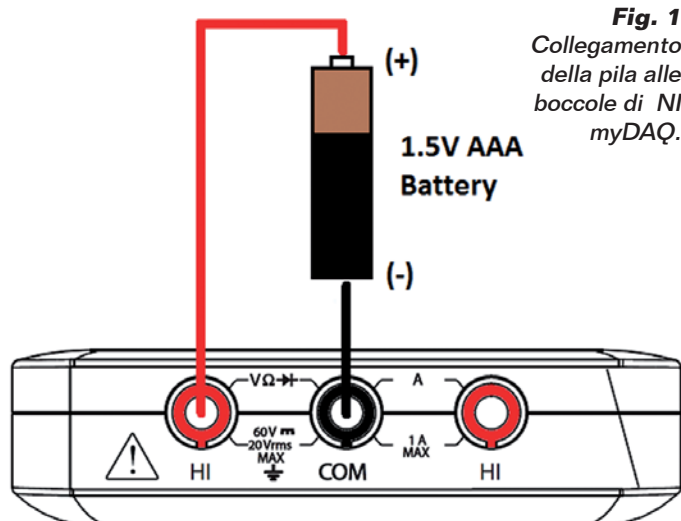


Fig. 1
Collegamento
della pila alle
boccole di NI
myDAQ.

(vedere l'articolo descrittivo di myKIT pubblicato nel fascicolo n° 171 della nostra rivista) per esperimenti facenti parte del set didattico myKIT e progettata per interfacciarsi direttamente con il connettore laterale di NI myDAQ; la protoboard ospita una breadboard sufficientemente estesa, nonché delle connessioni BNC e la preziosa USB, grazie alla quale NI myDAQ viene connesso ad un qualsiasi computer dotato di tale interfaccia di comunicazione.

Per questo esperimento si innesta NI myDAQ nella connessione laterale della protoboard, quindi si monta su quest'ultima il portapila AAA da c.s. compreso nel myKIT in modo che sia stabile e si collegano le boccole HI e COM dell'ingresso del Multimetro Digitale di NI myDAQ agli estremi del portapila. Il collegamento è meglio descritto dalla Fig. 1 e può essere facilmente effettuato mediante due coccodrilli e i puntali da multimetro inclusi nel set da esperimenti myKIT.

Va ricordato che myDAQ è un'interfaccia di acquisizione e output dotata di varie linee, ma anche di una comoda sezione che all'esterno si presenta come un multimetro; infatti dispone di tre boccole che hanno lo stesso significato di quelle in dotazione ai tester con selettore delle misure: una comune (di colore nero, siglata COM) alle misure di corrente e tensione, nonché a quelle ohmmetriche, una rossa (HI) che corrisponde al positivo delle misure di tensione e resistenza, ed un'altra rossa (sempre siglata HI) che è il positivo del circuito per la misura della corrente. Come nei multimetri, tra quest'ultima e COM è inserito un resistore di shunt che deriva la corrente ed ai capi del quale si verifica la caduta di tensione tramite cui l'interfaccia ricava il valore della corrente misurata.

Le grandezze (tensione misurata tra HI-tensioni e

COM e tra HI-correnti e COM) acquisite da questa interfaccia di misura vengono poi digitalizzate e comunicate tramite USB da NI myDAQ al computer cui quest'ultima unità viene collegata.

CODIFICA DELLA STRATEGIA

In LabVIEW è necessario creare un pannello frontale per visualizzare le misure di tensione in formato sia numerico che grafico. I dati saranno acquisiti mediante un VI utilizzando il controllo DAQ Assistant, il quale poi provvederà a passarli agli indicatori sul pannello frontale, secondo la logica mostrata nello schema a blocchi di codifica mostrato nella Fig. 4.

Il diagramma a blocchi LabVIEW è molto simile allo schema a blocchi di codifica; manca solamente la pila, che non deve essere inclusa nel diagramma.

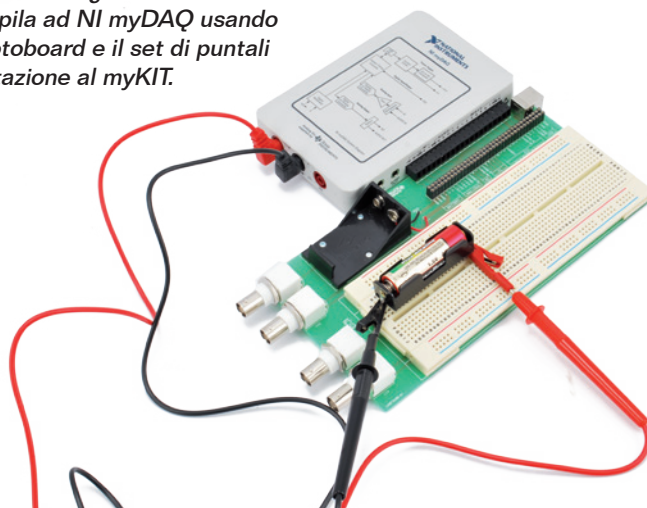
COME FUNZIONA

All'interno del ciclo, sulla parte sinistra, è presente il controllo DAQ Assistant, utilizzato per acquisire i valori di tensione misurati ai capi dei terminali del multimetro digitale (DMM). I valori di tensione sono trasferiti (a destra nel diagramma a blocchi) ai terminali dei controlli -presenti sul pannello frontale- numerico e grafico. Tutto il codice dentro il ciclo while continua a funzionare fino a che non viene premuto il pulsante di Stop sul pannello frontale. Il blocco Wait rappresentato dall'orologio (in alto a destra) ritarda ogni iterazione successiva del ciclo while di 100 ms. Pertanto la frequenza di uscita è di 10 campioni al secondo, ossia 10 Hz.

PANNELLO FRONTALE LABVIEW

L'interfaccia grafica utilizzata per visualizzare le misure di tensione propone sia un indicatore nume-

**Fig. 2 - Collegamento
della pila ad NI myDAQ usando
la protoboard e il set di puntali
in dotazione al myKIT.**



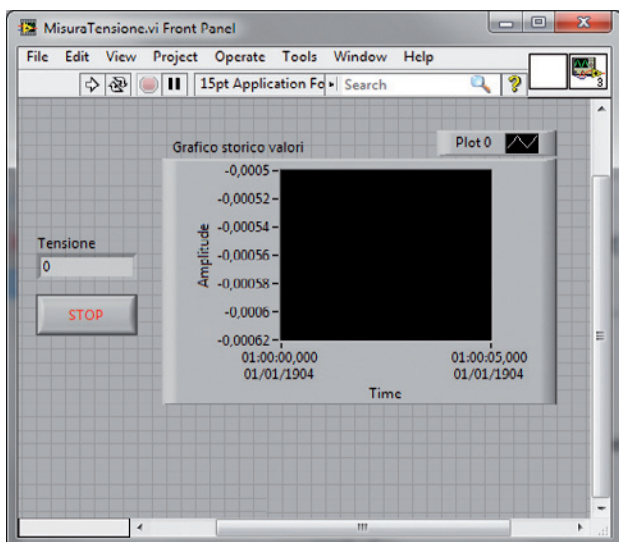


Fig. 3 - Aspetto del Pannello Frontale in NI LabVIEW.

rico che un pannello grafico: l'indicatore numerico visualizza i valori della tensione misurati istante per istante, mentre il pannello grafico tiene traccia di un certo numero di misurazioni nel tempo.

COSTRUZIONE DEL PANNELLO FRONTALE

Aprire un nuovo VI in LabVIEW: vi apparirà la finestra di disegno del pannello frontale del VI. Dal menu Window selezionate Show Block Diagram per aprire la finestra di disegno del diagramma a blocchi del VI.

Posizionate il cursore del mouse all'interno della finestra di disegno del diagramma a blocchi e fate clic col tasto destro del mouse: comparirà la Palette Functions; selezionate la libreria Structures e poi While Loop. Trascinate quest'ultimo all'interno della finestra di disegno del diagramma a blocchi e posizionalo, poi ingranditelo in modo da contenere i controlli che andrete ad aggiungere. Cliccate all'interno del controllo While Loop in modo da renderlo attivo e passate alla finestra di disegno del Pannello Frontale.

Sul Pannello Frontale, cliccate con il pulsante destro del mouse per far comparire la palette Controls, dove sono elencati tutti i tipi di controllo e di indicatore, ma anche decorazioni, tabelle, ecc. Per comodità, potete mantenere fissa sullo schermo la palette Controls, così da non dover premere ogni volta il tasto destro del mouse per farla ricomparire: allo scopo, basta fare clic sulla puntina in alto a sinistra della palette. Ora estendete la voce Modern (la prima) della palette Controls e selezionate Numeric, poi Numeric Indicator. Trascinate e posizionate l'indicatore sul Pannello Frontale. Ora, sempre dalle palette Controls e Modern, posizionatevi sulla libreria Graph e selezionate Waveform Chart, quindi trascinate e posizionate l'oggetto sul



Fig. 4 - Schema a blocchi di codifica dell'esercitazione.

Tabella 1

Tipo di collegamento	Scalare	Array 1D	Array 2D	Colore
Numerico				Arancione (virgola mobile), Blu (intero)
Booleano				Verde
Stringa				Rosa

pannello frontale. Infine, posizionatevi sulla libreria Boolean e selezionate Stop Button. Trascinate e posizionate anche questo controllo sul Pannello Frontale. Posizionate, ridimensionate ed etichettate i controlli come preferite.

COSTRUZIONE DEL DIAGRAMMA A BLOCCHI

Ora implementerete il diagramma a blocchi. Se non lo visualizzate, dal menu Window impartite il comando Show Block Diagram. Nel diagramma a blocchi, gli oggetti del Pannello Frontale appaiono sotto forma di terminali; questi sono porte di ingresso e di uscita che scambiano informazioni tra il Pannello Frontale e il diagramma a blocchi. Qui dovrete già visualizzare gli oggetti ed il pulsante Stop con le stesse etichette che avevate messo sul Pannello Frontale; dovrete anche visualizzare i terminali degli oggetti con questa etichetta: . È comunque preferibile visualizzarli con questa etichetta , che consente di ridurre lo spazio che essi occupano nel diagramma a blocchi. Per fare ciò,

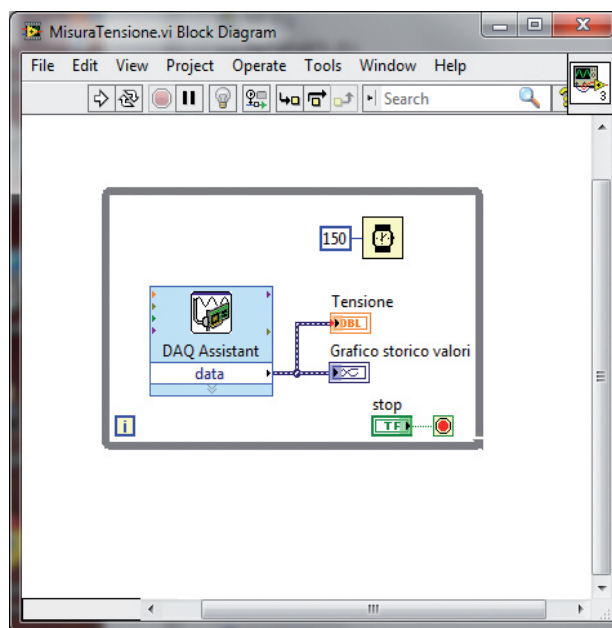


Fig. 5 - Diagramma a blocchi LabVIEW.

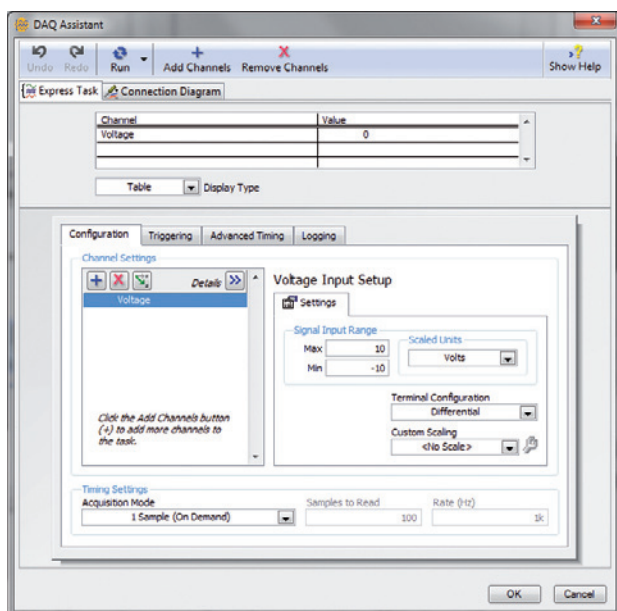


Fig. 6 - Configurazione del DAQ Assistant.

dal menu Tools impartite il comando Options; nella parte sinistra della finestra che si aprirà, fate clic su Block Diagram e, in General, disabilitate Place front panel terminals as icons.

Ora, prima di iniziare ad implementare il diagramma a blocchi, osservate che i rettangolini che rappresentano i controlli possono essere impostati sia come indicatori (cioè ricevono dati) sia come controlli (cioè producono dati). Nel diagramma a blocchi, il controllo si distingue dall'indicatore per due particolari: il primo è una freccia sul terminale che indica la direzione del flusso dei dati (i controlli hanno frecce che mostrano il dato partire dal terminale, mentre negli indicatori la freccia che mostra il dato entra nel terminale); il secondo è il bordo intorno al terminale (i controlli hanno un bordo più spesso di quello degli indicatori). Per passare da indicatore a controllo dovete cliccare col tasto destro del mouse sopra ogni rettangolino che rappresenta l'oggetto e impartire il comando *Change to Control*; questa operazione può essere effettuata anche dal Pannello Frontale. Notate che il pulsante Stop è già impostato come controllo.

Prestate attenzione ai collegamenti che effettuerete sul diagramma a blocchi: essi permettono di trasferire dati tra oggetti del diagramma a blocchi; ogni collegamento parte da una sorgente di dati e la collega ad uno o più VI e funzioni che leggono i dati stessi. I collegamenti hanno differenti colori, stili e spessori, in funzione del tipo di dati. Un collegamento spezzato ---✗--- appare come una linea nera tratteggiata con una X rossa in mezzo. I collegamenti possono rompersi per svariati motivi: ad esempio se si tenta di collegare due oggetti con tipi di dati tra loro incompatibili. La Tabella 1 illu-

stra i tipi di collegamento più comuni.

I collegamenti possono essere realizzati solamente tra ingressi e uscite che veicolino dati compatibili tra loro.

Nel diagramma a blocchi, facendo clic con il pulsante destro del mouse compare la palette *Functions*, che contiene i VI, le funzioni e le costanti da usare per creare il diagramma a blocchi.

Se volete bloccare la palette *Functions* sullo schermo, fate clic sulla puntina che si trova in alto a sinistra.

Ora potete iniziare la costruzione del diagramma a blocchi; vedrete che i terminali degli indicatori e dei controlli inseriti nel pannello frontale sono già presenti all'interno del While Loop.

Per prima cosa aggiungete il controllo DAQ Assistant.

1. Controllate che myDAQ sia collegato al PC;
2. Cliccate all'interno del While Loop con il tasto destro del mouse e, nella palette, estendete la voce Measurement I/O, quindi posizionatevi sulla libreria NI-DAQmx e selezionate il controllo DAQ Assistant, trascinatelo e posizionatelo all'interno del While Loop. Quando appare la finestra di configurazione del DAQ Assistant, selezionate nell'ordine:
 - a. Acquire Signals;
 - b. Analog Input;
 - c. Voltage;
 - d. myDAQ1 (NI myDAQ); questa voce potrebbe essere diversa a seconda di come è stata compiuta l'installazione nel PC;
 - e. DMM;
 - f. Finish;
3. Lasciate l'impostazione del campo Timing Settings su:

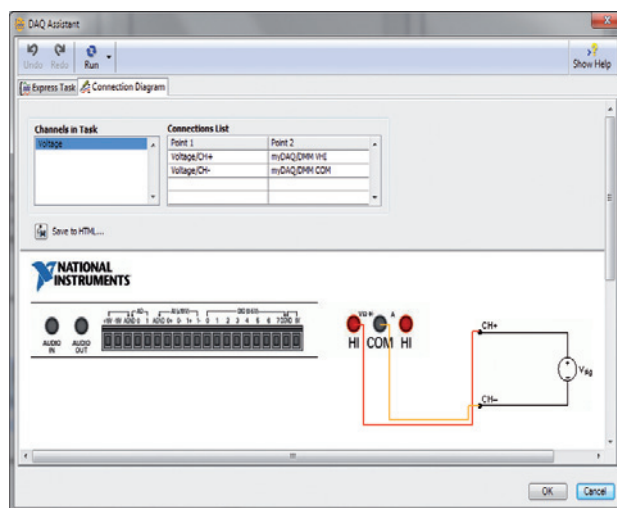


Fig. 7 - Schema del collegamento con NI myDAQ.

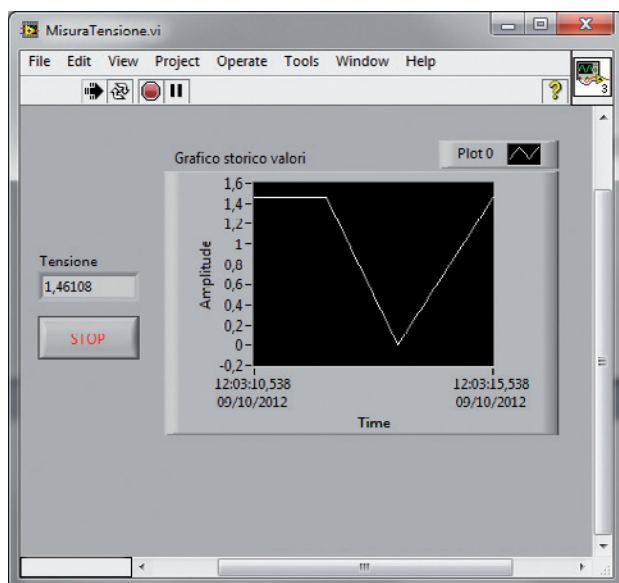


Fig. 8 - Il VI LabVIEW dell'esercitazione, in esecuzione: si noti il pulsante Stop che permette di interromperlo.

- a. 1 Sample (On Demand);
4. Fate clic su OK.

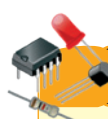
Nella **Fig. 7** potete notare e controllare lo schema di collegamento proposto da LabVIEW secondo la configurazione impostata in DAQ Assistant. Ora, nel pannello frontale, con lo strumento Wiring collegate il blocco DAQ Assistant ai terminali degli indicatori numerico e grafico. Sempre con lo strumento Wiring, collegate il controllo Stop al quadratino in basso a destra all'interno del ciclo While (**Fig. 5**). Questo consentirà, dopo aver mandato in esecuzione il programma, di uscire dal ciclo While e perciò di terminare il programma.

Successivamente, su Programming selezionate Timing e inserite il blocco Wait (ms) all'interno del ciclo While; ora, sempre utilizzando lo strumento Wiring, fate clic con il pulsante destro del mouse in corrispondenza del punto di collegamento sul lato sinistro del blocco e, dal menu contestuale, impartite il comando Create»Costant: apparirà un riquadro collegato al blocco Wait. Qui, con lo strumento Labeling inserirete il valore 150; questa operazione garantisce che ogni iterazione del ciclo While vada in esecuzione ogni 150 ms.

Se avete agito come appena descritto, il diagramma a blocchi dovrebbe presentarsi come in **Fig. 5**. Potete inserire i commenti a vostro piacimento utilizzando lo strumento Labeling, in quanto non interferiscono con il codice grafico ma ne migliorano la comprensione. Se volete cambiare le impostazioni di scrittura (font, dimensione, colore, ecc.), sia nel pannello frontale che nel diagramma a blocchi, nella barra del menu in alto cliccate su "Text Settings" (è il riquadro che riporta la dimensione e il font

che state utilizzando: 15pt Application Font). Quando avrete finito di costruire il vostro VI, aprite il menu File e impartite il comando *Save as*; nella finestra di dialogo che si apre, scegliete il nome e la destinazione del file corrispondente. Ora cliccate sul pulsante Run  nella barra degli strumenti, per mettere il VI in esecuzione (**Fig. 8**). Nel grafico, le variazioni di tensione sono state ottenute staccando e ricollegando la batteria.

Mentre il VI è in esecuzione, l'icona del pulsante Run si trasforma nella seguente: . Una volta terminata l'esecuzione, il pulsante Run torna al suo stato originale e gli indicatori del Pannello Frontale mantengono i dati visualizzati. Mentre il VI è in esecuzione, appare il pulsante Abort Execution (); facendovi clic potete fermare il VI immediatamente. Il pulsante Abort Execution va usato soltanto se non riuscite ad arrestare il VI in altro modo. Notate che se un VI non va in esecuzione, vuol dire che degli errori non lo rendono eseguibile. Se il VI che state creando o modificando contiene errori, il pulsante Run appare spezzato (); in questo caso, anche dopo aver finito i collegamenti nel diagramma a blocchi, il VI non è eseguibile. Generalmente l'errore può consistere in un ingresso richiesto non collegato o in un collegamento spezzato. Per risolvere il problema, fate clic sul pulsante Run spezzato in modo da accedere alla finestra di dialogo *Error List*, dove viene elencato e descritto ogni errore riscontrato; qui, facendo doppio clic su un errore andate direttamente alla posizione in cui esso si trova nel diagramma a blocchi. ■



per il MATERIALE

Questo progetto fa parte del set di esperimenti didattici realizzato da IRS-ElettronicaIn per la board NI myDAQ con il supporto del software LabVIEW. Il set (cod. SETmyKIT, Euro 232,00) comprende tutti i componenti per realizzare i 22 esperimenti, i VI relativi e una protoboard. È possibile richiedere protoboard aguntive (cod. PROTOBOARDmyDAQ) al prezzo di 52,00 Euro. Tutti i prezzi sono comprensivi di IVA e non comprendono la board NI myDAQ né il software LabVIEW. Futura Elettronica distribuisce questi prodotti presso gli utenti privati mentre la distribuzione presso gli Istituti Scolastici è a cura di IRS srl (Via Vigonovese 81 - 35127 PADOVA - tel: +39 049 8705156 - info@irsweb.it).

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

ALLARME GSM MULTIFUNZIONE

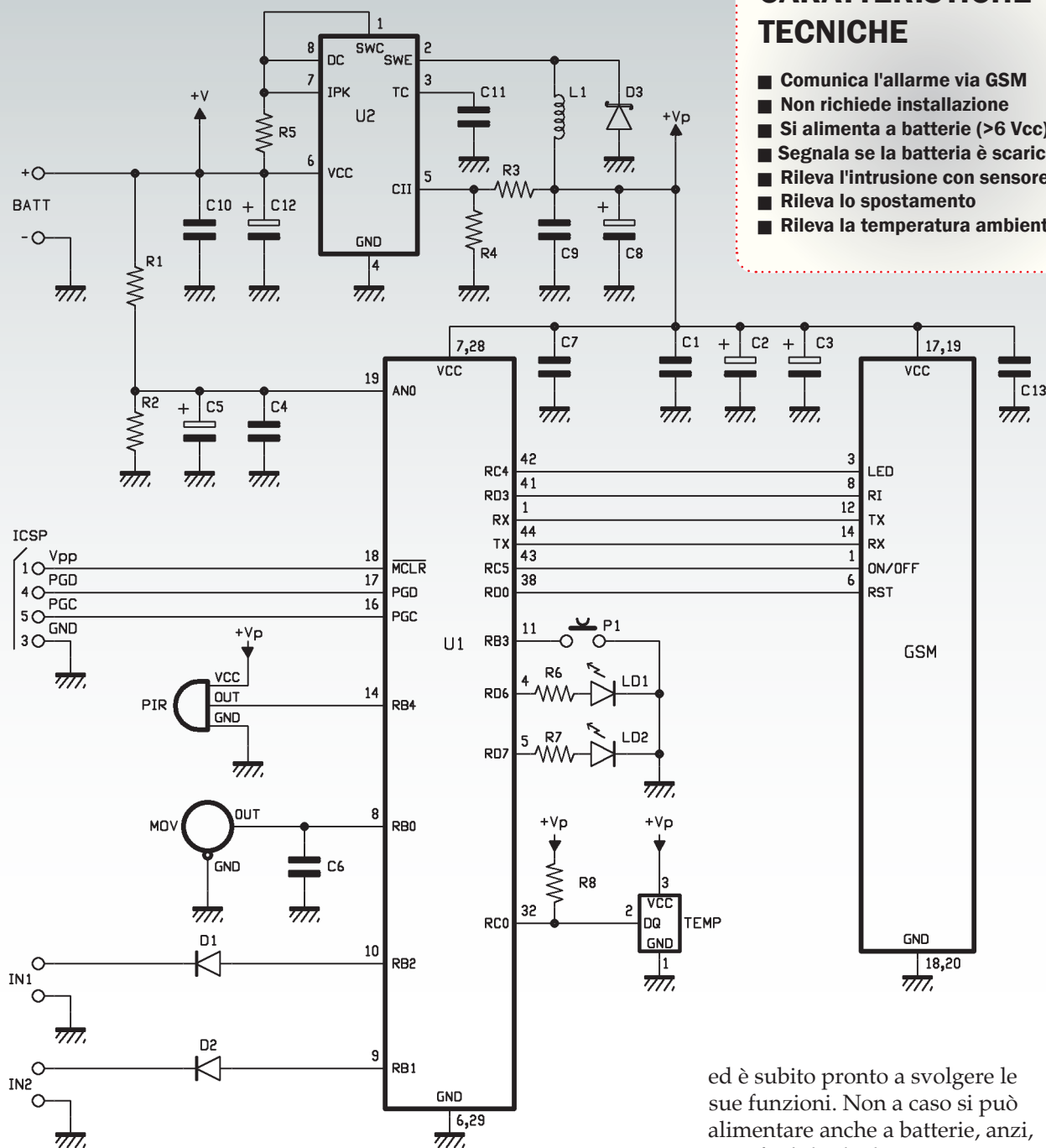
Integra un sensore di movimento P.I.R., funziona a batteria e comunica via cellulare per trasmettere le condizioni di allarme e ricevere comandi di gestione e configurazione da remoto. Segnala anche se viene spostato e se la tensione d'alimentazione è insufficiente.

di BORIS LANDONI

Con l'offerta disponibile oggi in commercio non è difficile trovare un antifurto che si adatti alle esigenze più disparate, ma sicuramente se andiamo nel campo dei dispositivi

portatili, da applicare ad automezzi o attivare in qualsiasi ambiente senza installazione, il cerchio si restringe; bisogna quindi progettarsi da soli un qualcosa di specifico come, ad esempio, il circuito che descriviamo in queste pagine. Si tratta di un





CARATTERISTICHE TECNICHE

- Comunica l'allarme via GSM
- Non richiede installazione
- Si alimenta a batterie (>6 Vcc)
- Segnala se la batteria è scarica
- Rileva l'intrusione con sensore P.I.R.
- Rileva lo spostamento
- Rileva la temperatura ambiente

antifurto non molto convenzionale, nel senso che in un solo apparato integra un sensore di oggetti in movimento basato su P.I.R. ed uno di movimento che rileva lo spostamento del sistema, ma anche una sonda di temperatura; invece di attivare segnalatori ed attuatori locali, trasmette

la situazione di allarme in modo silenzioso ma molto efficace: tramite SMS inviati dal modulo GSM/GPRS che porta a bordo. Questo circuito nasce come sistema d'allarme integrato, molto versatile, di pronto utilizzo e che non richiede installazione: infatti basta appoggiarlo dove serve

ed è subito pronto a svolgere le sue funzioni. Non a caso si può alimentare anche a batterie, anzi, è preferibile che lo sia; proprio in previsione di ciò, tra gli allarmi che può rilevare e trasmettere c'è anche quello di batteria scarica. A dirla tutta, non è solo un antifurto, ma un allarme multifunzione che rileva intrusione, spostamento, temperatura ambiente e anomalie nell'alimentazione; tutto in un circuito di 63x50x25 mm, tutto racchiudibile in un

contenitore per essere subito pronto all'uso. Basta appoggiarlo su un mobile o da qualsiasi parte serva, ed è pronto a svolgere il proprio compito. Inoltre dispone di due ingressi digitali, uno dei quali è impostato per rilevare la condizione di cortocircuito o l'apertura: con esso possiamo rilevare, tramite un contatto o un transistor, condizioni come l'attivazione o disattivazione di un sistema d'allarme fisso o parametri elettrici di vario genere, ovvero leggere lo stato di ulteriori sensori. In quest'ultimo caso il circuito richiede una vera e propria installazione: non si può pensare di arrivare, accenderlo e posarlo su un tavolo. Ma anche l'installazione fissa è una delle possibilità previste.

Gli utilizzi del nostro allarme sono molteplici: se abbiamo un camper possiamo accenderlo ed appoggiarlo in un posto della cabina che consenta al sensore P.I.R. di rilevare gli intrusi; in questo caso viene utile anche il sensore di movimento, che gli permette di avvertirci se il camper viene portato via senza il nostro consenso. Con il dispositivo possiamo anche rilevare se la batteria dell'automezzo si sta scaricando e riceverne un avviso via SMS in modo da intervenire prima che il veicolo non parta più e che l'allarme non sia in grado di proteggerlo. Il nostro allarme può anche essere messo in una baita o in un bungalow, per rilevare l'intrusione di estranei; in questo caso, la possibilità di alimentarlo a batterie è provvidenziale, dato che non sarebbe tanto semplice avere l'alimentazione di rete. Inoltre, incorporando la batteria nel contenitore non è possibile sabotarlo, in quanto prima di aprire e sconnettere la fonte di alimentazione sono scattati sia il P.I.R. che il sensore di movimento.

In un fabbricato, il nostro allarme può anche essere usato per rilevare, oltre all'intrusione, anche eventuali incendi: infatti il sensore di temperatura avvisa se fa più caldo del dovuto. La funzione di rilevamento della temperatura può essere usata

semplice, dato che abbiamo affidato i compiti più pesanti ad un microcontrollore Microchip PIC18F46K20 (U1); l'alimentazione dell'insieme si effettua con una tensione continua (anche non stabilizzata) di valore compreso tra 6 e 32 Vcc, grazie ad un DC/

Telit corre a 100 Mbps

Offre una velocità di trasmissione dati su reti LTE di 100Mbps in download e 50Mbps in upload il nuovo modulo LTE920 di Telit Wireless Solutions destinato al settore automotive in Europa e Nord America. Il nuovo modulo - che misura appena 34 x 40 x 2.8 mm in una soluzione 198-pad LGA - è pienamente compatibile con il fallback DC-HSPA+, garantendo prestazioni fino a 42Mbps e 5,76 Mbps rispettivamente in download e upload, se disponibili. Il Quad-band GSM / GPRS e le prestazioni EDGE garantiscono la connessione del modulo anche nelle zone remote prive di copertura 3G o 4G. Dotato anche di un ricevitore satellitare GPS e GLONASS altamente performante, il modulo LE920 offre una copertura superiore di navigazione anche in ambienti refrattari e nei canyon urbani con un posizionamento veloce e preciso, ideale per sistemi completi ed integrati di navigazione e servizi di localizzazione che utilizzano il sistema di infotainment della vettura.



anche per avvisarci se fa troppo freddo, in modo che possiamo, se abbiamo installato un telecomando, avviare da remoto il riscaldamento del luogo dove abbiamo posato il dispositivo.

SCHEMA ELETTRICO

Ma mettiamo da parte le applicazioni e vediamo di cosa si tratta, analizzando il circuito dell'allarme: modulo GSM a parte (che descriveremo più avanti), si tratta di qualcosa di abbastanza

DC converter a configurazione buck presente nel circuito. Partiamo dunque con la descrizione dello stadio d'alimentazione, che ricava, a partire dalla tensione di ingresso, una componente continua di 4 volt con cui alimenta sia il microcontrollore, sia il modulo GSM, il sensore P.I.R. e la sonda di temperatura. La tensione applicata ai punti + e - BATT viene filtrata dai condensatori C10 e C12 in modo da giungere ben livellata e

La board GSM

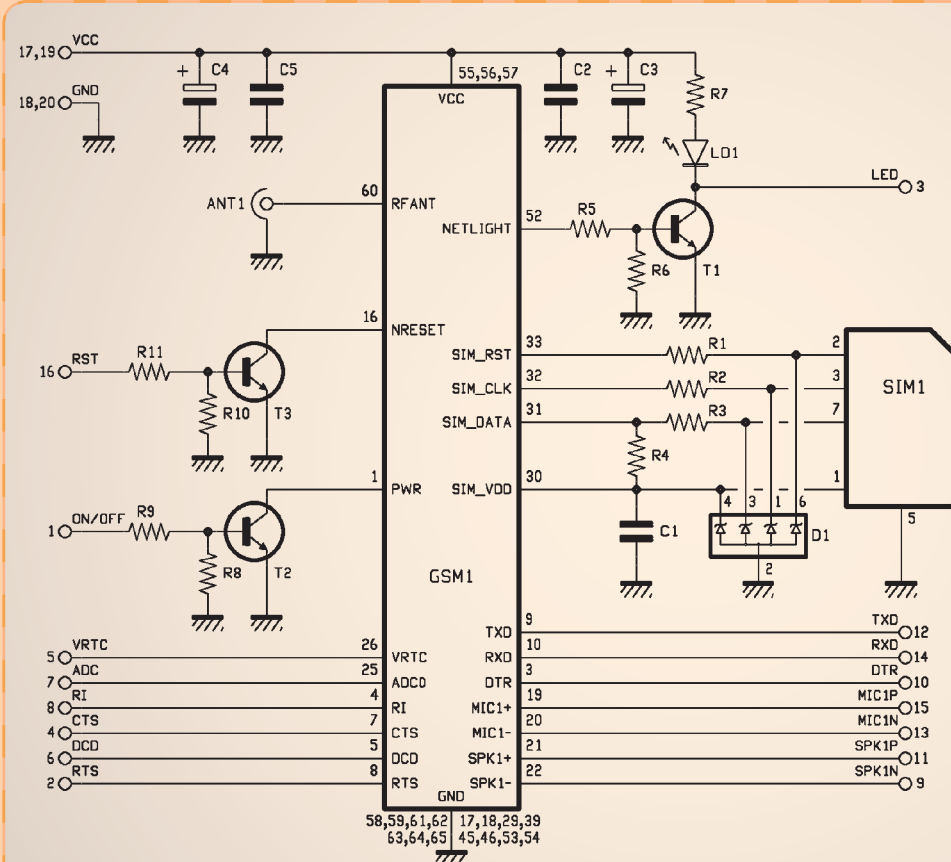
Il cellulare utilizzato in questo progetto è un modulo autonomo basato sul SIM900 della SIMCom montato su una basetta dotata di un connettore maschio a 20 contatti (due file da 10 l'una) a passo 2 mm che permette di inserirlo nel connettore femmina corrispondente della scheda base. Il connettore veicola l'alimentazione (Vcc, contatti 17 e 19) oltre alla linea di controllo dell'accensione (PWR) tutti i segnali e le linee di comunicazione seriale da e verso il modulo GSM, ma anche la

massa (GND, contatti 18 e 20).

La linea PWR viene usata per comandare l'accensione e lo spegnimento del modulo GSM, che comunque rimane sempre sotto tensione, fornita dalla linea Vcc ai piedini 55, 56, 57; la linea provvista di una resistenza di pull-up interna ed è attiva a zero logico, quindi il microcontrollore, per accendere il modulo cellulare, pone a livello logico alto PWR (contatto 1 del pin-strip) e manda in saturazione il transistor T2, il quale pone a livello basso

la linea PWR del GSM. Il controllo del reset avviene mediante una linea dedicata (il SIM900 prevede un ingresso di reset -NRESET- attivo a zero logico e provvisto di resistore interno di pull-up) e si ottiene facendo inviare al micro l'1 logico sulla linea RST, allorché il transistor T3 va in saturazione e pone a livello basso la linea NRESET del GSM1; contestualmente, RST viene portata ad 1 logico.

Procediamo ora con le linee di controllo dell'UART, ossia TXD, RXD, DTR, RTS,



Elenco Componenti:

- C1: 220 nF multistrato (0805)
- C2: 100 nF multistrato (0805)
- C3: 470 μ F 6,3 VL tantalio (CASE-D)
- C4: 470 μ F 6,3 VL tantalio (CASE-D)
- C5: 100 nF multistrato (0805)
- LD1: LED verde (0805)
- D1: vedi testo
- R1 ÷ R3: 15 ohm (0805)
- R4: 10 kohm (0805)
- R5: 4,7 kohm (0805)
- R6: 10 kohm (0805)
- R7: 330 ohm (0805)
- R8: 10 kohm (0805)
- R9: 4,7 kohm (0805)
- R10: 10 kohm (0805)
- R11: 4,7 kohm (0805)
- T1 ÷ T3: BC817
- GSM1: Modulo GSM SIM900
- SIM: Slot per SIM-CARD

Varie:

- Connettore Antenna MMCX 90°
- Strip maschio 2x10 poli 2mm
- Circuito stampato S900

priva di disturbi all'ingresso del regolatore integrato che segue, il quale ricava i 4 volt necessari a far funzionare il resto del circuito. Il regolatore switching è basato sul chip MC34063 (U2), utilizzato nella classica configurazione di regolatore PWM serie a carica d'induttanza, la cui tensione di uscita dipende dall'energia immagazzinata in L1; la tensione erogata dal DC/DC viene stabilizzata prendendo a riferimento il potenziale retrocesso al piedino

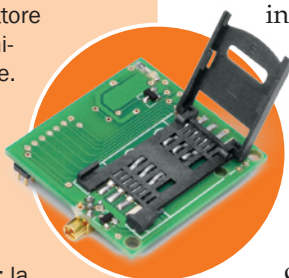
CII dal partitore resistivo R3/R4. I due componenti sono dimensionati per ottenere 4 V ai capi di C8 e C9. La componente continua (Vp) a valle di questi condensatori viene abbondantemente filtrata da altri condensatori collocati sulle linee di alimentazione del microcontrollore e del modulo GSM, in modo da evitare i disturbi al micro che il forte assorbimento impulsivo del modulo cellulare può causare quando ci si trova in trasmissione.

Passiamo adesso al microcontrollore, il quale, dopo il power-on-reset inizializza i propri I/O impostando RB0, RB1, RB2, RB3 ed RB4 come ingressi provvisti di pull-up (ad eccezione di RB4): il primo legge lo stato del sensore di movimento MOV (è un MS24M, acquistabile presso Futura Elettronica con il codice MOVMINI) mentre il secondo ed il terzo servono a rilevare la condizione logica agli ingressi IN1 e IN2; RB3 legge lo stato

CTS, DCD, che vanno all'esterno mediante i contatti, rispettivamente, 12, 14, 10, 2, 4 e 6 del connettore; lo stesso dicasi per VRTC (contatto 5) e ADC0 (25). L'audio, facente capo a due contatti per il microfono (ad ingresso differenziale) ed altrettanti per l'altoparlante, transita dai contatti 19, 20, 21, 22, che corrispondono rispettivamente a MIC1P e MIC1N (positivo e negativo del microfono) ed SPK1N e SPK1P (rispettivamente negativo e positivo dell'altoparlante). Il segnale di RI (indicatore di arrivo chiamata) esce dal contatto 18 del connettore e viene usato per informare il microcontrollore della scheda base.

L'antenna del modulo GSM in questo caso si collega direttamente sullo stampato del cellulare, la cui presa fa capo al piedino 60. Proseguiamo con il transistor T1, che comanda il LED di campo del cellulare: la sua base viene polarizzata dal livello logico presente sul piedino 52 (NETLIGHT) del GSM1. Dal collettore del transistor parte la linea che porta al contatto 3 del connettore che fa capo alla linea LED, con la quale il microcontrollore si informa sull'eventuale presenza della rete GSM e sullo stato di connessione del modulo (assenza segnale di rete, presenza rete ecc.).

Concludiamo l'analisi del circuito occupandoci delle linee di gestione della SIM, che si interfaccia al modulo GSM1 tramite il bus formato da SIM_CLK (clock) SIM_RST (reset) e SIM_DATA (canale dati); la linea SIM_VDD viene usata per accendere e spegnere la SIM ed è gestita dal modulo GSM. Notate la presenza del quadruplo Zener D1, i cui diodi proteggono la SIM-Card da eventuali picchi di tensione dovuti a disturbi, che possono arrivare dalla linea di alimentazione a quelle del bus di comunicazione tra SIM e cellulare.



del pulsante P1 ed RB4 la condizione logica dell'uscita del radar a infrarossi passivi siglato PIR. Quest'ultimo ha il pin OUT normalmente a riposo e fornisce impulsi da 5 volt quando rileva qualcosa nel proprio campo d'azione. Il pulsante P1 serve per la gestione locale della modalità di funzionamento del dispositivo. Quanto al sensore di movimento, è sostanzialmente un contatto miniaturizzato racchiuso in un case tipo quello dei transistor

BC107, 108 ecc., che quando viene scosso anche leggermente si chiude, cortocircuitando i piedini OUT e GND; per effetto di ciò, quando il circuito viene mosso o subisce una vibrazione, la linea RB0 riceve lo zero logico, che il firmware del microcontrollore interpreta come condizione di allarme.

Notate che ciascuno degli ingressi IN1 e IN2 è protetto da un diodo che impedisce l'applicazione ad RB1 ed RB2 di tensioni eccedenti quella con cui è alimentato il microcontrollore; per il PIC18F46K20, una tensione inferiore ad 1 V o un cortocircuito di IN1 o IN2 corrisponde allo stato logico basso, mentre l'apertura di detti ingressi o l'applicazione di una tensione continua di 1÷5 V equivale ad 1 logico.

La linea AN0 è un ingresso assegnato all'A/D converter contenuto nel microcontrollore e nel nostro caso viene usata per leggere il valore della tensione d'alimentazione attraverso il partitore resistivo formato da R1 e R2; tale partitore applica ad AN0 una tensione che è pari ad 1/11 di quella presente tra + e - BATT, quindi, ad esempio, se alimentiamo il circuito con una batteria da 12 V, AN0 si trova a circa 1,09 V. La linea RC0 viene inizializzata dal PIC come I/O bidirezionale, in quanto serve a dialogare con il sensore di temperatura, che nel nostro caso è una sonda Dallas DS18B20, molto precisa e capace di misurare temperature comprese tra -5 e 150 °C con un'accuratezza di appena ± 0,5 °C (nel campo -10÷85 °C), di digitalizzarle e di renderle disponibili su una linea seriale unidirezionale, in questo caso facente capo al pin RC0 del microcontrollore.

L'integrato DS18B20 esprime la

temperatura rilevata con una risoluzione di 11 bit; per interrogarla, il micro utilizza il comando "Convert T" (44 esadecimale) corrispondente all'avvio dell'operazione di conversione della misura. La sonda risponde con uno 0 mentre esegue la conversione e con un 1 ad operazione terminata. A questo punto nei registri del componente sono disponibili due byte corrispondenti al valore della temperatura rilevata in gradi Celsius. Quindi, non ci resta che leggerli con un codice comando "Read ScratchPad (codice BEh)". Nella modalità da noi scelta, dei 16 bit restituiti dal DS18B20 ogni volta che il microcontrollore lo interroga, i primi 11 (tutto il byte meno significativo e i primi tre bit di quello più significativo) esprimono il valore della temperatura e gli ultimi cinque il segno, che viene indicato con tutti 0 se la temperatura rilevata è positiva e tutti 1 se, invece, la misura si riferisce a una temperatura negativa (sotto lo zero).

La particolarità della sonda DS18B20 è che può esprimere la temperatura con i decimali: i primi quattro bit meno significativi si riferiscono a valori minori di 1, dato che il primo vale 2^{-4} , ossia 1/16, il secondo vale 0,125, il terzo 0,25 e il quarto 0,5. Così l'integrato può esprimere praticamente tutti i valori di temperatura che può misurare.

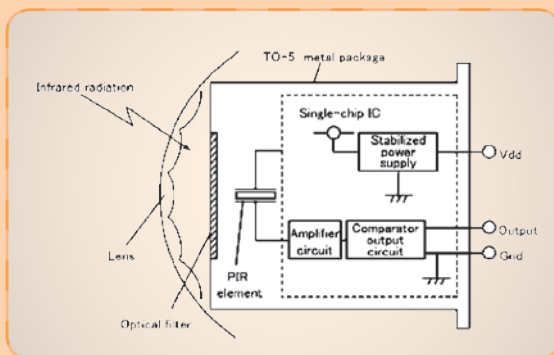
Per meglio chiarire come viene espressa la temperatura, immaginiamo di misurare 64,25°C positivi; all'uscita dell'integrato DS18B20 avremo due byte, il primo dei quali sarà (partendo dal bit più significativo) 00000100 e il secondo (sempre tenendo a sinistra il bit maggiormente significativo) 00000100. Analizzando i dati, il byte LSB dà il valore di 0,25 e il secondo byte fornisce +64 gradi.

Il sensore P.I.R. utilizzato

Per rilevare l'intrusione utilizziamo un sensore miniatura ad infrarossi passivi capace di funzionare con soli 5 volt, assorbendo meno di 300 microampere; il sensore permette di rilevare oggetti caldi (quindi persone ed animali in movimento; per caldi si intende che la loro temperatura supera di almeno 4 °C quella dell'ambiente circostante) che si spostano ad una velocità di 1 m/s, in un angolo di 102° in orizzontale e 92° in verticale, fino a una distanza di 12 metri. La divisione del campo di "osservazione" è in un massimo di 92 zone.

Il dispositivo fornisce in uscita un segnale logico TTL (0/5V) che vale 1 quando viene rilevato un corpo in movimento; l'uscita può erogare un massimo di 100 microampere. Affinché il sensore P.I.R. fornisca indicazioni attendibili bisogna attendere

un intervallo di tempo che ne garantisce il funzionamento stabile e che il costruttore indica in 30 secondi; per questa ragione, sconsigliamo di attivare l'allarme subito dopo averlo alimentato, perché conviene sempre aspettare da mezzo minuto ad un minuto.



