

Mensile di progettazione elettronica, attualità scientifica, novità tecnologiche.

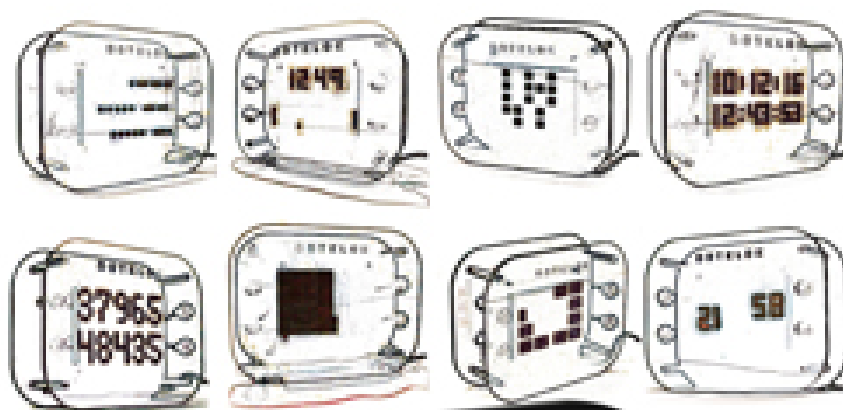
Elettronica In

oltre l'elettronica

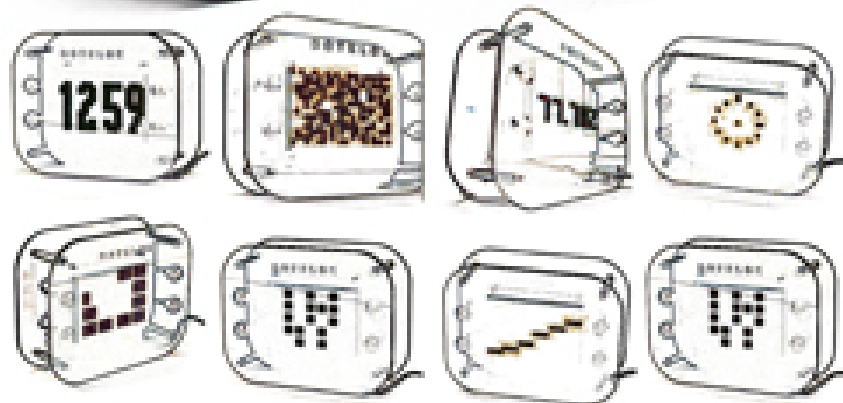
www.elettronica.in.it

€ 6,00

Anno XIX - n. 175
Aprile 2013



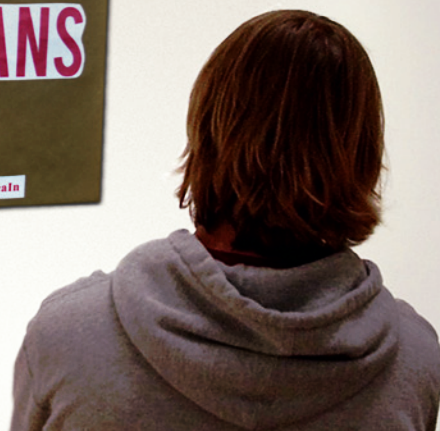
DOTKLOK
orologio grafico
open source



**Controller
avanzato per
3D RAG**

- Gestire i moduli GSM con Arduino
- OpenWheels, la telemetria
- Termostato Web con ADC per RaspberryPi
- Schede di sviluppo per hobbysti a confronto
- Facebook Counter: quanti [Mi piace](#) per la tua pagina?
- myKIT, rilevatore di movimento con sensore PIR
- Piattaforma robotica OmniWheels



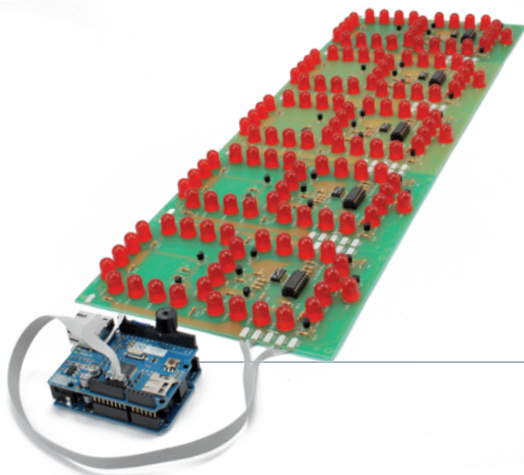


QUANTI Like PER LA TUA PAGINA?

..... di BORIS LANDONI

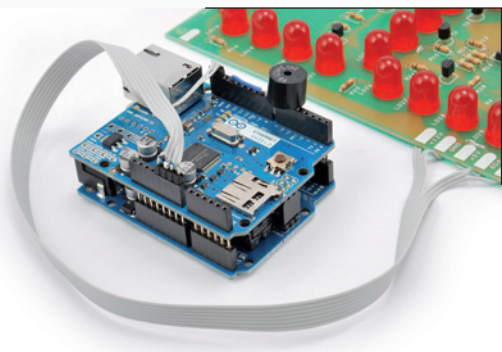
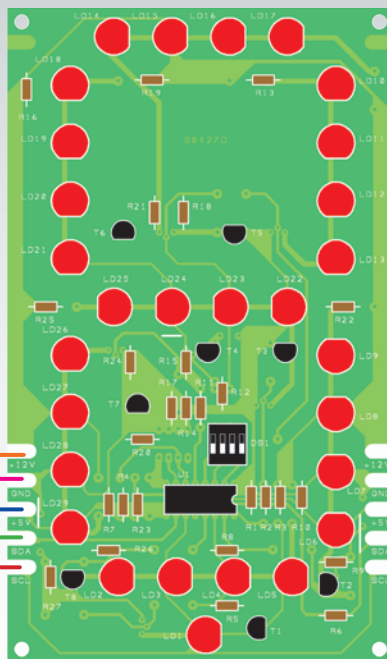
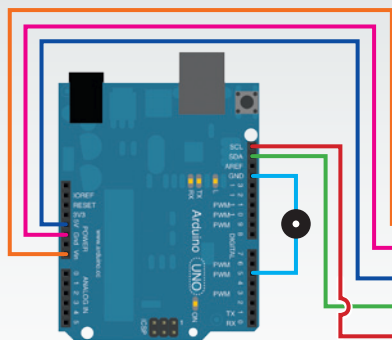
Usando la strategia
old+old=new abbiamo realizzato
un visualizzatore gigante dei
Like raccolti da una pagina
Facebook.

Forse qualcuno ha già detto che “noi siamo ciò che siamo, ma contiamo per i Like che abbiamo” e se non lo ha ancora detto nessuno, ci abbiamo pensato noi. Che si tratti di una pagina personale o di quella di un negozio o di un prodotto, la popolarità ormai si misura in “Like”, ovvero il numero di utenti che hanno espresso il loro gradimento per la pagina, il post, la foto o qualsiasi altro contenuto pubblicato su Facebook. In alcuni settori, come quello del marketing, ogni trucco è lecito per cercare di guadagnare Like e l'icona della manina con il pollice verso l'alto è ormai onnipresente nelle pubblicità e negli spot televisivi. Si tratta di un numero importante e signi-



Lo schema di collegamento

Per il funzionamento del display è necessario collegare l'alimentazione e le linee I2C bus.



ficativo - almeno per il cyber mondo - che sempre più soggetti desiderano mostrare quale misura concreta della propria popolarità; non vogliamo entrare nel merito dell'effettivo valore di questo numero, come non è il caso di cercare di dare una giustificazione ai milioni di visualizzazioni raccolte da alcuni video virali di YouTube, sta di fatto che i Like logorano chi non ce li ha, mentre chi ne ha tanti vuole che tutti lo sappiano. Per questo abbiamo pensato di mettere assieme un po' di idee e di circuiti per creare un display gigante, basato su Arduino, capace di visualizzare senza l'ausilio di un computer il numero di Like collezionati da una specifica pagina.

BASTA CERCARE UN PO'

Quando si tratta di applicazioni

con Arduino, la possibilità che qualcuno abbia già realizzato qualcosa di simile (o comunque collegato a quello che si desidera fare) è molto alta e infatti con qualche semplice ricerca su Google abbiamo individuato il lavoro pubblicato sul blog SkoltiLab denominato "Facebook Like Box" (<http://www.skolti.com/lab/exp8/>). Nato come elaborazione di un esempio fornito con la libreria Ethernet di Arduino, realizzato da Tom Igoe, il programma scritto da David Alcubierre utilizza le API di Facebook che permettono di inviare una stringa al servizio ed ottenere in risposta un XML al cui interno si trova la stringa che contiene il numero di Like relativi alla pagina indicata nella richiesta.

Il codice scritto da David utilizza come output un display LCD da due righe e 20 caratteri: il classico schermo montato sugli shield LCD disponibili da vari

fornitori, ma per i nostri scopi questo era decisamente troppo piccolo e discreto, mentre volevamo qualcosa di molto più visibile.

Cercando nei nostri archivi abbiamo trovato un progetto di Segnapunti Elettronico per Palalavolo, in cui una serie di 2 + 2 cifre a LED da 15 x 8 centimetri ciascuna costituivano il tabellone. La caratteristica del progetto pubblicato ad Aprile 2002 era la possibilità di collegare ciascuna cifra singola - realizzata con un proprio circuito stampato - in serie con altre senza complicare la comunicazione. Il chip utilizzato sul display è un PCF8574A, ovvero un expander di linee I/O pilotabile tramite I2C e indirizzabile tramite tre bit; ideale quindi per il collegamento ad un Arduino che nella comunicazione con questo protocollo è decisamente ben messo.

Ipotizzando anche una certa lentezza nel cambiamento del valore indicato, abbiamo pensato di aggiungere un cicalino che suona una breve melodia ad ogni incremento del valore ricevuto da Facebook.

GLI INGREDIENTI HARDWARE

Al termine della nostra ricerca fra i materiali in laboratorio e in rete ci siamo quindi trovati con il nostro schema a blocchi composto da un Arduino, uno shield Ethernet, cinque moduli da una cifra e un buzzer.

Arduino e la scheda Ethernet non necessitano di commenti, mentre due parole vale la pena spenderle sull'expander di I/O che grazie al protocollo di

0	1	0	0	A2	A1	A0	R/W
---	---	---	---	----	----	----	-----

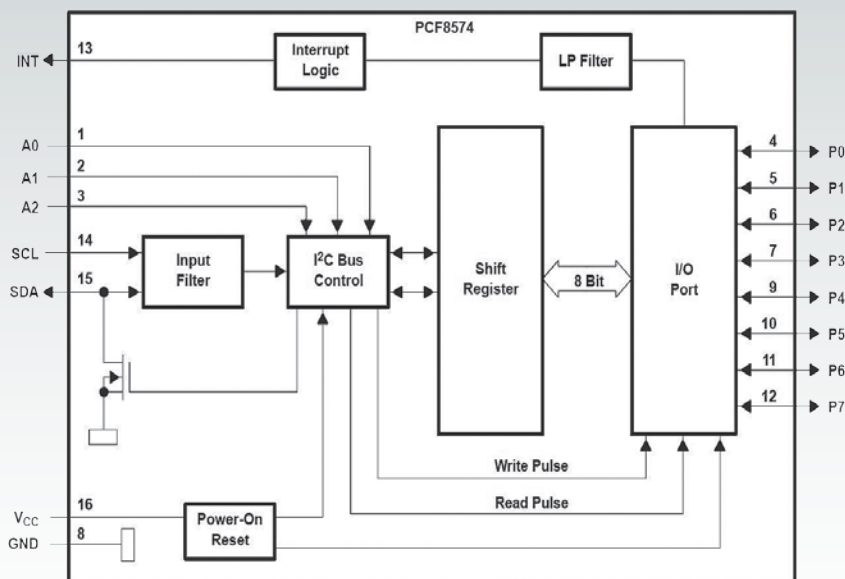
PCF8574

0	1	1	1	A2	A1	A0	R/W
---	---	---	---	----	----	----	-----

PCF8574A

Fig. 1

Fig. 2



comunicazione I2C consente di gestire un significativo numero di linee di comunicazione bidirezionale.

Di PCF8574 sul mercato se ne trovano due tipi: con e senza "A" alla fine. Quello senza A ha un byte di selezione dell'indirizzamento che ha i BIT 5 e 4 settati a 0, mentre il PCF8574A ha i medesimi bit impostati a 1 (Fig. 1).

Per tutto il resto le due versioni di integrato si comportano allo stesso modo, quindi come mai questa differenza? Si tratta di una soluzione molto pragmatica per passare da 8 a 16 chip collegabili in cascata sul medesimo bus I2C, portando le linee di I/O da 64 a 128.

Lo schema a blocchi interno del PCF8574 tratto dal datasheet ci presenta una struttura molto lineare e di facile comprensione, dove i dati seriali in ingresso su SDA vengono convertiti in paralleli da uno shift register e viceversa.

Le porte di I/O hanno caratteristiche particolari e non supportano a livello alto oltre il miliampere, mentre a livello basso possono arrivare a 25 milliam-

pere. Il range operativo del chip va da 2,5 a 6 V (Tabella 1).

Oltre a questo, la comunicazione I2C è di tipo a bassa velocità, con clock al massimo di 100 KHz che corrisponde alla prima fascia di velocità definita dalle specifiche di questo protocollo (Tabella 2).

Con queste caratteristiche, l'uso di questo integrato per i nostri display richiede inevitabilmente l'uso di un transistor per ciascuno dei sette segmenti che vorremo realizzare con i 4 LED rossi.

Nella pratica, inoltre, il fatto di usare un integrato per una cifra è solo una ingegnosa soluzione che trasforma gli otto bit disponibili in sette segmenti e un punto. Attraverso la definizione di una bitmap specifica per ogni numero avremo la trasformazione di un byte inviato all'integrato in una cifra visualizzata con l'accensione dei segmenti rilevanti.

La definizione dell'indirizzo di ciascun chip avviene tramite gli opportuni dip switch a tre vie presenti in ciascuna basetta del display, dopodiché basterà riportare l'opportuno indirizzo

Tabella 1

	MIN	MAX	UNIT
V _{cc} Supply voltage	2.5	6	V
V _{IH} High-level input voltage	0.7 x V _{cc}	V _{cc} + 0.5	V
V _{IL} Low-level input voltage	-0.5	0.3 x V _{cc}	V
I _{OH} High-level output current		-1	mA
I _{OL} Low-level output current		25	mA
T _A Operating free air temperature	-40	85	°C

Tabella 2

Standard mode (Sm)	I2C fino a 100 kbit/s
Fast-mode (Fm)	I2C fino a 100 kbit/s
Fast-mode (Fm)	I2C fino a 1 Mbit/s
High-speed mode (Hs-mode)	I2C fino a 3.4 Mbit/s

nello sketch per individuare correttamente ciascuna cifra. Questo ci permette anche di utilizzare indifferentemente integrati di tipo PCF8574 o PCF8574A purché si impostino i relativi Bit 4 e 5 nel modo corretto.

L'intero circuito è alimentato a 12 volt, con la scheda Arduino che provvede a fornire al PCF8574 l'alimentazione tramite il proprio regolatore che converte i 12V in ingresso nei 5V richiesti; sempre tramite l'alimentazione collegata alla presa di Arduino viene alimentato lo shield Ethernet.

Fate riferimento allo schema di cablaggio per avere tutte le indicazioni sui vari collegamenti fra le schede.

LO SKETCH

Dato che il punto di partenza è il lavoro fatto prima da Igoo con il suo Twitter Client e poi da David Alcubierre (Skolti) per l'aggiunta della parte di interrogazione dell'API di Facebook, a noi è restata la scrittura di codice per la sostituzione della visualizzazione da LCD al nostro display LED a 5 cifre,

con la musichetta in caso di incremento. Vediamo quindi le parti del programma che abbiamo aggiunto al materiale preesistente, mentre per l'utilizzo della libreria e delle funzioni di Ethernet e del codice di interrogazione delle API di Facebook, assieme al parsing dell'XML, potete far riferimento all'esempio "Twitter Client with Strings" fornito con l'installazione core di Arduino 1.0.x. Per Ethernet dobbiamo necessariamente includere le due librerie SPI e Ethernet, mentre per la comunicazione tramite I2C si usa la libreria Wire, ricordando che dall'IDE 022 al 1.0 sono cambiate delle funzioni per un generale riallineamento delle librerie legate alla comunicazione seriale: in pratica i termini "send" e "receive" diventano "write" e "read".

All'inizio dello sketch troviamo i define che associano ai cinque pannelli i relativi indirizzi attraverso le costanti "EXPANDER_n" con "n" da 0 a 4; nello sketch si ipotizza l'uso di integrati PCF8574A e quindi la maschera di bit è "0B0011xxx" mentre se gli integrati fossero stati di tipo PCF8574 avremmo usato "0B0000xxx". Per la melodia abbiamo scelto "twinkle little star" e una classica routine per la produzione di note "PlayNote" che trovate verso la fine dello sketch. In questa parte iniziale definiamo i vettori con le note e la durata delle stesse, assieme al tempo e alla lunghezza. Il pin a cui collegare il buzzer è il D5.

Per il collegamento in rete abbiamo preferito la configurazione con IP fisso attraverso l'istruzione

```
IPAddress ip(192,168,0,99);
```

ma è anche possibile utilizzare una modalità basata su DHCP. Nelle righe successive istanziamo il client Ethernet e creiamo le variabili che ci servi-

ranno durante l'esecuzione.

La parte di setup() la usiamo per inserire nell'array delle bitmap dei numeri – quello che contiene i segmenti da accendere per rappresentare ciascuna cifra – con valori opportuni, ricordando che il bit a "0" tiene il segmento spento e che quando viene resettato l'integrato PCF8574 questo attiva tutte le proprie uscite con il livello alto.

Per consentire un minimo di debug abbiamo anche attivato la porta seriale di comunicazione con il PC e sulla consolle dell'IDE potranno essere visualizzati del messaggi utili alla comprensione di cosa sta succedendo a livello software su Arduino.

Prima di terminare il setup() accendiamo per due secondi tutti i segmenti di tutte le 5 cifre, quindi facciamo un ciclo da 10 per visualizzare contemporaneamente su ciascun display le cifre da 0 a 9. Alla fine di questo ciclo, spegniamo tutte le cifre inviando ai 5 display il valore 0, ovvero con tutti i bit a "0".

Il loop principale controlla che il client sia connesso al server dichiarato come stringa all'inizio dello sketch con

```
char serverName[] = "api-read.facebook.com";
```

quindi inizia a fare il parsing di quanto ricevuto in xml, alla ricerca del valore dei Like con questa semplice routine

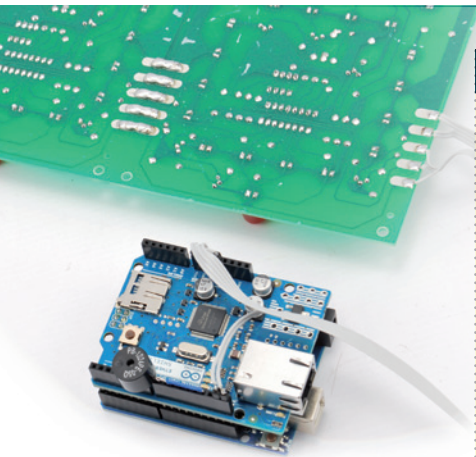
```
if ( currentLine.endsWith("<fan_count>")) {
    readingFbcount = true;
    fbcount = "";
}

if (readingFbcount) {
    if (inChar != '<') {
        if (inChar != '>') {
            fbcount += inChar;
        }
    }
}
```

Il dato caricato carattere per carattere nella variabile fbcount viene utilizzato dalle istruzioni successive che per cinque volte leggono un carattere e lo trasferiscono sul display posizionalmente corrispondente con la funzione PCFwrite definita a fine sketch. Prima della visualizzazione del valore di fbcount, viene anche fatto un confronto con il valore numerico della lettura precedente, così da individuare quando c'è stato un aumento dei Like e dare il via alla musichetta attraverso play(). Questa routine fa riferimento alle ulteriori due routine playTone e playNote che permettono di generare la frequenza corrispondente alla lettera della nota presente nell'array della melodia, per la

Listato 1

```
<?xml version="1.0" encoding="UTF-8" ?>
<fql_multiquery_response xmlns="http://api.facebook.com/1.0/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://api.facebook.com/1.0/
    http://api.facebook.com/1.0/facebook.xsd" list="true">
  <fql_result>
    <name>page_info</name>
    <fql_result_set list="true">
      <page>
        <name>Rivista Elettronica In</name>
        <fan_count>9253</fan_count>
        <page_url>http://www.facebook.com/ElettronicaIn</page_url>
        <type>PUBLISHER</type>
      </page>
    </fql_result_set>
  </fql_result>
</fql_multiquery_response>
```



L'unione dei display avviene con delle semplici saldature che garantiscono solidità meccanica e collegamento elettrico.

Listato 2

```
{
  "about": "Dedicato a tutti gli appassionati della rivista ELETTRONICA IN, mensile di elettronica applicata, attualit\u00e0 scientifica, novit\u00e0 tecnologiche.",
  "category": "Publisher",
  "company_overview": "Elettronica In \u00e8 una rivista mensile (10 numeri all'anno) di elettronica applicata con intenti didattici e divulgativi, rivolta a quanti operano nel campo della progettazione elettronica. In particolare ci rivolgiamo a quanti lavorano nei laboratori di Ricerca e Sviluppo e negli Uffici Tecnici di piccole e medie aziende nonch\u00e9 a quanti frequentano Corsi di Studio nel settore elettronico e informatico (studenti universitari e di scuola media superiore) ed ai loro insegnanti.\nPrestitiamo particolare attenzione anche a coloro che, pur non operando professionalmente in questi campi, sono affascinati dalla possibilit\u00e0 di realizzare in proprio dispositivi elettronici per gli impieghi pi\u00f9 vari. \n\nElettronica In con i suoi 70 mila lettori al mese \u00e8 la Rivista di Elettronica pi\u00f9 diffusa in Italia.",
  "founded": "05/07/2010",
  "is_published": true,
  "talking_about_count": 160,
  "username": "ElettronicaIn",
  "website": "http://www.elettronica.in",
  "were_here_count": 0,
  "id": "123250477713173",
  "name": "Rivista Elettronica In",
  "link": "http://www.facebook.com/ElettronicaIn",
  "likes": 9255,
  "cover": {
    "cover_id": 500242013347349,
    "source": "http://sphotos-a.ak.fbcdn.net/hphotos-ak-ash3/s720x720/544474_500242013347349_762380625_n.png",
    "offset_y": 0,
    "offset_x": 0
  }
}
```

durata specificata nel vettore beats. Il collegamento al server viene effettuato dal codice al di fuori del loop principale e contenente la chiamata al server definito nella parte iniziale dello sketch. La chiamata invia con un GET http la stringa contenente la query necessaria ad ottenere un XML come quello indicato nel **Listato 1**.

Perché i dati siano quelli relativi alla pagina che interessa è indispensabile inserire l'ID corretto (indicato dalla freccia rossa nel **Listato 2**), corrispondente alla pagina della quale si vuole visualizzare il numero di Like (nell'XML vedete la nostra pagina e il nostro ID).

La stringa da modificare è molto lunga e la pubblichiamo abbreviata al fine di evidenziare il punto dove inserire l'ID.

```
client.println("GET /restserver .... %20IN%20(123250477713173)%22&sdk=joe HTTP/1.1");
```

L'ID è fra parentesi e si trova quasi alla fine della stringa stessa.

Il recupero del vostro ID lo dovete fare online, prima accedendo alla vostra pagina e poi editando l'URL sostituendo a "www" la stringa "graph". Per chiarezza:

<http://www.facebook.com/ElettronicaIn> -> visualizza la pagina Facebook

<http://graph.facebook.com/ElettronicaIn> -> visualizza la scheda con varie info fra cui l'ID come riportato nel **Listato 2**.

Dovete usare l'ID che si trova prima del campo "name:" ignorando altri ID che trovate eventualmente all'interno dei dati visualizzati. Potete "colaudare" il dato che avete recuperato inserendo nel vostro browser – con una certa pazienza – la stringa che riportiamo, sostituendo >>vostroID<< con il numero che avete appena recuperato.

```
http://api-read.facebook.com/restserver.php?format=xml&method=fql_multiquery&pretty=0&queries={%22page_info%22%3A%22select%20name%2Cfan_count%20from%20page%20where%20page_id%20IN%20(>>vostroID<<)%22}&sdk=joe
```

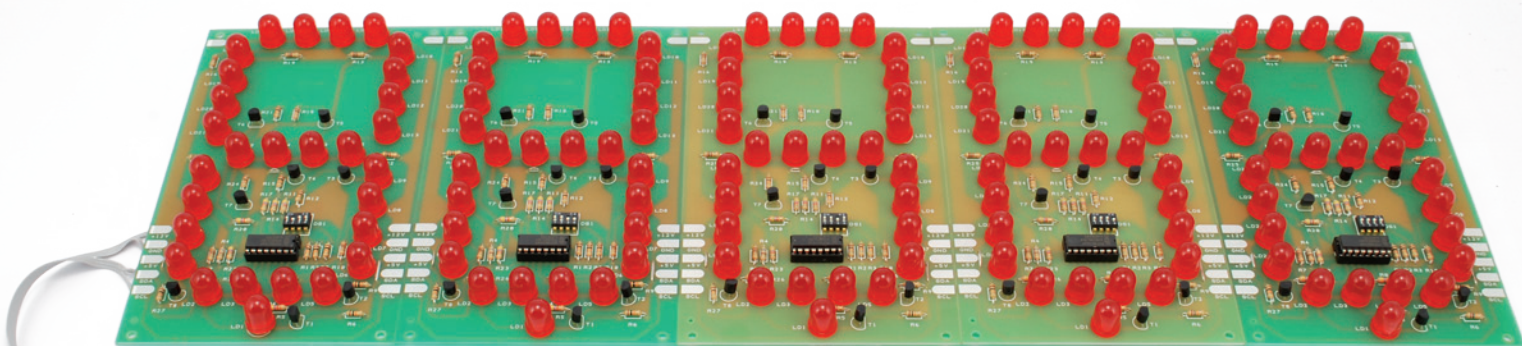
```
<fql_multiquery_response
  xmlns="http://api.facebook.com/1.0/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://api.facebook.com/1.0/
    http://api.facebook.com/1.0/facebook.xsd" list="true">
  <fql_result>
    <name>page_info</name>
    <fql_result_set list="true">
      <page>
        <name>Rivista Elettronica In</name>
        <fan_count>9255</fan_count>
      </page>
    </fql_result_set>
  </fql_result>
</fql_multiquery_response>
```

In questo caso la risposta è più breve in quanto abbiamo semplificato la stringa di query richiedendo meno campi.

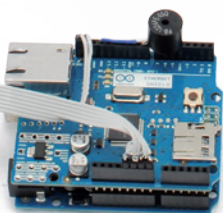
Le altre routine contenute nello sketch servono per la trasmissione ai display e per la conversione da stringa a numero, seguite dalle parti di generazione dei toni già illustrate prima.

LA COMPILAZIONE DELLO SKETCH

Prima di compilare e caricare sulla scheda, dovete necessariamente modificare i dati relativi all'IP



Per il corretto funzionamento è necessario impostare correttamente i dip switch presenti su ogni display.



CIFRA	DIP1	DIP2	DIP3	DIP4
Unità	OFF	ON	ON	ON
Decine	OFF	ON	ON	OFF
Centinaia	OFF	ON	OFF	ON
Migliaia	OFF	ON	OFF	OFF
Decine di migliaia	OFF	OFF	ON	ON

assegnato al vostro Arduino sulla vostra LAN e definito gli indirizzi dei display in base alla loro reale configurazione. Anche l'ID va inserito nello sketch prima della compilazione e dopo averlo "collaudato" online tramite browser seguendo le indicazioni appena fornite. Se non intervenite su tutti questi elementi prima di caricare il firmware sul vostro Arduino avrete ottime possibilità di avere un sistema non funzionante o comunque che visualizza i Like di qualcun altro.

Una volta che avete compilato e caricato il firmware potete fare un primo collaudo: le istruzioni di stampa su porta seriale delle stringhe di debug vi permettono di verificare il software senza collegare il display a LED, ma solo montando lo shield Ethernet e realizzando il collegamento alla vostra LAN. Ad ogni interrogazione del server avrete la stampa su console dell'IDE – che ovviamente dovete attivare – dei Like correnti; se c'è un incremento, avrete anche la stampa del numero di nuovi Like.

IL MONTAGGIO

Per questo progetto vi invitiamo innanzitutto a scaricare gratui-

tamente dal nostro sito l'articolo relativo al display modulare pubblicato sul numero 68 di Aprile 2002 e disponibile in kit con il codice FT427DK da Futura Shop. Poi dovete montare sul vostro Arduino UNO lo shield Ethernet. Il cablaggio del display a cinque cifre LED è quello riportato nello schema di montaggio, con i pin di SCL e SDA prelevati dagli appositi pin del connettore a pin strip in prossimità del tastino di Reset (Arduino UNO rev 3).

Ricordate anche di assegnare a ciascuna scheda il proprio indirizzo attraverso i dip switch così che il valore del define EXPANDER_n sia corrispondente.

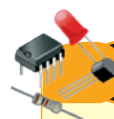
IL COLLAUDO FINALE

Anche in assenza di collegamento di rete, il firmware ha una routine di inizializzazione del display che prima accende tutti i segmenti e poi in sequenza compone i numeri da 0 a 9 in contemporanea sulle cinque cifre. In questa fase può anche essere utile tenere la console dell'IDE aperta e controllare che con il cavo di rete collegato, dopo un reset di Arduino, venga visualizzato il messaggio

My address: <indirizzo IP della scheda>

Al quale dovrebbero seguire i messaggi con il conteggio dei Like.

Se avete un sito di particolare successo e non vi bastano cinque cifre per i Like non c'è comunque problema: basta aggiungere fisicamente una o più cifre alla serie e aumentare in proporzione i define e i blocchi di codice che scrivono una cifra dopo l'altra il numero per ciascun display.



per il MATERIALE

Tutti i componenti necessari per realizzare questo progetto sono di facile reperibilità. Ciascuna unità del display (cod. FT427DK) costa in scatola di montaggio 24,00 Euro. Le board ARDUINOUNO-REV3 e SHIELDETHERNET costano rispettivamente 24,50 e 42,00 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>

Impariamo a gestire con Arduino i moduli GSM per comunicare tramite dati, chiamate ed SMS sfruttando le librerie dedicate più recenti. Prima puntata.

GESTIRE I MODULI GSM CON ARDUINO

di MARCO MARTINES

Nei precedenti numeri della rivista, in particolare nel fascicolo di Aprile 2012, è stato presentato un potente shield in grado di consentire ai principali dispositivi della famiglia Arduino (Uno e Mega) di interfacciarsi con i moduli della SIMCom: SIM900 e SIM908.

Tali moduli offrono la possibilità di interagire con il mondo GSM, GPRS e GPS attraverso semplici comandi impartiti da Arduino utilizzando l'apposita libreria open-source dedicata. Risulta dunque possibile effettuare chiamate, inviare SMS ed email con semplici chiamate a funzione.

Rispetto all'ultimo articolo presentato, inerente a tale shield, la libreria è stata migliorata e vi sono state inserite ulteriori funzioni, come ad esempio, la gestione dei toni DTMF e la possibilità di invio delle email.

In questo articolo, verranno illustrate le principali novità a livello software introdotte nella libreria e successivamente, in puntate successive, si mostrerà qualche esempio per lo svolgimento delle principali funzioni che tale shield mette a disposizione.

NOVITA' SOFTWARE

La libreria open source, con lo scopo di realizzare un'interfaccia ad alto livello tra utente e modulo SIM90X, è in continuo sviluppo e può essere reperita online all'indirizzo <http://code.google.com/p/gsm-shield-arduino/>.

Come fatto in precedenza nelle precedenti puntate riguardo l'utilizzo del GSM/GPRS/GPS shield, mostreremo le principali novità a livello software introdotte nella libreria. Tali novità riguardano l'implementazione di metodi per l'invio di toni

DTMF durante una telefonata (di grande utilità per comunicare eventuali stati degli ingressi durante una telefonata) e la possibilità di inviare email attraverso la connessione GPRS.

Le funzioni riguardo l'invio di messaggi di posta elettronica verranno illustrate nei prossimi articoli che tratteranno l'argomento in maniera più dettagliata. Ricordiamo che per motivi legati all'utilizzo della RAM la libreria è suddivisa in classi includibili o meno, ognuna con specifiche funzioni. Di conseguenza per l'utilizzo delle telefonate dovrà essere inclusa la classe apposita con il comando:

```
#include "call.h"
```

Oppure per gli sms si dovrà inserire all'interno dello sketch:

```
#include "sms.h"
```

L'invio dei toni DTMF (Dual-Tone Multi-Frequency), essendo una funzione strettamente legata alle chiamate, è stata inclusa all'interno della classe relativa a queste.

Ricordiamo che per l'utilizzo di tali funzioni è sufficiente, dopo aver incluso il file relativo, istanziare l'oggetto all'interno dello sketch. Nelle funzioni riportate nella Tabella 1 si fa riferimento a un oggetto creato con il seguente comando all'inizio dello sketch: *CallGSM call;*

ESEMPIO 1

Nel primo esempio realizzeremo uno sketch in grado di effettuare una chiamata al numero preimpostato quando si verificherà una condizione di allarme.

Tale condizione si verifica quando l'ingresso analogico relativo al pin scelto assume un valore inferiore a 50. Ricordando che

Listato 1

```
#include "SIM900.h"
#include <SoftwareSerial.h>
#include "call.h"

CallGSM call;
char number[]="3921234567";
int value=0;
int pin=A0;

void setup()
{
  Serial.begin(9600);
  if (gsm.begin(2400))
    Serial.println("\nstatus=READY");
  else Serial.println("\nstatus=IDLE");
};

void loop()
{
  value=analogRead(pin);
  Serial.println(value);
  if (value<50){
    Serial.println("Allarme");
    if (call.CallStatus() != CALL_ACTIVE_VOICE){
      Serial.println("Chiamo");
      call.Call(number);
    }
  }
  else
    Serial.println("Chiamata in corso");
  delay(1000);
};
```

Arduino campiona i voltaggi in ingresso con valori a 10 bit, se, con la configurazione di base, 1023 equivale a 5 Volt, 50 equivarrà a circa 0,25 Volt.

Una condizione simile, con un'opportuna scelta dei valori di soglia, può rappresentare il superamento di una temperatura limite da parte di un sensore di calore o analogamente l'incremento (o decremento) di luminosità misurata da una fotoresistenza.

Con pochi semplici passaggi è possibile realizzare un allarme in grado di comunicare senza spese di gestione (sarà sufficiente rifiutare la telefonata, per evitare l'addebito della chiamata) il verificarsi di una condizione di allarme.

Vediamo ora come realizzare quanto detto.

Per prima cosa viene inclusa la libreria per le telefonate con il

comando apposito e successivamente ne viene istanziato un oggetto attraverso i comandi:

```
#include "call.h"
```

```
CallGSM call;
```

Se si verifica la condizione di allarme sopracitata viene effettuata una chiamata attraverso la funzione

```
call.Call(number);
```

Per evitare che venga effettuata una telefonata, quando ve ne è già un'altra attiva, viene prima controllato lo stato delle telefonate e solo in caso di nessuna telefonata attiva viene chiamata la funzione precedente.

Per verificare lo stato delle chiamate è sufficiente richiamare la funzione *call.CallStatus()* e assicurarsi che restituisca un valore diverso da *CALL_ACTIVE_VOICE*.

ESEMPIO 2

L'esempio precedente potrebbe

```
void SendDTMF(char*
number_string, int time)
```

Durante una chiamata attiva, invia la stringa passata come parametro, specificando la durata di ogni tono in decimi di secondo. La stringa potrà essere composta dai seguenti caratteri: 0-9, #, *, A-D.

```
call.SendDT-
MF("1,2,3,4,*,2)
```

Tabella 1

Listato 2

```
#include "SIM900.h"
#include <SoftwareSerial.h>
#include "call.h"

CallGSM call;
char number[]="3921234567";
int value=0;
int value_old;
int pin=9;

void setup()
{
  Serial.begin(9600);
  if (gsm.begin(2400))
    Serial.println("\nstatus=READY");
  else Serial.println("\nstatus=IDLE");
  value=digitalRead(pin);
  value_old=digitalRead(pin);
};

void loop()
{
  value=digitalRead(pin);
  if (value!=value_old){
    Serial.println("Allarme");
    if (call.CallStatus() != CALL_ACTIVE_VOICE){
      Serial.println("Chiamo");
      call.Call(number);
      delay(1000);
    }
    else{
      Serial.println("Chiamata in corso");
      if (value==0){
        call.SendDTMF("0", 1);
        delay(5000);
      }
      else{
        call.SendDTMF("0,0", 1);
        delay(5000);
      }
    }
  }
  //aggiorniamo solo se la chiamata non è attiva
  if (call.CallStatus() != CALL_ACTIVE_VOICE){
    Serial.println("Aggiorno dati");
    value_old=value;
  }
  delay(500);
};
```

generare qualche problema per valori prossimi a quello di soglia, infatti per via del rumore sul sensore o per rapide oscillazioni della grandezza misurata intorno al valore limite, l'allarme potrebbe ripetutamente attivarsi e disattivarsi, generando disturbo inutile.

Per ovviare a tale situazione è consigliabile l'utilizzo o di un'isteresi o di un filtro sulla misura.

Per quanto riguarda l'isteresi (dal greco ὑστέρησις 'ritardo'), vengono scelti due valori di soglia differenti per ritardare l'attivazione e disattivazione dell'allarme, in modo tale che sia necessario allontanarsi in maniera significativa dalla condizione di allarme affinché questa venga disattivata (Fig. 1).

Si mostra ora, invece, come realizzare un piccolo filtro passa-basso a tempo discreto del primo ordine per evitare che le oscillazioni ad alta frequenza possano attivare involontariamente e ripetutamente la condizione di allerta.

A differenza del precedente esempio utilizzeremo in aggiunta le seguenti variabili:

```
int mov_av=0;
float threshold=0.9;
Nel setup inizializzeremo il valore filtrato al primo valore letto con il comando:
lpf_val =analogRead(pin);
```

E nel loop() aggiorneremo il filtro nel seguente modo:

```
value=analogRead(pin);
lpf_val = lpf_val *threshold+(1-
threshold)*value;
```

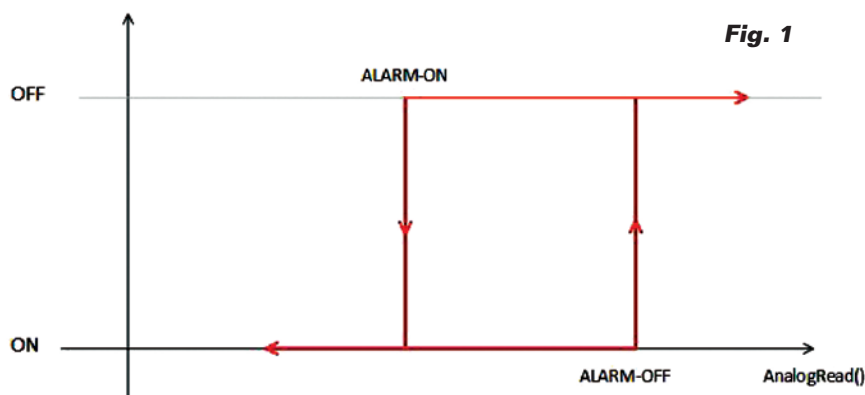


Fig. 1

La variabile lpf_val conterrà dunque il valore filtrato. La frequenza di taglio è determinata dal valore di threshold (in questo caso scelto pari a 0.9). Valori prossimi all'unità abbassano la frequenza di taglio, viceversa, valori prossimi allo zero alzano la frequenza di taglio facendo passare anche le frequenze più alte.

Ovviamente la condizione sarà valutata sul valore filtrato con:

```
if (lpf_val<50)
```

ESEMPIO 3

In questo esempio, simile al precedente, effettueremo una chiamata ogni qualvolta lo stato

Abbiamo parlato di GSM...

Le nostre attività di progettazione di applicazione con la rete GSM risalgono a parecchi anni fa, quando ancora non c'erano i moduli e si usavano i telefonini con degli appositi cavi dati per accedere alla rete cellulare. Con i moduli l'interfacciamento e il controllo si sono semplificati e già nel numero di Aprile 2011 abbiamo affrontato il tema dell'interfacciamento dei moduli GSM con Arduino. La soluzione pubblicata era basata su uno shield dedicato al SIM900 (TDGSM_900) mentre gli Arduino erano ancora alla versione Duemilanove. Nell'articolo, oltre a illu-



Fig. A

strare la nostra scheda su cui il modulo GSM era montato e reso più facilmente interfacciabile, venivano anche illustrate le funzioni accessibili tramite una ricca serie di comandi - implementati tramite libreria - da impartire con la comunicazione seriale alla scheda (Fig. A).

Le nostre attività di progettazione e presentazione dei moduli GSM ha però radici più lontane: il modulo SIM900 è stato utilizzato anche nell'articolo che abbiamo pubblicato a Marzo 2011; era il localizzatore GSM basato proprio su questo modulo e per il quale sono state fornite ampie spiegazioni sull'interfacciamento e il funzionamento. Sempre di SIM900 abbiamo parlato nel numero Maggio/Giugno 2010 con il Telecontrollo GSM Low Cost che iniziava la serie dei nostri articoli basati su questo modulo applicandolo a un funzionale apricancello tanto innovativo quanto economica.

La tecnologia dei moduli si è evoluta con rapidità e quindi a Marzo 2012 abbiamo realizzato un nuovo localizzatore basato sul modulo SIM908 che,

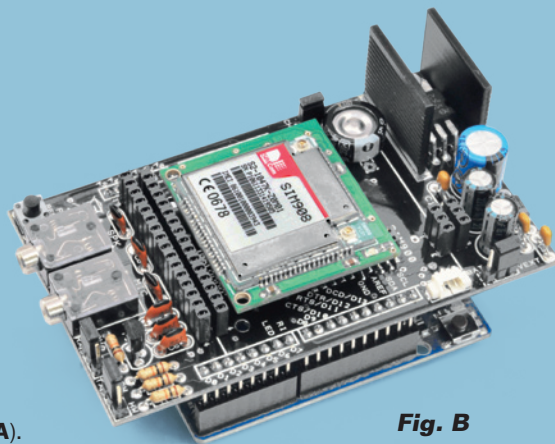


Fig. B

rispetto al precedente, dispone anche di un ricevitore GPS integrato. Nel progetto si spiega l'impiego di entrambe le funzioni del modulo per ottenere un posizionamento molto preciso e la trasmissione del dato con le varie modalità possibili.

Ad Aprile 2012, infine, abbiamo presentato uno shield universale (8190-GSMGPRSSHIELDV2) su cui montare le schede GSM basate su SIM900 e SIM908 di SIMcom per interfacciarle con schede Arduino UNO o compatibili (Fig. B).

Potete scaricare in versione elettronica tutti gli articoli citati utilizzando il nostro servizio per l'accesso e la consultazione degli arretrati disponibile su www.elettronica.in.it.

di uno degli ingressi digitali cambierà di stato. In caso di risposta, sarà emesso un singolo tono DTMF a simboleggiare che lo stato dell'ingresso scelto è basso, viceversa due toni DTMF nel caso sia alto.

Nelle future versioni del SIM908 sarà implementato anche un decoder DTMF con cui sarà possibile decifrare i toni inviati dall'utente al modulo della SIM-Com. Tale funzione risulta già presente sul SIM900.

Listato 3

```
state=call.CallStatusWithAuth(number,8,10);
if(state==CALL_INCOM_VOICE_AUTH)
{
    [...]
}
delay(500);
};
```

Osservando il codice, si nota la semplice logica per la condizione di allarme che, in caso affermativo, attiva la routine di comunicazione dello stato al numero indicato.

Come per i precedenti esempi, viene controllata la presenza di una chiamata attiva ed in caso negativo se ne effettua una verso il numero indicato, mentre in caso affermativo viene prima

byte CallStatus(void);	Restituisce lo stato riguardo le telefonate. I possibili byte di output sono enumerati con i seguenti nomi: CALL_NONE oppure 0: Nessuna chiamata. CALL_INCOM_VOICE oppure 1: Chiamata in arrivo. CALL_ACTIVE_VOICE oppure 2: Chiamata attiva.	If(call.CallStatus()== CALL_ACTIVE_VOICE) Serial.println("CHIAMATA IN CORSO");
byte CallStatusWithAuth(char* phone_number, byte first_authorized_pos, byte last_authorized_pos);	A differenza del precedente, distingue se la chiamata attiva, o in arrivo, appartiene a un numero salvato sulla SIM, in una posizione compresa tra quella iniziale e finale, passate come parametri. I possibili byte di output sono enumerati con i seguenti nomi: CALL_NONE oppure 0: Nessuna chiamata. CALL_INCOM_VOICE_AUTH oppure 3: Chiamata in arrivo da un numero autenticato. CALL_INCOM_VOICE_NOT_AUTH oppure 4: Chiamata in arrivo da un numero non autenticato.	If(call.CallStatusWithAuth (number,9,9)==CALL_INCOM_VOICE_AUTH) Serial.println("CHIAMATA IN ARRIVO DA UN NUMERO AUTENTICATO, SALVATO SULLA SIM IN POSIZIONE 9");

Tabella 2

controllato lo stato dell'ingresso indicato e successivamente questo viene comunicato tramite i toni DTMF.

```
call.SendDTMF("0", 1);  
call.SendDTMF("0,0", 1);
```

Il primo comando invia una stringa composta da un solo numero, al contrario, la seconda invia una stringa composta da due caratteri (si noti l'utilizzo della virgola per separare i caratteri). È importante l'aggiunta di un ritardo per evitare di comunicare in continuazione lo stato tramite toni DTMF con il rischio di fraintendere la comunicazione. In questo caso è stato scelto un delay di 5 s.

Alla fine del codice si può notare come l'aggiornamento dello stato venga effettuato solo a chiamata non attiva, per evitare di effettuare differenti comunicazioni durante la stessa telefonata.

ESEMPIO 4

Al contrario dell'esempio precedente, in questo caso saremo noi a decidere quando interrogare Arduino per la verifica degli ingressi attraverso una chiamata. Il modulo dovrà controllare se il numero che sta effettuando la chiamata risulta tra quelli autorizzati ed eventualmente accettare la telefonata comunicando come in precedenza lo stato attraverso i toni DTMF.

Per quanto riguarda il meccanismo di autenticazione, utilizzeremo la funzione:

```
call.CallStatusWithAuth(number,8,10);  
che verificherà se il numero che sta effettuando la telefonata risulta salvato sulla SIM in una posizione compresa tra la 8 e la 10. All'interno della prima stringa passata come parametro verrà salvato il numero che sta effettuando la chiamata. È sempre consigliabile scegliere intervalli non troppo grandi, in
```

Listato 4

```
#include "SIM900.h"  
#include <SoftwareSerial.h>  
#include "call.h"  
  
CallGSM call;  
char number[15];  
int value=0;  
int pin=9;  
byte state;  
  
void setup()  
{  
  Serial.begin(9600);  
  if (gsm.begin(2400))  
    Serial.println("\nstatus=READY");  
  else Serial.println("\nstatus=IDLE");  
  pinMode(9,INPUT);  
};  
  
void loop()  
{  
  state=call.CallStatusWithAuth(number,8,10);  
  //Serial.println(state);  
  if(state==CALL_INCOM_VOICE_AUTH)  
  {  
    value=digitalRead(pin);  
    delay(1000);  
    Serial.println("Chiamata in corso da numero autorizzato");  
    if (value==0) {  
      delay(1000);  
    }  
    else{  
      Serial.println("Hang Up");  
      call.HangUp();  
      delay(1000);  
    }  
  }  
  delay(500);  
};
```

quanto, ogni interrogazione della SIM per ottenere i numeri dalla rubrica, richiede qualche decimo di secondo.