Riepilogando, il valore complessivo formato dai due byte può essere compreso tra 11111101101110000 (-55 °C) e 000001111011010000 (+125 °C); in formato esadecimale, possiamo dire che l'escursione è compresa fra AB70h e 07D0h. Sempre ragionando in esadecimale, vediamo che una temperatura di 10,125 °C corrisponde a 00A2h e che una di -20 °C si esprime con BCC0h. La sonda DS18B20 dialoga usando il protocollo One-wire, che permette il colloquio bidirezionale su un unico filo; la resistenza di pull-up R8 mantiene, a riposo, il piedino 2 dell'integrato a livello alto. In questa applicazione utilizziamo linee distinte per alimentazione e dati, ma la DS18B20 potrebbe funzionare anche ponendo in comune l'alimentazione e i dati, ovvero utilizzando a pieno il protocollo one-wire.

Passiamo adesso agli I/O che il microcontrollore configura come uscite: si tratta di RD6 e RD7, utilizzati per gestire l'accensione dei LED di segnalazione LD1 e LD2. Per quanto riguarda il modulo

cellulare, è un FT900M della Futura Elettronica (www.futura-shop.it) basato sul SIM900 della SIMCom; il suo schema è riportato nel riquadro "Il modulo GSM". L'FT900M non è altro che una basetta ospitante il modulo SIM900, due transistor, un LED e pochi componenti passivi, oltre ad una presa coassiale dorata per l'antenna.

Il PIC18F46K20 contiene un UART accessibile dai piedini 44 (trasmissione) ed 1 (ricezione) che usa per dialogare con il modulo GSM; per l'esattezza, mediante il primo pin (TX) interroga ciclicamente il modulo allo scopo di verificare l'arrivo di SMS, mentre TX ed RX insieme servono, durante le procedure di chiamata, ricezione e trasmissione messaggi, per la comunicazione tra il microcontrollore e il GSM. Oltre al canale dati, per la gestione del cellulare il micro utilizza RC4, RD3: con RC4 verifica, sfruttando l'uscita del LED di campo del modulo GSM, che questo sia agganciato alla rete radiomobile; ciò serve per decidere come comportarsi in caso di al-

larne. Più esattamente, siccome il nostro circuito è pensato per inviare SMS nel caso si verifichi un allarme da rilevamento di intrusi, vibrazione o uscita della temperatura da un range preimpostato, il PIC18F46K20 deve sapere se il cellulare è al momento in rete o no: nel primo caso invia le segnalazioni, mentre nel secondo ritenta tre volte e poi abbandona la procedura di invio degli SMS, ovvero attende che il cellulare ritorni in rete e poi invia gli SMS. Quanto alla RD3, viene usata per rilevare l'arrivo delle chiamate entranti (si interfaccia con RI del modulo cellulare).

Completano il set di I/O destinati al cellulare, le linee RC5 ed RD0: la prima comanda l'accensione e lo spegnimento del GSM (tramite un transistor sistemato nella basetta del cellulare) mentre la seconda linea provvede al reset del cellulare, quando occorre e comunque dopo l'inizializzazione degli I/O conseguente all'accensione del circuito. La programmazione del PIC avviene in-circuit, mediante il connettore ICSP, che fa capo alle linee /MCLR, PGU e PGC; all'ICSP giungono anche l'alimentazione e la massa del micro.

COME FUNZIONA

Il sistema è progettato per essere configurato via SMS con appositi comandi descritti nella **Tabella 1**; quando va in allarme, provvede altresì ad inviare un SMS ad uno o più numeri di quelli memorizzati nell'apposita lista, in modo da avvertire una o più persone. Il nostro apparato può andare in allarme se si verifica una o più di queste condizioni:

- l'ingresso IN1 cambia di stato, si porta a livello alto o allo stato basso;
- il sensore di movimento rileva una vibrazione;

- il PIR rileva un'intrusione;
- la temperatura esce da un range preimpostato via SMS;
- la tensione d'alimentazione scende al di sotto di un certo valore.

Quando uno di questi eventi si verifica, il circuito prepara ed invia un messaggio di testo ai numeri (massimo 8) contenuti in un'apposita lista. È inoltre possibile personalizzare il testo dell'SMS inviato a seguito di ciascun allarme rilevato.

Per fare in modo che vada in allarme al verificarsi di uno solo degli eventi, bisogna escludere gli altri; ciò è possibile in quanto il dispositivo permette di abilitare o disabilitare i singoli sensori e ingressi, tramite appositi comandi da inviargli via SMS.

L'ingresso facente capo alla linea RB2 serve, diversamente dall'IN2, per attivare e disattivare l'allarme: lasciandolo aperto o ponendolo a livello logico alto, trascorso un intervallo (impostabile tra 0 e 59 secondi con il comando via SMS *TATT*) il circuito si attiverà e sarà pronto a rilevare le condizioni di allarme. Dopo l'attivazione dell'ingresso IN2 il LED verde LD2 inizierà a lampeggiare scandendo il tempo, trascorso il quale, il sistema diverrà attivo. Con il comando *AVV1* si imposta il dispositivo in modo che invii un SMS quando si attiva (*AVV0* disabilita questa funzione).

L'ingresso di allarme IN1 e gli allarmi P.I.R., di spostamento e di temperatura prevedono la possibilità di definire un tempo di inibizione regolabile con il comando da SMS *INI:xx*, dove al posto di *xx* si deve scrivere il tempo espresso in minuti; il tempo di inibizione si definisce per tutti insieme ma viene gestito separatamente per ogni fonte di allarme. È anche possibile azzerare

il tempo di inibizione qualora l'allarme cessi. Ad esempio, se si verifica un allarme all'ingresso IN1 e questo rimane a livello logico alto per un tempo superiore a quello di inibizione, allo scadere di questo tempo il circuito riceverà una nuova condizione di allarme e determinerà l'invio di un altro messaggio; diversamente, tornerà nella condizione di riposo.

Il tempo di inibizione è stato pensato per evitare che il circuito

generi inutilmente allarmi ed SMS multipli di uno stesso evento, ovvero per non ricevere falsi allarmi determinati da qualcosa che produce più impulsi (un contatto poco stabile all'ingresso IN1, una temperatura che oscilla intorno al valore di soglia ecc.) ripetutamente.

Attivando il PIR e/o il sensore di movimento è consigliabile non utilizzare la funzione di azzeramento del tempo di inibizione, visto che forniscono segnali

[piano di **MONTAGGIO**]

Elenco Componenti:

R1: 10 kohm (0805)
 R2: 1 kohm (0805)
 R3: 2,2 kohm (0805)
 R4: 1 kohm (0805)
 R5: 0,1 ohm 1W (1206)
 R6, R7: 330 ohm (0805)
 R8: 4,7 kohm (0805)
 C1, C4: 100 nF multistrato (0805)
 C2, C3: 470 μ F 6,3 VL tantalio (X)
 C5: 22 μ F 25 VL tantalio (D)
 C6, C7: 100 nF multistrato (0805)
 C8: 470 μ F 6,3 VL tantalio (D)
 C9, C10: 100 nF multistrato (0805)
 C11: 100 pF ceramico (0805)
 C12: 22 μ F 25 VL tantalio (D)
 L1: Induttore 22 μ H
 U1: PIC18F46K20-I/PT (MF909)
 U2: MC34063AD
 LD1: LED 5 mm rosso
 LD2: LED 5 mm verde
 P1: Microswitch
 GSM: Modulo GSM SIM900 (FT900M)
 D1, D2: GF1M
 D3: MBRS140TRPBF
 MOV: Sensore movimento (6710-MOVMINI)
 PIR: Sensore PIR (8100-MINIPIR)
 TEMP: DS18B20+

Varie:

- Strip maschio 5 poli
- Morsettiera 2 poli passo 2,54 mm (3 pz.)
- Connettore 2x10 femmina passo 2 mm (RS 507-7522)
- Circuito stampato

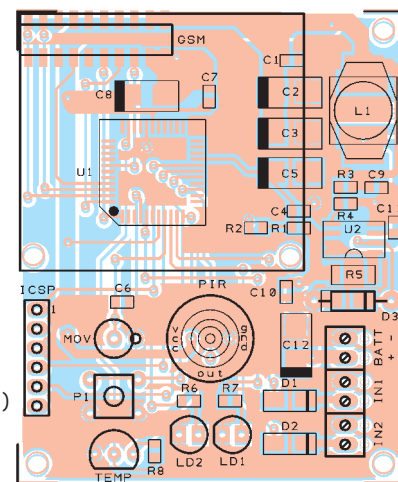


Tabella 1 - Funzioni e comandi

FUNZIONE	COMANDO SMS	VALORE PREDEFINITO	PWD OBBLIGATORIA
CAMBIO PASSWORD	PWDxxxx;12345	12345	*
MEMORIZZARE 1 NUMERO (massimo 8 numeri) (MAX 19 CIFRE PER NUMERO)	NUMx+393355760937;12345	-	SOLO SE LA POSIZIONE x NON È LIBERA
CANCELLARE UN NUMERO	NUMx;12345	-	*
VERIFICARE I NUMERI MEMORIZZATI	NUM?;12345	-	*
RESET COMPLETO	RES	-	*
RICHIESTA TEMPERATURA	TEMP		
IMPOSTA ALLARME TEMPERATURA MIN (DOVE y è + 0 - E xx VALE 00 ÷ 99)	ALLMIN:yxx	y=segno xx temp	
IMPOSTA ALLARME TEMPERATURA MAX (DOVE y è + 0 - E xx VALE 00 ÷ 99)	ALLMAX:yxx		
RESET TEMPERATURA MIN E MAX	TRES		
IMPOSTA IL NOME	NAME:xxxxx	xxxx=nome del localizzatore (massimo 15 caratteri)	
RICHIEDI IL NOME	NAME?		
RICHIEDI TEMPERATURE DI ALLARME	ALL?		
IMPOSTARE I NUMERI A CUI VERRANNO INVIATI GLI SMS	SMSxxxxxxxx:ON	tutti	*
INVIO SMS ALL'ACCENSIONE x VALE 1 PER ATTIVARLO E 0 PER DISATTIVARLO	AVVx	0	
TESTO ALLARME INGRESSO 1 ALTO (MAX 100 CARATTERI)	TINI1H:xxxxxxxxxx	ALLARME!! INGRESSO 1 ALTO	NO MULTIMESSAGGIO
TESTO ALLARME INGRESSO 1 BASSO (MAX 100 CARATTERI)	TINI1L:xxxxxxxxxx	ALLARME!! INGRESSO 1 BASSO	NO MULTIMESSAGGIO
TESTO ALLARME PIR MAX 100 CARATTERI	TINPIR:xxxxxxxxxx	ALLARME!! RILEVATA INTRUSIONE	
TESTO ALLARME TEMPERATURA (MAX 100 CARATTERI)	TINTEH:xxxxxxxxxx	ALLARME!! SUPERATA SOGLIA TEMPERATURA	NO MULTIMESSAGGIO
TESTO ALLARME TEMPERATURA (MAX 100 CARATTERI)	TINTEL:xxxxxxxxxx	ALLARME!! SUPERATA SOGLIA TEMPERATURA	NO MULTIMESSAGGIO
TESTO ALLARME MOVIMENTO (MAX 100 CARATTERI)	TINMOV:xxxxxxxxxx	ALLARME!! RILEVATO MOVIMENTO	NO MULTIMESSAGGIO
TESTO MESSAGGIO ACCENSIONE MAX 100 CARATTERI (INVIO SOLO AL PRIMO NUMERO IN MEM)	TSU:xxxxxxxxxx	SYSTEM STARTUP	NO MULTIMESSAGGIO
ABILITA'/DISABILITA IL SENSORE DI MOVIMENTO	SAM:x	0	
ABILITA'/DISABILITA IL SENSORE PIR	SAP:x	0	
ABILITA'/DISABILITA L'INGRESSO DI ALLARME	SAI:x	0	
ABILITA'/DISABILITA IL SENSORE TEMPERATURA	SAT:x	0	
ABILITA'/DISABILITA LA LETTURA DELLA BATTERIA	SAV:x	0	
STATO SENSORI (ABILITATO/DISABILITATO)	SAL		
IMPOSTA LIVELLO LOGICO INGRESSO DI ALLARME	LIV:x	x > A/B/V	
RICHIEDI LIVELLO LOGICO INGRESSO DI ALLARME	LIV?		
AZZERAMENTO TEMPO INIBIZIONE SE INGRESSO TORNA ALLA CONDIZIONE NORMALE	TIZ:x	0	
RICHIESTA SE AZZERAMENTO INIBIZIONE ATTIVO O DISATTIVO	TIZ?		
TEMPO OSSERVAZIONE INGRESSO	OSS:xx	0	
RICHIESTA TEMPO OSSERVAZIONE	OSS?		
TEMPO INIBIZIONE INGRESSO 1 E SENSORI	INI:xx	Xx=0+59 MINUTI	
DISABILITA LA RISPOSTA PER IL MULTIMESSAGGIO	RISP	-	
TEMPO DI INIBIZIONE DELL'ALLARME DOPO L'ATTIVAZIONE MEDIANTE L'INGRESSO IN2	TATT:xx	Xx=0+59 SEC	
ABILITAZIONE/DISABILITAZIONE ECHO SMS	ECHO:x	x > Posizione numero (0 disattivato)	
SLEEP PER xxx SECONDI	SLEEP:xxx	xxx da 000 a 999 secondi	
LEGGI LA TENSIONE DELLA BATTERIA	VOLT?		
IMPOSTA LA TENSIONE DI ALLARME (ES 115=11,5V)	ALLVOLT:xxx		

impulsivi ripetitivi. Per l'ingresso IN2 abbiamo anche previsto un tempo di osservazione.

Col comando *ECHO:x* si abilita l'inoltro dei messaggi di comando ricevuti dal circuito e privi di password o non provenienti da numeri in lista, al numero che si trova nella posizione *x* (*ECHO:0* disabilita l'inoltro); ciò permette al gestore del sistema, il cui numero telefonico deve essere *x*, di sapere se qualcuno tenta di variare la configurazione del circuito a sua insaputa.

Il sistema prevede la funzione Sleep, utile a prolungare l'autonomia quando viene alimentato a batteria: inviandogli il comando *SLEEP:xxx* il circuito va in condizione Sleep (low power) per *xxx* secondi. Durante tale modalità il funzionamento è minimale, cioè il circuito non rileva alcun allarme, non invia SMS e logicamente non legge l'arrivo di comandi SMS dal cellulare; l'uscita dallo Sleep avviene automaticamente ogni *xxx* secondi, allorché il circuito interroga il cellulare per verificare l'arrivo di eventuali messaggi ed eseguire i comandi che essi contengono. Fatto ciò, il sistema torna in Sleep e si risveglia ciclicamente ogni *xxx* secondi. Il circuito entra automaticamente nella modalità Sleep, ma solo se è disattivato, ovvero se l'IN1 è a livello logico basso; in caso contrario lo Sleep non è possibile. Se il circuito è andato in Sleep ma abbiamo bisogno di leggere da esso dei dati come la temperatura rilevata dalla sonda DS18B20 o la tensione della batteria, possiamo risvegliarlo momentaneamente in due modi: localmente, premendo il pulsante P1, o da remoto, facendo una telefonata al numero della SIM inserita nel suo cellulare. Al risveglio possiamo inviare un comando SMS per leggere i dati che vogliamo; il

circuito risponderà con l'SMS del caso e poi si riporterà automaticamente in Sleep.

La modalità Sleep è stata concepita per consentire di spegnere il sistema lasciandolo però disponibile per funzioni come la telelettura della temperatura e della tensione di batteria, funzione che in lunghi tempi di attività ci permette di capire se è il momento di intervenire in loco per ricaricare l'accumulatore. La funzione è utile anche se il circuito viene alimentato dalla batteria di un camper fermo per lungo tempo: in tal caso diventa un utile monitor.

CONFIGURAZIONE

Come appena detto, tutte le impostazioni del circuito si effettuano via SMS, ossia inviando al numero della SIM inserita nel modulo GSM un messaggio contenente il testo del comando voluto; il sistema accetta solamente SMS provenienti da numeri telefonici memorizzati nell'apposita lista dei numeri abilitati, che sono al massimo 8 e che coincidono con quelli cui destinerà gli SMS di allarme e di stato richiesti dalla configurazione che farete. In realtà prende per buoni anche i comandi che gli giungono da numeri non riconosciuti ma corredati dell'apposita password, la quale per impostazione predefinita e a seguito di un reset totale (imposto da SMS).

Il comando per memorizzare un numero è un SMS contenente il testo *NUMxnnnnnnnnnnnnnnnnnnnn;pwd*, dove *x* indica la posizione (1÷8) in cui salvare il numero rappresentato da *nnnnnnnnnnnnnnnnnnnn* e *pwd* è l'attuale password, necessaria se il comando SMS proviene da un telefono non in lista (serve comunque quando si memorizza il primo numero...) o se, pur arrivando da un telefono di quelli già memorizzati, la posizione speci-

ficata è già occupata da un altro numero. Ogni numero può essere lungo al massimo 19 caratteri e deve comprendere il prefisso di nazionalità, che per l'Italia è +39. Per cancellare il numero in una posizione della lista, il comando da usare è *NUMx;pwd*, dove *x* è la posizione (1÷8) e *pwd* l'attuale password; è anche possibile interrogare il dispositivo per conoscere tutti i numeri memorizzati: basta impartire il comando da SMS *NUM?;pwd*.

Il nostro dispositivo implementa la funzionalità Easy Setup, che permette di memorizzare il primo numero della lista in maniera semplice e senza richieste di password: dopo l'accensione e per circa tre minuti attende una telefonata, al sopraggiungere della quale salva in memoria il numero del telefono chiamante ed entra nel normale esercizio, dove si porta comunque anche se il tempo passa senza che giunga alcuna chiamata.

Una volta salvati i numeri della lista, bisogna definire quali sono quelli cui inviare gli SMS di allarme contenenti il tipo di allarme rilevato. Il comando corrispondente è *SMSxxxxxxx:ON*, dove *xxxxxxx* sono le posizioni della lista dove si trovano memorizzati i numeri. Ad esempio, *SMS1346:ON* imposta che a ricevere gli SMS siano i numeri nelle posizioni 1, 3, 4 e 6. Per disabilitare l'invio di SMS ad uno o più numeri bisogna inviare il comando *SMSxxxxxxx:OFF*, che dice al localizzatore di disabilitare le posizioni specificate. Ad esempio, *SMS2346:OFF* disattiva l'invio degli SMS ai numeri memorizzati nelle posizioni 2, 3, 4, 6. Alla prima accensione e dopo ogni reset, il localizzatore invia SMS a tutti i numeri che verranno memorizzati.

Alcuni comandi possono essere

eseguiti solamente se corredati di password; per altri, basta che giungano da telefoni i cui numeri siano presenti nella lista degli otto abilitati al comando e alla richiesta delle coordinate. La password predefinita (reimpostata dopo ogni reset totale del sistema) è 12345; è possibile sostituirla con una a piacimento (purché di cinque cifre) impartendo il comando del caso con un SMS contenente il testo *PWDnewpwd:pwd*, dove newpwd è la password da memorizzare e *pwd* l'attuale. E passiamo al comando che attiva l'ibernazione del dispositivo, ossia *SLEEP:xxx*; impartendolo, il localizzatore va a riposo (consumando meno di 6 mA) per il tempo, in secondi, scritto nel rispettivo SMS al posto delle *x*. Il valore dell'intervallo deve essere espresso sempre a tre cifre, quindi se volete l'ibernazione per 40 secondi dovete inviare il comando

SLEEP:040.

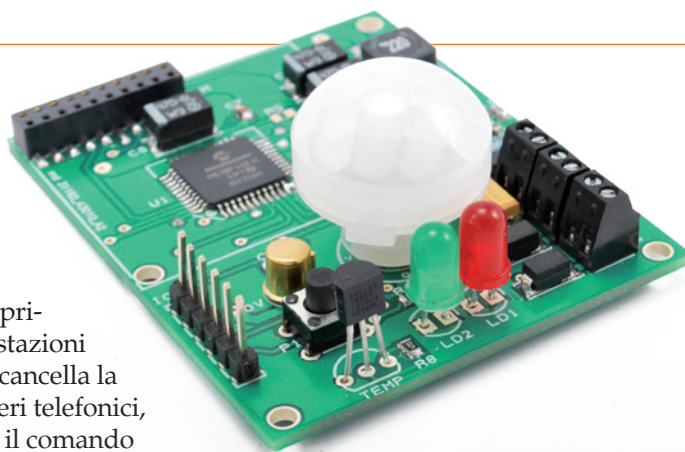
L'intervallo si sceglie tra 000 e 240 s.

Il reset, che ripristina le impostazioni predefinite e cancella la lista dei numeri telefonici, si ottiene con il comando *RES:pwd*, dove *pwd* è la password attuale.

Al circuito si possono inviare SMS contenenti più comandi purché separati da una virgola; con un apposito comando è possibile disattivare le risposte ad alcuni comandi inserendo, nell'SMS, il comando *RISP*.

Passiamo adesso alla gestione delle funzioni di allarme, iniziando con la temperatura: la richiesta dell'attuale temperatura ambiente si effettua inviando l'SMS con il testo *TEMP*; l'impostazione delle temperature massima e minima che danno origine ad allarme si effettua con i comandi rispettivamente *ALLMAX:yxx* e *ALLMIN:yxx*, nei quali *y* è il segno e *xx* corrisponde ai gradi. Per esempio, *ALLMAX:+85* imposta il sistema perché generi un SMS di allarme se vengono superati gli 85 °C sopra lo zero. Le impostazioni di temperatura possono essere azzerate inviando il comando *TRES*. Il comando *ALL?* interroga invece il sistema, che risponde con i valori di temperatura attualmente impostati.

L'allarme associato al sensore di temperatura si attiva con il comando *SAT:1*; se non lo volete, dovete impartire il comando *SAT:0*. L'impostazione resterà memorizzata fino al reset totale. Vediamo ora i comandi che riguardano l'allarme di movimento: *SAM:1* attiva il rilevamento dello stato del sensore e *SAM:0* lo disattiva; se volete che il sistema consideri il sensore di movimento dovete impartire il primo coman-

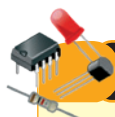


do. L'impostazione del caso resterà memorizzata fino al reset totale. Quanto al sensore PIR, il comando per attivarne la lettura e l'allarme è *SAP:1*; *SAP:0* disattiva la funzione. L'impostazione resta memorizzata fino al reset totale. Con il comando *SAL:1* diciamo al sistema di considerare l'ingresso IN1; con *SAL:0* gli ordiniamo di ignorarlo. I comandi per definire cosa il sistema debba considerare allarme, sono *LIV:A* per rendere l'ingresso sensibile al livello alto, *LIV:B* per attivarlo con lo zero logico e *LIV:V* per avere l'allarme a seguito della sola variazione di livello logico. Il comando *LIV?* produce l'invio da parte del sistema di un SMS contenente l'attuale impostazione per l'ingresso d'allarme IN2.

Il comando *SAV:1* permette di dare l'allarme se la tensione d'alimentazione scende sotto il valore impostato col comando *ALLVOLT:xxx* (*xxx* è la tensione espressa in decimi di volt); con *SAV:0* disattiviamo tale funzione. Per avere l'invio di un SMS all'accensione del circuito (per accensione s'intende l'attivazione quando IN2 è stato attivato) si deve inviare il comando *AVV:1*; si può disattivare questa funzione con *AVV:0*. Trovate tutti i comandi nella **Tavola 1**, con tanto di sintassi.

REALIZZAZIONE PRATICA

Il nostro allarme GSM è realizzato quasi interamente con componentistica in SMD, il che è un pregio perché consente di



per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono facilmente reperibili in commercio. Questo progetto è anche disponibile già montato e colludato (cod. FT909M) al prezzo di 126,00 Euro; sono compresi tutti i componenti, le minuterie, il sensore PIR, la board GSM FT900M e l'antenna GSM ANTGSMSTL-MM-CX. Non è compresa la SIM-card e il relativo abbonamento. Il circuito può essere alimentato con una tensione continua compresa tra 6 e 32 volt fornita da una batteria o da un alimentatore da rete. 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>

ridurne le dimensioni, ma può rappresentare una difficoltà per chi ha poca dimestichezza con questo tipo di montaggio perché realizzarlo richiede un po' d'attenzione e un minimo di attrezzatura consistente in una pinzetta per posizionare i componenti, una lente d'ingrandimento per verificare che sia tutto a posto e non vi siano cortocircuiti, un saldatore con punta fine e del filo di lega saldante del diametro di 0,5 mm o meno.

Gli unici componenti a montaggio tradizionale (passante) sono la sonda DS18B20, il sensore P.I.R. da stampato, quello di vibrazione, i LED ed il pulsante. Una volta ottenuta la basetta (le due tracce possono essere scaricate dal nostro sito www.

elettronica.in.it) sistemate per primo il microcontrollore, staginando un paio di piedini su due lati opposti per fermarlo, quindi procedendo con i restanti; completate le saldature ed accertato che non vi siano piazzole unite accidentalmente, si può passare a tutti gli altri componenti partendo dai passivi, quindi al quarzo e al transistor, proseguendo con il pulsante da c.s. e con l'interruttore P1. Non dimenticate il pin-strip ad 8 poli per connettere il programmatore in-circuit, e le morsettiere bipolari miniaturizzate per la batteria e i due ingressi IN1 e IN2. Per inserire il modulo FT900M dovete saldare un connettore femmina a 20 contatti a passo 2x2 mm. Completato il montaggio,

attraverso il connettore CN1 (8 poli) si può provvedere alla programmazione in-circuit del PIC. Ricordiamo che il modulo FT900M si compera già pronto, montato e collaudato, presso Futura Elettronica (www.futurashop.it) quindi non bisogna fare altro che montarlo sul circuito e funzionerà al primo colpo. Naturalmente il modulo necessita di un'antenna, che può essere una qualsiasi accordata per sistemi GSM dotata di cavetto terminante con un connettore MMCX.

Per far funzionare il dispositivo occorre un alimentatore capace di erogare una tensione da 6 a 32 volt e una corrente di almeno 1A, oppure una batteria agli ioni di litio da 800÷1.000 mA/h. ■




RADIOCOMANDO SU RETE Wi-Fi

cod. WIRC

€132,00



Utilizza il tuo iPhone o iPad
per pilotare da remoto qualsiasi
modello sfruttando la telecamera
montata a bordo.



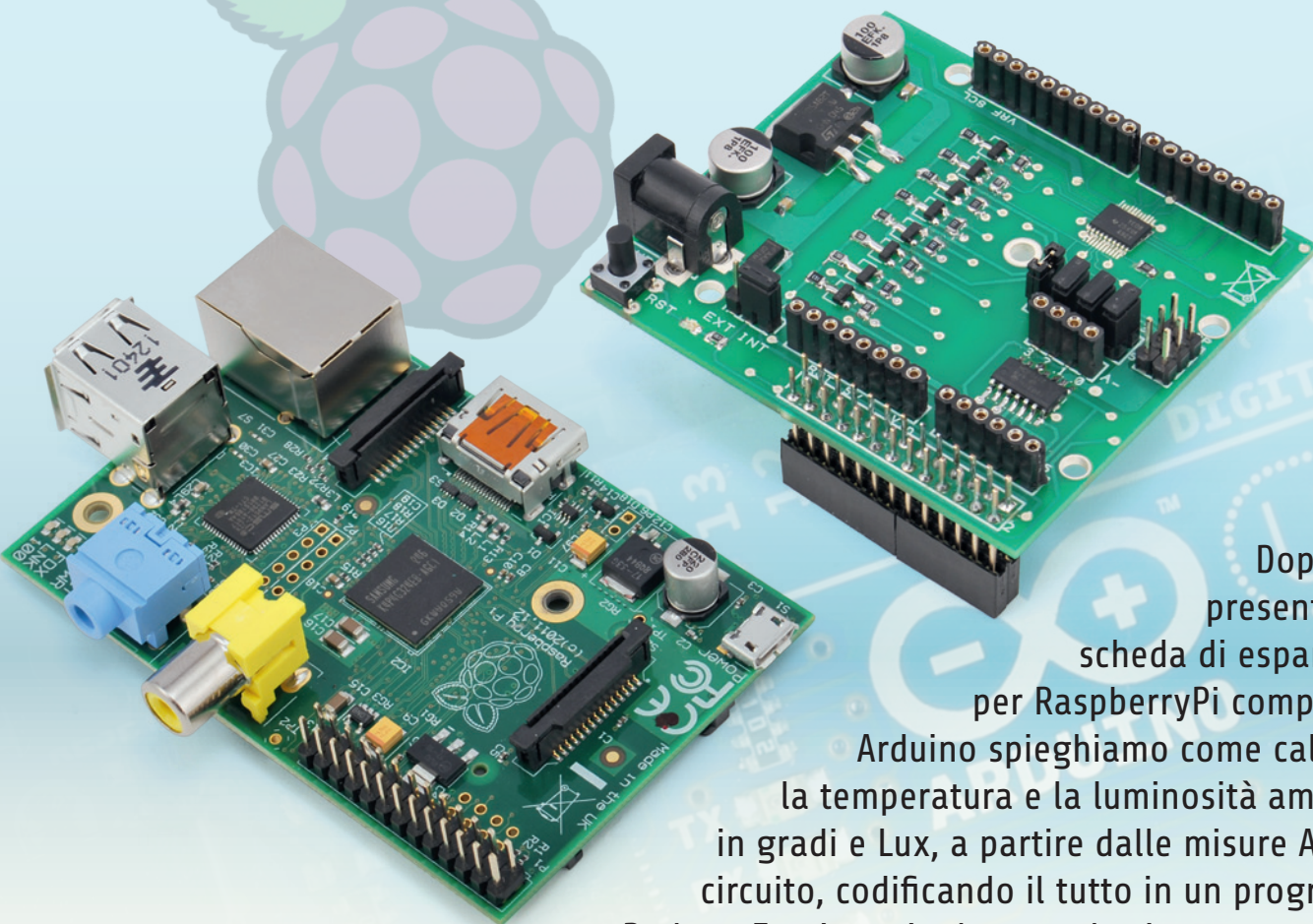

FUTURA
ELETTRONICA

Via Adige, 11
21013 Gallarate (VA)
Tel. 0331/799775 • Fax. 0331/792287

Maggiori informazioni su questo prodotto e su altri dispositivi
per robotica sono disponibili sul sito www.futurashop.it
tramite il quale è anche possibile effettuare acquisti on-line.

Prezzi IVA inclusa.

SHIELD DI ESPANSIONE ADC PER RASPBERRYPI COMPATIBILE CON ARDUINO



Dopo aver presentato la scheda di espansione per RaspberryPi compatibile Arduino spieghiamo come calcolare la temperatura e la luminosità ambiente in gradi e Lux, a partire dalle misure ADC del circuito, codificando il tutto in un programma Python. Facciamo inoltre un ulteriore passo avanti verso la compatibilità descrivendo come realizzare programmi che funzionino su entrambi i sistemi.

di MARCO MAGAGNIN

Dopo avere descritto la scheda di espansione del GPIO di RaspberryPi compatibile con la piedinatura di Arduino ed il modo di comunicare in i2C con il convertitore ADC montato sulla scheda stessa, abbiamo utilizzato un piccolo circuito per misurare temperatura e luminosità ambiente utilizzando una NTC ed una fotoresistenza. Continuiamo con la descrizione dei calcoli necessari a passare dalle misure di tensione ai capi delle resistenze, lette dal convertitore

ADC, ai corrispondenti valori di temperatura e luminosità espressi in gradi centigradi e lux; questi calcoli li codifichiamo in un programma in Python che presenta i risultati a video, ma è già predisposto per essere adattato in futuro per un progetto di integrazione con interfaccia Web. Abbiamo poi dedicato un approfondimento alla possibilità di scrivere programmi che, utilizzando la scheda di espansione su RaspberryPi, possano girare indifferente su Raspberry o su

Scriviamo programmi che funzionano sia su *RaspberryPi* che su *Arduino*

La nostra scheda di espansione è compatibile con la libreria ArduPi, una libreria per RaspberryPi, che “porta” molte delle funzioni disponibili nelle librerie standard di Arduino in ambiente RaspberryPi, mantenendone inalterata la sintassi.

La libreria è stata realizzata, in linguaggio C++, dalla Libelium Comunicaciones Distribuidas S.L. ed è un software open source distribuito secondo i termini della licenza GNU versione 3.

Questo approccio permettere di scrivere programmi che funzionano sia su Arduino, sia su RaspberryPi senza modifiche, anche se limitatamente

alle funzioni “compatibili”. Le funzioni disponibili nella libreria ArduPi sono elencate nel riquadro. Come potete vedere l'attenzione è posta all'utilizzo degli I/O digitali, all'uso della porta seriale, dei bus i2C (Wire) e SPI.

Sottolineiamo solamente la differenza di comportamento della funzione `attachInterrupt()`. Siccome RaspberryPi non supporta direttamente gli interrupt (e neanche potrebbe data la natura multithread e multi processo del sistema operativo), la funzione è stata realizzata in modo da attivare un thread separato, rispetto al programma principale, che esegue il polling del pin sul

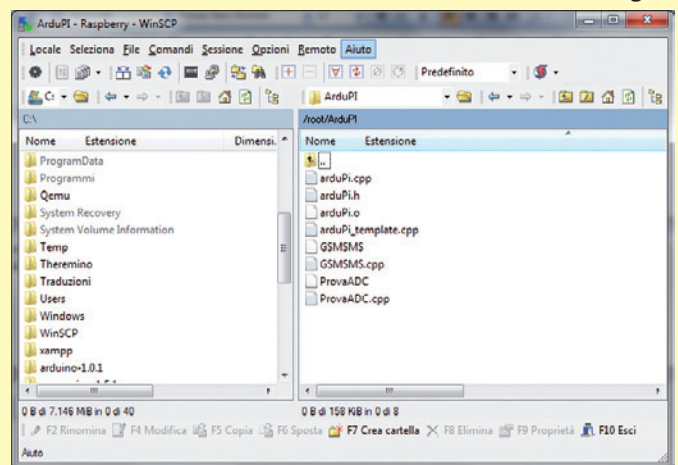


Fig. B

quale si vuole gestire l'interrupt. Gli eventi possono essere RISING, FALLING, HIGH e LOW. Possono essere utilizzati i pin da D2 a D9.

Per esempio, se vogliamo controllare un interrupt sul fronte di salita del pin 6 possiamo specificare l'istruzione

```
attachInterrupt(6,funzione_da_chiamare,RISING)
```

Per verificare il funzionamento della libreria ArduPi con la nostra scheda di espansione abbiamo riscritto il programma

Python di calcolo della temperatura e luminosità in stile Arduino; lo abbiamo compilato con l'IDE di Arduino (Fig A) e poi, dopo aver modificato il riferimento alla libreria, lo abbiamo compilato ed eseguito con RaspberryPi. Per installare la libreria ArduPi è sufficiente scaricarla dall'indirizzo

http://www.cooking-hacks.com/skin/frontend/default/cooking/images/catalog/documentation/raspberry_arduino_shield/arduPi_rev2_1-3.

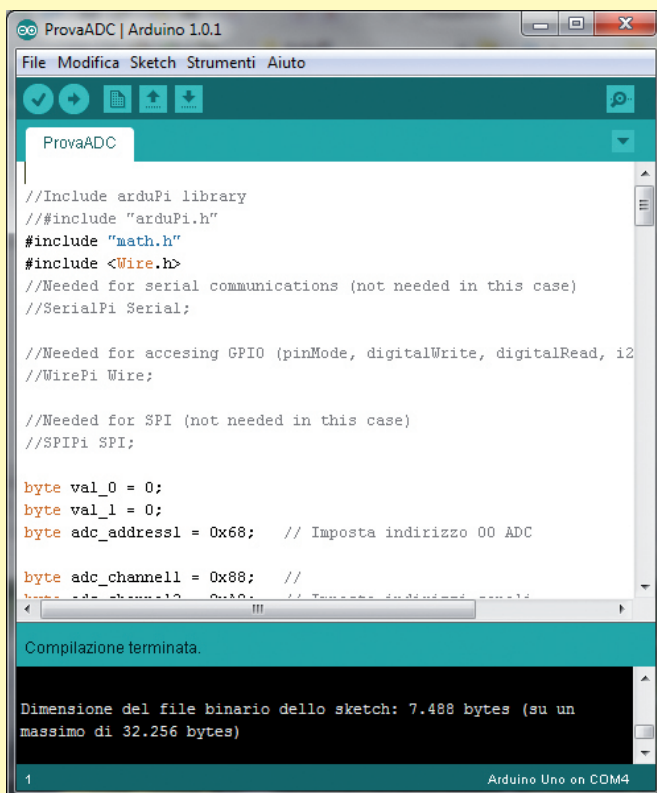


Fig. A

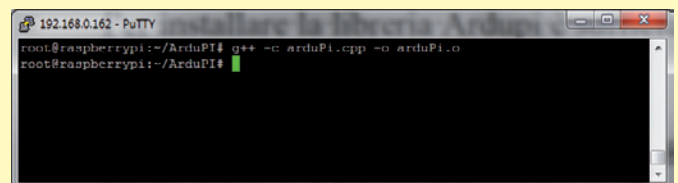


Fig. C

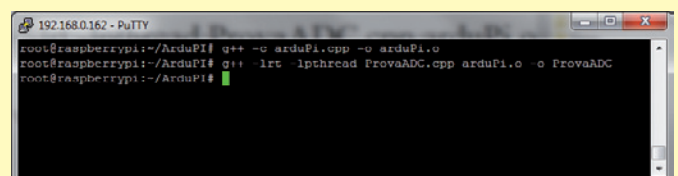


Fig. D

una scheda Arduino. Come esempio abbiamo duplicato in C il programma Python. Riportiamo in Fig. 1 il circuito dei sensori di temperatura e luminosità che abbiamo usato

negli esempi del numero scorso. Approfondiamo le caratteristiche dei sensori e le sequenze di calcoli, che ci permettono di ottenere valori significativi di temperatura e luminosità.

tar.gz per la relv. 2 e

http://www.cooking-hacks.com/skin/frontend/default/cooking/images/catalog/documentation/raspberry_arduino_shield/arduino_rev1_1-3.tar.gz per la rev. 1.

La scompattiamo nella cartella root o dove vogliamo e, per comodità, gli modifichiamo il nome in ArduinoPi. Sempre per comodità metteremo nella stessa cartella i programmi che andremo a realizzare. Compiliamo la libreria ArduinoPi con il comando (**Fig. C**):

```
g++ -c arduinoPi.cpp -o arduinoPi.o
```

Dopo aver scritto il programma ProvaADC.cpp visibile nel **Listato 1** (potete come sempre scaricarlo dal sito di Elettronica In) lo potete compilare e linkare alla libreria ArduinoPi con il comando (**Fig. D**):

```
g++ -lrt -lthread ProvaADC.cpp arduinoPi.o -o ProvaADC
```

Per eseguirlo diamo il comando (**Fig. E**)

```
./ProvaADC
```

Nella stessa figura vediamo il risultato, identico a quello ottenuto con il programma Python. Vedremo come farne buon uso per integrare ancora di più i due ambienti.

```
192.168.0.162 - PuTTY
root@raspberrypi:~/ArduinoPi# g++ -c arduinoPi.cpp -o arduinoPi.o
root@raspberrypi:~/ArduinoPi# g++ -lrt -lthread ProvaADC.cpp arduinoPi.o -o ProvaADC
root@raspberrypi:~/ArduinoPi# ./ProvaADC
Steinhart - Celsius = 22.442598
parametro B - Celsius = 22.574796
Lux = 178.719807
root@raspberrypi:~/ArduinoPi#
```

Fig. E

Funzioni generali:

- delay()
- delayMicroseconds()
- pinMode()
- digitalWrite()
- digitalRead()
- shiftIn()
- shiftOut()
- attachInterrupt()
- detachInterrupt()

Libreria Serial:

- available()
- begin()
- end()
- flush()
- peek()
- print()
- println()
- read()
- readBytes()
- readBytesUntil()
- find()
- findUntil()
- parseInt()
- parseFloat()
- setTimeout()
- write()

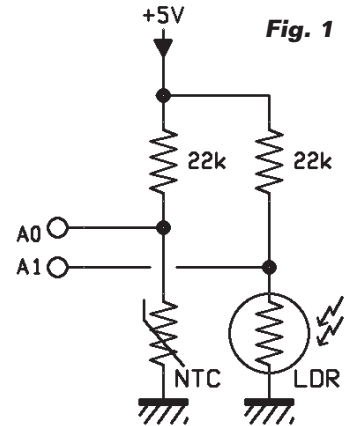
Libreria Wire:

- begin()
- requestFrom()
- beginTransmission()
- endTransmission()
- write()
- read()

Libreria SPI:

- begin()
- end()
- setBitOrder()
- setClockDivider()
- setDataMode()
- transfer()

Fig. 1



cient) è un componente che ha la caratteristica di variare il proprio valore resistivo al variare della temperatura, in particolare la sua resistenza decresce con l'aumentare della temperatura. Per questa caratteristica ben si adatta ad essere utilizzato come sensore di temperatura. La variazione del valore della resistenza non segue linearmente la variazione della temperatura, ma una curva che è possibile approssimare matematicamente, applicando due formule che possono essere scelte in base ai dati forniti nel datasheet del termistore stesso. La prima è la formula di Steinhart-Hart, un polinomio del terzo ordine in grado di interpolare i valori di temperatura rispetto alla resistenza con una precisione anche di 0,002 gradi.

$$T(R) = \left(A_1 + B_1 \ln \frac{R}{R_{ref}} + C_1 \ln^2 \frac{R}{R_{ref}} + D_1 \ln^3 \frac{R}{R_{ref}} \right)^{-1}$$

Equazione di Steinhart-Hart

dove R è la resistenza del termistore nel momento della misura, Rref è la resistenza nominale del termistore alla temperatura di riferimento di 25 C che, nel nostro caso, vale 10kOhm.

I coefficienti A, B, C e D del polinomio si possono determinare, per un termistore, con alcune misurazioni a temperature conosciute e con una serie di calcoli. In genere, fortunatamente, questi valori sono già specificati in un'apposita tabella nel datasheet del termistore che utilizziamo, congiuntamente ai gradi di precisione della misura ottenibile nei diversi intervalli di temperatura.

Per il nostro termistore abbiamo utilizzato i seguenti valori, che danno il grado di precisione migliore nell'intorno dei 25 gradi centigradi:

- A = 0.00335402
- B = 0.000256985
- C = 2.62013e-6
- D = 6.38309e-8

La misura di temperatura prodotta dalla formula di Steinhart-Hart è in gradi Kelvin e non tiene conto degli effetti del fattore di dissipazione K del termistore, cioè dell'autoriscaldamento del componente dovuto al passaggio della corrente

tà a partire dalle letture del convertitore ADC.

TERMISTORE NTC

Il termistore NTC (Negative Temperature Coeffi-

Listato 1

```
//Include arduPi library
#include "arduPi.h"
#include "math.h"
// #include <Wire.h>
// Needed for serial communications (not needed in this case)
// SerialPi Serial;

// Needed for accessing GPIO (pinMode, digitalWrite, digitalRead, i2C functions)
WirePi Wire;

// Needed for SPI (not needed in this case)
// SPIPi SPI;

byte val_0 = 0;
byte val_1 = 0;
byte adc_address1 = 0x68; // Imposta indirizzo 00 ADC

byte adc_channel1 = 0x88; //
byte adc_channel2 = 0xA8; // Imposta indirizzi canali
byte adc_channel3 = 0xC8; // con risoluzione di 16 bit
byte adc_channel4 = 0xE8; //

double R1 = 22000.0; // Resistenza R1
double R2 = 22000.0; // Resistenza R2
double R_th0 = 10000.0; // Resistenza riferimento termistore a 25 c
double R_fr0 = 70000.0; // Resistenza fotoresistore con illuminamento IL unitario

double V_IN = 5.0; // Tensione riferimento misura ADC
double A = 0.00335402; // Steinhart-Hart Costante A
double B = 0.000256985; // Steinhart-Hart Costante B
double C = 2.62013e-6; // Steinhart-Hart Costante C
double D = 6.38309e-8; // Steinhart-Hart Costante D

double pB = 4100.0; // Costante parametro B

double K = 6; // Fattore dissipazione K 6mV C

double Pend = 0.7; // Valore Gamma fotoresistenza

int channel_0 = 0;
int channel_1 = 0;
int channel_2 = 0;
int channel_3 = 0;

double analog_0 = 0.0;
double analog_1 = 0.0;
double analog_2 = 0.0;
double analog_3 = 0.0;

// ===== VARIABILI DI SERVIZIO
double V = 0.0;
double R_th = 0.0;
double logR = 0.0;
double logR2 = 0.0;
double logR3 = 0.0;
double Stein = 0.0;
double Celsius = 0.0;
double dec_val = 0.0;

double RRo = 0.0;
double logRRo = 0.0;
double parB = 0.0;
double Celsius2 = 0.0;

double dec_val2 = 0.0;
double V2 = 0.0;
double R_lm = 0.0;
double Lux = 0.0;

// ===== VARIABILI DI SERVIZIO END

void setup()
{
    Wire.begin(); // join i2C bus (address optional for master)
}

void loop()
{
    // Inizio elaborazione

    // Lettura canale 1 ADC (channel 0) - due byte esadecimale
    // La prima istruzione imposta la richiesta di un singolo campionamento
    // La seconda istruzione attende per il tempo necessario ad ADC per campionare
    // La terza legge il buffer con il valore letto
```

e che alza, anche se di poco, la temperatura effettivamente misurata.

Il fattore di dissipazione è reperibile anch'esso nel datasheet e per il nostro termistore vale $K = 6$

Il valore in gradi da sottrarre al risultato della formula di Steinhart-Hart è dato dalla formula

$$Td = V * V / K * Rth$$

Ancora una considerazione: le formule appena descritte utilizzano nei calcoli il valore della resistenza che il termistore assume alla temperatura di misurazione, mentre all'ingresso del convertitore ADC noi invece leggiamo il valore della tensione ai capi dell'NTC, che è funzione della resistenza che l'NTC assume in base alla temperatura.

Sapendo che ciascuno "scalino" del valore letto dal convertitore ADC a 16 bit di precisione corrisponde una tensione di 6,5 microvolt, possiamo calcolare il valore di tensione corrispondente al valore decimale della lettura.