Riportiamo solo la porzione di codice di interesse per la verifica del numero, dove al suo interno troveremo anche le funzioni per gestire la risposta e la chiusura della telefonata:

```
call.PickUp();  
call.HangUp();
```

ESEMPIO 5

Sempre in un'ottica di risparmio, vediamo come realizzare un esempio simile al precedente senza generare alcun costo relativo alla telefonata.

L'idea è di comunicare lo stato di input rifiutando o meno una chiamata in arrivo.

In caso di stato basso, non verrà intrapresa alcuna azione, lasciando squillare il modulo. Viceversa,

se lo stato dell'ingresso sarà alto, la telefonata verrà rifiutata.

In questo modo si potrà essere a conoscenza dello stato senza preoccuparsi dei costi di ogni comunicazione.

All'interno del codice si potrà notare il meccanismo di autenticazione. Infatti, nel caso di una chiamata in arrivo da parte di un numero non autorizzato, non verrà intrapresa nessuna azione, lasciando squillare a prescindere dallo stato degli input.

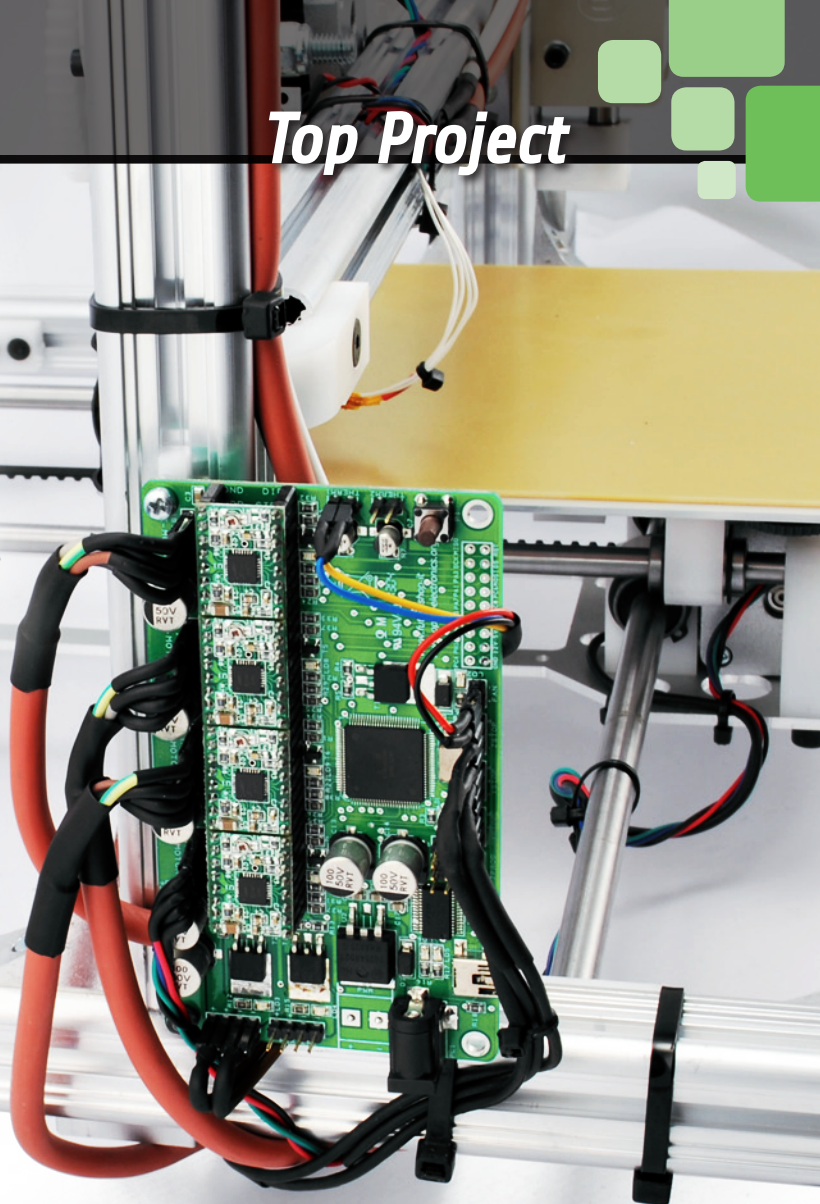
Con questo articolo sono state mostrate alcune delle principali funzioni che lo shield unito a un modulo GSM mette a disposizione con il semplice utilizzo delle telefonate.

Nelle prossime puntate verranno affrontate problematiche simili utilizzando le classi relative agli SMS, alla connessione GPRS e all'utilizzo del GPS.

UNA NUOVA SCHEDA PER

3DRAG

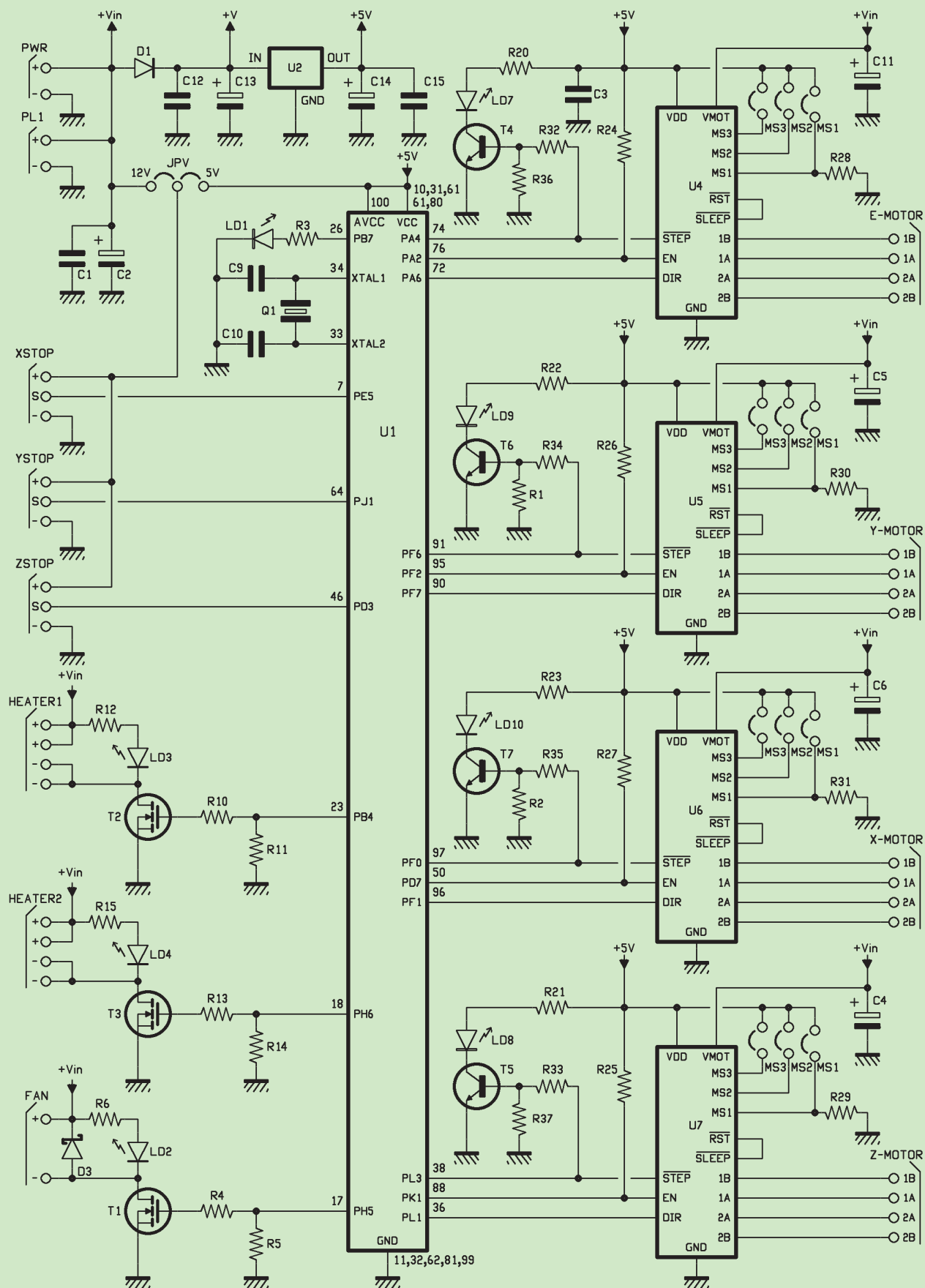
Miglioriamo la nostra 3Drag con questa nuova scheda che permette il controllo software di una ventola e, grazie al processore ATmega2560, consente di aggiungere codice e dispositivi accessori come un display o una scheda SD Card.

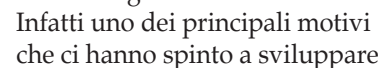


di BORIS LANDONI

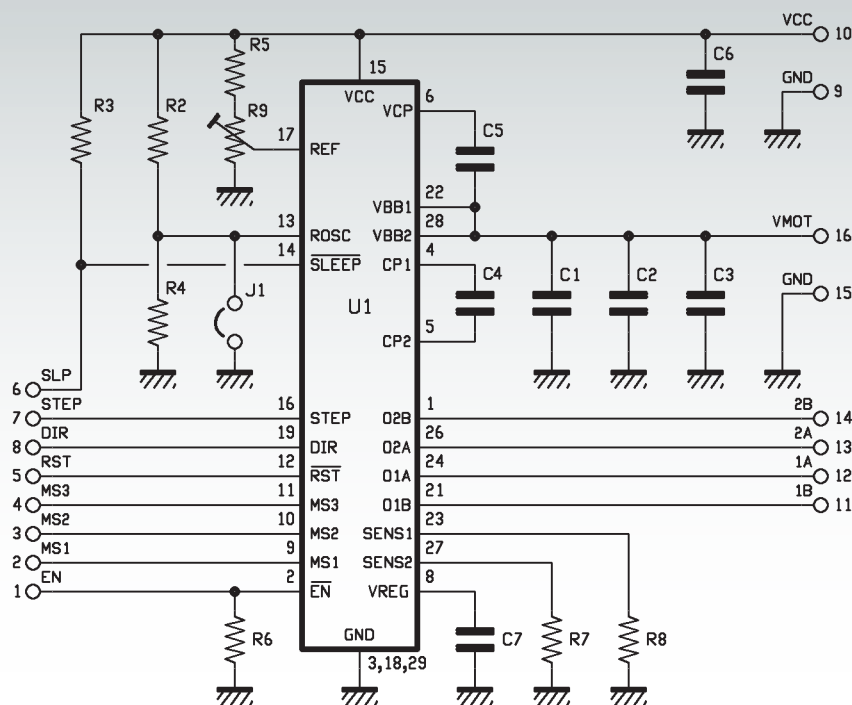
Quando abbiamo realizzato la nostra stampante 3D, per praticità abbiamo cercato di utilizzare alcune parti che fossero già collaudate e reperibili, nel pieno rispetto della filosofia open source. In particolare, abbiamo deciso di utilizzare la scheda elettronica Sanguinololu che offriva un hardware ben integrato, funzionale e ben collaudato. Il firmware "open" che abbiamo scelto, Marlin di Erik van der Zalm, ha risolto gran parte dei problemi tecnici di interfacciamento con il computer, di interpretazione dei codici creati dal software di slicing, di gestione dei motori passo-passo e del riscaldamento dell'ugello estrusore. Con Sanguinololu abbiamo fatto tanta strada; essendo però innovatori per vocazione, ci siamo guardati attorno valutando nuovi sviluppi, uno riguardava la possibilità di gestire una ventola in grado di soffiare aria fresca sugli strati appena depositi dall'estrusore secondo delle specifiche situazioni degli strati, mentre il secondo nasceva dal fatto che l'elettronica di gestione del piatto riscaldato ci sembrava su Sanguinololu un po'

debole. La prima nuova funzione è molto utile perché consente di raffreddare più rapidamente il materiale depositato in modo da renderlo più viscoso rispetto alla deposizione di altro materiale successivo, limitando le irregolarità dovute alla deformazione pur minima degli strati ancora caldi, mentre la seconda è per chi vuole usare l'ABS e serve ad impedire la deformazione degli strati inferiori del pezzo stampato; questo materiale, a differenza del PLA, viene estruso ad una temperatura più elevata e, raffreddandosi fino alla temperatura ambiente, tende a contrarsi e a inarcarsi staccandosi dal piano d'appoggio, cosa che comporta una deformazione della parte bassa del pezzo e una crescita irregolare. Riscaldando il piano d'appoggio è possibile limitare notevolmente la contrazione termica durante tutta la fase di stampa ed evitare quindi inarcamento e distacco, ottenendo stampe eccellenti. Purtroppo con la scheda originale Sanguinololu, che pure dispone di un'uscita dedicata al controllo di un piatto riscaldante, potevamo controllare riscaldatori di





[schema elettrico dei **DRIVER**]



questa nuova piattaforma è stato il fatto che il firmware Marlin occupa praticamente tutta la Flash del ATmega 644 di Sanguinololu e che se avessimo desiderato aggiungere nuove funzionalità non saremmo riusciti a trovare spazio libero per scrivere le righe di codice richieste ad implementarle.

Nel caso in cui si volesse ad esempio gestire una SD-Card e un display per effettuare la stampa in modalità stand-alone (cioè caricare i file in G-Code su SD e gestirli localmente tramite pulsanti e display) ossia senza il PC, l'ATmega 644 andrebbe cambiato almeno con un ATmega 1284.

La nostra scheda è un mix di prodotti open source, cioè ha un'architettura Arduino Mega e condivide buona parte dell'hardware della soluzione RAMPS (RepRap Arduino Mega Pololu Shield). L'architettura Arduino permette anche a chi è poco esperto di programmare la scheda a piacimento, sfruttando la grande quantità di firmware e librerie che la comunità RepRap scambia e aggiorna quotidiana-

mente sul web. Il processore ATmega2560 dispone di ben 256 kB di memoria di programma e lavora con un clock di 16 MHz.

La nostra scheda può essere programmata direttamente dall'IDE Arduino e dispone allo scopo di una connessione USB per collegarla al computer tramite un normale cavo miniUSB; la stessa porta permette il controllo da PC durante le stampe.

A differenza di Sanguinololu questa scheda è stata realizzata completamente in SMD (tranne i connettori) il che ha permesso di aumentare la densità dei componenti a parità di dimensioni. Rispetto a Sanguinololu qui abbiamo, come accennato, l'uscita per il controllo di una ventola di raffreddamento, utilissima perché il software di stampa Repetier host e quello di slicing Slic3r permettono di gestire il raffreddamento del materiale estruso, andando ad attivare la ventola solo quando necessario e con una velocità adeguata, in base alle caratteristiche degli strati in fase di stampa.

Sulla scheda trovano anche spazio dei LED collegati ai driver dei motori passo-passo, cosa che ci permette di verificare immediatamente se i comandi vengono impartiti dal microcontrollore ai driver, quindi di effettuare un'efficace diagnosi del funzionamento del sistema in caso di anomalia. Anche ai MOSFET che pilotano il riscaldatore dell'estrusore e il piatto riscaldato sono abbinati LED di segnalazione, che hanno uguale funzione. Lo stesso è stato fatto per l'FT232RL che si occupa della conversione USB/seriale, il quale è stato provvisto dei LED che segnalano lo scambio di informazioni col PC.

Per quanto riguarda il controllo degli stepper-motor, la configurazione predefinita dei driver montati sulla scheda è 1/16 di step, ma tagliando le tre sottili piste sul lato inferiore che collegano a due a due le piazzole dei jumper, è possibile saldare la selezione manuale degli step mediante dei jumper a passo 2,54 mm; questa operazione può essere fatta per uno o più driver a seconda delle necessità.

Vediamo ora lo schema elettrico che descrive la nostra unità.

SCHEMA ELETTRICO

Il circuito, per quanto esteriormente appaia abbastanza piccolo, è un tantino complesso, dato che conta un microcontrollore a 100 pin, quattro moduli driver per motori passo-passo (abbiamo previsto la possibilità di pilotare un motore per ciascuno degli assi X, Y, Z ed un quarto motore per comandare l'ingranaggio che spinge il filo di materiale plastico dentro l'estrusore) più tre MOSFET che ci permettono di pilotare due riscaldatori (uno è quello del piatto e l'altro quello che scalda l'ugello dell'estrusore) ed una ventola a bassa tensione, oltre ad un conver-

ter USB/seriale per interfacciare l'ATmega con il computer e ad uno stadio di alimentazione, che completa l'insieme.

Sul microcontrollore abbiamo già detto; in esso è caricato il firmware "Marlin" opportunamente configurato per pilotare la nostra elettronica. Per la programmazione della scheda è possibile utilizzare un IDE Arduino 022.

I driver per i motori passo-passo sono i moduli prodotti dalla Futura Elettronica (cod. 3DDRIVER) e nello schema li trovate siglati U4, U5, U6, U7. Ciascun modulo è sostanzialmente un integrato A4988 della Allegro, molto versatile perché può essere impostato per definire sia la direzione di rotazione dell'albero del motore, sia il numero di gradi che il rotore del motore deve compiere alla ricezione di ogni comando; in altre parole, possiamo decidere se quando gli forniamo un impulso di comando, il modulo deve far ruotare l'albero di uno step alla volta, oppure di 1/2, 1/4, 1/8 o 1/16, in base all'accuratezza che si desidera ottenere. Come accennato, benché lo stampato sia predisposto per ospitare tre ponticelli per ciascun modulo (sono siglati MS1, MS2, MS3) in realtà abbiamo forzato il funzionamento ad 1/16 di step, così da ottenere la massima precisione di movimento; questa impostazione è ottenuta cortocircuitando i ponticelli, cosa che corrisponde a porre fissi ad 1 logico i contatti MS1, MS2, MS3 di tutti i moduli.

Se volete un'impostazione differente (ma state attenti che in questo caso dovete modificare il firmware per generare impulsi di comando a frequenza conseguente) dovete tagliare le piste sotto i jumper, almeno quelle che devono restare aperte.

Volendo, al posto del modulo Futura Elettronica si può utilizzare il Pololu MD09B, basato sull'integra-

Il modulo driver per stepper motor

Il controllo dei motori passo-passo è stato affidato a quattro moduli prodotti dalla Futura Elettronica e basati sull'integrato A4988 della Allegro e dotati ciascuno di un doppio driver a ponte H governato da una logica che controlla l'accensione e permette di far compiere al motore passi ridotti rispetto a quelli previsti dalla sua parte elettromeccanica; ad esempio, per ogni impulso di comando inviato a STEP (il contatto che riceve gli impulsi di azionamento del motore) lo stepper-motor può compiere un passo intero oppure frazioni da 1/2 a 1/16, a seconda della combinazione logica presentata ai piedini MS1, MS2, MS3, secondo la tabella riportata in questo riquadro. La direzione del movimento conseguente ad ogni impulso ricevuto sul contatto STEP dipende dal livello logico applicato all'ingresso DIR (1 logico forza il verso antiorario e 0 quello orario). Ogni ponte può erogare una corrente di 2 A, mentre il modulo si alimenta in continua con tensioni fino a 35 V; un piccolo trimmer posto sulla basetta e connesso al pin REF (17) dell'A4988, permette di regolare la corrente erogabile dai ponti (**Tabella 1**).

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase

Tabella 1 - Impostazione degli step compiuti dal motore rispetto agli impulsi di comando.

Il modulo permette di disattivare il ponte ad H pur rimanendo alimentato: allo scopo basta portarne la linea EN a livello logico alto; per il normale funzionamento, questo contatto deve essere posto a zero; notate che l'EN può anche non essere gestito, in quanto il modulo incorpora un resistore di pull-down (R6) che in mancanza di tensione di controllo fornita dall'esterno lo mantiene abilitato.

La linea di SLEEP serve invece per porre l'integrato U1 a riposo, pur senza privare il modulo dell'alimentazione; essa è mantenuta a livello logico alto dal resistore di pull-up R3, quindi in assenza di comando dall'esterno il modulo è sempre attivo. Ponendo SLP a zero logico l'integrato U1 si porta in modalità sleep, ossia in standby, assorbendo poche decine di microampere; in tale modalità non pilota il ponte ad H e non risponde ad alcun comando.

Il funzionamento degli stadi a ponte è governato da un segnale PWM generato da un circuito controllato da un timer one-shot, la cui durata dell'impulso di off determina il tempo di spegnimento di ogni MOSFET. In base alla tensione applicata ad Rsc sono possibili tre modalità di funzionamento:

- Vdd= il tempo di off viene impostato fisso a 30 μ s e il modo di decay è Mixed decay tranne quando si opta per la modalità 1/1 passo;
- massa (J1 chiuso)= il tempo di off-time è 30 μ s, mentre il modo di decay è Mixed decay sia per correnti in crescita che in calo, tranne che nel modo 1/1 passo;
- tensione intermedia= il tempo di off viene definito (in microsecondi) dalla formula $t_{off} \approx R_{sc}/825$.

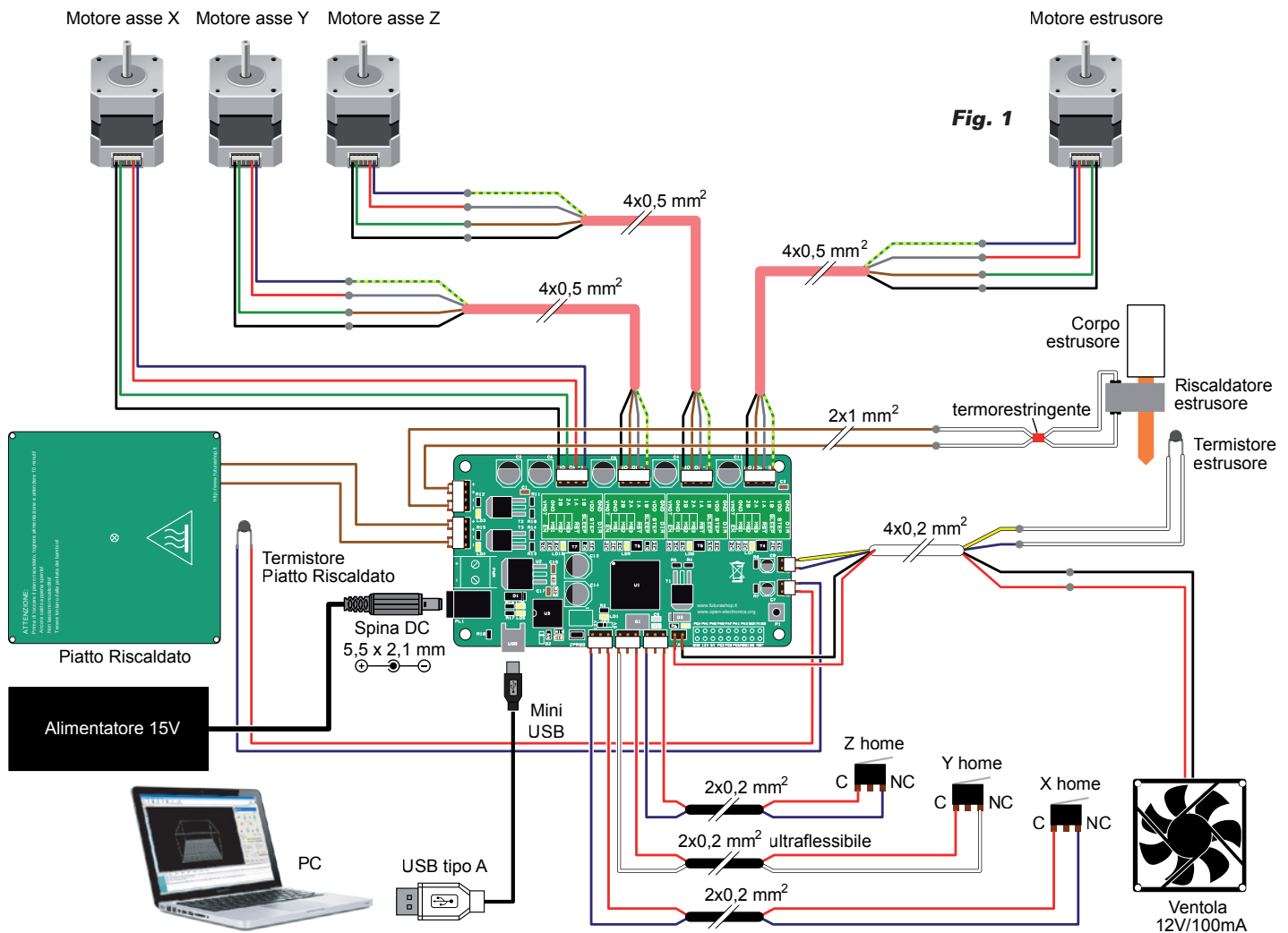
Queste modalità riguardano l'andamento degli impulsi di comando dei motori, il cui involuppo prevede un tempo di salita rapido e poi un decadimento (decay) misto (Mixed decay) nel caso la corrente erogata raggiunga il valore limite fissato dal trimmer applicato a Vref (Imax). Quando si verifica ciò, i MOSFET del ponte fanno decrescere la corrente erogata inizialmente (per il 32,5% del tempo toff) in modo rapido e poi lentamente.

La corrente erogata da ogni ponte può essere agevolmente regolata e limitata mediante l'applicazione del potenziale di controllo al piedino 17 dell'A4988; a ciò provvede il trimmer R9. La corrente massima erogata dal driver (Imax) è definita dalla formula:

$$I_{max} = V_{ref} / 8 \times R_s$$

dove Vref è la tensione applicata al piedino 17 dell'integrato ed Rs il valore della resistenza collegata al piedino di Sense (SENS1 per un ponte e SENS2 per l'altro). A riguardo diciamo che per avere la stessa limitazione di corrente su entrambi i ponti è opportuno avere R7 uguale ad R8.

Schema di collegamento elettrico della stampante 3Drag



to Allegro A4983; la differenza tra questo e l'A4988 è che gli manca un resistore di pull-down tra la linea che fa capo al contatto MS1 del modulo e massa. Ecco perché nel circuito trovate le resistenze R28, R29 R30, R31, che laddove i ponticelli MS1 vengano aperti provvedono al necessario pull-down. I driver dei motori passo-passo vengono gestiti dal microcontrollore U1, il quale ne imposta lo stato delle linee /STEP, EN e DIR; alla prima invia gli impulsi che determinano l'avanzamento di un passo alla volta nella rotazione dell'albero del motore, mentre DIR viene definito di volta in volta a seconda che si voglia ottenere la rotazione in senso orario o antiorario. EN serve per abilitare o disabilitare il

modulo. Dunque, per comandare la rotazione di un passo dell'albero del motore passo-passo, il microcontrollore pone EN a livello logico alto e lo mantiene in questa condizione, poi imposta DIR ad uno logico se la rotazione deve avvenire in senso antiorario o zero per quello orario (mantenendo anche questa condizione logica) e infine fornisce alla linea /STEP un impulso a livello basso; fatto ciò, riporta a riposo sia EN che DIR. Se l'avanzamento dev'essere di più step consecutivi, mentre EN e DIR sono attivi il micro invia a /STEP gli impulsi del caso. Naturalmente il motore collegato al driver si muove, ad ogni impulso che l'ATmega invia su /STEP, in base all'impostazione di MS1,

MS2, MS3; nel nostro caso, avendo impostato il massimo fattore di divisione, per compiere un passo completo occorrono 16 impulsi, quindi ipotizzando che il motore passo-passo sia a 360 passi, per far compiere al suo albero un giro completo il microcontrollore deve inviargli $360 \times 16 = 5.760$ impulsi. Se l'albero del motore deve compiere 1 giro al secondo, la frequenza degli impulsi di comando deve essere 5.760 Hz. Detto ciò, proseguiamo dicendo che ogni driver consta di un doppio ponte ad H governato da un'elettronica che permette di impostare il verso di rotazione del campo elettromagnetico e quindi dell'albero dello stepper-motor. Ogni volta che sul piedino STEP

giunge un impulso (la durata minima ammissibile è 1 μ s) a livello alto, a meno di diversa impostazione, le uscite 1A, 1B, 2A e 2B forniscono gli impulsi per comandare lo spostamento del rotore del motore di uno step o frazione, in base a come sono impostati, salvo diversa impostazione degli ingressi MS1, MS2, MS3.

Concludiamo la panoramica sui driver con il piedino SLEEP, il quale attiva la modalità di riposo (logica attiva e driver spenti) e con RESET, che azzerà il controllore che governa i driver e pone le uscite di comando dei motori (1A, 1B, 2A, 2B) a zero logico anche se STEP continua a ricevere impulsi. La logica è TTL-compatibile, comunque il modulo accetta sugli ingressi di controllo tensioni fino a 0,7 volte quella applicata tra il contatto 2 (Vcc) e massa (GND); il livello basso non deve superare 0,3xVcc. Notate che la linea /STEP di ciascun controller è stata collegata ad un transistor NPN, che serve come amplificatore di corrente per pilotare un LED, il quale pulserà analogamente alla corrispondente linea di comando del microcontrollore (PA4 per U4, PF6 per U5 PF0 per U6 e PL3 per U7) in modo da farci monitorare quello che accade. Così, nel caso un motore non giri malgrado il relativo LED pulsi, vuol dire che il problema è all'interno del driver o nel motore, ovvero nei cablaggi. Resta inteso che siccome il LED pulsa alla stessa frequenza degli impulsi di comando, potremo vederlo lampeggiare solo quando il motore corrispondente girerà a velocità molto ridotta, dato che già a 25 Hz l'occhio umano vede il diodo sempre illuminato.

Passiamo adesso ai driver a MOSFET, utilizzati per alimentare i riscaldatori e la ventola: sono tutti dei BUK6215-75C, prodotti da NXP, a canale N enhancement mode, capaci di erogare una

corrente di drain che raggiunge 57 A e sopportare una Vds in stato di interdizione, pari a 75 V; la loro bassissima Rds(on) (15 milliohm max a 15 A di Id) consente di minimizzare la potenza dissipata e perciò di saldarli direttamente ad una pista dello stampato (che funge da dissipatore). Il case è un SOT428, per montaggio superficiale, quindi permette questa soluzione.

Nel nostro caso i MOSFET vengono utilizzati a correnti molto ridotte, perché siamo nell'ordine dei 2÷2,5 ampere per il riscaldatore dell'estrusore (che va collegato all'uscita HEATER1) e dei 5÷6 A per il piatto riscaldato (da collegare all'HEATER2); ecco perché non c'è bisogno di dotarli di dissipatore e a raffreddarli basta l'effetto dissipativo delle piste cui la lamina di collettore è saldata.

I transistor vengono pilotati tutti con un segnale a impulsi, generato con la frequenza opportuna dall'ATmega, così da permettere una precisa regolazione della potenza ceduta agli utilizzatori, che si traduce in un controllo ottimale del calore generato dalle resistenze dei riscaldatori e della velocità della ventola del raffreddatore del pezzo. L'uso del PWM consente altresì una regolazione a bassissime perdite, dato che i MOSFET lavorano in modo on-off pur variando la tensione effettivamente applicata ai carichi che alimentano.

Ciascuna delle uscite riservate ai riscaldatori è provvista di un LED che pulsa insieme al relativo segnale PWM da circa 4 Hz, permettendo di verificare visivamente lo stato di funzionamento; il PWM della ventola di raffreddamento è invece a una frequenza più alta e il LED corrispondente sembrerà sempre acceso, quando la ventola è attiva, indipendentemente dalla velocità di rotazione.

Proseguiamo con l'analisi degli I/O del microcontrollore, analiz-

zando PE5, PJ1 e PD3, che servono a leggere lo stato degli switch finecorsa sui tre assi di spostamento; per l'esattezza, PE5 e PJ1 rilevano rispettivamente la fine della corsa del piatto (in orizzontale) avanti/indietro e lateralmente. Invece PD3 rileva il finecorsa del carrello che fa salire e scendere la barra orizzontale che sostiene e sposta verticalmente l'estrusore. Tutte le linee sono provviste di pull-up interno, attivato dal firmware.

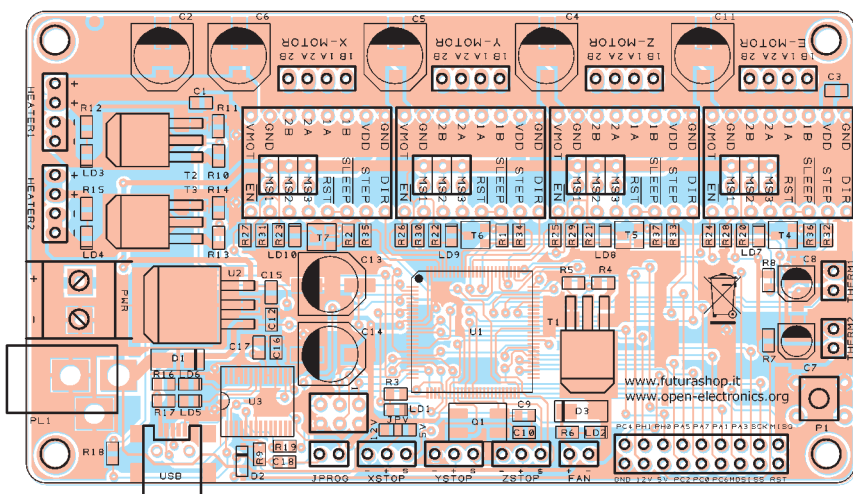
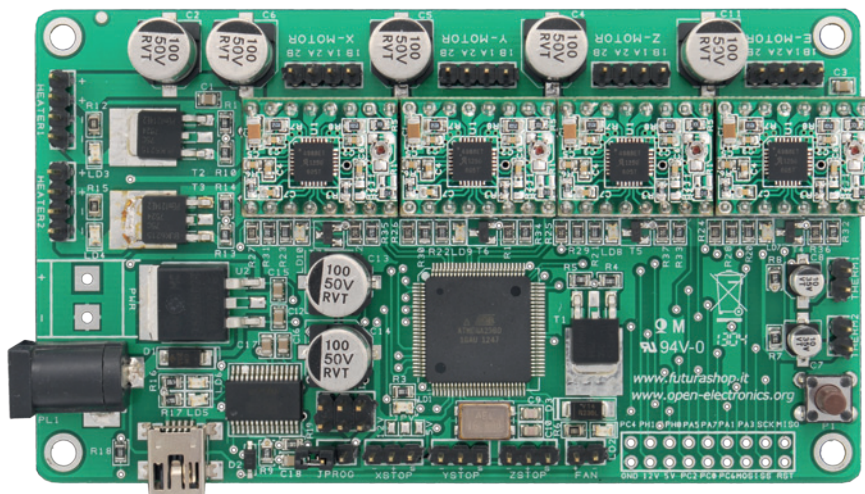
I finecorsa sono degli interruttori o commutatori che, opportunamente posizionati, vengono azionati dall'arrivo a fondo corsa del piatto o del sostegno della testa di stampa; nel nostro caso, vengono situati sul telaio.

Quando un finecorsa si aziona, il microcontrollore ne riceve informazione sotto forma di stato logico e provvede ad arrestare il movimento corrispondente e a far rientrare il carrello o rialzare la testa di stampa.

Come potete notare dallo schema elettrico, la nostra scheda dispone di tre ingressi per i finecorsa: XSTOP che corrisponde a PE5, YSTOP che coincide con PJ1 e ZSTOP che corrisponde a PD3. Per ciascun finecorsa sono stati previsti tre contatti: +, S e -. Questa soluzione è stata scelta per avere la possibilità di collegare sia finecorsa di tipo elettromeccanico (micro-switch e vari) sia rilevatori ottici, composti da un fotodiodo e un LED ad infrarossi affacciati tra loro in una cavità: quando il carrello arriva, una linguetta fissata ad esso passa nella cavità e blocca la luce che dovrebbe raggiungere il fotodiodo, determinando l'attivazione del finecorsa.

Questo genere di dispositivi esiste in vari modelli, che di solito hanno tre terminali: appunto uno per l'alimentazione positiva (da collegare a +) uno per quella negativa (da connettere a -) e l'altro per l'uscita

[piano di montaggio della **SCHEDA BASE**]



Elenco Componenti:

R1; R2: 10 kohm (0805)
 R3: 330 ohm (0805)
 R4, R10, R13: 10 ohm (0805)
 R5, R11, R14: 100 kohm (0805)
 R6: 1,8 kohm (0805)
 R7÷R9: 4,7 kohm (0805)
 R12, R15: 1,8 kohm (0805)
 R16, R17: 470 ohm (0805)
 R18, R19: 10 ohm (0805)
 R20÷R23: 330 ohm (0805)
 R24÷R27: 10 kohm (0805)
 R28÷R31: 100 kohm (0805)
 R32÷R35: 4,7 kohm (0805)
 R36, R37: 10 kohm (0805)
 C1, C3: 100 nF ceramico (0805)
 C2, C4÷C6: 100 µF 25 VL elettrolitico (E)
 C7, C8: 10 µF 35 VL elettrolitico (B)
 C9, C10: 22 pF ceramico (0805)
 C11, C13, C14: 100 µF 25 VL elettrolitico (E)
 C12, C15÷C18: 100 nF ceramico (0805)
 D1: GF1M
 D2: BAT42W

D3: MBRA140TRPBF
 T1÷T3: BUK6215-75C
 T4÷T7: BC817
 U1: ATMEGA2560-16AU
 U2: MC7805ABD2T (D2PAK)
 U3: FT232RL
 U4÷U7: Driver motori cod. MD09/FT1031
 LD1: LED verde (0805)
 LD2÷LD10: LED rosso (0805)
 P1: Microswitch
 Q1: Quarzo 16 MHz (C7S)

Varie:

- Plug alimentazione
- Morsetteria 2 poli
- Connettore mini-USB
- Strip maschio 2 poli (4 pz.)
- Strip maschio 3 poli (5 pz.)
- Strip maschio 4 poli (6 pz.)
- Strip femmina 32 poli (2 pz.)
- Jumper
- Circuito stampato

logica (da collegare ad S). Siccome esistono dispositivi alimentati a 5 volt ed altri funzionanti a 12 volt, abbiamo previsto la possibilità, mediante il jumper JPV, di portare ai punti + sia i 5 volt (chiudendo il ponticello su 5V) sia i 12 volt (chiudendo JPV su 12 V).

Utilizzando un finecorsa ad interruttore, possiamo collegarlo con il contatto tra S e - (massa) o tra S e +, fermo restando che occorrerà istruire il firmware sia per leggere lo stato logico alto o basso, sia per impostare il pull-up interno sui piedini 7, 64 e 46 del microcontrollore. Più esattamente, collegando l'interruttore finecorsa tra + ed S il firmware deve interpretare zero logico come stato normale e 1 come arrivo a fine corsa, mentre connettendolo tra - ed S, la condizione di arrivo a fine corsa coincide con il livello basso; in questa configurazione bisogna che gli I/O PE5, PJ1 e PD3 del microcontrollore abbiano attivata la resistenza di pull-up interna, altrimenti la lettura non sarà affidabile perché con l'interruttore aperto i piedini potranno leggere stati logici casuali dovuti ai disturbi captati dal circuito.

Resta inteso che utilizzando finecorsa il ponticello JPV può essere lasciato aperto. La configurazione di 3Drag, anche con la scheda Sanguinololu, è comunque con finecorsa normalmente chiusi per offrire protezione in caso di cablaggio difettoso o distacco del relativo connettore dalla scheda: in entrambi i casi verrebbe rilevato un finecorsa fittizio evitando il movimento del motore corrispondente e salvaguardando la meccanica da sforzi dovuti a finecorsa difettosi o scollegati.

Bene, passiamo adesso alla sezione di comunicazione con il computer, che fa capo alle linee PE0, PE1 ed RST del microcontrollore: alle prime due viene assegnato l'UART interno, che dialoga con

il convertitore USB/seriale siglato U3. Quest'ultimo è il canonico FT232RL della FTDI, e contiene la logica che occorre a trasformare i dati dal formato seriale TTL all'USB, smistando i dati in arrivo da TXD ed RXD sui piedini DP e DM. L'integrato viene alimentato direttamente dai 5 volt dell'USB del sistema cui la scheda viene connessa, quindi dal computer. I LED siglati LD5 ed LD6 indicano localmente l'attività delle linee di trasmissione e ricezione dei dati. Notate che l'integrato è provvisto di piedini per alcuni dei segnali di controllo dello standard RS232; di questi, usiamo il DTR (Data Terminal Ready) per far avviare il bootloader quando serve caricare il firmware nel microcontrollore. Questa prassi è ormai consolidata nello standard Arduino, in quanto una caratteristica dei microcontrollori dell'architettura Arduino è di essere programmati con un minimo di firmware, detto bootloader, che possiamo considerare come il BIOS del computer e che, opportunamente stimolato da un evento, avvia l'acquisizione dal computer dei dati che rappresentano il firmware e lo carica nella Flash di programma. Nel nostro caso, quando l'IDE invia l'apposito comando di programmazione, la linea DTR dell'FT232RL fornisce un impulso a livello logico basso, che attraversa il condensatore C18 e porta momentaneamente allo stato zero il piedino /RST dell'ATmega; il bootloader, che è il firmware di base che gira nel microcontrollore, riconosce questa condizione come la richiesta di comunicazione con il computer (via USB) per iniziare il download del firmware che dovrà poi essere eseguito normalmente, ossia quello che controllerà la stampante 3D.

Quando DTR torna a livello logico alto, ovvero il condensatore C18 si è caricato e smette l'impulso di

reset, lo stesso C18 si scarica attraverso il diodo Schottky D2. Notate che lo stesso effetto si può ottenere premendo localmente il pulsante di reset, esattamente come avviene nelle schede Arduino. Passiamo oltre e vediamo il connettore ICSP, che praticamente non ha bisogno di spiegazioni: serve a voi per fare la programmazione di base dell'ATmega2560, ovvero per caricargli il bootloader Arduino; questo connettore riporta l'alimentazione a 5 volt, nonché le linee MISO, MOSI ed SCK dell'SPI, nonché il RESET del microcontrollore. Le stesse linee le ritroviamo, insieme agli I/O dell'ATmega2560 rimasti inutilizzati, sul pin strip di espansione che abbiamo previsto per consentirvi di personalizzare la vostra scheda di controllo; ad esempio, sfruttando le linee del bus SPI (MISO, MOSI ed SCLK) potete collegare un lettore di SD-Card, mentre con sette degli I/O rimasti liberi potete collegare e controllare un display LCD standard da 2 righe per 20 caratteri, rendendo così la stampante capace di lavorare senza un computer collegato. Prossimamente vi illustreremo una soluzione con LCD, slot SD e tasti proprio per rendere autonoma 3Drag con questa nuova scheda di controllo.

Proseguiamo l'analisi del circuito con le linee PK5 e PK6, grazie alle quali il micro legge le sonde di temperatura che riportano il feedback sull'attività dei riscaldatori; queste sonde sono termistori NTC da 100 kohm a 25 °C da collegare ai contatti THERM1 e THERM2. Più esattamente, a THERM1 va collegato il sensore interno all'estrusore, mentre a THERM2 va collegato l'NTC che rileva la temperatura raggiunta dall'eventuale piatto riscaldato.

Le due sonde rientrano nel programma di gestione della temperatura e sono fondamentali per stabi-

lizzare quella raggiunta soprattutto dall'estrusore; infatti permettono al microcontrollore di regolare la corrente erogata alla resistenza che riscalda l'ugello in modo da mantenere sempre la temperatura impostata.

Senza questa retroazione non si potrebbe stabilizzare la temperatura, in quanto anche ammesso di lavorare a corrente costante e di supporre che a quella corrisponde una certa temperatura, le variazioni ambientali e operative potrebbero alterare la temperatura alla quale viene portato il materiale plastico da estrarre, con evidenti problemi di deposizione e di composizione dell'oggetto in stampa. Rilevando la temperatura dell'estrusore attraverso la tensione presente ai capi THERM1, il microcontrollore sa qual è la temperatura raggiunta e verifica il dato con quello imposto dal processo di stampa: se la temperatura eccede quella impostata riduce la corrente nel filamento riscaldatore, mentre in caso contrario la aumenta.

Analogo è il discorso per quanto riguarda il riscaldatore del piatto riscaldante.

Notate che per leggere le temperature colleghiamo i termistori a partitore resistivo con altrettanti resistori connessi al positivo di alimentazione, i cui valori sono stati opportunamente calcolati; avendo a che fare con dei termistori NTC, è chiaro che ad un aumento della temperatura rilevata corrisponde un calo di tensione e viceversa.

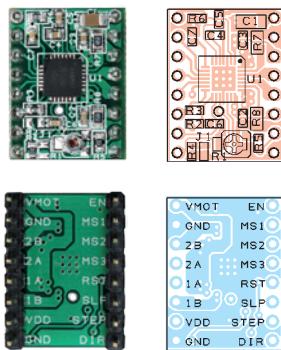
I termistori utilizzati nella nostra macchina per rilevare la temperatura dell'estrusore e del piatto riscaldato sono NTC da 100 kohm a 25 °C (Fig. 1).

REALIZZAZIONE PRATICA

La scheda viene fornita già montata e pronta per essere sostituita nella vostra 3Drag al posto di Sanguinololu; chi vuole può però scaricare

Elenco Componenti:

R1: Trimmer 10 kohm	ceramico (0603)
R2: -	C6: 220 nF
R3: 100 kohm (0603)	ceramico (0603)
R4: 10 kohm (0603)	C7: 220 nF
R5: 20 kohm (0603)	ceramico (0603)
R6: 100 kohm (0603)	U1: A4988SETTR-T
R7: 0,05 ohm (0805)	
R8: 0,05 ohm (0805)	Varie:
C1: 4,7 μ F ceramico (1206)	- Strip Maschio 8 poli passo 2.54mm (2 pz.)
C2÷C5: 100 nF	- Circuito stampato



tutti i file necessari dal nostro sito e realizzare autonomamente il proprio stampato su cui potrà montare tutti i vari componenti SMD. Ovviamente sono necessari degli strumenti e delle attrezzature specifici per il montaggio superficiale, soprattutto di componenti con pin ad elevata densità come il microcontrollore ATmega2560 che deve essere saldato usando gli strumenti opportuni.

Ottenuta o realizzata la scheda, si procede con l'inserimento dei moduli (che possono essere realizzati autonomamente o acquistati già pronti presso Futura Elettronica, www.futurashop.it) nei rispettivi zoccoli basati su connettori S.I.L. a passo 2,54 mm.

Per poter utilizzare la scheda bisogna caricare il bootloader (cioè il firmware di base) nell'ATmega 2560; la scheda acquistata da Futura Elettronica è già programmata, mentre se è stata realizzata autonomamente, va programmata direttamente sulla scheda.

Allo scopo dovete usare il pin-strip a 3+3 pin siglato ICSP ed un programmatore per ATmega, ma nulla vieta di caricare il bootloader usando come programmatore un Arduino, secondo quanto spiegato negli articoli che abbiamo dedicato, in passato, alla programmazione stand-alone (in particolare, sulla tecnica descritta nel fascicolo n° 159). In quest'ultimo caso l'ope-

razione è abbastanza semplice in quanto nell'IDE di Arduino trovate già la voce ATmega2560 e il rispettivo bootloader.

Ora potete montare la scheda a bordo della macchina, su un piano in materiale isolante fissato ad un montante, impiegando viti 3 MA e colonnine esagonali.

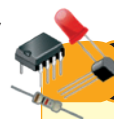
Quanto al cablaggio, ricordate di connettere i fili dei motori passo-passo ai connettori corrispondenti rispettando la sequenza dei colori indicata nello schema di cablaggio. La corrispondenza fra motori e connettori è la seguente: il motore dell'asse X va su X-MOTOR, quello dell'asse Y va collegato ad Y-MOTOR, quello che alza e abbassa la testa di stampa va su Z-MOTOR, mentre quello che fa avanzare il filo nell'estrusore va su E-MOTOR. Adesso dovete collegare i fincorsa: quelli degli assi X ed Y vanno rispettivamente ad XSTOP e YSTOP, mentre quello del carrello della testa di stampa va connesso a ZSTOP. Se adottate dei fincorsa ottici che richiedono alimentazione, chiudete il jumper JPV su 12V o 5V a seconda che i dispositivi richiedano 12 o 5 volt.

Il riscaldatore dell'estrusore deve essere collegato, senza badare alla polarità, ai contatti HEATER1, mentre l'NTC corrispondente va a THERM1.

Se nella 3Drag avete montato anche il piatto riscaldato, ne con-

netterete la serpentina riscaldante ed il sensore termico NTC rispettivamente a HEATER2 e THERM2. L'eventuale ventola raffreddatrice va collegata a + e - FAN, rispettando la polarità.

Fatto ciò, alimentate la scheda di controllo con un alimentatore dotato di plug adatto a quello montato (PL1): va bene un alimentatore di quelli AC/DC per portatili, che fornisca 15 volt c.c. ed una corrente dell'ordine dei 4 ampere se non usate il piatto riscaldato e di almeno 10 ampere se invece adottate il piatto. In alternativa potete utilizzare un alimentatore da rete modulare, collegandone i cavi di uscita positivo e negativo rispettivamente ai morsetti + e - PWR. Collegate la scheda al computer mediante un cavo USB A/B e avviate l'IDE Arduino per caricare il firmware che avete scelto. Fatto ciò, potete avviare la macchina con una stampa di test, per verificare se i motori si muovono correttamente e nella giusta direzione, sincronizzati ognuno con gli altri; in caso contrario invertite il cablaggio degli avvolgimenti di quello che gira alla rovescia.



per il MATERIALE

La scheda di controllo completa di quattro driver è disponibile già montata e programmata con firmware Marlin al prezzo di Euro 88,00 (cod. 3DCONTR-DRIVER).

La stessa scheda è disponibile anche senza driver (acquistabili separatamente a Euro 11,00 caduno, cod. 3DDRIVER) al prezzo di Euro 48,00 (cod. 3DCONTROLLER).

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)

Tel: 0331-799775 • Fax: 0331-792287
<http://www.futurashop.it>



Chi desidera realizzare sistemi "intelligenti" può utilizzare una delle varie schede oggi disponibili, a basso costo, nel mondo dei sistemi embedded. Ma quale scegliere?

SCHEDE DI SVILUPPO PER HOBBYISTI A CONFRONTO

dell'Ing. VINCENZO MENDOLA

Confrontiamo le ultime novità presenti nel mondo degli hobbysti comparandole con i dispositivi già presenti evidenziando i pro e i contro delle principali alternative disponibili sul mercato:

- Arduino Uno
- Arduino Due
- MBED
- Discovery

- Raspberry Pi
- Mini pc Android

I continui progressi tecnologici ci hanno abituati a disporre di un gran numero di dispositivi dotati di funzionalità sempre maggiori e capacità di interconnessione che crescono continuamente tanto che per ognuno di noi è normale avere la posta elettronica, l'agenda e i contatti sincronizzati

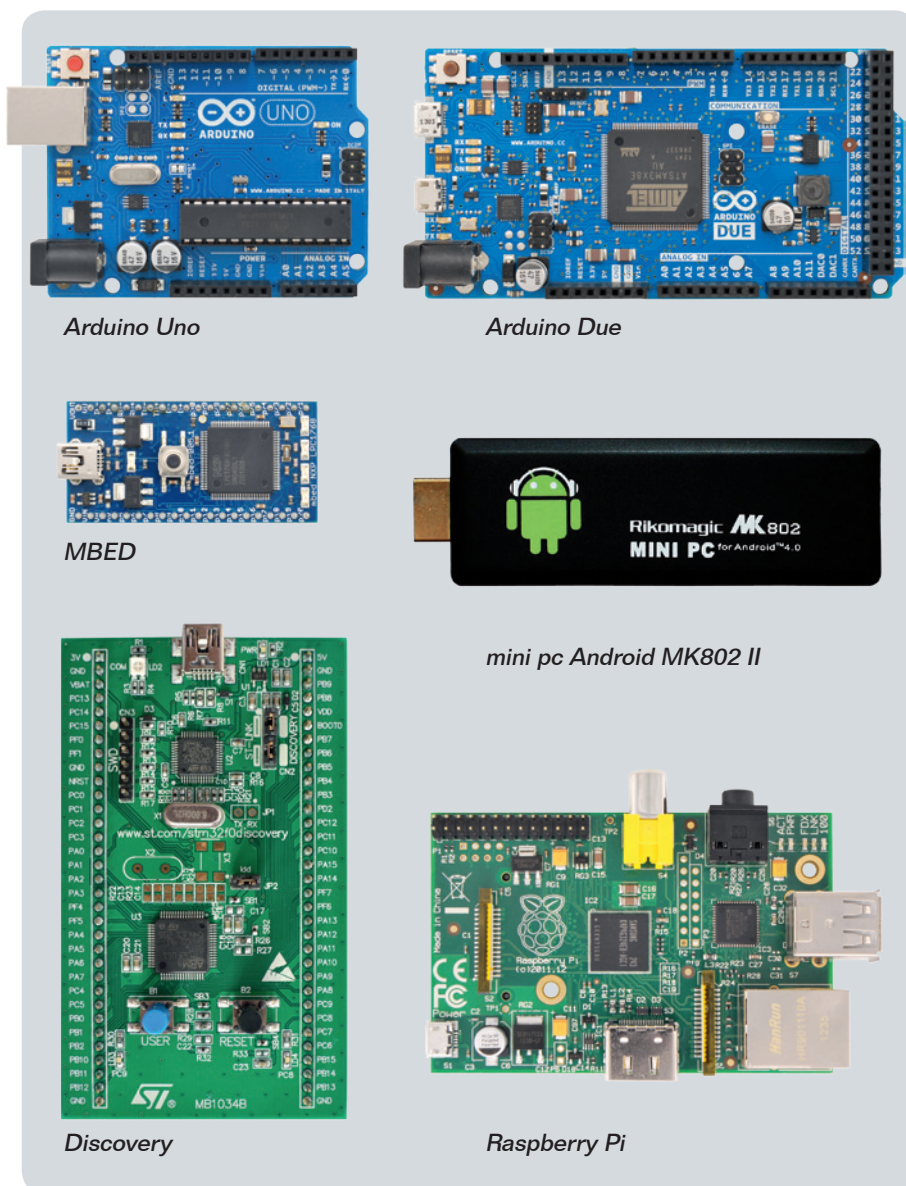


Fig. 1

Arduino Uno, Arduino Due, MBED, Discovery, Raspberry Pi e mini pc Android MK802 II. Tra quelli analizzati in questo articolo, il dispositivo di gran lunga più conosciuto ed utilizzato dagli hobbysti di tutto il mondo è sicuramente l'Arduino Uno che ha permesso di diffondere in maniera capillare tra gli appassionati la cultura dell'open hardware, permettendo anche a chi è alle prime armi nel mondo dell'elettronica e dei microcontrollori, di ottenere rapidamente risultati tangibili, ripetibili e personalizzabili, attraverso un processo di apprendimento graduale, progressivo e relativamente rapido, facilitato dalla grandissima mole di documentazione ufficiale, tutorial, esempi, libri e suggerimenti disponibili nei forum e nelle riviste di settore come la nostra. Questa semplice scheda con microprocessore ad 8 bit ha segnato una vera svolta nel mondo hobbistico, permettendo a chiunque dotato di semplice curiosità, di realizzare progetti di varia complessità; il successo dell'Uno è stato ed è talmente elevato e la diffusione tanto capillare e globale che in poco tempo gli utilizzatori più esperti sono stati in grado di realizzare applicazioni che hanno spinto al limite tale piattaforma.

LA GENESI DI ARDUINO DUE

La conseguente esigenza di disporre di un hardware più prestante in grado di soddisfare le accresciute esigenze progettuali e dal desiderio di essere presenti anche nel mondo dei microcontrollori a 32 bit dominati da altri tool di sviluppo disponibili da tempo, portando la stessa filosofia di immediatezza, semplicità e universalità, hanno spinto il team di svi-

tra i vari apparati: smartphone, tablet, desktop, notebook, netbook, lista a cui sempre più rapidamente si aggiungono nuovi dispositivi. Di questa continua evoluzione beneficia anche l'hobbista che può disporre di "gadget" tecnologici a prezzi competitivi e vantaggiosi, rendendo possibile in maniera relativamente semplice e rapida, la realizzazione di sistemi in ambiti applicativi fino a pochi anni fa appannaggio dei professionisti. I rapidi progressi della tecnologia hanno messo a disposizione dell'hobbista una notevole quantità di schede

di sviluppo con prestazioni impressionanti e fino a poco tempo fa impensabili a prezzi assolutamente accessibili, tanto da portare erroneamente gli appassionati a paragoni che spesso sono azzardati o perlomeno impropri. Per questo motivo abbiamo deciso di presentare una rapida carrellata di quanto attualmente facilmente accessibile all'hobbista medio analizzando nel dettaglio i pro e i contro delle varie architetture.

I MAGNIFICI SEI

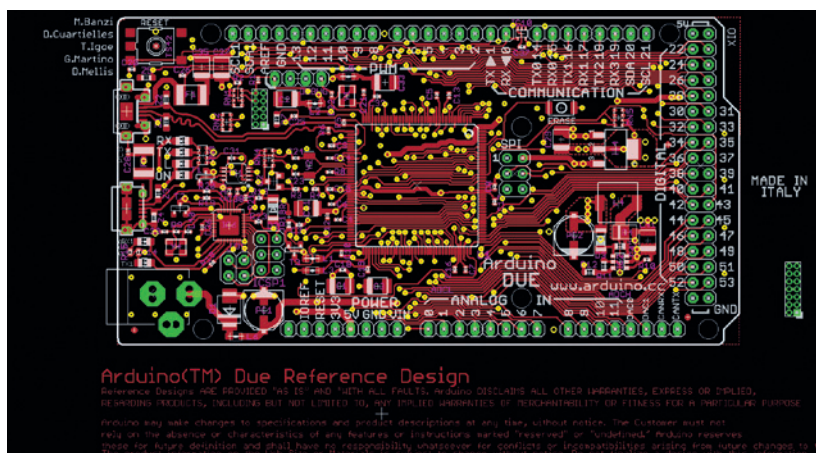
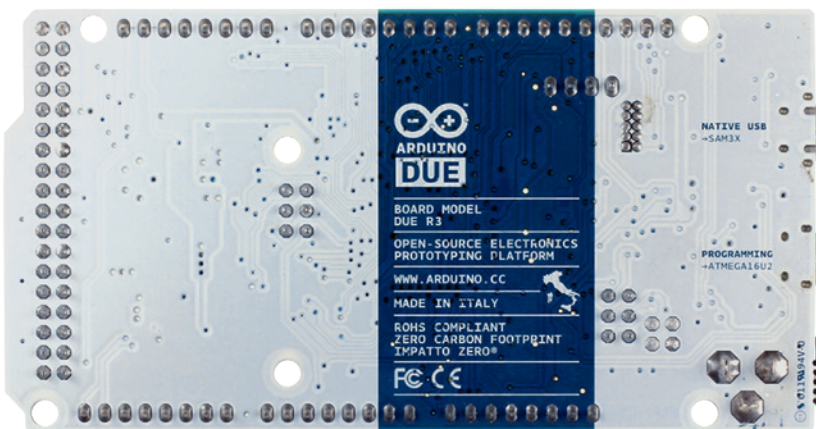
Nella Fig. 1 riportiamo i dispositivi che vogliamo confrontare:

Fig. 2

luppo di Arduino a realizzare il Due, la prima board della famiglia ad utilizzare un potente microcontrollore ARM Cortex M3. Questa scelta ha obbligato lo stesso team a realizzare una nuova versione dell'IDE che è stata chiamata 1.5 per distinguersi totalmente dalla precedente 1.0 in quanto questa deve contenere oltre al compilatore per gli AVR, anche il compilatore per gli ARM. Il tentativo di mantenere la retrocompatibilità con le precedenti board ad 8 bit ha creato non pochi problemi agli sviluppatori che hanno dovuto necessariamente scendere a compromessi per ottenere il miglior equilibrio tra la portabilità del codice già realizzato e lo sfruttamento delle maggiori potenzialità della più complessa architettura.

MBED CON CORTEX

Molti non sanno che è disponibile una valida alternativa all'Arduino Due da svariati anni: l'MBED, disponibile in due versioni una con Cortex M0 e l'altra con Cortex M3, entrambi della NXP. Il modello paragonabile al DUE utilizza un LPC1768 [1]. Facendo una comparazione diretta dei costi, apparentemente il DUE sembrerebbe economicamente notevolmente più vantaggioso rispetto ad MBED ma non è così: se guardiamo in dettaglio entrambi gli schemi elettrici, notiamo subito che le due schede utilizzano un Cortex M3 dotato di un Ethernet MAC (livello 2 "datalink" della pila ISO/OSI) compatibile con lo standard IEEE 802.3. A differenza del DUE, l'MBED è stato realizzato in modo da permettere la connessione del dispositivo alle reti LAN e WAN in maniera semplice; a tale scopo,

**Fig. 3**

nel pcb è stato aggiunto l'integrato DP83848J della National Semiconductor (ora acquisita dalla Texas Instrument) che si occupa di garantire l'accesso fisico (livello 1 della pila OSI/ISO) alla rete. Com'è possibile rendersi conto facilmente guardando i file Eaglecad ufficiali dell'ultimo nato nella famiglia Arduino [2], durante la realizzazione della scheda è stata già presa in considerazione questa "opportunità" di connettività aggiuntiva [3] non possibile nativamente con il DUE, infatti si vedono le connessioni ad un connettore "ETH" che fa capo come già visto all'ethernet MAC presente nel SAM3X [4]; tale connettore è anche chiaramente visibile sul lato destro del pcb, all'esterno del circuito stampato (Fig. 2). È quindi chiaro che la possibilità di

inserire un' interfaccia ethernet è già stata presa in considerazione dal team di sviluppo e molto probabilmente assisteremo a successive release della board. È probabile che l'ethernet non sia stata ancora sviluppata in quanto non basta l'implementazione hardware per avere un'interfaccia funzionante: lo sviluppo delle relative librerie avrebbe richiesto altro tempo che avrebbe ritardato ulteriormente l'uscita del DUE, la cui distribuzione è iniziata solo 14 mesi dopo il suo annuncio ufficiale ed è già alla revisione 3 pur essendo appena entrata in commercio (Fig. 3). Ovviamente queste sono solo ipotesi che, per quanto verosimili, rimangono comunque congetture dedotte dall'analisi dei file cad disponibili.



**RaspberryPi con Processore Broadcom BCM2835 700 MHz
ARM1176JZF5 con FPU e GPU Videocore 4.**

QUESTIONE DI VOLTAGGI

Un'importante differenza applicativa tra l'MBED e il Due è che mentre il primo è 5V tolerant, il secondo non lo è, pur essendo entrambi alimentati a 3.3V e questo richiede un'ulteriore maggiore attenzione nell'uso dell'arduino rispetto ad MBED per evitare irreparabili danneggiamenti. Un'altra accortezza di cui tenere conto durante il primo utilizzo del nuovo Arduino con Windows 7 è la necessità di caricare i driver seguendo il percorso "Pannello di control-

lo", "Sistema", "Gestione dispositivi", "Porte (COM e LPT)", "Arduino Due programming port". A questo punto si deve selezionare "Aggiornamento software driver" e "cerca il software del driver nel computer" indicando il percorso della cartella "Drivers" all'interno dell'IDE 1.5. L'MBED invece viene riconosciuto dal sistema operativo senza alcuna necessità di installare alcun driver. Se consideriamo la necessità di utilizzare un traslatore di livello per garantire completa compatibilità con gli shield realizzati per l'Uno e la mancanza di un'interfaccia ethernet, qualora volessimo colmare queste differenze, il costo del DUE diventerebbe in realtà maggiore di MBED. Proseguendo con la comparazione dei costi, il DUE, oltre a non avere l'interfaccia ethernet, può essere ancora considerato dal punto di vista

dell'IDE e delle relative librerie, in una fase non completamente matura. Sicuramente il desiderio di mantenere il più possibile la compatibilità con il codice e le librerie scritte per l'UNO, rende l'operazione di sviluppo lato software del DUE piuttosto complessa dato che le due schede sono in realtà molto più diverse di quanto si potrebbe dedurre da una semplice rapida occhiata ai due circuiti stampati.

KIT DI SVILUPPO

Le aziende produttrici di semiconduttori hanno cominciato negli ultimi anni a non trascurare più gli hobbysti che sempre più spesso possono trasformarsi in futuri progettisti professionisti e di conseguenza potenziali clienti per cui quasi tutti i principali marchi hanno reso disponibile un proprio tool di sviluppo ingegnerizzato con l'obiettivo di essere utilizzato il più velocemente possibile in maniera semplice ed immediata. Anche la ST non è rimasta a guardare ed ha realizzato una propria famiglia di schede di sviluppo e prototipazione praticamente per ogni tipologia di microcontrollore ARM che commercializza. In questo articolo prendiamo in considerazione la Discovery F0 [5] con ST32 che utilizza un microcontrollore Cortex M0 dalle prestazioni comparabili a quelle di MBED con NXP 11U24 [6]. Una peculiarità di questo dispositivo che salta subito all'occhio è il prezzo veramente basso a cui è venduta: circa 8 euro comprese le tasse. Se si prendesse solo in considerazione il prezzo ci si chiederebbe allora perché spendere 65 euro per l'MBED o 47 per l'Arduino DUE se è comunque possibile disporre della potenza dei 32

Listato 1

```
uint16_t G_ADCValue=0;
float G_ReadVal =0;
char G_SendBuff[64];

void setup(void){
  uint16_t i = 0;
  char L_readChar;
  Serial.begin(115200); }

void loop(){
  delay(2000);
  while(1){

    delay(2000);
    G_ADCValue = analogRead(A2);
    G_ReadVal = (float)(POWER_SUPPLY * G_ADCValue) / 4095;

    sprintf(G_SendBuff,"%f Volt",G_ReadVal);
    Serial.print("ADC read: ");
    Serial.println(G_SendBuff);
    i++;
    if(i>10) i=0;
    while(Serial.available()) {
      L_readChar = Serial.read();
      Serial.write(L_readChar);
    }
    Serial.println("");
  }
}
```


*MBED è una scheda basata sul microcontrollore a 32-bit
ARM® Cortex™ NXP LPC1768 funzionante a 96 MHz.*

bit del F0 ad un prezzo notevolmente inferiore. Innanzitutto per i Discovery non esiste un IDE dedicato come invece si ha per il Due e l'MBED, ST fornisce solo un elenco di toolchain di sviluppo consigliati per la programmazione:

IAR™ Embedded Workbench for ARM (EWARM) available from www.iar.com;
RealView Microcontroller Development Kit (MDK-ARM) toolchain available from www.keil.com;
Atollic TrueSTUDIO® STM32 available from www.atollic.com;
Altium™ TASKING VX-toolset for ARM® Cortex-M available from www.tasking.com.

Già questo è segno del fatto che il prodotto è stato sviluppato per un target di utilizzo che coinvolge solo parzialmente il mondo degli hobbysti, anche se nei firmware d'esempio realizzati dalla stessa ST sono presenti le librerie per il porting del codice di Arduino. I tool software elencati non sono gratuiti per cui è necessario utilizzare le versioni limitate o acquistare la relativa licenza che può costare anche varie migliaia di euro. Se si considera questo aspetto è chiaro che il costo apparentemente superiore di MBED o del DUE è compensato dalla disponibilità di un IDE dedicato, sviluppato ed ottimizzato per lo specifico hardware con cui verrà utilizzato e ciò rende la programmazione e lo sviluppo del codice più semplice rispetto al caso del Discovery.

La forma ed il passo del Discovery non agevolano l'utilizzo di breadboard e nel caso si voglia utilizzare questo sistema, sarà necessario utilizzarne due

unità distinte, ognuna connessa ad un lato del Discovery. Analizziamo ulteriormente le caratteristiche hardware: sotto questo aspetto sono sicuramente le dimensioni a saltare all'occhio: il dispositivo realizzato dalla ST è circa il triplo di un MBED. Com'è facile intuire, la miniaturizzazione ha un costo: la maggior complessità dei circuiti stampati legata ad un numero superiore di strati comporta un conseguente aumento dei costi realizzativi; anche l'ingegnerizzazione generalmente richiede più tempo per ottimizzare le dimensioni finali senza compromettere le prestazioni e ciò contribuisce ad incrementare ulteriormente il prezzo finale del prodotto finito.

CODICE A CONFRONTO

Consideriamo ora un semplice codice di esempio per l'acqui-



sizione e la visualizzazione su seriale di una temperatura misurata mediante un trasduttore analogico come potrebbe essere il MCP9700A [7], utilizzando le librerie sviluppate per rendere compatibile il codice di Arduino con la board di ST. Il codice del Discovery sarà come riportato nel **Listato 1**. Mentre per Arduino le istruzioni che svolgono un compito analogo possono essere quelle riportate nel **Listato 2**.

Come possiamo vedere la struttura ed i comandi sono simili a quelli a cui siamo abituati con

Listato 2

```
#define TEMP 2
float val = 0.0;
float T= 0.0;
float VADC= 5;
double STAMPA_T = 0;

void setup() {
  Serial.begin(9600);
}

void loop() {

  STAMPA_T= (temp());
  Serial.print(STAMPA_T);
}

float temp() {
  double nread = 10.0;
  double somma = 0.0;
  for (int i=0; i<nread; i++)
  {
    val = analogRead(TEMP);
    T = (((VADC/1024.0*val)-0.5) * 100);
    somma += T;
  }
  delay(100);
  return (somma/nread);
}
```



All'interno del mini PC MK802II si trova un SoC Alwinner A10 che contiene un Cortex A8 a 1 GHz e una GPU Mali-400 capace anche di gestire il video 3D.

Arduino ma non sono esattamente uguali. Questo ci fa subito capire che per utilizzare il codice scritto per Arduino con il Discovery è necessario fare prima delle modifiche che, se pur piccole, sono indispensabili per la corretta compilazione ed esecuzione. È chiaro ancora una volta che una scheda come questa non è adatta al neofita alle prime armi, che potrebbe incontrare difficoltà non indifferenti per compilare il proprio codice in caso di errore.

PENSATO PER LE SCUOLE?

Un prodotto che sta riscuotendo un meritato successo e che stiamo già presentando sulle pagine della nostra rivista è il Raspberry Pi, ora ulteriormente migliorato con il raddoppio della memoria RAM che è stata incrementata da 256MB a 512MB mantenendo inalterato l'interessantissimo prezzo di poco più di 32 euro per il modello B, il più completo dei due disponibili. Il "cuore" della scheda è costituito da un SOC Broadcom BCM2835 che contiene al suo interno sia un ARM 11 con clock a 700 MHz sia una GPU in grado di riprodurre filmati con qualità comparabile a quella dei supporti BluRay. Le capacità di calcolo sono para-

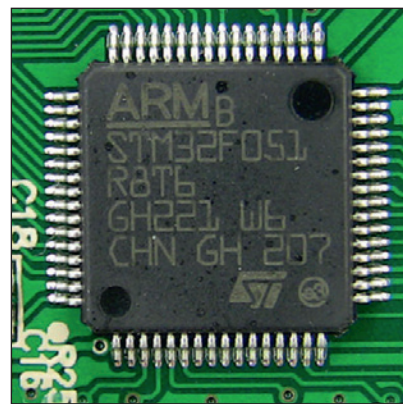
gonabili a quelle di un vecchio Pentium III, senza considerare le prestazioni della GPU, di gran lunga superiori ad una scheda video media disponibile negli stessi anni in cui il Pentium era lo stato dell'arte del mercato; i consumi, invece, sono notevolmente inferiori, diciamo di circa 10 volte (5W contro 50W di un pc completo con il PIII). Sulla board sono presenti un classico connettore audio femmina per jack a 3.5mm, un connettore RCA per la connessione analogica ai vecchi televisori ed anche una più moderna interfaccia HDMI per il trasferimento audio/video mediante connessione digitale ai nuovi monitor e televisori, universalmente dotati di almeno una di queste porte. È disponibile un connettore RJ45 per interfacciarsi alla rete ed è presente un doppio connettore USB tipo A per consentire il collegamento, ad esempio, di mouse e tastiera. Il sistema operativo utilizzato è LINUX, disponibile in varie distribuzioni appositamente ottimizzate per l'hardware del Raspberry Pi [8]; il boot viene effettuato tramite scheda SD, inserita nell'apposito slot presente sul lato inferiore della board, vicino al connettore mini USB disponibile esclusivamente per l'alimentazione. Dopo l'installazione delle distribuzioni LINUX modificate ad hoc per il nostro "lamponi", potrebbe essere necessario formattare la scheda SD qualora si decidesse di destinarla ad altri usi. In ambiente Windows, in caso di problemi nel ripristinare la corretta capacità della scheda di memoria flash, potrebbe essere d'aiuto un tool come questo, scaricabile dopo aver accettato la licenza d'uso:

https://www.sdcard.org/downloads/formatter_3/eula_windows/

L'utilizzo è estremamente semplice, ad ogni modo è disponibile un pdf con le istruzioni di utilizzo (in inglese):

https://www.sdcard.org/downloads/formatter_3/SDFormatter_3.1e.pdf

Le dimensioni della scheda sono simili a quelle dell'Arduino Uno (circa il 30% più grande della scheda ad 8 bit) ed anche il prezzo è comparabile anche se l'ambito di utilizzo è notevolmente diverso: il Raspberry Pi nasce come scheda per permettere un facile accesso agli strumenti informatici a costi accessibili a tutti con l'obiettivo di garantire l'informatizzazione anche dei più piccoli; l'Arduino è stato invece concepito come strumento hardware per il quale il codice è solo un mezzo per realizzare in maniera semplice e rapida le proprie idee in ambito elettronico. Di conseguenza i numeri che riguardano la produzione dei due prodotti sono notevolmente differenti: il Raspberry Pi è prodotto in larga scala e tuttavia la richiesta è ancora talmente alta che al momento non è facile riuscire ad acquistarne un'unità; l'Arduino è prodotto in quantità inferiori e questo determina un costo finale maggiore rispetto al



STM32F0DISCOVERYDiscovery con STM32F051 MCU.

Arduino Due è la prima scheda basata su un microcontrollore di tipo ARM a 32bit, il SAM3X8E ARM Cortex-M3 di Atmel.

primo in quanto i prezzi della componentistica diminuiscono sensibilmente all'aumentare dei quantitativi acquistati, inoltre nel primo caso i costi di ingegnerizzazione sono distribuiti su di un numero maggiore di unità abbassandone l'incidenza sul singolo pezzo.

Per questo motivo non è sensato paragonare brutalmente le prestazioni ed il prezzo di un Raspberry Pi con quelli di un Uno dato che già in fase di sviluppo del prodotto le scelte iniziali sono state profondamente differenti e conseguentemente i costi ed il target di potenziali acquirenti. L'attrattiva della grande potenza di calcolo e il gran numero di interfacce del Pi rispetto all'Arduino unite alla versatilità di LINUX, ha messo in moto la grande creatività degli appassionati tanto che sono già disponibili schede di bridging per utilizzare gli shield della scheda made in Italy con Raspberry Pi, come ad esempio i seguenti:

<http://www.cooking-hacks.com/index.php/documentation/tutorials/raspberry-pi-to-arduino-shields-connection-bridge>

<http://baldwisdom.com/introducing-raspberry-pi-a-la-mode/>.

CHIAVETTA O PC?

I lettori assidui della nostra rivista hanno già avuto modo di fare conoscenza con un oggetto che possiamo considerare un vero e proprio gioiellino tecnologico: il mini pc android mk802II, presentato sul numero 170 di ottobre di Elettronica In, progettato sfruttando al meglio le potenzialità del System On a Chip Allwinner A10, un Cortex A8 ad 1 Ghz con 1 GB di memoria RAM e 4GB di flash, in grado di connettersi alle reti WiFi 802.11 b/g/n ed alle

rete cablate mediante un modulo adattatore USB/ethernet. Grazie al Video Processing Unit di cui è dotato, l'MK802 è in grado di riprodurre in maniera fluida anche i filmati HD. Come già visto in dettaglio sull'articolo di presentazione, tale dispositivo è un minipc con un chipset che permette di realizzare un apparato di buone prestazioni che viene utilizzato da alcuni produttori di tablet per i loro apparati. La versione Android 4.0.4 (Ice Cream Sandwich) preinstallata permette di servirsi del Google play per aggiungere le applicazioni desiderate, rendendo l'apparato molto versatile e semplice da utilizzare. Per chi non si accontentasse delle funzionalità offerte dagli applicativi disponibili per Android e volesse sfruttare ulteriormente le potenzialità dell'hardware o fosse desideroso di sperimentare e testare i limiti e le capacità offerte dall'hardware, è possibile utilizzare il minipc con LINUX, semplicemente utilizzando una microSD da cui il sistema effettua il boot all'accensione del dispositivo. È possibile trovare delle distribuzioni LINUX per dispositivi ARM ed ottimizzate per l'MK802II qui:

<http://dl.miniand.com/allwinner10/fedora/fedora-17-1-xfce-720p-miniand.com.img.7z> (Fedora 17 XFCE, dettagli nella pagina <https://www.miniand.com/forums/>



forums/2/topics/42;

<http://store.cloudsto.com/downloads/lubuntu-1204-rikomagicuk.7z> (Lubuntu 12.04 LXDE, dettagli nella pagina <http://rikomagic.co.uk/forum/viewtopic.php?f=2&t=90>).

I test con LINUX possono essere tranquillamente eseguiti senza andare a modificare minimamente il firmware del minipc attualmente installato, in quanto per effettuare la sperimentazione si fa uso di una microSD opportunamente configurata; ad ogni modo, in caso di problemi, è sempre possibile reinstallare il firmware preinstallato sull'MK802 servendosi del tool, delle istruzioni e dell'immagine presenti sul sito ufficiale del nostro Rikomagic, selezionando la versione hardware corretta, cioè la MK802II: <http://www.rikomagic.com/download.asp>. Per installare l'immagine desiderata su una microSD da 8 GB si può utilizzare un programma come Image Writer o simile se si utilizza windows o più semplicemente su LINUX servendosi del

RIFERIMENTI

- [1] <http://mbed.org/handbook/mbed-NXP-LPC1768>
- [2] <http://arduino.cc/en/uploads/Main/arduino-Due-Reference-design.zip>
- [3] <http://www.hwmakers.eu/arduino/arduino-due-the-missing-ethernet/>
- [4] <http://www.atmel.com/Images/doc11153.pdf>
- [5] <http://www.st.com/internet/evalboard/product/253215.jsp>
- [6] <http://mbed.org/handbook/mbed-NXP-LPC1114>
- [7] <http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf>
- [8] <http://www.raspberrypi.org/downloads>

mini pc Android
MK802 IIIS.



comando `"dd if=lubuntu-1204-rikomagicuk.img of=/dev/"` che permette di copiare a blocchi il file sorgente nella destinazione richiesta; nel campo `"of=/dev/"` andrà aggiunto il corrispondente valore relativo alla propria memoria flash, identificabile con il comando `"sudo fdisk -l"`. Per ulteriori dettagli si può fare riferimento alla pagina del forum precedentemente indicata. Entrambe le distribuzioni si avviano senza problemi e in maniera sufficientemente rapida, con tempi paragonabili a quelli richiesti da android per avviarsi sullo stesso dispositivo. Dai test da noi effettuati Fedora ci è sembrata più stabile, matura e completa; entrambe comunque, possono essere ulteriormente migliorate cosa che siamo sicuri non mancherà di essere fatta dato il grande interesse e il successo che questi dispositivi e quelli ARM-based in genere stanno suscitando. Dopo l'installazione delle distribuzioni LINUX modificate ad hoc per il nostro minipc, è necessario formattare la microSD qualora si decidesse di destinarla ad altri usi. Come già visto per Raspberry Pi, per ripristinare la corretta capacità della scheda di memoria flash in maniera rapida e senza problemi in ambiente windows si può utilizzare il medesimo tool:

https://www.sdcard.org/downloads/formatter_3/eula_windows/

Come indicato sul numero 170, al fine di evitare comportamenti imprevisti e malfunzionamenti

dell'apparato, è necessario utilizzare un alimentatore in grado di erogare almeno 1,5 A (ad esempio il modello cod. 8220-PS-SEUSB11 della Futura Elettronica, dal costo di 5 euro) fornendo così una potenza sufficiente a garantire un funzionamento stabile del Rikomagic; per la configurazione e l'utilizzo va aggiunta una tastiera che potrebbe integrare le funzionalità di un mouse come ad esempio il versatissimo modello wireless a 2.4 GHz Rii mini i8 distribuito dalla Futura Elettronica (cod. 8140-MINI8, 38 euro), che è dotata di batteria integrata al litio, simile nel formato e nella capacità a quella utilizzata nei cellulari, ricaricabile tramite cavetto USB-mini USB direttamente all'interno della Rii, senza essere rimossa. La i8 può essere utilizzata anche (e non solo) con pc windows e LINUX e persino con il Raspberry Pi, senza la necessità di caricare manualmente driver o eseguire operazioni di configurazione da parte dell'utilizzatore per la corretta identificazione da parte del sistema operativo. Se le dimensioni non sono un problema è anche possibile utilizzare una tastiera e un mouse wireless tradizionali, come ad esempio il set Logitech mk320 da noi testato.

Quindi agli 82 euro del prezzo del minipc è necessario aggiungere i 5 euro dell'alimentatore switching e i 38 della tastiera, per un totale di 122 euro, prezzo di gran lunga superiore a tutti gli altri prodotti presi in considerazione in questa analisi ma più che giustificato dalle prestazioni

e dalle caratteristiche del prodotto che più che nella categoria hobbistica rientra in quella consumer.

TUTTO SI EVOLVE

La velocità con cui i prodotti elettronici si evolvono è veramente impressionante: la Rikomagic ha già realizzato una versione evoluta con il nome eloquente di MK802IIIS, in grado di fornire prestazioni ancora più elevate grazie al nuovo processore dual core Rockchip RK3066 Cortex A9 a 1,6 GHz, 1 GB di memoria RAM e ben 8 GB di memoria FLASH e dotato di connessione Bluetooth. Il nuovo modello è disponibile presso la Futura Elettronica allo stesso prezzo del precedente. Al momento non siamo a conoscenza di una versione stabile di LINUX specifica per questo hardware anche se ci sono alcuni progetti in corso ancora in fase pre-alfa; ad ogni modo siamo sicuri che anche in questo caso il grande interesse che ruota attorno a questi gioiellini tecnologici e i vantaggi derivanti dalle maggiori prestazioni disponibili con la versione dual core, non mancheranno di attirare l'interesse degli sviluppatori che sicuramente molto presto renderanno disponibile anche il codice per far girare LINUX anche sulla versione III. I prodotti che sarebbero potuti rientrare in questo articolo sono innumerevoli per cui abbiamo dovuto fare delle scelte, anche per via della rapidità con cui ogni settimana nuovi dispositivi vengono messi in commercio. Solo per questo motivo non abbiamo inserito schede come la "Beagleboard" (<http://beagleboard.org/>) o la "Aria" (<http://www.acmesystems.it/aria>) o la "IOIO" (<https://github.com/ytai/ioio/wiki>) per citarne alcune. ■

The collage consists of 18 individual images arranged in a grid-like fashion, showcasing various digital clock prototypes and designs. The designs are diverse, ranging from simple digital displays to more complex, interactive, or artistic interpretations. Some use traditional digital fonts, while others use dot-matrix LEDs, analog-style hands, or even game-like interfaces (like Pong). The materials used for the clock faces include cardboard, wood, plastic, and metal. Some are mounted on walls, while others are portable or integrated into other objects. The colors of the displays vary, with red, blue, and white being prominent. The overall theme is creative and innovative digital timekeeping.

di ANDREW O'MALLEY
e SIMONE MAJOCCHI



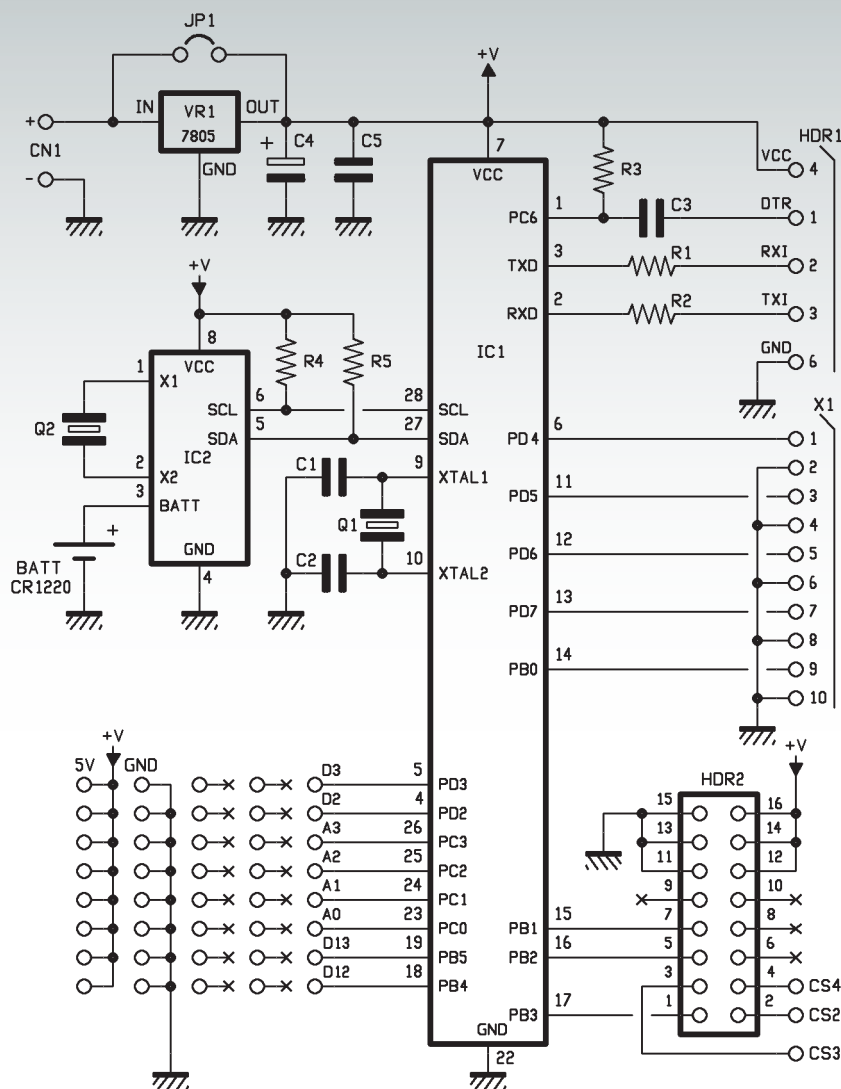
Qualche tempo fa stavamo cercando in rete delle idee su orologi strani e un po' sfiziosi quando abbiamo trovato il sito del designer Sander Mulder (www.sandermulder.com). Alcuni orologi richiedevano sofisticati sistemi di taglio e formatura, altri erano complessi sistemi a ingranaggi, ma fra questi spiccava una specie di cornice con schermo LCD incorporato da circa una dozzina di pollici: a prima vista sembrava la riesumazione del gioco PONG, quello che nel 1972 con Atari ha di fatto dato il via al mondo dei videogiochi. Guardando meglio abbiamo scoperto che l'orologio era una infinita partita di Pong in cui un giocatore segnava ogni minuto e l'altro ogni ora... e il punteggio era la visualizzazione di ore e minuti correnti. Decisamente una bella idea che ci ha portato a cercare subito "Pong Clock" su Google, ottenendo una serie di risultati abbastanza varia. Fra questi abbiamo trovato in numerose immagini quello che poi avremmo scoperto

essere il nostro Dotklok.

L'IDEA DI ANDREW

Rispetto ai vari orologi basati su Arduino, questo si basa su due elementi distintivi: una basetta singola con il chip ATmega 328P più l'RTC (e quindi non uno shield) e il display a matrice di led da 16 x 24 pixel basato su chip Holtek HT1632. Il display è un componente già montato e disponibile in due colori, con un'interfaccia di tipo seriale.

Grazie a questa soluzione hardware, in cui dei chip specializzati vengono in aiuto di Arduino e lo liberano dalle complessità della gestione di un display a matrice e della tenuta del tempo, si potrebbe pensare che il software sia fatto di poche righe di codice. Sarebbe effettivamente così se Andrew O'Malley non avesse scelto di creare un oggetto di design tecnologico, passando dalla semplice visualizzazione dell'ora a un sistema in Open Source capace di visualizzare animazioni anche complesse per indicare l'ora. Nasce quindi il progetto Dotklok.



Elenco Componenti:

R1: 1 kohm
 R2: 1 kohm
 R3: 10 kohm
 R4: 2,2 kohm
 R5: 2,2 kohm
 C1: 22 pF ceramico
 C2: 22 pF ceramico
 C3: 100 nF ceramico
 C4: 10 μ F 63 VL elettrolitico
 C5: 100 nF ceramico
 IC1: ATMEGA328
 IC2: DS1307
 VR1: 7805
 Q1: Quarzo 16 MHz basso profilo
 Q2: Quarzo 32.768 kHz

Varie:

- Distanziale F/F 40mm (4 pz.)
 - Distanziale F/F 7mm (2 pz.)
 - Distanziale F/F 30mm (4 pz.)

- Vite 8 mm 3 MA (18 pz.)
 - Morsetto 10 poli passo 2,54mm
 - Zoccolo 4+4
 - Zoccolo 14+14
 - Strip maschio 2 vie
 - Strip maschio 6 vie 90°
 - Strip maschio 8 vie (2 pz.)
 - Jumper
 - Porta batteria CR1220
 - Batteria CR1220
 - Plug alimentazione
 - Display a matrice di LED
 - Pulsante da pannello N.O. (4 pz.)
 - Deviatore a slitta da pannello
 - Flat cable 10 vie 20cm
 - Fascetta 7cm
 - Pannello posteriore
 - Pannello frontale
 - Alimentatore 220Vac/5Vdc
 - Circuito stampato

PRIMA CON ARDUINO

Nelle fasi iniziali della progettazione, Dotklok nasce come prototipo classico con Arduino da una parte e una breadboard per il collegamento del modulo RTC, dei pulsanti e l'interfacciamento del pannello a matrice di punti (un modello oggi non più in produzione) che Andrew aveva recuperato da un altro progetto. Verificata la funzionalità, Andrew decide che vale la pena dedicare del tempo alla realizzazione di un PCB dove ospitare microcontrollore, RTC e il necessario per interfacciare il restante hardware. Nella progettazione dello stampato, viene tenuta presente la natura di Arduino e quindi sono previsti dei pin di collegamento alle porte non impegnate del micro, assieme alle linee di alimentazione e massa. In pratica una mini breadboard su cui montare componenti aggiuntivi per potenziare il dispositivo. Più open di così...

LO SCHEMA ELETTRICO DI DOTKLOK

L'attuale versione dell'hardware, pubblicata con licenza Creative Commons Attribution-Sharealike (CC BY-SA 3.0), è la 1.1. che utilizza un ATmega 328P in versione DIL a 28 pin, l'orologio in tempo reale di



Il connettore HDR1 è una interfaccia seriale TTL a cui va collegato un modulo TTL/USB come questo, realizzato da Sparkfun e disponibile da Futura Elettronica con il codice FTDI5V.

In fondo al file delle animazioni si trova il template che riportiamo in questo box. La sua struttura permette di capire cosa sia necessario prevedere in ciascuna animazione perché questa possa essere inserita assieme alle altre senza che Dotklok perda di funzionalità. Il programmatore ha scelto di mettere in questa parte del codice una serie di funzioni, fra cui la lettura dei pulsanti per l'eventuale cambio dell'ora o la pausa per le foto. La lettura dell'ora dal chip Dallas DS1307 viene effettuata in questa parte, così come deve essere gestita l'eventuale visualizzazione di un messaggio di avvio dell'orologio. Dato che ogni animazione potrebbe essere quella selezionata per l'avvio tramite la variabile *animation* (che si trova all'inizio dello sketch principale appena sotto la definizione di DEBUG e PAUSE) la parte di *power_up* diventa quindi necessaria.

La gestione dei pulsanti per la gestione del cambio di data e ora ha invece senso solo se l'animazione che state pensando di inserire ha una visualizzazione "in chiaro" e non arrotondata di questi due valori. Anche osservando le animazioni originali si nota infatti che non c'è regolazione nelle modalità puramente grafiche come *accumulator_time* o *percent_time*.

Se la vostra animazione visualizza i dati in modo completo e chiaro, potete copiare dalle prime due animazioni le routine di gestione della regolazione di data e ora.

Codice

```
// template for new time functions
void time_template(){

    // declare + initialize local variables

    boolean power_up=true; // used to force time display on function startup

    /* INIT DISPLAY */

    ht1632_clear();

    /* TIME LOOP */
    do{ // exit based on button presses, checked later in loop

        time_now = RTC.now();

        if( power_up ){
            // generate any display stuff if needed,
            // this is optional depending on the animation

            // display initialisation based on existing time here

            power_up = false;
        }

        if( ( time_now.minute() != time_prev.minute() ) || power_up ){
            // condition for update based on time change,
            // may not need power_up check if previous if(power_up) routine used

            if( !power_up && midnight_random() ) return;
            // check if need to switch animation at midnight
            power_up=false; // power_up is only true once

            // animation stuff here
            // end of animation stuff

            time_prev = time_now;

        } // end if time_now.min != time_prev.min

        /* CHECK BUTTONS, return if necessary */
        if( change_animation() ) return;

    }while(1);
} // end function time_template
```

Dallas Semiconductors DS 1307, un regolatore di tensione 7805 opzionale e due quarzi. Il primo quarzo è da 16 MHz ed è per il microcontrollore; il secondo è il tipico quarzo cilindrico per orologi, da 32,768 kHz. Pochi altri componenti passivi completano il circuito. La batteria tampone al litio è collegata, come da schema di riferimento, al pin 3 (VBAT) dell'RTC.

L'alimentazione di microcontrollore e display è a 5V; con tutti i led accesi la matrice arriva a circa 200 mA di assorbimento, mentre la parte del microcontrollore è nell'ordine di qualche decina di milliamper.