$V = \text{Valore ADC in decimale per } 0,65 \text{ microvolt}$

Per passare dal valore di tensione alla resistenza utilizziamo la formula di trasferimento:

$$Rth = (R1 * V) / (V_{IN} - V)$$

Nel **Listato 2** potete analizzare la conversione delle formule descritte nel programma in Python che descriviamo nel seguito dell'articolo. Nel caso in cui il datasheet del termistore in nostro possesso non ci fornisca i valori dei coefficienti da utilizzare con

(Continua)

Listato 1 (seguito)

la formula di Steinhart-Hart, molto probabilmente ci fornirà i valori per utilizzare un secondo metodo che si basa sul parametro B del termistore, un valore costante che è possibile utilizzare con l'equazione:

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \ln\left(\frac{R}{R_0}\right)$$

Il valore di temperatura ottenuto ha una precisione più grossolana rispetto al metodo precedente, e la formula produce risultati attendibili solo nell'intervallo di temperatura indicato nel datasheet. Il valore R_0 è il valore di resistenza del termistore alla temperatura di riferimento T_0 , in genere i soliti 25 gradi Centigradi. Anche questi valori si trovano nel datasheet, e nel nostro caso $R_0 = 10\text{k}\Omega$ alla temperatura T_0 di 298,15 K (25 C).

Nel **Listato 2** trovate la codifica in Python anche di questo metodo di calcolo.

FOTORESISTENZA

Passiamo ora a descrivere come misurare la luminosità in Lux utilizzando la fotoresistenza.

La fotoresistenza è un componente elettronico la cui resistenza è inversamente proporzionale all'intensità della luce che riceve. Dalle specifiche fornite dal costruttore del sensore, la resistenza per un illuminamento di 0 lux ha un valore almeno di 1 M Ω , mentre a 10 lux varia da 8 a 20 k Ω . Per comprendere meglio questi dati è utile introdurre la definizione di lux. Il lux (simbolo lx) è l'unità di misura dell'illuminamento adottata dal Sistema Internazionale. Un lux è pari a un lumen su metro quadrato; il lumen è l'unità di misura del flusso luminoso. Di seguito sono elencati alcuni esempi di

```
Wire.beginTransaction(adc_address1);
Wire.write(adc_channel1);
Wire.endTransmission();
delay(700);
Wire.requestFrom(adc_address1,2.0);

val_0 = Wire.read(); // receive high byte (overwrites previous reading)
// delay(200);
val_1 = Wire.read();
channel_0 = int(val_0) * 255.0 + int(val_1);
dec_val = channel_0;

// Trasformazione in tensione del valore letto
V = (dec_val * 62.5) / 1000000.0;

// Calcolo della resistenza
R_th = (R1 * V) / (V_IN - V);

// Calcolo dei gradi Kelvin con la formula di Steinhart-Hart
logR = log(R_th / R_th0);
logR2 = pow(logR, 2.0);
logR3 = pow(logR, 3.0);
Stein = 1.0 / (A + B * logR + C * logR2 + D * logR3);

// Conversione in gradi Celsius e applicazione fattore di dispersione
Celsius = Stein - 273.15 - pow(V, 2.0) / (K * R_th);

// Stampa del risultato
printf("Steinhart - Celsius = %f\n", Celsius);

// Calcolo dei gradi Kelvin con la formula del parametro B
RRo = (R_th / 10000.0);
logRRo = log(RRo);
parB = 1.0 / ((1.0 / 298.15) + (logRRo / pB));

// Conversione in gradi Celsius e applicazione fattore di dispersione
Celsius2 = parB - 273.15 - pow(V, 2.0) / (K * R_th);

// Stampa del risultato
printf("parametro B - Celsius = %f\n", Celsius2);

// Lettura canale 2 ADC - due byte esadecimale
// La prima istruzione imposta la richiesta di un singolo campionamento
// La seconda istruzione attende per il tempo necessario ad ADC per campionare
// La terza legge il buffer con il valore letto

Wire.beginTransaction(adc_address1);
Wire.write(adc_channel2);
Wire.endTransmission();
delay(700);
Wire.requestFrom(adc_address1,2.0);

val_0 = Wire.read(); // receive high byte (overwrites previous reading)
// delay(500);
val_1 = Wire.read();
channel_1 = int(val_0) * 255.0 + int(val_1);
dec_val2 = channel_1;
// Trasformazione in tensione del valore letto
V2 = (dec_val2 * 62.5) / 1000000.0;

// Calcolo della resistenza
R_lm = (R1 * V2) / (V_IN - V2);

// Calcolo della luminosità
Lux = pow((R_lm / R_fr0), (1.0/-Pnd));

// Stampa del risultato
printf("Lux = %f\n", Lux);

}

int main () {
int x = 1;
  setup();
  while(x == 1) {
    loop();
    x = 0;
    break;
    delay(5000);
  }
  return (0);
}
```

Istruzioni libreria Smbus

Di seguito il riassunto delle funzioni principali fornite dalla libreria smbus per il linguaggio Python.

All'inizio di ciascun programma che utilizza il bus i2C è necessario importare la libreria con l'istruzione.

```
from smbus import SMBus
```

Prima di configurare o accedere ad un device sul bus è necessario "aprire" il bus stesso creando un'istanza del bus con l'istruzione

```
bus = SMBus(1)
```

le istruzioni di comunicazione sul bus sono metodi (funzioni) richiamabili nella forma:

```
bus.read_byte_data(0x68,0x88)
```

Nella tabella che segue sono indicati i principali metodi utili per progetti come il nostro.

Metodo	Descrizione	Parametri
<code>read_byte(addr)</code>	Legge un byte	Indirizzo
<code>write_byte(addr,val)</code>	Scrive registro config.	Indirizzo, comando
<code>read_byte_data(addr,cmd)</code>	Legge un byte di dati	Indirizzo, comando
<code>write_byte_data(addr,cmd,val)</code>	Scrive un byte di dati	Indirizzo, comando, Valore
<code>read_word_data(addr,cmd)</code>	Legge una parola di dati	Indirizzo, comando
<code>write_word_data(addr,cmd,val)</code>	Scrive una parola di dati	Indirizzo, comando, Valore

illuminamento espressi in lux:

- la luce del sole mediamente varia tra i 32.000 lx (32 klx) e i 100.000 lx (100 klx);
- sotto i riflettori degli studi televisivi si hanno circa 1.000 lux (1 klx);
- in un ufficio luminoso si hanno circa 400 lx;
- la luce della luna è pari a circa 1 lx.

Nel datasheet del sensore si trova il grafico che esprime l'andamento della resistenza, espressa in kΩ, in funzione dell'illuminamento, espresso in lux (**Fig. 2**). Prestate attenzione al fatto che il grafico è in scala logaritmica e quindi l'equazione che lega luminosità e resistenza è l'equazione di una "retta" in forma logaritmica:

$$\log R = -a \log IL + \log R1$$

In questa formula:

- R = resistenza del fotorestore;
- a = pendenza della retta chiamata valore Gamma (fornito dal costruttore e nel nostro caso uguale a 0,7);

- IL = illuminamento espresso in lux;
- R1 = resistenza del fotorestore in condizioni di illuminamento IL unitario (ricavabile dal grafico e comunque nel nostro caso uguale a 70kΩ).

Da questa equazione è semplice ricavare il valore di illuminamento IL espresso in lux, in funzione degli altri dati. Si ottiene così:

$$\log \frac{R}{R1} = \log IL^{-a} \rightarrow IL = \sqrt[-a]{\frac{R}{R1}}$$

Questa è la formula che abbiamo utilizzato nel programma Python per ricavarvi il valore di illuminamento IL. Per eseguire tutti questi calcoli abbiamo preparato il programma in linguaggio Python visibile nel **Listato 2** e scaricabile dal sito della rivista www.elettronica.in.it, che ci permette di introdurre sia la modalità di utilizzo del bus i2C in un linguaggio di programmazione, sia le istruzioni Python utili per gestire stringhe di dati, conversioni di valori ed esecuzione dei calcoli che realizzano le

formule descritte in precedenza, ma utilizzabili in tutti i casi di lettura di valori provenienti da sensori.

Prima di iniziare dobbiamo scaricare una libreria che ci fornisca le funzioni per poter comunicare con il bus i2C. Ne esistono diverse, noi abbiamo scelto smbus che è più diretta da usare, potete anche sperimentare con Quick2Wire che adotta un approccio orientato alle classi ed alla programmazione ad oggetti. Aggiorniamo il dizionario dei pacchetti con i soliti comandi (**Fig. 3**):

```
apt-get clean  
apt-get update
```

Installiamo la libreria smbus (**Fig. 4**):

```
apt-get install python-smbus
```

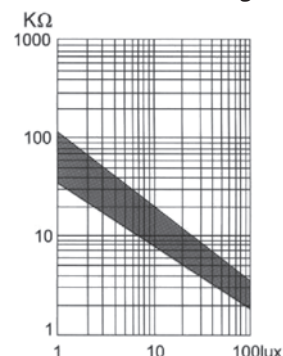
Guardando il **Listato 2**, all'inizio del programma troviamo gli include delle librerie che comprendono la libreria smbus per la gestione del bus i2C, la libreria time per la gestione degli intervalli di ritardo e la libreria math che contiene, tra le altre, la funzione logaritmo, necessaria per i nostri calcoli.

Nel listato abbiamo utilizzato il bus(1) che è il bus esposto sul connettore GPIO della rev. 2 di RaspberryPi. Se possedete RaspberryPi rev. 1, modificate l'istruzione

```
bus = SMBus(1)  
con  
bus = SMBus(0)
```

Il programma legge i valori di tensione in esadecimale dai canali analogici CH1 e CH2 ai

Fig. 2



Listato 2

quali sono collegati rispettivamente il termistore NTC e la fotoresistenza. I valori letti sono convertiti in decimale e poi ne viene calcolato il valore di tensione corrispondente.

Dal valore di tensione si calcola la resistenza sia per l'NTC che per la fotoresistenza e poi si applicano le formule che abbiamo descritto in precedenza per ottenere i valori di temperatura e luminosità. Infine vengono stampati a video i risultati. Per la logica del programma vi rimandiamo al **Listato 2** e ai commenti che abbiamo inserito pressoché ad ogni istruzione. Approfondiamo solamente le istruzioni di gestione del bus i2C fornite dalla libreria smbus, al quale abbiamo dedicato il riquadro "Istruzioni libreria smbus" e la gestione delle stringhe.

Nel **Listato 2** potete individuare il seguente blocco di istruzioni.

```
hex_val = "00" + hex_val
hex_hi = hex_val[-2:]
hex_low = hex_val[-4:-2]
hex_raw = hex_hi + hex_low
dec_val = int(hex_raw,16)
```

Il loro scopo è di invertire i due byte ottenuti dalla lettura del canale ADC, in quanto il byte meno significativo (low), arriva sempre prima del byte più significativo (high). Una volta ricostruito il valore esadecimale con i byte nell'ordine corretto, viene calcolato il valore decimale equivalente.

La prima istruzione aggiunge due zeri in testa alla stringa esadecimale, per assicurarsi che la stringa stessa sia lunga almeno quattro caratteri (due byte esadecimali).

Le istruzioni successive dividono la stringa in porzioni, in particolare vengono estratti il byte high con l'operatore [pos_i:pos_f] che

```
#!/usr/bin/env python
# Import librerie
from smbus import SMBus
import math
import time
# Definizione costanti
#bus = SMBus(0) # RaspberryPi rev 1
bus = SMBus(1) # RaspberryPi rev 2
adc_address1 = 0x68 # Imposta indirizzo 00 ADC
adc_channel1 = 0x88 #
adc_channel2 = 0xA8 # Imposta indirizzi canali
adc_channel3 = 0xC8 # con risoluzione di 16 bit
adc_channel4 = 0xE8 #
R1 = 22000.0 # Resistenza R1
R2 = 22000.0 # Resistenza R2
R_th0 = 10000.0 # Resistenza riferimento termistore a 25 c
R_fr0 = 70000.0 # Resistenza fotoresistore con illuminamento IL unitario
V_IN = 5.0 # Tensione riferimento misura ADC
A = 0.00335402 # Steinhart-Hart Costante A
B = 0.000256985 # Steinhart-Hart Costante B
C = 2.62013e-6 # Steinhart-Hart Costante C
D = 6.38309e-8 # Steinhart-Hart Costante D
pB = 4100.0 # Costante parametro B
K = 6 # Fattore dissipazione K 6mV C
Pend = 0.7 # Valore Gamma fotoresistenza
# Inizio elaborazione
# Lettura canale 1 ADC - due byte esadecimale
# La prima istruzione imposta la richiesta di un singolo campionamento
# La seconda istruzione attende per il tempo necessario ad ADC per campionare
# La terza legge il buffer con il valore letto
bus.write_byte(adc_address1,adc_channel1)
time.sleep(0.07)
raw_val = bus.read_word_data(adc_address1,adc_channel1)
# Riordino del byte low e high del valore letto e trasf. In decimale
hex_val = hex(raw_val)[2:].rstrip('L')
hex_val = "00" + hex_val
hex_hi = hex_val[-2:]
hex_low = hex_val[-4:-2]
hex_raw = hex_hi + hex_low
dec_val = int(hex_raw,16)
# Trasformazione in tensione del valore letto
V = (dec_val * 62.5) / 1000000.0
# Calcolo della resistenza
R_th = (R1 * V) / (V_IN - V)
# Calcolo dei gradi Kelvin con la formula di Steinhart-Hart
logR = math.log(R_th / R_th0)
logR2 = logR**2
logR3 = logR**3
Stein = 1.0 / (A + B * logR + C * logR2 + D * logR3)
# Conversione in gradi Celsius e applicazione fattore di dispersione
Celsius = Stein - 273.15 - V**2 / (K * R_th)
# Stampa del risultato
print "Steinhart - Celsius = ",
print Celsius
# Calcolo dei gradi Kelvin con la formula del parametro B
RRo = (R_th / 10000.0)
logRRo = math.log(RRo)
parB = 1.0 / ((1.0 / 298.15) + (logRRo / pB))
# Conversione in gradi Celsius e applicazione fattore di dispersione
Celsius2 = parB - 273.15 - V**2 / (K * R_th)
# Stampa del risultato
print "parametro B - Celsius = ",
print Celsius2
# Lettura canale 2 ADC - due byte esadecimale
# La prima istruzione imposta la richiesta di un singolo campionamento
# La seconda istruzione attende per il tempo necessario ad ADC per campionare
# La terza legge il buffer con il valore letto
bus.write_byte(adc_address1,adc_channel2)
time.sleep(0.07)
raw_val2 = bus.read_word_data(adc_address1,adc_channel2)
# Riordino del byte low e high del valore letto e trasf. In decimale
hex_val2 = hex(raw_val2)[2:].rstrip('L')
hex_val2 = "00" + hex_val2
hex_hi2 = hex_val2[-2:]
hex_low2 = hex_val2[-4:-2]
hex_raw2 = hex_hi2 + hex_low2
dec_val2 = int(hex_raw2,16)
# Trasformazione in tensione del valore letto
V2 = (dec_val2 * 62.5) / 1000000
# Calcolo della resistenza
R_lm = (R1 * V2) / (V_IN - V2)
# Calcolo della luminosità
Lux = (R_lm / R_fr0)**(1.0/-Pend)
# Stampa del risultato
print "Lux = ",
print Lux
```

```

192.168.0.162 - PuTTY
root@raspberrypi:~# apt-get clean
root@raspberrypi:~# apt-get update

```

Fig. 3

```

192.168.0.162 - PuTTY
root@raspberrypi:~# apt-get install python-smbus

```

Fig. 4

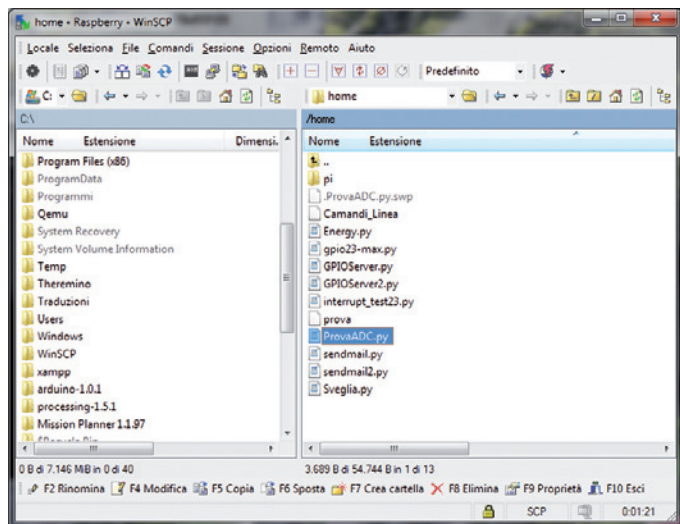


Fig. 5

```

192.168.0.162 - PuTTY
root@raspberrypi:~# modprobe i2c-dev
root@raspberrypi:~#

```

Fig. 6

```

192.168.0.162 - PuTTY
root@raspberrypi:~# python /home/ProvaADC.py
Steinhart Celcius - 22.5572726458
parametro B - Celsius - 22.6834833551
Lux - 175.952023202
root@raspberrypi:~#

```

Fig. 7

```

192.168.0.162 - PuTTY
root@raspberrypi:~# python /home/ProvaADC.py
Steinhart Celcius - 30.1969955354
parametro B - Celsius - 29.9397752651
Lux - 174.229899128
root@raspberrypi:~#

```

Fig. 8

permette di estrarre una porzione della stringa in base ai valori di posizione iniziale e finale della porzione desiderata. Per esempio l'istruzione della seconda riga estrae i caratteri dalla posizione -2 (dalla fine della stringa) alla fine, cioè gli ultimi due caratteri della stringa che rappresentano il byte più significativo della lettura (high). La terza riga estrae la porzione che rappresenta il byte meno significativo e che si trova davanti al precedente, cioè dalla posizione -4, sempre partendo dal fondo, alla posizione -2.

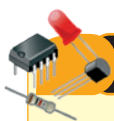
La quarta istruzione riordina i byte concatenando il byte high al byte low, mentre l'ultima istruzione converte il risultato finale nel corrispondente valore decimale. Per eseguire il programma basta copiarlo (o scriverlo) nella cartella /home (Fig. 5). Come ricordato nell'articolo precedente, se abbiamo appena acceso RaspberryPi dobbiamo dare anche il comando per caricare nel kernel il driver i2C (Fig. 6):

modprobe i2c-dev

Per eseguire il programma utilizziamo il comando (da root) (Fig. 7):

python /home/ProvaADC.py

per ottenere i risultati visibili sempre in Fig. 7. Dopo aver tenuto la resistenza NTC tra le dita per un po' di tempo proviamo a ridare il comando e vedremo la temperatura salire come in Fig. 8. Nei prossimi articoli vedremo come utilizzare programmi come questo per aggiornare un sito web, sempre ospitato su RaspberryPi, comunicando i valori letti via API HTTP.



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 programma di esempio che verrà presentato nel prossimo numero. Di questo shield Futura Elettronica ha realizzato la versione montata e collaudata (cod. FT1041M) disponibile al prezzo di 27 Euro, IVA compresa.

Il materiale va richiesto a:

Futura Elettronica, Via Adige 11, 21013 Gallarate (VA)
Tel: 0331-799775 - Fax: 0331-792287 - www.futurashop.it



MICROLOGIO: MINI OROLOGIO-SVEGLIA A MICROCONTROLLORE

Abbiamo presentato diversi progetti basati sul microcontrollore di Arduino, l'ATmega328P, sempre orientando gli articoli alla Didattica, quindi fornendo ampie spiegazioni sia sull'hardware, sia sul software. Anche questa volta presentiamo un progetto che sfrutteremo per farvi conoscere nuove tecniche hardware e nuove funzionalità software: il Micrologio, cioè un piccolo orologio/dattario da tavolo, dotato della funzione sveglia, che potrà tranquillamente sostituire ciò che avete attualmente sul comodino della vostra camera da letto.

Il Micrologio è un vero banco di prova su cui sperimentare diverse caratteristiche avanzate dei microcontrollori Atmel. Il progetto è nato per essere semplice e per utilizzare il minor numero di componenti esterni possibile; è stato quindi necessario studiare in modo approfondito il data-sheet del microcontrollore ATmega328P. Grazie al Micrologio potremo spiegarvi l'uso dei timer e del risparmio energetico estremo per ottenere consumi

bassissimi, oltre alle tecniche di multiplexing per pilotare in maniera diretta ben quattro display a 7 segmenti, senza ricorrere a componenti esterni.

LO SCHEMA ELETTRICO

Iniziamo con l'analisi dello schema elettrico. Il cuore del circuito è naturalmente il microcontrollore ATmega328P - nella

tipica configurazione stand-alone - impostato in modo differente da come viene usato su Arduino dato che i fuse sono settati con questi valori: low = \$62, high = \$DF, extended = \$07, da cui deriva la predisposizione del microcontrollore per lavorare con l'oscillatore interno ad 1 MHz. Nello schema però è visibile il quarzo esterno da 32.768 Hz

(Q1), collegato ai soliti pin 9 e 10 del micro; come sarà meglio specificato nella sezione dedicata al software, tale quarzo, il tipico modello a cilindro usato nei comuni orologi da tavolo o parete, servirà per ricavare dal micro una precisa frequenza di 1 Hz, indispensabile per il conteggio dei secondi e quindi per il funzionamento dell'in-

a cura del Prof. MICHELE MENNITI
e di LEONARDO MILIANI



3:33

Piccolo orologio/datario da tavolo, dotato della funzione sveglia, che potrà tranquillamente sostituire ciò che avete attualmente sul comodino della vostra camera da letto. Il Micrologio è un vero banco di prova sui cui sperimentare diverse caratteristiche avanzate dei microcontrollori Atmel.

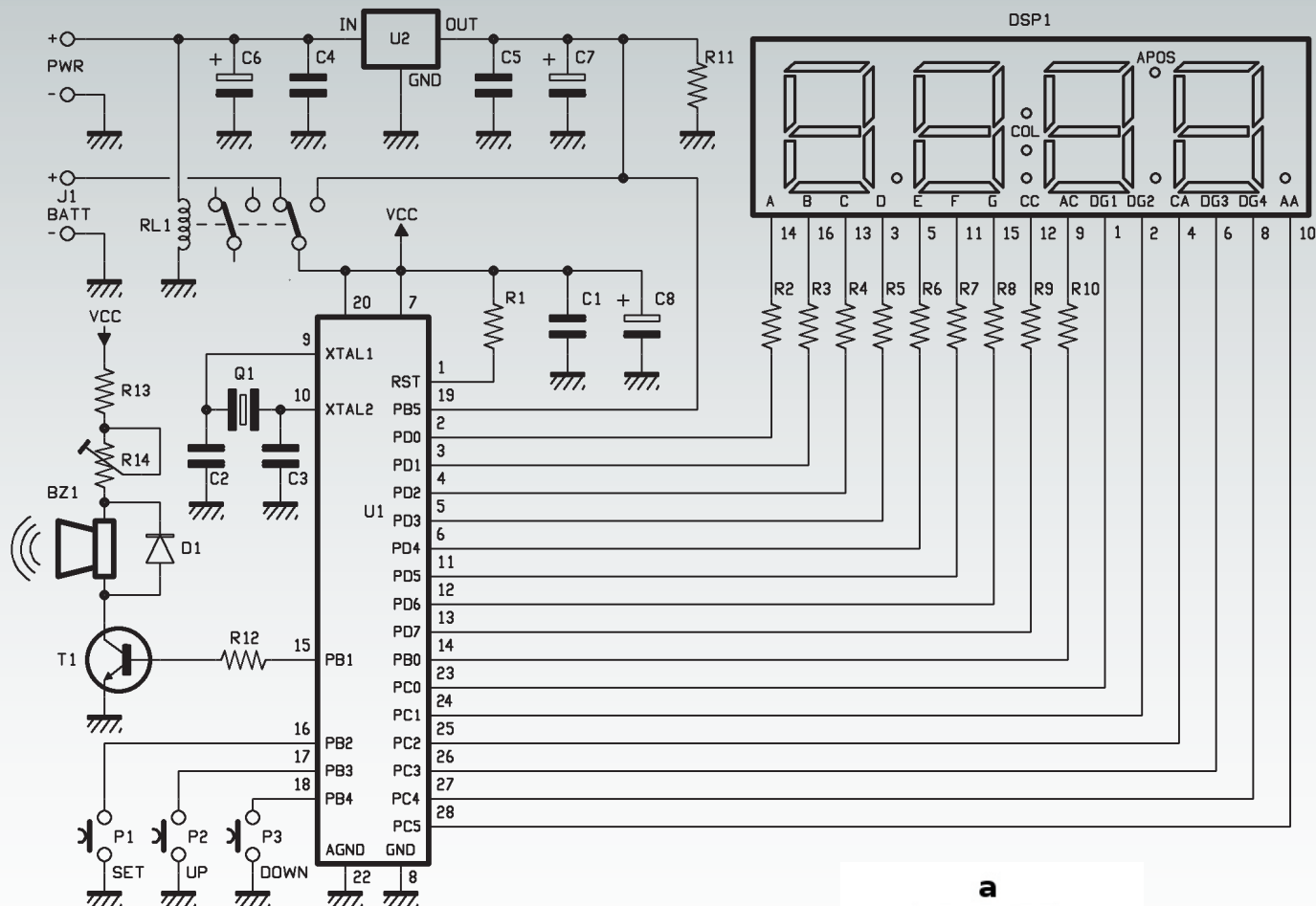
definire correttamente i segmenti esiste una denominazione standard che assegna ad ognuno di essi una lettera da "a" a "g". Il punto decimale viene chiamato indifferentemente "h" o "dp". (Fig. 1)

In base alla disposizione elettronica dei LED, i display vengono definiti a catodo o ad anodo comune: nel primo caso tutti i catodi dei LED sono collegati fisicamente tra loro su un pin del display (det-

to catodo comune), che va collegato al negativo di alimentazione; per accendere un segmento basta collegargli una tensione positiva, mediante opportuna resistenza di limitazione. Viceversa il display ad anodo comune richiede il collegamento dell'anodo al positivo di alimentazione, e per accendere un segmento bisogna collegarlo al negativo dell'alimenta-

tero Micrologio. Vediamo la gestione del display. Come è noto un display a 7 segmenti è costituito da altrettanti LED che, accesi secondo diverse combinazioni, compongono cifre o lettere. Spesso sono presenti altri elementi, in genere puntiformi, costituiti sempre da LED, con funzione di punto decimale o altro. Per poter





zione, sempre mediante resistenza di limitazione. Nel nostro progetto abbiamo in realtà usato un display multiplo a quattro cifre ed anodo comune che, oltre ai normali segmenti, con i quali visualizzeremo le cifre dell'orario e le lettere della programmazione, possiede due "simboli" che si prestano benissimo all'uso del nostro Micrologio. Per gestire in modo agevole lo stato dei sette segmenti (anche questo aspetto sarà approfondito nella sezione software), abbiamo collegato gli anodi dei relativi LED delle quattro cifre (pin 1, 2, 6 e 8 del display DSP1) ai pin di una stessa porta logica del microcontrollore (pin PC0, PC1, PC3 e PC4 del micro U1). Sempre alla stessa porta (pin PC2 e PC5 di U1) sono collegati rispettivamente: l'anodo del LED COL (pin 4 di DSP1), che gestisce il simbolo ":", che

separa prima e seconda cifra da terza e quarta cifra del display, e l'anodo del LED APOS (pin 10 di DSP1), che gestisce il "." che separa in alto terza e quarta cifra. I catodi dei LED dei sette segmenti (pin 14, 16, 13, 3, 5, 11, 15 di DSP1) ed il catodo del LED COL (pin 12 di DSP1) sono collegati alla porta D (da PD0 a PD7 di U1), mentre il rimanente catodo del LED APOS (pin 9 di DSP1) è collegato al pin PB0 di U1.

Una porta logica è costituita da un gruppo di piedini esterni mappato dal microcontrollore su appositi registri interni che ne indicano lo stato. Questi registri sono ad 8 bit, per cui ogni porta logica raggruppa otto pin fisici ed è identificata da una lettera dell'alfabeto, mentre i relativi otto pin sono indicati con i numeri da 0 a 7. P.es., osservando la

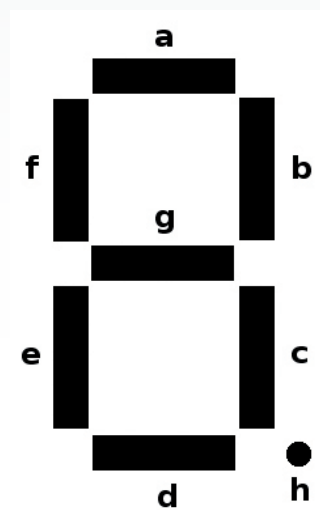


Fig. 1
La denominazione dei segmenti di un display.

Fig. 2, i pin indicati con PB1, PB2, PC0, PD3 rappresentano rispettivamente: il pin 1 della porta B, il pin 2 della porta B, il pin 0 della porta C, il pin 3 della porta D. In serie ad ogni catodo è stata posta una resistenza di limitazione da 22 ohm (da R2 a R10) che, nello specifico, visto l'utilizzo in multiplexing del display, potrebbe apparire sprecata; in realtà lo

scopo è più di protezione contro eventuali guasti che di limitazione, in quanto la corrente che scorre in esse a regime è molto inferiore a quella che farebbero comunque passare. Infatti ogni segmento, durante l'alimentazione a rete elettrica, assorbe circa 60 μ A, per scendere a 50 μ A con l'alimentazione a batterie.

Nelle normali condizioni di funzionamento a corrente elettrica, l'intero circuito assorbe mediamente 34 mA, mentre durante l'alimentazione a batterie l'assorbimento scende a circa 2 mA per "precipitare" a 85 μ A nella condizione di riposo (Micrologio funzionante ma display spento). Tali consumi aumentano, nella fase di funzionamento della sveglia, a 47 mA (rete elettrica) e 11 mA (batterie); questi ultimi due valori sono "di picco" in quanto dipendono dal pilotaggio del buzzer: in alcuni momenti scendono notevolmente e ciò incide positivamente sulla durata della batteria.

Sui pin 16, 17 e 18 del micro sono collegati i tre pulsanti P1, P2 e P3, denominati rispettivamente: SET, UP, DOWN, che servono per regolare l'orologio ed impostare la sveglia. Su tali pin non sono presenti le canoniche resistenze di pull-up in quanto abbiamo scelto di attivare via software quelle interne al microcontrollore, visto che questa funzionalità ci permette di risparmiare componenti esterni.

Ora vediamo il circuito audio: quello che vi permetterà di ascoltare i toni della sveglia al momento in cui si attiverà l'allarme impostato. L'allarme sveglia è costituito da due segnali a frequenza diversa che si alternano, per un periodo di tempo preimpostato (v. sezione software), sul pin 15 di U1. Tali segnali, mediante R12 piloteranno il

transistor TR1, usato in modalità di amplificatore in corrente: in tal modo si potrà disporre di un segnale molto potente senza sovraccaricare il pin del micro. Sul collettore di TR1 si trova collegato l'altoparlante BZ1, con il trimmer R14 che svolge la funzione di regolatore del volume. Abbiamo previsto R13 come limitatore; infatti il trimmer viene usato come reostato (resistenza variabile) quindi ruotando il cursore tra gli estremi la sua resistenza vale da 500 ohm a 0 ohm, ed in quest'ultima situazione non ci sarebbe limitazione alcuna, col rischio di bruciare l'altoparlante. A proposito di questo componente la scelta è vasta: potete usare indifferentemente un comune altoparlante miniaturizzato, un cicalino o un buzzer passivo; da evitare invece i buzzer attivi in quanto emettono una frequenza propria appena vengono alimentati e tale frequenza si sovrapporrebbe a quelle emesse dal micro, generando un rumore non molto piacevole per svegliarsi la mattina.

Il diodo D1 serve per proteggere il transistor TR1 nel caso si usi un altoparlante, mentre si può omettere per cicalini e buzzer di

piccole dimensioni.

Concludiamo la descrizione dello schema elettrico con la descrizione del funzionamento della sezione di alimentazione. Il connettore PWR prevede l'ingresso di una alimentazione a 5 V che viene ridotta a 3,3 V e stabilizzata dal regolatore U2, la cui uscita va al pin 19 del micro U1 ed al contatto 5 del relé reed RL1. I 5 V in ingresso eccitano il relé, che collega i contatti 5 e 6 e permette di far arrivare all'intero circuito la tensione di 3,3 V (VCC). Il

pin 19 del micro serve appunto per controllare la presenza della tensione di rete; quando è sullo stato logico HIGH esso comanda l'attivazione, come vedremo nella sezione software, della **modalità 0** del multiplexing, inoltre il display viene mantenuto stabilmente acceso; in queste condizioni, infatti, possiamo evitare di preoccuparci di qualche milliamperere di consumo in più.

Il connettore J1 prevede il collegamento di una coppia di batterie AA da 1,5 V, collegate in serie per generare 3 V, con funzione di batteria tampone o di alimentazione principale. Appena viene meno la tensione di 5 V (mancanza di corrente elettrica, distacco

(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

Fig. 2
La piedinatura del
microcontrollore
ATmega328P.

I Timer

I timer sono delle periferiche interne ai microcontrollori che aggiornano un contatore, ossia un registro, in maniera periodica e del tutto indipendente dalla CPU. I contatori sono dei registri di memoria e come tali possono avere diverse dimensioni, a seconda di quanti byte vengono utilizzati: sui microcontrollori Atmel della famiglia ATmega questi registri sono a 8 bit, quindi 1 byte, oppure a 16 bit, quindi 2 byte. L'ATmega328P, che abbiamo utilizzato per questo progetto, contiene 3 timer: due a 8 bit, denominati rispettivamente timer 0 e timer 2, e uno a 16 bit, denominato timer 1.

Come detto, un timer vede il suo contatore aggiornato in maniera indipendente dalla CPU grazie a dei circuiti dedicati: questi circuiti aggiornano il contenuto del registro ad intervalli pilotati da un segnale di clock. Normalmente il segnale di clock è prelevato dal clock stesso del sistema, raggiungendo una frequenza di aggiornamento pari a quella a cui lavora il microcontrollore. In determinati tipi di applicazione, questa frequenza può risultare eccessiva, per cui si raggiunge troppo rapidamente il massimo valore memorizzabile nel registro, dando origine al cosiddetto overflow di registro, ossia quella particolare condizione in cui il valore del registro riparte da zero avendo raggiunto la massima capienza. In questo caso si utilizza il prescaler, ossia un circuito che si occupa di dividere il segnale di clock in ingresso per ottenerne uno con una frequenza minore mediante dei rapporti predefiniti. Ad esempio, un fattore di prescaler pari a /8 significa che il timer lavorerà con un clock pari ad 1/8 di quello ricevuto in ingresso. I timer hanno diverse modalità operative: essi possono operare come semplici contatori oppure possono generare segnali PWM utili, ad esempio, per simulare una tensione analogica (questa modalità viene utilizzata da Arduino nella funzione `analogWrite()`); oppure possono generare segnali compresi in un ampio intervallo di frequenza. Nel primo caso il timer aggiorna continuamente il valore del proprio contatore; nel secondo caso il timer può essere agganciato ad alcuni pin esterni del microcontrollore (prefissati dal costruttore e riportati sul data-sheet) in modo da cambiarne lo stato al verificarsi di particolari eventi, quali ad esempio il raggiungimento di un valore prefissato o il suo valore massimo. Le modalità di generazione del segnale PWM sono diverse, permettendo all'utente di controllare la frequenza e il duty cycle dell'onda quadra generata. Nel Micrologio utilizziamo il timer 2 in modalità contatore per aggiornare una serie di variabili con le quali mantenere in memoria un orologio software e il timer 1 in modalità Phase Correct PWM per generare le frequenze usate per la funzione sveglia.

dell'alimentazione esterna, ecc.) il relé reed RL1 va in posizione di riposo e questa volta vengono collegati i contatti 6 e 7; in tal modo il circuito viene alimentato dai 3 V delle batterie. Questo provoca due diverse reazioni da parte del circuito: la prima è una leggera diminuzione della luminosità del display, a motivo della differenza di 0,3 V tra l'alimentazione a corrente elettrica e quella a batterie; la seconda è che in questa modalità il pin 19 non riceve alcuna tensione e passa allo stato LOW (in funzione della resistenza di pull-down R11); in questa condizione il software attiva la **modalità 1** del multiplexing ed ogni possibile accorgimento per il risparmio energetico.

C'è da sottolineare che il circuito non prevede la ricarica delle batterie, quindi i due sistemi di alimentazione sono indipendenti; le batterie devono essere necessariamente alcaline e di tipo non ricaricabile; il motivo è sia tecnico, sia meccanico. Le batterie ricaricabili infatti hanno una tensione nominale di 1,25 V, quindi una coppia non sarebbe bastata ad alimentare il circuito, ed una terza batteria avrebbe creato grosse difficoltà meccaniche nel contenitore. Peraltro il limitatissimo consumo del circuito fa sì che una coppia di batterie AA alcaline garantirà un'autonomia tale da non far rimpiangere un sistema ricaricabile.

Osservando lo schema elettrico con attenzione avrete notato che sui pin di alimentazione del micro c'è il condensatore C8 da 100 µF, apparentemente inutile, visto che anche sull'uscita del regolatore U2 ce n'è uno identico (C7). La funzione di C7 è quella tipica di serbatoio per una migliore stabilizzazione nei picchi di assorbimento, mentre la funzio-

ne di C8 è quella di una sorta di mini-tampone. Infatti, per quanto velocissimo, il relé reed ha un tempo di commutazione tale che nella fase di passaggio da uno stato all'altro lascerebbe il micro senza alimentazione, provocandone il reset o addirittura il blocco. Nello schema pratico potrete verificare che abbiamo previsto il montaggio di tale condensatore vicinissimo ai pin di alimentazione del micro. In tal modo, nel momento in cui viene scollegata l'alimentazione (passaggio da corrente elettrica a batterie o viceversa) è il C8 che garantisce l'alimentazione al circuito per i pochi istanti necessari.

L'importanza della presenza di C7 si riscontra nel momento in cui avviene il passaggio tra batterie e rete elettrica: il C8, quasi completamente scarico si comporta come una resistenza di bassissimo valore applicata al regolatore finché non si ricarica; in queste condizioni il regolatore potrebbe entrare in istante in protezione ed il micro non verrebbe alimentato correttamente. Invece sarà il condensatore C7 a ricaricare C8, mentre il regolatore garantirà l'erogazione della tensione corretta.

Ricorrendo alla coppia di batterie AA, che troveranno alloggio nel contenitore che abbiamo adottato per il Micrologio, l'autonomia sarà comunque elevatissima; si potrebbe anche fare a meno della alimentazione esterna dato che una coppia di batterie del tipo alcalino dovrebbe garantire circa 1 anno di autonomia, facendo un uso normale del Micrologio, quindi a voi la scelta!

FIRMWARE

Analizziamo ora il firmware che può essere scaricato dal sito della rivista e che non pubblichiamo a causa della sua lunghezza. Dopo

i flag di compilazione, che servono per l'eventuale attivazione di alcune funzioni del Micrologio e dei relativi menu, troviamo la definizione dei valori per ottenere le frequenze dei toni dell'allarme. Questi valori sono passati ad un timer interno al microcontrollore, al fine di generare il segnale PWM (Pulse-Width Modulation, o Modulazione di larghezza d'impulso) necessario per pilotare l'altoparlante. I timer, o temporizzatori (vedi **BOX** di approfondimento), sono tra le numerose periferiche integrate in un microcontrollore, che è un sistema digitale complesso costituito da diversi componenti. L'ATmega328P contiene tre timer: il primo, con registro a 8 bit, è denominato timer 0 e viene utilizzato dal core Arduino per la gestione delle funzioni temporali (ad esempio, le funzioni *delay* e *millis*) mentre gli altri due, disponibili per i nostri usi, sono il timer 1 (con registro a 16 bit) e il timer 2 (con registro a 8 bit). Il Micrologio usa il timer 1 come generatore di frequenza, per dotarsi della funzionalità sonora. A seconda della modalità utilizzata, cambia il modo per calcolare la frequenza ottenuta. Il valore della frequenza generata è strettamente dipendente dalla modalità di uso del Timer; nel nostro caso si tratta della *Phase Correct PWM*, (che permette di generare un segnale PWM più simmetrico rispetto ad altre modalità quali la Fast PWM) in cui il registro del timer viene incrementato fino a che il suo valore non raggiunge quello impostato da noi in un registro di confronto, chiamato OCR1A: al verificarsi di questa condizione avviene l'inversione ed il registro viene decrementato fino a che non torna a zero, raggiunto il quale

torna ad essere incrementato. Tutto ciò si ripete ciclicamente ed ogni volta che viene raggiunto il valore fissato da OCR1A avviene un cambio di stato logico del pin collegato al timer: esso passa quindi dallo stato LOW allo stato HIGH e viceversa, generando un'onda quadra la cui frequenza è inversamente proporzionale al tempo che intercorre fra due inversioni successive dello stato del pin. Il valore della frequenza generata si ottiene con la seguente formula:

$$F_{PWM} = F_{CLK} / (2 * prescaler * OCR1A * 2)$$

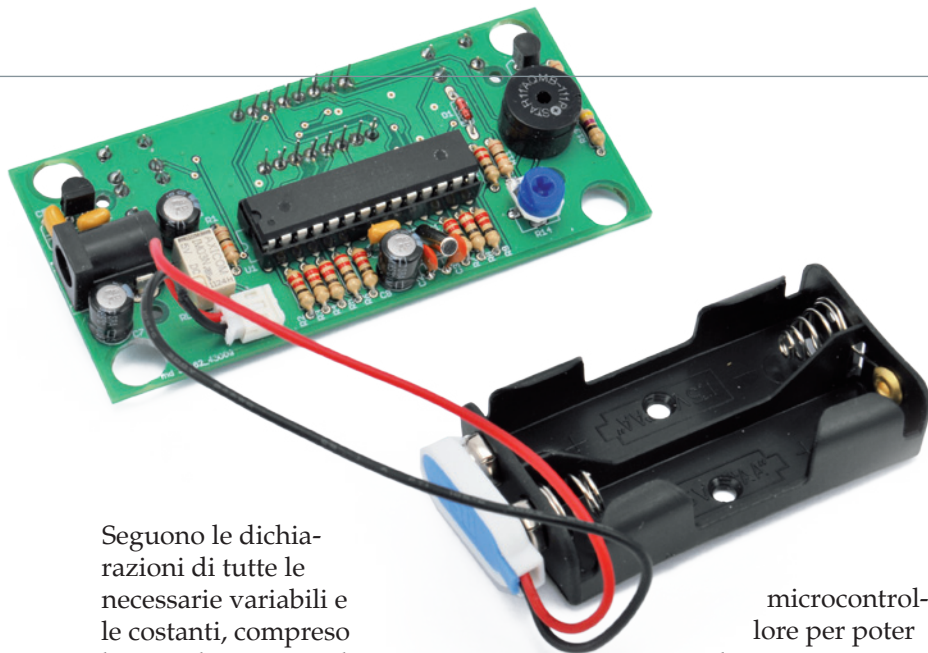
dove F_{PWM} è la frequenza sul pin in Hz, F_{CLK} è la frequenza in Hz del clock di sistema (nel nostro caso è 1.000.000 Hz), **prescaler** è il fattore di divisione del clock di sistema all'ingresso del timer (nel nostro caso è 1, a motivo del basso clock di sistema) e **OCR1A** è il valore del registro di confronto. Con i valori di OCR1A pari a 312 e 192 si ottengono, rispettivamente, le frequenze di ~800 Hz e ~1300 Hz, da noi proposte per la funzione "sveglia"; naturalmente potete cambiare tali valori a vostro piacimento, se preferite ottenere frequenze diverse. I timer si devono impostare manualmente manipolando i registri del microcontrollore secondo le modalità illustrate dalla scheda tecnica fornita dal costruttore. Vi consigliamo di studiare la funzione *attivaSuenaria* del nostro firmware per comprendere

come si imposta il timer 1. Nel nostro firmware sono incluse le seguenti librerie:

- **EEPROM.h**: contenuta nell'IDE, basta includerla e permette di accedere in lettura e scrittura alla memoria EEPROM interna;
- **sleep.h**: questa libreria fa parte delle librerie della toolchain AVR (che è un pacchetto di librerie e tool necessari a compilare programmi per i microcontrollori Atmel) distribuita con l'IDE e serve per gestire le modalità di risparmio energetico del microcontrollore;
- **PinChangeInt.h**: grazie a questa libreria è possibile attivare un interrupt di cambio di stato su un determinato piedino del microcontrollore.

Mentre le prime due librerie sono in dotazione all'IDE 1.0.x di Arduino (1.0.1 è la versione da noi usata per questo firmware), l'ultima deve essere scaricata dal seguente link: <http://code.google.com/p/arduino-pinchangeint/> e posizionata nella cartella delle librerie personali (ossia in /sketchbook/libraries oppure \Documenti\Arduino\libraries, a seconda del sistema operativo utilizzato) o in quella originale dell'IDE (arduino-1.0.x/libraries).





Seguono le dichiarazioni di tutte le necessarie variabili e le costanti, compreso l'array che contiene la mappatura dei segmenti che dovranno essere accesi o spenti a seconda del carattere alfanumerico da visualizzare sul display.

Nella funzione principale *setup* viene richiamata la funzione *clockOff*, che disattiva il display manipolando direttamente le porte logiche del microcontrollore. I pin del microcontrollore collegati agli anodi del display appartengono tutti alla porta logica PORTC così come i quattro catodi del display appartengono tutti alla porta logica PORTD. Questo espediente ci permette di impostare lo stato dei pin usando direttamente i registri del microcontrollore. Ad esempio, l'istruzione *PORTC = 0b00000000* imposta tutti i pin della porta logica C su "input". Dopo la funzione *clockOff* viene richiamata la funzione *impostaRisparmioEnergetico*, il cui scopo è appunto quello di massimizzare il risparmio energetico del microcontrollore spegnendo tutte le periferiche interne non utilizzate (ad esempio l'interfaccia seriale, quella per la comunicazione tramite I2C, il modulo ADC, o Analog-Digital Converter, per le letture analogiche). Come nel caso della manipolazione delle porte, ricorriamo all'utilizzo diretto dei registri del

microcontrollore per poter disattivare a nostro piacimento tutto quello che non ci serve.