Se disponete di un alimentatore stabilizzato a 5V capace di erogare almeno 250 mA, potete evitare di montare il regolatore e montare il jumper JP1, ma ricordatevi che in questo caso non c'è regolazione e limitazione nella tensione e se l'alimentatore collegato supera la tensione nominale di 5V, può danneggiare il Dotklok. In linea teorica, vanno bene anche quelli che si trovano sulle bancarelle per ricaricare via cavetto USB i cellulari: dovrebbero infatti erogare fra i 350 e i 700 mA a 5V stabilizzati e il loro costo è veramente irrisorio; in questo caso potete adattare un cavetto

USB – tagliando l'estremità che andrebbe al telefonino - per saldare direttamente i fili che portano l'alimentazione alle piazzole di CN1. Vi consigliamo di controllare con un tester le tensioni presenti perché non è detto che siano rispettati i codici colore dei cavi USB standard, dove il positivo è sul rosso e la massa è sul nero.

Proseguendo nell'analisi dello schema, possiamo notare come la piedinatura tipica di Arduino sia stata riportata sui vari connettori seguendo una logica di compatibilità.

La porta seriale fa capo ai pin 2 (RXD) e 3 (TXD) del microcon-

Le animazioni di Dotklok

Ecco cosa potete aspettarvi da Dotklok. Il passaggio da una animazione alla successiva avviene premendo i pulsanti b3 e b4. L'ordine scelto è il medesimo che si trova nel file `time_animations.pde`.



random_dot_time

Un altro tipo di visualizzazione che utilizza un pixel per rappresentare Quattro minuti. Questa volta però la posizione del punto è del tutto casuale e quindi capire a colpo d'occhio (o anche contando) che ore sono diventa decisamente arduo, soprattutto quando ci si inoltra nella giornata.



percent_time

Partendo sempre dal tempo totale di una giornata, questa volta le ore e i minuti trascorsi dalla mezzanotte sono mostrati come percentuale a due decimali.



morse_time

Ore, minuti e secondi rappresentati come tre righe che scorrono in codice Morse. Un buon sistema per far pratica con i numeri Morse.



basic_time

Questa modalità non è attualmente visualizzata in quanto è stata sostituita dalla successiva "big_time"; per ripristinarla basta agire sul `loop()` e prevedere nuovamente la sua chiamata. In questa modalità l'ora è visualizzata con il font da 4x7 pixel e un punto singolo si sposta sul bordo esterno con la cadenza di uno spostamento al secondo. In questa modalità è possibile impostare ora e modalità 12/24 ore.



big_time

In mezzo allo schermo viene visualizzata l'ora con caratteri da 5 x 12 pixel; il font utilizzato per i caratteri è memorizzata nel file `fonts.h` posizionato nella cartella principale assieme ai PDE. Nel file è già presente un font alternativa denominata "heavy metal" e in generale si tratta di bitmap facilmente modificabili. In questa modalità è possibile impostare ora e modalità 12/24 ore.



seconds_time

La somma del numero superiore con quello inferiore è sempre pari a 86400, ovvero il numero di secondi che compone una giornata. Il numero superiore indica i secondi già trascorsi dalla mezzanotte, mentre il numero inferiore indica i secondi che mancano per arrivare alla mezzanotte successiva. Una visualizzazione decisamente suggestiva e dalla quale è possibile regolare anche la luminosità dello schermo.



pong_time

Questo è il tipo di visualizzazione per il quale ci siamo avvicinati al progetto Dotklok. Apparentemente si tratta di una partita in modalità demo del mitico gioco Pong di Atari (1972). Il gioco non è però casuale in quanto il giocatore di sinistra segna molto meno di quello di destra e a ben vedere il rapporto è di 60 a 1. Questo perché i punti a sinistra sono le ore e quelli a destra sono i minuti.

trollore, con il DTR della porta seriale che va al pin 1 (RESET) al fine di attivare il bootloader quando viene avviata la programmazione. Il circuito dispone solo dell'interfaccia TTL e per il collegamento a un PC per utilizzare l'IDE Arduino risulta necessario utilizzare la classica interfaccia USB/Seriale come quella che abbiamo presentato

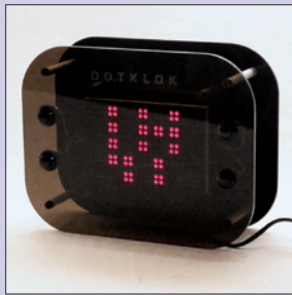
qualche numero fa. Lo stampato prevede un header (HDR1) realizzato con una strip da 6 pin proprio per il collegamento a questa interfaccia. Il secondo connettore (HDR2) è invece stato scelto appositamente a 16 pin su due file per essere uguale a quello della matrice di led; su questo connettore abbiamo il collegamento dei pin

D9, D10 e D11 (corrispondenti ai pin 15, 16 e 17 del micro) che vengono utilizzati per la comunicazione con il chip HT1632. D11 (Pin 17) è collegato a CS1, D10 (Pin 16) è il Write Clock, mentre il D9 (Pin 15) è il Data; da questi pin si può intuire che la comunicazione seriale è paragonabile a SPI su due fili e in effetti la libreria di supporto



float_time

Utilizzando il font piccolo, vengono visualizzate due coppie di cifre nelle due metà dello schermo; a sinistra le ore e a destra i minuti. Ogni secondo si muovono di un pixel in una direzione casuale alternativamente i minuti o le ore.



binary_year_time

Se non state più nella pelle per sapere quanti secondi sono trascorsi dall'inizio dell'anno, questa modalità di visualizzazione fa per voi. L'unico dettaglio è la rappresentazione con cifre binarie a cinque bit organizzate in una matrice da 5 x 5.



accumulator_time

Una via di mezzo fra un orologio analogico e uno digitale: dodici blocchetti da 12 pixel ciascuno sono organizzati per creare un quadrante di orologio rettangolare. Ogni 5 minuti si accende un pixel nel blocchetto corrispondente all'ora corrente, riempiendo durante il corso delle 12 ore tutto il quadrante.



basic_date_time

Sfruttando il font piccolo, da 3 x 5 pixel, vengono mostrati data e ora completi, su due righe. In questa modalità è possibile modificare la data.



relative_column_time

Se non vi interessa molto sapere data e ora, probabilmente questa è la modalità che fa per voi: da sinistra a destra sono indicate dei segmenti la cui altezza è proporzionale a secondi, minuti, ore, giorno, mese e anno. L'indicazione è decisamente "spannometrica" ma non per questo è meno interessante delle altre.



analog_dot_time

Sfruttando il ridotto numero di pixel disponibile, questa modalità si ispira alle lancette e al quadrante di un orologio analogico. Un anello di 12 led indica le 12 ore del quadrante, poi un punto all'esterno segna i minuti e uno all'interno le ore. La precisione è di 5 minuti a pixel; in centro un pixel indica il pomeriggio (PM).



game_time

Questa animazione utilizza Pacman, uno space invader o i pezzi del Tetris per cancellare lo schermo ogni minuto. Nel file delle animazioni è presente la logica di questa modalità, mentre nel file principale si trovano i tre personaggi dei videogiochi con le relative animazioni; nel file fonts.h si trovano invece le bitmap dei personaggi.



column_time

L'intero schermo viene riempito partendo da sinistra e dal basso verso l'alto, con un pixel che corrisponde a quattro minuti. La riga in basso resta libera per ospitare un pixel che si sposta ogni secondo. Una colonna ha quindi 15 pixel e corrisponde a un'ora (15x4=60), mentre le colonne dello schermo sono 24, tante quante sono le ore del giorno. Contando le colonne accese e l'altezza dell'ultima è quindi effettivamente sapere che ore sono, con 4 minuti di approssimazione. Non sono disponibili regolazioni.

al chip, come vedremo nella sezione dedicata al software, utilizza questa modalità di comunicazione. Sullo schema si può vedere come siano possibili quattro diversi chip select (CS) per poter collegare in cascata fino a quattro moduli per una matrice complessiva da 16 x 96 pixel. Il connettore X1 collega i pin da

D8 a D4 (pin 14, 13, 12, 11 e 6 del micro) a una morsettiera a vite con alternati i pin del micro e la massa: qui collegheremo i quattro pulsanti normalmente aperti e il deviatore per gestire le varie modalità di funzionamento del Dotklok. La piccola breadboard prevista sullo stampato corrisponde nello schema elettrico alla serie

di piazzole che fanno capo ai pin in basso a sinistra di IC1; su una scheda Arduino questi corrisponderebbero ai pin A0, A1, A2, A3, D2, D3, D12 e D13. Si tratta di una buona quantità di pin sia digitali, sia analogici con i quali inventare delle espansioni e delle migliorie funzionali al Dotklok. IC2, ovvero l'RTC DS1307,

Come impostare l'ora

Il codice per cambiare l'ora è presente solo nelle animazioni in cui è possibile visualizzare le cifre di ore e minuti, mentre la regolazione del

mezzo e dell'anno avvengono solo con l'animazione date_time. Riportiamo in questo box il codice utilizzato per la regolazione dell'ora nell'animazione basic_time. D'interesse sono tutte le chiamate alla libreria del chip RTC con cui vengono effettuate le letture

e le scritture. Il codice utilizza anche la libreria BUTTONS per gestire la pressione dei tasti che sono stati istanziati nel file principale.

```
/* CHECK BUTTONS (return to main loop if necessary) */

if( change_animation() ) return;

while( PAUSE && b5.isPressed() ); // pause mode for photos

// set hours
if(b1.isPressed()){
  if(b3.uniquePress()){
    if(time_now.hour()==0) RTC.adjust( DateTime(time_now.year(), time_now.month(),
    time_now.day(), 23, time_now.minute(), 0 ) );
    else RTC.adjust( DateTime(time_now.year(), time_now.month(),
    time_now.day(), time_now.hour()-1, time_now.minute(), 0 ) ); time_now=RTC.now();
    puttime_4x7(2,4,13,4,time_now.hour(),time_now.minute()); time_prev = time_now;
  }
  if(b4.uniquePress()){
    if(time_now.hour()==23) RTC.adjust( DateTime(time_now.year(), time_now.month(),
    time_now.day(), 0, time_now.minute(), 0 ) );
    else RTC.adjust( DateTime(time_now.year(), time_now.month(),
    time_now.day(), time_now.hour()+1, time_now.minute(), 0 ) ); time_now=RTC.now();
    puttime_4x7(2,4,13,4,time_now.hour(),time_now.minute()); time_prev = time_now;
  }
  if(twelve_hour){
    if(time_now.hour()<12){
      plot(0,14,1);
      plot(23,14,0);
    }
    else{
      plot(0,14,0);
      plot(23,14,1);
    }
  }
}

// set minutes
if(b2.isPressed()){
  if(b3.uniquePress()){
    if(time_now.minute()==0) RTC.adjust( DateTime(time_now.year(), time_now.month(),
    time_now.hour(), 59, time_now.second() ) );
    else RTC.adjust( DateTime(time_now.year(), time_now.month(),
    time_now.day(), time_now.hour(), time_now.minute()-1, time_now.second() ) );
    time_now=RTC.now(); puttime_4x7(2,4,13,4,time_now.hour(),time_now.minute()); time_prev = time_now;
  }
  if(b4.uniquePress()){
    if(time_now.minute()==59) RTC.adjust( DateTime(time_now.year(), time_now.month(), time_now.day(),
    time_now.hour(), 0, time_now.second() ) );
    else RTC.adjust( DateTime(time_now.year(), time_now.month(), time_now.day(),
    time_now.hour(), time_now.minute()+1, time_now.second() ) ); time_now=RTC.now();
    puttime_4x7(2,4,13,4,time_now.hour(),time_now.minute());
    time_prev = time_now;
  }
}

// change 12/24 hour mode
if( b1.isPressed() && b2.isPressed() ){
  if( b1.uniquePress() || b2.uniquePress() ){
    b1_time=millis()+1000; // set time to 1 second after initial press
  }
  else if(b1_time < millis()){ // toggle 12hour flag if buttons held down for more than a second,
    // then update display
    if(twelve_hour){
      twelve_hour=false;
      plot(0,14,0);
      plot(23,14,0);
    }
    else{
      twelve_hour=true;
      if(time_now.hour()<12){
        plot(0,14,1);
        plot(23,14,0);
      }
      else{
        plot(0,14,0);
        plot(23,14,1);
      }
    }
  }
  b1_time=millis()+1000;
  time_now=RTC.now();
  puttime_4x7(2,4,13,4,time_now.hour(),time_now.minute());
  time_prev = time_now;
}
}
```


viene interfacciato al micro con la classica connessione I2C sui pin 27 e 28 – SDA ed SCL per il micro – che anche sulla scheda Arduino UNO sono identificati per questo tipo di comunicazione. Questa scelta permette di utilizzare le librerie già disponibili per interfacciare l’RTC senza dover fare particolari modifiche. È interessante notare come sia possibile, con queste informazioni circa le corrispondenze dei vari pin con i pin di Arduino, realizzare il Dotklok in forma prototipale senza modificare il software: usando un po’ di cavallotti e una breadboard è infatti possibile prendere una scheda Arduino UNO, aggiungere il chip Dallas, il display (codice 7719-DISP24X16 di Futura Elettronica), i pochi componenti passivi e i pulsanti ottenendo il medesimo risultato funzionale.

LE LIBRERIE DI APPOGGIO

Prima di analizzare la struttura dello sketch che permette a Dotklok di funzionare, vale la pena dare uno sguardo alle librerie che Andrew O’Malley ha pensato di utilizzare per rendere più semplice il suo lavoro.

Il programma inizia con questa serie di “include”:

```
#include <Wire.h>
#include <RTCLib.h>
#include <Button.h>
#include "ht1632.h"
```

La prima libreria è presente nell’installazione base dell’IDE di Arduino e serve per la gestione del protocollo I2C usato dall’orologio in tempo reale.

Oltre alla comunicazione, l’RTC ha bisogno anche della propria libreria di supporto che rende più intuitiva la lettura di data e ora o la riprogrammazione dei dati memorizzati.

La libreria Button è invece un’interessante soluzione, scritta da Alexander Brevig, per la gestione di pulsanti. Con questa libreria si ottiene un livello di astrazione dall’hardware e la lettura di un pulsante, definito semplicemente tramite il pin a cui è collegato, avviene tramite funzioni di facile comprensione (isPressed, stateChanged, wasPressed ecc.).

La libreria permette di creare varie istanze per vari pulsanti che possono essere quindi nominati in modo opportuno. Nel nostro caso, saranno da b1 a b5. Chi volesse approfondire questa libreria, trova tutti i riferimenti a questo indirizzo: playground.arduino.cc/Code/Button.

L’ultima libreria è stata scritta appositamente per gestire il controller HT1632 attraverso istruzioni di alto livello che ne sfruttano tutte le varie funzioni. In pratica, nella cartella della libreria si trova solo il file con i vari define relativi ai comandi, mentre tutta la gestione a livello più alto è sviluppata in uno sketch di esempio, poi preso da Andrew per realizzare Dotklok.

Questo sketch di esempio implementa tutte le funzioni necessarie e permette innanzitutto di gestire i comandi per l’attivazione di uno dei quattro chip che possono essere collegati in cascata, poi quelli per la luminosità o per scrivere direttamente dei byte in memoria; in aggiunta la libreria/esempio supporta sia l’accensione dei singoli pixel con un’istruzione plot, sia la copia di caratteri in specifici punti dello schermo secondo matrici di varie dimensioni.

In pratica ci sono tutti gli elementi che servono per gestire con facilità un pannello grafico.

IL SOFTWARE DI DOTKLOK

La struttura scelta dall’autore per

Animazioni Vers. 1.5

```
big_time
basic_date_time
game_time
pong_time
float_time
seconds_time
percent_time
column_time
random_dot_time
analog_dot_time
binary_year_time
scroll_time
```

Animazioni Vers. 1.4

```
big_time
basic_date_time
game_time
pong_time
float_time
seconds_time
percent_time
column_time
random_dot_time
relative_column_time
morse_time
analog_dot_time
accumulator_time
binary_year_time
```

lo sketch deriva da una serie di fattori ed il risultato è composto da tre file più tre librerie. Prima di realizzare Dotklok, Andrew si era cimentato in un altro progetto, sempre basato su un display a matrice con il controller HT1632, denominato “Electric Window 3”; da questo ha ereditato il file di controllo della matrice, a sua volta ottenuto modificando il file demo16x24.c di Bill Westfield. Si tratta di una ottima dimostrazione di come la condivisione dei codici sorgente sia di stimolo alla realizzazione di progetti anche molto diversi fra loro a livello finale, ma accomunati dal successivo perfezionamento e riciclo di un medesimo codice. Questo è anche il motivo per cui abbiamo il file principale “dotklok_rev_1_x.pde”, “ht1632.pde”

Sketch 1

```
void loop()
{
  // all time and button checking handled in animation functions
  // main loop just handles animation number (init. to 1)
  // bound the animation number
  // animation = constrain(animation,1,ani_max);
  // if(DEBUG) animation=6;
  // used to force any animation during testing/debugging
  if(DEBUG) { Serial.print("loop / animation: "); _
              Serial.println(animation);}

  // run the clock animation
  switch(animation) {
    case 1:
      big_time();
      break;
    case 2:
      basic_date_time();
      break;
    case 3:
      game_time();
      break;
    case 4:
      pong_time();
      break;
    case 5:
      float_time();
      break;
    case 6:
      seconds_time();
      break;
    case 7:
      percent_time();
      break;
    case 8:
      column_time();
      break;
    case 9:
      random_dot_time();
      break;
    case 10:
      relative_column_time();
      break;
    case 11:
      morse_time();
      break;
    case 12:
      analog_dot_time();
      break;
    case 13:
      accumulator_time();
      break;
    case 14:
      binary_year_time();
      break;
  } // end switch(animation)

} // end main loop
sett=1
ELSE
```

e "time_animations.pde" assieme alle tre librerie e a un file con i font sia grafici, sia numerici (non ci servono le lettere, ovviamente). I file sono scaricabili dal sito www.andrewomalley.org/dotklok/ e le versioni disponibili, al momento in cui scriviamo, vanno dalla 1.0 alla 1.5; quella con il maggior numero di animazioni è la 1.4, mentre la 1.5 contiene l'orario con i caratteri che scorrono, ma perde tre delle animazioni. La memoria disponibile per il codice è di 32K e

Dotklok ne raggiunge sostanzialmente il limite in quanto è un'applicazione con numerose routine e funzioni, alcune delle quali sono presenti nel codice solo per consentire a chi volesse sviluppare una nuova animazione di avere a disposizione uno strumento per il debug tramite porta seriale.

I FILE

Per evitare di occupare troppe pagine con il listato dei vari sketch, ne riportiamo solo alcune parti e vi suggeriamo di scaricare i file dall'indirizzo indicato prima di procedere con la lettura. In questo modo potrete seguire i nostri commenti con il codice a portata di mano. Dato che l'ultima release di Dotklok è antecedente al rilascio della versione 1 dell'IDE, i file hanno ancora l'estensione PDE e non ancora INO. Quando li aprirete con le attuali versioni di IDE, verrà cambiata l'estensione. In alternativa, potete scaricare e usare la versione 0.22 di Arduino ed evitare così qualsiasi problema di compatibilità degli sketch e delle librerie. Più avanti vi spiegheremo cosa fare per rendere compatibile il firmware con l'IDE 1.0.x.

Il file principale - dotklok_rev_1_x.pde - contiene i *define*, le variabili globali e tutte le parti di codice di utilità e visualizzazione, assieme al classico *setup()* e *loop()*. Nella nostra analisi utilizzeremo la versione 1.4 che contiene anche il maggior numero di animazioni.

Come abbiamo già spiegato, l'autore della libreria per la gestione del chip HT1632 in realtà ha scritto uno sketch di esempio che contiene il necessario per disegnare e scrivere sul display a matrice di LED, quindi Andrew è partito da quell'esempio per creare prima la sua finestra elettronica e poi Dotklok, lasciando a questa parte di codice la gestione vera e propria del display; il file si chiama HT1632.pde e non va confuso con la libreria. Questo sketch crea anche una mappa in memoria della situazione dei singoli pixel - chiamandola *shadowram* -, così da evitare la lettura dello stato degli stessi dal chip, cosa che potrebbe risultare lenta e poco efficiente. Le funzioni di *plot*, *clear* e di stampa dei caratteri sono in questo sketch, mentre altre funzioni per la gestione della grafica sono nello sketch principale. Chi volesse sperimentare con il solo display collegato ad Arduino, può ripartire dal file di esempio che si trova nella cartella all'interno di `\libraries\ht1632\examples\demo16x24`.

Nel file *time_animations.pde* si trovano invece i quindici segmenti di codice per le varie modalità di visualizzazione del tempo. In fondo al file si

trova anche un template per la realizzazione di nuove animazioni.

LO SKETCH PRINCIPALE

Torniamo ora al file `dotklok_rev_1_4` per analizzarne la struttura con un maggiore dettaglio. Innanzitutto va considerato che questo software è frutto della modifica di uno sketch dimostrativo del display a cui è stato costruito attorno il resto. Per questo motivo abbiamo un po' di comandi e funzioni "sparpagliate" fra i vari file. Probabilmente si potrebbe rendere il tutto un po' più leggibile e anche più compatto riorganizzando il codice in due soli file: uno per le animazioni e uno per tutto il resto.

Comunque, dato che stiamo pubblicando il lavoro che Andrew O'Malley ha scelto di condividere con il mondo rilasciandolo in open source sin dalle prime versioni, lasciamo a voi l'eventuale "ristrutturazione" del codice e la successiva ricondivisione in Open Source.

La dimostrazione di quanto il codice sia strutturato in modo diverso dall'abituale sviluppo di uno sketch Arduino è confermato dal loop principale, riportato in **Sketch 1**.

Dal codice appare evidente che tutte le logiche e le routine siano state messe da qualche altra parte. Appare anche chiaro come l'autore abbia previsto un sistema per il debugging attraverso la porta seriale che si attiva quando viene valorizzata a 1 la variabile `DEBUG`.

Questa variabile è proprio all'inizio dello sketch assieme ad una seconda variabile globale:

```
/** DEFINES AND GLOBAL VARIABLES */  
boolean DEBUG=false;  
// can be forced true by pressing B4 during setup  
boolean PAUSE=false;  
// if true, use B5 to pause the clock for taking photographs
```

Numerosi messaggi vengono trasferiti sulla seriale quando `DEBUG` è impostato come `true` e questo permette a chi volesse cimentarsi nella scrittura di nuove animazioni di capire cosa sta succedendo nel codice.

La seconda variabile è stata invece predisposta per congelare l'animazione dello schermo per consentire la realizzazione di fotografie. Da notare che `b5` non è un pulsante, ma è il deviatore a slitta che solitamente serve per abilitare il cambiamento casuale dell'animazione alla mezzanotte di ogni giorno. Se `PAUSE` viene impostato su `true` cambia il funzionamento di `b5` che diventa un comando di pausa dell'animazione; l'orologio continua a incrementare data e ora, ma sul display l'immagine

Sketch 2

```
void setup()  
{  
  // init the button inputs  
  pinMode(4, INPUT);  
  pinMode(5, INPUT);  
  pinMode(6, INPUT);  
  pinMode(7, INPUT);  
  pinMode(8, INPUT);  
  digitalWrite(4, HIGH);  
  digitalWrite(5, HIGH);  
  digitalWrite(6, HIGH);  
  digitalWrite(7, HIGH);  
  digitalWrite(8, HIGH);  
  
  // debug mode forced by B4 pressed during start up  
  if(!digitalRead(7)) DEBUG=true;  
  
  if(DEBUG){  
    Serial.begin(57600);  
    Serial.print("DOTKLOK "); Serial.println(REV);  
    Serial.println("DEBUG true, serial port open at 57600");  
    //Serial.print("Avail mem = ");  
    //Serial.println(availableMemory());  
  }  
  else{  
    Serial.begin(57600);  
    Serial.print("DOTKLOK "); Serial.println(REV);  
    Serial.println("DEBUG false, serial port now closed");  
    Serial.end();  
  }  
  
  // display screen setup, software version display, (and test)  
  ht1632_setup();  
  ht1632_clear();  
  putchar_3x5(7, 5, REV[0]);  
  putchar_3x5(13, 5, REV[2]);  
  plot(11, 9, 1);  
  delay(1000);  
  if(DEBUG) screenTest();  
  
  // general set up  
  pinMode(13, OUTPUT);  
  randomSeed(analogRead(0));  
  
  // RTC setup  
  Wire.begin();  
  RTC.begin();  
  
  if (!RTC.isrunning()) {  
    if(DEBUG) Serial.println("RTC is NOT running!");  
    // following line sets the RTC to the date & time  
    // this sketch was compiled  
    RTC.adjust(DateTime(__DATE__, __TIME__));  
  }  
  else {  
    if(DEBUG) Serial.println("RTC is running.");  
  }  
  
  // test buttons  
  /*  
  while( b5.isPressed() ){  
    for(int i=0; i<4; i++){  
      if( buttons[i].isPressed() ) plot(i,0,1);  
      else plot(i,0,0);  
    }  
  } */  
}  
// end setup()
```

resta immobile fino all'apertura del contatto. La gestione della pausa si trova in tutte le animazioni, verso il fondo, con questa riga di codice:

```
while( PAUSE && b5.isPressed() ); // pause mode for photos
```

A differenza di `DEBUG`, che può anche essere lasciato a `false` e poi gestito tramite la pressione di `b4` all'accensione, la funzionalità di `b5` come `PAUSE` per le foto richiede una compilazione specifica

Mappa 1

File: **Dotklok_rev_1_4.PDE**

Define e variabili globali

...

Utility

int availableMemory()

Funzioni Display

```
void screenTest()
void shift_display_left()
void shift_display_right()
void shift_display_down()
void shift_display_up()
void clear_column(byte col)
void clear_row(byte row)
void update_display()
void clear_shift_right()
void clear_shift_left()
void clear_display()
void clear_display_buffer()
```

Funzioni di animazione

```
boolean change_animation()
boolean midnight_random()
void puttime_3x5(int x_hour, int y_hour, int x_min, int y_min, byte hour, byte minute)
void puttime_4x7(int x_hour, int y_hour, int x_min, int y_min, byte hour, byte minute)
void bufchar_4x7(byte x, byte y, char c)
void buftime_4x7(int x_hour, int y_hour, int x_min, int y_min, byte hour, byte minute)
void puttime_5x12(int x_hour, int y_hour, int x_min, int y_min, byte hour, byte minute)
void put_sprite(int x, int y, const uint8_t sprite[8])
void del_sprite(int x, int y, const uint8_t sprite[8])
void invader()
void put_tetris(byte x, byte y, char c)
void del_tetris(byte x, byte y, char c)
void tetris_fall(int del)
void put_pacman(byte x, byte y, char c)
void del_pacman(byte x, byte y, char c)
void pacman(int del)
```

Setup

Loop

File: **HT1632.PDE**

```
void ht1632_chipselect(byte chipno)
void ht1632_chipfree(byte chipno)
void ht1632_writebits(byte bits, byte firstbit)
static void ht1632_sendcmd(byte chipno, byte command)
static void ht1632_senddata(byte chipno, byte address, byte data)
void ht1632_setup()
void plot(char x, char y, char val)
void ht1632_clear()
void putchar_3x5(byte x, byte y, char c)
void putchar_4x7(byte x, byte y, char c)
void putchar_5x12(byte x, byte y, char c)
```

File: **Time_Animations.PDE**

```
void basic_time()
void big_time()
void basic_date_time()
void column_time()
void float_time()
void seconds_time()
void percent_time()
void random_dot_time()
void pong_time()
void relative_column_time()
void analog_dot_time()
void game_time()
void morse_time()
void binary_year_time()
void accumulator_time()
```

e il caricamento sul micro tramite ICSP o modulo USB/Seriale. Una seconda struttura tipica degli sketch Arduino è la parte di setup() dove viene inizializzato l'hardware; nel nostro caso ecco cosa troviamo (**Sketch 2**).

In questa parte del codice, a parte l'impostazione dei pin a cui sono collegati i quattro pulsanti e il deviatore, abbiamo l'inizializzazione dello schermo, la stampa della versione del firmware e l'inizializzazione del modulo RTC con il classico metodo

```
if (!RTC.isrunning())
```

per l'impostazione di data e ora di compilazione se il chip RTC non è stato ancora impostato o è stata rimossa la batteria temporaneamente senza alimentazione da rete. Anche in questa sezione sono presenti routine specifiche per il debug del codice e in particolare si nota come la pressione di b4 determini l'impostazione di questa modalità. Senza DEBUG, viene chiusa la comunicazione seriale.

A questo punto, le parti classiche di uno sketch si sono esaurite, eppure non abbiamo ancora trovato nulla che ci permetta di capire come il firmware di questo progetto funzioni in pratica. Come vi abbiamo già in parte spiegato, le varie funzioni sono "sparpagliate" nei tre file principali, con questo che contiene solo una parte di routine e di funzioni di utilità.

Per aiutarvi nella navigazione fra i file, abbiamo fatto una mappa dei tre file con le funzioni e routine contenute in ciascuno (**Mappa 1**).

Come si può notare dai vari nomi scelti, le funzioni di disegno che sono state implementate nello sketch principale costitui-

scono un buon arsenale con cui sviluppare nuove animazioni. Nello sketch principale troviamo anche il necessario per visualizzare o preparare la visualizzazione dell'orario con i caratteri piccoli, medi e grandi. Sempre nello sketch principale sono presenti le funzioni grafiche evolute che servono a stampare i cosiddetti "sprite", ovvero degli elementi grafici che vengono sovrastampati sulla grafica esistente. Quando lo sprite viene cancellato, non viene però ripristinata la grafica sottostante: per ottenere questo è necessario salvare prima l'area grafica su cui lo sprite verrà stampato.

La funzione viene utilizzata per la gestione del game_time dove a turno Pacman, uno space invader e dei pezzi di tetris invadono il display e cancellano l'ora. Una certa attenzione merita anche HT1632.PDE che, nella parte iniziale, contiene una serie di routine di configurazione legate ai possibili tipi di display collegati. Non le abbiamo inserite nella mappa in quanto sono superate dall'utilizzo dell'attuale modello di display prodotto da Sure Electronics e dotato di caratteristiche già previste e impostate nel codice.

COMPILARE DOTKLOK

Come abbiamo già rilevato all'inizio dell'articolo, il programma è nato ed è stato sviluppato con l'IDE 0.22 di Arduino. Anche le tre librerie utilizzate sono state scritte ai tempi delle versioni precedenti di Arduino. Chi vuole usare la versione 1.0 dell'IDE deve quindi metter mano al codice per modificare quegli elementi che altrimenti renderebbero la compilazione impossibile. Innanzitutto c'è la libreria

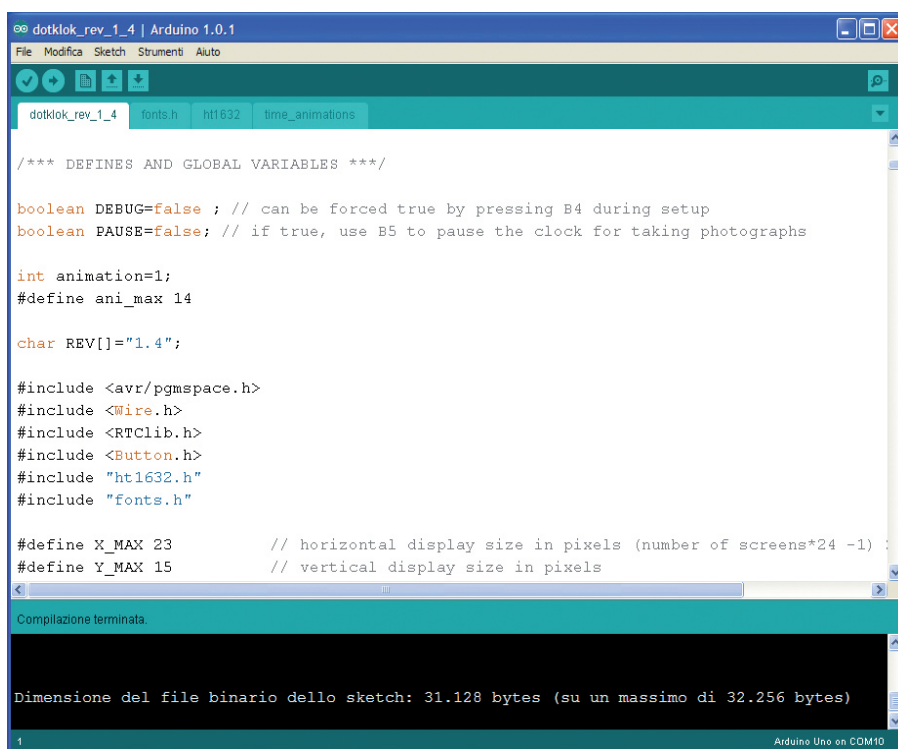


Fig. 1

"Wprogram.h" che va sostituita con "Arduino.h"; nel nostro caso la troviamo in RTCLib.h, utilizzata assieme a Wire.h che è un'altra delle librerie soggette a cambiamenti. Se per Wprogram basta sostituire con Arduino, per Wire la situazione è un po' più antipatica in quanto sono stati cambiati due comandi:

Wire.send() diventa *Wire.write()*
Wire.receive() diventa *Wire.read()*

Nella libreria dovreste quindi cambiare un certo numero di righe, ma utilizzando il Trova dell'IDE o usando un altro editor di testo, la sostituzione non dovrebbe essere più faticosa di un tanto.

Fatte le dovute modifiche, la compilazione andrà a buon fine. Potete impostare l'IDE per la programmazione di un Arduino UNO e otterrete il risultato indicato nella Fig. 1.

AGGIUNGERE ANIMAZIONI

Fra istruzioni per il debug e scelte di programmazione, il codice di Dotklok è già al limite della capacità dell'ATmega328P e quindi se pensate di aggiungere delle funzionalità o delle animazioni, dovreste obbligatoriamente rinunciare a qualcosa, commentando negli sketch forniti qualche parte.

Il suggerimento dell'autore è quello di intervenire sulle animazioni: in questo modo non toccherete parti di codice che potrebbero essere collegate ad altri punti nel medesimo file o negli altri utilizzati. Lo spazio occupato dallo sketch viene visualizzato al termine della compilazione e in base allo spazio occupato per eccesso o per difetto potrete scegliere se ripristinare qualche animazione. Il mese prossimo ci occuperemo del montaggio e dell'utilizzo di Dotklok.

OpenWheels

LA TELEMETRIA

Prosegue la serie di articoli dedicati alla piattaforma OpenWheels con la verifica e la gestione dei dati rilevati dai sensori attraverso la telemetria e il collegamento wireless.



di MIRCO SEGATELLO

A questo punto del progetto la scheda OpenWheels è testata e collaudata e pronta per funzionare con il suo firmware. Appare logico attendersi che il firmware subisca miglioramenti con l'andare del tempo, per questo motivo questa prima versione sarà chiamata

OpenwheelsV1.0. Il firmware consiste in sette file - ciascuno sviluppato per una specifica funzione - assemblati dal compilatore durante la fase di caricamento nel microcontrollore. Nel caso di programmi con un numero rilevante di righe si preferisce spezzare il codice in sottoprogrammi, ognuno riportato in un file se-

parato; in questo modo la lettura del codice è semplificata e la modifica di una parte del codice implica mettere le mani su uno solo di questi file. Ovviamente è necessario che il software sia progettato per questa modalità, facendo in modo che le modifiche ad un modulo non influenzino il funzionamento degli altri. I sette file che compongono il firmware con la loro descrizione sono riportati

nella **Tabella 1**. Si tratta di poco meno di novecento righe di codice ottimizzate per ottenere un programma snello e veloce, pur dovendo eseguire parecchie operazioni con il piccolo microcontrollore ad 8 bit con clock di soli 16MHz. In linea generale, come filosofia, si è puntato sull'uso di variabili di tipo *float* solo se strettamente necessario e, ove possibile, sono stati usati accessi

Tabella 1 - Moduli contenuti nel firmware

Nome modulo	Funzione
EEPROM	in questo modulo sono riportate le funzioni per la gestione delle scritture e letture nella memoria non volatile dell'ATmel328.
IMU	è il codice necessario all'elaborazione dei dati del modulo inerziale ovvero l'analisi dei valori dell'accelerometro e giroscopio.
Motor	in questo modulo sono riportate le funzioni per il setup del modulo PWM ed il controllo del duty-cycle dei motori.
OpenWheels	modulo principale con la definizione delle variabili, il ciclo principale di elaborazione e la gestione del modulo LCD.
PID	è il codice per l'elaborazione della stabilizzazione.
serLCD	modulo contenente le istruzioni per la gestione del display seriale LCD
Telemetry	modulo contenente le funzioni relative all'invio e alla ricezione dei dati di telemetria.

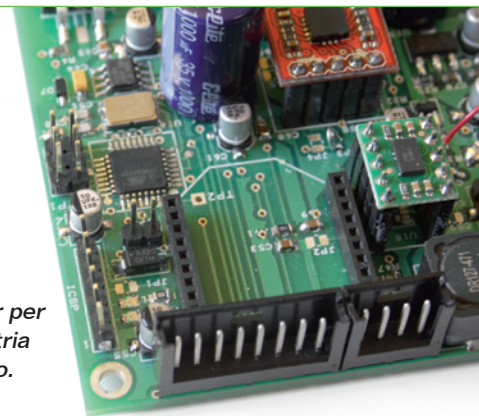
diretti ai registri tramite istruzioni assembler che permettono di ridurre il codice aumentando la velocità. Sembrerà strano, ma le sole righe che permettono la stabilizzazione di OpenWheels non sono poi molte, mentre la gestione della telemetria e del display LCD occupano parecchie righe.

Il firmware, in dettaglio, sarà analizzato in un articolo apposito più avanti, mentre adesso ci occupiamo della telemetria e degli ultimi collaudi.

Appena installato il firmware, la scheda OpenWheels è perfettamente operativa e la telemetria risulta già attiva; quest'ultima può essere utilizzata connettendo la scheda ad un PC tramite il normale sistema di programmazione (cavo + adattatore USB/seriale). Su OpenWheels non viene installato il modulo XBee ed il jumper JP6 è lasciato aperto, **Fig. 1**.

Sul lato PC dovrà essere avviata l'apposita applicazione scritta in Processing chiamata OpenWheels GUI (Graphic User Interface), in grado di gestire la comunicazione con la scheda. Essendo scritta in Java non necessita di installazione ed è sufficiente copiare i file in una cartella ed avviare l'applicazione; è compatibile sia con Windows, sia con Linux. Per il corretto funzionamento è necessario avviare la scheda OpenWheels e aspettare almeno 5 secondi, affinché siano completate le operazioni di inizializzazione, dopodiché è possibile avviare il software per la telemetria. Sulla sinistra è disponibile un menù a tendina per la scelta della porta seriale da usarsi, questa dovrà corrispondere alla COM assegnata dall'adattatore USB/Seriale (oppure USB/XBee). Questo collaudo può essere fatto anche prima di installare la sche-

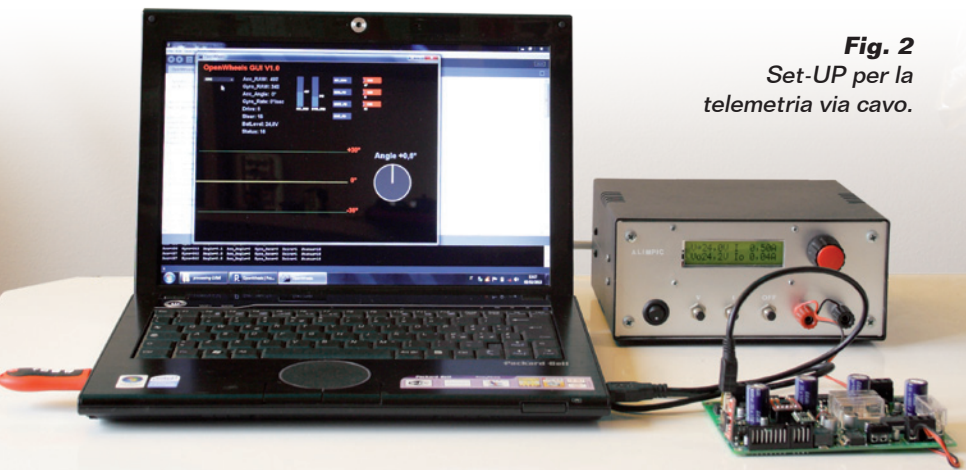
Fig. 1
Jumper per telemetria via cavo.

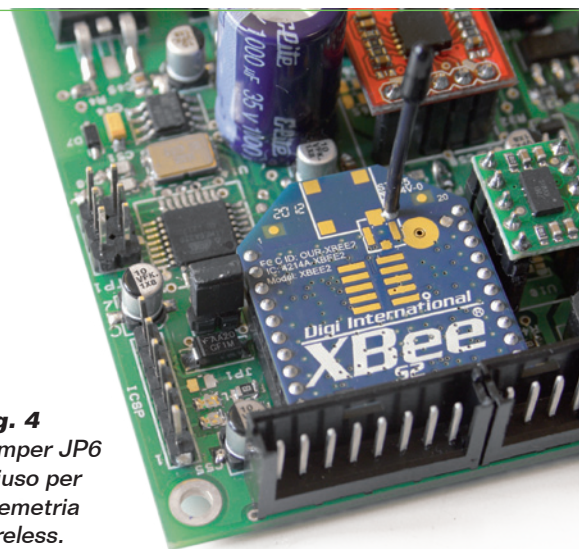
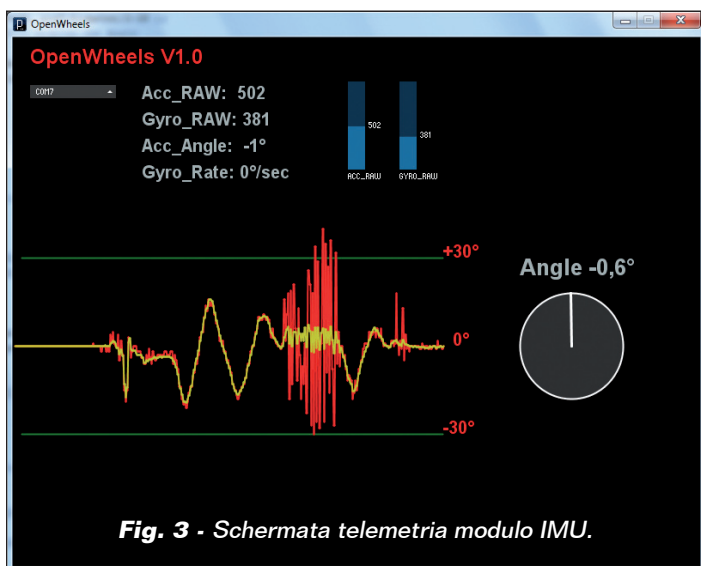


da sulla meccanica OpenWheels, a patto che siano inseriti sia il giroscopio, sia l'accelerometro. Questa interfaccia fornisce indicazioni dirette sulla lettura dell'accelerometro e del giroscopio, facilmente interpretabili grazie alla rappresentazione grafica in tempo reale dei dati acquisiti. Per sfruttare al meglio questa metodologia di analisi sarebbe opportuno che la scheda fosse lasciata libera ed alimentata separatamente tramite un alimentatore, in questo modo sarebbe più facilmente orientabile, **Fig. 2**. Questo software, come anche quello della scheda, hanno subito notevoli miglioramenti e la primissima versione consisteva nella sola lettura del modulo IMU per verificare la correttezza degli algoritmi implementati nel software.

In **Fig. 3** vedete la traccia rossa relativa alla misura dell'accelerometro e la traccia gialla relativa alla misura dell'angolo ricavato con il filtro complementare. È stato simulato un sobbalzo di OpenWheels, agitando velocemente la scheda, per vedere come si comporta la lettura dell'angolo elaborato matematicamente; come si vede, la misura dell'angolo rimane molto pulita. Per meglio comprendere questo test si raccomanda la visione dei relativi video, disponibili sul sito ufficiale del progetto <http://openwheels.elettronica.in.it>. Il grafico è del tipo Roll-mode, come

Fig. 2
Set-UP per la telemetria via cavo.





quelli usati dai monitor cardiaci, perché permette di avere una chiara visione istantanea e passata dello stato della variabile; il semplice invio dei dati su *Serial Monitor* di Arduino non avrebbe avuto la stessa utilità.

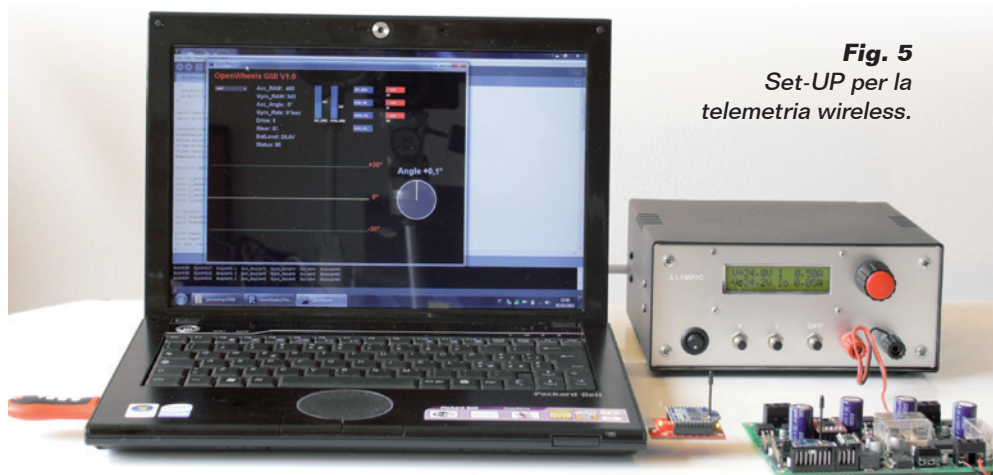
Se installate la scheda sulla meccanica OpenWheels e volete vedere i dati della telemetria, dovrete dotarvi di un netbook da portare a bordo con l'ovvia scomodità (e pericolosità) della lettura dei dati mentre vi muovete; per questo motivo abbiamo previsto un sistema di telemetria wireless tramite moduli XBee. Per questa modalità dovete procurarvi due moduli XBee; vanno bene di qualsiasi versione tranne quelli PRO che, essendo più lunghi non entrano nella scheda. È necessario che vi procuriate anche un adattatore USB/XBee per interfacciare uno dei due moduli al PC; nel nostro caso abbiamo usato due moduli VBee serie 2.5 distribuiti da Futura Elettronica con codice 7300-XBEE2MWWIRE e l'adattatore - sempre di Futura Elettronica - con codice 7300-XBEE2DONGLE oppure l'adattatore con codice 7300-EXPLORERXBEE.

ATTENZIONE: quando utilizzate la telemetria con i moduli XBee è necessario cortocircuitare il

jumper JP6 (**Fig. 4**) che però deve essere lasciato aperto durante la fase di programmazione; questo evita un conflitto tra il modulo XBee e l'adattatore USB-Seriale. Mentre i moduli della serie 1 possono essere utilizzati direttamente senza alcuna programmazione, i moduli della serie 2 necessitano dell'impostazione di alcuni parametri, questo perché in un sistema XBee vi deve essere almeno un *coordinator* e un *end-device*. A noi interessa programmare i moduli in modo che supportino la modalità denominata *transparent mode*, ossia una comunicazione in grado di sostituire un normale collegamento a filo, inoltre vogliamo che solo i due moduli XBee utilizzati nel progetto

comunicino tra loro, ignorando ogni altra possibile trasmissione. Per maggiori dettagli sull'uso dei moduli XBee si rimanda alla lettura del relativo articolo sul numero 152 della rivista.

Per prima cosa scarichiamo ed installiamo il software fornito dall'azienda produttrice dei moduli, chiamato X-CTU; se necessario aggiorniamo l'elenco dei firmware dei moduli. Inseriamo uno dei due moduli nell'adattatore USB/Xbee e lo connettiamo al PC, tramite l'applicazione eseguiamo la funzione *Test/Query* per verificare che il modulo XBee sia correttamente rilevato, successivamente usiamo la funzione *read* per leggere i parametri attualmente settati sul modulo



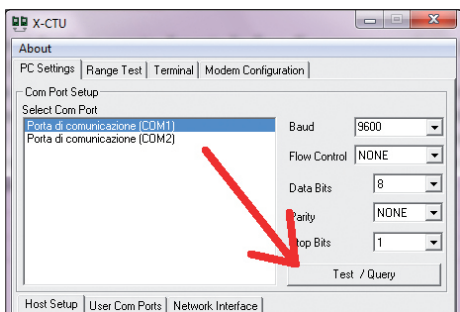


Fig. 6 - Uso del software X-CTU.

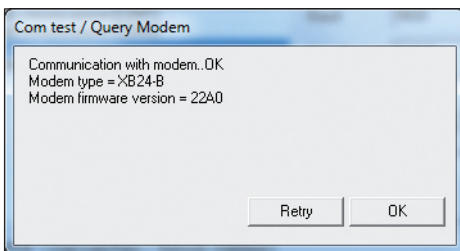


Fig. 7 - Modulo Xbee rilevato.

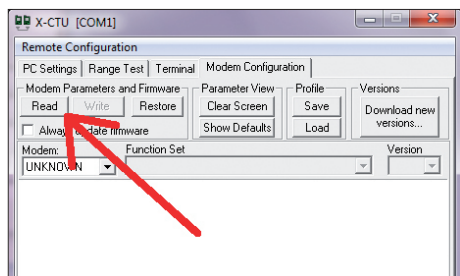


Fig. 8 - Lettura parametri modulo Xbee con X-CTU.

(Fig. 6, Fig. 7 e Fig. 8). Copiatevi su di un foglio di carta il numero identificativo del modulo, impostato in fabbrica e univoco per ciascun modulo; si tratta dei due valori SH e SL.

Sostituire il modulo Xbee e ripetete la procedura di lettura dei dati; sul nuovo modulo dobbiamo impostare, nei rispettivi campi, i valori riportati in **Tabella 2**, **Fig. 9**. La riduzione della potenza è una precauzione per evitare che un eccessivo assorbimento in potenza possa surriscaldare il piccolo stabilizzatore di tensione che fornisce i 5volt sulla scheda OpenWheels. Scambiamo nuovamente il modulo Xbee e settiamo il nuovo modulo come indicato in **Tabella 3** e **Fig. 10**.

Verificate che il campo ID, l'identificativo della rete (Personal Area Network), sia posto a zero

Tabella 2 - Impostazioni parametri modulo XBee da inserire su OpenWheels.

Nome modulo	Funzione	Nuovo valore
FS	modo funzionamento	ZIGBEE END DEVICE AT
DH	indirizzo di destinazione alto	indirizzo SH dell'altro modulo
DL	indirizzo di destinazione basso	indirizzo SL dell'altro modulo
PL	potenza trasmissione	2 (medium power)
BD	velocità di trasmissione	7-115000

per entrambi i moduli. Adesso i due moduli possono comunicare solo tra di loro, sostituendo di fatto un collegamento seriale fisico alla velocità di 115Kbps. Per meglio comprendere le funzionalità della telemetria vi consigliamo di visionare i filmati associati a questo articolo. Due V-meter a barra forniscono, in tempo reale, la misura dell'accelerometro (*Acc_RAW*) e del giroscopio (*Acc_RAW*); questo permette di valutare la quantità di scala effettivamente usata nelle misure e se vi sono situazioni che portano ad un fuori scala. La telemetria fornisce anche il valore dell'inclinazione con la sola misura dell'accelerometro (*Acc_Angle*) e la velocità di rotazione ottenuta dalla misura del giroscopio (*Gyro_Rate*). La variabile *Drive* è l'effettivo valore inviato per il controllo dei motori, mentre la variabile *Steer* è il valo-

re dello sterzo. Anche i livelli della tensione e della corrente sulla batteria sono riportati dalla telemetria. Sulla destra, l'indicatore rotondo riporta invece l'angolo calcolato tramite la fusione della misura dei due sensori, valore che oltre ad essere molto accurato è anche insensibile a disturbi meccanici esterni. Sulla destra delle schermata trovate alcuni pulsanti utili alla gestione remota di OpenWheels. La funzione *SET_ZERO* imposta lo zero per l'accelerometro ed il giroscopio; il valore rilevato viene automaticamente salvato nella memoria del microcontrollore e questo evita che ogni volta che si accende OpenWheels venga reimpostato lo zero. L'impostazione dello zero dura pochi secondi e richiede che OpenWheels sia perfettamente sulla verticale. La funzione *READ_PID* permette di leggere i parametri del controllo

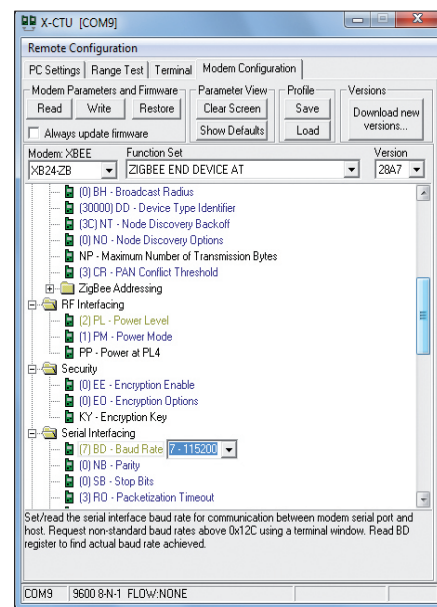
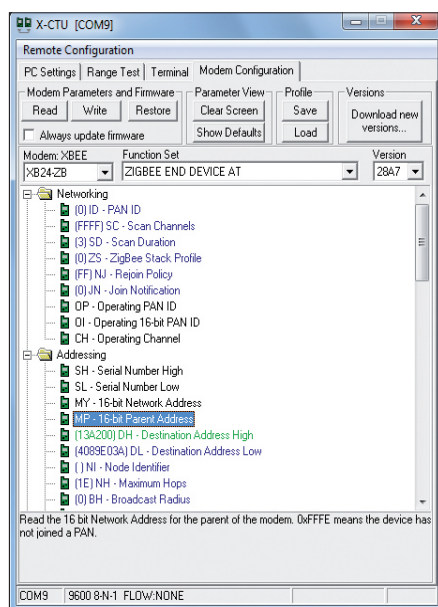


Fig. 9 - Impostazione modulo XBee END DEVICE.

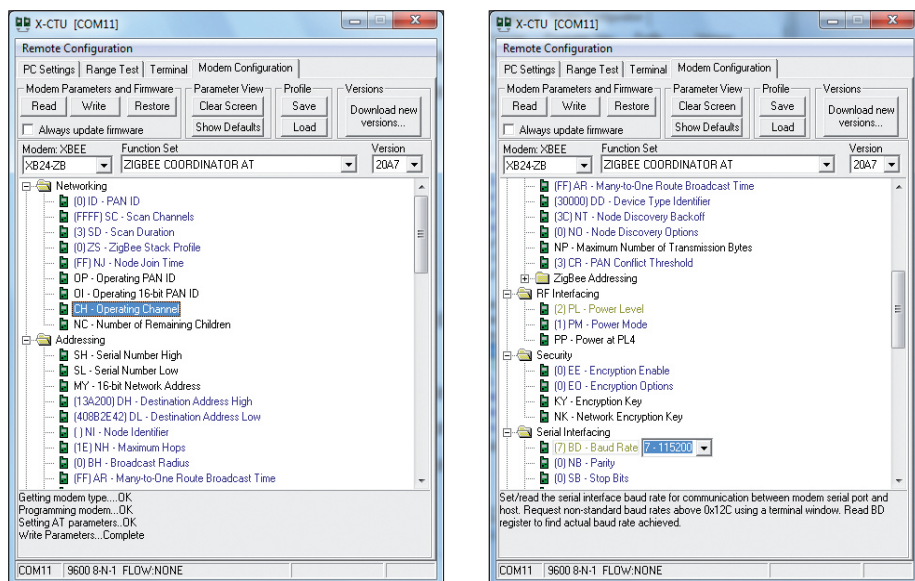


Fig. 10 - Impostazione modulo XBee COORDINATOR.

Tabella 3 - Impostazioni parametri modulo da usarsi con il PC.

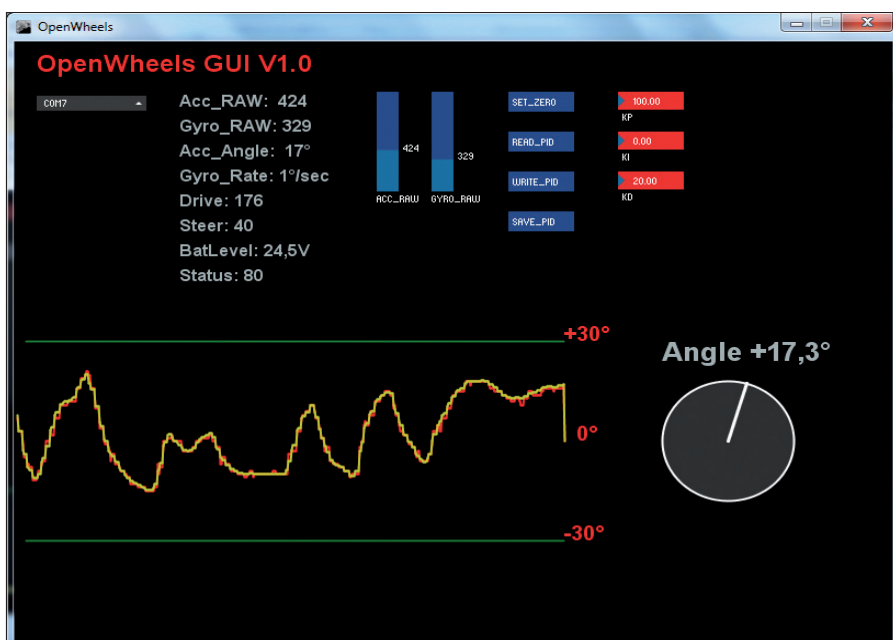
Parametro	Funzione	Nuovo valore
FS	modo funzionamento	ZIGBEE COORDINATOR AT
DH	indirizzo di destinazione alto	indirizzo SH dell'altro modulo
DL	indirizzo di destinazione basso	indirizzo SL dell'altro modulo
PL	potenza trasmissione	2 (medium power)
BD	velocità di trasmissione	7-115000

PID implementato sul firmware; essi possono essere modificati cliccando sulla casellina in rosso riportante il valore. Dopo ogni modifica è necessario che i nuovi dati vengano inviati al microcontrollore tramite il comando WRITE_PID, mentre il salvataggio nella memoria EEPROM avviene cliccando sul comando SAVE_PID.

Al primo utilizzo della scheda OpenWheels i valori dello zero dei sensori non sono ancora stati rilevati, è quindi necessario provvedere al settaggio dello ZERO; i valori dello zero del sensore di corrente e dello sterzo vengono invece rilevati automaticamente durante la fase di accensione. Anche questa volta siamo giunti al termine; sul prossimo numero vi illustreremo l'utilizzo pratico di OpenWheels.

Fig. 11 - Schermata di OpenWheels GUI.

Ricordiamo che per questo progetto abbiamo realizzato un minisito (<http://openwheels.elettronica.in>) nel quale vengono descritti in maniera molto più dettagliata i particolari costrutti-



Guarda come gestire la telemetria dell'Open Wheels



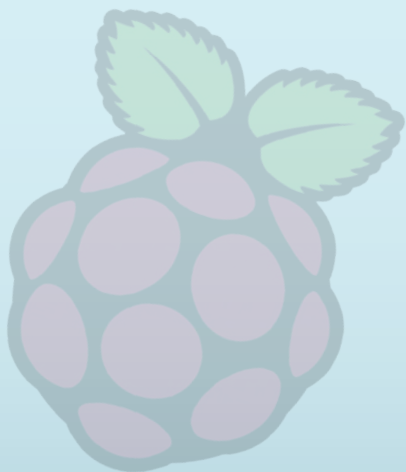
Vuoi vedere il software di gestione dei dati rilevati dai sensori dell'Open Wheels attraverso la telemetria e il collegamento wireless? Scansiona il QR Code con il tuo smartphone!



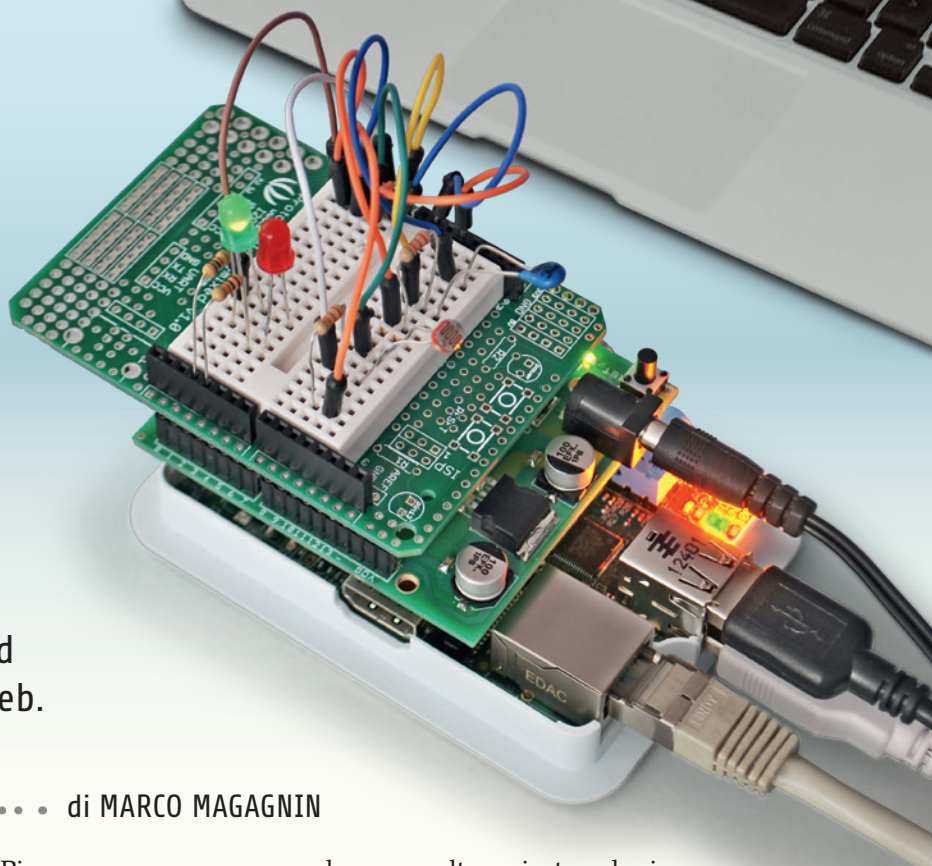
vi del nostro veicolo, le modifiche apportate nel tempo e quelle suggerite dai lettori, nonché i circuiti di controllo proposti dai lettori. Lo scopo ultimo è quello di realizzare un veicolo perfettamente funzionante ad un costo di gran lunga inferiore rispetto a quello di analoghi prodotti disponibili in commercio nonché quello di poter personalizzare il veicolo in relazione alle proprie esigenze.

TERMOSTATO WEB

CON LO SHIELD DI ESPANSIONE ADC PER RASPBERRYPI



Dopo aver presentato la scheda di espansione per RaspberryPi compatibile Arduino ed aver realizzato il programma python, che calcola la temperatura e la luminosità ambiente in gradi e Lux, utilizziamoli per realizzare un termostato "didattico" con un'interfaccia grafica dinamica ed interattiva, gestita da browser web.



..... di MARCO MAGAGNIN

Abbiamo sempre descritto RaspberryPi come un ottimo "banco di scuola" per studiare e sperimentare sistemi che vengono generalmente classificati come "Linux embedded". Vi abbiamo già descritto più volte le differenze esistenti tra un sistema a microcontrollore, tipicamente monoprogramma, ed un sistema embedded dotato di un sistema operativo di controllo come GNU/Linux. Nel primo caso si progetta e si realizza un unico

programma che, una volta caricato sul microcontrollore, entra in funzione immediatamente all'accensione del microcontrollore stesso, esegue una fase di inizializzazione "fuori ciclo" e poi entra in un ciclo infinito dove sono contenute tutte le istruzioni operative richieste dall'applicazione. Per un buon funzionamento bisogna suddividere correttamente le porzioni di codice, tenendo conto dei tempi di elaborazione in relazione alle frequenze di campionamento dei sensori e della necessità di

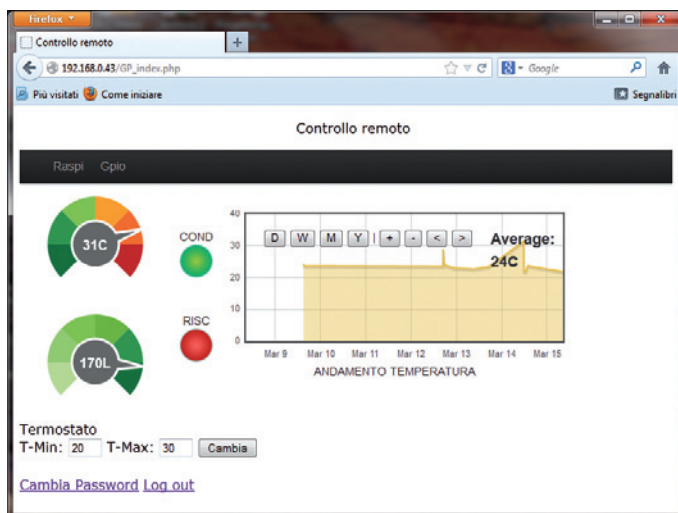


Fig. 1

esempio iniziale di architettura embedded, l'utilizzo della centralina meteo. In quel progetto lo scopo era di rilevare delle grandezze meteo e con queste alimentare ad intervalli regolari un servizio di raccolta dati esterno.

In questo articolo vi presentiamo un progetto con il servizio di raccolta e distribuzione dati ospitato su RaspberryPi, con anche la possibilità di modificare da remoto i parametri di controllo.

Come base utilizziamo la scheda di espansione presentata negli ultimi numeri, dotata di circuiti di conversione dei livelli logici e di convertitore analogico digitale a bordo e "ricicliamo", con l'aggiunta di poche istruzioni, anche il programma python di misura e calcolo della temperatura e luminosità assieme al "vecchio" programma GPiOServer.py2 presentato nel numero 171 della rivista, che gestisce i pin GPIO in modo centralizzato.

Quello che vogliamo realizzare è un sistema di controllo di tipo Termostato in grado di misurare la temperatura ambiente, confrontarla con i valori di soglia memorizzati in una tabella di database e attivare, se la temperatura si colloca al di sopra o al di sotto dei valori di soglia, un ipotetico impianto di condizionamento o di riscaldamento. Nel nostro caso i due impianti sono rappresentati da due LED. Vogliamo, inoltre, che il sistema sia gestibile da un'interfaccia Web, in modo da poterlo utilizzare da remoto.

Nella **Fig. 1** vediamo l'interfaccia grafica, da realizzare con tecnologia Web.

La pagina web dell'interfaccia grafica è divisa in due aree. Un'area è dedicata alla visualizzazione dello stato dei sensori di temperatura, di luminosità e dei LED che rappresentano gli impianti termici. Quest'area si deve aggiornare automaticamente al variare dello stato del sistema, senza necessità di intervenire sulla pagina, in modo da presentare sempre una situazione il più possibile aggiornata. La seconda area è quella dei comandi, dove sono contenuti i campi per inserire i valori di soglia del "Termostato" ed il pulsante di invio.

Analizziamo in dettaglio le diverse funzionalità che vogliamo vengano realizzate.

1. Controllo temperatura con termostato di azionamento degli impianti di riscaldamento e condizionamento. Il sistema rileva la temperatura mediante un sensore a termistore NTC e confronta il valore letto con due valori di soglia, uno superiore e uno inferiore. Se la temperatura misurata scende sotto il valore di soglia inferiore viene acceso un LED che simula un impianto di riscaldamento, spegnendolo al superamento della

gestire alcuni eventi con degli interrupt o dei timer. Il tutto da gestire rigorosamente all'interno di un unico codice monolitico. Su un microcontrollore tipico, il programma è sempre monolitico, anche se il codice sorgente iniziale è composto da più moduli e librerie fisicamente separati in file differenti. Al momento della compilazione tutti i moduli vengono "inclusi" in un unico sorgente cumulativo che viene poi compilato in un unico eseguibile in formato comprensibile al microcontrollore, come .bin, .hex o simili.

Un sistema embedded, al contrario, si presenta come una costruzione di mattoncini LEGO, di ogni forma, colore e dimensione. L'applicazione finale si presenta come un'architettura composta per la maggior quantità possibile di mattoncini preconfezionati, con alcuni componenti realizzati ad hoc per coprire le funzionalità specifiche della nostra applicazione. Il grande vantaggio sta nel poter assegnare al sistema diversi compiti, che possono essere svolti in contemporanea, anche di natura eterogenea come la lettura di sensori, la gestione di un database e di un server web.

Lo svantaggio è che è richiesta una maggiore conoscenza della struttura e dei "mattoni" del sistema e nel dover progettare un'architettura multi processo dove a ciascun processo sono assegnati compiti ben precisi, le prestazioni e l'affidabilità dell'applicazione sono garantite principalmente dall'orchestrazione della comunicazione tra processi e dal coordinamento tra i diversi componenti. Se un microcontrollore può essere paragonato ad un musicista solista che porta su di sé l'intera responsabilità dell'esecuzione, un sistema embedded si configura come una grande orchestra composta da molti elementi specializzati dove il risultato finale è merito delle capacità organizzative e direttive del direttore d'orchestra.

Negli articoli scorsi vi abbiamo presentato, come

soglia. Se la temperatura sale oltre il valore di soglia superiore viene acceso un secondo LED che simula un impianto di condizionamento; anche questo viene spento quando la temperatura torna al di sotto della soglia. Nell'area grafica della pagina Web si vuole visualizzare un indicatore a lancetta per la rappresentazione della temperatura, il relativo valore testuale, un grafico lineare dell'andamento della temperatura nel tempo e due icone che riportano lo stato dei LED, che simulano il riscaldamento ed il condizionatore. Nell'area di comando della pagina web vi devono essere due box di testo che permettono di inserire i valori di soglia per l'attivazione e lo spegnimento di riscaldamento e condizionamento con il relativo pulsante di invio dei dati;

2. Misura della luminosità ambiente – Il sistema rileva la luminosità ambiente per mezzo di una foto resistenza e ne calcola il valore in Lux. La misura viene riportata nell'area grafica della pagina sia con un indicatore a freccia, sia con il valore testuale.

Nel più squisito stile di progettazione "embedded", invece di lanciarsi a sviluppare codice, iniziamo consultando l'immenso catalogo di applicazioni open disponibili, alla ricerca di mattoni che possano risolvere in tutto (difficilmente) o almeno in parte le necessità del nostro progetto.

Il problema vero nella realizzazione pratica della nostra applicazione web è la presentazione grafica degli indicatori dei sensori e dello stato dei LED del nostro sistema Termostato.

Noi vogliamo che i valori e gli stati presentati nella pagina siano costantemente aggiornati, senza la necessità di ricaricare la pagina. Questa funzionalità, per essere realizzata richiede l'adozione di tecniche di programmazione web particolari, che permettono di mantenere aggiornata la pagina aperta sul browser anche senza che venga richiesto un refresh della pagina stessa.

Con una realizzazione della pagina web in HTML tradizionale, se il LED che simula l'impianto di riscaldamento cambia di stato, accendendosi, noi vedremmo cambiare lo stato del LED in locale, con l'aggiornamento dello stato anche nel record database, ma per poter vedere la variazione di stato anche nella pagina web dovremmo chiedere di ricaricare la pagina.

Il risultato è di avere sulla pagina web dati non aggiornati e quindi non affidabili, un po' come se il cruscotto della nostra automobile indicasse la velocità alla quale andavamo dieci minuti fa. Si rende necessario quindi trovare un modo di mantenere aggiornato il contenuto della pagina web senza necessità di chiederne il "refresh".

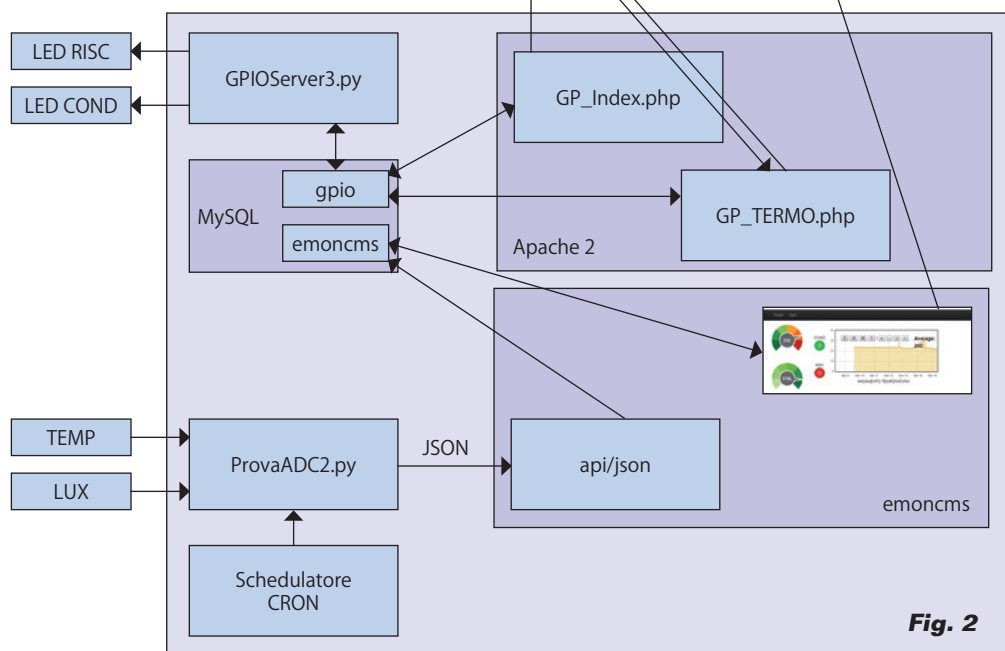
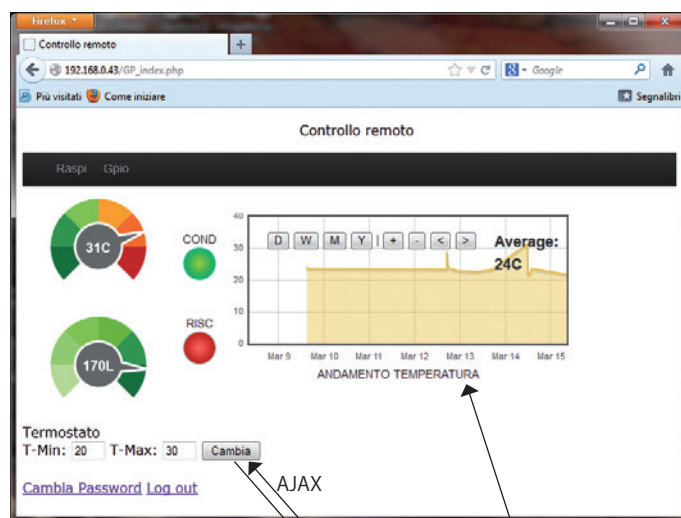
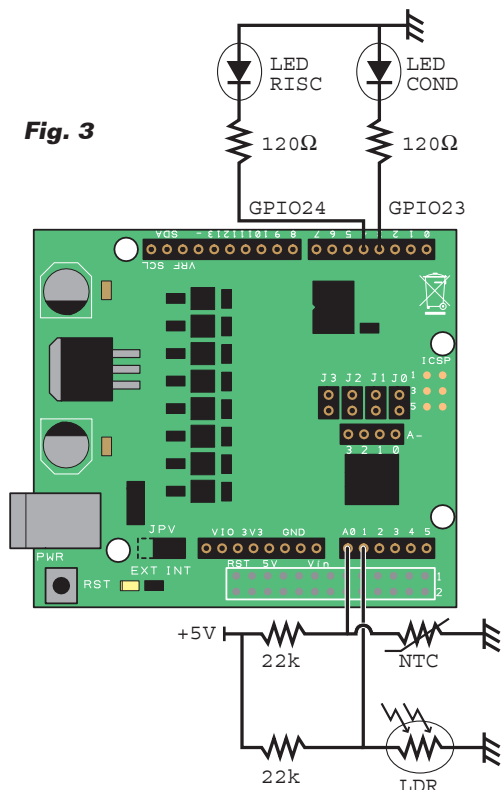


Fig. 2

Fig. 3



Una soluzione possibile è l'utilizzo di una delle tante tecnologie "push" che permettono di realizzare architetture applicative in grado, una volta aperta una pagina web su un browser, di alimentare e modificare dati presenti nella pagina stessa da parte del server, senza che sia necessario intervenire dal lato browser. Per realizzare queste funzionalità è possibile utilizzare Flash, oppure architetture reverse Ajax o ancora Comet e diverse altre. Tutte tecniche complesse e troppo "informatiche" da presentare su una rivista di elettronica. In alternativa abbiamo preferito presentarvi e utilizzare un framework open source finalizzato alla raccolta, elaborazione e divulgazione di dati provenienti da sensori. Si chiama emoncms, è scritto in linguaggio PHP, gira sotto il server web Apache2 e richiede MySQL come database. Tutti prerequisiti pienamente soddisfatti dalla configurazione del nostro RaspberryPi. Lo utilizzeremo per realizzare l'area grafica all'interno della

pagina web principale dove sono inseriti i comandi per modificare le soglie di temperatura per l'attivazione degli impianti (LED). Cogliamo anche l'occasione per introdurre un'altra tecnica di programmazione, che ci permette di inviare i comandi di variazione delle soglie di temperature in modo interattivo, cioè senza necessità di ricaricare la pagina web ad ogni "invio", a tutto vantaggio delle prestazioni del nostro RaspberryPi. Nello schema di **Fig. 2** possiamo vedere l'architettura generale della nostra applicazione, realizzata anche con molti "pezzi" riciclati e adattati dai nostri progetti precedenti. Ovviamente abbiamo fatto delle copie dei programmi recuperati dai progetti precedenti. Nel resto dell'articolo ci riferiremo al programma GPIOServer3.py "figlio" del programma demone che gestisce accensione e lo spegnimento fisico dei LED ed al programma ProvaADC2.py, "figlio" del programma che calcola i valori di temperatura e luminosità a partire dalle misure del sensore NTC e fotoresistenza collegati ai pin del convertitore analogico digitale. Nella **Fig. 3** possiamo vedere lo schema elettrico, fondamentalmente identico a quello dell'applicazione presentata nei numeri scorsi, con l'aggiunta di due LED e delle relative resistenze da 120 ohm sui piedini D3 e D4 (rispettivamente GPIO23 e GPIO24) della scheda di espansione. Qui abbiamo operato una piccola "eresia" elettronica. La nostra scheda di espansione gestisce la conversione dei livelli di tensione tra quelli a 3,3V richiesti dai pin del GPIO di RaspberryPi e i 5 V degli ingressi/uscite di Arduino. Trattandosi di convertitori di livelli non è previsto applicare dei carichi sui terminali dei

convertitori stessi, in quanto la corrente erogata è minima e sufficiente unicamente a mantenere lo stato dei livelli. Nel nostro caso l'integrato TXB0108 è in grado di fornire complessivamente, sulle uscite convertite, un massimo di 50mA. In deroga a ogni regola li abbiamo utilizzati per accendere i LED, dei quali ne è normalmente acceso uno solo alla volta, evitando di realizzare un circuito con transistor di potenza. È il massimo del "peccato" che ci possiamo "permettere": già tre LED non sono sopportabili dallo shield di espansione che smette di funzionare correttamente. Vediamo a grandi linee, con riferimento allo schema di **Fig. 2**, il funzionamento dell'architettura "informatica". Partiamo dalla pagina web. È composta da due aree, come abbiamo già anticipato, una grafica - che presenta i valori di temperatura, luminosità e lo stato dei LED - e l'altra di comando. L'area grafica è "inclusa" nella pagina principale con un tag HTML <iframe> (inline frame) che permette di creare un frame "al volo" in qualsiasi punto di un documento HTML non strutturato a frame. Il link di riferimento dell'iframe richiama la pagina pubblica fornita dal server emoncms, che gira autonomamente sotto Apache2 e svolge due funzioni principali, fornire la pagina grafica ed accettare in ingresso i flussi di messaggi che alimentano i valori dei sensori e lo stato dei LED. Le misure di temperatura e luminosità vengono inviate al server emoncms con il protocollo JSON dal programma ProvaADC2.py, nostra vecchia conoscenza, opportunamente modificato per sottomettere i messaggi e controllare la temperatura rispetto alle soglie impostate da web e immagazzinate nella tabella xTemp del database "gpio". Sem-

Fig. 4

pre il programma ProvaADC2.py, se necessario, aggiorna lo stato dei LED nella tabella pinStatus del database.

Il programma GPIOServer3.py, gira in continuazione con un ciclo while temporizzato e, se verifica un cambio di stato nei LED, ne cambia lo stato fisico sui rispettivi pin del GPIO e sottomette i messaggi JSON al server emoncms per aggiornare lo stato dei simboli dei LED nell'area grafica della pagina web.

Nella parte inferiore della pagina web è presente il comando per modificare i valori di soglia delle temperature per l'attivazione degli impianti termici. Abbiamo anticipato che questa operazione avviene sfruttando la tecnica interattiva AJAX che non richiede di ricaricare la pagina dopo la pressione del pulsante "Cambia". Ci sembra che basti, quindi benvenuti nel mondo "embedded".

REALIZZAZIONE PRATICA

Iniziamo dalla modifica del database "gpio". Dobbiamo creare la tabella xTemp che contiene i valori di soglia delle temperature alle quali vengono attivati e disattivati gli impianti termici e modificare la tabella pinStatus per adattarne i campi alle esigenze della nuova architettura.

Poi, modifichiamo i programmi ProvaADC2.py, GPIOServer3.py, installiamo e personalizziamo la pagina grafica del server emoncms, prepariamo la pagina web della applicazione e ... lanciamo il tutto.

Iniziamo con le modifiche al database.

MODIFICA DATABASE GPIO

Utilizziamo lo strumento, già noto, phpMyAdmin, che richiamo da browser con l'URL: <http://<IP-RaspberryPi>/phpmyadmin> autenticiamoci con utente e pas-

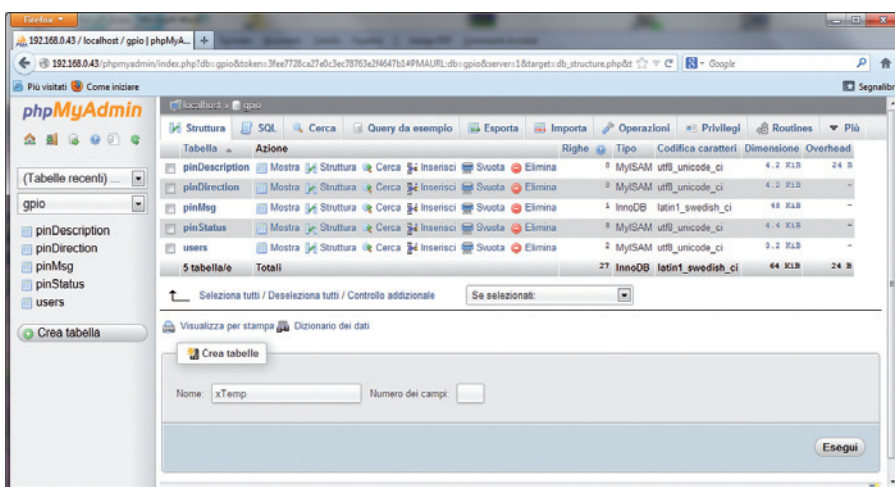


Fig. 5

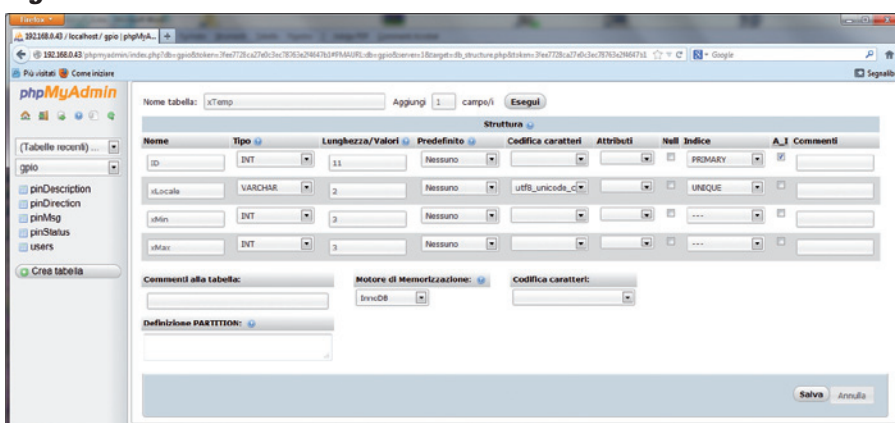
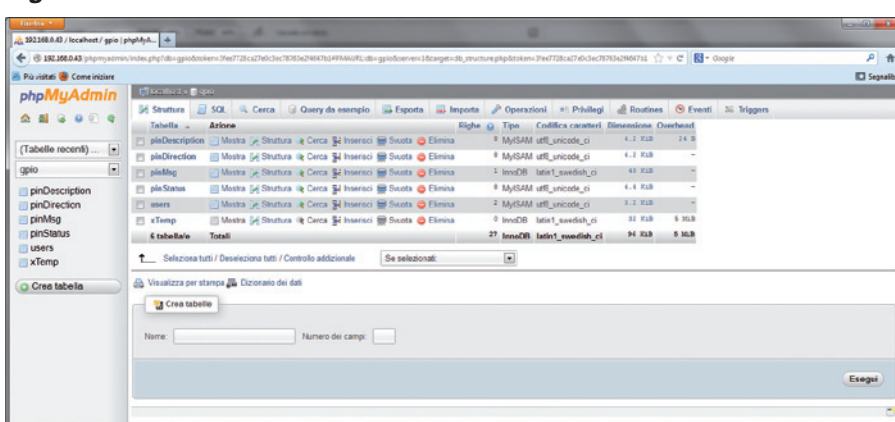


Fig. 6



sword ed entriamo nel database gpio. Nell'area "Crea tabelle" aggiungiamo una tabella con nome "xTemp" e premiamo il pulsante "Esegui" come visibile in Fig. 4. Compiliamo i campi della tabella xTemp come visibile nella Fig. 5 e confermiamo premendo il pulsante "Salva".

Oltre ai valori di soglia xMin e xMax per le temperature di accensione e spegnimento dell'impianto di condizionamento e riscaldamento abbiamo previsto un campo xLocale per permettere, in una ipotetica applicazione reale, di gestire più di un ambiente. Vediamo la tabella creata nella pagina di riepilogo del database

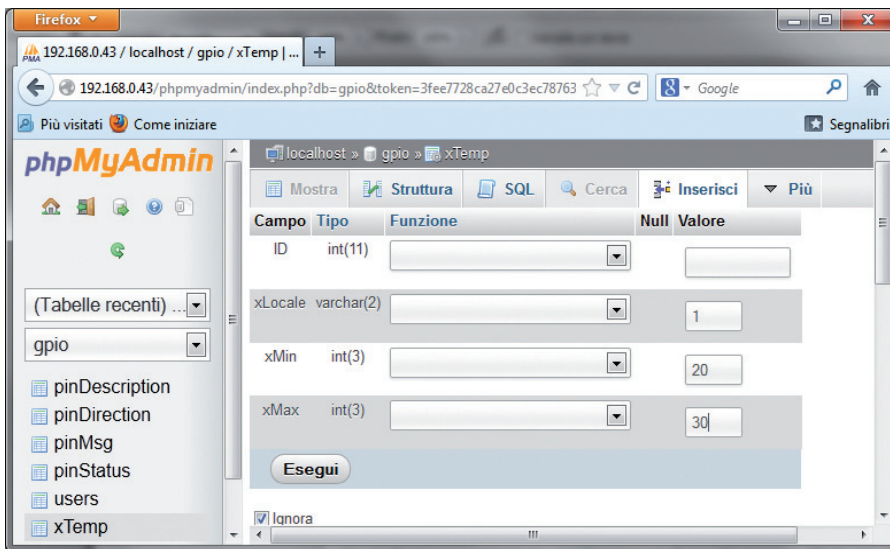


Fig. 7

uno - pinMess - per la vecchia gestione del messaggio vocale, se la vogliamo mantenere.

Inoltre abbiamo inserito il campo pinDir direttamente nella tabella pinStatus per compattare un po' i codici dei programmi. Per eseguire la modifica utilizziamo sempre lo strumento phpMyAdmin.

Questa volta vi lasciamo un po' più soli nella realizzazione delle modifiche alla struttura e ai contenuti della tabella. Introduciamo invece l'utilizzo di MySQL a linea di comando, che utilizziamo per visualizzare il risultato finale delle modifiche alla tabella pinStatus.

Per accedere a MySQL digitate il comando:

```
mysql -u root -p
```

che apre una shell mysql, con l'utente root e la richiesta della password.

Digitate la password quando vi viene richiesta e vi troverete il prompt di mysql.

Per vedere la struttura della tabella pinStatus digitate l'istruzione riportata nel **Comando 1**.

Per vedere il contenuto della stessa tabella digitate il comando riportato nel **Comando 2**.

Per uscire digitate il comando:

```
mysql> exit
```

Utilizzando phpMyAdmin cerca-

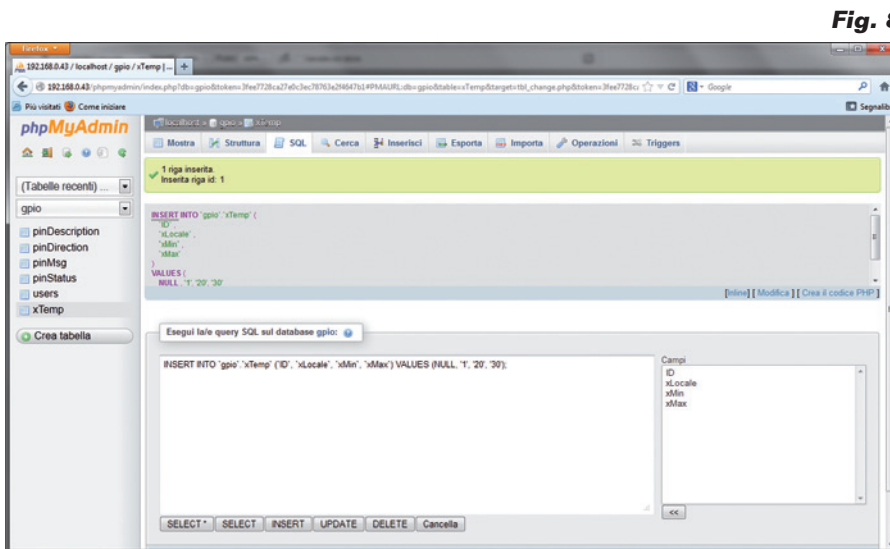


Fig. 8

come in **Fig. 6** e selezioniamo "inserisci" in corrispondenza della nuova tabella xTemp per popolare i campi. Inseriamo i valori visibili in **Fig. 7** e confermiamo premendo il pulsante "Esegui". Nella **Fig. 8** vediamo la pagina di conferma di avvenuto inserimento dei valori e selezionando la voce xTemp sul menu a sinistra torniamo alla classica rappresentazione tabellare di **Fig. 9**.

Per completezza apriamo il pannello "Struttura" nella pagina di **Fig. 10** ed espandiamo il campo "Indici". Possiamo verificare in questo modo la correttezza della struttura della tabella.

Ora le modifiche alla tabella pinStatus. In questa applicazione

ci serve un campo - pinCms - per comunicare al programma GPIO-Server3.py se deve inviare un messaggio a emoncms o meno a fronte di una variazione dello stato dei LED, poi un flag - pinMod - che avvisa della avvenuta modifica dello stato dei LED e

```
mysql> show fields in gpio.pinStatus;
```

Field	Type	Null	Key	Default	Extra
pinID	int(11)	NO	PRI	NULL	auto_increment
pinNumber	varchar(2)	NO	UNI	NULL	
pinPos	varchar(2)	NO	UNI	NULL	
pinDir	varchar(3)	NO		NULL	
pinStatus	varchar(1)	NO		NULL	
pinCms	varchar(10)	YES		NULL	
pinMod	varchar(1)	NO		NULL	
pinMess	varchar(1)	NO		NULL	

8 rows in set (0.01 sec)

Comando 1

Fig. 9

te di riprodurre lo stesso risultato, sia come struttura, sia come valori contenuti (Fig. 11 e Fig. 12). Se avete difficoltà potete scaricare dal sito di ElettronicaIn, come sempre, il set completo dei programmi e lo script per generare il database da nuovo, come descritto nel numero 170 della rivista.

MODIFICHIAMO IL PROGRAMMA PROVAADC2.PY

Per comodità abbiamo creato una copia del programma originale ProvaADC.py che abbiamo chiamato, con molta fantasia, ProvaADC2.py.

In questo programma abbiamo aggiunto le istruzioni necessarie a: realizzare la richiesta JSON verso il server emoncms e le istruzioni necessarie a leggere i valori di soglia della temperatura dalla tabella xTemp; a verificare il superamento o meno delle soglie e a impostare di conseguenza lo stato dei LED che rappresentano l'impianto di riscaldamento e condizionamento sulla tabella pinStatus.

```
import MySQLdb as mdb
import requests
```

```
DbServer = "localhost"
DbUser = "gpio"
DbPasswd = "pi"
```

```
URL_EMONCMS = 'http://<IP-RaspberryPI>/emoncms'
EMONCMS_API_KEY = <APIkey dell'in-
```

```
mysql> select * from gpio.pinStatus;
```

pinID	pinNumber	pinPos	pinDir	pinStatus	pinCms	pinMod	pinMess
1	4	7	xxx	0	NULL	0	0
2	17	11	xxx	0	NULL	0	0
3	18	12	xxx	0	NULL	0	0
4	27	13	xxx	0	NULL	0	0
5	22	15	xxx	0	NULL	0	0
6	23	16	out	0	COND	1	0
7	24	18	out	0	RISC	1	0
8	25	22	xxx	0	NULL	0	0

8 rows in set (0.00 sec)

Comando 2

stallazione emoncms>

Abbiamo già visto nei programmi presentati nei numeri precedenti l'importazione delle librerie, la novità qui è l'importazione della libreria requests per la realizzazione della chiamata

HTTP verso il server emoncms con il messaggio JSON di comunicazione dei valori di temperatura e luminosità.

La costante URL_EMONCMS deve essere personalizzata con l'indirizzo IP del vostro RaspberryPi.