A questo punto si impostano su INPUT i pin collegati ai pulsantini, attivando contestualmente la resistenza interna di pull-up, evitando così di doverle montare esternamente. Tale funzionalità è molto comoda e può tornare utile in molte situazioni; per ottenerla è sufficiente impostare su HIGH lo stato di un pin precedentemente attivato come INPUT. A partire dalla versione 1.0.1 dell'IDE di Arduino tale funzione può essere svolta semplicemente impostando il pin in modalità INPUT_PULLUP. Viene poi chiamata la funzione *caricaValoriPredefiniti*, che accede alla memoria EEPROM interna al microcontrollore e carica le impostazioni predefinite del Micrologio, se presenti: lo schema di multiplexing da utilizzare, il numero di secondi che devono trascorrere prima che venga spento il display, l'attivazione o meno dell'allarme ed altre impostazioni. Se esse non sono presenti significa che è la prima volta che si avvia il Micrologio e quindi tali informazioni vengono invece scritte nella EEPROM. Com'è noto la EEPROM è un tipo di memoria che mantiene il proprio contenuto memorizzato anche in assen-

za di alimentazione. Senza alcun dubbio la successiva funzione, *impostaTimer2*, è la più importante di tutto il firmware; infatti il timer 2 dell'ATmega328P ha la particolarità, a differenza degli altri due, di riuscire ad operare in maniera asincrona: può essere, cioè, pilotato da una fonte di clock esterna, diversa dal clock di sistema e, conseguentemente, funzionare anche quando il microcontrollore viene posto in stand-by. È proprio grazie a questa caratteristica che possiamo sfruttare una modalità di risparmio energetico molto potente, che ci permette di spegnere anche il generatore di clock di sistema e quindi tutte le periferiche ad esso collegate. In tali condizioni il micro arriva a consumare solo qualche microampere, mantenendo nel contempo attivo il computo del tempo. Tutto questo è possibile grazie al modulo RTC (Real Time Counter, ovvero "contatore in tempo reale") presente nell'ATmega328P. Collegando un quarzo esterno ed impostando opportunamente il timer 2, si può ottenere un incremento estremamente preciso del contatore interno. Il data-sheet dell'ATmega328P chiarisce che l'oscillatore interno del timer 2 è

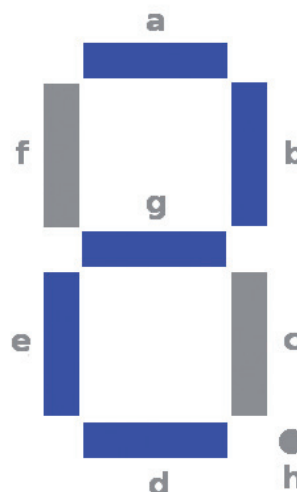


Fig. 3
I segmenti
del numero 2.

ottimizzato per l'uso con un cristallo esterno da 32.768 Hz, del tipo comunemente utilizzato nei cosiddetti "orologi al quarzo". Come già detto, il registro interno del timer 2 è a 8 bit e quindi può assumere fino a 256 valori. Per ottenere un periodo esatto di un secondo, che sarà la base dei tempi del nostro Micrologio, dobbiamo usare un prescaler (cioè un divisore di frequenza) con fattore pari a /128, quindi si avrà: $32.768/128/256=1$ Hz per cui tra un azzeramento del contatore e il successivo trascorre un secondo. Grazie a ciò possiamo realizzare un vero contatore in tempo reale: intercettando il segnale di interrupt che il timer genera quando azzerà il contatore e facendo eseguire un nostro algoritmo che incrementa delle variabili contenenti il valore delle ore, minuti, secondi e del giorno, mese, anno, possiamo ottenere un orologio in tempo reale completamente software, aggiornato dal timer 2. Inoltre, essendo l'esecuzione del nostro algoritmo indipendente dall'esecuzione del codice principale ed essendo il contatore aggiornato anche quando il microcontrollore è in stand-by (grazie alla modalità asincrona) l'orario del nostro orologio verrà aggiornato in maniera precisa e costante. L'ultima funzione del blocco di *setup* è *primoAvvio*, una routine che viene chiamata solo quando diamo per la prima volta alimentazione al circuito. Essa si occupa di inizializzare l'orologio software a 00:00 e di far lampeggiare il display fino a quando l'utente non preme per un certo lasso di tempo il pulsante "SET" per regolare l'orologio. Non appena si verifica questa condizione, viene chiamata la funzione *programClock*, che si occupa di interagire con l'utente per la

programmazione del Micrologio. Viene visualizzata la scritta "Prog" e, dopo qualche istante, si inizia la programmazione dell'orario, modificando prima il valore delle ore e poi quello dei minuti. Successivamente, se si è compilato il firmware abilitando la visualizzazione della data, si impostano il giorno, il mese e l'anno. La programmazione di questi dati avviene in maniera sequenziale: premendo il pulsante "UP" si incrementa il valore visualizzato, premendo il pulsante "DOWN" lo si decrementa. Quando il display mostra il valore corretto, bisogna premere il pulsante "SET" per memorizzarlo e passare alla regolazione del dato successivo. Terminata la fase di programmazione, vengono lette le impostazioni dell'allarme dalla EEPROM, se già in precedenza si era usato il Micrologio, quindi il controllo passa alla funzione *loop*. La funzione *loop*, per scelta degli sviluppatori di Arduino, viene eseguita ciclicamente dal microcontrollore (loop in inglese significa ciclo) senza fermarsi mai. La prima funzione chiamata è *controllaReteEsterna* per verificare se il Micrologio è connesso alla rete elettrica oppure è alimentato a batterie: nel primo caso il display viene visualizzato costantemente e lavora con una modalità di multiplexing che accende una intera cifra per volta; in tal modo i numeri appaiono più luminosi rispetto all'altro schema di multiplexing, più conservativo in termini energetici. In caso di alimentazione a batterie il display viene spento dopo un periodo di tempo prefissato detto "timeout" che, di default, è impostato a 5 secondi; inoltre, in questo caso, la modalità di multiplexing accende ogni volta

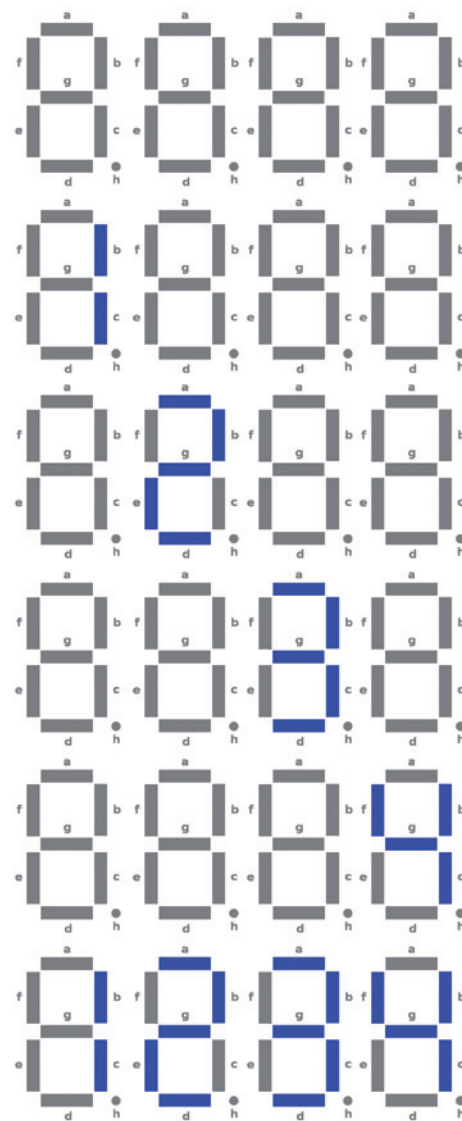


Fig. 4 - Il funzionamento della Modalità 0 del multiplexing

1 segmento per cifra; in tal modo il display è leggermente meno luminoso, ma il risparmio energetico è maggiore. Il *loop* prosegue controllando se la variabile globale *suoneriaAttiva* è impostata ad 1: in tal caso l'orologio interno ha raggiunto l'orario d'allarme impostato dall'utente e bisogna attivare la suoneria; di questa operazione si occupa la funzione *suonaAllarme*, che riattiva il timer 1 (normalmente disattivato per risparmiare energia) e lo imposta in modalità Phase Correct PWM, che ci servirà per generare i toni



Fig. 5 - Il funzionamento della Modalità 1 del multiplexing

dell'allarme. A questo punto la funzione inizia a suonare l'avviso acustico, costituito da una esecuzione alternata dei due toni predefiniti, secondo la seguente sequenza:

- suona per 2 minuti;
- si ferma per 8 minuti;
- riprende a suonare per altri 2 minuti;
- si ferma ancora per 8 minuti;
- suona nuovamente, e per l'ultima volta, per ulteriori 2 minuti.

Durante tale periodo il display lampeggia, fornendo in tal modo anche una sorta di "allarme visivo" nella fase in cui la suoneria è disattiva (la frequenza è differente rispetto a quella del display da programmare, per distinguere i due eventi). Se il circuito viene alimentato a batterie, il display lampeggia solo in concomitanza dell'avviso acustico, per un ulteriore risparmio energetico. La gestione dell'alternanza dei toni si ottiene modificando il valore del registro OCR1A che viene

usato per confrontare il valore del contatore ed invertire il livello sul piedino agganciato al timer 1 su cui si genera il segnale PWM. L'allarme cessa o perché si è raggiunta la fine della sequenza oppure perché l'utente ha premuto il pulsante "SET". In entrambi i casi il Micrologio torna alla normale funzionalità, uscendo dalla funzione *suonaAllarme*. Prima di uscire dalla funzione, viene controllato se l'allarme deve essere ripetuto oppure no: nel primo caso lo disattiva, nel secondo lo lascia attivo.

Successivamente viene controllato se il pulsante "SET" è premuto, nel caso viene chiamata la funzione *mainLoop*, che si occupa di accendere il display se questo è spento e di visualizzare l'orario e, eventualmente (in base alle opzioni attive), la data. Una leggera pressione del pulsante "SET" fa scorrere fra le varie voci mentre una pressione più lunga (1,5 secondi) fa entrare il Micrologio in modalità programmazione, mediante la chiamata alla

funzione *programClock*, già vista nel *setup*. Se l'utente, in fase di visualizzazione dell'ora e/o della data non tocca il pulsante "SET" per il tempo di timeout, il Micrologio torna alla visualizzazione dell'orario (se connesso ad un'alimentazione esterna) oppure rientra in stand-by (nel caso di alimentazione a batteria).

L'ultimo controllo che viene eseguito all'interno del *loop* è quello relativo alla pressione del pulsante "UP". Se l'utente preme questo pulsante durante il normale funzionamento dell'orologio (visualizzazione dell'orario con alimentazione esterna oppure stand-by con alimentazione a batterie) sul display compare l'orario della sveglia per tutto il tempo in cui il pulsante resta premuto. Al suo rilascio, l'orologio torna in modalità normale. Analizziamo adesso il modo in cui vengono gestite le informazioni per pilotare il display. L'array dichiarato ad inizio sketch contiene in forma di ottetto le informazioni per accendere o spegnere i sette segmenti ed il punto decimale, e comporre il numero o la lettera desiderata. L'ottetto rappresenta lo stato dei pin collegati ai catodi dei segmenti: un bit di valore 0 indica appunto che il relativo pin deve essere impostato su LOW (V-), mentre un bit di valore 1 indica che il relativo pin deve essere impostato su HIGH (V+). Poiché un pin a livello logico LOW è come se fosse collegato al negativo di alimentazione, il LED collegato ad esso si accenderà a motivo della differenza di potenziale presente tra anodo e catodo. Tale differenza di potenziale invece non sarà presente nei casi in cui i catodi siano collegati a pin posti a livello logico HIGH, il cui valore di tensione è prossimo a quello di alimentazione.

L'ottetto indica i segmenti partendo dal bit meno significativo (quello più a destra) per il segmento "a" per terminare al bit più significativo (quello più a sinistra) per il segmento "h", quindi la corrispondenza è: hgfedcba. Per esempio l'ottetto 10100100, corrispondente al numero "2", viene tradotto secondo il seguente schema: "a" acceso, "b" acceso, "c" spento, "d" acceso, "e" acceso, "f" spento, "g" acceso, "h" spento (Fig. 3).

Come già anticipato, per gestire più facilmente l'ottetto che indica lo stato degli otto segmenti, abbiamo collegato i catodi dei relativi LED ai pin di una stessa porta logica del microcontrollore. Per ogni porta vengono usati 3

registri a 8 bit che permettono di manipolarne lo stato:

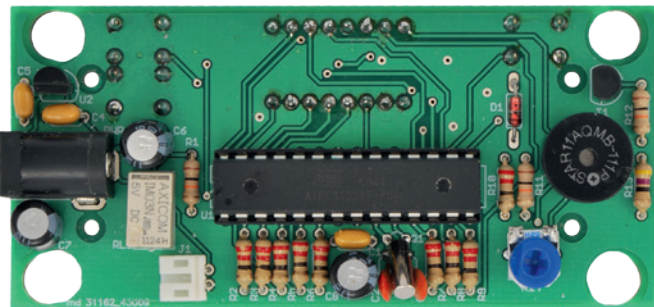
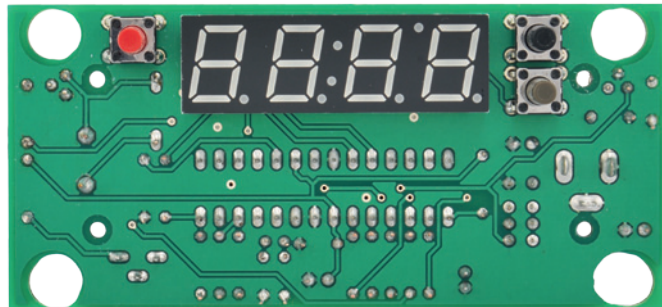
- **DDRx**: in cui "x" è rappresentato dalla lettera indicante la porta (nel nostro caso B, C o D), è il registro che indica la direzione del segnale dei pin della porta: un bit di valore 0 indica che un pin è usato come INPUT, un bit di valore 1 indica invece un pin usato come OUTPUT;
- **PORTx**: indica invece lo stato logico che i pin devono assumere: 0 indica lo stato logico LOW, 1 quello HIGH; entrambi questi registri sono accessibili sia in lettura, sia in scrittura;
- **PINx**: questo registro è accessibile solo in lettura e indica

lo stato logico presente sui pin, quando sono in modalità INPUT. Questo registro non verrà utilizzato dal firmware.

In realtà il punto decimale o segmento "h" non lo abbiamo usato per cui il bit più significativo sarà sempre impostato su "1" (spento).

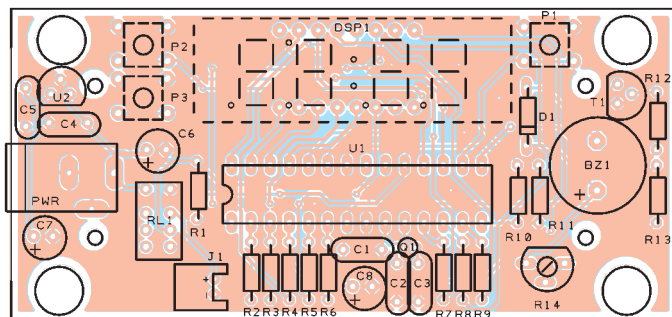
Collegando i sette catodi del display a sette pin di una unica porta logica possiamo pilotare lo stato dei LED manipolando direttamente i registri e semplificando enormemente il software. È proprio ciò che abbiamo fatto con il Micrologio: i sette catodi del display sono stati collegati tutti alla porta D mentre gli anodi sono stati collegati tutti

[piano di **MONTAGGIO**]



Elenco Componenti:

R1: 10 kohm	C7: 100 µF 16 VL elettrolitico
R2÷R10: 22 ohm	C8: 100 µF 16 VL elettrolitico
R11: 10 kohm	D1: 1N4148
R12: 10 kohm	T1: BC547
R13: 47 ohm	U1: ATMEGA328P-PU (MF1049)
R14: Trimmer 500 ohm	U2: 78L033
C1: 100 nF 63 VL poliestere	DSP1: Display YSD 439AR6B-35
C2: 12 pF ceramico	Q1: Quarzo 32,768 kHz
C3: 12 pF ceramico	RL1: Relé reed 5V (G6JU-2P-Y)
C4: 100 nF 63 VL poliestere	P1: Microswitch 5 mm
C5: 100 nF 63 VL poliestere	P2: Microswitch 5 mm
C6: 100 µF 16 VL elettrolitico	



P3: Microswitch 5 mm
BZ1: Buzzer passivo
Varie:
- Plug alimentazione
- Contenitore G1068

- Porta batteria 2xAA
(BH321B)
- Connettore JST 2 poli
da CS
- Circuito stampato



Fig. 6
La mascherina del
contenitore dopo
la foratura.

alla porta C del microcontrollore, permettendoci di scrivere un codice più semplice e snello. A questo punto entra in gioco un fattore importante: la corrente necessaria ad accendere i vari segmenti delle 4 cifre del display. Vediamo il caso limite del numero "8" di una qualsiasi cifra: immaginando di limitare la corrente che scorre nei segmenti ai canonici 20 mA, per accendere tutti gli otto segmenti il pin del relativo anodo dovrebbe erogare 160 mA. Ma un pin del micro può erogare al massimo 40 mA, quindi si brucerebbe in brevissimo tempo. Avremmo potuto adottare una "batteria" di quattro transistor per l'amplificazione in corrente degli anodi, ma in realtà il problema non esiste. Infatti il display che abbiamo adottato è strutturato in maniera tale da dover essere pilotato necessariamente mediante una tecnica che permette di accendere l'intero display con pochissima corrente: il **"multiplexing"**.

Questa tecnica consiste, genericamente, nel trasmettere un certo numero di segnali, sequenzialmente e ciclicamente, attraverso un unico canale di trasmissione, a frequenza molto elevata. Peraltro il display che abbiamo usato ha gli anodi delle cifre separati, ma i catodi dei segmenti sono tutti in parallelo; ciò significa che se tutti gli anodi fossero attivati contemporaneamente, ogni cifra mostrerebbe lo stesso valore. Di conseguenza, per pilotare questo display bisogna alimentare solo

un anodo per volta (quindi una cifra) e fornire la combinazione voluta di catodi per visualizzare un dato valore sulla cifra attiva. Nel nostro caso abbiamo usato due diverse modalità di multiplexing:

- la **modalità 0** (attiva automaticamente quando il Micrologio è alimentato dalla rete elettrica) accende una cifra per volta, attivando solo il relativo anodo e i segmenti necessari. P.es. volendo mostrare sul display il numero globale 1234, bisogna attivare il solo anodo della prima cifra a sinistra ed i catodi a-b, poi si attiva il solo anodo della seconda cifra ed i catodi a-b-d-e-g, quindi si attiva il solo anodo della terza cifra ed i catodi a-b-c-d-g, infine si attiva il solo anodo della quarta cifra ed i catodi b-c-f-g; a questo punto ricomincia il ciclo. Per costruire le 4 cifre dell'orario quindi 4 passaggi. Per meglio comprendere questo meccanismo abbiamo preparato la **Fig. 4** in cui abbiamo simulato una sorta di rallentatore che ci permettesse di vedere ciò che succede nella realtà. Come potete vedere osservando le immagini dall'alto verso il basso, dallo stato spento si passa all'accensione della 1ª cifra (con le altre spente), quindi si spegne la 1ª e si accende la 2ª, e così via; tutto ciò avviene ad una velocità tale che, per effetto del fenomeno della persistenza visiva, il nostro occhio vede

tutte le cifre accese contemporaneamente (immagine più in basso della **Fig. 4**);

- la **modalità 1** (attiva automaticamente quando il Micrologio è alimentato a batterie) attiva un segmento, e poi cicla in sequenza tra gli anodi, ponendoli in ON o OFF, in base allo stato che deve avere quel segmento nelle quattro cifre; quindi passa al segmento successivo e ricomincia il ciclo degli anodi. Riprendendo l'esempio precedente (numero 1234), in questo caso vediamo il comportamento del solo segmento "a" che, nel numero 1234, è OFF, ON, ON, OFF rispettivamente per 1ª, 2ª, 3ª e 4ª cifra; quindi viene eseguito questo ciclo:
 - Anodo 1 acceso (gli altri spenti), segmento "a" disattivo, nella 1ª cifra il segmento non si vede;
 - Anodo 2 acceso (gli altri spenti), segmento "a" attivo, nella 2ª cifra il segmento si vede;
 - Anodo 3 acceso (gli altri spenti), segmento "a" attivo, nella 3ª cifra il segmento si vede;
 - Anodo 4 acceso, segmento "a" disattivo, nella 4ª cifra il segmento non si vede;
 - Ora inizia un nuovo ciclo per il segmento "b", quindi per "c", e così via, fino a ritornare ad "a".

In pratica per costruire le quattro cifre dell'orario, con questa modalità sono necessari ben 28 passaggi. Per ben comprendere anche questa modalità abbiamo realizzato la **Fig. 5**, però abbiamo accorpato i passaggi degli anodi, mostrando solo quelli dei catodi, quindi vedremo i quattro passaggi degli anodi raggruppati per ogni

catodo; la figura va osservata dall'alto verso il basso, prima la colonna a sinistra, poi quella a destra. Nella prima immagine si vedono accesi i catodi "a"; come spiegato in realtà si accende prima uno e poi l'altro, non sono contemporanei; tutti gli altri catodi sono spenti. Poi si accendono i catodi "b" e si spengono tutti gli altri, e così via fino ai catodi "g". Se provate a sovrapporre virtualmente le prime sette immagini della Fig. 5, vi accorgete che alla fine il risultato sarà quello dell'ottava immagine, quella con le cifre complete. Anche in questo caso i 28 passaggi sono così veloci che noi continuiamo a vedere le quattro cifre accese contemporaneamente. Entrambe queste tecniche fanno sì che il brevissimo tempo di attivazione dell'anodo lo metta in condizione di erogare una corrente minima, pur garantendo una buona visualizzazione della cifra. Infatti, poiché questa serie di operazioni viene eseguita a grande velocità, per effetto del fenomeno della persistenza visiva, noi vediamo sul display contemporaneamente le quattro cifre accese, cioè il numero "1234", mentre in realtà l'intero display sta mostrando una sequenza 1-2-3-4 accendendo una cifra per volta o addirittura, nel caso della modalità 1, un solo segmento di una sola cifra per volta. Dal punto di vista software tutto ciò è gestito dalla funzione *displayClock*, che accende i segmenti del display in base al carattere da visualizzare (sia esso una cifra o una lettera) sia in base alla modalità di multiplexing adottata. Concludiamo consigliando di analizzare il firmware e studiare tutte le funzioni descritte: il codice è ben documentato pro-

prio per permettere a chiunque di comprendere facilmente il funzionamento di ogni riga di codice.

IL MONTAGGIO DELLA SCHEDA E DEL CONTENITORE

Abbiamo disegnato il PCB in funzione del contenitore che abbiamo deciso di adottare per il nostro Micrologio: si tratta del modello **8220-G1068** (reperibile presso Futura Elettronica), realizzato in ABS antiurto, ideale per la nostra applicazione. Il contenitore permette anche di installare il portabatterie **8220-BH321B** (reperibile sempre presso Futura Elettronica), in grado di ospitare due batterie AA ad alta capacità, per chi decidesse di usare il Micrologio senza alimentatore esterno, oppure per chi volesse dotarlo di un efficiente sistema tampone.

Vedremo più avanti il montaggio nel contenitore plastico. Il montaggio dei componenti sul PCB sarà effettuato su entrambe le facce, in quanto sul lato inferiore abbiamo previsto di saldare tutto ciò che dovrà essere rivolto verso il frontalino del Micrologio. Potete iniziare dal lato superiore, montando prima le resistenze e i diodi, lo zoccolo del microcontrollore e quindi tutti i restanti componenti, compreso il buzzer, se decidete di montare il modello tipico da stampato. Nel caso in cui decidiate di alimentare il circuito esclusivamente a batterie, allora potete evitare di montare una serie di componenti: il connettore PWR, i condensatori C4, C5, C6 e C7, il relé reed RL1. Dovrete però ponticellare i pad destinati ad ospitare i pin 6 e 7 del relé reed.

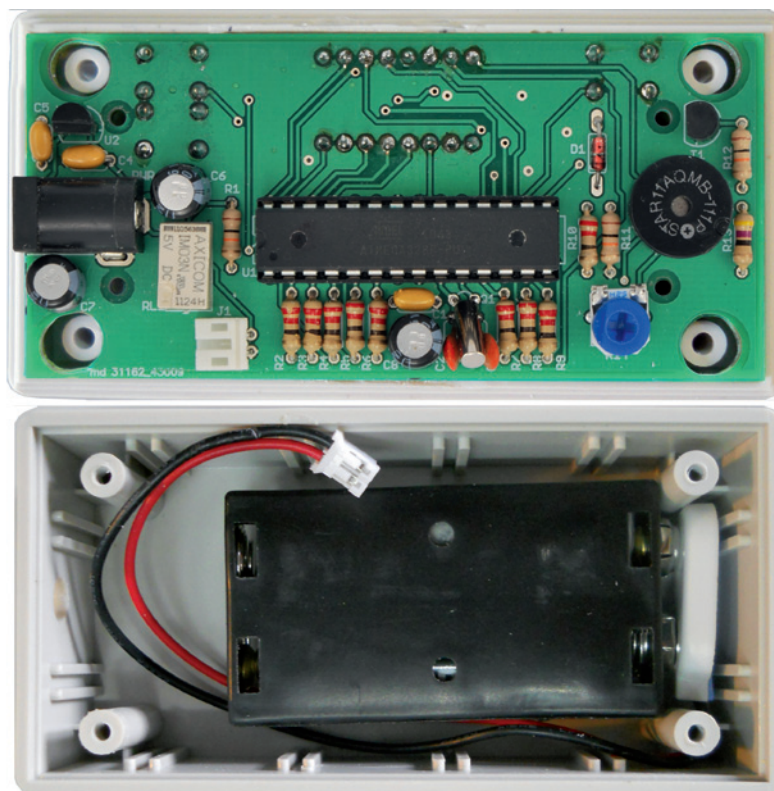


Fig. 7 - Il montaggio all'interno del contenitore.



Fig. 8
Il micrologio
in funzione.

L'operazione è resa semplice dal fatto che sulla serigrafia abbiamo aggiunto una piccola linea che unisce questi due pad: se si monta il relé sarà coperta da quest'ultimo, se non si monta sarà visibile e vi indicherà quali sono i due punti da unire. Se invece vorrete usare il Micrologio anche alimentandolo da rete elettrica, oltre a montare i componenti sopraelencati, dovrete prendere un alimentatore da 5 V a 9 V massimi; il basso drop-out del regolatore di tensione che abbiamo usato vi permette quindi di alimentare tutto con 5 V, evitando così aumenti di temperatura del regolatore; allo scopo andrà benissimo il modello **8400-AL08-5N** (reperibile presso Futura Elettronica), oppure un qualsiasi alimentatore da 5 V 0,5 A, purché con caratteristiche garantite. Sul lato inferiore andranno montati: il display, facendo molta attenzione al verso (fate riferimento alle varie Figure) ed i tre pulsanti N.A. (lunghezza dell'attuatore circa 5mm, per una giusta sporgenza fuori dal contenitore). Nel box del piano di montaggio potete vedere le due facce del PCB dopo il montaggio di tutti i componenti.

NOTA IMPORTANTE: a proposito del display è bene sapere che il modello da noi usato esiste in diverse colorazioni dei LED costituenti i segmenti: blu, verde, giallo e rosso.

Alimentando a 3-3,3 V il Micrologio solo le versioni a LED rossi o gialli risultano efficaci in termini di visibilità, mentre le altre due versioni (verde e blu) perdono molto in luminosità e contrasto, a motivo della elevata tensione inversa dei LED, quindi consigliamo decisamente di comprare il tipo con LED rossi o gialli (quest'ultimo è reperibile presso Futura Elettronica col codice **1446-DI-SPLAY4DGIALLO**).

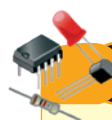
Il montaggio del PCB nel contenitore richiede un po' di pazienza e la giusta attenzione solo per quanto riguarda le forature: andranno infatti predisposti i fori per i tre pulsanti ed il display ed un foro, in un punto qualsiasi del contenitore per permettere al suono di uscire e diffondersi; bisognerà prevedere un foro laterale per la presa PWR, qualora l'abbiate montata. In **Fig. 6** la mascherina del contenitore dopo la foratura. In **Fig. 7** potete vedere come andranno alloggiati il PCB e il porta batterie nel contenitore: da notare sulla parete sinistra il foro per l'ingresso dell'eventuale spinotto dell'alimentatore esterno.

Prima di poter collocare il PCB nel contenitore resta ancora da fare la programmazione del microcontrollore; nella sezione download del sito www.elettronica.in troverete il firmware denominato Micrologio.ino che

dovrete caricare nel microcontrollore tramite la tecnica ISP che più volte abbiamo usato in progetti presentati sui numeri precedenti della Rivista.

Una volta fatta questa operazione, inserite il micro nello zoccolo (attenzione al verso!) e collegate la fonte di alimentazione che avete prescelto; se avete fatto tutto per bene il display si dovrà accendere mostrando "00:00" lampeggiante, ad indicare che il Micrologio è pronto per essere racchiuso nel contenitore, impostato ed entrare in funzione! In **Fig. 8** ecco il Micrologio già regolato e funzionante, il lampeggio dei soli ":" scandirà i secondi.

L'articolo termina qui in quanto abbiamo preferito creare un piccolo manuale d'uso in formato pdf, che potrà essere stampato e tenuto a portata di mano, per imparare ad usare rapidamente il Micrologio; troverete il manuale nella sezione download; esso risulterà molto utile anche nel caso in cui decidiate di realizzare il Micrologio e regalarlo a qualcuno.



per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono facilmente reperibili in commercio. Il master del circuito stampato ed il firmware col quale programmare il microcontrollore sono scaricabili gratuitamente dal sito della rivista (www.elettronica.in).

Il micro Atmel è anche disponibile già programmato (cod. MF1049) al prezzo di 12,00 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>

Mio nonno ha avuto una vita dura, scandita da due guerre mondiali, una dittatura, una vita in campagna, al gelo in inverno, sotto il sole torrido in estate (tranne qualche furioso temporale che poteva azzerare in pochi minuti il raccol-

to di un anno), niente riscaldamento, acqua dal pozzo e "bagno" in cortile. Tutto questo in un luogo che oggi è alla periferia di Milano, a pochi passi da dove ha sede l'Università Bicocca. Mio nonno si riteneva una persona fortunata perché ai suoi

tempi aveva avuto la possibilità di frequentare la scuola, dove si andava con gli zoccoli, un libro, un quaderno ed un pezzo di legno sotto il braccio, per la stufa. In quel luogo aveva appreso un bagaglio di conoscenze che gli hanno poi permesso

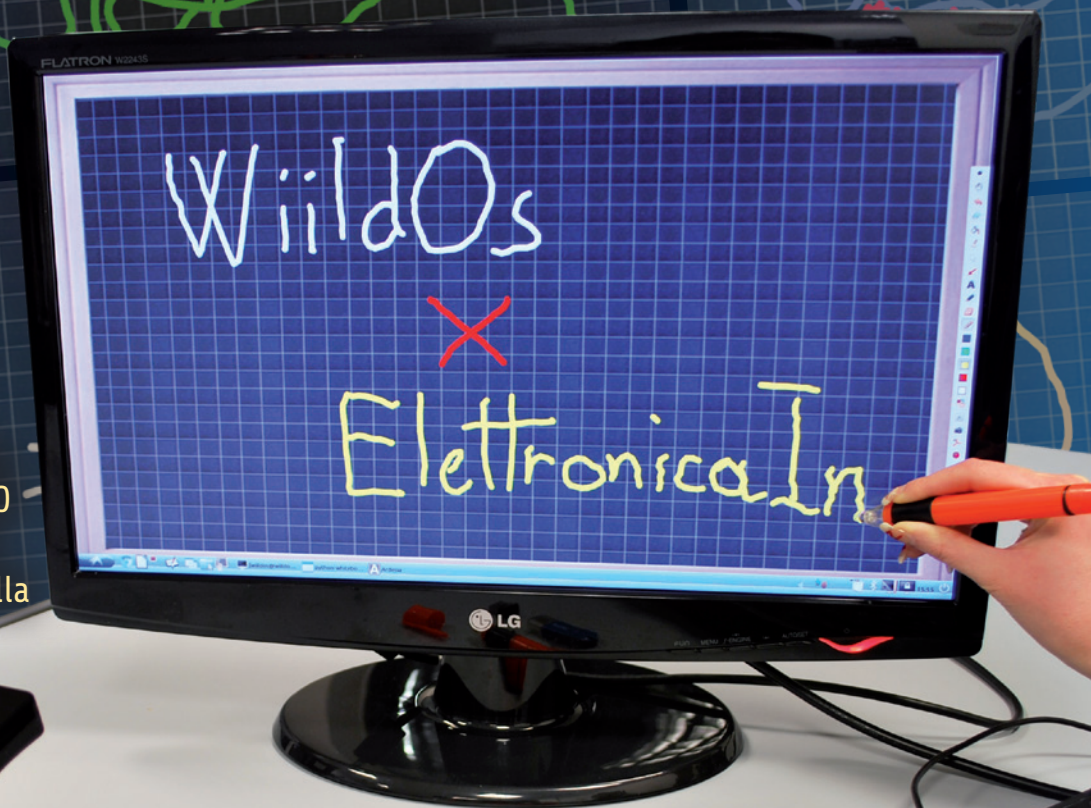
di analizzare gli eventi che hanno caratterizzato la sua vita, e di fare scelte consapevoli su come affrontarli, a volte in controtendenza rispetto alle masse. Tutto questo ascoltando una maestra e copiando da una lavagna. Due generazioni e mezzo secolo

Vado a scuola con la Wii

"DIETRO LA LAVAGNA!"

di MARCO MAGAGNIN

Vi presentiamo il progetto Open Source di lavagna interattiva WiildOS, sviluppato in ambiente GNU/Linux e frutto della passione e della fatica di un gruppo italiano. Per il suo uso vi proponiamo una penna a infrarosso (il gessetto della LIM) ed anche un supporto stampabile in 3D per il telecomando Wii, altro componente "essenziale" della soluzione WiildOS.





Progetto WiildOs

Durante l'anno 2008 Massimo Bosetti - collaboratore del Laboratorio di Comunicazione del Dipartimento di Scienze Fisiche dell'Università di Trento - e Matteo Ruffoni - docente di matematica - iniziano a visionare alcuni video su internet della Wii, in particolare il materiale prodotto da Johnny Lee Chung che dimostra tra i vari utilizzi il funzionamento della Wii come lavagna interattiva. Massimo e Matteo tramite una amica comune vengono a conoscenza che all'Hacklab j1 del Centro Sociale Bruno di Trento dove Pietro Pilolli assieme ad altri da mesi studia le tecnologie a infrarossi e il wiimote, provando e modificando i programmi trovati in rete e autocostruendo. Pietro, dottore informatico e programmatore affascinato dalle nuove tecnologie applicate a facilitare l'interazione uomo-macchina, convinto dal progetto, continua nello studio sulle tecnologie a basso costo che possono rendere facile e intuitivo l'utilizzo di programmi per computer, a fine anno abbandona, spinto da un moto di libertà il centro sociale. L'idea

di utilizzare le proprie conoscenze e applicarle alla didattica piace molto. Il primo gruppo di lavoro si è formato, sono in tre: Pietro, Matteo e Massimo. I tre, presi dalla passione comune, nel dicembre 2008 scrivono il primo progetto didattico "Lavagna Multiwii" e cercano senza successo finanziamenti privati presso banche della zona. Nonostante tutto, il gruppo non si perde d'animo e porta avanti lo sviluppo del progetto che ritengono essere il collante di un supporto didattico integrato. Non soddisfatti dall'utilizzo e integrazione di strumenti esistenti decidono per uno sviluppo specifico, che prende il nome di Ardesia, un annotatore della scrivania che

usa le librerie cairo.

Dopo tre mesi di lavoro Pietro viene contattato dal MIUR per scrivere un articolo su Ardesia. Il progetto è in uno stato pre-embriale ma solo l'idea di avere finalmente un programma del genere su Linux fa girare la testa. Decisamente meglio degli annotatori che si possono trovare gratuitamente per Windows.

"Si avevano a disposizione molti programmi educativi ma mancava un'applicazione che rendesse facile le più comuni operazioni che si fanno con una lavagna, come ad esempio, disegnare su una superficie a mano libera." racconta Pietro Pilolli in un'intervista rilasciata ad InnovaScuola nel 2009.

Ardesia nasce per essere un supporto all'insegnamento. Non ha nessun intento

di imporre nuovi metodi didattici e, proprio per questo motivo, non è dipendente da nessuna applicazione. Basta aprire un qualunque programma ed è possibile fissare i concetti e i commenti come se si scrivesse su una

classica lavagna. Tradizione e innovazione convivono nella semplicità. Il progetto globale viene chiamato, da Massimo "Wiild", ed è caratterizzato dalla disponibilità open source e dallo sviluppo cooperativo.

Nel 2010 nasce il sito dedicato alla Wiild e nel mese di marzo il progetto viene istituzionalizzato e formalizzato con l'incontro ufficiale al Palazzo dell'Istruzione a Trento. L'incontro mira a presentare il progetto nel dettaglio con particolare attenzione all'ambiente virtuale di condivisione, sia dei materiali, sia delle esperienze.

Pietro, dopo aver visionato vari programmi per lavagna wiimote, inizia a collaborare con Pere Negre, lo scrittore di python-whiteboard. L'usabilità e l'interfaccia del programma migliorano notevolmente, ed in poco tempo, grazie ai suggerimenti di Pietro e la disponibilità e la bravura di Pere. Nel gennaio 2011, Pietro, per migliorare il supporto ai docenti, che sembra che ne abbiano sempre una per cui il proprio computer non funziona, decide di creare una distribuzione live contenente tutto il necessario per fare una lezione con la Wii. Nasce WiildOs! Nel corso del 2011 Ardesia viene inclusa nelle distribuzioni ufficiali di Debian ed Ubuntu.



Fig. B



Fig. 1
Lavagna classica.

dopo, è stato il mio turno. I libri erano diventati due, i quaderni qualcuno in più. La maestra non era la stessa. La lavagna era ancora lì, al suo posto, assieme al calamaio per l'inchiostro. Già, la lavagna, pronta a ricevere l'ennesima riscrittura di lettere e numeri (niente grafica, allora) da parte dell'insegnante e rimbalzarli verso una moltitudine di alunni più o meno attenti e ricettivi (allora, rispetto ad oggi, gli alunni erano tanti e gli insegnati pochi). La lavagna, specchio e amplificatore di conoscenza. La "mia" lavagna era una presenza monumentale, con la struttura in legno massiccio, e con un meccanismo leonardesco che le permetteva di ruotare in modo da essere utilizzabile su entrambe le facce, con cassette e accessori in stile neoclassico. Era anche uno strumento punitivo, non ci voleva molto a finirci dietro, anche per una semplice marachella. Nel mezzo secolo successivo, il pianeta è stato stravolto totalmente, le lavagne sono cambiate solo di poco, la "mia", più o meno come quella visibile nella Fig. 1, era in formato 4:3, poi sono state appese al muro e diventate 16:9, da nere a verdi, poi bianche, da usare con i pennarelli (un disastro, chi lascia la strada vecchia per la nuova ...).

Però, ancora oggi, la prevalenza assoluta nel mondo, nelle scuole e università di ogni ordine e gra-

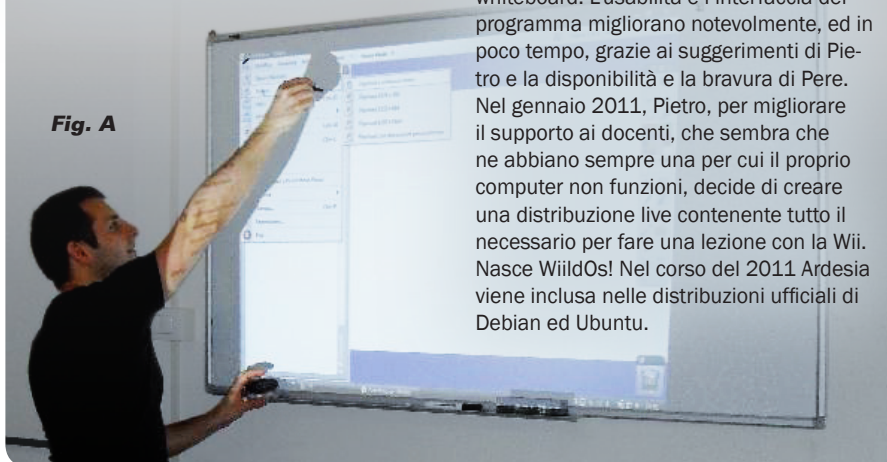
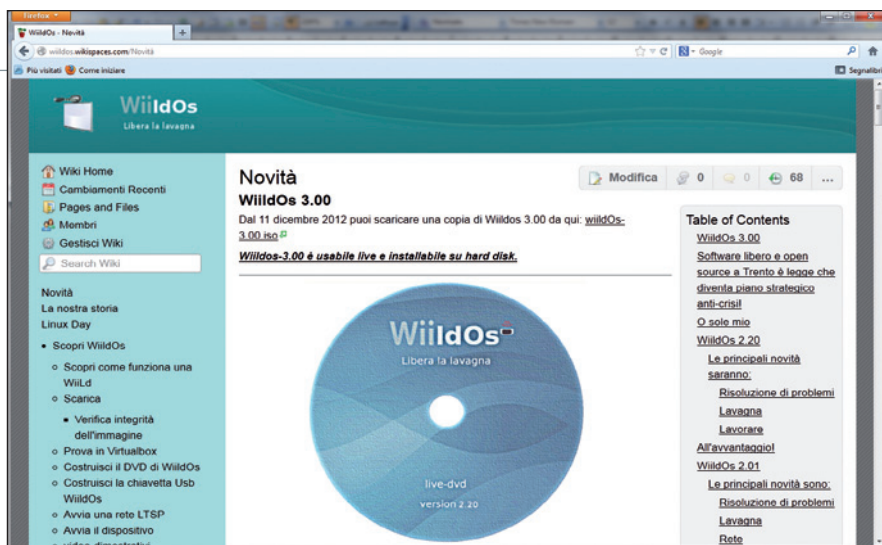


Fig. A

Fig. 3



do, latitudine e regime politico, rimane delle lavagne nere, da usare col gesso, e conseguente polvere a fine lezione, farina del sapere. Nella Fig. 2 un'aula della prestigiosa Università di Harvard.

Negli ultimi tempi la tecnologia sta tentando l'attacco alla supremazia della lavagna "classica", proponendo quelle che vengono definite LIM o lavagne interattive multimediali. Inizialmente non hanno trovato una grande diffusione a causa del loro costo elevato, ad una certa resistenza degli insegnanti a superare le difficoltà iniziali nell'utilizzo e alla necessità di riparametrare le lezioni (adesso si chiamano unità didattiche) per "adattarle" al nuovo strumento. In questo contesto socio-tecnologico un gruppo di coraggiosi "innovatori", noncuranti degli avvertimenti del Signor Niccolò Macchiavelli ("De Principatibus") hanno deciso di dar vita ad un progetto didatticamente innovativo, che organizza, struttura, documenta e divulga una ricca raccolta di prodotti, strumenti e supporti raggruppati in un unico "tool-box".

Si tratta del progetto Open Source WiildOs che, a detta degli autori, "non è un semplice programma per la gestione della tua lavagna interattiva multimediale, ma un nuovo sistema operativo modulare, semplice, completo,

Fig. 4



libero, open source e gratuito. È il sistema libero a disposizione di tutti gli utenti che intendono fare formazione nell'era della multimedialità, dei sistemi operativi e del cloud computing". Giunto alla versione 3.0 è disponibile sul sito <http://wiildos.wikispaces.com/> (vedi Fig. 3) dove sono presenti la documentazione, le istruzioni di utilizzo, i riferimenti ad altri progetti correlati, le esperienze degli utilizzatori e da dove è possibile scaricare il software, in forma di distribuzione GNU/Linux, pronta a funzionare. Nel riquadro dedicato trovate ulteriori informazioni sul progetto e sul gruppo che lo mantiene. Già, ma che c'entra con l'elettronica? E con la rivista Elettronica In? C'entra con entrambe, e per più di un motivo. Il progetto propone, e si avvale come supporto interattivo alla didattica (la lavagna), di una LIM auto costruita, totalmente open source, di basso

costo e di facile utilizzo. I componenti principali sono ovviamente un computer ed il relativo software, un proiettore (ma funziona anche con lo schermo del PC o un televisore), una penna a led infrarosso, un dongle Bluetooth e un ... controller WiiMote (Fig. 4).

FUNZIONAMENTO DELLA LIM

Nella Fig. 5 è visibile lo schema di funzionamento. Le immagini di qualunque applicazione su PC (dal browser web a Geogebra, il software libero per l'apprendimento e l'insegnamento della matematica) vengono proiettate sullo schermo, che può essere anche una semplice parete bianca. Il telecomando WiiMote, collegato al PC via connessione Bluetooth, collocato su un apposito supporto alla corretta distanza dalla proiezione, inquadra l'area di proiezione con la telecamera ad infrarossi incorporata. Al posto del mouse si utilizza la

Fig. 2 - Università di Harvard.



Fig. 5 - Schema di funzionamento.



penna a LED infrarosso per compiere le azioni di navigazione, direttamente sull'immagine proiettata. Quando viene premuto il pulsante sulla penna (che equivale al clic del mouse) viene emesso un fascio di luce infrarossa che, riflessa dalla superficie di proiezione, viene intercettata dalla telecamera del WiiMote. Il WiiMote trasmette le coordinate della posizione della traccia infrarossa via Bluetooth al software di gestione su PC. Di fatto si comporta come nel funzionamento normale, quando intercetta i LED della barra della Wii, che è ferma ed è il telecomando a muoversi. I programmi del pacchetto di gestione python-whiteboard (si, è scritto in Python e funziona anche su RaspberryPi) si occupa del riconoscimento automatico del telecomando, come dispositivo bluetooth, e poi, durante il normale funzionamento, trasforma le coordinate ricevute in azioni del mouse, che vengono passate alla applicazione attiva al momento. Per fare in modo che il tutto funzioni correttamente è necessario eseguire una calibrazione iniziale, che, come vedremo in seguito, si esegue "cliccando"

con la penna a infrarosso su quattro marcatori dell'apposito "foglio di calibrazione". Da questo momento si apre tutto un mondo di applicazioni e supporti didattici. L'elenco delle materie coperte è vastissimo, dalla grafica all'astronomia, dalla matematica al latino e, ovviamente, nulla vieta di aggiungere altri pacchetti o proprie "unità didattiche". Tra i supporti di corredo la tastiera "a schermo" utilizzabile con la penna a infrarosso come su un tablet e ... un programma di simulazione di lavagna "vera" (in questo caso si usa la "penna" al posto del gessetto), dal nome evocativo di "Ardesia", con lo sfondo personalizzabile e una nutrita serie di funzioni; è stata sviluppata da uno dei fondatori del gruppo. Veniamo ai motivi della presenza di questo articolo sulla rivista. Il primo motivo è che la rivista viene inviata in abbonamento gratuito a tutti gli Istituti Tecnici d'Italia: ci è sembrato opportuno segnalare l'esistenza di uno strumento didattico di questa portata, open source, facilmente implementabile e di bassissimo costo, il che in tempi di tagli non guasta.