La costante EMONCMS_API_KEY deve essere inizializzata con il valore dell'APIkey caratteristica dell'installazione di emoncms, come vedremo nel paragrafo dedicato alla sua installazione. Nel frammento di codice successivo vediamo la connessione al database, la lettura dei valori di soglia della temperatura dalla tabella xTemp e il confronto con la temperatura appena letta. A

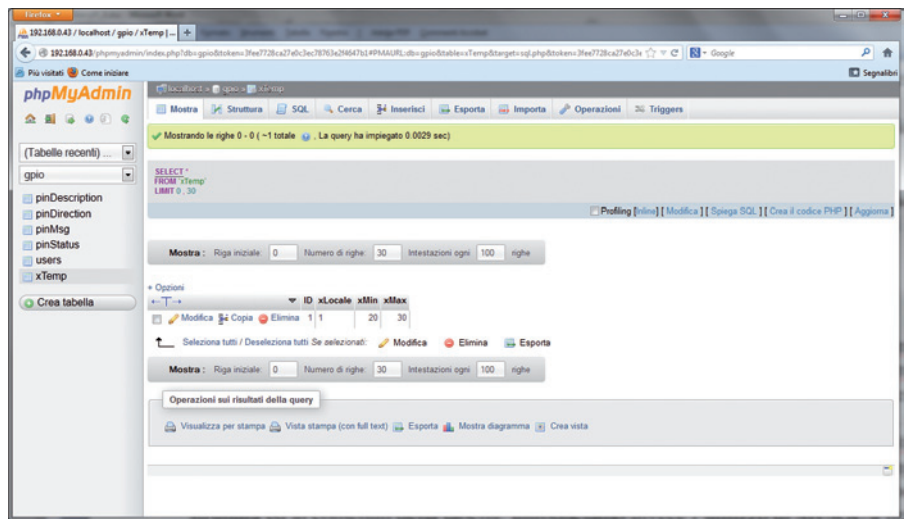
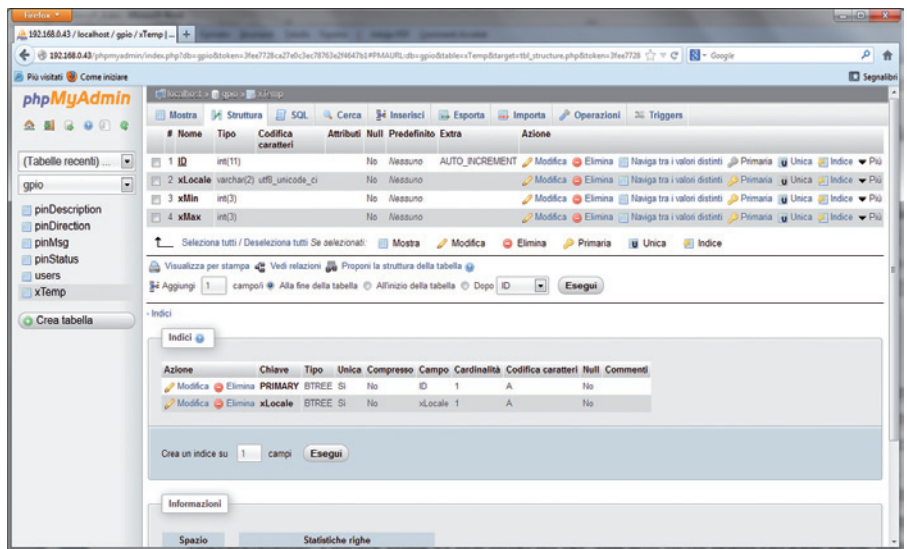


Fig. 10



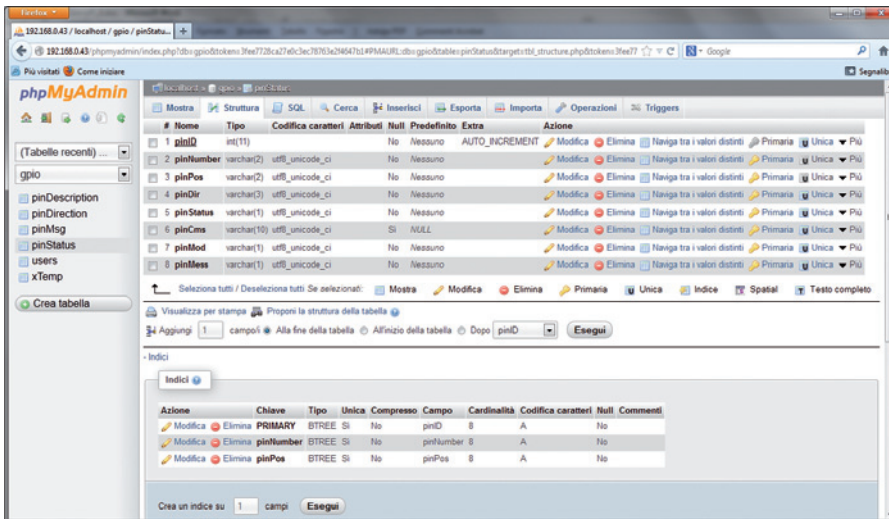


Fig. 11

```
x_LEDCOND = 0
cur2 = con.cursor(mdb.cursors.DictCursor)
cur2.execute("""UPDATE pinStatus SET pinStatus=%s , pinMod=%s WHERE pinNumber=%s""", (x_LEDCOND, "1", "24"))
cur2.execute("""UPDATE pinStatus SET pinStatus=%s , pinMod=%s WHERE pinNumber=%s""", (x_LEDCOND, "1", "23"))
con.close()
```

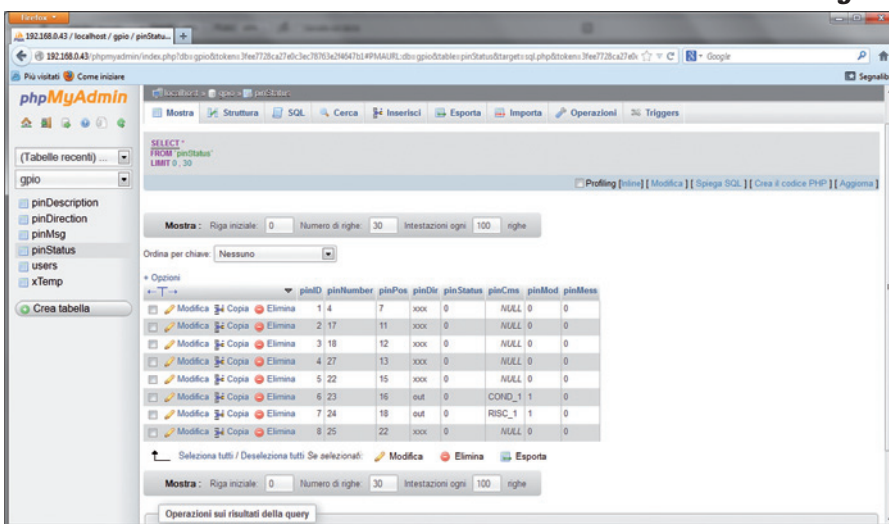


Fig. 12

```
requests.post(URL_EMONCMS+"/api/post?apikey="+ EMONCMS_API_KEY + "&json={TEMP_1:"+str(Celsius)+"}")
requests.post(URL_EMONCMS+"/api/post?apikey="+ EMONCMS_API_KEY + "&json={LUX_1:"+str(Lux)+"}")
```

MODIFICHIAMO IL PROGRAMMA GPIOSERVER3.PY

Anche questo programma è la copia del server demone originale GPIOServer2.py. Con l'occasione abbiamo aggiornato la libreria RPi.GPIO con l'ultima versione, che ci permette di gestire i pin con la numerazione corretta, come prevista dallo schema GPIO e non con la posizione del pin sul connettore come nelle versioni precedenti. La libreria è entrata a far parte delle distribuzioni ufficiali e quindi la possiamo installare con il comando:

```
apt-get install python-rpi.gpio
```

La nuova versione della libreria richiede di dichiarare quale numerazione si vuole utilizzare:

```
GPIO.setmode(GPIO.BCM)
```

per utilizzare la numerazione GPIO oppure la dichiarazione:

```
GPIO.setmode(GPIO.BOARD)
```

per utilizzare la posizione del pin sul connettore. La seconda differenza è la necessità, per evitare segnalazioni di errore, di passare il numero del pin come

seconda delle condizioni riscontrate vengono impostati i valori dei LED. Viene anche impostato a "1" il valore del pinMod in modo da istruire il programma GPIOServer3.py di eseguire la variazione fisica dello stato del pin e di inviare il messaggio JSON sullo stato dei LED stessi al server emoncms. Abbiamo lasciato tutta la gestione dei pin del GPIO al server GPIOServer3.py per evitare problemi di concorrenza tra i programmi sull'accesso al GPIO. Ricordiamo che il server GPIOServer3.py tiene aperti i file descrittori del GPIO in permanenza, impedendo ad altri programmi di accedervi. Come ultime istruzioni vengono eseguiti i metodi requests che ag-

giornano i valori di temperatura e luminosità nel server emoncms.

```
con = mdb.connect(DbServer, DbUser, DbPasswd, 'gpio');
cur1 = con.cursor(mdb.cursors.DictCursor)
cur1.execute("SELECT * FROM xTemp WHERE xLocale = 1")
rows = cur1.fetchall()
for row in rows:
    T_min = row ["xMin"]
    T_max = row ["xMax"]
    if Celsius <= T_min:
        x_LEDTRISC = 1
    else:
        x_LEDTRISC = 0
    if Celsius >= T_max:
        x_LEDCOND = 1
    else:
```

un valore numerico intero, come per esempio:

```
X_pinNumber = int(row["pinNumber"])
```

Nel **Listato 1** vediamo le altre modifiche importanti.

La costante URL_EMONCMS dev'essere personalizzata con l'indirizzo IP del vostro RaspberryPi. La costante EMONCMS_API_KEY dev'essere inizializzata con il valore dell'APIkey caratteristica dell'installazione di emoncms, come vedremo nel paragrafo dedicato alla sua installazione. L'altra aggiunta sono le istruzioni di invio del messaggio JSON al server emoncms.

I nomi dei canali di Input al server sono contenuti nei campi pinCms della tabella pinStatus.

FRAMEWORK EMONCMS

In questo articolo abbiamo scelto di presentarvi l'utilizzo del framework emoncms, un framework di gestione e rappresentazione di dati provenienti da sensori, per diverse ragioni:

- Permette di ottenere un buon risultato funzionale e grafico in modo abbastanza semplice;
- È un sistema completo di funzioni applicative, di interfaccia grafica e di database dedicato;
- La sua interfaccia grafica può essere "incorniciata" ed integrata in pagine web "proprietarie";
- La personalizzazione dell'applicazione e dell'interfaccia grafica avviene con un'applicazione web che offre un sistema di programmazione interattivo "grafico" abbastanza agevole e "didattico". Molte applicazioni, anche destinate al mondo business si stanno orientando verso questo tipo di approccio, come il CMS Sugar, per fare un solo esempio;
- Si appoggia ad Apache2 come

Listato 1

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# GPiOServer3.py

import time
import RPi.GPIO as GPIO
import MySQLdb as mdb
import os
import requests

DbServer = "localhost"
DbUser = "gpio"
DbPasswd = "pi"

URL_EMONCMS = 'http:// ://<IP-RaspberryPI>/emoncms/'
EMONCMS_API_KEY = <APIkey dell'installazione emoncms>

X_pinNumber = 99
X_pinStatus = 0
X_pinDirection = "zzz"
X_pinMod = 0
X_pinZero = 0

GPIO.setmode(GPIO.BCM)

while True:

    con = mdb.connect(DbServer, DbUser, DbPasswd, 'gpio');

    with con:

        curl = con.cursor(mdb.cursors.DictCursor)
        curl.execute("SELECT * FROM pinStatus WHERE pinDir = %s", ("out"))

        rows = curl.fetchall()
        for row in rows:

            X_pinNumber = int(row["pinNumber"])
            X_pinStatus = row["pinStatus"]
            X_pinMod = row["pinMod"]
            X_pinDirection = row["pinDir"]
            X_pinCms = row["pinCms"]
            X_pinMess = row["pinMess"]

            if X_pinDirection == "out":
                #print X_pinNumber
                GPIO.setup(X_pinNumber, GPIO.OUT)
                if X_pinStatus == "1":
                    GPIO.output(X_pinNumber, True)
                else:
                    GPIO.output(X_pinNumber, False)
                if X_pinCms is not None:
                    if X_pinMod == "1":
                        a=1
                        requests.post(URL_EMONCMS+"/api/post?apikey=" + EMONCMS_API_KEY +
                                      "&json="+str(X_pinCms)+":"+str(X_pinStatus)+")")
                    if X_pinMess == "1":
                        cur3 = con.cursor(mdb.cursors.DictCursor)
                        cur3.execute("SELECT * FROM pinMsg WHERE pinNumber = %s", (X_pinNumber))

                        if cur3.rowcount > 0:
                            rows3 = cur3.fetchall()

                            for row3 in rows3:

                                if X_pinStatus == "1":
                                    parla=row3["pinMsgON"]
                                else:
                                    parla=row3["pinMsgOFF"]

                                os.system('espeak "' + parla + '" -v it -p 70 -s 155 > _/dev/null 2> /dev/null')

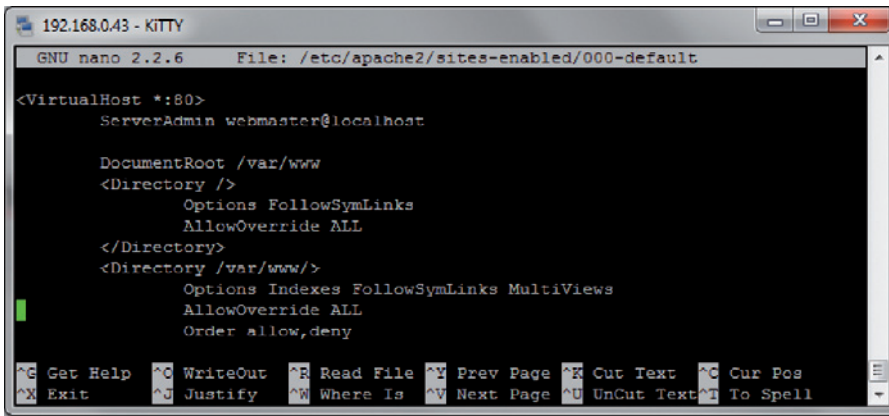
                        cur4 = con.cursor(mdb.cursors.DictCursor)
                        cur4.execute("UPDATE pinStatus SET pinMod=%s, pinMess=%s WHERE pinNumber=%s", (X_pinZero, X_pinZero, X_pinNumber))

            # print X_pinNumber, X_pinDirection, X_pinStatus, X_pinMod

    if con:
        con.close()
    time.sleep(0.5)
```

server web e MySQL come server database, che sono anche le scelte "architeturali" adottate per lo sviluppo dei progetti presentati sulla rivista;

- È dotato di un'interfaccia di raccolta dati basata su API web che utilizzano il protocollo JSON, che permette una interfacciabilità ottimale con



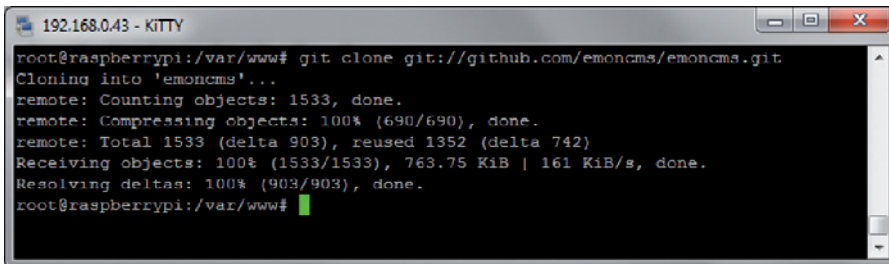
```
192.168.0.43 - KITTY
GNU nano 2.2.6 File: /etc/apache2/sites-enabled/000-default

<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www
    <Directory />
        Options FollowSymLinks
        AllowOverride ALL
    </Directory>
    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride ALL
        Order allow,deny
    </Directory>
</VirtualHost>

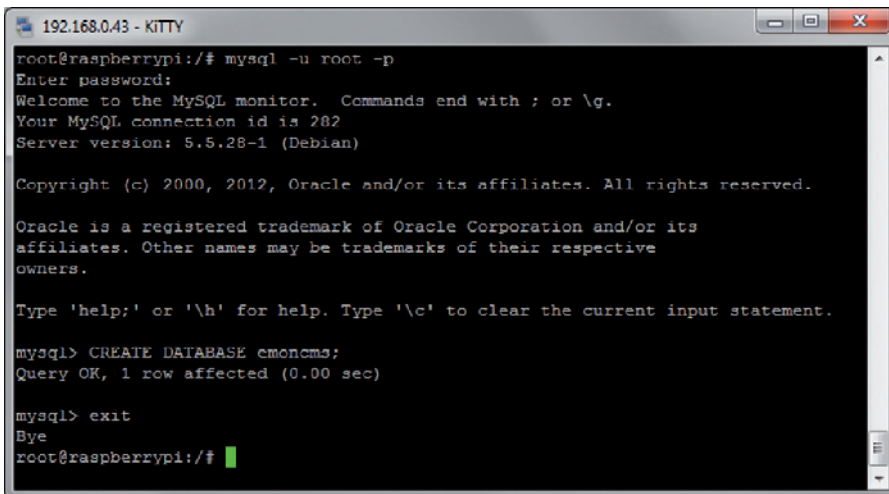
^G Get Help ^C WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Fig. 14



```
192.168.0.43 - KITTY
root@raspberrypi:/var/www# git clone git://github.com/emoncms/emoncms.git
Cloning into 'emoncms'...
remote: Counting objects: 1533, done.
remote: Compressing objects: 100% (690/690), done.
remote: Total 1533 (delta 903), reused 1352 (delta 742)
Receiving objects: 100% (1533/1533), 763.75 KiB | 161 KiB/s, done.
Resolving deltas: 100% (903/903), done.
root@raspberrypi:/var/www#
```

Fig. 15



```
192.168.0.43 - KITTY
root@raspberrypi:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 282
Server version: 5.5.28-1 (Debian)

Copyright (c) 2000, 2012, Oracle and/or its affiliates. All rights reserved.

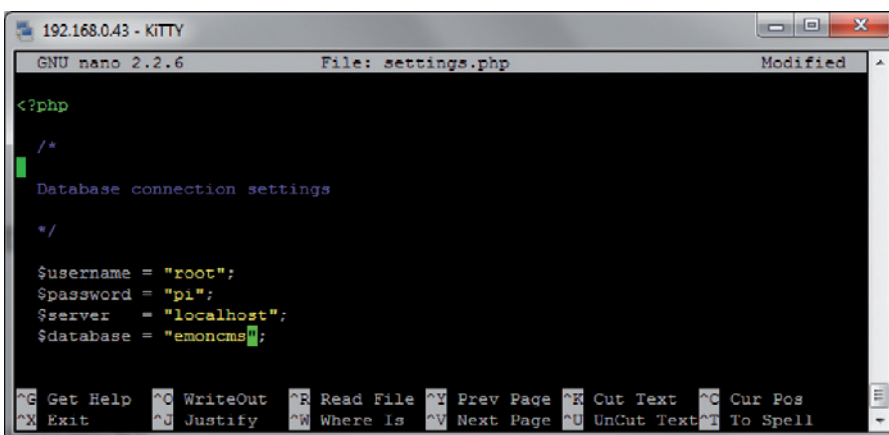
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE emoncms;
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye
root@raspberrypi:/#
```

Fig. 16



```
192.168.0.43 - KITTY
GNU nano 2.2.6 File: settings.php Modified

<?php
/*
Database connection settings
*/

$username = "root";
$password = "pi";
$server = "localhost";
$database = "emoncms";

^G Get Help ^C WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Fig. 13

pressoché qualsiasi computer, embedded e non, e con pressoché ogni microcontrollore;

- Emoncms segue l'architettura MVC (Model-View-Controller), ormai diffusissima nello sviluppo di piattaforme e applicazioni web in tutti i campi e nella maggior parte dei linguaggi;
- Infine presenta un particolare metodo per l'aggiornamento della base dati a passi o a scatti, che descriviamo in seguito e che è adottato nelle applicazioni real time che impiegano processori e metodi di calcolo "paralleli".

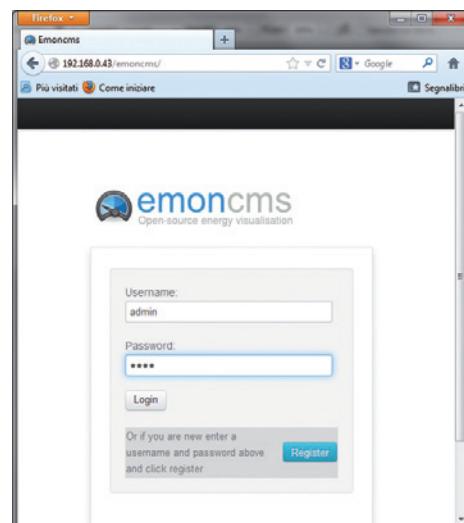
Dopo l'installazione e la configurazione iniziale, se i server web e database sono avviati, emoncms è pronto a funzionare.

Per aggiornare le tabelle dei campionamenti dati è necessario che la o le applicazioni che gestiscono i sensori inviino i dati ad emoncms utilizzando la tecnica web api con il protocollo JSON. Semplificando, questo metodo consiste in una chiamata HTML fatta dall'applicazione al server emoncms dove, nella parte "action" dell'URI, viene inserita una lista di coppie nel formato variabile=valore; il campo variabile identifica il sensore del quale si vuole inviare la misura ed il campo valore contiene l'ultimo valore da trasferire.

Non è necessario configurare in anticipo tabelle di database o altre opzioni per contenere i dati provenienti dai sensori.

La prima chiamata JSON per un nuovo sensore da gestire (Input, nella terminologia emoncms), inizializza automaticamente le tabelle ed i campi necessari per la gestione. Da questo momento si può utilizzare lo strumento di costruzione grafica per configurare il tipo di trattamento al

Fig. 17



quale si desidera vengano sottoposti i dati e la rappresentazione grafica che si vuole ottenere. Per esempio vorremmo costruire un grafico che presenti i dati storici pervenuti ed un processo di calcolo che produca la media dei dati in un periodo temporale. Alla fine organizziamo tutto in una pagina grafica con indicatori a lancetta, LED, indicatori a cilindro, grafici, valori numerici e campi testuali. Come abbiamo anticipato, l'aggiornamento dei dati avviene con un processo a scatti che funziona nel modo seguente. L'aggiornamento dei dati su emoncms per i dati provenienti da sensori avviene in base ad un frequenza di campionamento definita: ad esempio una misura di temperatura ogni cinque minuti. La prima misura che arriva aggiorna il database con il nuovo dato, se è prevista una presentazione grafica, anche questa viene aggiornata su una eventuale pagina web aperta. La seconda misura che arriva segue il processo precedente, ma in più, viene innescato un processo che porta la "prima" misura ad aggiornare il grafico e la sua presentazione grafica, se prevista. La terza misura che arriva aggiorna il valore corrente, innesca l'aggiornamento del grafico con la "seconda" misura ed innesca il processo per il calcolo del valore medio tra le prime due misure. E così di seguito. Si può dire che le misure in arrivo, in successione, "spingono" in avanti nella catena di elaborazione e calcolo le misure precedenti. Questa architettura, nello sviluppo di applicazioni informatiche è utilizzata moltissimo in applicazioni critiche, realtime, che "girano" su sistemi di calcolo parallelo. In questi sistemi, ciascun ciclo di elaborazione deve avere una durata inferiore all'intervallo di

campionamento dei fenomeni in esame. I valori vengono poi memorizzati in shift register, stack o pipe per essere elaborati nei cicli successivi magari da processori differenti per ciascuna operazione. Nella programmazione dei controllori FPGA, per esempio, si adotta comunemente questa architettura. È ora di installare emoncms sul nostro RaspberryPi. I prerequisiti per l'installazione di emoncms sono i pacchetti *apache2, php5 libapache2-mod-php5 mysql, install php5-mysql*. Fortunatamente sono già stati tutti installati da tempo sul nostro RaspberryPi e possiamo passare ad abilitare la funzionalità mod rewrite per il server Apache2, un nome criptico per indicare fondamentalmente la possibilità di creare diversi web sullo stesso server e semplificare gli url per raggiungerli. Il comando è:

```
a2enmod rewrite
```

bisogna anche modificare il parametro AllowOverride del sito di default apriamo il file di configurazione con:

```
nano /etc/apache2/sites-enabled/000-default
```

e modifichiamo, come in Fig. 13, le linee 7 e 11 del modulo da:

```
AllowOverride None
```

a:

```
AllowOverride All
```

Salviamo e usciamo con CTRL/X, rispondiamo "Y" alla conferma del file e Invio.

Riavviamo il server Apache2 per fargli acquisire le modifiche:

```
service apache2 restart
```

Ora possiamo concentrarci sull'installazione del framework emoncms. Emoncms non è un pacchetto che fa parte (almeno per ora) dei repository ufficiali e dovremo reperirlo dal repository di sviluppo. Il repository del pacchetto emoncms è gestito con il sistema di controllo versione distribuito Git, sviluppato dallo stesso Linus Torvalds. Il sistema è utilizzato per il controllo versioni nello sviluppo del kernel di GNU/Linux e di moltissimi altri progetti. Il nome non è particolarmente significativo delle funzionalità espresse, in gergo identifica una persona stupida. Una delle versioni sull'origine del nome si riferisce alle difficoltà di utilizzo della prima release che faceva sentire stupidi gli utilizzatori. La cosa più conveniente da fare è installare il pacchetto Git, che ci verrà utile in molte altre occasioni e poi utilizzarlo per scaricare l'ultima versione di emoncms. Git ci permetterà anche di aggiornare successivamente il pacchetto alle nuove versioni. Installiamo Git con il comando:

```
apt-get install git-core
```

portiamoci nella cartella var con:

```
cd /var/
```

e impostiamo i diritti di proprietà sulla cartella www per l'utente

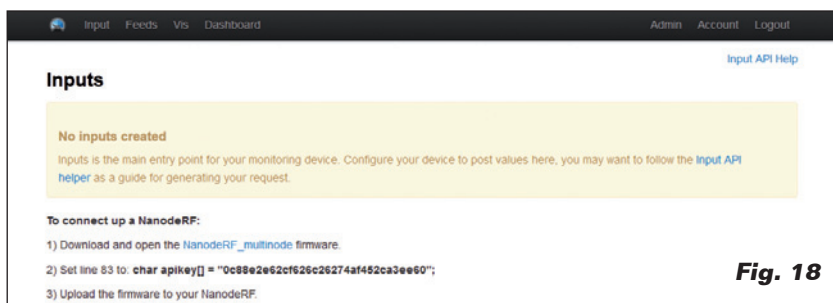


Fig. 18

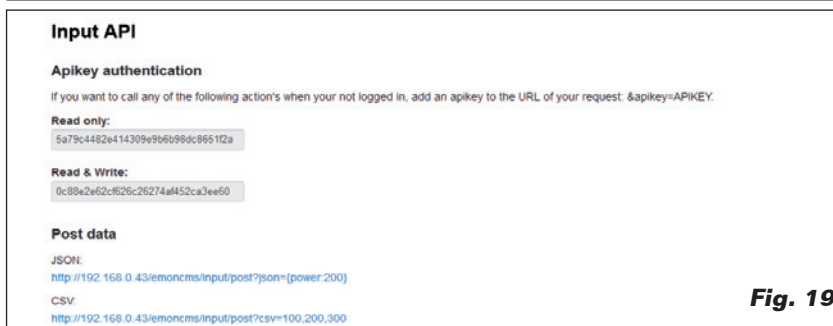


Fig. 19

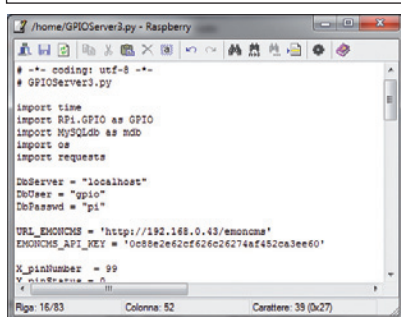


Fig. 20

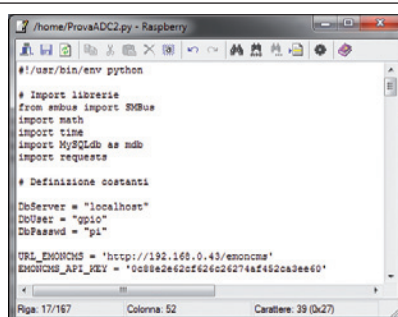


Fig. 21

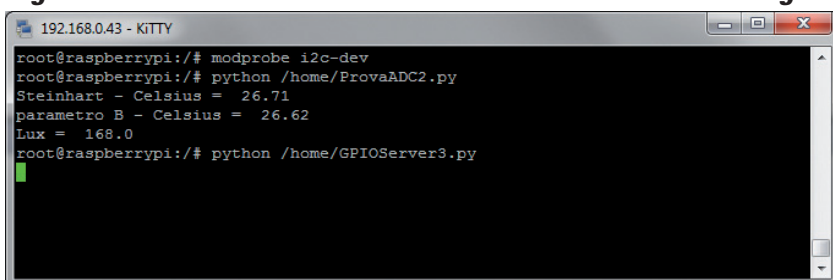


Fig. 22

pi con il comando chown (change owner):

```
chown pi www
```

entriamo nella cartella www con:

```
cd www
```

e infine scarichiamo l'ultima versione di emoncms (Fig. 14):

```
git clone git://github.com/emoncms/emoncms.git
```

Ora creiamo il database MySQL di supporto ad emoncms utiliz-

zando MySQL a linea di comando (Fig. 15):

```
mysql -u root -p
```

Autentichiamoci con la password di amministratore e creiamo il database con l'istruzione a linea di comando:

```
mysql> CREATE DATABASE emoncms;
```

Usciamo da MySQL:

```
mysql> exit
```

Nella cartella /var/www/emoncms troviamo il file di configurazione default.settings.php; in GNU/Linux si usa spesso l'approccio di chiamare il file di configurazione dei pacchetti con il prefisso "default". Il vero file di configurazione nel nostro caso si deve chiamare settings.php. Questo comportamento costringe a fare una copia del file predefinito e garantisce una certa sicurezza in caso di configurazioni poco accorte. Dunque entriamo nella cartella emoncms con:

```
cd /var/www/emoncms/
```

copiamo il file di configurazione con il nome corretto:

```
cp default.settings.php settings.php
```

Apriamo e configuriamo il file settings.php con il comando:

```
nano settings.php
```

Inseriamo i parametri di connessione e di autenticazione per il database di emoncms (Fig. 16):

```
$username = "root";
$password = "pi";
$server = "localhost";
$databse = "emoncms";
```

chiudiamo con CTRL/X, confermiamo con Y e invio.

Ora possiamo aprire un browser e collegarci alle funzioni di amministrazione di emoncms per le prime configurazioni.

Dal browser ci colleghiamo ad emoncms con l'URL:

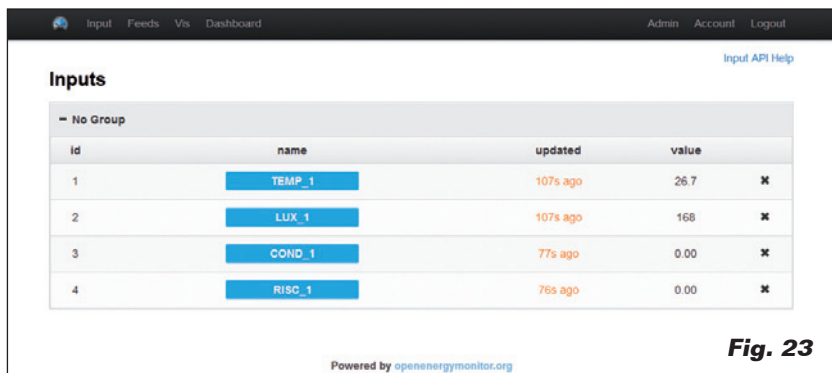
```
http://<IP-RaspberryPi>/emoncms
```

La prima volta che ci connettiamo al server emoncms viene creata automaticamente la struttura del database del quale precedentemente avevamo solo creato il "contenitore". Al termine verrà presentata la pagina di login.

Per creare un utente basta inserire un nome (non necessariamente un'e-mail) e una password nei campi previsti e poi premere il pulsante "Register". Noi abbiamo utilizzato l'utente "admin" e password "gpio" come in Fig. 17.

PERSONALIZZIAMO IL SERVER EMONCMS

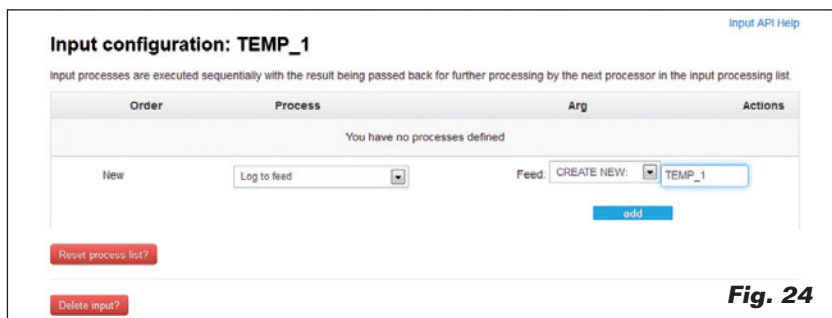
Per fare in modo che il server emoncms possa gestire i valori dei sensori e lo stato dei LED che vogliamo visualizzare nell'area grafica della pagina web è necessario creare degli Input, che sono i canali di alimentazione dei dati provenienti dalle chiamate JSON dei singoli programmi. In emoncms, per creare un Input è sufficiente inviare il primo messaggio JSON relativo all'Input che si vuole creare. Visto che abbiamo i programmi pronti, possiamo eseguirli, ciascuno una volta, e trovarci tutti gli Input generati correttamente. Facciamo una pausa per descrivere la notazione JSON (JavaScript Object Notation): non è un linguaggio ma un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi. JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C, come C, C++, C#, Java, JavaScript, Perl, Python, e molti altri. Questa caratteristica fa di JSON uno strumento ideale per lo scambio di dati. Un messaggio JSON assume la forma di una lista ordinata di coppie di nomi/valori, ciascun nome è separato, dal rispettivo valore, da un segno ":" (due punti) e le coppie di valori sono separate tra loro da virgole. Niente di complicato. No?



Inputs			
- No Group			
id	name	updated	value
1	TEMP_1	107s ago	26.7
2	LUX_1	107s ago	168
3	COND_1	77s ago	0.00
4	RISC_1	76s ago	0.00

Powered by openenergymonitor.org

Fig. 23



Input configuration: TEMP_1

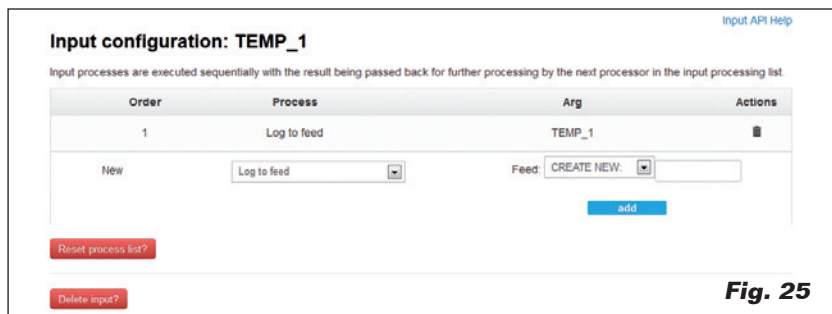
Input processes are executed sequentially with the result being passed back for further processing by the next processor in the input processing list.

Order	Process	Arg	Actions
You have no processes defined			

New Log to feed Feed: CREATE NEW: TEMP_1 add

Reset process list? Delete input?

Fig. 24



Input configuration: TEMP_1

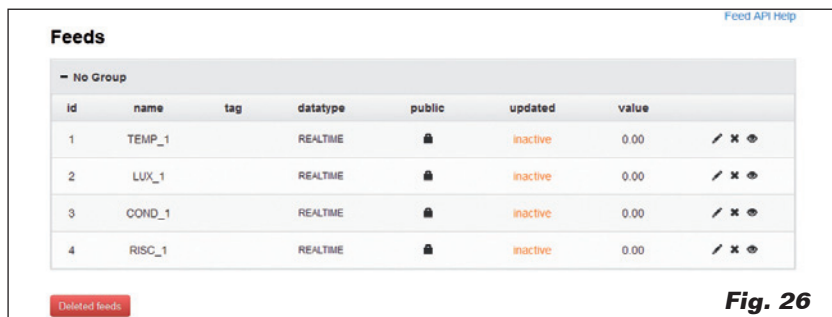
Input processes are executed sequentially with the result being passed back for further processing by the next processor in the input processing list.

Order	Process	Arg	Actions
1	Log to feed	TEMP_1	

New Log to feed Feed: CREATE NEW: add

Reset process list? Delete input?

Fig. 25



Feeds						
- No Group						
id	name	tag	datatype	public	updated	value
1	TEMP_1		REALTIME		inactive	0.00
2	LUX_1		REALTIME		inactive	0.00
3	COND_1		REALTIME		inactive	0.00
4	RISC_1		REALTIME		inactive	0.00

Deleted feeds

Fig. 26

Quindi, ora, con i Server Apache2 e MySQL attivi, apriamo un browser e puntiamo alla pagina di amministrazione del server emoncms, tipicamente <http://<indirizzo-IP>/emoncms>. Ci vengono richiesti utente e password che abbiamo impostato al momento dell'installazione, ed entriamo. Ci viene proposta la pagina di amministrazione dove possiamo cambiare la password e altre

impostazioni, la saltiamo per andare al pannello Input e clicchiamo sul link in alto a destra "Input API Help" che ci apre una pagina dove troviamo due lunghe stringhe di dati, le chiavi API per utilizzare le funzionalità emoncms in lettura e scrittura (Fig. 18). Useremo l'Apikey in scrittura come chiave di autenticazione all'interno dei messaggi JSON inviati dai programmi di gestione dei sensori e l'Apikey

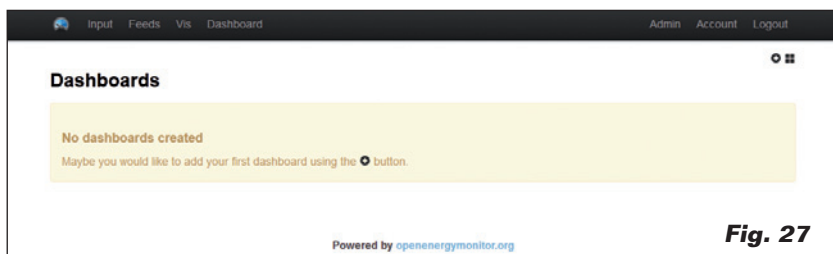


Fig. 27



Fig. 28



Fig. 29

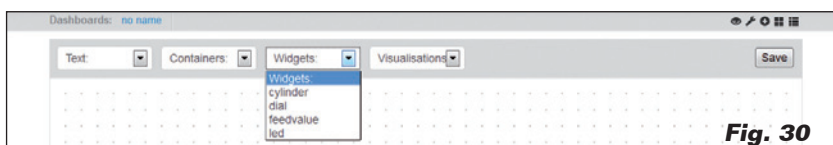


Fig. 30

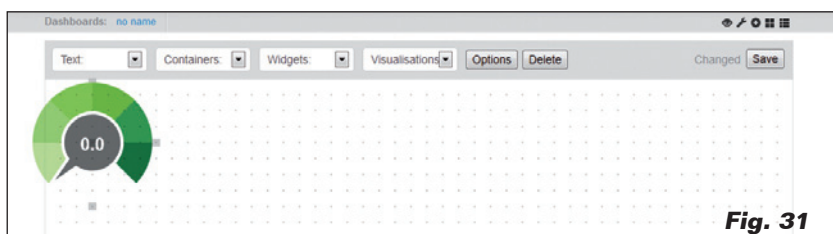


Fig. 31

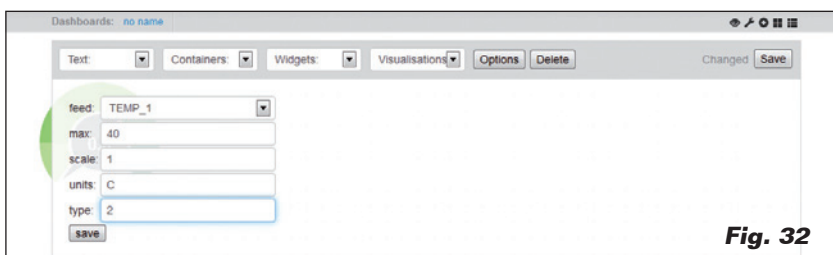


Fig. 32

in lettura come autenticazione per l'accesso alla pagina grafica dall'interno della pagina web. Torniamo ad aprire il pannello Input, che ci apparirà vuoto. Apriamo i programmi ProvaADC2.py e GPiOSeRver3.py e inseriamo l'Apikey di scrittura con copia e incolla nell'apposita variabile. Inseriamo anche l'indirizzo IP corretto di RaspberryPi (Fig. 20 e 21).

Ora dopo aver verificato che tutti i collegamenti su RaspberryPi siano corretti, lanciamo i programmi python uno ad uno, ProvaADC2.py e GPiOSeRver3.py. Prima di lanciare il programma ProvaADC2.py carichiamo il driver per il bus i2c (Fig. 22):
`modprobe i2c-dev`
e poi:
`python /home/ProvaADC2.py`
e:
`python /home/GPiOSeRver3.py`

Quest'ultimo resta in esecuzione e quindi lo possiamo interrompere con CTRL+C.

Torniamo sulla pagina Input, clicchiamo sul carattere di espansione "+" e, come per incanto vedremo i nomi dei nostri Input, elencati nella pagina (Fig. 23). Gli ingressi a emoncms di tipo Input creano solo il punto di ingresso per i valori, una specie di porta assegnata a ciascun canale di alimentazione. Per creare i contenitori dove i dati vengono accumulati e conservati e necessario creare dei Feed. Lo facciamo dalla pagina Input. Per ciascun Input dobbiamo creare un Feed. Clicchiamo due volte sul nome blu di un Input, per esempio TEMP_1 e ci si apre la pagina di creazione del Feed per TEMP_1, diamo un nome al Feed, per esempio lo stesso nome TEMP_1 e premiamo il pulsante ADD (Fig. 24).

Vediamo creato un Feed per il nostro Input (Fig. 25), potremmo crearne altri per lo stesso Input, ma ce ne basta uno, torniamo alla pagina Input e ripetiamo l'operazione per gli altri Input. Al termine selezioniamo la Pagina Feed e clicchiamo sull'icona "+" di espansione. Troviamo la situazione di Fig. 26.

Una volta creati tutti i Feed, possiamo passare alla composizione della pagina grafica: lo facciamo selezionando il pannello Dashboard, che ci presenta una pagina dove saranno contenuti i riferimenti alle pagine grafiche, con la scritta "No dashboards created" (Fig. 27). Per creare la nostra dashboard clicchiamo sul piccolo pulsantino rotondo in alto a destra con il segno "+" al centro. Ci appare una riga che indica il riferimento a una dashboard ancora vuota e vedremo una serie di piccoli pulsanti sotto la voce Action (Fig. 28). Se avviciniamo

Fig. 33

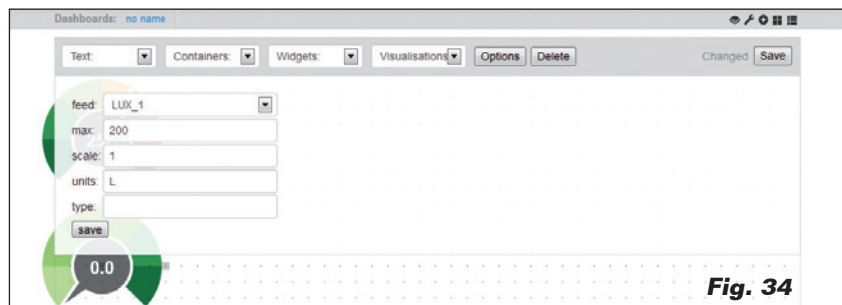
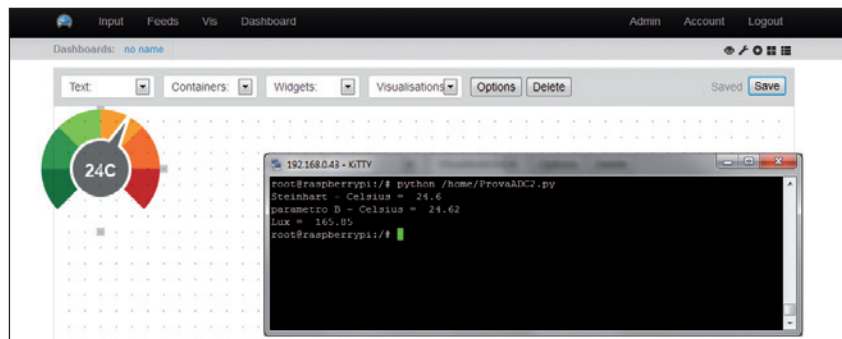
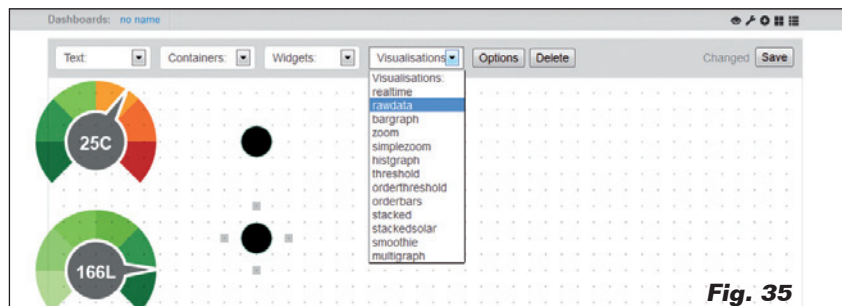
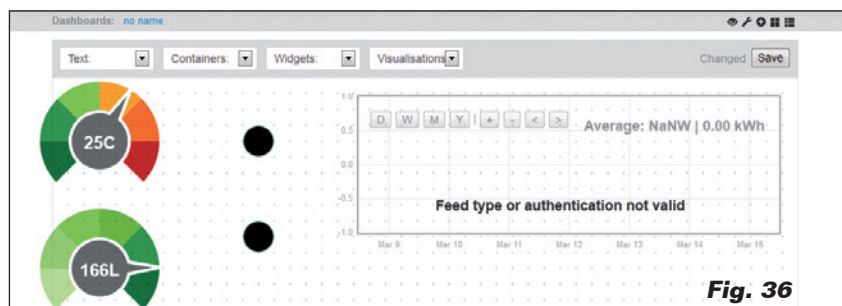
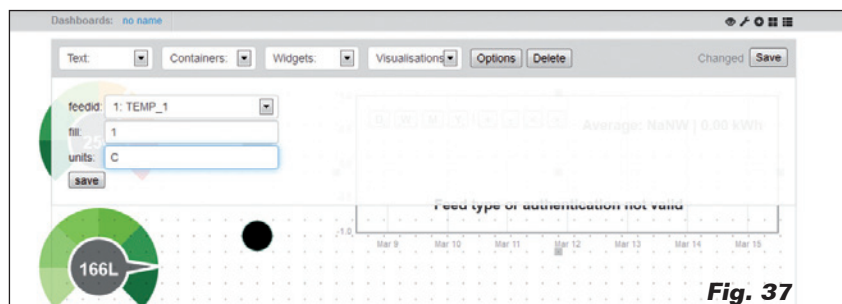
il mouse a ciascuno vedremo apparire il relativo significato. Clicchiamo subito sulle icone "Published" e "Public" in modo da rendere disponibile la nostra pagina grafica all'esterno, poi clicchiamo sul pulsante "Draw" (Fig. 29) e otteniamo un pannello vuoto dove possiamo cominciare a comporre la nostra pagina grafica (Fig. 30).

Per inserire gli elementi nella pagina abbiamo a disposizione i combo box Widgets, Text, Containers e Visualizations.

Il metodo per comporre la nostra pagina grafica consiste nello scegliere un componente dai combo box nel menù, spostare il cursore nel punto dove si vuole posizionare il componente e cliccare col tasto sinistro del mouse per fissarlo.

Poi dobbiamo associare un Feed al componente. Lo possiamo fare subito oppure successivamente selezionando il componente e premendo il pulsante Option. Se ci pentiamo, possiamo cancellare il componente selezionandolo con il mouse e premendo Delete. Ricordatevi di salvare spesso la configurazione della pagina con il pulsante Save, per evitare di perdere il lavoro fatto. I componenti si possono posizionare dove si vuole, selezionandoli e spostandoli tenendo premuto il tasto sinistro del mouse, al rilascio rimangono fissati nell'ultima posizione. Salvate. In una certa misura è anche possibile ridimensionare i componenti, ma la funzionalità non lascia grandi margini di manovra.

Iniziamo ad inserire un indicatore a lancetta per la temperatura. Scegliamo dal combo box Widgets il componente Dial e lo posizioniamo a piacere (Fig. 31). Ora selezioniamo Option e configuriamo alimentazione e apparenza. Selezioniamo il Feed TEMP_1, il

**Fig. 34****Fig. 35****Fig. 36****Fig. 37**

valore massimo Max, per esempio 40, impostiamo il fattore di scala a 1 ed il campo units a "C". Ora nel campo type possiamo scegliere l'aspetto che vogliamo

dare al Dial.

Vi diamo i valori possibili con i relativi significati, sperimentate pure, se lo desiderate:
0. Dial verde;

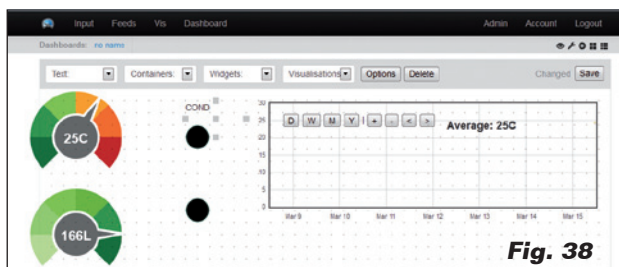


Fig. 38

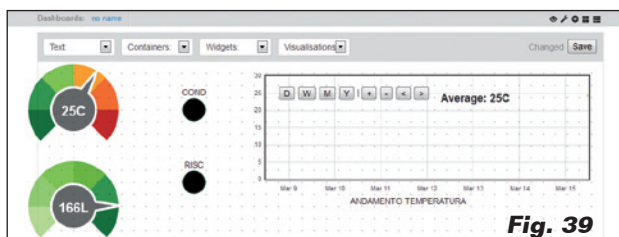


Fig. 39

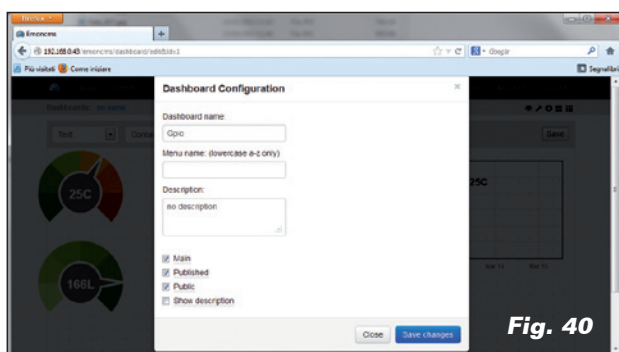


Fig. 40

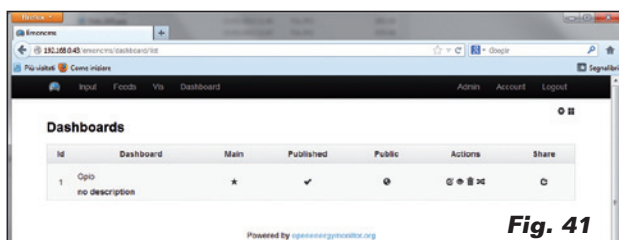


Fig. 41

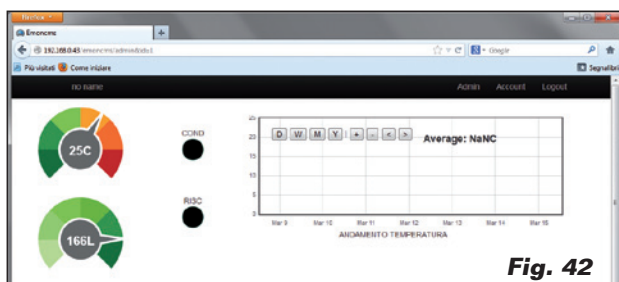
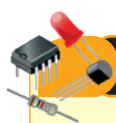


Fig. 42

1. A settori con colori dal Rosso al Verde;
 2. Dal Verde a Rosso con il valore 0 a sinistra;
 3. Dal Verde a Rosso con il valore 0 al centro;
 4. Dal Rosso al Verde con il valore 0 a sinistra;
 5. Dal Rosso al Verde con il valore 0 al centro;
 6. Centro Verde che diventa Arancione verso entrambe le estremità;
 7. Dial Blu elettrico;
 8. Da Blu a Rosso chiaro a Rosso intenso.
- Scegliamo il valore 2 (Fig. 32). Ricordatevi di salva-

re e provate a rieseguire il programma che legge la temperatura. Dovrebbe apparire il valore di temperatura al centro dell'indicatore e la lancetta portarsi in una posizione che rappresenta graficamente il valore letto (Fig. 33). Procediamo allo stesso modo per configurare l'indicatore a lancetta per la luminosità (Fig. 34). Selezioniamo due LED, posizioniamoli dove vogliamo e associamoli ai relativi Feed. Infine possiamo aggiungere un grafico dell'andamento della temperatura nel tempo. Dalla combo box "Visualizations" selezioniamo rawdata (Fig. 35), posizioniamo il grafico dove desiderate (Fig. 36) e personalizzate le opzioni, selezionate il Feed TEMP_1, con fill = 1 viene riempita l'area del grafico al di sotto della linea, e units = C (Fig. 37). Salvate le opzioni del grafico e salvate la dashboard. Usate la combo box "text" per inserire le descrizioni che preferite (Fig. 38 e Fig. 39). Possiamo impostare un nome per la dashboard cliccando sull'icona a forma di piccola chiave da meccanico. Inserite un nome nella pagina di configurazione e salviamo (Fig. 40). Salvate ancora. Da questo momento possiamo passare ad alimentare regolarmente i dati, aprire una seconda finestra di terminale lanciate in una il programma GPIOServer3.py, nell'altra lanciate più volte il programma ProvaADC2.py, a intervalli nella giornata, così potrete vedere incrementarsi il grafico della temperatura. Modificate i valori di soglia del termostato con phpMyAdmin e modificate la temperatura della NTC tenendola tra le dita. Nel prossimo numero vi presenteremo la pagina web di controllo remoto, con le funzioni di login, l'inclusione della dashboard emoncms e la possibilità di variare i valori di soglia del termostato.



per il MATERIALE

Tutti i componenti utilizzati per realizzare lo shield di espansione per RaspberryPi utilizzato in questo progetto e descritto nel numero 173 sono di facile reperibilità. Il master del circuito stampato può essere scaricato dal sito della rivista così come i programmi di esempio. Di questo shield Futura Elettronica ha realizzato la versione montata e collaudata (cod. FT1041M) disponibile al prezzo di 27 Euro, IVA compresa.

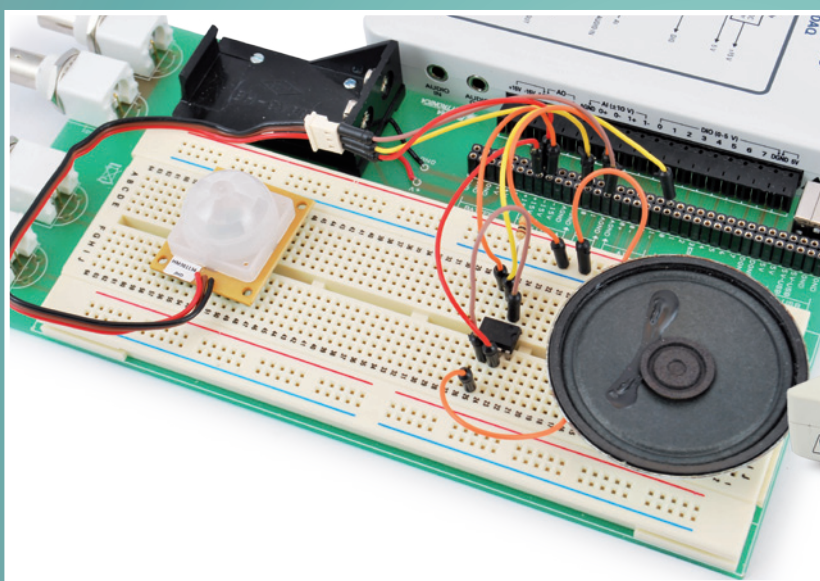
Il materiale va richiesto a:

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

myKIT, RILEVATORE DI MOVIMENTO CON SENSORE PIR

..... di ANDREA LETTIERI

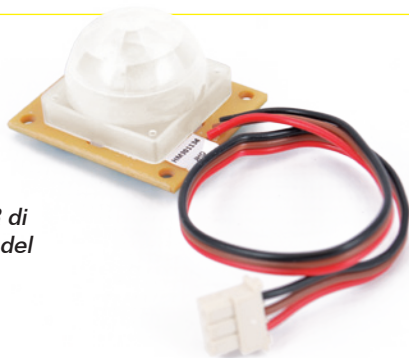
Impariamo a leggere un sensore di movimento a raggi infrarossi tramite myDAQ e LabVIEW.



In questa esperienza di laboratorio utilizzerete la scheda di acquisizione myDAQ, il software LabVIEW, un sensore di movimento PIR a basso costo ed un altoparlante con un circuito amplificatore per costruire un sistema di allarme che emette una nota sonora quando viene rilevato un movimento dal sensore PIR. L'acquisizione dei dati dal sensore e l'emissione del suono di allarme saranno realizzati con LabVIEW e DAQ Assistant, che viene installato in LabVIEW con il driver NI DAQmx. L'obiettivo è utilizzare un sensore PIR per rilevare un movimento ed attivare un ingresso digitale su myDAQ, con lo scopo di attivare un segnale sonoro

di avviso attraverso un altoparlante e un segnale luminoso sul pannello frontale del VI.

Un sensore PIR (Passive Infra-Red) è un dispositivo molto economico utilizzato per rilevare un movimento nell'ambiente circostante, in un raggio di alcuni metri. Sensori del genere sono utilizzati in molte applicazioni di robotica e di sicurezza domestica, ma sono anche impiegati negli edifici per accendere l'illuminazione al passaggio di persone e automezzi. Utilizzando il jack di uscita audio di



Sensore PIR di rilevamento del movimento.

myDAQ, è possibile emettere un suono di avviso quando viene rilevato un movimento. Il sensore PIR comunica con myDAQ per mezzo di un segnale digitale; il livello alto (+5V) corrisponde alla rilevazione di movimento. Il sensore richiede un'alimentazione in corrente continua che viene prelevata dall'uscita +15V di myDAQ, facilitando l'interfacciamento tra PIR e myDAQ. Il sensore PIR richiede l'alimentazione a +12V e la massa (GND), e presenta un collegamento di uscita digitale che può assumere i valori True e False. Anche se la tensione fornita da myDAQ è +15V, è possibile alimentare con essa il sensore perché questo dispone internamente di un regolatore di tensione che garantisce un'uscita costante a 5 V. Per pilotare un altoparlante con myDAQ in modo da produrre il segnale sonoro di allarme, è necessario interporre un circuito buffer composto da un amplificatore operazionale TL081 la cui uscita (che piloterà l'altoparlante) deve essere direttamente collegata all'ingresso invertente. In questo modo il guadagno del circuito risulta uguale a 1. Questo significa che la tensione in uscita dall'amplificatore operazionale coincide con la tensione in ingresso.

Il circuito buffer permette di fornire in uscita la corrente necessaria a pilotare l'altoparlante senza caricare l'uscita di myDAQ. Come si può notare nello schema della **Fig. 2**, la corrente all'altoparlante è fornita dall'amplificatore operazionale e non direttamente dal segnale presente sull'uscita analogica DO 0 di myDAQ, e quindi l'impedenza vista dal generatore di segnali nel caso del circuito rappresentato nella **Fig. 2** è data dal valore della resistenza R1 (10 kohm). Ora è il momento di collegare i terminali del sensore PIR sulla breadboard e di realizzare il circuito di pilotaggio dell'altoparlante visibile nella **Fig. 2**. L'interfaccia utente che creerete dovrà avere un controllo numerico per l'impostazione della frequenza del suono di avviso e un indicatore booleano che segnala con il cambio di colore se il sensore PIR ha rilevato o meno un movimento.

CODIFICA DELLA STRATEGIA

Utilizzando LabVIEW dovete fare in modo, quando viene rilevato un movimento, di creare un file contenente la forma d'onda da trasformare suc-

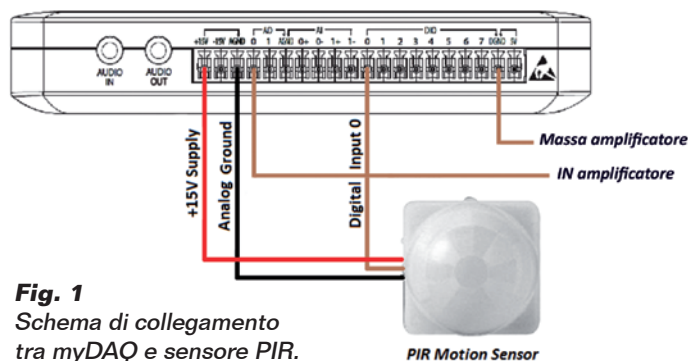


Fig. 1
Schema di collegamento
tra myDAQ e sensore PIR.

cessivamente in segnale sonoro udibile per mezzo dell'altoparlante. Il file con la forma d'onda sonora deve essere associato alle condizioni di True e False di una struttura di elaborazione case, in modo da attivare la segnalazione sonora quando si verifica la condizione True. Il diagramma a blocchi LabVIEW è molto simile allo schema a blocchi di codifica.

COME FUNZIONA

Nella parte sinistra del diagramma a blocchi, prima del ciclo While viene eseguito il blocco di generazione del segnale audio, che utilizza come valore di frequenza quello impostato nella casella di testo del Pannello Frontale. Una volta generato il file con la forma d'onda richiesta, il controllo passa al ciclo While del VI. Nel ciclo While viene letto in continuazione il segnale digitale presente sul canale 0 del registro degli ingressi digitali port0, al quale è collegata l'uscita del sensore PIR. La lettura viene eseguita utilizzando il VI array che contiene i valori di tutti gli ingressi digitali, dei quali viene tenuto in considerazione solo quello relativo al canale 0.

Il livello basso False indica che non viene rilevato alcun movimento dal sensore PIR, mentre il segnale True indica che il sensore si è attivato a seguito del rilevamento di un movimento nella zona che esso controlla. Lo stato del canale 0 viene trasferito sia al terminale dell'indicatore booleano presente sul Pannello Frontale, sia al blocco condizionale a destra nel diagramma a blocchi. L'indicatore booleano a forma di grosso LED è spento quando il segnale è False, mentre si accende con segnale True. Contemporaneamente, il livello True attiva il blocco condizionale che produce sull'uscita analogica DO 0 di myDAQ il segnale audio, quindi viene emesso dall'altoparlante un suono che riproduce la forma d'onda contenuta nel file e con frequenza impostata al valore indicato nel Pannello Frontale.

COSTRUZIONE DEL PANNELLO FRONTALE

Andate sul pannello frontale e facendo clic con il pulsante destro del mouse comparirà la palette Controls, dove sono elencati tutti i tipi di controllo e di indicatore, ma anche decorazioni, tabelle, ecc. Bloccatela

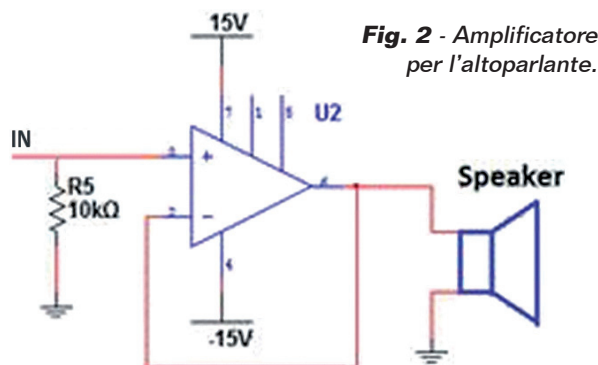


Fig. 2 - Amplificatore per l'altoparlante.

nella posizione meno ingombrante facendo clic sulla "puntina" in alto a destra della palette, in modo da non doverla richiamare ogni volta. Ora sulla palette, alla voce Modern » Numeric, selezionate un Numeric Control e trascinatelo sul pannello frontale. Con lo strumento Labeling rinominate il controllo numerico scrivendo "Frequenza sirena di allarme (Hz)" (Fig. 4). Inserite il valore di frequenza in Hz nella casella di testo: per esempio 850. Ora, sulla palette Controls alla voce Modern » Boolean selezionate Round Led e posizionate questo sul Pannello Frontale. Dal percorso Properties » Appearance selezionate Visible, Enabled, 60" come Height e Width, ed i colori verde chiaro per lo stato ON e verde scuro per lo stato OFF. Sempre sulla palette Controls, alla voce Modern » Boolean selezionate Stop Button e posizionate anche questo sul pannello frontale. Per fare in modo che i valori inseriti restino tali anche nei successivi utilizzi del VI, è opportuno selezionare dal menu in alto Edit » Make Current Values Default. Ora salvate momentaneamente il VI con il nome e la destinazione che preferite.

COSTRUZIONE DEL DIAGRAMMA A BLOCCHI

Per prima cosa, nel diagramma a blocchi si deve inserire il blocco di creazione del file con la forma d'onda da utilizzare per la creazione del segnale

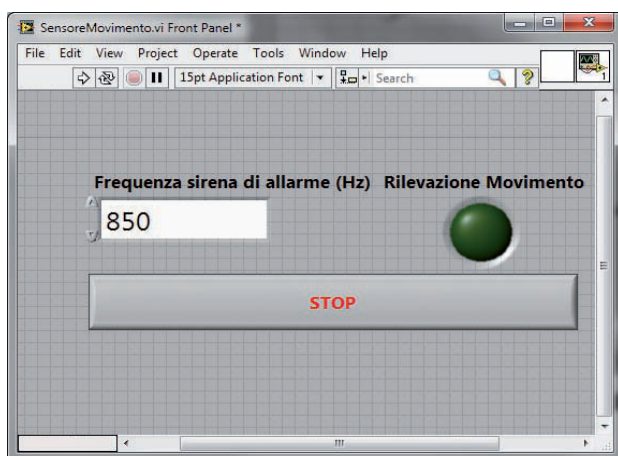


Fig. 3 - Pannello Frontale LabVIEW.



Fig. 4 - Schema a blocchi di codifica.

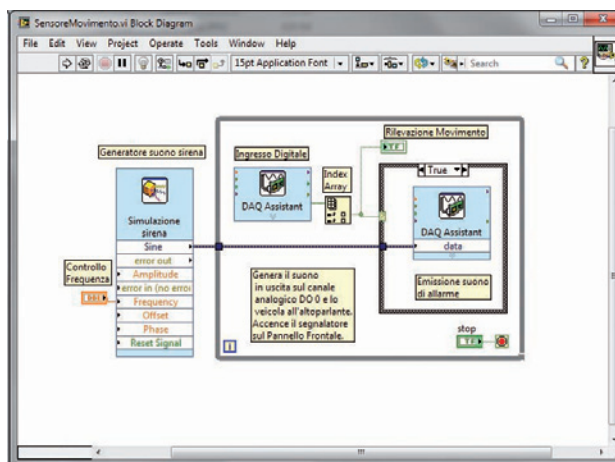


Fig. 5 - Diagramma a Blocchi LabVIEW.

audio. Anche sul diagramma a blocchi, se fate clic col pulsante destro del mouse si apre 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 in alto a sinistra. Per inserire il blocco di creazione del file, procedete con i passaggi seguenti.

1. Assicuratevi che myDAQ sia collegato al PC;
2. Premete Ctrl-Spazio per aprire la finestra Quick Drop (ciò, al primo utilizzo richiederà circa un minuto);
3. Cercate la voce Simulate Signal e fate doppio clic su di essa quando appare nella lista, oppure cliccate col tasto destro del mouse all'interno della finestra di costruzione del diagramma a blocchi, selezionate Signal Processing, Waveform Generation e fate clic sul controllo Simulate Signal;
4. Trascinate il controllo sul diagramma a blocchi;
5. Quando appare la finestra di configurazione per il controllo Simulate Signal, lasciate tutti i valori predefiniti tranne:
 - a. Sample Per Seconds, da impostare a 44.100 Hz;
 - b. Number of Signals, da impostare deselezionando Automatic;
 - c. Number of Samples, da impostare a 44.100 (in modo da generare un segnale della durata di un secondo);
6. Fate clic su OK;

PREPARARE IL WHILE LOOP

All'interno del While Loop dovete inserire un DAQ assistant necessario a intercettare il valore digitale proveniente dal canale cui è collegato il sensore PIR. Utilizzate un Index Array VI per recuperare unica-

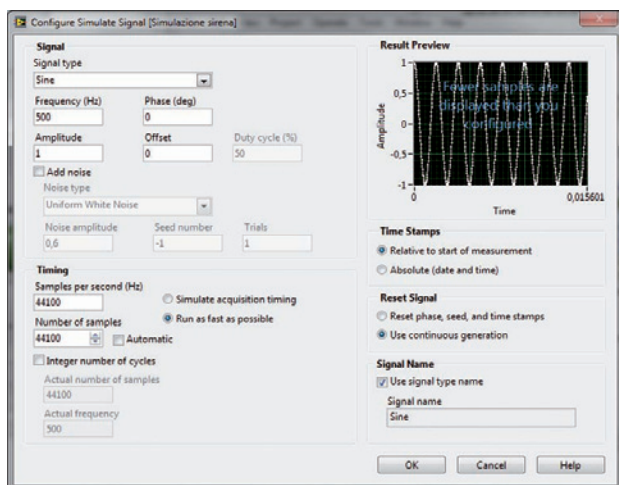


Fig. 6 - Configurazione della finestra di dialogo Simulate Signal.

mente il primo valore dall'array Booleano trasmesso da DAQ assistant, in quanto il sensore PIR è collegato alla line0, che corrisponde appunto al primo valore dell'array. Il processo prosegue con il passaggio del valore sia all'indicatore Booleano Rilevazione Movimento che al terminale selettore della struttura case. In LabVIEW, una struttura case è l'equivalente di una istruzione If/Then in un linguaggio di programmazione di tipo testuale. Il collegamento al terminale selettore corrisponde all'istruzione condizione. L'ingresso digitale di DAQ assistant è configurato per un input on-demand sul canale digitale. I passi che seguono permettono la configurazione del controllo DAQ assistant da zero.

1. Assicuratevi che myDAQ sia collegato al PC;
2. Premete Ctrl-Spazio per aprire la finestra Quick Drop.
3. Cercate la voce DAQ Assistant e fate doppio clic su di essa quando appare nella lista, oppure cliccate col tasto destro all'interno del While Loop, scegliete Measurement I/O, poi NI-DAQmx e infine cliccate su DAQ Assistant;
4. Trascinate DAQ Assistant sul diagramma a blocchi;
5. Quando appare la finestra di configurazione per il controllo DAQ Assistant selezionate:
 - a. Acquire Signals;
 - b. Digital Input;
 - c. Line Input;
 - d. myDAQ1 (o il device type che corrisponde a myDAQ sul vostro PC);
 - e. Port0/Line0;
 - f. Finish;
6. Lasciate il valore del campo Timing Settings a 1 Sample (On Demand);
7. Lasciate deselezionato il campo Invert Line;
8. Fate clic su OK.

All'interno della struttura case sono presenti un caso True ed uno False. Per il caso True dovrà essere inviato in uscita un segnale analogico al terminale di

uscita audio di myDAQ utilizzando DAQ Assistant. Il segnale in uscita è il file audio che è stato creato precedentemente nel VI. Questo si ottiene collegando il connettore blu direttamente al terminale Data input di DAQ Assistant all'interno del controllo Case True. Per il caso False non è prevista alcuna azione, in quanto non è necessario gestire la situazione di movimento non rilevato. Occorre ora inserire un controllo Case, reperibile sempre cliccando col tasto destro del mouse e selezionando Programming, Structure ed infine Case Structure. Inserite ora, dalla libreria Array, un controllo Index Array per convertire il primo elemento del vettore in uscita da DAQ Assistant, che è pilotato dallo stato del PIR, in un valore booleano per pilotare il controllo Case e il terminale del segnalatore sul pannello frontale. Inserite un ulteriore controllo DAQ Assistant all'interno del controllo Case e configuratelo come segue.

1. Quando appare la finestra di configurazione per il controllo DAQ Assistant selezionate:
 - a. Generate Signals;
 - b. Analog Output;
 - c. Voltage;
 - d. myDAQ1 (o il device type che corrisponde a myDAQ sul vostro PC);
 - e. ao0;
 - f. Finish;
2. Impostate per il controllo Signal Output Range i seguenti valori:
 - a. Max: +2V;
 - b. Min: -2V;
3. Impostate il campo Timing Settings al valore "N

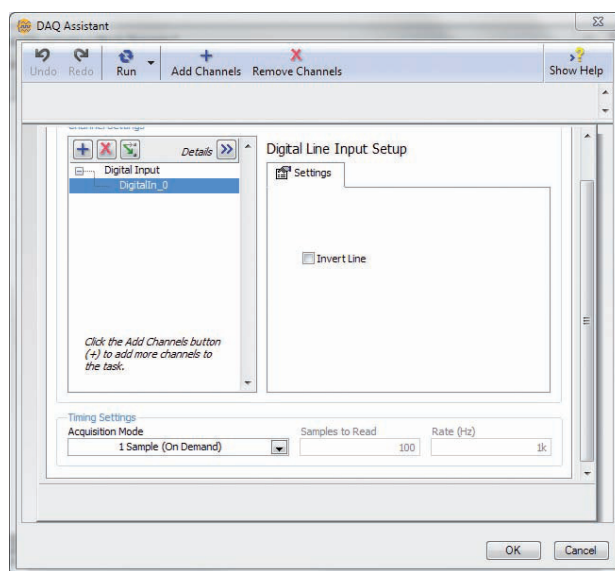


Fig. 7 - Configurazione dell'ingresso digitale di DAQ Assistant.

Fig. 9 - VI in funzione durante il rilevamento di un movimento.

Samples";
4. Fate clic su OK.

Ora, con lo strumento Wiring collegate tutti i blocchi come visibile nella **Fig. 5**. Collegate con lo strumento Wiring 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 quindi di terminare il programma. Se avete agito come appena descritto, il diagramma a blocchi dovrebbe presentarsi come nella **Fig. 5**. Potete inserire i commenti a vostro piacimento utilizzando lo strumento Labeling, in quanto essi non interferiscono con il codice grafico ma - se mai - ne aumentano la comprensione. Se preferite cambiare le impostazioni di scrittura (font, dimensione, colore, ecc.), sia nel pannello frontale sia 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 andate su File » Save as e scegliete un nome e una destinazione per salvare il vostro VI. Ora cliccate sul pulsante Run  nella barra degli strumenti per mettere il VI in esecuzione. Quando il sensore rileva un movimento viene emesso un suono dall'altoparlante con una frequenza corrispondente a quella impostata nell'apposito controllo sul Pannello Frontale e si accende l'indicatore luminoso (**Fig. 9**). 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 ori-



ginale 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. ■

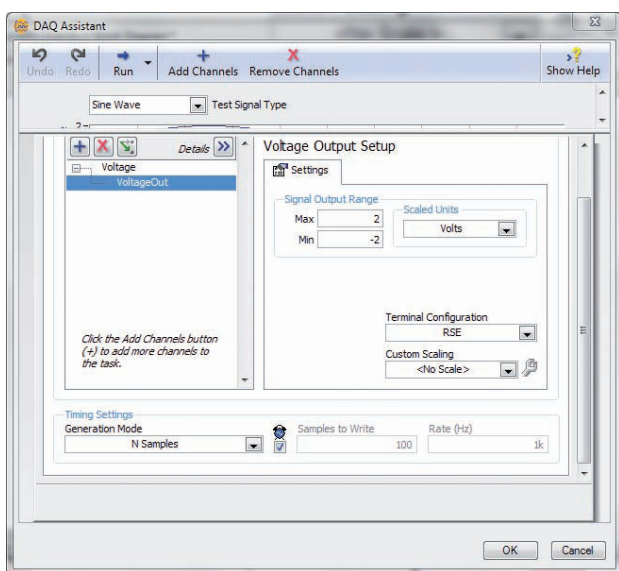
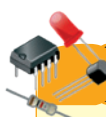


Fig. 8 - Configurazione dell'uscita analogica del DAQ Assistant.



per il MATERIALE

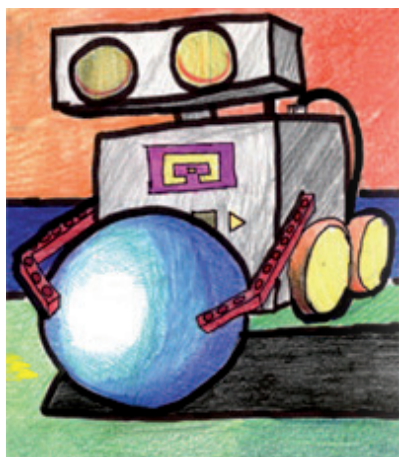
Questo progetto fa parte del set di esperimenti didattici realizzato da IRS-Elettronica per la board NI myDAQ con il supporto del software LabVIEW. Il set (cod. SETmyKIT, Euro 232,00) comprende tutti i componenti per realizzare i 22 esperimenti, i VI relativi e una protoboard. È possibile richiedere protoboard agguinate (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

DAI ROBOT PER GIOCARE AI ROBOT PER IMPARARE

Anche nel milanese un Festival della Robotica Educativa in programma per maggio: lo organizza Amico Robot, una rete di scuole locali.

a cura della REDAZIONE

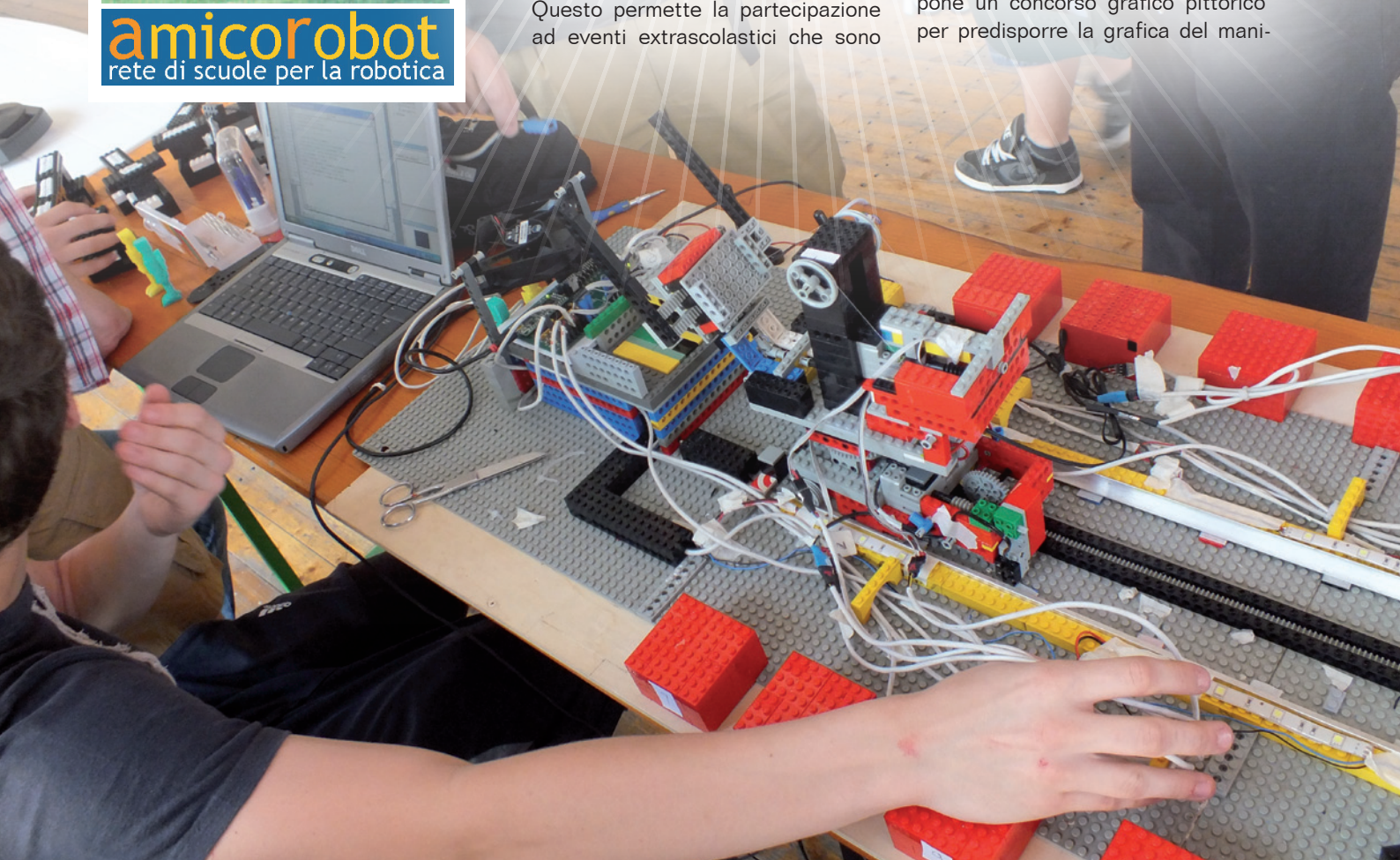


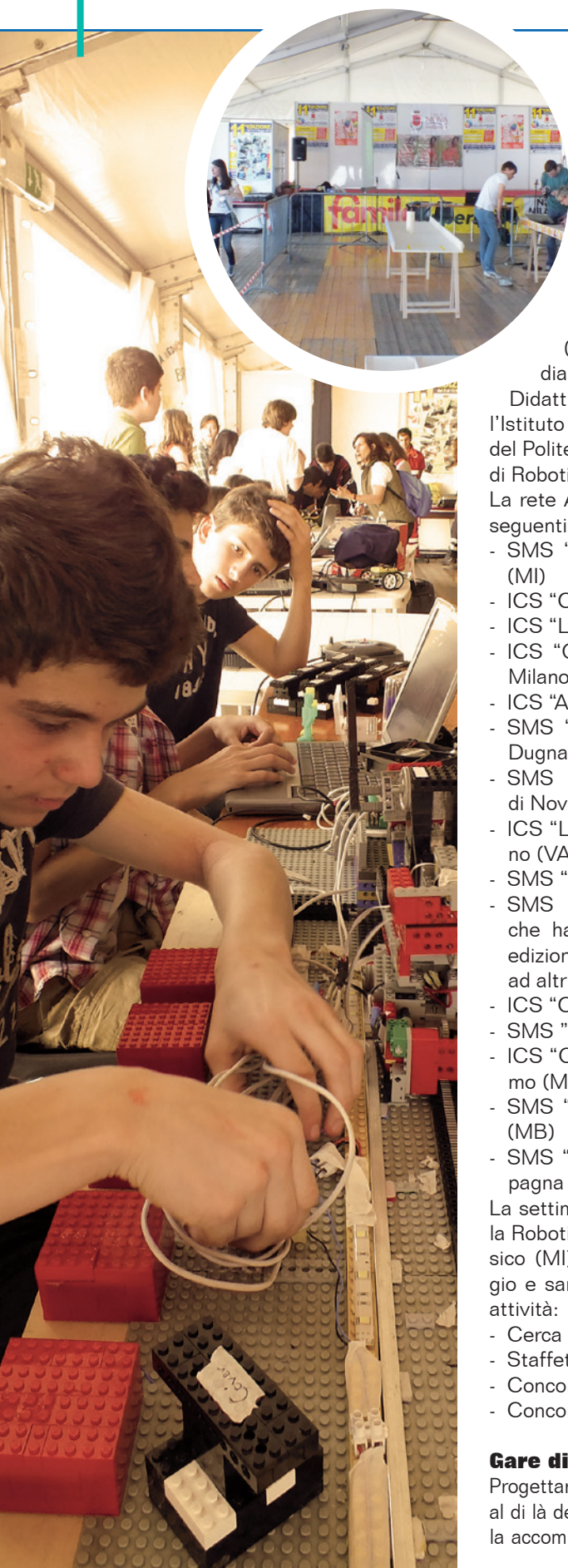
amicorobot
rete di scuole per la robotica

Da molti anni Amico Robot, una rete di scuole a cui aderiscono scuole secondarie di primo grado ed istituti comprensivi delle province di Milano e di Monza e Brianza (fra le quali la scuola secondaria di primo grado Giovanni XXIII – Segantini di Nova Milanese), organizza il Festival della Robotica Educativa, al fine di promuovere lo sviluppo della "robotica" per ragazzi in campo educativo attraverso una modalità esperienziale. Questo permette la partecipazione ad eventi extrascolastici che sono

occasione di confronto, cooperazione e competizione tra le scuole e costituisce una naturale e positiva occasione di integrazione e completamento dell'attività didattica svolta dalle singole scuole all'interno del laboratorio di robotica.

Il Festival della Robotica è articolato in diverse iniziative rivolte agli alunni delle scuole primarie e secondarie di primo grado: in particolare propone un concorso grafico pittorico per predisporre la grafica del mani-





festo che promuove il festival, un concorso letterario e gare di robotica. Il Festival viene organizzato in collaborazione con altre scuole, esterne alla rete, con istituzioni scientifiche (fra cui Ex IRRE Lombardia, l'Istituto per le tecnologie Didattiche del CNR di Genova, l'Istituto di Elettronica e Robotica del Politecnico di Milano e la Scuola di Robotica di Genova).

La rete Amico Robot comprende le seguenti Scuole:

- SMS "Manzoni-Benzi" di Bresso (MI)
- ICS "Copernico" di Corsico (MI)
- ICS "Lamarmora" di Lainate (MI)
- ICS "Quintino Di Vona-Speri" di Milano
- ICS "A. Scarpa" di Milano
- SMS "Allende-Croci" di Paderno Dugnano (MI)
- SMS "Giovanni XXIII-Segantini" di Nova Milanese (MB)
- ICS "Leonardo da Vinci" di Saronno (VA)
- SMS "Calvino" di Milano
- SMS "Rinascita-Livi" di Milano che hanno partecipato all'ultima edizione del Festival unitamente ad altri istituti della zona:
- ICS "Cantello" di Cantello (VA)
- SMS "Montale" di Carnate (MB)
- ICS "Garibaldi" di Cinisello Balsamo (MI)
- SMS "B. Luini" di Usmate Velate (MB)
- SMS "D. Alighieri" di Sommacampagna (VR)

La settima edizione del Festival della Robotica Educativa si terrà a Corsico (MI) il prossimo mese di maggio e sarà articolato nelle seguenti attività:

- Cerca la casa
- Staffetta parallela
- Concorso letterario
- Concorso grafico - pittorico

Gare di robot, come e perché

Progettare e realizzare gare di robot, al di là dell'aspetto motivazionale che la accomuna a tutte le forme e moda-

lità di "uscita dalla scuola" (in senso reale e metaforico), è particolarmente interessante per realizzare un modello di apprendimento innovativo basato sul concetto di comunità e sulla sinergia cooperazione-compete. Infatti, la partecipazione ad una gara di robot può essere vissuta come semplice uscita sul campo (dopo aver realizzato a scuola un'attività di preparazione si affronta la gara), ma può anche essere proposta e realizzata con una forte integrazione tra il dentro e il fuori, promuovendo una comunità di attori che, legati da un compito-evento nel quale saranno in competizione, si preparano ad esso in modo cooperativo. Il punto critico in questo modello non è tanto l'ambiente telematico più o meno sofisticato che costituisce lo strumento di comunicazione tra i soggetti impegnati nel compito, quanto la proposta e la modalità di lavoro. Ogni gruppo di studenti impegnato nella realizzazione del proprio autonoma, e quindi ad affrontare e risolvere i problemi che il compito posto dalla gara richiede, deve incorporare nella propria attività anche la produzione di una documentazione, periodica ma sistematica, leggera ma esplicativa, sul proprio operato (... oggi io e Giorgio abbiamo provato a ...). È la messa in comune di questa documentazione, realizzata appunto con un qualche sistema di condivisione online, che crea da un lato l'interesse ad andare a vedere cosa hanno fatto gli altri (scoprendo soluzioni e possibilità nuove o ricevendo rinforzo sul proprio operato) e dall'altro creando nei soggetti una maggior consapevolezza di quanto stanno facendo (riflettendo appunto sul proprio operato per comunicarlo ad altri). Si ottiene così il valore aggiunto di un apprendimento cooperativo (auspicabile ma non spontaneamente riproducibile) dentro un evento competitivo (la gara appunto), dove è giusto, poi che vinca il migliore.

Il progetto: nuove tecnologie e nuove pedagogie

Sembra esistere un preciso rapporto tra nuove tecnologie e nuove pedagogie. Anche se si tratta di un rapporto complesso, e non solo stru-

mentale, la prospettiva più semplice, adottabile in prima istanza, è quella di vedere le nuove tecnologie come strumenti per dare concretezza alle nuove pedagogie.

Secondo il modello papertiano dei micromondi (ambienti significativi di apprendimento e di esperienza) esiste uno stretto rapporto, anche se dialettico e circolare, tra costruzione mentale della conoscenza e manipolazione concreta degli oggetti della conoscenza stessa.

In altre parole, e tutt'altro che nuove, l'allievo impara facendo. Questa prospettiva pedagogica trova oggi una nuova frontiera con la commercializzazione del mattoncino programmabile (Lego Mindstorms Robotic Invention System), un kit di costruzione della Lego, che contiene, insieme ad altri e più tradizionali elementi, un vero e proprio computer programmabile travestito da "mattoncino" componibile con gli altri pezzi della scatola di montaggio. Il sistema Lego Mindstorms permette di progettare e costruire veri robot il funzionamento dei quali può essere determinato e controllato attraverso un programma per computer.

Se nell'epoca della robotica industriale era ragionevole considerare determinate conoscenze come specialistiche e riservarle ad indirizzi e percorsi di formazione particolari, nell'era della robotica domestica è necessario che almeno una parte di queste conoscenze entrino a far par-

te di quel sapere di base che cerca di dare a tutti gli strumenti per comprendere e orientarsi nella vita quotidiana.

A questa prima motivata ragione per introdurre la robotica a scuola, più legata alla selezione del sapere, se ne aggiunge una seconda, più di ordine metodologico, che può essere rappresentata dalla contrapposizione tra didattica d'aula e didattica di laboratorio.

A favore del primo termine c'è soprattutto l'esigenza di garantire che una quantità predefinita di conoscenze venga assimilata dagli studenti entro vincoli temporali dati (la programmazione per obiettivi, ad esempio, è una tipica risposta a questa esigenza).

Questa finalità, che appare ragionevole e condivisibile, a sua volta però deve fare i conti con risultati spesso deludenti: la sequenza lezione-esercitazione tende a produrre una conoscenza superficiale. Sequenze di informazioni che calano dall'alto non hanno tempo di sedimentarsi, di interagire con le conoscenze del soggetto che apprende, di essere oggetto di dialogo e confronto tra punti di vista diversi, di concretizzarsi in esperienze significative e inevitabilmente "scivolano come l'acqua sulla pietra".

Al contrario la modalità laboratorio cerca di promuovere una partecipazione motivata del soggetto al proprio processo di apprendimento, di



creare un rapporto attivo con la realtà e collaborativo con i compagni, per promuovere una comprensione profonda.

Questa seconda motivata ragione per occuparsi di robotica si concretizza nella possibilità di creare le condizioni di un apprendimento che sia attivo e costruttivo, contestuale e problematico, conversazionale e collaborativo, intenzionale e riflessivo, tratti e caratteri tipici, al di là delle differenze tra scuole di pensiero, del paradigma costruttivista, che è emerso con forza negli ultimi anni. L'esperienza di Amico Robot si dimostra ogni anno sempre più valida, con il raggiungimento di significativi traguardi educativi, oltre al valore dell'incontro di studenti delle scuole dell'area e questo fa ben sperare nel proseguimento della manifestazione che è affidata all'iniziativa dei docenti dei vari istituti e alla sensibilità delle amministrazioni locali per la concessione dello spazio e il supporto tecnico logistico. □

RM ELETTRONICA

Forniture per hobbisti, scuole e industria

Distributore autorizzato:

**FUTURA
ELETTRONICA**

**ELSE
Kit**

**NUOVA
ELETTRONICA**

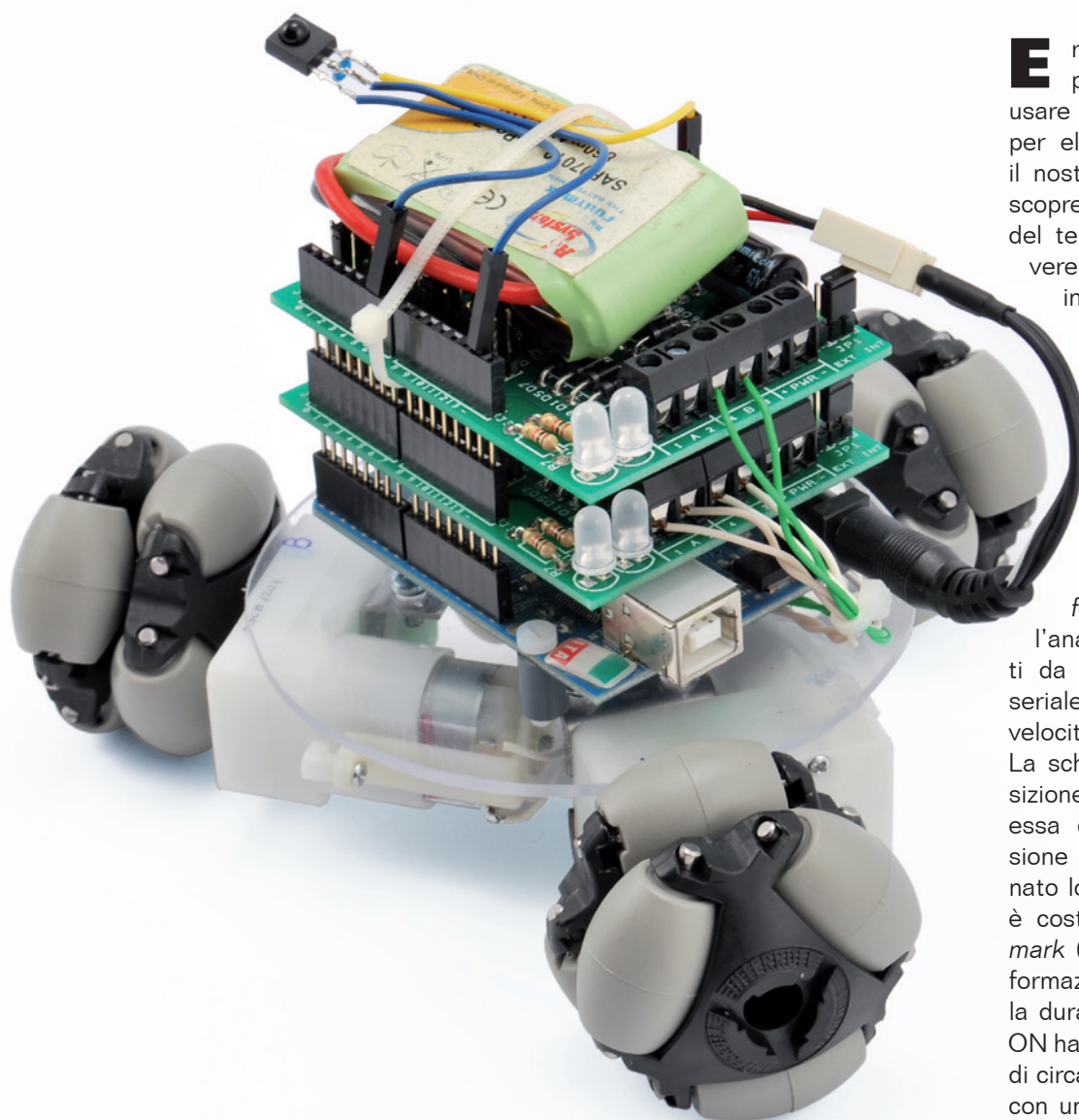
**Il tuo punto
di riferimento
per l'elettronica
a ROMA**

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

OmniWheels

In questa seconda puntata impariamo a decodificare un telecomando IR e scriviamo lo sketch per pilotare la nostra piattaforma robotica a tre ruote OmniWheels.

dell'Ing. MIRCO SEGATELLO



Eravamo rimasti, alla fine della prima parte, con la scelta di usare un telecomando a infrarossi per elicottero e ora riprendiamo il nostro progetto da quel punto, scoprendo e ricostruendo i codici del telecomando scelto. Per ricevere i segnali ottici e convertirli in impulsi elettrici ci è bastato un semplicissimo sensore tipo il PNA4602 a 38 kHz, facilmente collegabile ad Arduino. Dalla pagina web <https://github.com/adafruit/Raw-IR-decoder-for-Arduino> abbiamo scaricato lo sketch di nome *Raw-IR-decoder-for-Arduino* che consente l'analisi dei singoli bit provenienti da una qualsiasi trasmissione seriale, indipendentemente dalla velocità e dal tipo di bit inviati. La schermata relativa a un'acquisizione è riportata in **Fig. 1**; da essa deduciamo che la trasmissione avviene solo se viene azionato lo stick del gas: la sequenza è costituita da una sequenza di *mark* (ON) e *space* (OFF) e le informazioni sono codificate tramite la durata dei soli OFF, mentre gli ON hanno sempre la stessa durata di circa 400 μ s. La sequenza inizia con uno space della durata di cir-

Fig. 1
I dati ricevuti dal telecomando.

ca 1,8 ms cui seguono trentadue space la cui durata può valere circa $170 \mu s$ oppure circa $570 \mu s$; la sequenza si ripete dopo circa 20 ms.

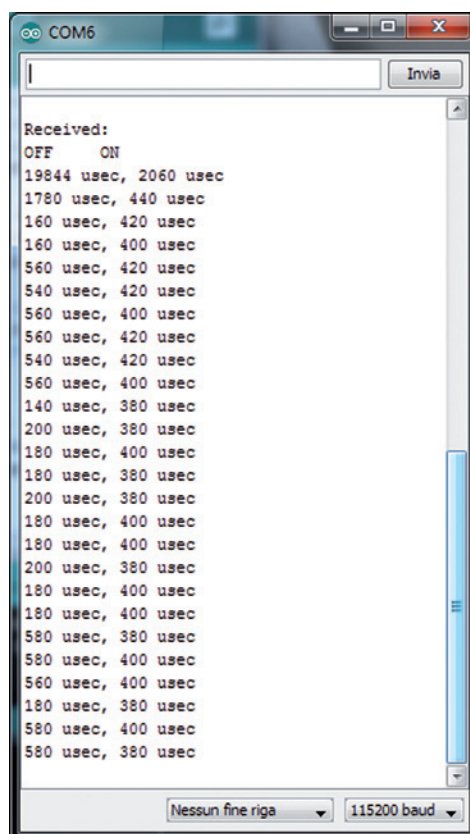
Se associamo uno "0" logico quando gli space valgono circa $170 \mu s$ e un "1" logico quando gli space valgono circa $570 \mu s$, otteniamo una sequenza di uno e zero contenente l'informazione relativa alla posizione degli stick del telecomando. Suddividiamo ora la sequenza in quattro byte, scartiamo il bit più significativo di ciascun byte, perché non utilizzato, otteniamo il valore (con risoluzione di 7 bit) di ciascun canale più il trimmer (Fig.2).

La Tabella 1 ci fornisce i dati dei singoli comandi rilevati. Dopo aver estrapolato correttamente i segnali del telecomando ed elaborato la sequenza dei bit si ottengono i valori di ogni singolo canale.

I nomi delle variabili fanno riferimento alle funzioni svolte dal piccolo elicottero ovvero, il termine YAW indica la rotazione su se stesso, il termine PITCH indica invece l'avanzamento e il THROTTLE la salita e la discesa.

Nella nostra applicazione, invece, utilizziamo le funzioni in modo diverso: lo stick di destra, composto da due canali, fornirà indicazione sulla direzione e la velocità della piattaforma mentre il trimmer sarà usato per impostare la rotazione su se stessa.

Non ci resta che scrivere il firmware per Arduino che aziona la piattaforma, dopo aver capito come si controllano le ruote omnidirezionali. Nelle applicazioni consuete è abbastanza intuitivo azionare i motori,



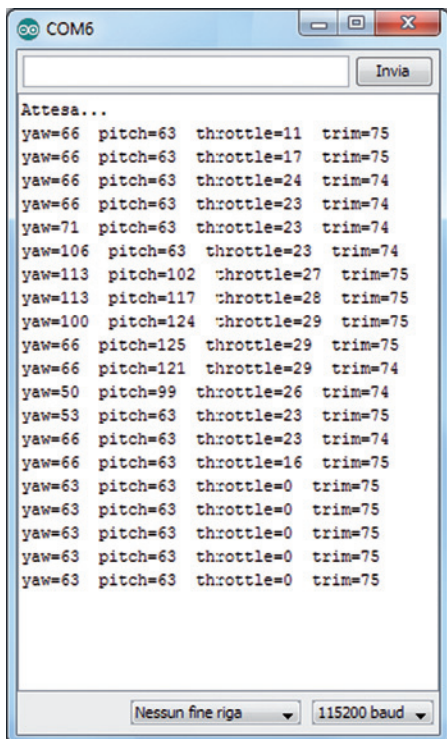
che siano due o quattro ruote, il principio di funzionamento è lo stesso. Basti pensare ad un robot cingolato, se le ruote girano nello stesso verso il robot avanza se sono in rotazione opposta il robot gira su se stesso; utilizzando ruote omnidirezionali tutto questo non è così intuitivo, tantomeno se si utilizza una struttura su tre ruote invece che quattro. Per risolvere l'enigma è necessario rispolverare un po' di matematica, nello specifico i vettori e ragionare su come si comportano le ruote messe in rotazione.

Proveremo ora, nel modo più semplice possibile, a spiegare come controllare questo tipo di piattaforma, ricordando che il tutto può essere adattato all'uso di un maggiore numero di ruote omniwheels, ma non è direttamente applicabile nel caso si utilizzino le ruote *mecanum wheels*.

Dobbiamo analizzare il problema dal punto di vista geometrico sapendo che ciascuna ruota spinge il robot con una velocità V nella sola direzione ortogonale al suo asse di rotazione, mentre è possibile una traslazione in qualsiasi altra direzione. Possiamo quindi rappresentare questo con tre vettori - uno per ciascuna ruota - che rappresentano la velocità applicata dalle ruote



**Il telecomando
usato per
controllare
il nostro robot**



alla struttura; in Fig. 3 la situazione in cui tutti i motori girano alla massima velocità in senso orario. Sappiamo che la struttura è in grado di avanzare in una direzione ad un certa velocità e nel contempo potrebbe ruotare su se stessa. Dobbiamo quindi ricavare delle equazioni nelle quali inserita la velocità, la direzione e la rotazione desiderate venga fornita la velocità di ogni singola ruota, sapendo che essa può ruotare sia in senso orario, sia antiorario. Vediamo graficamente questa situazione riportata in Fig. 4.

Il vettore della velocità di traslazione V_t indica la velocità e la direzione (angolo α) con la quale si muove la struttura, esso è senz'altro la somma dei contributi di ciascuna ruota, pertanto ciascuna ruota dovrà contribuire con un valore proporzionale alla proiezione del vet-

Fig. 2
Decodifica del comando a distanza.

tore totale, rispetto alla direzione di spinta. È possibile calcolare rapidamente la componente di un vettore, rispetto una certa direzione, usando le leggi della trigonometria e conoscendo l'angolo tra il vettore e la direzione sulla quale si vuole la sua proiezione.

Con riferimento alla Fig. 13 possiamo scrivere: $V_x = \cos(\alpha)$ mentre $V_y = \sin(\alpha)$ oppure $V_y = \cos(90 - \alpha)$. Applicando queste relazioni alla nostra struttura otteniamo:

$$\begin{aligned} V_a &= V_t \cdot \cos(\alpha - 30) \\ V_b &= V_t \cdot \cos(\alpha - 150) \\ V_c &= V_t \cdot \cos(\alpha - 270) \end{aligned}$$

Ciascuna ruota induce anche un momento torcente alla struttura rispetto il suo centro, la velocità di rotazione della struttura V_r rispetto il centro è data dal contributo di ciascuna ruota:

$$V_r = V_a + V_b + V_c$$

Le prime tre equazioni riportanti la velocità di ciascuna ruota sono tali che il contributo alla rotazione sia nullo, infatti $V_a + V_b + V_c = 0$, per fare in modo che la struttura possa anche ruotare su se stessa è necessario inserire anche un momento torcente, quindi le equazioni diventano:

$$\begin{aligned} V_a &= V_t \cdot \cos(\alpha - 30) + V_r \\ V_b &= V_t \cdot \cos(\alpha - 150) + V_r \\ V_c &= V_t \cdot \cos(\alpha - 270) + V_r \end{aligned}$$

Data la velocità di traslazione V_t , la direzione α e la velocità di rotazione V_r le tre equazioni forniscono la velocità alla quale dovranno ruo-

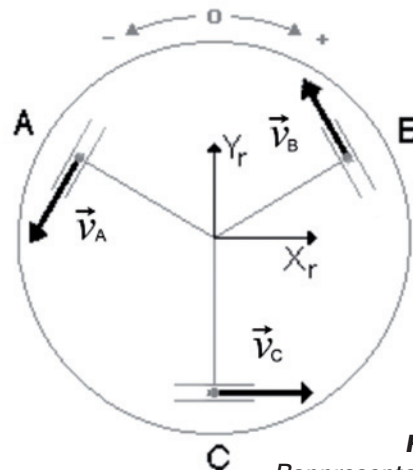


Fig. 3
Rappresentazione grafica dei vettori in gioco.

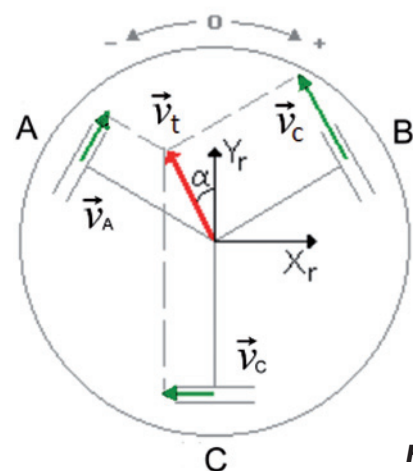


Fig. 4
Rappresentazione dei contributi di ciascuna ruota.

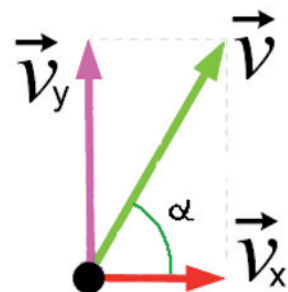


Fig. 5
Legge di trigonometria per un vettore.

Tabella 1

byte	nome	funzione	Valore a riposo	Aumenta se stick spostato a	Diminuisce se stick spostato a
0	yaw	rotazione attorno l'asse	63	sinistra	destra
1	pitch	avanzamento	63	indietro	avanti
2	throttle	potenza	0	avanti	indietro
3	trim	trimmer	-	sinistra	destra

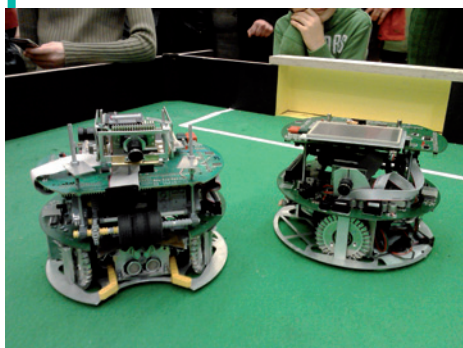


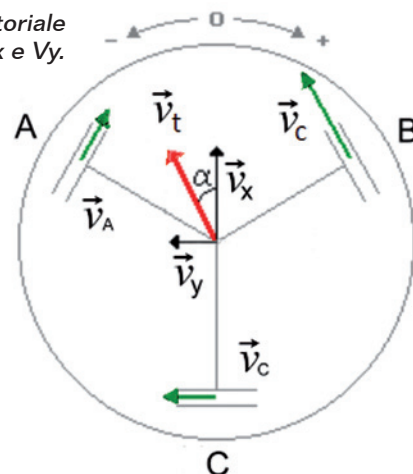
Fig. 6 Robot per gare di soccer.

tare le singole ruote; tralasciamo in questa trattazione la determinazione dell'esatta velocità della struttura in valore assoluto, nota la velocità di rotazione delle ruote. Queste equazioni a carattere generale possono essere adattate anche a strutture con più di tre ruote, per la loro implementazione è però necessario poter eseguire calcoli in virgola mobile oltre al calcolo di cosinusoidi. Una classica applicazione la possiamo incontrare nei robot utilizzati nelle gare di soccer della RoboCup Junior come ad esempio il robot riportato in Fig. 6.

Per la nostra specifica applicazione possiamo introdurre un'ulteriore considerazione che tiene conto del tipo di controllo adottato; lo stick di destra si muove su due assi (YAW e PITCH) fornendo

due valori tra loro ortogonali che tratteremo esattamente come due vettori. Il valore del PITCH fornirà indicazione sull'avanzamento mentre il valore dello YAW indicherà la traslazione laterale e questi due valori potranno combinarsi tra loro. Il vettore per l'asse X sarà ricavato dalla lettura della variabile YAW che, sul telecomando, corrisponde al movimento orizzontale dello stick destro e per tenere conto dell'offset e del segno sarà calcolato come $V_x = 63 - \text{YAW}$. Il vettore per l'asse Y sarà ricavato dalla lettura della variabile PITCH che, sul telecomando, corrisponde al movimento orizzontale, sempre dello stick destro e per tener conto dell'offset e del segno sarà calcolato come $V_y = 63 - \text{PITCH}$. Il valore del THROTTLE, contrariamente a quello che si pensa, non viene utilizzato in quanto il solo stick di destra ed il trimmer sono sufficienti al totale controllo del robot, è necessario però che il Throttle sia azionato, perché solo in questo modo il telecomando invierà la sequenza, altrimenti l'elettronica si posiziona in modalità di standby per risparmiare le batterie. Possiamo nuovamente ricavare le velocità di ogni singolo motore

Fig. 7 Situazione vettoriale partendo dai valori V_x e V_y .



partendo dal valore della velocità e direzione di marcia fornito tramite i due vettori V_x e V_y come in Fig. 7. Si tratta di scomporre questi due vettori rispetto la direzione di trazione delle ruote:

$$\begin{aligned} V_a &= -V_x \cdot \sin(30) - V_y \cdot \sin(60) \\ V_b &= -V_x \cdot \sin(30) + V_y \cdot \sin(60) \\ V_c &= V_x \end{aligned}$$

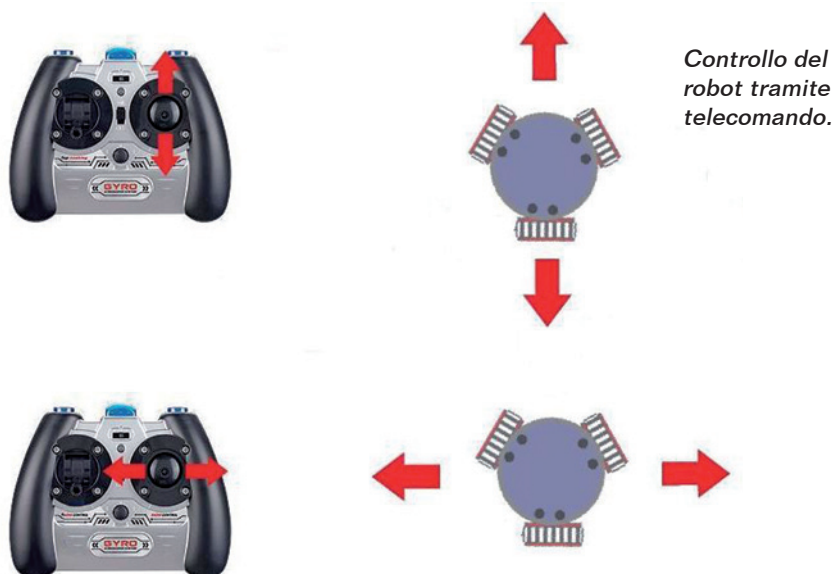
e risolvendo i valori noti:

$$\begin{aligned} V_a &= -(1/2) \cdot V_x - (\sqrt{3}/2) \cdot V_y \\ V_b &= -(1/2) \cdot V_x + (\sqrt{3}/2) \cdot V_y \\ V_c &= V_x \end{aligned}$$

Come prima, a queste equazioni va aggiunto il momento torcente che, nel nostro caso, sarà controllato tramite il trimmer del telecomando con il valore Roll=63-trim. Adesso ci sono tutti gli elementi per poter scrivere lo sketch per Arduino; non riportiamo tutto il listato ma ci limitiamo e descriverne le funzioni più importanti; lo sketch completo può essere scaricato dal sito della rivista.

La prima operazione è attendere una sequenza valida dal trasmettitore ed estrapolarne i valori relativi alla posizione dello stick destro e la posizione del trimmer. Siccome il trimmer determina la modifica del valore del dato relativo allo YAW lo correggiamo.

Per determinare la sequenza in arrivo dal telecomando usiamo la funzione nativa denominata pul-



Listato 1