Poi c'è la penna a LED infrarosso che è possibile procurarsi all'estero oppure auto costruirsi: per questo vi proponiamo la trasformazione in penna a infrarosso di un prodotto ultraeconomico disponibile presso Futura Elettronica.

Terzo, il supporto per il WiiMote, che possiamo costruire secondo le nostre esigenze utilizzando la nostra stampante 3Drag che è un altro strumento didattico dalle potenzialità ancora tutte da scoprire. Infine, beh, sì, la LIM funziona anche con RaspberryPi, anche se un po' a fatica. Con l'evoluzione in atto nel campo dei micro PC pensiamo però che entro la fine dell'anno saranno disponibili novità hardware (si parla anche di una nuova versione di RaspberryPi, chissà) che offriranno a prezzi accessibili la potenza di calcolo sufficiente a reggere applicazioni di questo tipo, con innegabili vantaggi nel mondo dell'istruzione e della formazione.

LA PENNA A LED INFRAROSSO

Come dice il nome, la penna a LED infrarosso è costituita da un diodo LED che emette un fascio di luce nello spettro degli infrarossi (circa 940nm) e che

Fig. 7



Fig. 6

viene alimentato a comando per mezzo di un pulsante, simulando il posizionamento e il clic del mouse. L'alimentazione proviene da una pila a stilo e il tutto è racchiuso in un contenitore simile a un pennarello. Esistono alcuni prodotti commerciali disponibili all'estero ma, per acquisti singoli, il costo della spedizione supera quello del prodotto. Per ovviare al problema, vengono proposti diversi tutorial per auto costruire una penna simile, a partire da contenitori di pennarelli o evidenziatori usati. Il vero problema per auto costruire una penna di questo tipo è il collocamento della batteria a stilo da 1,5V, in genere difficile da alloggiare nel corpo di un pennarello o evidenziatore.

Curiosando tra gli oggetti in vendita presso Futura Elettronica abbiamo trovato una penna biro "luminosa", alimentata con una stilo da 1,5V e in grado di accendere due LED bianchi per illuminare il campo di scrittura con una luminosità notevole. Immediatamente l'abbiamo presa e smontata, anche perché eravamo

incuriositi da come era possibile accendere, con quella luminosità, due LED bianchi a partire da un'alimentazione con un solo stilo AAA da 1,5V, come visibile nelle Fig. 6 e Fig. 7.

Una volta disassemblata, abbiamo deciso che quello era l'oggetto giusto per realizzare la penna a LED infrarosso, non

Dopo aver rigirato tra le mani, senza venirne a capo, il "misterioso" circuitino, abbiamo deciso di farci dare un aiutino dal nostro fido oscilloscopio.

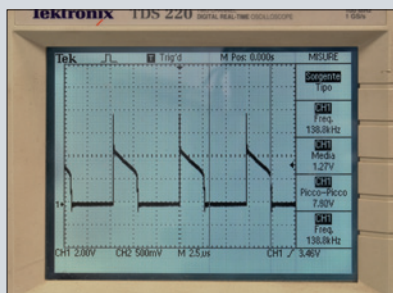


Fig. C

Fissati i puntali ai capi dei LED e data tensione ci è apparsa la forma d'onda visibile in Fig. C.

Un segnale a circa 138 KHz di circa 3,5 V di picco.

Che dire? Con tutta probabilità il minuscolo componente è un integrato convertitore DC DC switching, (che siano le prime avvisaglie delle nanotecnologie?) e quindi quella che a prima vista si camuffava da resistenza non può che essere un'induttanza.

Nessuna indicazione dalle cifre microscopiche stampate sull'integrato, mentre l'induttanza, dai colori, dovrebbe essere da 22 μ H.

Una ulteriore ricerca in Internet ci fa trovare un componente che, con tutta probabilità, assomiglia molto a quello montato sulla penna.

Si tratta del Boost Driver per LED Bianchi PR4401 della Prema Semiconductor. Il contenuto di questo integrato consiste di un convertitore step-up con uscita in corrente, che è in grado di fornire una tensione di uscita di 3,6 V tipica dell'alimentazione dei LED bianchi. La tensione di ingresso può arrivare anche ad un minimo di 0.9 V, consentendo l'utilizzo con una singola batteria a stilo (ed anche non proprio carica). Nella Fig. D è riportato il pinout dell'integrato mentre nella

Fig. E è riportato il circuito di utilizzo con lo schema a blocchi dell'integrato stesso.

La corrente di uscita è determinata

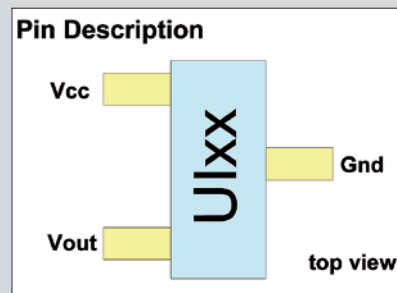


Fig. D

dal tipo e numero di LED che si vogliono alimentare ed è determinabile utilizzando un'induttanza di valore opportuno.

I valori di induttanza rispetto alla corrente in uscita, che si vuole ottenere, sono riportati nella tabella di Fig. F. Il datasheet raccomanda l'utilizzo

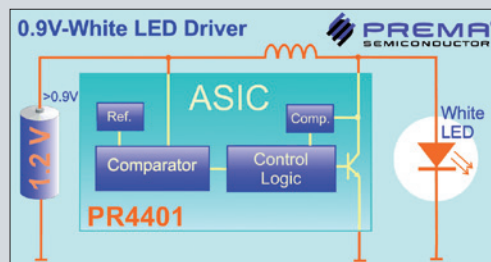


Fig. E

di un'induttanza da 22 μ H (guarda caso) per alimentare due LED bianchi. Questo valore di induttanza, sempre dal datasheet, è anche il valore raccomandato per garantire la maggior durata della batteria, circa 55 ore, con una stilo tipo AA.

Fig. F

Inductance L1	Mean current	for
47 μ H	6.5 mA	PR4401
32 μ H	8.3 mA	PR4401
26.7 μ H	10.8 mA	PR4401
22 μ H	11 mA	PR4401
14.7 μ H	14 mA	PR4401
10 μ H	22 mA	PR4401/4402
6.8 μ H *	32 mA *	PR4402
4.7 μ H *	40 mA *	PR4402

Whiteboard con RaspberryPi

La tentazione era irresistibile; provare a caricare la lavagna interattiva su RaspberryPi. Ultimamente proviamo a caricare su RaspberryPi tutto quello che ci capita a tiro. La sorpresa è che, nella maggior parte dei casi, funziona.

Quindi abbiamo provato.

Innanzitutto è necessario caricare il pacchetto di gestione Bluetooth. Se seguite i nostri articoli su RaspberryPi avete già capito.

```
apt-get clean
apt-get update
apt-get install bluetooth
```

Partirà una lunghissima sequenza di installazioni e messaggi che, molto probabilmente terminerà con un lungo elenco di pacchetti mancanti.

Le così dette “dipendenze”. Poco male, tutto si risolve con il comando

```
apt-get -f install
```

che, un po' come Mary Poppins, esamina la configurazione dei

pacchetti installati, verifica le dipendenze e, in caso di pendenze come nel nostro caso, scarica e installa i pacchetti mancanti, rimettendo tutto in ordine.

Ora tocca al software della lavagna, il pacchetto python-whiteboard che è distribuito come pacchetto .deb.

Lo scarichiamo con git (ricordate?).

Creiamo una cartella Python-whiteboard ci posizioniamo all'interno e poi scarichiamo il pacchetto con

```
git clone http://github.com/pnegr/python-whiteboard.git
```

Lo installiamo con il comando (Fig. G):

```
dpkg -i python-whiteboard_1.0.3_all.deb
```

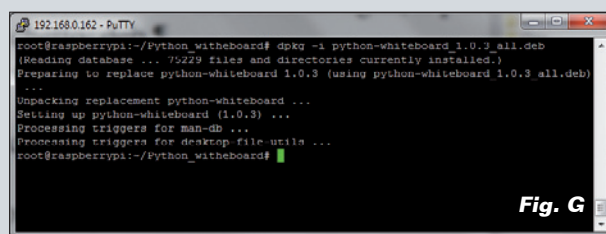
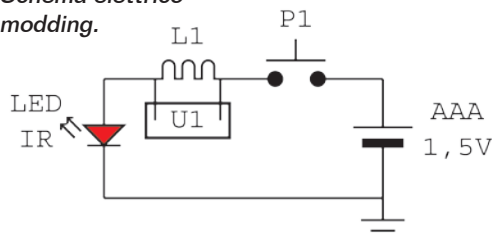


Fig. G

prima però di aver soddisfatto la curiosità circa i LED bianchi. In effetti, per accendere i LED bianchi è necessaria una tensione di almeno 3,5 V e quindi valeva la pena di approfondire il “misterioso” circuitino stampato con i due componenti che alimentano i due LED (vedi riquadro “Da dove viene tutta questa luce?”). L’obiettivo del “modding” della penna sta nel sostituire i due LED bianchi con un LED che emetta nel campo dell’infrarosso e nell’inserire nel circuito un pulsante che permetta di accendere il LED stesso a comando. Quindi ci serve un LED all’infrarosso da 5mm di diametro, che emetta un fascio di luce sulla frequenza dei 940 nm, come il modello 1418-LED5IR reperibile presso Futura Elettronica od i modelli Vishay TSAL6400,

Fig. 8
Schema elettrico
modding.



TSAL5300 o TSAL5100, consiglia- ti nel sito WiildOs.

Un pulsante da circuito stampato come il modello 8220-KRS0610 sempre reperibile presso Futura Elettronica, qualche spezzone di filo di collegamento ed un po' di pazienza completano l'elenco dei materiali necessari.

Quello che dobbiamo realizzare è il semplicissimo circuito visibile nella Fig. 8. Come vedete, abbiamo deciso di mantenere il circuito “misterioso” anche per accendere il LED all’infrarosso, anche se per questo è sufficiente una tensione di 1,35 V, abbiamo fatto un po' di prove ed abbiamo notato che la luminosità del LED è molto più elevata se alimentato dal “circuitino” senza avere riscontrato inconvenienti di alcun genere. Iniziamo con la descrizione delle operazioni. Come prima operazione, con una piccola lima tonda (coda di topo) e molta attenzione, allarghiamo il foro di uscita del terminale trasparente della penna fino a quando è in grado di alloggiare il LED infrarosso (Fig. 9).

Poi, con un trapano ed una punta da 3,5 mm foriamo il corpo della



Fig. 9
Punta della penna.



Fig. 10
Foro pulsante.

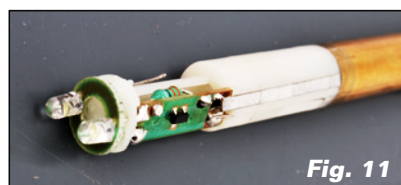


Fig. 11

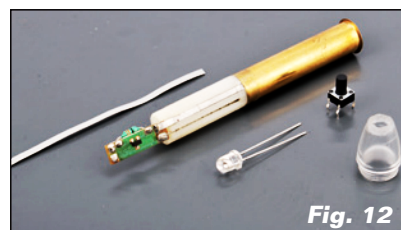


Fig. 12

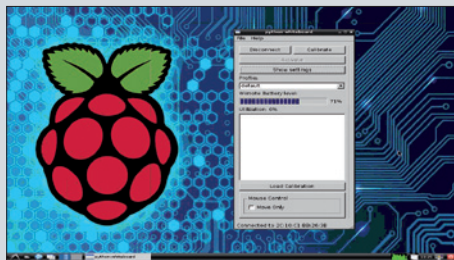


Fig. H

infine è il turno della lavagna Ardesia che si installa col comando

```
apt-get install ardesia
```

Ora inseriremo il dongle Bluetooth in uno slot libero dell'hub USB di RaspberryPi; possiamo verificare il riconoscimento del dispositivo con i soliti comandi dmesg e lsusb.

Apriamo il desktop di RaspberryPi e lanciamo python witheboard dal menu Education.

Il resto lo conoscete. (**Fig. H**)

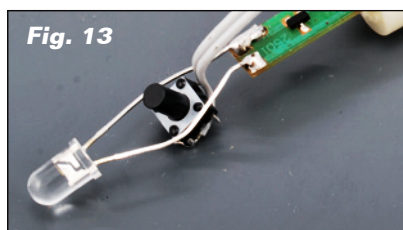


Fig. 13



Fig. 14



Fig. 15

penna, nel punto visibile in **Fig. 10**, per realizzare l'alloggiamento per il pulsante. Con un taglierino ripuliamo bene il foro dai residui dell'impugnatura in plastica nera per fare in modo che il perno del pulsantino possa scorrervi liberamente, senza incepparsi.

Ora è il momento della pazienza. Con un tronchesino tagliamo molto attentamente il supporto di plastica dei due LED bianchi dove si congiunge al corpo cilindrico dove è incastrato il pcb. (**Fig. 11**).

Poi, con il saldatore, sciogliamo i due punti di saldatura che fissano i terminali di uno dei due LED al piccolo circuito stampato e, sollevandoli leggermente prima da una parte e poi dall'altra, eliminiamo il supporto in plastica con i due LED bianchi come mostrato in **Fig. 12**, dove sono visibili anche gli altri componenti del modding.

Come prima operazione saldiamo il LED infrarosso sull'estremità esterna del piccolo circuito stampato; ricordiamo che il terminale positivo del LED è quello più lungo, che va saldato sulla pista del piccolo circuito stampato interrotta dai terminali del microscopico integrato. L'altra pista, quella continua, è la massa, dove collegheremo il terminale negativo del LED infrarosso.

Non ci resta che saldare il piccolo pulsante. Accorciamo i piedini con un tronchesino e saldiamo ad un cavetto. Facciamo in modo che il pulsante possa stare tra i terminali del LED, per poterlo inserire nel corpo della penna. Ora stacciamo il terminale positivo della batteria (quello piatto) dalla pista del circuito stampato e saldiamo i due capi liberi del cavetto, uno al terminale della batteria e l'altro sulla pista del circuito stampato.

Il risultato è visibile in **Fig. 13**.

Isoliamo i due punti appena saldati con un pezzetto di nastro adesivo.

Isoliamo il piccolo circuito stampato con un piccolo spessore di carta fissato con un pezzetto di nastro isolante come in **Fig. 15**. Ora inizia la parte ... acrobatica del modding. Dobbiamo posizionare il pulsante tra i terminali del LED e inserire il supporto nel corpo della penna in modo che il pulsante si trovi allineato con il foro che abbiamo eseguito sul corpo della penna per alloggiarlo.

Inseriamolo con precauzione fino a quando la testa del pulsante "appare" dal foro di alloggiamento. Con una pinzetta cerchiamo di "guidare" il pulsante in modo da farne uscire il perno dal foro (**Fig. 16**).

Teniamolo con la pinzetta dalla parte opposta e continuiamo a spingere il supporto all'interno della penna, senza arrivare a fondo. In questo modo il pcb isolato con la carta si infilerà sotto al pulsante e lo sosterrà anche quando viene premuto.

Avvitiamo il puntale trasparente della penna sul suo alloggiamento, facendo attenzione a non far attorcigliare i terminali del LED, e poi completiamo l'inserimento del contenitore della batteria nel corpo della penna. Il LED andrà a posizionarsi nel foro che abbiamo praticato per alloggiarlo e poi i terminali si piegheranno leggermente facendo da "molla" e tenendolo in sede (**Fig. 17**, **Fig. 18** e **Fig. 19**).

Chiudiamo il tappo della batteria e verifichiamo il funzionamento



Fig. 16

Fig. 17



Fig. 18



Fig. 19



con una telecamera, vale anche la webcam del PC o del telefonino. Inseriamo la batteria, posizioniamo il coperchietto posteriore per realizzare il collegamento della batteria, avviciniamo il LED alla telecamera e premiamo il pulsante. Se il risultato è quello visibile in Fig. 14 significa che la nostra penna funziona correttamente, oltre ad avere scoperto che le nostre telecamere sono, in genere, sensibili all'emissione infrarossa. Possiamo anche rimettere il cappuccio originale sul puntale. Attenzione a che la clip non tenga premuto il pulsante, altrimenti resteremo senza "gessetto" proprio nel momento del bisogno. Per un uso "scolastico" professionale portiamoci sempre un "gessetto", pardon una pila, di ricambio. Adesso occupiamoci del supporto per il telecomando della Wii.

IL SUPPORTO PER IL WIIMOTE

Ora serve un supporto stabile per il telecomando WiiMote. Sempre a catalogo Futura Elettronica è disponibile un mini cavalletto per macchine fotografiche che ci è parsa la soluzione migliore per costruire il supporto. Si tratta solo di trovare un raccordo adatto a sostenere il telecomando e ad essere avvitato sul cavalletto. Il pensiero è andato immediatamente alla stampante 3Drag. Una ricerca su Thingiverse ci ha fatto trovare un supporto utilizzabile. Dopo una prima stampa abbiamo realizzato che si trattava di un supporto per la vecchia versione del WiiMote. Un riadattamento delle dimensioni per adattarlo al WiiRemote Plus in nostro possesso ed una ristampa hanno risolto il problema. Per la descrizione della 3Drag ed il suo utilizzo rimandiamo

alla raccolta di articoli apparsi sulla rivista e che appariranno in futuro. Vogliamo solo ricordare le immense potenzialità didattiche di uno strumento di questo genere.

Per quanto riguarda il processo di stampa possiamo riassumerlo nei pochi soliti passi. Si carica il file .stl nel pacchetto Repetier, si impostano le caratteristiche di stampa, si esegue lo "slicing" con la produzione del codice G e ...

Fig. 20

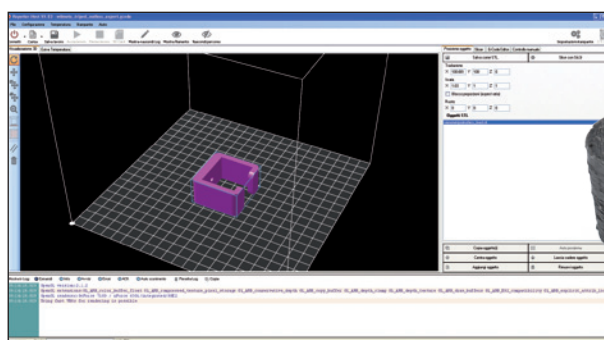
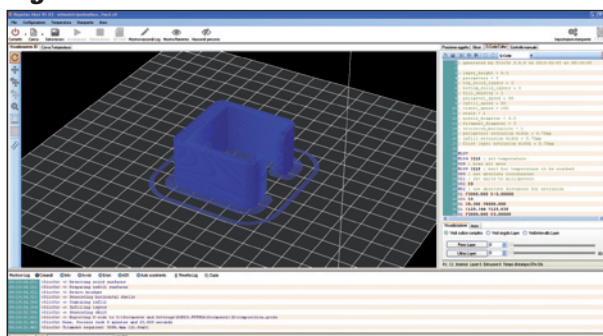


Fig. 21

si stampa. Noi abbiamo usato il PLA nero. Processo e risultato sono visibili nelle Fig. 20, Fig. 21, Fig. 22 e Fig. 23.

Restava il fissaggio sul cavalletto, che possiede una filettatura standard per le macchine fotografiche ma che non trova riscontro nelle misure della viteria meccanica "classica". La soluzione è stata di "filettare" il foro del supporto utilizzando la vite del cavalletto

stesso come "maschio", previo adattamento del foro nel supporto con una punta da trapano da 5 mm come visibile nelle Fig. 24 e Fig. 25.

Terminata la parte dedicata alla preparazione dell'hardware, possiamo pensare al software.

PREPARIAMO UNA CHIAVETTA CON IL SOFTWARE WIILDOS

Per utilizzare la distribuzione WiildOs abbiamo a disposizione tre possibilità:

- Installare la distribuzione su

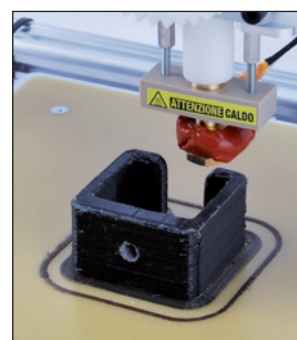


Fig. 22

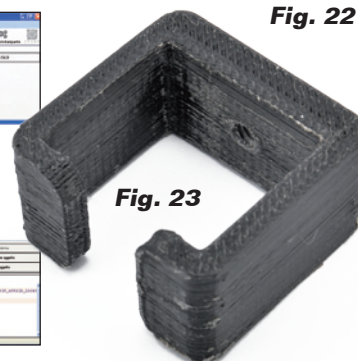


Fig. 23



Fig. 24



Fig. 25



Fig. 26

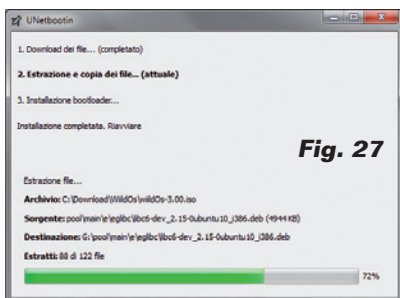


Fig. 27

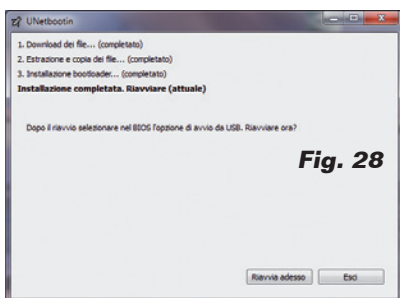


Fig. 28

- un computer dedicato;
- Utilizzare la versione live su CDROM avviabile. In questo caso possiamo provare tutte le funzionalità senza installare nulla sul PC ma con l'inconveniente di non poter salvare modifiche o documenti in modo permanente;
- La terza possibilità è di creare una distribuzione "persistente" su una chiavetta USB avviabile, in modo da poter sperimentare la distribuzione senza installare nulla sul PC, ma conservando documenti, configurazioni e modifiche.

Noi abbiamo adottato la terza soluzione, creando una chiavetta USB persistente da 8Gb. Ci perdonino gli autori della distribuzione ed i puristi GNU/Linux; per questa operazione noi abbiamo utilizzato l'am-

biente Windows, con la speranza che questo faciliti l'avvicinamento a questo mondo di un maggior numero di persone. In ogni caso nel sito del progetto <http://wiildos.wikispaces.com> sono presenti istruzioni dettagliate per preparare CD e chiavette USB per installare la distribuzione su un PC dedicato oppure per creare una distribuzione live su CD o chiavetta USB o ancora persistente su chiavetta USB. Come prima operazione bisogna scaricare l'immagine .iso della distribuzione, giunta alla versione 3.0, dall'indirizzo <http://wiildos.wikispaces.com/Scarica>. La versione 3.0, oltre ad essere stata arricchita di molte nuove funzionalità, risolve totalmente i problemi di riconoscimento automatico Bluetooth delle diverse versioni di telecomandi Wii e Wii Motion Plus. Per "creare" la chiavetta USB abbiamo utilizzato il tool Unetbootin scaricabile dal sito <http://unetbootin.sourceforge.net/>. Unetbootin è un programma multiplatforma, in grado di creare una chiavetta USB "bootabile" di una distribuzione GNU/Linux, a partire da una immagine .iso della distribuzione stessa. Le distribuzioni .iso sono quelle generalmente disponibili per masterizzare CD e DVD di

installazione o da utilizzare in modalità "live".

Inseriamo la chiavetta in una presa USB del PC, lanciamo il programma Unetbootin e impostiamo i parametri di creazione nel modo seguente:

- Lasciamo invariata l'impostazione "Distribuzione" e selezioniamo invece l'opzione "Immagine disco";
- Selezioniamo l'immagine .iso della nostra distribuzione WiildOs;
- Scegliamo la lettera che contraddistingue la chiavetta sulla quale vogliamo installare l'immagine .iso. Controlliamo più volte ciò che facciamo, perché il programma "ricoprirà" con l'immagine .iso qualsiasi drive avremo indicato, anche il disco fisso principale, e non sono ammessi ripensamenti. Quindi, ancora, attenzione (Fig. 26);
- Se stiamo usando una chiavetta da 8 Gb, come vi raccomandiamo, impostiamo un'area "persistente" di 4.48 Mbyte, Diamo OK.

Seguiamo il trascorrere dell'installazione con molta pazienza, alla fine ci verrà chiesto di riavviare il computer. In realtà possiamo continuare tranquillamente il nostro lavoro rimandando il

Fig. 29

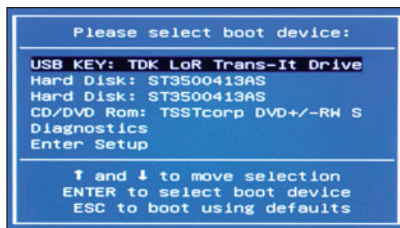


Fig. 30

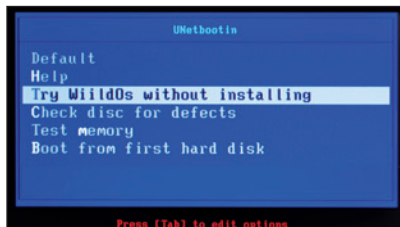


Fig. 31

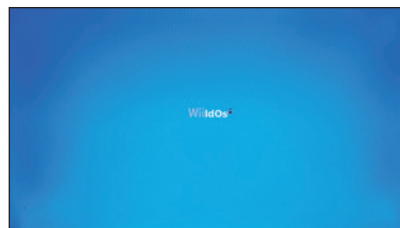
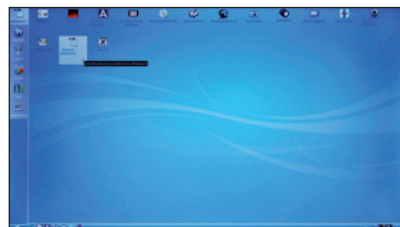


Fig. 32



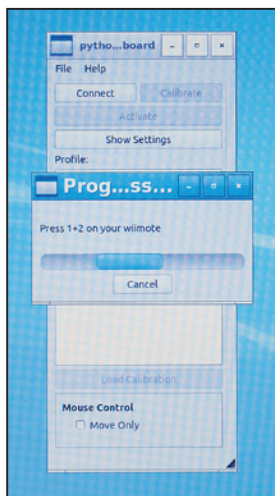


Fig. 33

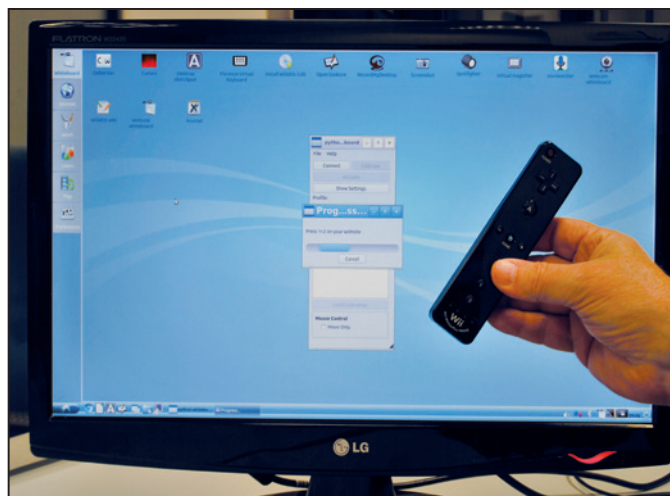


Fig. 34

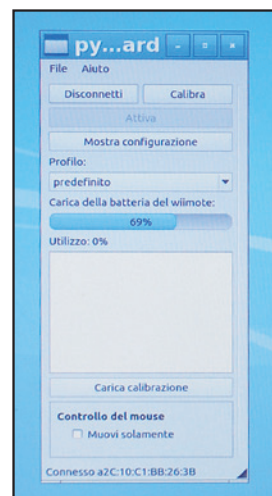


Fig. 35

test della chiavetta al momento che riteniamo più opportuno (Fig. 27 e Fig. 28).

L'UTILIZZO

Bene, ora passiamo a provare se il tutto funziona a dovere. Non avendo a disposizione un proiettore abbiamo allestito un "banco" di prova come quello visibile in Fig. 27. Inseriamo la chiavetta USB con la distribuzione WiildOs e accendiamo il PC. Interrompiamo il processo di boot con la pressione del tasto F2 o F12 o quello richiesto dal bios del nostro PC. Andiamo alle opzioni di boot e scegliamo di eseguire il boot dalla chiavetta USB (Fig. 29). Una volta apparsa la prima videata del bootloader scegliamo l'opzione "live persistent" e diamo invio di nuovo (Fig. 30). Segue tutto il processo di boot (Fig. 31) fino a quando ci appare il menu a schede iniziale della distribuzione. L'impostazione del menu è molto intuitiva, con le scelte delle categorie di applicazioni nella barra di sinistra. Cliccando su ciascuna categoria si apre la scheda corrispondente con tutte le applicazioni collegate alla categoria e così per i menu di livello inferiore. Inseriamo il dongle Bluetooth USB che verrà riconosciuto automaticamente. Chi segue i nostri articoli su Ra-

spberryPi, o comunque conosce GNU/Linux, può verificare il "caricamento" del driver aprendo una finestra di terminale con Ctrl-Alt-T e utilizzando i comandi dmesg e lsusb.

Nella categoria "Lavagna" troviamo l'applicazione "Wiimote LIM" (Fig. 32). Clicchiamoci sopra e ci apparirà la schermata di connessione automatica del telecomando Wiimote Bluetooth. Per attivarlo accendiamo il telecomando e premiamo contemporaneamente i tasti 1 e 2 oppure il piccolo tastino giallo al di sotto. Potrebbe essere necessario eseguire

l'operazione più volte, non scoraggiatevi. Durante la fase di auto connessione lampeggiano i quattro LED azzurri sul WiiMote; quando la comunicazione è stabilita il telecomando ci avvisa con una breve vibrazione e uno solo dei LED rimane acceso (Fig. 33, Fig. 34 e Fig. 35).

Posizioniamo il telecomando sul suo supporto (Fig. 36) in posizio-



Fig. 37

Fig. 36



Fig. 38

Fig. 39

ne frontale davanti allo schermo ad una distanza di circa una volta e mezza la larghezza dello schermo, tenendo presente che è bene non superare una distanza di 5 metri tra penna e telecomando, anche utilizzando un proiettore (Fig. 37).

Sullo schermo apparirà la videata di calibrazione: uno schermo bianco con quattro marcatori tondi di calibrazione. Seguendo le istruzioni sullo schermo tocchiamo con la punta

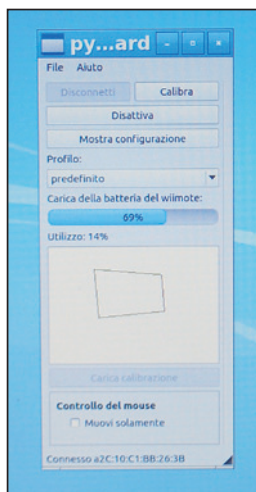


Fig. 40

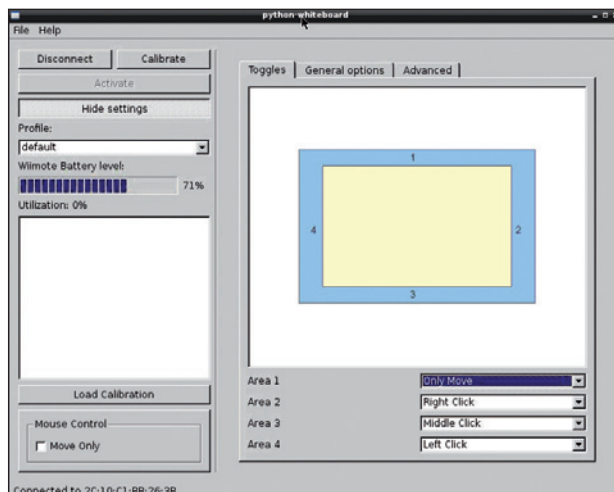


Fig. 41

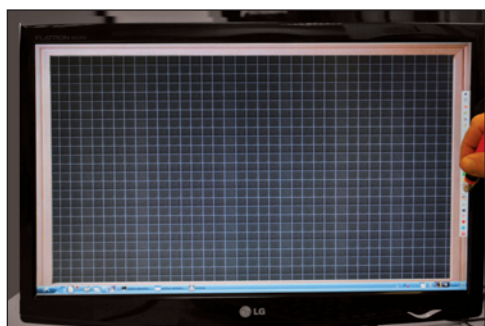


Fig. 42

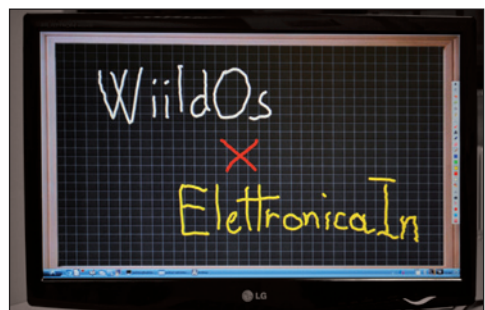


Fig. 43

della penna il primo marcatore in alto a sinistra e premiamo il pulsante di accensione del LED infrarosso. Il marcatore si riempirà di colore rosso e le istruzioni ci chiederanno di passare al marcatore successivo. Completiamo la procedura di calibrazione e ci troveremo nelle condizioni di utilizzare la nostra penna come il mouse del PC (Fig. 38 e Fig. 39). Completata la procedura di calibrazione, sull'interfaccia della applicazione appare un "dise-

gno", di come il telecomando Wii "vede" lo schermo e ci indica il livello di carica della batteria del telecomando stesso (Fig. 40). Una delle possibilità offerte dal programma è di utilizzare le aree attorno allo schermo per attivare le "funzioni speciali" della penna, come simulare il tasto destro del mouse o quello centrale.

Per fare questo basta premere il bottone "Show settings", scegliere il pannello "Toggles" e configurare le opzioni desiderate (Fig. 41).

Per fare qualche prova siamo andati nella sezione "Lavorare", sottosezione "Accessori" e abbiamo attivato la "lavagna virtuale" Ardesia.

Come sfondo abbiamo scelto, ovviamente, la nostra preferita, quella nera a quadretti con la cornice di legno (Fig. 42 e Fig. 43).

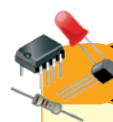
E qui il controllo è sfuggito di mano, tutti i collaboratori della redazione hanno iniziato una gara di produzione di disegni, salvati e stampati, da fare vedere a "casa", saltando anche la "pausa caffè". Ovviamente siamo stati presto scoperti, redarguiti a dovere e ... rispediti a lavorare. Quindi, per tutte le possibilità di utilizzo della LIM, della penna e di tutte le applicazioni presenti nella distribuzione e dedicate ai

Guarda il video della nostra lavagna luminosa su YouTube!



Vuoi vedere come funziona la nostra lavagna interattiva? Nulla di più semplice: guarda il breve video che abbiamo girato e postato su YouTube all'indirizzo:
<http://www.youtube.com/watch?v=nX2wN0xa1H8>

diversi ordini, gradi e indirizzi di istruzione non ci resta che rimandarvi alla consultazione dell'ottimo sito del progetto WiildOs, agli autori e sostenitori del quale va ancora il nostro ringraziamento ed incitamento a continuare, con l'augurio che sempre più istituti possano aderire all'iniziativa ed adottare la soluzione proposta, nella loro attività didattica.



per il MATERIALE

La penna luminosa al centro di questo progetto è disponibile presso Futura Elettronica al costo di 2,80 Euro (cod. ZLLPEN2). Questo dispositivo va modificato aggiungendo un LED IR (cod. 1418-LED5IR, Euro 1,00) ed un pulsante miniatura (cod. KRS0610, Euro 0,16). 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>



Mai più bollette con cifre stratosferiche: collegate questo dispositivo prima del telefono e bloccherete le numerazioni speciali.

KEY TEL LA CHIAVE TELEFONICA

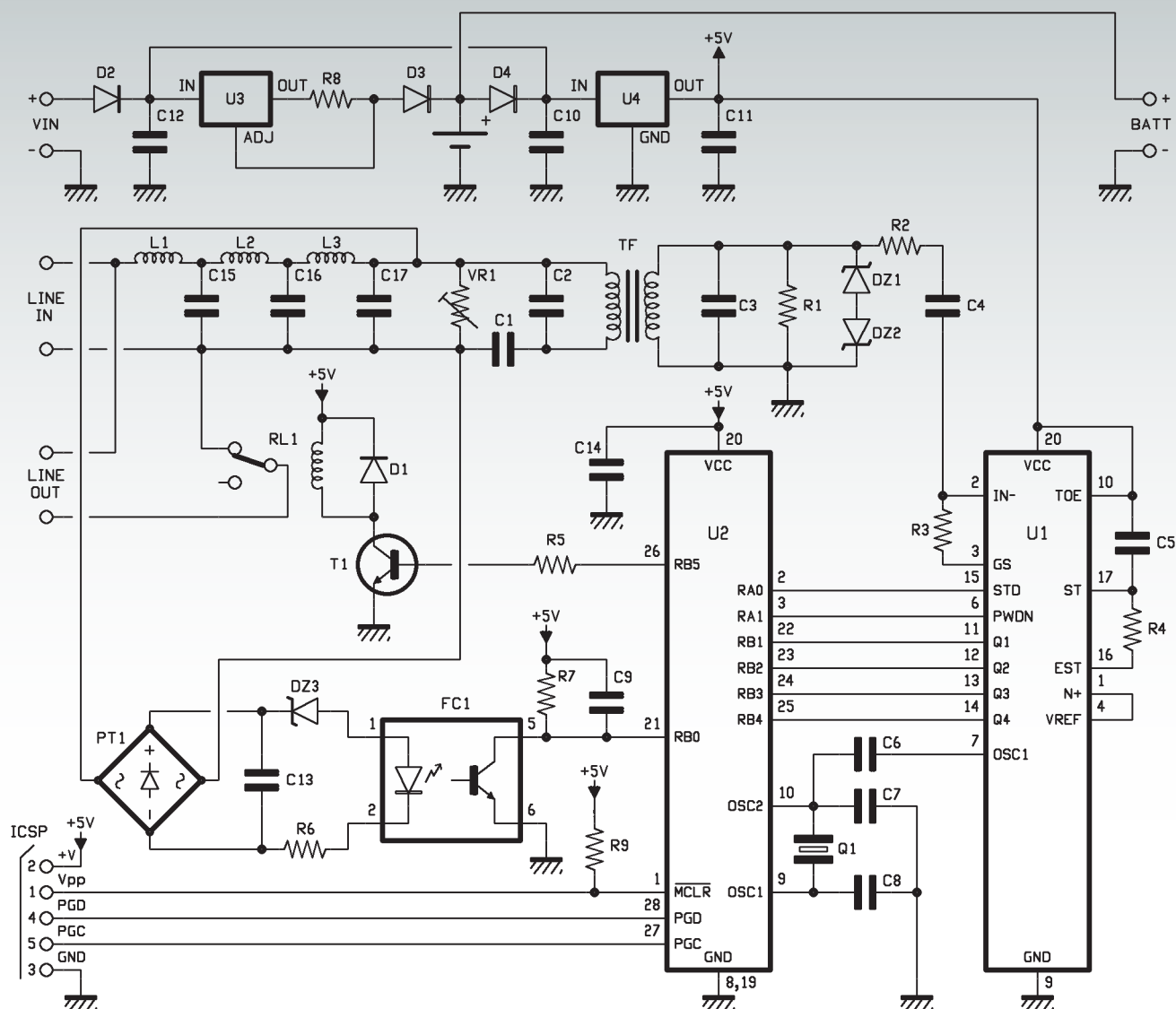
..... di FULVIO DE SANTIS

Quante volte ci è capitato di ricevere una bolletta telefonica inaspettatamente salata? E ciò, malgrado avessimo sottoscritto con il nostro gestore telefonico abbonamenti flat internet e a telefonate illimitate? Trovarsi addebitate chiamate pagate "oro" può sovente dipendere

che iniziano con numerazioni tipo 199, 899, 709, 166, 167, 892. Talvolta siamo noi stessi o i nostri figli a cadere nelle trappole mediatiche tese da certi spot pubblicitari e televendite in TV o nei banner e pop-up durante la navigazione sul web, che propongono loghi e suonerie per telefonini o pseudo concorsi che ci invitano a chiamare numeri del genere. Ebbene, si tratta sempre di spot che attirano i più o meno ignari utenti (i più esposti sono i ragazzi e i bambini, lasciati soli di fronte alla TV o al computer) facendoli "dissanguare" con telefonate costosissime e i cui costi non sono contestabili all'operatore, il quale al massimo offre la possibilità di bloccare determinate numerazioni, anche se tale operazione è un po' come "chiudere il recinto dopo che i buoi sono fuggiti". Una soluzione più praticabile è sicuramente il circuito proposto in queste pagine, che è

dal fatto che qualcuno in casa o in ufficio fa -più o meno maliziosamente- telefonate a numeri internazionali o intercontinentali, o a servizi di società nostrane ma con sede rigorosamente all'estero,





una sorta di filtro per le telefonate uscenti su numerazioni sia fisse, sia cellulari; KEYTEL, così abbiamo chiamato questo dispositivo, può essere programmato e gestito via telefono, sia localmente che a distanza da un altro apparecchio telefonico da qualsiasi località. Il circuito va collegato in serie al telefono da controllare, ovvero direttamente sulla scatola di arrivo del cavo telefonico, anche a monte dell'eventuale filtro ADSL e prima di tutti gli apparecchi telefonici; dopo l'accensione (notate che se viene a mancare la tensione, il circuito è alimentato dalla pila da 9 V) il sistema chiede all'utente in-

testatario della linea di inserire il codice di blocco, di due cifre: inserito questo, per effettuare chiamate senza limitazioni viene richiesto di comporre un codice a quattro cifre che sblocchi le limitazioni. Per reintrodurre tali limitazioni occorre reinserire il codice di due cifre. I codici di inserimento e disinserimento della chiave telefonica possono essere modificati solo da chi gestisce il circuito, ovvero dall'intestatario della linea telefonica. Per aumentare la versatilità del circuito è possibile definire o modificare, tramite un'apposita procedura, fino ad un massimo di 21 prefissi di numeri telefonici

fissi e 5 numeri di cellulari che fanno eccezione.

Notate che per garantire all'utente la necessaria tranquillità, la protezione telefonica, anche quando è inserita, non disabilita mai le chiamate di emergenza, ossia quelle a numeri che iniziano con 11 (112 dei Carabinieri, 113 della Polizia, 115 dei Vigili del Fuoco, 118 del soccorso medico e via di seguito). Ciò è indispensabile sia perché quando capita una situazione di emergenza non si ha la concentrazione necessaria a ricordarsi i codici di sblocco, sia perché in caso di necessità il telefono deve poter essere usato da tutti.

COME FUNZIONA

Il funzionamento del KEYTEL è basato sul PIC 16F876A, un microcontrollore della Microchip, che gestisce il riconoscimento dei numeri telefonici delle chiamate in uscita abilitate mediante algoritmi di confronto con il database dei numeri prescelti e inseriti dall'utente all'interno della sua stessa EEPROM.

Per comprendere come opera la nostra chiave, dovete sapere che ogni cifra digitata dall'utente sulla tastiera del telefono genera in linea un segnale DTMF (Dual Tone Multi Frequency) che il nostro circuito legge attraverso l'interfaccia di linea e decifra grazie al circuito integrato MT8870; questo componente è un decoder dei bitoni dello standard telefonico DTMF, il quale amplifica il segnale d'ingresso e tramite filtri sincronizzati dal quarzo esterno a 3,58 MHz con cui lavora, identifica le 16 coppie di toni DTMF, indicando di volta in volta in formato binario a 4 bit sulle proprie uscite Q1, Q2, Q3, Q4 di quale bitono si tratta.

Lo stato di queste uscite viene aggiornato all'arrivo di ogni bitono e mantenuto fino al riconoscimento del successivo grazie ad un latch quadruplo; per fare in modo che il dispositivo destinato a leggere i dati distingua più bitoni uguali consecutivi, l'MT8870 dispone di un'uscita chiamata StD, che pulsa ogni qualvolta viene riconosciuto un bitono DTMF.

I quattro bit che esprimono i bitoni e l'StD vengono letti dal PIC attraverso le linee RB1, RB2, RB3, RB4 della PORTB e mediante gli algoritmi del firmware viene effettuato il controllo delle cifre nel database dei numeri abilitati per il riconoscimento della cifra digitata e così di seguito per tutte le cifre componenti il numero telefonico. Se il numero selezionato non è compreso fra i numeri abilita-

ti, il microcontrollore forzerà il riaggancio (apertura della linea) attraverso l'apertura del contatto del relé RL1.

SCHEMA ELETTRICO

Il circuito è composto dai seguenti blocchi funzionali: interfaccia di utente della linea telefonica, sezione di ricezione e decodifica DTMF, microcontrollore PIC 16F876A, relé di disconnessione del telefono dalla linea telefonica, rivelatore di sgancio (impegno linea) e stadio di alimentazione. Vediamoli nel dettaglio, partendo dall'interfaccia di utente della linea telefonica, che si collega alla linea stessa attraverso un filtro ADSL LC passivo, passa basso del 6° ordine, in configurazione a T, costituito dalle induttanze L1, L2, L3 e dai condensatori C15, C16, C17; il filtro è stato previsto per evitare che l'eventuale portante ADSL vada a disturbare il funzionamento del decoder DTMF.

L'interfaccia è inoltre costituita dal varistore VR1, che protegge il circuito da eventuali picchi di tensione provenienti dalla linea, dal trasformatore di linea 1:1 (TF) dai condensatori C1 e C2. In particolare, il trasformatore provvede all'isolamento galvanico della logica dalla linea telefonica ed al trasporto del segnale di linea, bitoni DTMF inclusi; il condensatore C1 serve ad evitare che la resistenza in continua del primario del trasformatore carichi la linea impegnandola costantemente, perché altrimenti non sarebbe possibile fare o ricevere telefonate.

I condensatori C2 e C3 servono a filtrare il segnale dai disturbi impulsivi eventualmente presenti sulla linea, mentre la resistenza R1 provvede ad adattare l'impedenza dell'interfaccia a quella della linea telefonica, che è 600 ohm; quanto ai diodi Zener DZ1 e DZ2, operano un'adeguata protezione

dell'MT8870 quando in linea arriva l'alternata di chiamata (che tipicamente è a 70 Veff.) o si presentano picchi di tensione sui quali il varistore non sia intervenuto.

Dal secondario del trasformatore di linea, il segnale contenente i bitoni raggiunge l'MT8870 tramite il bipolo serie R2-C4.

Perché tutto funzioni, l'apparecchio telefonico deve essere collegato alla morsettiera LINE OUT, mentre la linea telefonica va alla morsettiera LINE IN; in queste condizioni, il contatto del relé RL1, normalmente chiuso, connette il telefono alla linea, mentre provvederà alla disconnessione in caso di tentativo di chiamata ad un numero non abilitato.

La sezione di ricezione e decodifica del doppio tono DTMF è costituita dall'integrato U1 (MT8870). Il segnale DTMF proveniente dall'interfaccia di linea entra nell'ingresso IN- ed è amplificato da uno stadio invertente interno, il cui guadagno in tensione è determinato dal rapporto tra R3 ed R2. Dopo l'amplificazione, l'integrato MT8870 identifica il bitono e ne fornisce il corrispondente numero in formato binario sulle uscite Q1÷Q4 secondo la **Tabella 1**, estratta dal data-sheet dell'MT8870, dove per ogni tasto premuto sul telefono è riportata la corrispondente combinazione binaria alle uscite Q1, Q2, Q3, Q4 dell'integrato.

Passiamo adesso al microcontrollore PIC 16F876A, che funziona sincronizzato dallo stesso quarzo che scandisce le temporizzazioni dell'8870; notate che i condensatori C7 e C8 contribuiscono a far funzionare l'oscillatore del micro, il cui segnale, attraverso C6, fa da clock anche all'integrato U1 MT8870 (C6 disaccoppia in continua i due circuiti di clock).

Collegato alle linee di programmazione del PIC è previsto il

connettore ICSP, per le necessità di diagnostica o di aggiornamento del firmware; in questo progetto, per la programmazione del PIC e debug è possibile utilizzare il tool della Microchip ICD2.

Il PIC riceve sulle linee RB1, RB2, RB3 ed RB4 le word a 4 bit che esprimono i bitoni riconosciuti, mentre si aiuta con l'StD (letto dalla linea RA0) per distinguere bitoni uguali consecutivi. Il firmware che gira nel microcontrollore legge uno ad uno i valori binari a quattro bit e quando il numero è terminato lo confronta con quelli presenti nella memoria EEPROM interna del PIC per verificare se l'indicativo telefonico digitato risulta fra i numeri abilitati.

Se il numero digitato non è abilitato alla chiamata in uscita, il PIC, attraverso la linea RB5, porterà in saturazione il transistor T1, il quale attiverà per un breve intervallo di tempo il relé RL1, aprendo così il contatto e sconnettendo il telefono dalla linea telefonica per circa un secondo. Ciò causerà il disimpegno della linea e il ritorno del tono di libero, quindi ripor-

terà tutto come se l'utente avesse appena sganciato la cornetta per fare il numero.

Una parte molto importante del circuito è il rivelatore di impegno linea, composto dal ponte a diodi PT1, dallo Zener DZ3 e dall'optoisolatore FC1 (4N35) che insieme svolgono la funzione di rivelare lo sgancio del microtelefono dell'apparecchio telefonico.

A riposo, con microtelefono agganciato (linea non impegnata), sui morsetti 1-2 di LINE IN sarà presente una tensione di centrale di circa 48Vcc che, indipendentemente dalla polarità, attraverso il ponte a diodi PT1 manterrà in conduzione il diodo zener DZ3 e l'optoisolatore U3. Il collettore di U3 avrà così un livello di tensione basso (0 logico) presente quindi anche sulla linea RB0/INT del PIC programmata per attivare un interrupt del software al cambio di stato da livello basso ad alto sul pin RB0/INT. Nel momento in cui avviene lo sgancio del microtelefono (linea impegnata) la tensione sul doppio scenderà a circa 8 volt, quindi lo Zener non condurrà più e conseguentemente l'optoisolatore, determinando lo stato logico 1 sulla linea RB0/INT, generando così un interrupt e il consecutivo start del programma del PIC.

In altre parole, il rivelatore di sgancio serve ad avviare la sequenza di analisi del numero telefonico che viene composto; se non ci fosse, il microcontrollore non saprebbe da quando considerare i bitoni e iniziare a leggerli, né quando il numero è finito. Grazie al segnale fornito dal rivelatore, allo sgancio la logica si prepara ad acquisire i bitoni, mentre al riaggancio si predispone a leggere un nuovo numero.

Bene, concludiamo l'analisi dello schema elettrico con la sezione di alimentazione, che partendo da

una tensione di +12 Vcc applicata ai punti Vin ricava i 5 volt occorrenti alla logica; essa è costituita dall'integrato LM317L (protetto contro l'inversione di polarità dal diodo D2) che abbiamo configurato per fornire corrente costante per la carica della batteria da 9V, connessa mediante clip al connettore BATT. La batteria serve a consentire alla chiave di funzionare anche se c'è un blackout.

Completa l'alimentatore il regolatore di tensione 78L05, che ricava 5 volt stabilizzati con cui funzionano il microcontrollore, l'MT8870 ed il resto della logica, nonché il relé.

L'inserzione della batteria è regolata dal diodo D4: in presenza dei 12 V, il potenziale a valle dell'LM317L è certamente maggiore di quello sul catodo del diodo D4, quindi questo non conduce e ad alimentare U4 è la tensione d'ingresso; se questa viene a mancare, la tensione fornita dalla batteria fa condurre D4 e la corrente raggiunge il 78L05. Notate che il diodo D3 serve a proteggere il regolatore U3 quando manca la tensione all'ingresso del circuito, consentendo la ricarica dell'accumulatore quando invece il circuito è regolarmente alimentato dall'esterno.

IL FIRMWARE

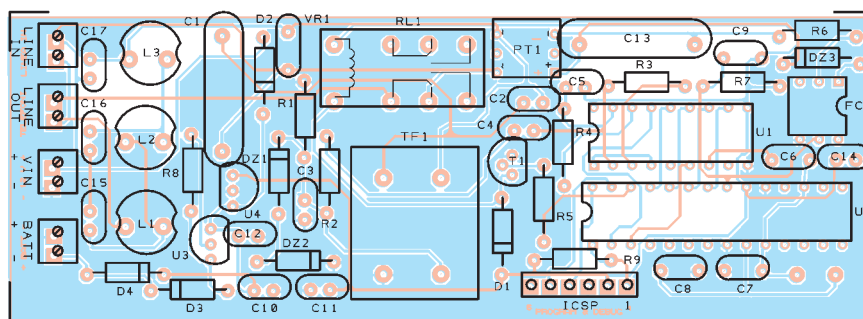
Dopo la descrizione dell'hardware passiamo all'analisi del firmware che gira nel microcontrollore e che è stato scritto completamente in Assembler e compilato in ambiente MPLAB Microchip. Nel firmware vengono impostate le direttive, le macro istruzioni e le variabili del programma; nella EEPROM dati del PIC sono inseriti i numeri telefonici di emergenza e di pubblica utilità sempre abilitati, i codici programma necessari per l'avvio delle routine delle varie funzioni e i codici delle chiavi di

Tabella 1

Digit	TOE	INH	ESL	Q ₄	Q ₃	Q ₂	Q ₁
ANY	L	X	H	Z	Z	Z	Z
1	H	X	H	0	0	0	1
2	H	X	H	0	0	1	0
3	H	X	H	0	0	1	1
4	H	X	H	0	1	0	0
5	H	X	H	0	1	0	1
6	H	X	H	0	1	1	0
7	H	X	H	0	1	1	1
8	H	X	H	1	0	0	0
9	H	X	H	1	0	0	1
0	H	X	H	1	0	1	0
*	H	X	H	1	0	1	1
#	H	X	H	1	1	0	0
A	H	L	H	1	1	0	1
B	H	L	H	1	1	1	0
C	H	L	H	1	1	1	1
D	H	L	H	0	0	0	0
A	H	H	L	undetected, the output code will remain the same as the previous detected code			
B	H	H	L				
C	H	H	L				
D	H	H	L				

Elenco Componenti:

R1: 680 ohm
 R2, R3: 100 kohm
 R4: 330 kohm
 R5, R9: 10 kohm
 R6: 15 kohm
 R7: 4,7 kohm
 R8: 150 ohm
 C1÷C2: 33 nF ceramico
 C4, C5: 100 nF ceramico
 C6÷C8: 22 pF ceramico
 C9: 4,7 nF ceramico
 C10, C11: 10 nF ceramico
 C12, C14: 100 nF ceramico
 C13: 1 µF 250 VL poliestere
 C15÷C17: 47 nF 100 VL poliestere
 L1, L2: Induttanza 22 mH
 L3: Induttanza 10 mH
 U1: MT8870
 U2: PIC16F876A-I/P (MF1044)
 U3: LM317L
 U4: 78L05
 FC1: 4N35
 T1: BC547
 D1÷D4: 1N4148
 DZ1: Zener 4,7V 400mW
 DZ2: Zener 4,7V 400mW
 DZ3: Zener 10V 400mW



PT1: Ponte diodi
 Q1: Quarzo 3,5795 MHz
 RL1: Relé 5V 1 scambio
 TF1: Trasformatore 1:1 600 ohm
 VR1: Varistore

Varie:
 - Batteria ricaricabile 9V
 - Morsettiera 2 poli passo 2,54 mm (4 pz.)
 - Strip maschio 6 poli

- Zoccolo 3+3
 - Zoccolo 9+9
 - Zoccolo 14+14
 - Circuito stampato

abilitazione e disabilitazione dei prefissi e dei cellulari.

Il circuito è dotato di un connettore per la programmazione del PIC, per le attività di aggiornamento e gli interventi di diagnostica del firmware.

Seguiamo il flusso del programma: all'accensione della scheda avviene il reset hardware del sistema e il programma si porta automaticamente all'indirizzo "0000" della memoria Flash del PIC, in cui si trova l'istruzione che rinvia alla routine "Start" di inizio del programma. Di seguito, all'indirizzo "0004" della Flash troviamo il gruppo di istruzioni per l'impostazione e abilitazione dell'interrupt del PIC che, come vedremo, si attiva mediante la transizione BASSO/ALTO della linea RB0/INT del microcontrollore. Con la routine "Start" vengono resettate le variabili, impostate le

PORTA e PORTB e il timer TMR0 del PIC, viene inserita la chiave generale e settata la linea RB5 a 0 logico, così da lasciare che il contatto del relé chiuda la linea telefonica sul telefono; infine viene acceso il ricevitore DTMF MT8870 e abilitato l'interrupt sulla linea RB0/INT. L'accensione dell'U1 si ottiene gestendo il piedino PWDN (6) di quest'ultimo; a riposo, PWDN è ad uno logico e mantiene in standby l'integrato allo scopo di limitare il consumo di corrente. Dopo la routine "Start" il programma prosegue con la routine "Main", la quale verifica se la linea è impegnata a seguito dello sgancio del microtelefono; ciò si ottiene mediante il monitoraggio dello stato della linea RB0. Se la linea non risulta impegnata, dalla Main il programma si porta alla routine "Standby". Qui avvengono il reset di alcuni flag, il power-down

del ricevitore MT8870 (ottenuto riportando la linea RA1 a livello logico alto) e il successivo ritorno in standby del PIC, mediante l'istruzione "sleep".

L'impegno della linea fa sì che dallo stato di sleep si riattivi il PIC e si riaccenda U1; la linea del programma si porta alla routine "Main" che, verificato il corretto livello alto della RB0 corrispondente all'impegno linea in corso, verifica lo stato della linea RA0 configurata come INPUT e collegata al pin StD del MT8870: se la linea StD è a livello basso il programma torna in standby poiché ciò indica nessun tono DTMF valido rilevato, mentre se StD è a livello alto (ad indicare la presenza di un tono DTMF valido) il programma viene indirizzato alla subroutine "LeggiSTD" di acquisizione del codice a 4 bit del tono DTMF e da qui alla routine

“IdentiTONO” per l’identificazione del codice funzione.

Infatti, il primo tono ricevuto viene analizzato per stabilire se quella composta è la prima cifra di un numero in composizione oppure la prima cifra di un codice di richiesta di attivazione di una determinata funzione.

Analizziamo quindi le varie prime cifre significative acquisite dal programma e la relativa funzione: “4” è la prima cifra del codice programma per l’inserimento in memoria di numeri telefonici, quindi viene indirizzata la subroutine “SETCODPROG” che setta i parametri per l’identificazione delle due cifre successive che completano il codice funzione, imposta i banchi di memoria per il salvataggio delle cifre dei codici programma, infine setta il flag FLAGCODPROG, così da indicare al firmware che si sta trattando di un codice programma.

Dalla “SETCODPROG” il programma rinvia alla “PREFCELLABILIT” che continua con il salvataggio delle cifre rinviando alle routine “CHECKPREFCELL” e “CHECKCIFRE” in cui, attraverso il controllo del flag FLAGCODPROG, stabilisce che è in corso un codice programma valido rinviando alla routine “SELECTPROGR”. La routine “SELECTPROGR” si occupa di individuare il codice programma scelto dall’utente abilitato. I codici completi digitati dalla tastiera telefonica “402”, “403”, “404”, attivano le subroutine “PROGPREF2”, “PROGPREF3”, “PROGPREF4” che riguardano rispettivamente i prefissi a due, tre, quattro cifre, mentre digitando il codice “439” si attiva la subroutine “PROGCELL”. Ognuna di queste subroutine setta il banco di memoria dedicato ai rispettivi prefissi e ai cellulari, rinviando poi il programma alla routine “Main” in attesa della digitazione

della password di accesso e delle cifre dei prefissi o cellulari da inserire da parte dell’utente.

I numeri abilitati saranno inseriti nella memoria EEPROM del PIC e vi rimarranno quindi anche in assenza di alimentazione.

La prima cifra di codice funzione asterisco “*” attiva la routine “RichiediCHIAVE” che predispose il puntatore del banco di memoria per l’inserimento delle cifre del codice chiave, poi rinvia alla routine “InserCHIAVE”; questa routine continua l’inserimento delle due cifre della chiave generale, di seguito alle quali devono essere digitate le due cifre di codice riconosciute dal programma (le cifre “1, 2” sono quelle predefinite) e cancellato # a chiusura dell’inserimento codice. Una volta inserita la chiave saranno consentite solo chiamate verso numeri abilitati ad eccezione dei numeri di emergenza che risulteranno sempre abilitati. La sequenza delle cifre della chiave da digitare è:

* 1 2 #

Quindi, successivamente verifica che le cifre digitate siano corrispondenti al codice chiave inserito in EEPROM e se corrispondono, la chiave è inserita, il registro CHIAVE viene settato e il tutto viene confermato dal tono di linea libera conseguente al riaggancio della linea mediante il rinvio alla routine “LINEAOFF”; se invece il codice viene riconosciuto ma non il “#” finale, il programma rinvia alla routine “Vernewcod” per verificare se si tratta dell’inserimento di un nuovo codice chiave e rinvia alla “Main” in attesa di ricevere altre cifre; se riconosce che l’ultima cifra del codice è l’asterisco, stabilisce che si tratta di una richiesta dell’utente di modificare il codice chiave e pertanto il programma salta alla routine “INSERTCOD”.

La INSERTCOD si occupa di acquisire le cifre del nuovo codice predisponendo il puntatore del banco di memoria di appoggio delle cifre da controllare e rinvia alla routine “NEWCOD” con la quale viene richiesta la ripetizione delle due nuove cifre inserite dall’utente. La sequenza di digitazione per l’inserimento di un nuovo codice chiave è:

* 1 2 * X X # X X #

(le X rappresentano le nuove cifre del codice)

Una volta confermato il nuovo codice il programma salta alla routine “APGINCOD” la quale effettua l’inserimento del nuovo codice nella memoria EEPROM, infine la routine “LINEAOFF” conclude le operazioni con il riaggancio della linea e conseguente ascolto del tono di linea libera.

La prima cifra di codice funzione cancellato “#” attiva la routine “VerDisins” che predispose il puntatore delle locazioni di memoria delle cifre di codice delle chiavi, e da qui alla “DisinserCHIAVE”, routine di disinserimento chiave generale; dopo cancellato “#” devono seguire altre quattro cifre riconosciute dal programma (codice “1 2 3 4” di default) e asterisco a chiusura dell’inserimento codice. Il disinserimento comporta il reset del registro CHIAVE e di conseguenza sarà consentita la chiamata a qualsiasi numero telefonico sia fisso che cellulare. I codici di inserimento/disinserimento chiave, possono essere modificati da utente abilitato.

La sequenza di digitazione per l’inserimento del codice di disinserimento chiave è:

1 2 3 4 *

Analogamente alla routine “InserCHIAVE”, la “DisinserCHIAVE”,

15 anni senza ricarica



Novità presentata al CES 2013, si chiama SpareOne Plus ed è un cellulare che, a detta del costruttore, presenta un'autonomia in standby di ben 15 anni (10 ore di conversazione) alimentato da una pila stilo Energizer. Spare One non è certo il telefono di tutti i giorni (il suo display è numerico e non ci sono funzioni aggiuntive) ma è stato concepito per essere usato come cellulare di scorta... di emergenza (non a caso dispone di una luce a LED che con la pila in dotazione può restare accesa 24 ore); il nome Spare One evidenzia questa sua vocazione... Può funzionare anche senza SIM per chiamare il numero di emergenza e viene fornito in due versioni: una a 900/1.800 MHz per l'Europa e l'Africa ed una a 850/1.900 MHz per il resto del mondo tranne il Giappone.

se il programma riconosce le cifre corrispondenti al codice di disinserimento chiave, resetta il registro CHIAVE e conclude con il riaggancio del telefono mediante la routine "LINEAOFF", altrimenti salta alla "Vernewkey" per verificare se è richiesta la modifica del codice di disinserimento chiave oppure se è stata digitata la sequenza di cifre di un codice programma per l'inserimento di prefissi o cellulari da parte dell'utente abilitato. La sequenza di digitazione per l'inserimento di un nuovo codice di disinserimento chiave è:

1 2 3 4 # X X X X * X X X X *

Riconosciuta valida la sequenza di cifre per la richiesta di un nuovo codice la routine salta alla "INSERTKEY" routine con la quale, analogamente alla "INSERTCOD", vengono acquisite le cifre del nuovo codice e predisposto il puntatore del banco di memoria di appoggio delle cifre da controllare rinviando poi alla routine "NEWKEY" con la quale viene richiesta la ripetizione delle quattro nuove cifre inserite dall'utente. Se una delle cifre non viene riconosciuta secondo la sequenza prevista, il programma salta alla "LINEAOFF" interrompendo ulteriori inserimenti. Viceversa, se il controllo dà esito positivo, la routine salta alla "APGINKEY" la quale effettua l'inserimento delle cifre del nuovo codice nella EEPROM del PIC.

Se la "Vernewkey" non identifica la richiesta di inserimento nuovo codice, rinvia il programma alla routine "VERINSERTPREFCELL" per la verifica di una richiesta di inserimento/modifica prefisso/cellulare; infatti il codice di disinserimento chiave # 1 2 3 4* viene utilizzato anche come password per l'accesso alla funzione di

inserimento o modifica di prefissi e cellulari.

Questa routine predispone il banco di memoria della EEPROM per il salvataggio delle cifre dei prefissi e cellulari da inserire, poi punta alla routine "SELECTBANCO"; qui viene selezionata la corretta posizione del banco di memoria dove inserire le cifre di un nuovo prefisso o cellulare; se non viene identificato come valido il banco di memoria perché diverso da quello richiesto dall'utente mediante il codice programma, mediante la "LINEAOFF" si determina il riaggancio della linea. Se viene identificato il corretto banco come da richiesta, il programma si porta alla routine "INSERTPREFCELL", la quale inserisce in EEPROM le cifre di un nuovo prefisso o cellulare, oppure verifica se la sequenza di cifre digitate indica che si tratta di una richiesta di cancellazione di un prefisso o cellulare rinviando quindi alla CLEARPREFCELL, o ancora, se si tratta di inserire, modificare o cancellare un ulteriore prefisso o

cellulare (andando per questo alla routine "ALTROPREFCELL"). Per l'inserimento di nuovi prefissi da abilitare, si dovrà digitare il codice programma 402, 403, 404 per prefissi rispettivamente a due, tre, o quattro cifre seguito dalla password che corrisponde al codice di disinserimento chiave, poi dalla cifra "n" corrispondente alla posizione del prefisso nel database, (fino alla posizione 5 per i prefissi a due cifre, e fino alla posizione 8 per i prefissi a tre e quattro cifre) ed infine dal prefisso desiderato, che va abilitato come di seguito indicato:

4 0 2 # # 1 2 3 4 n X X #

per prefissi a due cifre e

4 0 3 # # 1 2 3 4 n X X X #

per prefissi a tre cifre.

Invece, per i prefissi composti da quattro cifre il comando da usare è:

4 0 4 # # 1 2 3 4 n X X X X #

(ricordate che le X rappresentano le cifre del nuovo prefisso da inserire, mentre 1234 sono il codice di disinserimento chiave o password).

L'inserimento di un prefisso in una posizione "n" già occupata sostituisce il dato esistente. La cancellazione di un prefisso si

la sequenza su indicata con "n" uguale al numero di posizione del cellulare da modificare.

Invece, per la cancellazione di un cellulare basterà, come per i prefissi, indicare la posizione nel database del cellulare da eliminare preceduto dalla password e dal codice programma:

4 3 9 # # 1 2 3 4 n

La prima cifra "3" è interpretata come l'inizio della digitazione di un numero di cellulare, pertanto il firmware si porta alla routine "IMPOSTACELL" (con la quale viene memorizzata la prima cifra "3" e vengono impostati i banchi di memoria per il salvataggio delle cifre dei numeri dei cellulari) dopodiché rimanda alla routine "PREFCELLABILIT" che continua con il salvataggio delle cifre, rinviando alle routine "CHECKPREFCELL" e "CHECKCIFRE" le quali si occupano di verificare se il numero di cellulare è nel database dei cellulari abilitati oppure no; nel primo caso il programma si porta alla routine "Standby", mentre nel secondo va alla routine "LINEAOFF", con la quale l'apparecchio telefonico verrà disconnesso dalla linea.

Se la prima cifra composta non è neanche un "3", la "IdentITONO" salva la cifra dell'ipotetico prefisso, imposta i parametri e i banchi di memoria per il controllo dei prefissi abilitati, setta il flag ad indicare che si tratta del controllo di un prefisso e rinvia poi il programma alla routine "PREFCELLABILIT", la quale come per i cellulari effettua il salvataggio delle cifre, questa volta dei prefissi, per poi proseguire con le routine "CHECKPREFCELL" e "CHECKCIFRE" allo scopo di accertare se il prefisso è nel database dei prefissi abilitati oppure no. Come per i cellulari, se il prefisso

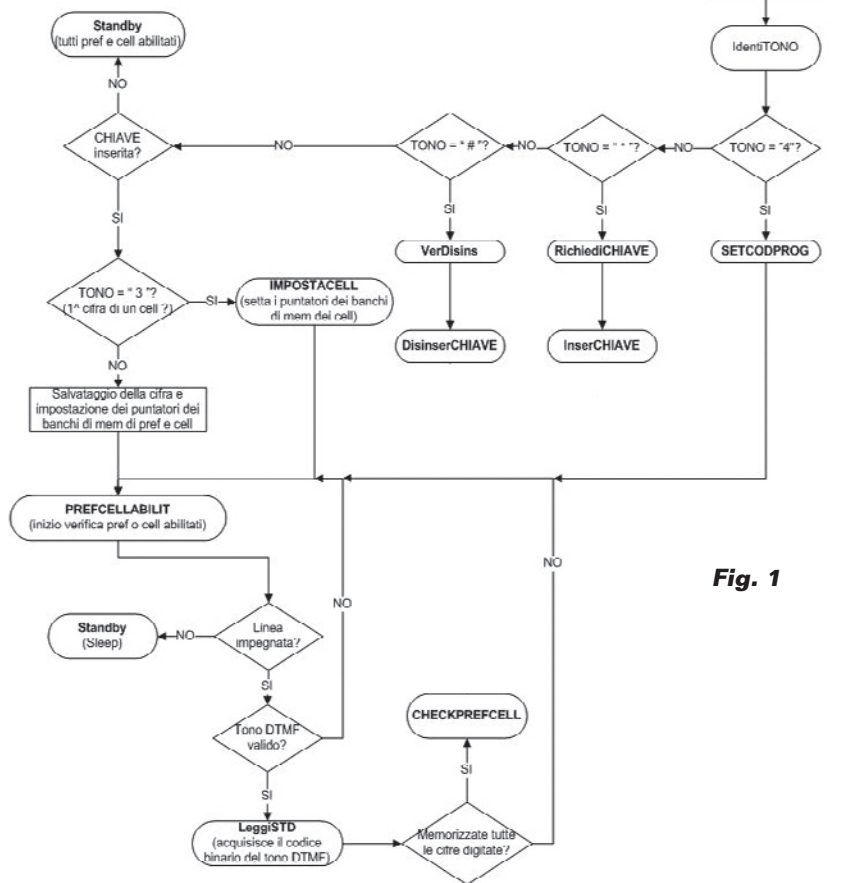


Fig. 1

ottiene ripetendo una delle tre sequenze di caratteri suindicate omettendo le cifre del prefisso e terminando con il carattere asterisco; ad esempio per la cancellazione di un prefisso a due cifre precedentemente inserito nella posizione n=5 si digiterà:

4 0 2 # # 1 2 3 4 5

Un discorso a parte merita l'inserimento di numeri cellulari da abilitare, che possono essere inse-

riti con 439 seguito dalla password, dalla cifra "n" corrispondente alla posizione del numero nel database dei numeri di cellulari, (fino alla posizione 5); in questo caso va inserito l'intero numero a dieci cifre del cellulare desiderato da abilitare:

439##1234nXXXXXXXXXX#

Come per i prefissi, volendo modificare un numero di un cellulare abilitato, basterà ripetere

risulterà abilitato il programma si porterà alla routine "Standby", se no andrà alla routine "LINEAOFF" con la quale l'apparecchio telefonico verrà disconnesso dalla linea (Fig. 1).

REALIZZAZIONE E INSTALLAZIONE

Per il montaggio dei circuiti (trovate le tracce lato rame degli stampati nel nostro sito www.elettronica.in.it) è consigliabile procurarsi un saldatore da 10÷15 W o una stazione saldatrice con regolazione della temperatura, dotati di punta sottile. Iniziate con i componenti a basso profilo (le resistenze e i diodi) quindi proseguite con gli integrati (che è meglio montare su zoccoli) i condensatori (facendo attenzione alla polarità degli elettrolitici), il transistor e, infine, morsettiere e connettori.

L'alimentazione della scheda deve essere fornita da un alimentatore esterno a +12 Vcc da collegare alla morsettiere Vin, il quale deve fornire corrente sufficiente sia a far funzionare il circuito, sia a ricaricare la batteria da 9 V che avrete cura di collegare con una presa a strappo agli appositi contatti previsti sul circuito stampato.

Una volta completato il montaggio occorre programmare il PIC caricandovi il file .hex compilato che trovate nel nostro sito www.elettronica.in.it. Allo scopo scrivetevi dell'ICD2 Microchip, collegandolo al connettore ICSP dello stampato. Per quanto riguarda l'installazione, se avete spazio sufficiente, dopo aver isolato il circuito con una guaina di termorestringente o altro di simile inseritelo nella stessa scatola dove arriva il doppino telefonico, altrimenti inserite lo stampato in un suo contenitore plastico e collocatelo vicino alla borchia della linea telefonica.

In questa, separate i conduttori telefonici del doppino in arrivo dalla

centrale dai cavi degli interni (potete lasciare attaccati solo quelli dell'eventuale filtro ADSL), quindi collegateli in parallelo alla morsettiere LINE IN; collegate ora il doppino della linea a LINE OUT senza curarvi della polarità. Inserite la batteria ricaricabile nell'apposita clip, ricollegate l'alimentatore alla morsettiere CN4 e collegate la spina dell'alimentatore ad una presa a 230V; a questo punto la scheda è pronta per il collaudo funzionale.

CONFIGURAZIONE

La protezione è inserita automaticamente all'accensione e il codice di inserimento predefinito è 31; è abilitata la chiamata solo a prefissi e cellulari abilitati. In ogni altro momento, per inserire la protezione sollevare la cornetta e premere in sequenza i tasti *12#. Notate che 1 e 2 rappresentano un codice qualsiasi scelto da voi. Appena sentite il riaggancio automatico della linea telefonica abbassate la cornetta; la chiave è inserita.

Il codice di disinserimento della protezione predefinito è 2755; disinserendo la protezione si può chiamare qualsiasi numero. La procedura è analoga a quella appena descritta per l'inserimento, ossia dopo aver sganciato il telefono bisogna digitare *1234#; le cifre 1234 rappresentano un codice qualsiasi scelto da un utente abilitato. State attenti che se avete telefoni dotati di display, per evitare la visualizzazione del codice di disinserimento chiave da parte di utenti

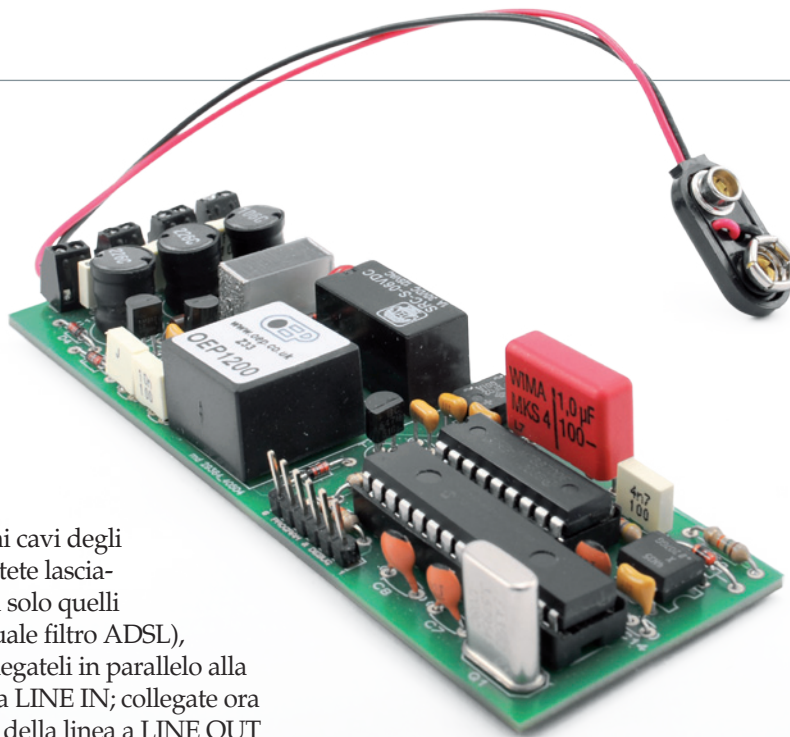
non autorizzati vi conviene cancellare il codice digitato.

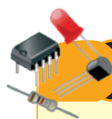
In ogni momento potete modificare il codice di inserimento della protezione: basta sollevare la cornetta, premere in sequenza i tasti *12*XX#XX# e, appena sentite il riaggancio automatico della linea telefonica, abbassare la cornetta. Le X indicano il nuovo codice. Analogo è il processo di cambio del codice di disinserimento, solo che dopo aver sollevato la cornetta bisogna premere in sequenza i tasti

1 2 3 4 # X X X X * X X X X *.

Vediamo ora l'inserimento e la modifica di prefissi e numeri di cellulari abilitati: la programmazione prevede l'inserimento del codice programma di inserimento/modifica dei prefissi abilitati a due, tre e quattro cifre, la password (il codice di disinserimento chiave), il numero "n" di posizione del prefisso nell'elenco prefissi e infine le cifre "X...X" del prefisso da abilitare.

Se si seleziona una posizione "n" già occupata viene cancellato il prefisso ivi memorizzato, che verrà sostituito dal nuovo inserito. Ciò equivale alla funzione di modifica prefissi abilitati.





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 PIC16F876A-I/P; quest'ultimo è anche disponibile già programmato (cod. MF1044) al prezzo di 12,00 Euro. Il chip decodificatore DTMF MT8870 costa invece 7,20 Euro. 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>

Inserimento prefissi a 2 cifre

- 1) Sollevare la cornetta
- 2) Premere in sequenza i tasti

4 0 2 # # 1 2 3 4 n X X #

Si possono inserire fino a 5 prefissi a due cifre quindi "n" va da 1 a 5.

- 3) Appena si ode il riaggancio automatico della linea telefonica abbassare la cornetta, il nuovo prefisso abilitato è inserito.

Inserimento prefissi a 3 cifre

- 1) Sollevare la cornetta.
- 2) Premere in sequenza i tasti

4 0 3 # # 1 2 3 4 n X X X #

Si possono inserire fino a 8 prefissi a tre cifre quindi "n" va da 1 a 8.

- 3) Appena si sente il riaggancio automatico della linea telefonica abbassare la cornetta; il nuovo prefisso abilitato è inserito.

Inserimento prefissi a 4 cifre

- 1) Sollevare la cornetta.
- 2) Premere in sequenza i tasti

4 0 4 # # 1 2 3 4 n X X X X #

Si possono inserire fino ad 8 prefissi a quattro cifre, quindi "n" va da 1 a 8.

- 3) Appena si ode il riaggancio automatico della linea telefonica abbassare la cornetta; il nuovo prefisso abilitato è inserito.

Inserimento cellulari

La programmazione prevede l'inserimento del codice programma inserimento/modifica cellulari abilitati, la password (il codice di disinserimento chiave), il numero "n" di posizione del cellulare nell'elenco dei cellulari abilitati e infine le 10 cifre "X...X" del cellulare.

È bene rammentare che se si seleziona una posizione "n" già occupata, viene cancellato il numero di cellulare ivi memorizzato, che verrà sostituito dal nuovo numero di cellulare inserito. Ciò equivale alla funzione di modifica dei numeri di cellulare abilitati.

- 1) Sollevare la cornetta.
- 2) Premere in sequenza i tasti

4 3 9 # # 1 2 3 4 n X X X X X X X X X #

Si possono inserire fino a 5 cellulari quindi "n" va da 1 a 5.

- 3) Appena si ode il riaggancio automatico della linea telefonica abbassare la cornetta; il nuovo cellulare abilitato è inserito.

Cancellazione dei prefissi e numeri di cellulari

La cancellazione dei prefissi e cellulari prevede la digitazione dei codici programma 4 0 2 #, 4 0 3 #, 4 0 4 #, 4 3 9 # di inserimento/modifica rispettivamente dei prefissi a due, tre e quattro cifre e dei cellulari, la password corrisponde alle cifre del codice di disinserimento chiave 1 2 3 4, il numero "n" di posizione del prefisso/cellulare da cancellare dall'elenco e infine il tasto asterisco * a chiusura dell'operazione di cancellazione.

Cancellazione dei prefissi 2 cifre

- 1) Sollevare la cornetta;
- 2) Premere in sequenza i tasti;

4 0 2 # # 1 2 3 4 n *

- 3) Appena si ode il riaggancio automatico della linea telefonica abbassare la cornetta; il prefisso è cancellato dall'elenco.

Cancellazione dei prefissi 3 cifre

- 1) Sollevare la cornetta.
- 2) Premere in sequenza i tasti

4 0 3 # # 1 2 3 4 n *

- 3) Appena si ode il riaggancio automatico della linea telefonica abbassare la cornetta; il prefisso è cancellato dall'elenco.

Cancellazione dei prefissi 4 cifre

- 1) Sollevare la cornetta.
- 2) Premere in sequenza i tasti

4 0 4 # # 1 2 3 4 n *

- 3) Appena si ode il riaggancio automatico della linea telefonica abbassare la cornetta; il prefisso è cancellato dall'elenco.

Cancellazione dei cellulari

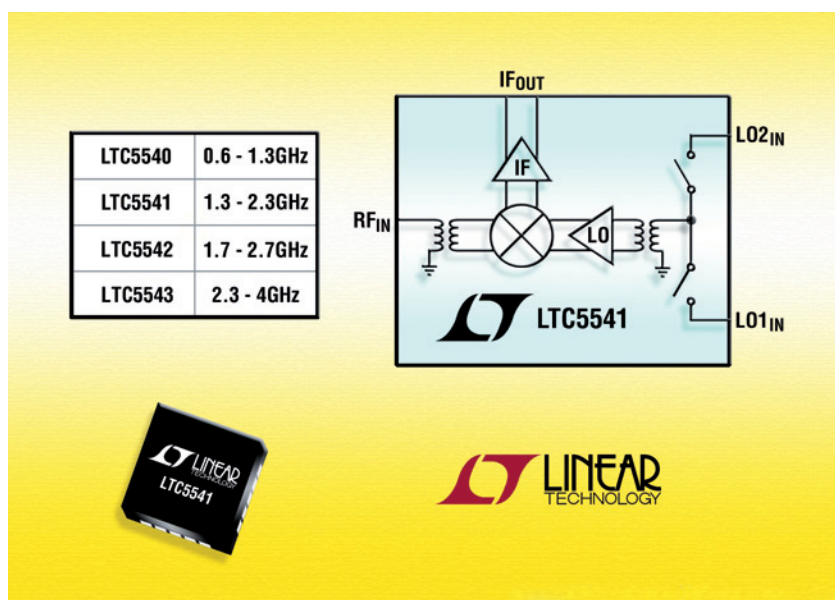
- 1) Sollevare la cornetta
- 2) Premere in sequenza i tasti

4 3 9 # # 1 2 3 4 n *

- 3) Appena si ode il riaggancio automatico della linea telefonica abbassare la cornetta, il cellulare è cancellato dall'elenco.

In tutti i casi, notate che se dopo aver inserito tutte le cifre non si sente nella cornetta il riaggancio della linea telefonica, vuol dire che l'inserimento non è stato effettuato, cioè che l'operazione non ha avuto successo; allora bisogna abbassare la cornetta e ripetere i punti 1, 2 e 3. ■

PIÙ GUADAGNO E MENO RUMORE CON I MIXER PASSIVI



Nelle applicazioni di conversione verso il basso i mixer passivi aumentano il guadagno e riducono il rumore rispetto ai mixer attivi.

di T.Schiltz, B. Beckwith,
X. Wang, e D. Stuetzle

La famiglia LTC554x di mixer downconverter passivi copre frequenze comprese tra 600MHz e 4GHz e offre un guadagno di conversione elevato assieme a una figura di rumore (NF) bassa con elevata linearità. Questi mixer sono adatti per ricevitori di infrastrutture wireless che richiedono un mixer a guadagno elevato per ovviare all'elevata perdita di tensione dei moderni filtri SAW IF ad alta selettività. Mentre i mixer passivi esistenti hanno normalmente una perdita di conversione di 7dB, i nuovi mixer LTC554x sono dotati di amplificatori IF integrati (Fig. 1) che producono un guadagno di conversione totale di 8dB, il che comporta un'ulteriore perdita di 15dB del filtro IF, pur consentendo al ricevitore di soddisfare i requisiti in termini di sensibilità e SFDR (Spurious-Free Dynamic Range).

MIXER ATTIVI E PASSIVI A CONFRONTO

La maggior parte dei mixer a circuito integrato è di tipo attivo o a commutazione di corrente. Linear Technology possiede una vasta gamma di mixer

attivi (es. LT5527 e LT5557) che sono molto diffusi per la loro facilità d'uso e i bassi consumi. Tuttavia i 2dB-3dB di guadagno di conversione non sono sufficienti per alcuni progetti di infrastrutture wireless. Inoltre la figura di rumore dei mixer attivi è normalmente maggiore rispetto a quella dei mixer passivi, a linearità equivalente. I mixer LTC554x utilizzano un core per mixer passivi per ottenere una NF minima ad alta linearità. Nella **tabella 1** vengono confrontate le prestazioni del mixer passivo LTC5541 con quelle del mixer attivo LT5557. Come si evince dalla tabella, il mixer passivo ha un guadagno maggiore di circa 5dB, una NF inferiore di 2dB e un IIP3 maggiore di 1,7dB; però l'LT5557 consuma molto meno.

FIGURA DI RUMORE AD AMPIO SEGNALE

Un altro importante parametro indicativo delle prestazioni del mixer è la figura di rumore ad ampio segnale. Come nel caso dell'amplificatore, la NF di un mixer è il rapporto tra segnale-rumore

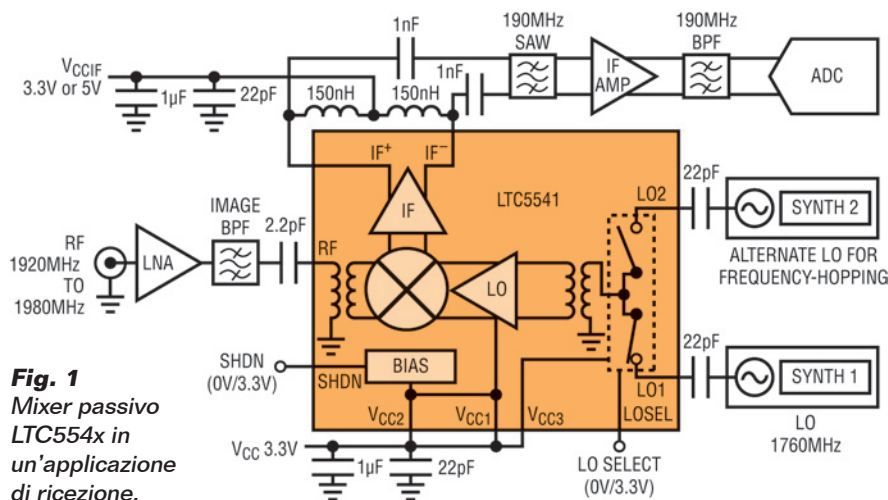


Fig. 1
Mixer passivo
LTC554x in
un'applicazione
di ricezione.

in entrata e segnale-rumore in uscita. Tutti i mixer subiscono una NF maggiore in presenza di segnali RF ad alto livello. Questo fenomeno è definito anche "figura di rumore in caso di blocco" nelle applicazioni di ricezione, dove il segnale "di blocco" è un segnale ad ampiezza elevata in un canale adiacente. La figura di rumore elevata si verifica perché il rumore di fondo in uscita è proporzionale all'ampiezza del segnale in entrata RF, moltiplicato per il rumore del tratto LO ($ARF \cdot NLO$).

Molte volte accade che un ricevitore debba captare un segnale debole in presenza di un forte 'blocker'. Il blocker può comportare un aumento del rumore di fondo tale da causare una perdita del segnale debole nel rumore. La Fig.2 mostra la potenza degli

ingressi NF e RF per l'LTC5540. La NF si avvicina al valore del segnale debole con livelli in ingresso bassi, ma non appena la potenza del segnale RF aumenta, il contributo $ARF \cdot NLO$ diventa dominante e la NF aumenta.

Con un ingresso RF elevato a +5dBm e una potenza LO nominale di 0dBm, la NF aumenta solo di 6dB dal valore del segnale debole (16,2dB). Il grafico dimostra chiaramente che il rumore del segnale forte aumenta con un livello di potenza LO maggiore, il che consente eventualmente di ottenere prestazioni anche migliori.

Se non è possibile eliminare del tutto l'aumento della figura di rumore, è invece possibile migliorare le prestazioni grazie a un'attenta progettazione. Tutte le parti della famiglia LTC554x

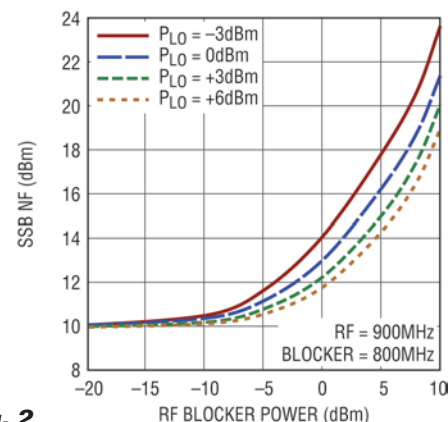


Fig. 2
LTC5540: figura di
rumore vs blocco RF.

hanno un ottimo comportamento della figura di rumore del segnale forte (v. Tabella 2).

CONFRONTO DELLE PRESTAZIONI CALCOLATE NELLA CATENA DI RICEZIONE

I vantaggi dei nuovi mixer passivi sono dimostrati nella seguente analisi della catena di ricezione. Nella Fig. 3 è illustrata la configurazione del ricevitore di una stazione base a unico ciclo di conversione, che viene usata per confrontare le prestazioni di sistema generali quando il mixer attivo LT5557 è usato con lo stesso ricevitore che usa il nuovo mixer passivo LTC5541 (Tabella 3). L'amplificatore IF LTC6400-26, con un guadagno di 26dB, viene usato con la configurazione basata sul 5557, e l'LTC6400-20, con un guadagno di 20dB, viene usato con la configurazione basata sul 5541. In questo modo il guadagno del ricevitore totale rimane praticamente uguale in entrambi i casi. Per ogni caso viene usato un filtro SAW ad alta selettività sull'uscita del mixer, come richiesto dalla stazione base ad elevate prestazioni. Come si evince dalla figura 3, la configurazione del ricevitore che usa il mixer passivo LTC5541 ha una NF inferiore di 0,76dB e un IIP3 maggiore di 1,6dB. Ne conseguono un rapporto

Tabella 1

Part	Gain (dB)	NF (dB)	IIP3 (dBm)	Input P1dB (dBm)	DC Power (mW)
LTC5541 (passive)	7.8	9.6	26.4	11.3	630
LT5557 (active)	2.9	11.7	24.7	8.8	270

Tabella 2

Part	RF Frequency (MHz)	LO Injection	Small-Signal NF (dB)	Large-Signal NF (dB)
LTC5540	900	High-Side	9.9	16.2
LTC5541	1950	Low-Side	9.6	16.0
LTC5542	2400	Low-Side	9.9	17.3
LTC5543	2500	High-Side	10.2	17.5

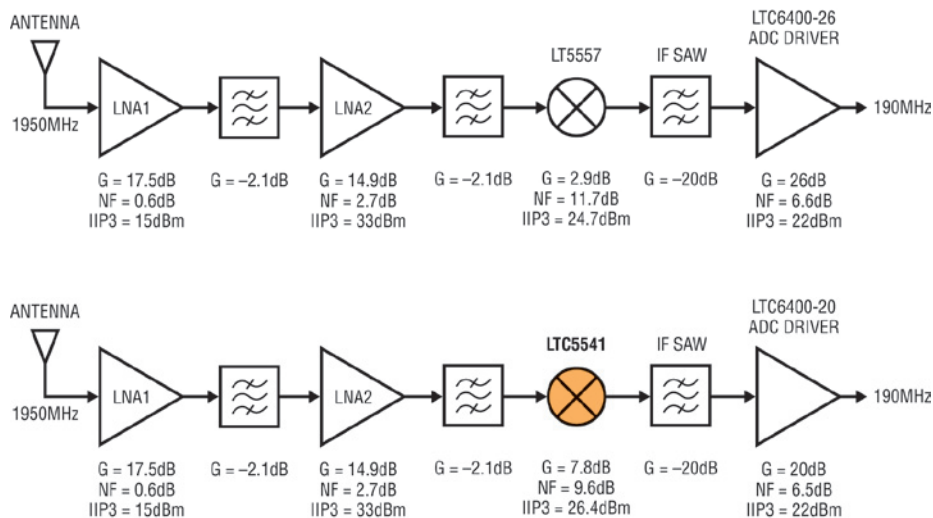
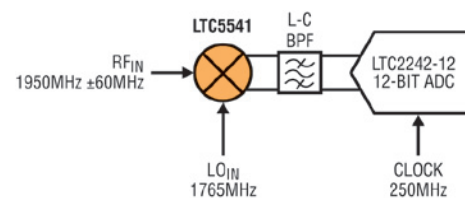


Fig. 3 - Confronto tra la tipica configurazione di un ricevitore per stazione base wireless con utilizzo dell'LT5557 e dell'LTC554.

Fig. 4 - Diagramma a blocchi del prototipo di un ricevitore DPD.



segnale-rumore (SNR) e un SFDR (Spurious-Free Dynamic Range) maggiori per il ricevitore basato sul 5541.

CONFRONTO DELLE PRESTAZIONI MISURATE IN UN'APPLICAZIONE DPD DI TRASMISSIONE

Nella sua forma più semplice un ricevitore digitale a conversione singola è formato da un mixer downconverter, un filtro passa-basso o passabanda e un convertitore analogico-digitale (ADC). Questo tipo di ricevitore può essere usato come ricevitore a pre-distorsione digitale (DPD) nei trasmettitori delle stazioni base ad alta linearità. In questa applicazione i parametri prestazionali più importanti sono linearità, gain flatness, ampia

larghezza di banda IF e, naturalmente, semplicità. A differenza dell'applicazione di ricezione descritta in precedenza, la NF non è un fattore determinante nelle applicazioni DPD a causa del segnale a grande ampiezza abbinato dall'uscita del trasmettitore. I mixer LTC554x sono perfetti per le applicazioni di ricezione DPD per la loro elevata linearità, l'alto guadagno di conversione e la risposta dell'uscita IF piatta rispetto alla frequenza. Nella Fig. 4 è illustrato il prototipo di un ricevitore DPD che utilizza l'LTC5541. Questo ricevitore è stato costruito e testato per un'applicazione da 1,95GHz con una IF a larga banda di 185 ± 60MHz. Ai fini del confronto è stato costruito un altro rice-

vitore usando il mixer attivo LT5557. Il ricevitore DPD basato sull'LT5557 necessitava di un amplificatore IF esterno prima del filtro passabanda per compensare il guadagno inferiore di 5dB del mixer attivo. Il vantaggio principale offerto dall'LTC5541 è che rende superfluo questo amplificatore IF. Inoltre, come indicato nella Tabella 4, il ricevitore DPD basato sull'LTC5541 garantiva un SNR e un IIP3 maggiori e una distorsione armonica inferiore.

CONCLUSIONI

La nuova famiglia LTC554x di mixer downconverter passivi offre le alte prestazioni necessarie per i ricevitori delle infrastrutture wireless moderne. La combinazione tra guadagno di conversione elevato, bassa NF, ottima NF in fase di blocco e alta linearità può migliorare il rapporto segnale-rumore e l'SFDR. Le ottime prestazioni contribuiscono inoltre a migliorare le prestazioni del ricevitore DPD, mentre la copertura della gamma di frequenze comprese tra 600MHz e 4GHz della famiglia LTC554x rende questo dispositivo la soluzione ideale per numerose applicazioni di ricezione.

Tabella 3

LINE-UP	GAIN (dB)	NF (dB)	IIP3 (dBm)
LT5557-BASED	35.0	4.03	-1.6
LTC5541-BASED	33.9	3.27	0.0

Tabella 4

Mixer	0.5dB	Input Level at -1dBFS	SNR at -1dBFS	HD2 at -7dBFS	IM3 at
LTC5541	IF BW	-0.6dBm	63.4dB (120MHz)	-54.5dBc @ 123MHz -78.2dBc @ 184MHz -69.5dBc @ 243MHz	-7dBFS
LT5557	126MHz	-1.8dBm	62.8dB (120MHz)	-52.4dBc @ 123MHz -63.1dBc @ 184MHz -67.4dBc @ 243MHz	-64.8dBc

Riparte la stagione delle competizioni di robotica

I mesi primaverili, in particolare marzo e aprile, sono i mesi durante i quali si svolgono le più importanti gare di robotica tra team di studenti.

Dopo un inverno passato a progettare, costruire e programmare piattaforme robotiche di ogni tipo, le squadre di studenti si confrontano tra loro mettendo sui campi di gara - oltre ai robot - grinta e passione, nel tentativo di superare gli avversari di turno e di qualificarsi per le finali mondiali.

Venerdì 8 e sabato 9 marzo va in scena la finale nazionale della First Lego League, un campionato a squadre tra ragazzi dai dieci ai sedici anni, che utilizzano robot Lego per compiere delle missioni in maniera completamente autonoma. Il percorso di gara con le missioni viene fornito ai concorrenti almeno 2 mesi prima all'atto dell'iscrizione in modo che questi possano testare le soluzioni adottate con largo anticipo e scegliere la strategia di gara adeguata. Nel nostro paese la manifestazione è organizzata dal Museo Civico di Rovereto e la finale si svolge proprio in questa cittadina del Trentino. Dal 20 al 22 marzo si svolgerà a Roma RomeCup gara tra squadre provenienti da tutta Italia nelle categorie Soccer, Dance, Theatre, Explorer e Rescue. Infine, dal 17 al 20 aprile, è in programma a Pescara la finale italiana della RoboCup Junior con la partecipazione di ben 129 squadre provenienti da tutta Italia. A proposito di RoboCup, ed in particolare della categoria Rescue, nelle pagine seguenti ci occupiamo delle nuove regole introdotte quest'anno dal Comitato Internazionale che avranno un notevole impatto sulle strategie di gara e sulla progettazione dei robot.



IN QUESTO NUMERO

126 ■ EVENTI

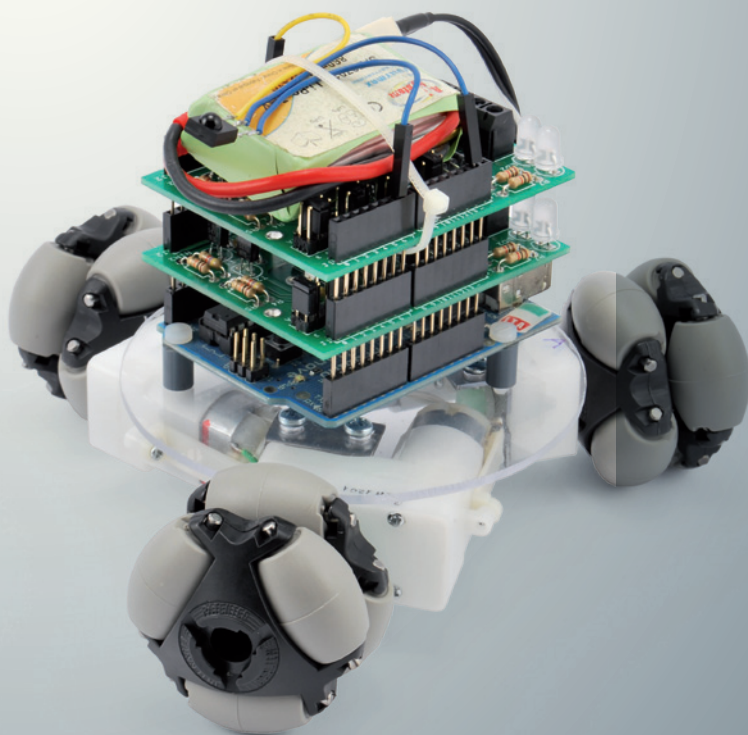
Verso la finale nazionale di Pescara: i team della Rescue A alla prova delle nuove regole in vigore per le gare del 2013.

129 ■ OMNIWHEELS

Piccola piattaforma robotica basata su Arduino che utilizza tre ruote di tipo OmniWheels che consentono anche uno spostamento laterale senza che sia necessario sterzare. Prima puntata.

134 ■ NEWS

Dall'esoscheletro pensato per gli operatori che si muovono sul luogo dei disastri naturali ai robot che riparano i satelliti in orbita: tutte le news dal mondo della robotica.



NUOVE REGOLE PER LE GARE DI RESCUE DELLA **RoboCup Junior**

**Verso la finale nazionale di Pescara: i team della Rescue A
alla prova delle nuove regole in vigore per le gare del 2013.**

di ARSENIO SPADONI



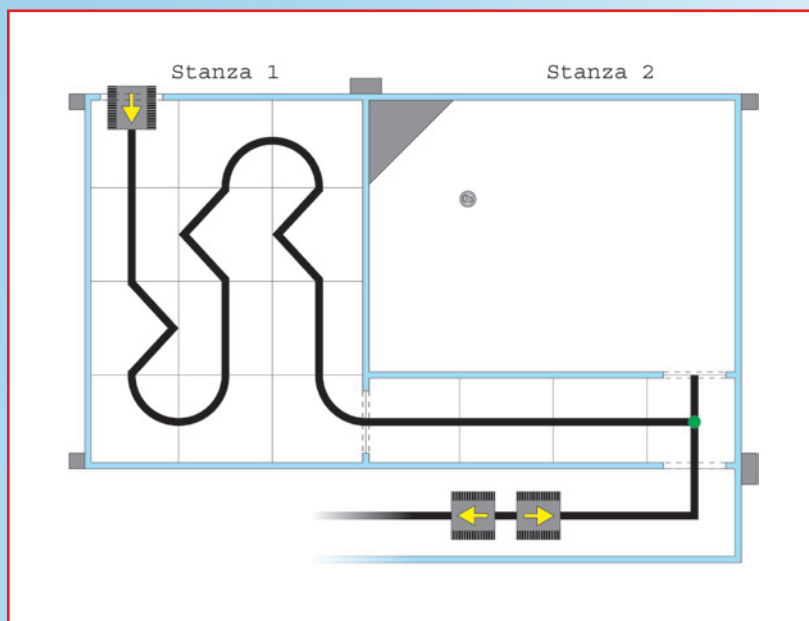
A poco più di un mese dalla finale nazionale di Pescara della RoboCup Jr Italia, e a cinque mesi dalla gara mondiale di Eindhoven, i numerosi team che si sfideranno nella categoria Rescue A sono impegnati a modificare l'hardware ed il software dei propri robot per adattarli alle nuove regole di gara introdotte all'inizio di quest'anno. Infatti, dopo un periodo abbastanza lungo durante il qua-

le il regolamento era rimasto pressoché immutato, quest'anno sono state introdotte nuove norme che hanno obbligato i team ad adottare soluzioni in grado di affrontare le nuove difficoltà introdotte dal regolamento 2013.

La Rescue è una specialità che ha come scopo la realizzazione di un robot in grado di muoversi all'interno di uno scenario "catastrofico" superando ostacoli di varia natura fino a identificare e porre in salvo una vittima dell'evento. Il tutto, ovviamente, in maniera completamen-

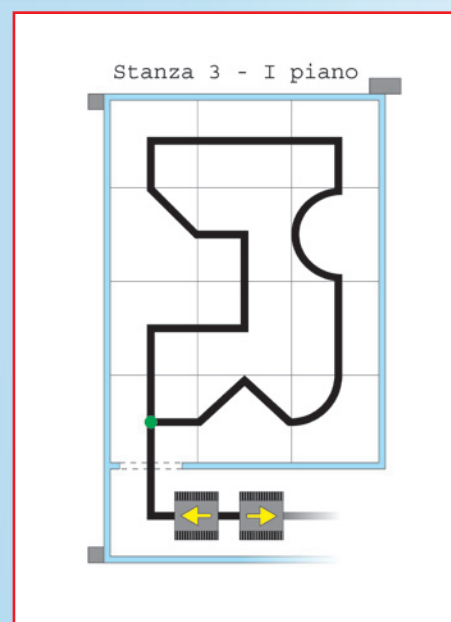
te autonoma. Il robot segue una traccia sul terreno costituita da una striscia nera e si muove all'interno di tre stanze nell'ultima delle quali è posta la vittima rappresentata da una lattina di bibita. Durante questa fase il robot deve identificare la vittima all'interno della stanza, la deve afferrare con un apposito braccio e la deve porre su una pedana situata in un'angolo della stanza; anche la posizione della pedana non è sempre la stessa e va identificata. Due stanze si trovano al piano inferiore, la terza al piano superiore, raggiungibile mediante una rampa. Fino all'anno scorso la terza stanza (quella con la vittima) era quella posta al piano superiore per cui il robot, dopo aver percorso la rampa in salita, concludeva la sua prova cercando la lattina, afferrandola e ponendola sull'apposita pedana. Da quest'anno la stanza con la vittima è la numero due ed il percorso prevede la salita mediante la rampa alla stanza superiore, un percorso più o meno accidentato in questa zona e la discesa lungo la rampa stessa per accedere alla seconda stanza dove ora è posta la vittima. La difficoltà introdotta dal nuovo regolamento consiste dunque nella capacità del robot di scendere lungo la rampa (oltre che di salire).

Un'altra difficoltà è rappresentata dalle intersezioni previste dal nuovo regolamento, necessarie per poter seguire il percorso prestabilito; in pratica alcune linee del percorso si intersecano ed il robot, quando arriva a questo punto, deve sempre prendere la prima traccia a destra o, nel caso questa non esista, andare dritto. L'intersezione è segnalata da un bollino verde del diametro di circa 4 centimetri. È evidente, dunque, che i robot che si cimenteranno in questa gara dovranno disporre di un sensore di colore in grado di rilevare la presenza dell'intersezione, attivando una strategia di gara che metta in atto quanto previsto dal nuovo regolamento. Alcune categorie di robot (come quelli della serie Lego) dispongono già di sensori di colore mentre in altri casi (come per i robot autocostruiti) questa nuova funzionalità hardware va aggiunta costruendo il sensore oppure facendo ricor-



so ad un sensore commerciale. Nei prossimi numeri della rivista ci cimenteremo anche noi nella progettazione e costruzione di sensori di questo genere che possono essere realizzati anche in maniera molto semplice utilizzando tre LED colorati, una fotoresistenza ed un circuito di controllo. Una soluzione un "tantino" più professionale prevede l'impiego di un economico chip programmabile TCS3200D della TAOS in grado di fornire un segnale molto preciso. Questo chip dispone di un array di fotodiodi con filtri colorati (blu verde e rosso); l'utilizzo è molto semplice e prevede, oltre al chip, l'impiego di quattro LED a luce bianca.

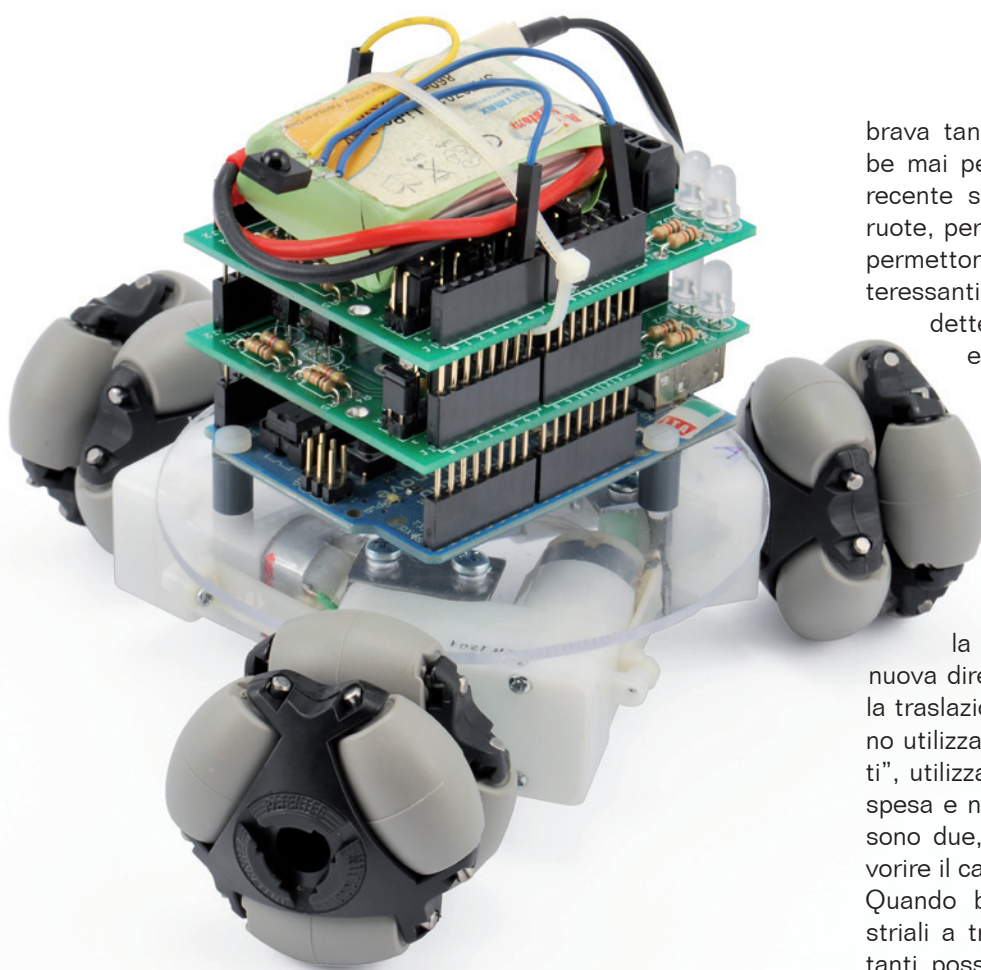
Tornando alla gara di Pescara, alla manifestazione parteciperanno nella categoria Rescue A Under 19 (ovvero i ragazzi delle scuole medie superiori) ben 57 squadre mentre saranno 13 i team della Under 14 (ovvero i ragazzi delle scuole medie inferiori). Entrambe le categorie gareggeranno sullo stesso percorso che, nel secondo caso, sarà soltanto meno accidentato; inoltre, nella evacuation room, non è prevista la pedana dove porre la lattina ma semplicemente uno spazio, in un angolo, allo stesso livello del pavimento. □



OmniWheels

Piccola piattaforma robotica basata su Arduino che utilizza tre ruote di tipo OmniWheels che consentono anche uno spostamento laterale senza che sia necessario sterzare. Prima puntata.

dell'Ing. MIRCO SEGATELLO



brava tanto perfetta che nessuno avrebbe mai pensato di migliorarla, eppure di recente sono state realizzate particolari ruote, per certi aspetti rivoluzionarie, che permettono di aggiungere nuove ed interessanti funzionalità. Si tratta di ruote dette "omnidirezionali" e costituite essenzialmente da un insieme di normali ruote assemblate in modo tale da permettere nuovi movimenti.

Una normale ruota è capace di ruotare rispetto il suo asse ma non di traslare lateralmente; per modificare la direzione di marcia è necessario che la ruota sterzi rivolgendosi nella nuova direzione di marcia. Per consentire la traslazione in tutte le direzioni si possono utilizzare le cosiddette ruote "pivottanti", utilizzate ad esempio nei carrelli della spesa e nei passeggini, dove tipicamente sono due, montate anteriormente per favorire il cambio di direzione.

Quando bisogna realizzare carrelli industriali a trazione elettrica, le ruote pivottanti possono non essere più sufficienti; la soluzione a questo problema è stata trovata in speciali ruote capaci di esercitare la trazione in una direzione ma capaci, nel contempo, di scorrere in altre direzioni senza necessariamente doverle sterzare. Da questa idea, negli anni settanta sono nate le prime ruote omnidirezionali; ci sono voluti anni per perfezionarle e pro-

L' invenzione della ruota ha letteralmente rivoluzionato la storia dell'uomo perché, grazie ad essa, è stato possibile creare mezzi di trasporto, macchinari industriali e via di seguito; la ruota sem-

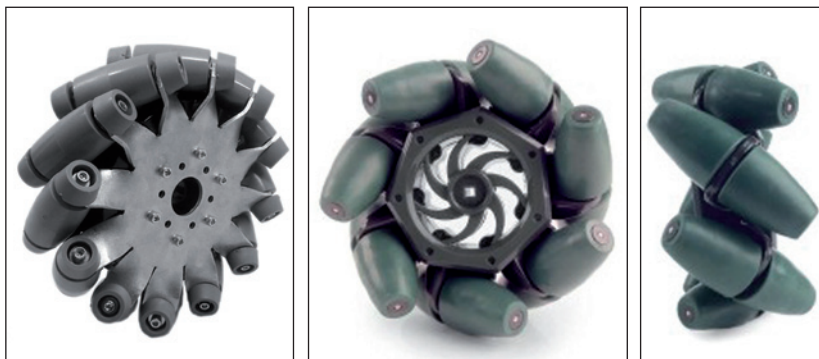


Fig. 1 - Mecanum wheels

porle al mercato a prezzi vantaggiosi, così come c'è voluto del tempo per poterle utilizzare in applicazioni utili nella normale attività quotidiana. La prima versione di questo tipo di ruote fu chiamata *mecanum wheels*, costituita da una ruota nella cui circonferenza è posizionata una serie di rulli il cui asse di rotazione ha un'inclinazione di quarantacinque gradi rispetto alla tangente della ruota. Questa ruota presenta un'elevata trazione lungo l'asse di rotazione ma pone un attrito nullo se traslata di 45° rispetto a tale asse. La Fig. 1 mostra questo tipo di ruota in vari sviluppi.

Il fatto di avere trazione in una direzione ma attrito nullo in un'altra direzione consente di aumentare i gradi di libertà, permettendo un movimento che gli anglosassoni definiscono come *Holonomic movement*. Con quattro di queste ruote è possibile realizzare una piattaforma che ha la possibilità di spostarsi in qualsiasi direzione mantenendo la parte frontale rivolta nella direzione desiderata (Fig. 2).

Le applicazioni più classiche consistono nei ponteggi semoventi (Fig. 3) e nei carrelli elevatori (Fig. 4).

Con questo tipo di ruote è garantita un'ottima robustezza, una buona trazione nella

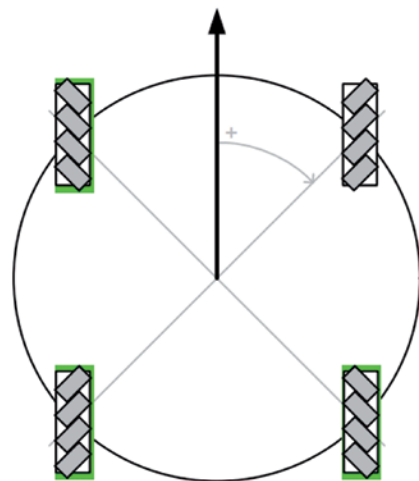


Fig. 2 - Piattaforma con mecanum wheels.

direzione di marcia e una moderata difficoltà di pilotaggio.

Successivamente sono state introdotte le OmniWheels (Fig. 5) costituite da due (ma anche tre) ruote ciascuna composta da una serie di rulli il cui asse di rotazione è posto tangenzialmente alla ruota principale. Queste ruote presentano la massima trazione nella direzione tangenziale alla ruota, mentre l'attrito è nullo rispetto all'asse di rotazione della ruota; questo significa che tali ruote possono essere traslate lateralmente senza alcuno sforzo.

Con queste ruote è possibile realizzare delle piattaforme semoventi con tre o quattro ruote (Fig. 6), risparmiando sui costi rispetto la versione con le ruote viste prima. Si perde però la capacità di sfruttare tutta la coppia dei motori e nel caso

Fig. 5
Esempi di ruote
OmniWheels



Fig. 3 - Carrello semovente.



Fig. 4 - Carrello elevatore.



Fig. 6 - Configurazioni di piattaforma con OmniWheels.

di superfici sconnesse non si riesce a garantire l'esatta direzione di avanzamento, per questo motivo non ancora utilizzate a livello commerciale.

Con queste ruote è però possibile realizzare delle semplici piattaforme a tre ruote, specialmente robot di piccole dimensioni, come ad esempio quelli che gareggiano nelle gare di soccer alla RoboCup.

Esiste una terza possibilità per realizzare piattaforme a movimento omnidirezionale, che consiste nell'utilizzare le più recenti ruote denominate *Swerve Crab Drive* (Fig. 7); queste sono essenzialmente delle ruote che, oltre alla classica rotazione, hanno la possibilità di girare attorno al loro diametro. Per evitare che i fili del motore si intreccino, il meccanismo di trazione è convogliato tramite ingranaggi attraverso il perno di fissaggio. Un secondo meccanismo permette alla ruota di girare su se stessa, quindi sono necessari in totale da 5 a 8 motori (per una piattaforma a quattro ruote), a seconda che le ruote possano sterzare tutte o solo alcune.

Una simile soluzione implica costi di realizzazione non indifferenti, tuttavia la libertà di movimento che ne deriva è notevole, perché ogni ruota gira esattamente secondo il verso di spostamento della piattaforma, riuscendo a scaricare a terra tutta la potenza trasmessa dal motore; inoltre la

piattaforma è insensibile ai terreni sconnessi e la manovrabilità e la precisione sono le massime possibili.

Al momento l'unico vero impiego pratico consiste nei rover che la NASA ha inviato su Marte, ma sicuramente a breve si osserveranno anche applicazioni per un uso più su larga scala.

Il nostro robot

Dopo esserci documentati sulla funzionalità di questi tipi di ruota abbiamo pensato di realizzare una piccola piattaforma robotica utilizzando tre ruote del tipo OmniWheels, disposte con un angolo tra loro di 120° . L'obiettivo era una piattaforma economica, facile da assemblare, basata su Arduino, da realizzare con componenti recuperati da altri progetti.

Le ruote omnidirezionali disponibili sul mercato sono in generale assai costose, quindi abbiamo preferito utilizzare delle ruote disponibili a catalogo RS Components (codice 408-542) con diametro di 48 mm; si tratta di mezze ruote, perché per realizzare una ruota omnidirezionale completa è necessario assemblare due di queste ruote sfalsandole di 60° ; sono quindi necessarie sei ruote di codice 408-452 per realizzare la nostra piattaforma. Come motori abbiamo utilizzato tre motoriduttori



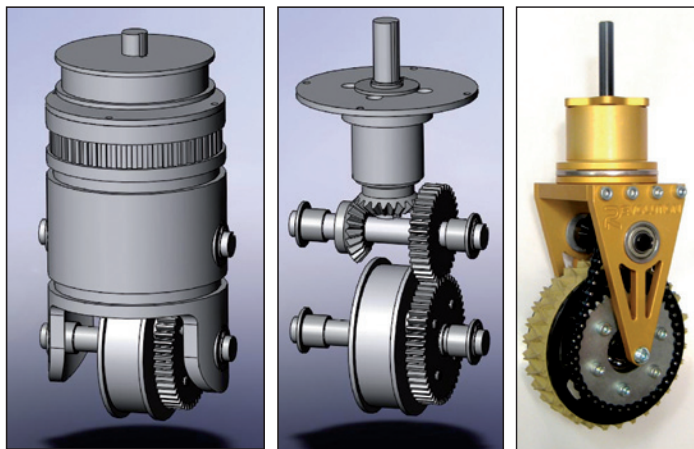


Fig. 7
Swerve Crab
Wheels.

forniti da Futura Elettronica (codice MOTORID1201) mentre per il loro controllo abbiamo utilizzato due motorshield della stessa Futura Elettronica (codice MOTORSHIELD_FE). Abbiamo deciso di utilizzare questo shield perché ci dà la possibilità di configurare i pin per il controllo del driver di potenza; in questo modo è possibile utilizzare contemporaneamente due shield, così da poter controllare un totale di quattro motori. Per la configurazione degli shield fate riferimento alla **Tabella 1**.

È indispensabile rispettare le indica-

Tabella 1 - Assegnazione pin ai due Shield (Cod. MOTORSHIELD_FE) utilizzati.

Shield	pin	Connessione Arduino	Motore
1	PWMA	3	A
1	DIRA	2	A
1	PWMB	9	B
1	DIRB	8	B
2	PWMA	5	Non usato
2	DIRA	4	Non usato
2	PWMB	10	C
2	DIRB	12	C

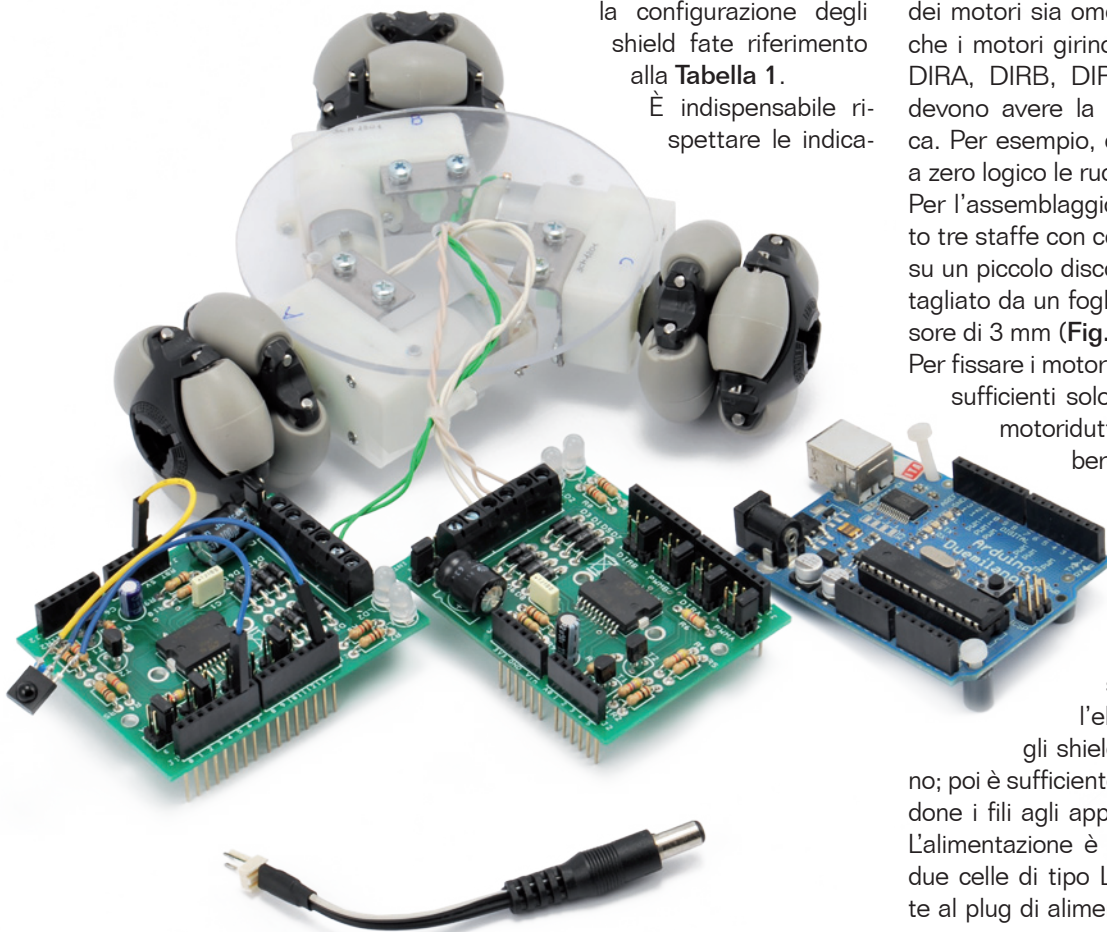
zioni in quanto tutti e tre i motori utilizzati devono essere alimentati allo stesso modo; nel nostro caso, con un segnale PWM alla stessa frequenza.

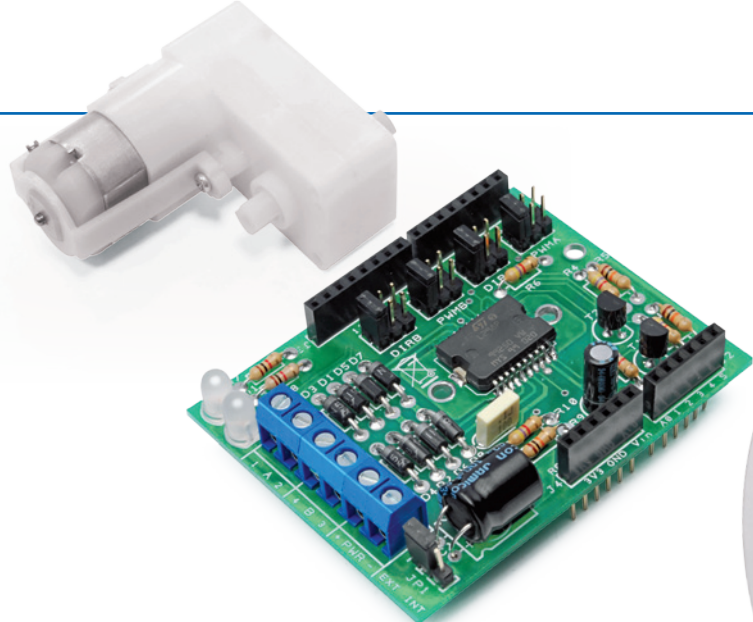
Nella **Tabella 2** sono riportati i registri e i timer usati dalla funzione *analogWrite*, che permette di impostare le uscite a PWM. Il segnale PWM facente capo ai pin 5 e 6 utilizza il registro *Timer0* che però genera un segnale PWM con frequenza doppia rispetto agli altri, causando leggerezza nel funzionamento dei motori.

È fondamentale che la direzione di rotazione dei motori sia omogenea; per fare in modo che i motori girino nello stesso senso, i bit DIRA, DIRB, DIRC dei driver dei motori devono avere la stessa impostazione logica. Per esempio, quando tali bit sono posti a zero logico le ruote girano in senso orario. Per l'assemblaggio dei motori abbiamo usato tre staffe con codice GM28SUPP fissate su un piccolo disco del diametro di 10 cm ritagliato da un foglio di plexiglass dello spessore di 3 mm (**Fig. 8**).

Per fissare i motori e la scheda Arduino sono sufficienti solo pochi fori; nel montare i motoriduttori fate in modo che l'albero di ciascuno sia disposto a centoventi gradi rispetto a quello degli altri e perfettamente indirizzato verso il centro della piattaforma.

Il cablaggio è molto semplice in quanto per l'elettronica basta impilare gli shield l'uno sull'altro, su Arduino; poi è sufficiente cablare i motori collegandone i fili agli appositi morsetti degli shield. L'alimentazione è fornita da una batteria a due celle di tipo LiPo connessa direttamente al plug di alimentazione di Arduino, la cui





tensione, anche a batteria completamente carica (oltre 8 V), non crea problemi grazie alla caduta di tensione dovuta al diodo interno ad Arduino e alle perdite sul driver di potenza. Per completare l'hardware è necessario installare un sensore IR del tipo PNA4602 che utilizzeremo come ricevitore per un controllo a distanza molto economico con tecnologia ad infrarossi (Fig. 9).

È sufficiente cablare l'alimentazione del sensore ai pin +5V e GND di Arduino mentre l'uscita va connessa al pin digitale 6. Per filtrare eventuali disturbi provenienti dai motori abbiamo saldato un piccolo condensatore da 100 nF direttamente sui pin di alimentazione del sensore IR. Per il fissaggio delle ruote all'asse dei motori dovrete ingegnarvi un po' di più, perché l'albero dei motoriduttori ha un diametro inferiore a quello delle ruote: serve quindi un piccolo adattatore o della buona colla. Per il controllo a distanza di questa piattaforma abbiamo pensato ad una soluzione economica ma molto funzionale, che consiste nel ricicla-

re il telecomando a infrarossi di un elicottero giocattolo; questa soluzione è ideale in quanto si può entrare in possesso di oggetti del genere spendendo davvero "quattro soldi". Visto che non è facile far volare questi elicotteri, è molto facile ritrovarsi con il solo telecomando... Esistono in commercio diverse versioni di comandi a distanza per elicotteri giocattolo, tutti accomunati dal tipo di tecnologia impiegata: la trasmissione seriale tramite modulazione di raggi infrarossi. Nel nostro caso disponevamo del telecomando a raggi infrarossi di un piccolo elicottero a tre canali del valore commerciale di 20 Euro (Fig. 10). Nella prossima puntata spiegheremo come acquisire il codice utilizzato in questo telecomando ed integrarlo nel firmware utilizzato per controllare la nostra piccola piattaforma robotica. □



Fig. 8 - Per l'assemblaggio dei motori abbiamo usato tre staffe fissate su un piccolo disco del diametro di 10 cm ritagliato da un foglio di plexiglass.

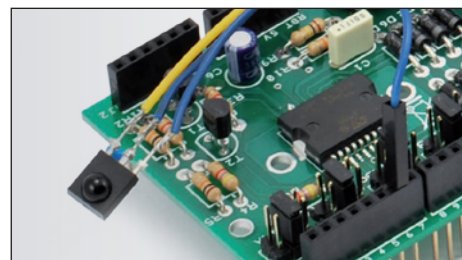


Fig. 9 - Il sensore IR PNA4602.



Fig. 10 - Elicottero giocattolo e relativo telecomando.

Tabella 2 - Registri associati al PWM di Arduino.

pin	funzione	shield	Timer	Registro	Frequenza PWM
3	PWMA	1	2	OC2B	488,28
5	PWMC	2	0	OC0B	976,56
6	Non usato		0	OC0A	976,56
9	PWMB	1	1	OC1A	488,28
10	PWMD	2	1	OC1B	488,28
11	Non usato		2	OC2A	488,28

Presentiamo una nuova applicazione in grado di gestire chiamate GSM ed SMS, affrontiamo la creazione di un servizio Android ed approfondiamo l'utilizzo di alcuni sensori del nostro smartphone, come accelerometro e localizzatore GPS. Sesta puntata.

di Andrea Chiappori

6

PROGRAMMIAMO CON ANDROID

Dopo aver spiegato l'interazione tra il modulo Bluetooth RN-42 e Android, ora vedremo come gestire direttamente chiamate e messaggi GSM e in che modo interagire con i sensori di movimento e localizzazione dello smartphone allo scopo di pilotare lo shield Arduino GSM presentato nelle puntate precedenti. In particolare, dopo questa lezione saremo in grado di attivare dispositivi esterni collegati allo shield, tramite una chiamata gestita internamente dalla nostra applicazione, ed impartire diversi comandi grazie ad SMS personalizzati.

Potremo dotare la nostra applicazione di maggiori funzionalità: ad esempio far partire una chiamata verso un numero telefonico prestabilito in seguito ad uno "scuotimento" ripetuto dello smartphone, oppure appena questo si troverà in prossimità di una certa zona geografica, anch'es-

sa impostabile (Fig. 1). Questo ci dà l'occasione di fare una panoramica sui sensori di movimento e sul localizzatore GPS di Android, ma anche di introdurre un nuovo concetto Android che è il "servizio", il quale ci risparmia di dover tenere sempre attiva e a pieno schermo la nostra applicazione durante il suo ciclo di vita. Anche se questa applicazione nasce per scopo didattico, potrà comunque tornarci utile per realizzare automatismi comandati a distanza, come ad esempio un apricancello o un avviso di presenza.

SERVIZIO ANDROID

Mentre nell'applicazione descritta nel fascicolo precedente (*RGBLedController*) abbiamo sempre avuto a che fare con le Activity, che sono classi dotate di un'interfaccia grafica verso l'utente e con le quali è possibile interagire attivamente, questa volta occorre fare in modo che il nostro codice funzioni in background,

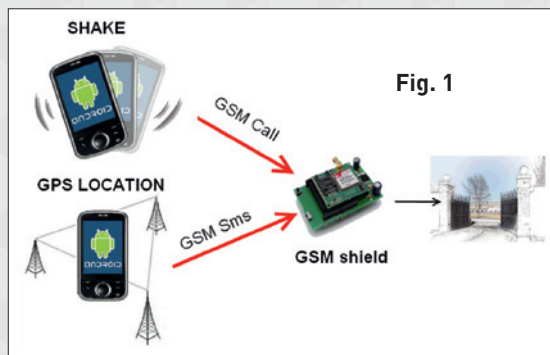


Fig. 1

permettendoci di utilizzare il nostro dispositivo normalmente, ma rimanendo comunque in attesa di alcuni eventi esterni (come segnali da sensori di movimento, luminosità, GPS ed altri) ai quali esso reagirà di conseguenza.

Per fare ciò useremo appunto un servizio, che in Android altro non è che una classe (e quindi un file) estesa della classe base *Service*.

Prima di addentrarci nella creazione del nostro primo servizio, creiamo un nuovo progetto seguendo gli stessi passi delle puntate precedenti, anche perché la prima classe con cui viene lanciata l'applicazione sarà sempre una Activity (quindi dotata di interfaccia grafica) attraverso la quale potremo avviare il servizio vero e proprio, arrestarlo e soprattutto configurarlo a nostro piacimento, ad esempio per cambiare i numeri di telefono di destinazione, la sensibilità dei sensori ed altre opzioni che vedremo più avanti.

Quindi, inizialmente aggiungeremo due semplici pulsanti Start e Stop con cui avvieremo e arresteremo il servizio, ed una *TextView* che ci indicherà lo stato del servizio (Fig.1).

Senza scendere nei dettagli, associeremo alla pressione di due pulsanti le funzioni di sistema

con cui potremo lanciare e arrestare il servizio dall'Activity.

Come visibile in Fig. 2, per avviare il servizio occorre creare, sempre nell'Activity principale, un Intent al quale passeremo il nome della nostra classe *GSMService* (che poi è lo stesso nome del file *GSMService.java*) e che aggiungeremo al progetto nella sottocartella *src*.

Successivamente chiameremo il metodo *startService()* passandogli come parametro l'Intent appena creato. Analogamente, per fermare il servizio useremo il metodo *stopService()* usando sempre lo stesso Intent.

Soffermiamoci ora sulla scrittura del nostro servizio, ovvero del file *GSMService.java*.

Per facilitare la comprensione abbiamo aggiunto i messaggi Toast all'ingresso delle funzioni che gestiscono il servizio e che ci segnalano il ciclo di vita del servizio.

In Fig. 3 è mostrato lo scheletro di un servizio che ora andremo a completare secondo le nostre esigenze. In seguito al lancio del servizio tramite *startService()*, questo viene prima creato e subito dopo avviato. In queste fasi (rispettivamente metodo *onCreate()* e *onStart()*) inseriremo il codice per permettere lo svolgimento delle azioni che vogliamo siano eseguite in background. Solitamente andremo a registrare alcune particolari classi dette "Listener" che si metteranno in ascolto di particolari eventi del sistema operativo e che vedremo in seguito.

Il metodo *onDestroy()* viene chiamato quando il servizio termina in seguito alla chiamata del metodo *stopService()* dell'Activity, oppure a quella

Fig. 2

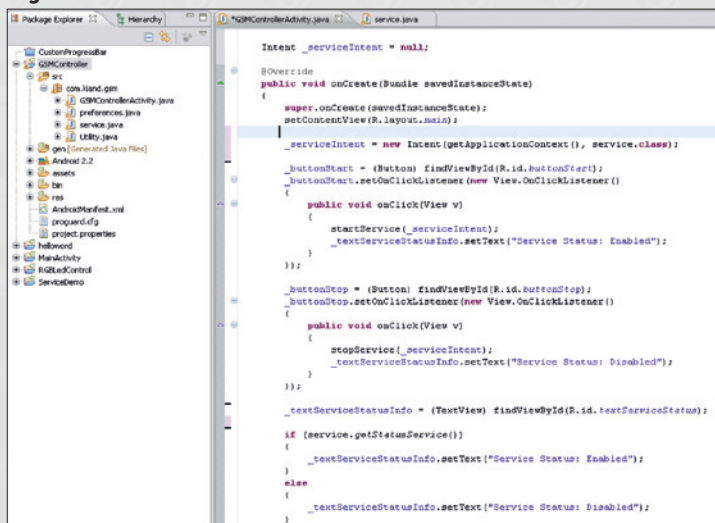
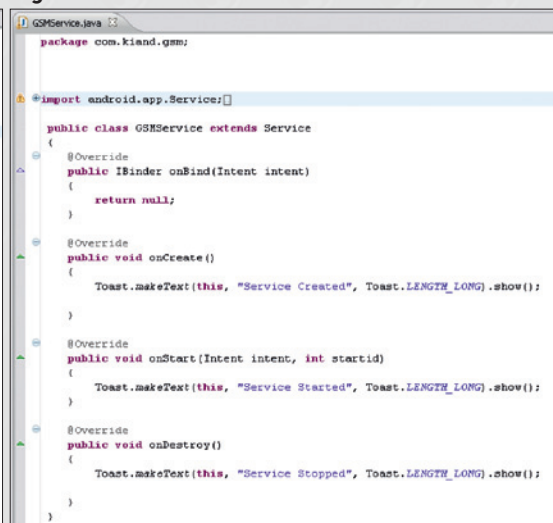


Fig. 3



del metodo *stopSelf()* interno alla classe *Service*; in questa fase vengono anche deregistrate le classi *Listener*.

Infine il metodo *onBind()* nel nostro caso ritornerà sempre valore nullo, perché in questa applicazione non gestiremo servizi di tipo *Bounded* nei quali un'altra *Activity* può interfacciarsi al servizio.

Possiamo ora aggiungere alla nostra classe, anche un nostro metodo privato *getStatusService()* attraverso il quale controlleremo lo stato del servizio (attivo o arrestato) all'avvio dell'applicazione: si tratterà semplicemente di settare una nostra variabile interna a *true* nella fase di *Start* e a *false* nella fase di *Destroy*.

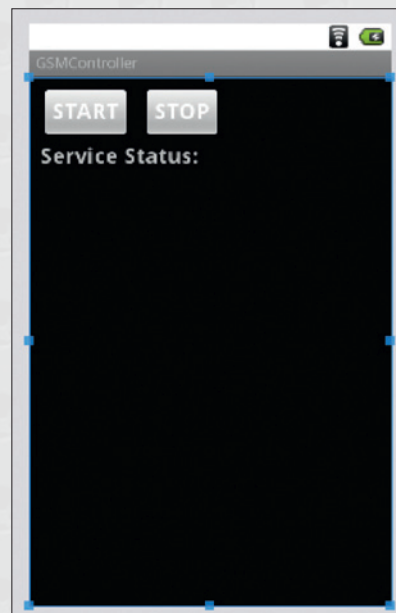
È bene, infine, ricordare che un servizio, in quanto tale, può essere attivo anche se l'*Activity* principale della nostra applicazione è stata chiusa; esso continuerà a funzionare e lo potremo arrestare riaprendo la nostra applicazione e

premendo il pulsante *Stop*. Per verificare lo stato del servizio, useremo la casella di testo *textServiceStatusInfo*.

Un primo layout (*main.xml*) dell'applicazione è mostrato in **Fig. 4**.

Prima di continuare può essere utile una breve precisazione sulla gestione della RAM in un sistema Android: quando chiudiamo una qualsiasi applicazione Android, questa in realtà continuerà a rimanere in memoria fino a quando una successiva applicazione richiederà una quantità di memoria superiore

Fig. 4



Listato 1

```
// GSMService.java

@Override
public void onCreate()
{
    Toast.makeText(this, "Service Created", Toast.LENGTH_LONG).show();
    Log.d(TAG, "onCreate");

    // sensor
    _sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    Sensor s = _sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

    _sensorManager.registerListener(_accelerometerSensorListener, s, SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
public void onStart(Intent intent, int startid)
{
    Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    Log.d(TAG, "onStart");

    _statusService = true;
}

@Override
public void onDestroy()
{
    Toast.makeText(this, "Service Stopped", Toast.LENGTH_LONG).show();
    Log.d(TAG, "onDestroy");

    _sensorManager.unregisterListener(_accelerometerSensorListener);
    _sensorManager = null;

    _statusService = false;
}
```

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.kiand.gsm"
    android:versionCode="1"
    android:versionName="2.0" >

    <uses-sdk android:minSdkVersion="8" />

    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.ACCESS_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_GPS"/>
    <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".GSMControllerActivity"
            android:screenOrientation="portrait" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service android:enabled="true" android:name=".GSMService" />

    </application>
</manifest>

```

a quella rimasta libera, oppure finché non sarà trascorso un certo periodo di tempo dall'ultimo utilizzo.

Un servizio in background ha una priorità minore di un'Activity in primo piano, ma sempre maggiore di un'Activity che rimane in memoria dopo essere stata chiusa; in questo modo siamo sicuri che, salvo condizioni di sistema estremamente particolari, difficilmente Android interromperà il nostro servizio.

Così come fatto in precedenza, quando abbiamo aggiunto altre Activity al progetto, dobbiamo informare il sistema dell'utilizzo della classe *GSMService*, aggiungendo la seguente istruzione al file *AndroidManifest.xml*:

```
<service android:enabled="true" android:name=".GSMService" />
```

Questa riga è fondamentale per la corretta esecuzione del servizio, altrimenti anche se la compilazione del progetto avrà ugualmente luogo, otterremo un errore a run-time.

Il file *AndroidManifest.xml* completo è visibile nella Fig. 5.

A questo punto, dopo aver lanciato il nostro servizio dall'Activity principale, possiamo anche chiudere l'applicazione (tramite il pulsante fisico *back*) ed usare il nostro cellulare normalmente, sapendo che potremo arrestarlo rilanciando l'applicazione e premendo il pulsante Stop. Quando accederemo nuovamente all'applicazione, la casella di testo ci informerà sullo stato del servizio.

Adesso abbiamo un servizio, che tuttavia, per il momento, non fa ancora alcunché; per verificare la sua presenza nel sistema basta utilizzare una di quelle "app manager" scaricabili gratuitamente

Fig. 5

dal market Android (Google Play) o già presenti nel dispositivo, che sono in grado di monitorare tutti i servizi attivi sul tablet o smartphone.

GESTIONE DEI SENSORI ANDROID

Andiamo ora a "riempire" il servizio creato, facendo in modo che reagisca ad eventi di sistema come l'attivazione di sensori presenti nel nostro dispositivo. Per capire quanti e quali sensori sono presenti nel nostro smartphone, inseriamo le seguenti linee, per esempio, nel metodo *onCreate()* dell'Activity principale:

```
SensorManager manager = (SensorManager)
```

```
getSystemService(SENSOR_SERVICE);
```

```
List<Sensor> list = manager.getSensorList(Sensor.
TYPE_ALL);
```

La classe *SensorManager* rappresenta il gestore "globale" di tutti i sensori presenti. Dall'oggetto *manager* siamo in grado di ottenere un oggetto appartenente alla classe *Sensor* che rappresenta il nostro sensore vero e proprio. Il metodo *getSensorList()* dell'oggetto *manager* ci restituisce il sensore desiderato.

Avendo usato l'identificativo *TYPE_ALL* per indicare tutti i dispositivi, il *manager* ci restituirà la lista completa dei sensori presenti; se invece avessimo voluto cercare un solo sensore, avremmo potuto usare direttamente il tipo appropriato, come ad esempio *TYPE_PRESSURE* nel caso del sensore di pressione.

Nel nostro caso useremo il metodo *getDefaultSensor()* che restituisce direttamente il singolo oggetto *Sensor* corrispondente. Nel riquadro "Sensori Android" che trovate in queste pagine potete vedere con quali tipi poter accedere a tutti i sensori disponibili.

Di questi, nella nostra applicazione useremo l'accelerometro e quindi accederemo al nostro oggetto *Sensor* tramite il tipo *TYPE_ACCELEROMETER*. Dall'oggetto *Sensor* siamo anche in grado di ottenere alcune interessanti informazioni come la risoluzione, la potenza usata dal sensore, il raggio d'azione ed altre funzionalità. Volendo ottenere tutti i dati dei sensori in modalità asincrona, senza perciò interrogarli a polling, occorrerà registrare un oggetto di classe *Listener* (*_accelerometerSensorListener*) attraverso il metodo *registerListener()* del nostro gestore *SensorManager*, come si vede nel **Listato 1**.

In questo modo stiamo indicando al sistema Android di intercettare tutti i dati disponibili del sensore e di notificarli alla nostra classe *Listener*

Listato 2

```

private SensorEventListener _accelerometerSensorListener = new SensorEventListener()
{
    private int _count = 0;
    private long firstShake = 0;

    private int SHAKE_DURATION = 600;
    private int SHAKE_COUNT = 4;
    private int SHAKE_THRESHOLD = 2000;
    private int SHAKE_INTERVAL = 500;

    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
        //...
    }
    public void onSensorChanged(SensorEvent event)
    {
        float[] values = event.values;
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
        {
            long curTime = System.currentTimeMillis();

            // update every 100ms.
            if ((curTime - lastUpdate) > 100)
            {
                long diffTime = (curTime - lastUpdate);
                lastUpdate = curTime;

                x = values[SensorManager.DATA_X];
                y = values[SensorManager.DATA_Y];
                z = values[SensorManager.DATA_Z];

                NumberFormat numberFormat = new DecimalFormat("0.00");
                String strX = numberFormat.format(x);
                String strY = numberFormat.format(y);
                String strZ = numberFormat.format(z);

                float speed = Math.abs(x+y+z - last_x - last_y - last_z) / diffTime
                    * 10000;

                if (speed > (SHAKE_THRESHOLD) )
                {
                    Log.d(TAG, "_count:" + _count );
                    if (_count == 0)
                    {
                        firstShake = curTime;
                    }
                    _count++;

                    long durationStep = (curTime - firstShake);
                    if(durationStep > SHAKE_INTERVAL)
                    {
                        _count = 0;
                        Log.d(TAG, "durationStep:" + durationStep);
                    }

                    if (_count >= SHAKE_COUNT)
                    {
                        _count = 0;
                        long duration = (curTime - firstShake);
                        if( duration <= SHAKE_DURATION )
                        {
                            Log.d(TAG, "SHAKE duration:" + duration + "count:"
                                + _count );

                            _vibrate.vibrate(500);
                            callNumber("3346710868");
                        }
                    }
                }
                last_x = x;
                last_y = y;
                last_z = z;
            }
        }
    }
};

```


SENSORI ANDROID

Android, attraverso la classe *SensorManager*, ci permette di accedere ai sensori presenti nel nostro smartphone o tablet. Non tutti i dispositivi Android hanno, però, stesse caratteristiche e i medesimi sensori, perciò può essere utile ottenere una lista dei sensori presenti tramite le seguenti righe di codice:

```
SensorManager manager = (SensorManager)
getSystemService(SENSOR_SERVICE);
List<Sensor> list = manager.getSensorList(Sensor.
TYPE_ALL);
```

Se invece sapessimo già con che tipo di sensore avremo a che fare, possiamo direttamente usare il metodo *getDefaultSensor()* passandogli il tipo corrispondente come nella seguente riga:

```
Sensor s = _sensorManager.getDefaultSensor(Sensor.
TYPE_ACCELEROMETER);
```

Tipo	Sensore
TYPE_GYROSCOPE	Giroscopio
TYPE_LIGHT	seniore di luce
TYPE_ACCELEROMETER	Accelerometro
TYPE_MAGNETIC_FIELD	seniore di campo magnetico
TYPE_PRESSURE	seniore di pressione
TYPE_ORIENTATION	seniore di orientazione
TYPE_PROXIMITY	seniore di prossimità
TYPE_TEMPERATURE	seniore di temperatura

Metodo sensore	Descrizione
getMaximumRange()	il raggio di azione del sensore
getName()	il nome del sensore
getPower()	la potenza consumata dal sensore in mA
getResolution()	la risoluzione del sensore
getVendor()	il produttore del modulo
getVersion()	la versione del modulo

Nelle tabelle in questo riquadro vedete la lista completa dei sensori ed alcuni metodi dell'oggetto *Sensor* tramite i quali siamo in grado di conoscere alcune interessanti informazioni.

che analizzeremo a breve. Da notare anche il terzo parametro passato al metodo *registerListener()* che rappresenta la frequenza di aggiornamento del sensore. Per la nostra applicazione abbiamo scelto un valore normale (*SENSOR_DELAY_NORMAL*), ma in alcuni casi potrebbe essere necessario un campionamento più alto (*SENSOR_DELAY_FAST*) o anche molto elevato (*SENSOR_DELAY_GAME*).

Come mostrato nel **Listato 1**, al momento della distruzione del servizio (metodo *onDestroy()*) occorre deregistrare la classe Listener, per evitare di lasciare la CPU in attività anche dopo la chiusura dell'applicazione.

Occupiamoci, adesso, della scrittura della classe *SensorEventListener* presente nel **Listato 2**, nella quale abbiamo accesso ai dati veri e proprio letti dal nostro sensore; in particolare, implementeremo il metodo *onAccuracyChanged()* che fa parte dell'interfaccia, ma che lasceremo vuoto, ed il metodo *onSensorChanged()* che verrà chiamato dal sistema ad ogni cambiamento dei dati del sensore e nel quale prenderemo le nostre decisioni.

In questo metodo, grazie alla precedente registrazione abbiamo anche l'informazione (sempre passata dal sistema) dell'evento scatenato dal sensore (*event*) dal quale possiamo ottenere i dati veri e propri (*event.values*) che nel nostro caso sono rappresentati da un array di float contenente il valore di accelerazione sui tre assi, ed anche l'indicazione del sensore (*event.sensor*) che ha scaturito l'evento.

L'obiettivo è quello di generare una chiamata verso un numero di telefono ogni qual volta il dispositivo venga scosso ripetutamente, quindi dobbiamo verificare che ci siano forti variazioni di accelerazione su tutti gli assi, in un tempo più o meno breve. Questo viene realizzato semplicemente memorizzando gli ultimi valori e sottraendoli a quelli correnti, per poi contare il numero di "scosse" fino a raggiungere un numero limite (*SHAKE_COUNT*) oltre il quale verificheremo che il tempo trascorso dal primo scuotimento non superi un certo valore massimo (*SHAKE_DURATION*) per decidere se effettivamente il dispositivo è stato scosso sufficientemente. Per gestire la variabile "tempo" abbiamo usato la funzione di sistema *System.currentTimeMillis()* che restituisce i millisecondi trascorsi dal gennaio 1970.

Se tutte queste soglie vengono superate con successo, attiveremo la chiamata vera e propria al numero di telefono desiderato tramite la funzione *callNumber()* che vedremo adesso.

FUNZIONI GSM

Con Android possiamo gestire chiamate e messaggi GSM anche all'interno della singola applicazione; noi utilizzeremo questa possibilità per inviare comandi ad un *GSM Shield* montato su Arduino, che sulla base dei messaggi ricevuti sarà in grado di attivare apparati esterni o risponderci a sua volta con invio di SMS contenenti particolari informazioni.

Per gestire una chiamata in uscita abbiamo cre-

Listato 3

```
private void callNumber(String numberToCall)
{
    try
    {
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:" + numberToCall));
        startActivity(callIntent);

        Log.v(TAG, "GSM Call Number:" + numberToCall);
    }
    catch (ActivityNotFoundException e)
    {
        Log.e("GSM Call Example", "Call failed", e);
    }
}

private void sendMessage(String phoneNumber, String message)
{
    SmsManager smsManager = SmsManager.getDefault();
    smsManager.sendTextMessage(phoneNumber, null, message, null, null);
}
```

Listato 4

```
private PhoneStateListener _phoneListener = new PhoneStateListener()
{
    public void onCallStateChanged(int state, String incomingNumber)
    {
        try
        {
            switch (state)
            {
                case TelephonyManager.CALL_STATE_RINGING:
                    Toast.makeText(GSMControllerActivity.this, "CALL_STATE_RINGING FROM " + incomingNumber, Toast.LENGTH_SHORT).show();
                    break;

                case TelephonyManager.CALL_STATE_OFFHOOK:
                    Toast.makeText(GSMControllerActivity.this, "CALL_STATE_OFFHOOK", Toast.LENGTH_SHORT).show();
                    break;

                case TelephonyManager.CALL_STATE_IDLE:
                    Toast.makeText(GSMControllerActivity.this, "CALL_STATE_IDLE", Toast.LENGTH_SHORT).show();
                    break;

                default:
                    Toast.makeText(GSMControllerActivity.this, "default", Toast.LENGTH_SHORT).show();
            }

            Log.i("Default", "Unknown phone state=" + state);
        }
        catch (Exception e)
        {
            Log.i("Exception", "PhoneStateListener() e = " + e);
        }
    }
};
```

ato la funzione *callNumber()* nella quale, come si vede dal **Listato 3**, utilizza un Intent di sistema (*ACTION_CALL*) che, dopo essere stato configurato col numero di telefono desiderato, viene passato alla funzione *startActivity()* in seguito alla quale ci troveremo l'interfaccia grafica che siamo abituati a vedere dopo aver composto un numero sul nostro telefono Android.

Al termine della chiamata (il dispositivo remoto sarà impostato, tramite Arduino, a concludere la chiamata e ad effettuare l'azione desiderata) verrà nuovamente visualizzata l'interfaccia della nostra Activity.

Come al solito, non dobbiamo dimenticarci di inserire nel file *AndroidManifest.xml*, tra i vari permessi, la seguente riga:

Listato 5

```

#include <GSM_Shield.h>
GSM gsm;

void setup()
{
    gsm.TurnOn(9600);
    gsm.InitParam(PARAM_SET_1);
    gsm.Echo(1);

    pinMode(13, OUTPUT);
    pinMode(11, OUTPUT);
}

void loop()
{
    int call;
    call=gsm.CallStatus();
    switch (call)
    {
        case CALL_NONE:
            Serial.println("no call");
            break;
        case CALL_INCOM_VOICE:
            delay(5000);
            gsm.PickUp();
            break;
        case CALL_ACTIVE_VOICE:
            delay(5000);
            gsm.HangUp();

            // do something
            digitalWrite(13, HIGH);

            break;
    }

    String sms = getSMS();

    if (sms == "cmd1")
    {
        digitalWrite(13, LOW);
    }

    if (sms == "cmd2")
    {
        digitalWrite(11, LOW);
    }

    delay(1000);
}

String getSMS()
{
    char indexSms;
    indexSms =gsm.IsSMSPresent(type_sms);

    char numberCalling[15];
    char smsBody[122];

    String smsStr = "";

    if (indexSms!=0)
    {
        gsm.GetSMS(indexSms,numberCalling,
        smsBody,120);
        smsStr = smsBody;
    }

    return smsStr;
}

```

`<uses-permission android:name="android.permission.CALL_PHONE"/>`

Per quanto riguarda l'invio di un messaggio, nulla ci vieta di usare anche in questo caso l'Activity di sistema con cui siamo soliti inviare messaggi, ma c'è anche l'opportunità di ottenere lo stesso risultato usando la classe *SmsManager*, senza visualizzare ulteriori interfacce. Questo è possibile, sempre nel **Listato 3**, attraverso la funzione *sendMessage()* in cui la stringa *message* rappresenta il corpo del messaggio e *phoneNumber()* il numero di destinazione.

Nella nostra applicazione assoceremo l'invio di particolari messaggi preconfigurabili a tre pulsanti che inseriremo con la procedura ormai standard nell'Activity principale. Anche in questo caso non dovremo dimenticarci di inserire il seguente permesso:

`<uses-permission android:name="android.permission.SEND_SMS"/>`

nel file *AndroidManifest.xml* e di aggiungere agli import precedenti il seguente:

`import android.telephony.SmsManager;`

Anche per la gestione delle chiamate in entrata, l'approccio è asincrono, ma ormai sappiamo usare le classi Listener viste prima per il caso dell'accelerometro e quindi non dovrebbero esserci problemi. Useremo la classe *TelephonyManager* e pertanto inseriremo il relativo import: `import android.telephony.TelephonyManager;` e il permesso nel *ManifestAndroid.xml*:

`<uses-permission android:name="android.permission.READ_PHONE_STATE"/>`

A questo punto nella *onCreate()* del nostro servizio, attraverso la classe *TelephonyManager*, registreremo la classe *PhoneStateListener* con le seguenti righe di codice:

`TelephonyManager _telephonyManager=TelephonyManager.getSystemService(TELEPHONY_SERVICE);`

`_telephonyManager.listen(_phoneListener, PhoneStateListener.LISTEN_CALL_STATE);`

Ovviamente dobbiamo prima implementare la classe *_phoneListener* come nel **Listato 4**, in cui andremo a riempire il metodo *onCallStateChanged()* che viene chiamato dal sistema operativo ogni volta che viene cambiato lo stato della chiamata.

Sapendo in automatico anche il numero chiamante (*incomingNumber*) potremmo controllarlo ed eseguire le opportune funzioni. Per ora, siccome quella proposta è un'applicazione di-

Listato 6

```

private LocationListener _locationListener = new LocationListener()
{
    public void onLocationChanged(Location location)
    {
        _vibrate.vibrate(300);

        if (location == null)
        {
            return;
        }

        _lastLocation = location;
        _buttonStoreLocation.setEnabled(true);
        _buttonStoreLocation.setBackgroundColor(Color.GREEN);

        float speed = location.getSpeed();
        double altitude = location.getAltitude();
        double longitude = location.getLongitude();
        double latitude = location.getLatitude();

        float distanceMeters = (float)0;
        if (_storedLocation != null)
        {
            distanceMeters = location.distanceTo(_storedLocation);
            if (distanceMeters < _radiusGPS)
            {
                // do somethings
            }
        }

        String dbgInfoString = "LAT:" + latitude + "\n" + "LON:" + longitude + "\n" + "ALT:" +
            altitude + "\n" + "SPD:" + speed + "\n" + "DST:" + distanceMeters;
        Log.v(TAG, dbgInfoString);
        Toast.makeText(GSMControllerActivity.this, dbgInfoString, Toast.LENGTH_LONG).show();

        NumberFormat numberFormat = new DecimalFormat("0.000");
        _textCurrentLocation.setText("LAT:" + numberFormat.format(latitude) + "    LON:" +
            numberFormat.format(longitude));
    }

    public void onStatusChanged(String provider, int status, Bundle extras)
    {
        Toast.makeText(GSMControllerActivity.this, status, Toast.LENGTH_SHORT).show();
    }
};

```

dattica, ci siamo limitati a visualizzare messaggi Toast.

Notate che i messaggi Toast sono “permessi” dal servizio, seppure rappresentino una sorta di interfaccia grafica con l’utente. Questo vuol dire che se anche la nostra Activity non è in primo piano, ma il nostro servizio è attivo, quando riceveremo una chiamata, indipendentemente dallo stato del nostro dispositivo riceveremo sullo schermo la notifica grafica tramite il messaggio Toast.

Adesso diamo una rapida occhiata al lato Arduino che riceverà le chiamate ed attiverà le uscite verso altri dispositivi ad esso collegati.

Nel **Listato 5** possiamo vedere un esempio di come usare la libreria *GSM_Shield* su Arduino, per gestire il controllo delle chiamate e degli SMS in ingresso. Periodicamente controlleremo le chiamate in arrivo e verificheremo la presenza

di SMS in arrivo per abilitare o meno due piedini di uscita a seconda del messaggio ricevuto.

UTILIZZO DEL GPS

La nostra applicazione ha ora una sua praticità, ma possiamo ancora dotarla di ulteriori caratteristiche, come far effettuare una chiamata o inviare un SMS verso lo shield GSM ogni qualvolta lo smartphone o tablet si trovi nell’area intorno ad un punto geografico.

Useremo, allo scopo, il GPS, che non è un sensore della lista accennata in precedenza ma costituisce un modulo a parte; il meccanismo con il quale possiamo venire in possesso dei dati (latitudine e longitudine) corrispondenti alla posizione in cui si trova il dispositivo, è tuttavia il medesimo: esiste un *manager* (*LocationManager*) che si occupa esclusivamente di gestire la posizione e che viene utilizzato per registrare una classe Listener

Listato 7

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
  <PreferenceCategory
    android:title="Gsm numbers information"
    android:summary="Destination phone number" >
    <EditTextPreference
      android:key="numberToCallAfterShaking"
      android:title="Phone Number to Call after shaking"
      android:summary="Called after shaking your device" />
    <EditTextPreference
      android:key="numberToCallInLocation"
      android:title="Phone Number to Call in location X"
      android:summary="Called when your device is in stored location" />
  </PreferenceCategory>

  <PreferenceCategory
    android:title="Accelerometer Settings"
    android:summary="Set the shaking sensibility" >

    <CheckBoxPreference
      android:key="chbShaking"
      android:title="Call Shaking Enable"
      android:summary="On/Off"
      android:defaultValue="true" />

    <EditTextPreference
      android:key="shakeDuration"
      android:title="Shake Duration [ms]"
      android:summary="start call after this time"
      android:defaultValue="600"
      android:dependency="chbShaking" />

    <EditTextPreference
      android:key="shakeCount"
      android:title="Shake Count"
      android:summary="start call after count shake"
      android:defaultValue="4"
      android:dependency="chbShaking" />
  </PreferenceCategory>

  <PreferenceCategory
    android:title="Location Settings"
    android:summary="Set the calling in location" >

    <CheckBoxPreference
      android:key="chbLocation"
      android:title="Call Location Enable"
      android:summary="On/Off"
      android:defaultValue="true" />

    <ListPreference
      android:key="listProviderLocation"
      android:title="Location Provider"
      android:summary="Choice GPS or Network"
      android:entries="@array/listOptions"
      android:entryValues="@array/listValues" />
  </PreferenceCategory>
</PreferenceScreen>
```

(*LocationListener*) in diverse modalità. Questo è presentato nelle seguenti righe, che vanno scritte nel metodo *onStart()* del nostro servizio:

```
LocationManager locationManager = (LocationManager) this.getSystemService(LOCATION_SERVICE);
```

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, _locationListener);
```

È possibile determinare la posizione anche tramite Internet (*NETWORK_PROVIDER*) anche se con una precisione minore. Ovviamente è necessario che il nostro smartphone sia impostato per utilizzare rete Wi-Fi o GPS, cosa che potremmo sempre verificare da codice ed eventualmente utilizzare un Intent di sistema per visualizzare direttamente la configurazione Android con cui settare i metodi da utilizzare, attraverso queste due semplice linee di codice:

```
Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
startActivity(intent);
```

Il risultato dell'operazione è visibile in Fig. 6.

La corrispondente classe Listener è rappresentata nel Listato 6; ad ogni cambiamento di posizione viene chiamata la *onLocationChanged()* la quale ci fornisce l'oggetto *Location* contenente la posizione espressa con longitudine, latitudine e, se il numero di satelliti "agganciati" lo permette, anche altitudine. Oltre a questo, tramite *Location* siamo anche in grado di calcolare la distanza in metri da un punto desiderato (grazie al metodo *distanceTo()*) che avremo settato in precedenza premendo il pulsante *_storedLocation*.

A questo punto, per far scattare la stessa chiamata di prima o l'invio di particolari messaggi basterà controllare quando questa distanza diventi minore di un certo raggio, dato che ora abbiamo tutto il necessario per farlo.

Gli import per gestire il GPS sono i seguenti:

```
import android.location.LocationManager;
import android.
location.LocationLi-
stener;
import android.
location.Location;
```

I permessi nel file *ManifestAndroid.xml* sono come in Fig. 5. In particolare, *ACCESS_FINE_LOCATION* è utilizzato se usiamo il GPS, mentre *ACCESS_COARSE_LOCATION* serve se usiamo la rete. Possiamo comunque metterli



Fig. 6

PASSAGGIO PARAMETRI TRA ACTIVITY E SERVIZI

Dovendo spesso gestire applicazioni con più di una Activity e servizi, occorre trasferire alcuni parametri tra le varie classi, non solo tra un'Activity ed uno o più servizi, ma anche tra più Activity. Esistono diversi modi più o meno eleganti; ne considereremo due oltre a quello più complesso di creare una finestra di opzioni (Preferences) che rimangono condivise in tutta l'applicazione come abbiamo analizzato in queste pagine.

Il primo è quello di fornire all'Intent informazioni aggiuntive e personalizzate, dette appunto "extra" che verranno lette dal servizio o dall'Activity di destinazione, dopo aver chiamato la `startActivity()` o la `startService()` dalla classe principale. Per scrivere e leggere queste informazioni "extra" utilizzeremo due metodi dell'Intent, rispettivamente `putExtra()` e `getExtra()`.

L'esempio in Fig. A chiarisce ciò.

ACTIVITY

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    _serviceIntent = new Intent(getApplicationContext(), GSSMService.class);

    _buttonStart = (Button) findViewById(R.id.buttonStart);
    _buttonStart.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            _serviceIntent.putExtra("Latitude", _latitude);
            _serviceIntent.putExtra("Longitude", _longitude);
            startService(_serviceIntent);

            _textViewServiceStatusInfo.setText("Service Status: Enabled");
        }
    });
}
```

SERVICE

```
@Override
public void onStart(Intent intent, int startId) {
    Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    Log.d(TAG, "onStart");

    _statusService = true;

    Bundle bundle = intent.getExtras();
    double lat = bundle.getDouble("Latitude");
    double lon = bundle.getDouble("Longitude");

    Log.d(TAG, "Received values: " + lat + " " + lon);
}
```

Le stringhe "Latitude" e "Longitude" rappresentano le chiavi, mentre i valori veri e propri (in questo

Fig. A

```
@Override
public void onDestroy() {
    //...
    // save the preferences
    SharedPreferences prefs = getSharedPreferences(MYPREFS, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putString("PHONE_NUMBER", newPhoneNumber);
    editor.putInt("RADIUS", newRadius);
    editor.commit();
}
```

```
@Override
public void onCreate() {
    // load the preferences
    SharedPreferences prefs = getSharedPreferences(MYPREFS, Context.MODE_PRIVATE);
    String phoneNumber = prefs.getString("PHONE_NUMBER", "No Phone");
    int radius = prefs.getInt("RADIUS", 0);
    //...
}
```

Fig. B

caso di tipo double...) sono `_latitude` e `_longitude`.

Nell'esempio, il destinatario è il servizio che ha già l'Intent come parametro, ma se fosse stata un'altra Activity occorre ricavare l'Intent associato tramite la funzione `getIntent()`. Inoltre l'oggetto *Bundle*, ha naturalmente, oltre a `getDouble()`, tutti gli altri metodi per ritornare i tipi corrispondenti. Il secondo metodo si basa sulla classe *SharedPreferences* ed usa lo stesso meccanismo delle Preferences, senza però dover creare un'interfaccia grafica di settaggi. Potremmo un po' paragonarlo al *registry* di Windows, perché si basa sull'associazione *chiave-valore* e perché rimane persistente nel sistema anche dopo la chiusura dell'applicazione. Queste informazioni vengono salvate in un file all'interno del percorso *data/data/* e possono essere condivise anche da altre applicazioni. Nell'esempio di Fig. B sono state usate all'interno della stessa Activity per memorizzare informazioni e recuperarle al successivo riavvio, ma nulla vieta di usarle anche tra Activity o servizi diversi, in qualsiasi punto del codice. La stringa *MYPREFS* identifica l'associazione tra le due sezioni e può contenere qualsiasi testo, basta che sia univoca. La costante *MODE_PRIVATE* indica, invece, che le informazioni saranno valide solo per la nostra applicazione. Infine al metodo `getString()` deve essere passato come parametro, oltre alla chiave, anche il valore predefinito nel caso in cui non venga trovata nessuna chiave corrispondente.

entrambi, qualora prevedessimo di poter gestirli nella stessa applicazione.

PREFERENCES ANDROID

L'applicazione sarebbe conclusa, ma per renderla più versatile inseriremo una serie di opzioni con le quali gestire alcuni parametri come il raggio d'azione della localizzazione, l'entità dello scuotimento superata la quale viene effettuata la chiamata, i numeri destinatari, il corpo dei messaggi che verranno inviati sulla pressione dei pulsanti e tante altre. Android ci viene incontro fornendoci gli strumenti per creare, con sem-

plicità, una schermata di opzioni raggruppabili con le relative intestazioni. Questa schermata, o meglio Activity, è detta **Preferences** ed è la stessa usata da Android quando accediamo alle impostazioni di sistema (o *Settings*). Viene usata generalmente per gestire grosse quantità di opzioni ed offre i vantaggi di condividere le informazioni scelte dall'utente anche a tutte le altre Activity del progetto. In questo modo le modifiche che faremo nelle Preferences (avviate dall'Activity principale) saranno accessibili anche all'interno del nostro servizio, dove avranno il loro effetto. Inoltre le modifiche che effettueremo

Fig. 7

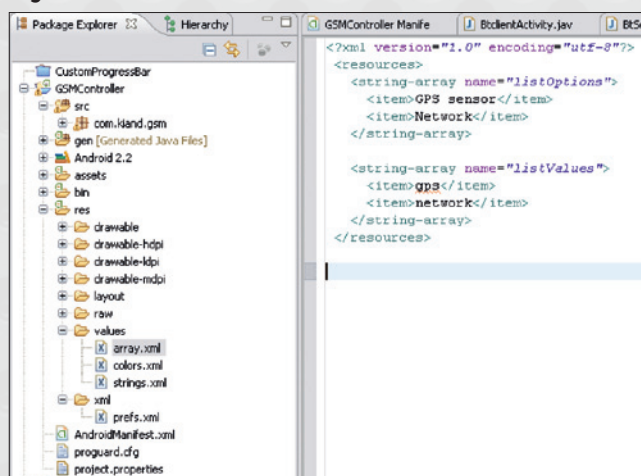


Fig. 8

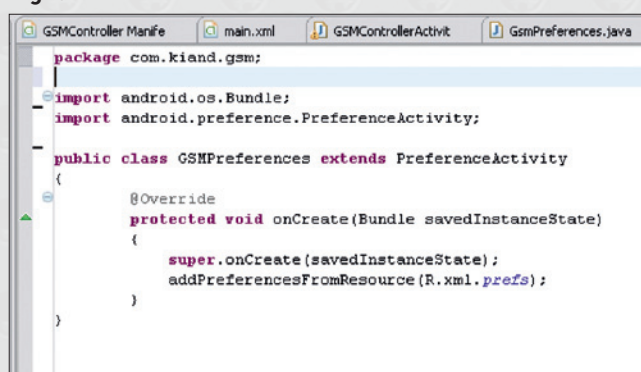


Fig. 9



mo all'interno delle Preferences verranno salvate automaticamente nel sistema e le ritroveremo uguali anche al successivo riavvio dell'applicazione senza bisogno di alcun intervento da parte nostra.

Vediamo ora come creare una nostra finestra di Preferences a partire da quello che rappresenterà la nostra interfaccia grafica: il file xml (nel nostro caso *prefs.xml*) che andremo a creare all'interno della cartella *xml* interna alla cartella di sistema *res*. Se questa non fosse già presente, potremo crearla da browser ed eseguire un "refresh" del progetto. Un frammento (non

integrale) di questo file è visibile nel Listato 7. Come si può notare, la filosofia è la stessa dei file di layout, ma questa volta non dobbiamo preoccuparci delle dimensioni e della posizione degli oggetti grafici, i quali potranno essere principalmente di tre tipi: **CheckBoxPreference** (usata per valori booleani), **EditTextPreference** (usata per le stringhe) e **ListPreference** (usata per scegliere valori all'interno di una lista). Ognuno di questi oggetti viene rappresentato dall'intestazione principale (*android:title*), da una descrizione che comparirà sotto il titolo (*android:summary*) e da una chiave (*android:key*) che deve essere univoca per ogni controllo, oltre ad altri campi opzionali come ad esempio *android:defaultValue* per impostare valore di default ed *android:dependency* che verrà impostato con la chiave di un controllo **CheckBoxPreference**, ad indicare che l'oggetto grafico sarà abilitato o meno a seconda del valore del checkbox indicato.

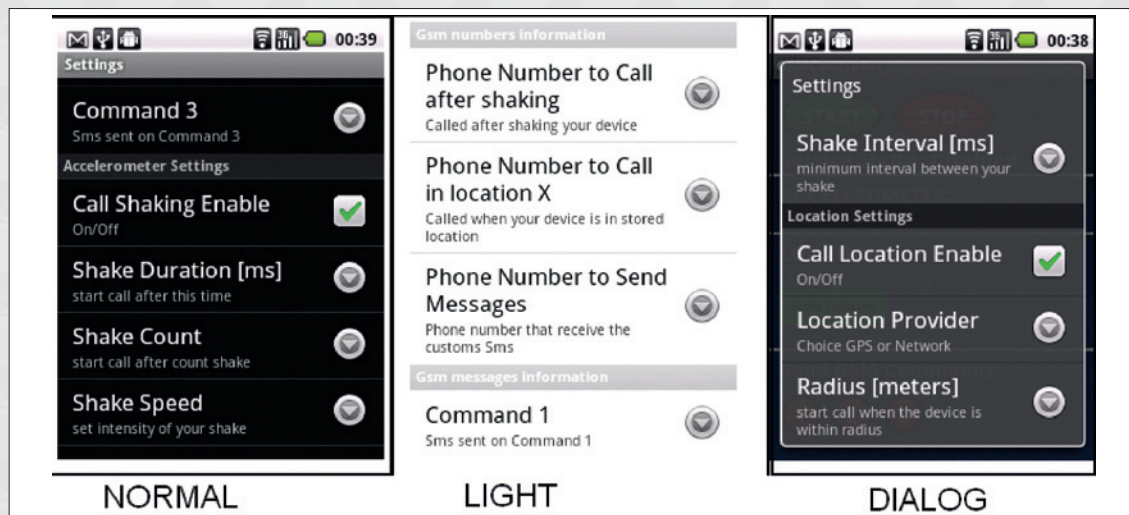
Ognuno di questi oggetti potrà anche essere raggruppato all'interno di una categoria mediante il tag xml **PreferenceCategory**, anch'esso dotato del campo *title* e *summary*.

Per quanto riguarda l'oggetto **ListPreference**, esso merita un discorso a parte, perché avrà un campo per contenere gli elementi della lista (*android:entries*) ed un altro (*android:entryvalues*) che contiene i valori restituiti in seguito alla scelta dell'elemento corrispondente. Solitamente si sceglie di elencare questi valori in un file xml a parte (nel nostro caso *array.xml*) che conterrà questi array e che andremo ad aggiungere nella cartella *values* all'interno della cartella *res*, come in Fig. 7. Prima di scendere nei dettagli possiamo vedere in Fig. 10 un'anticipazione di come si presenterà.

Ora è il momento di associare l'interfaccia xml ad una nuova Activity, quindi andremo a creare un nuovo file ed una nuova classe che chiameremo **GSMPreferences** e che sarà una classe estesa della classe **PreferenceActivity**. Nel metodo *onCreate()* di Fig. 8 useremo la funzione *addPreferencesFromResources()* cui passeremo il nostro file xml precedentemente creato (*prefs.xml*); da notare l'import **PreferenceActivity** necessario per la compilazione della classe.

Questa classe, come vedete, è molto corta perché contiene solamente l'associazione con il file xml, ma sempre nella *onCreate()* potremmo aggiungere il codice per accedere agli oggetti grafici

Fig. 10



delle Preferences ed usarlo per controllare, ad esempio, la consistenza dei valori inseriti. Anziché usare il classico `findViewById()` al quale eravamo abituati con le Activity, basterà usare il metodo `findPreference()` cui passeremo la chiave (`android:key`) dell'oggetto. Ottenuto il controllo potremmo usare allo stesso modo, una classe Listener (`setOnPreferenceChangeListener`) che implementeremo a piacere e che verrà chiamata dal sistema non appena quel controllo verrà impostato ad un certo valore.

Adesso dobbiamo solo creare un pulsante o una voce di menu (già visto nella puntata precedente) da cui lanciare le nostre Preferences e per farlo useremo il solito modo utilizzato per le Activity normali, creando una nuova Intent col nome della classe (`GSMPreferences.class`) ed usando la `StartActivity()` come nelle righe seguenti:

```
Intent iPrefs = new Intent(this, GSMPreferences.class);
```

```
startActivity(iPrefs);
```

Essendo le Preferences un'Activity in tutto e per tutto, non dobbiamo dimenticarci di aggiungerle al solito file `AndroidManifest.xml` come in figura Fig. 9.

Potremo infine personalizzare anche l'aspetto grafico delle Preferences a piacere, utilizzando il campo `android:theme` all'interno dell'Activity, come ad esempio:

```
android:theme="@android:style/Theme.Dialog"
```

e:

```
android:theme="@android:style/Theme.Light.NoTitleBar.Fullscreen"
```

Le due hanno diversi effetti, come si può vedere nella Fig. 10.

In ultimo vediamo come sia semplice accedere ai valori modificati nelle Preferences, da qualsiasi Activity o servizio. Basterà, infatti, un oggetto della classe **SharedPreferences** ed usare i suoi metodi opportuni per ricavare i numeri, le stringhe o i booleani dai controlli associati semplicemente utilizzando le chiavi che abbiamo inserito nel campo `android:key` del file `prefs.xml`. Un esempio è mostrato nel **Listato 8**, di cui la funzione `showUserSettings()` può essere chiamata in ogni file del progetto.

PULSANTI PERSONALIZZATI

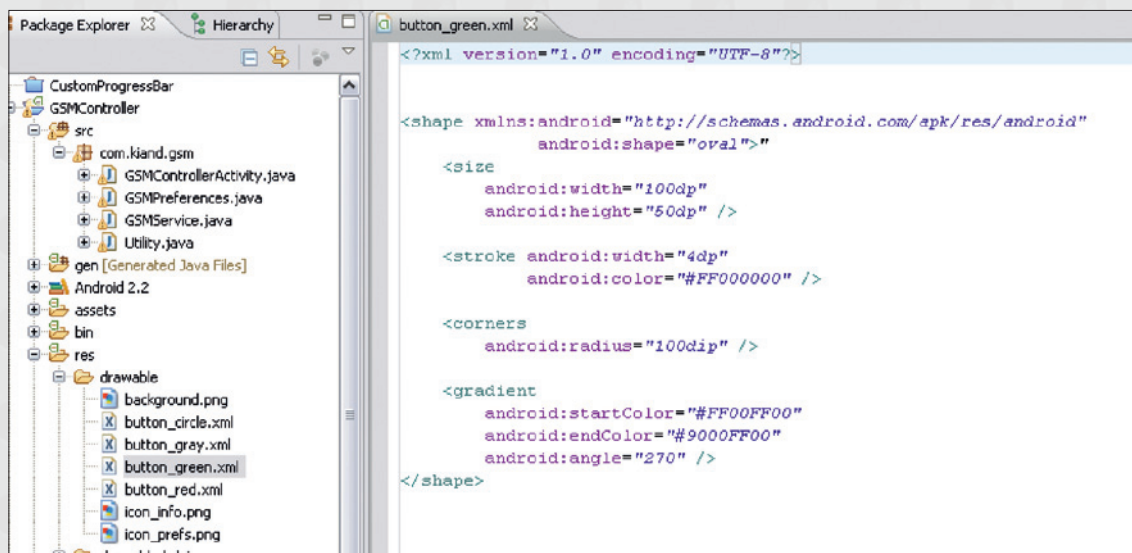
Infine, per personalizzare ulteriormente la nostra applicazione possiamo dedicarci allo stile dei pulsanti che attivano e arrestano il

Listato 8

```
private void showUserSettings()
{
    SharedPreferences sharedPrefs = PreferenceManager.getDefaultSharedPreferences(this);

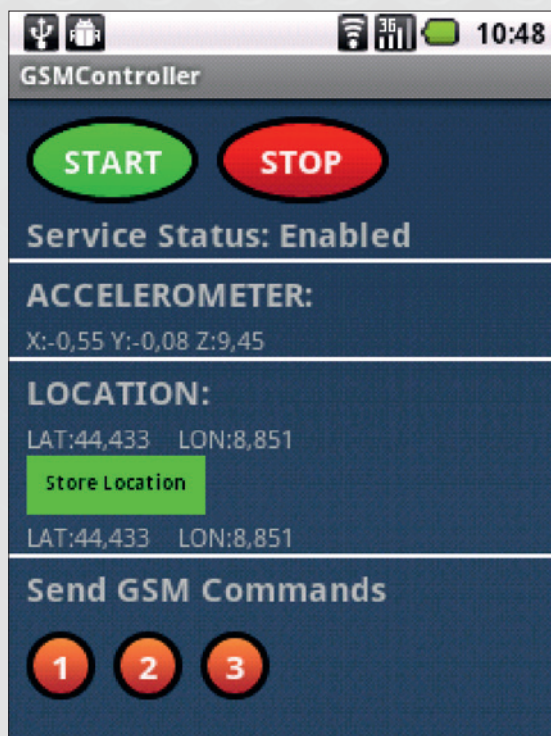
    String numberAfterShaking = sharedPrefs.getString("numberToCallAfterShaking", "111111");
    String numberAfterLocation = sharedPrefs.getString("numberToCallInLocation", "222222");
    boolean checkBoxShake = sharedPrefs.getBoolean("chbShaking", false);
    boolean checkBoxLocation = sharedPrefs.getBoolean("chbLocation", false);
    long shakeDuration = sharedPrefs.getLong("shakeDuration", 1000);
    int numberOfShakeCount = sharedPrefs.getInt("shakeCount", 4);
    int radius = sharedPrefs.getInt("radius", _defaultRadius);
}
```


Fig. 11



servizio. Abbiamo visto nella puntata precedente come cambiare il colore ad una SeekBar tramite aggiunta di un file di stile xml che andavamo ad aggiungere nella cartella *drawable*; se adesso vogliamo dare una forma particolare ai nostri pulsanti, possiamo seguire lo stesso procedimento andando a modificare il tag *shape* del nuovo file *button_green.xml* impostandolo al valore *oval*. Modificando anche i valori *width* e *height* del tag *size* possiamo decidere il grado di ovalizzazione o fare il pulsante precisamente rotondo. Potete trovare un esempio in Fig. 11.

Fig. 12



Nel nostro caso abbiamo creato tre file xml di pulsanti “nuovi” che renderemo attivi andando ad impostare il campo *android:background* nel tag *Button* del nostro file di Layout. Allo stesso modo della SeekBar potremmo anche creare gli effetti di ombra e colore per simulare la pressione del tasto. Il risultato finale è mostrato in Fig. 12.

Anche se l'accelerometro e il localizzatore GPS funzionano a livello di servizio, abbiamo aggiunto le stesse classi anche nell'Activity principale in modo da poter facilitare il debug.

Il tasto “Store Location” permette di memorizzare la posizione corrente che diventerà quella di riferimento e che attiverà la chiamata verso Arduino ogni qual volta ci troveremo nella stessa zona. Il passaggio di questo dato, dall'Activity al servizio avviene tramite il meccanismo dei metodi *putExtra()* e *getStringExtra()* associati all'Intent, come mostrato nel riquadro “Passaggio parametri tra Activity e Servizi” di queste pagine. Allo stesso modo avremmo potuto usare la classe *SharedPreferences* illustrata sempre nello stesso riquadro.

CONCLUSIONI

Ora abbiamo gli strumenti necessari per realizzare una nostra applicazione che risponda agli eventi esterni tramite i sensori analizzati e con la quale comunicare informazioni verso apparati elettronici esterni. Il tutto è reso anche più comodo grazie all'uso del servizio che non ci obbliga a tenere sotto controllo l'applicazione. Nella prossima puntata vedremo come creare un semplice Widget in modo da avere anche un feedback visivo sempre attivo sullo schermo del nostro dispositivo. ■