```
while(pulseIn(IRpin, HIGH) < SIGNAL_START){ }  
for(i=0; i < SIGNAL_SIZE; i++)  
{  
    SignalWidth[i]=pulseIn(IRpin, HIGH);  
}
```

seln che permette di misurare la durata di un impulso giunto sul pin specificato.

Dopo aver letto la sequenza di tutti i trentadue impulsi, convertiamo il valore del tempo in bit logico ottenendo la sequenza numerica, dalla quale vengono estrapolati i valori numerici YAW, PITCH, THROTTLE e TRIM corrispondenti alla posizione degli stick del telecomando (**Listato 1**).

Questo tipo di approccio, pur essendo efficace e di semplice implementazione, non è in generale molto efficiente dato che impone di rimanere bloccati in attesa dell'arrivo di tutta la sequenza senza, nel contempo, poter fare altre operazioni. Per la nostra applicazione questo non è un problema in quanto l'intera sequenza dura meno di 30msec e - in fin dei conti - dobbiamo solo leggere il valore del telecomando ed impostare i motori; abbiamo aggiunto anche un delay per far sì che l'aggiornamento dello stato dei motori avvenga non più di cinque volte al secondo.

La successiva operazione è quella di ricavarne la corretta velocità per i motori: per questo usiamo le righe di programma che trovate nel **Listato 2**.

Il valore moltiplicativo "4.0" ci serve per scalare il valore numerico che, per ciascuna variabile, assume un valore compreso tra -63 e +63; sapendo che Arduino utilizza un PWM con risoluzione a otto bit, il fattore moltiplicativo "4.0" permette di avere dei valori che variano tra -255 e +255. Le variabili di tipo float vengono poi convertite in interi e limitate al massimo valore tramite le istruzioni contenute nel **Listato 3**.

A questo punto è sufficiente impostare il valore del PWM di ciascuna ruota con la funzione analogWrite. Occorre però tenere conto del segno delle variabili

Listato 3

```
PWMA = int(constrain(motA,-255,255));  
PWMB = int(constrain(motB,-255,255));  
PWMC = int(constrain(motC,-255,255));
```

Listato 2

```
motA = 4.0*(-0.5* float(63-yaw)-0.866*float(63-pitch)) + float(63-trim);  
motB = 4.0*(-0.5* float(63-yaw)+0.866*float(63-pitch)) + float(63-trim);  
motC = 4.0*(float(63-yaw)) + float(63-trim);  
}
```

Listato

```
// OmniWheels  
// Piattaforma robotica con 3 ruote omnidirezionali  
// Controllo con IR transmitter SYNAS107G 3CH  
  
#define IRpin 6 // pin for IR sensor  
#define SIGNAL_SIZE 32 // number bit of sequence  
#define SIGNAL_START 1600 // width bit start  
#define SIGNAL_0_min 150 // width bit 0 min  
#define SIGNAL_0_max 250 // width bit 0 max  
#define SIGNAL_1_min 450 // width bit 1 min  
#define SIGNAL_1_max 650 // width bit 1 max  
unsigned int SignalWidth[SIGNAL_SIZE];  
byte yaw, pitch, throttle, trim;  
  
#define DIRA_pin 2  
#define PWMA_pin 3  
#define DIRB_pin 8  
#define PWMB_pin 9  
#define DIRC_pin 12  
#define PWMC_pin 10  
#define DIRD_pin 4  
#define PWMD_pin 5  
  
float motA, motB, motC;  
int PWMA, PWMB, PWMC;  
  
int i;  
  
void setup()  
{  
    Serial.begin(9600);  
    pinMode(IRpin, INPUT);  
    pinMode(DIRA_pin, OUTPUT);  
    pinMode(DIRB_pin, OUTPUT);  
    pinMode(DIRC_pin, OUTPUT);  
    analogWrite(PWMA, 0);  
    analogWrite(PWMB, 0);  
    analogWrite(PWMC, 0);  
    Serial.println("Wait IR Signal...");  
}  
  
void loop()  
{  
    if(IR_Receive()==0)  
    {  
        Serial.print("yaw=");  
        Serial.print(yaw, DEC);  
        Serial.print("\tpitch=");  
        Serial.print(pitch, DEC);  
        Serial.print("\tthrottle=");  
        Serial.print(throttle, DEC);  
        Serial.print("\ttrim=");  
        Serial.print(trim, DEC);  
        Serial.println();  
        SetMotor();  
        delay(200);  
    }  
}  
  
void SetMotor()  
{  
    motA = 4.0 * (-0.5* float(63-yaw)-0.866*float(63-pitch)) + float(63-trim);
```

Listato

```

motB = 4.0 * (-0.5* float(63-yaw)+0.866*float(63-pitch)) + float(63-
trim);
motC = 4.0 * (float(63-yaw)) + float(63-trim);

PWMA = int(constrain(motA,-255,255));
PWMB = int(constrain(motB,-255,255));
PWMC = int(constrain(motC,-255,255));

Serial.print("PWMA=");
Serial.print(PWMA);
Serial.print("\tPWMB=");
Serial.print(PWMB);
Serial.print("\tPWMC=");
Serial.println(PWMC);

Serial.println();

if (PWMA < 0)
{
  analogWrite(PWMA_pin, abs(PWMA));
  digitalWrite(DIRA_pin, HIGH);
}
else
{
  analogWrite(PWMA_pin, PWMA);
  digitalWrite(DIRA_pin, LOW);
}

if (PWMB < 0)
{
  analogWrite(PWMB_pin, abs(PWMB));
  digitalWrite(DIRB_pin, HIGH);
}
else
{
  analogWrite(PWMB_pin, PWMB);
  digitalWrite(DIRB_pin, LOW);
}

if (PWMC < 0)
{
  analogWrite(PWMC_pin, abs(PWMC));
  digitalWrite(DIRC_pin, HIGH);
}
else
{
  analogWrite(PWMC_pin, PWMC);
  digitalWrite(DIRC_pin, LOW);
}
}

int IR_Receive()
{
  int Status=0;
  while(pulseIn(IRpin, HIGH) < SIGNAL_START)
  {
  }

  for(i=0; i < SIGNAL_SIZE; i++)
  {
    SignalWidth[i]=pulseIn(IRpin, HIGH);
  }

  yaw=0;
  pitch=0;
  throttle=0;
  trim=0;

  for( i=1; i < 8; i++)
  {
    if (SignalWidth[i] > SIGNAL_0_min && SignalWidth[i] <
    SIGNAL_0_max)
      BitClear(yaw, 7-i);
  }
}

```

(continua)

Listato 4

```

if (PWMA < 0)
{
  analogWrite(PWMA_pin, abs(PWMA));
  digitalWrite(DIRA_pin, HIGH);
}
else
{
  analogWrite(PWMA_pin, PWMA);
  digitalWrite(DIRA_pin, LOW);
}

```

PWMA, PWMB e PWMC: se queste variabili sono negative il valore del PWM deve risultare comunque positivo, ma il pin DIR associato a ciascun motore, dovrà assumere il livello alto. In questa situazione ciascun motore ruoterà in senso antiorario (**Listato 4**).

Abbiamo previsto che lo sketch invii al Serial Monitor di Arduino le informazioni relative alla lettura del telecomando assieme al valore PWM dei motori; la schermata è mostrata in **Fig. 8**.

Ci preme raccontare anche i limiti che abbiamo riscontrato nel funzionamento di questo tipo di base robotica. Un pri-

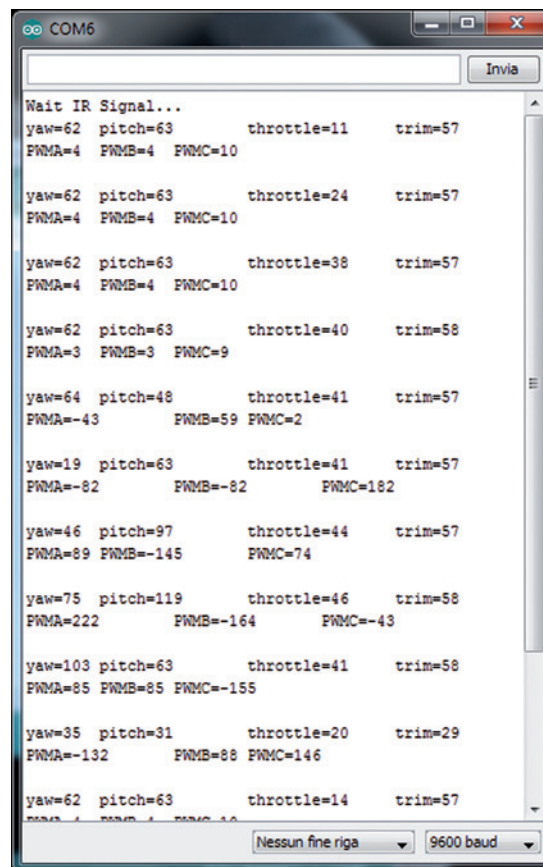


Fig. 8 - Schermata di SerialMonitor con sketch OmniWheels.

Listato

```

else if (SignalWidth[i] > SIGNAL_1_min && SignalWidth[i] < SIGNAL_1_max)
    bitSet(yaw, 7-i);
else
    //Serial.print("Error");
    Status = -1;
}

for(i=1; i < 8; i++)
{
    if (SignalWidth[i+8] > SIGNAL_0_min && SignalWidth[i+8] < SIGNAL_0_max)
        bitClear(pitch, 7-i);
    else if (SignalWidth[i+8] > SIGNAL_1_min && SignalWidth[i+8] < SIGNAL_1_max)
        bitSet(pitch, 7-i);
    else
        Status = -1; //Serial.print("Error");
}

for(int i=1; i < 8; i++)
{
    // il primo bit è sempre zero
    if (SignalWidth[i+16] > SIGNAL_0_min && SignalWidth[i+16] < SIGNAL_0_max)
        bitClear(throttle, 7-i); //viene trasmesso prima il più significativo
    else if (SignalWidth[i+16] > SIGNAL_1_min && SignalWidth[i+16] < SIGNAL_1_max)
        bitSet(throttle, 7-i);
    else
        Status = -1; //Serial.print("Error");
}

for(int i=1; i < 8; i++)
{
    if (SignalWidth[i+24] > SIGNAL_0_min && SignalWidth[i+24] < SIGNAL_0_max)
        bitClear(trim, 7-i);
    else if (SignalWidth[i+24] > SIGNAL_1_min && SignalWidth[i+24] < SIGNAL_1_max)
        bitSet(trim, 7-i);
    else
        Status = -1; //Serial.print("Error");
}

// yaw è modificato da trim
int crz = float(yaw)-0.3*float(trim-63); //correzione
crz = constrain(crz, 12, 115); //limitazione
yaw = map(crz, 12, 115, 0, 127); //adattamento

return Status;
}

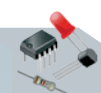
```

mo limite è intrinseco al sistema di funzionamento che conta sul fatto che ciascuna ruota giri ad una precisa velocità e con il dovuto attrito al suolo. Le ruote da noi usate sono in plastica rigida e su superficie lisce non è raro che slittino, anche di poco, modificando l'effetto della rotazione dei motori sul vettore di spostamento. In secondo luogo, matematicamente è facile determinare l'esatto valore del PWM per ciascun motore, non è però detto che a questo corrisponda l'esatta rotazione della ruota; situazione comunque aggirabile utilizzando un controllo ad anello chiuso con degli encoder sulle ruote. Il caso più significativo si ha quando una ruota deve girare

molto lentamente, e i piccoli attriti nel movimento si fanno sentire, infatti un motoriduttore inizia a girare solo quando al motore viene fornita almeno l'energia necessaria per vincere gli attriti interni. Per segnali PWM di valore basso, diciamo inferiori al 10-20%, le ruote non girano per la scarsa potenza trasmessa al motore. Per questi motivi, aggiunti alle varie imprecisioni nella parte meccanica e alle naturali differenze tra i motori, il movimento così come previsto a livello matematico non potrà essere preciso; essenzialmente si assiste ad un lieve momento di rotazione non richiesto, che porta il frontale della piattaforma a non rimanere perfettamente allineato. □

Le ruote utilizzate in questo progetto sono disponibili presso RS Components (cod. 408-542) mentre presso Futura Elettronica è possibile acquistare Arduino Uno (Euro 24,50), i tre motoriduttori (cod. 7300-GM2 Euro 9,00 cad.), le staffe (cod. 7300-GM28SUPP Euro 2,90 cad.) e i due Motor Shield (cod. 7100-MOTORSHIELD_FE Euro 18,00 cad.). I prezzi sono IVA compresa.

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



per il MATERIALE

In questa puntata descriveremo la creazione di un Widget Android attraverso il quale potremo controllare e monitorare dispositivi remoti, collegati allo shield GSM Arduino, direttamente dallo schermo principale del nostro smartphone. Settima e ultima puntata.

di Andrea Chiappori

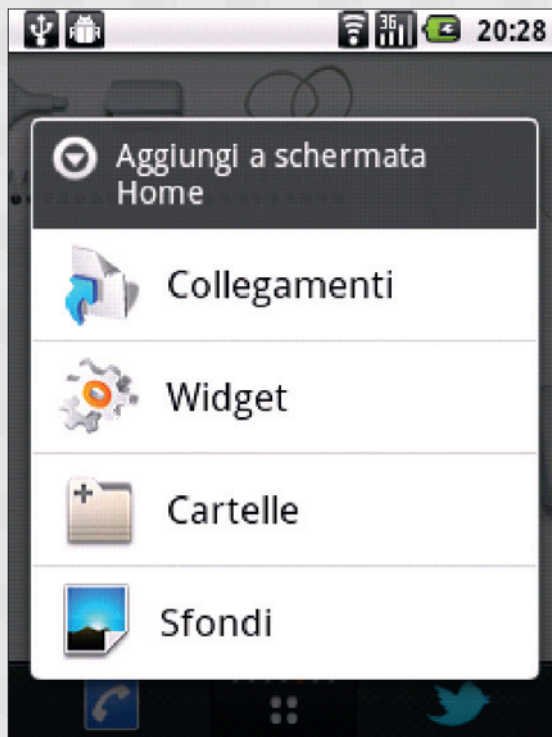
7

PROGRAMMIAMO CON ANDROID

Abbiamo visto nello scorso articolo come implementare un servizio con il quale siamo stati in grado di attivare una chiamata telefonica o un messaggio SMS attraverso lo scuotimento del nostro dispositivo Android, o anche semplicemente attraversando una determinata area geografica. Un servizio ci permette di far funzionare la nostra applicazione in background senza dover tenerla in primo piano e continuando ad usare normalmente il nostro smartphone, ma non ci fornisce la possibilità di interagire con esso attraverso le solite interfacce grafiche come bottoni, liste e caselle di testo già esaminate in questo corso. Può, allora, risultare molto comodo l'utilizzo di un Widget, che altro non è che una vera e propria applicazione Android visualizzata sullo schermo del cellulare sempre in primo piano (nella *homescreen*) e che potremo utilizza-

re per accedere in tempo reale alle informazioni del nostro apparato remoto. Solitamente ogni dispositivo Android, già con il firmware di base, offre una serie di Widget al proprio interno come ad esempio informazioni meteo, aggiornamenti sulle ultime notizie, riproduttori audio e tanti altri che permettono di avere sempre sotto controllo, in forma più semplificata ed immediata, i contenuti o le funzioni dell'applicazione a cui il Widget è legato. Per poter abilitare ed utilizzare questi Widget basta tenere premuto su una qualsiasi zona dello schermo principale fino alla comparsa di una finestra come quella in **Fig.1** in cui andremo a selezionare la voce Widget e successivamente a scegliere quello desiderato. Facilmente, una volta installata qualche applicazione, questa lista potrà aumentare perché spesso i Widget sono inclusi all'interno delle applicazioni e dopo l'installazione vengono resi disponibili al sistema. Tramite il Widget si ha spesso accesso

Fig. 1



alle stesse informazioni e agli stessi comandi dell'applicazione vera e propria, ma con la comodità di poterlo fare direttamente in primo piano e sempre a portata di touch. Ovviamente nulla vieta di creare un Widget con funzionalità proprie, slegate da un'applicazione. In questo articolo vedremo come creare un primo Widget, che avrà una vita propria e che verrà aggiunto a quelli già presenti nel nostro dispositivo. Sarà un progetto dimostrativo senza particolare utilità, ma solo per facilitare la comprensione di

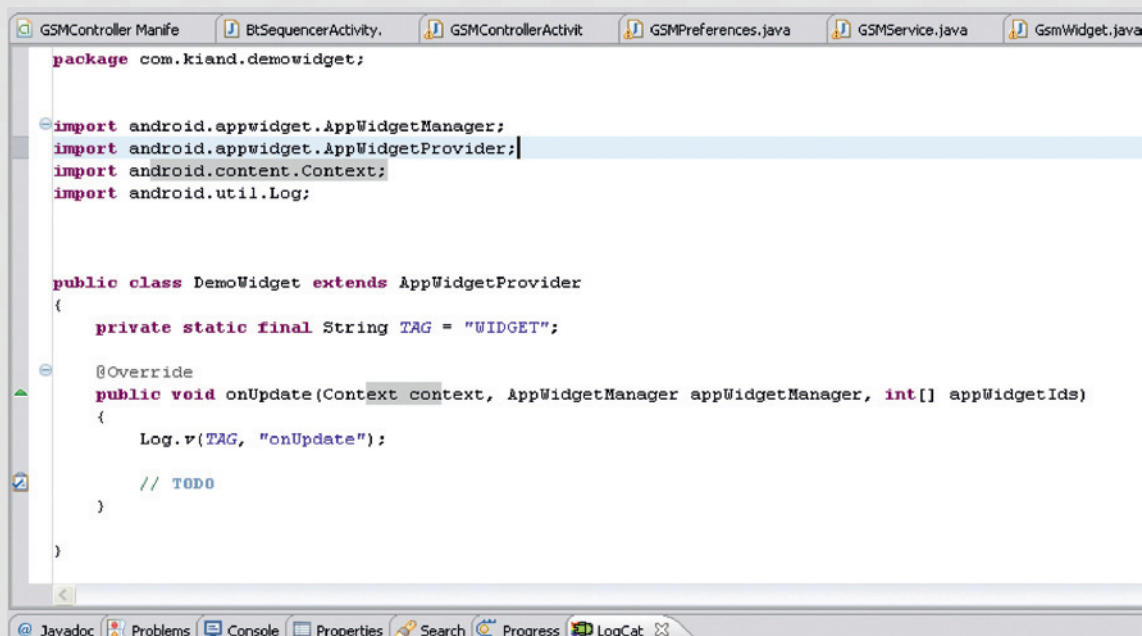
questo elemento, mentre più interessante sarà vedere come dotare l'applicazione GSMController, vista nel numero scorso, di un suo Widget attraverso il quale inviare SMS e ottenere notizie a schermo sullo stato del servizio o su altre informazioni che possono risultare utili e offrire un valore aggiunto all'applicazione principale.

IL NOSTRO PRIMO WIDGET

Creiamo innanzitutto un nuovo progetto al solito modo, questa volta la nostra classe non sarà una classe estesa di un'Activity, ma della classe di sistema *AppWidgetProvider* della quale implementeremo inizialmente il metodo *onUpdate()* che sarà alla base del funzionamento del nostro Widget. Ricordiamo nuovamente che il nome del file .java dovrà essere lo stesso della nostra classe e quindi modificheremo il file *DemoWidget.java* come in Fig. 2, in cui è visibile la struttura base, anche se ancora vuota. Vedremo poi come andrà implementata, ma prima dobbiamo aggiornare il file *AndroidManifest.xml* come in Fig. 3 in modo da indicare al sistema operativo il nome della classe che si occuperà di gestire il Widget e che è, appunto, *DemoWidget*.

Noterete come, non trattandosi di un'Activity, nel file *AndroidManifest.xml*, non compaia il tag *<activity>* ma un tag *<receiver>* nel quale è presente anche un'altra importante informazione sul nome del file xml che rappresenta la risorsa di informazione con la quale andremo

Fig. 2



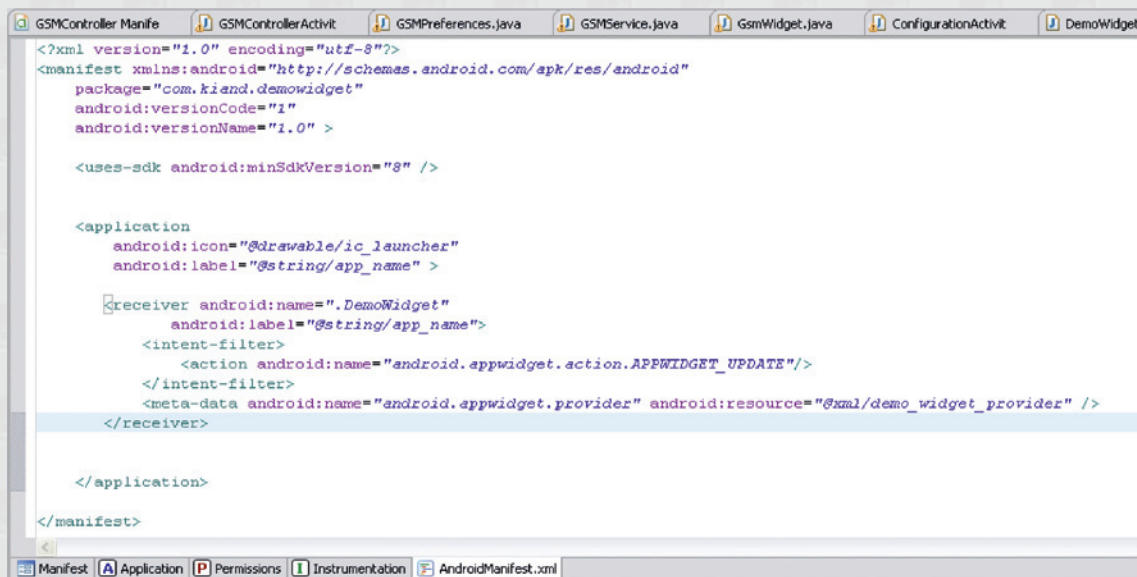


Fig. 3

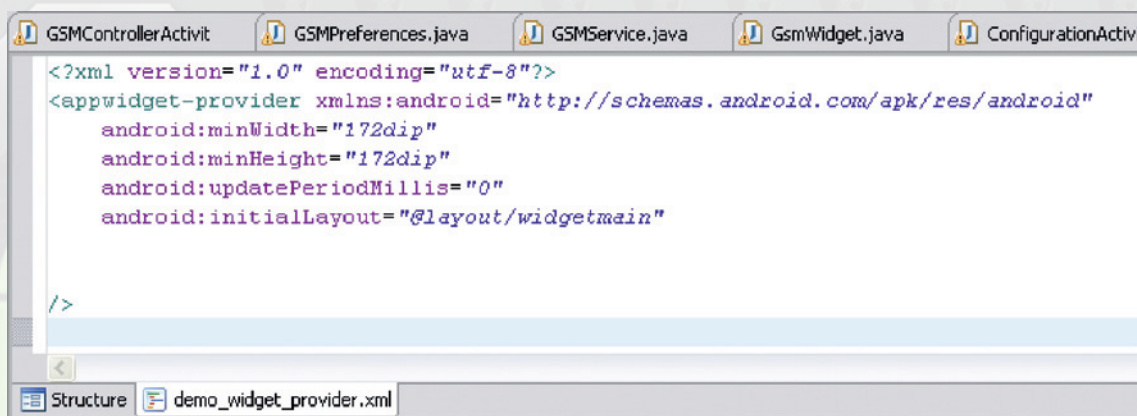
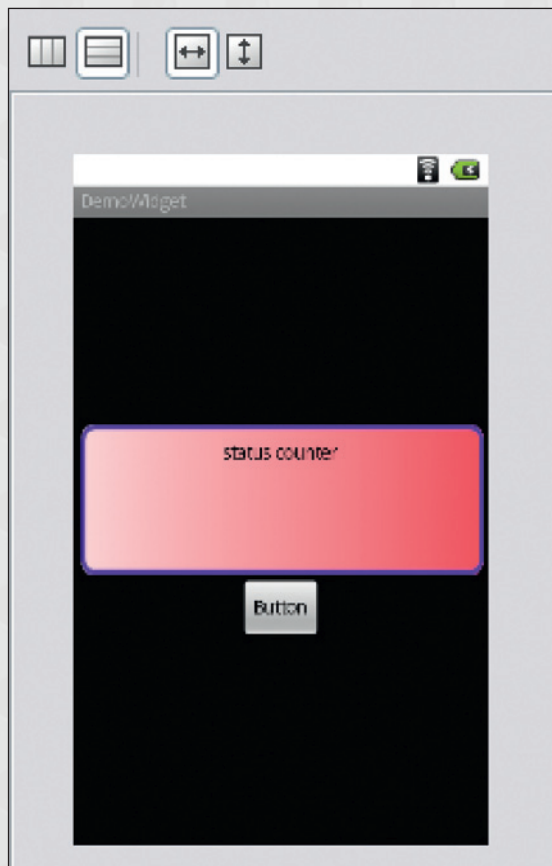


Fig. 4

ad impostare il nostro Widget. In questo caso si tratta del file *demo_widget_provider.xml* che è visibile in Fig. 4 e tramite il quale siamo in grado di impostare alcune proprietà fondamentali come il periodo di refresh (*updatePeriodMillis*) con cui il sistema operativo chiama il metodo *onUpdate()* visto prima, le dimensioni minime con cui comparirà sullo schermo (*minWidth* e *minHeight*) ed il layout grafico (*initialLayout*) in cui andremo a disegnare l'interfaccia grafica personalizzata. In particolare, riguardo alle dimensioni minime, occorre cercare un compromesso evitando di utilizzare valori troppo piccoli che renderebbero il tutto illeggibile, ma nemmeno valori troppo grossi che ne impedirebbero la visualizzazione, specialmente nel caso in cui la *homescreen* fosse già occupata anche da altri Widget o da altre icone. Iniziamo ora a disegnare l'aspetto esteriore del nostro Widget creando il file di layout *widgetmain.xml* (indicato nel campo *initialLayout*) all'interno della cartella *res/xml*. Ricordiamo che nel caso in cui questa cartella non fosse già presente occorrerà crearla manualmente. Questo

è un file di layout come quelli già visti e utilizzati nella creazione di una Activity ed è visibile in Fig. 5. Abbiamo usato un'immagine *png* per creare il riquadro colorato, una *TextView* per visualizzare una stringa (in particolare un semplice contatore) ed un controllo *Button* al quale assoceremo l'invio di una chiamata telefonica verso un numero di telefono preimpostato nel codice. A livello grafico non dobbiamo preoccuparci se l'immagine ed i controlli occupano tutta la parte dello schermo, perché al momento della visualizzazione sul dispositivo reale faranno fede i valori di altezza e larghezza impostati nel file di risorsa visto prima (*demo_widget_provider.xml*). A questo punto il progetto è consistente, possiamo passare a scrivere il cuore del nostro Widget implementando il metodo *onUpdate()* della classe *DemoWidget*, come nel **Listato 1**. Nel caso di un Widget, per accedere agli elementi grafici utilizzati nel layout viene usata la classe *RemoteView*, il cui costruttore riceve il percorso completo del package del progetto (nel nostro caso *com.kiand.demowidget*) ed il file di layout

Fig. 5



(*R.layout.widgetmain*); ora tramite il metodo *setTextViewText()* di questa classe saremo in grado di andare a scrivere una stringa nella *TextView*

identificata da *id.widget_textview*. Per rendere attiva questa modifica occorre ancora effettuare un refresh su tutti i controlli tramite il metodo *updateAppWidget()* della classe *appWidgetmanager* che viene passata a *onUpdate()* e che riceve come parametri la classe del *Widget* e, appunto, le 'viste' (*remoteViews*) intese proprio come controlli. A questo punto, per testare il nostro *Widget* abbiamo creato una variabile interna alla classe, che incrementiamo ad ogni chiamata di *onUpdate()* da parte del sistema operativo e che stamperà la stringa sulla *TextView*. Purtroppo, però, a seconda della versione dell' ADK di Android che stiamo utilizzando potrebbe ancora non funzionare. Questo problema è dovuto al fatto che chiamare molte volte il metodo *onUpdate()* richiederebbe un consumo eccessivo della batteria anche quando il nostro dispositivo fosse in modalità *sleep*; la community Android suggerisce di utilizzare l'update non più di due volte all'ora e pertanto nelle prime versioni del sistema operativo il parametro *updatePeriodMillis* viene preso in considerazione solo per valori maggiori di 1800000 ms (appunto trenta minuti). Noi a scanso di equivoci utilizzeremo un metodo alternativo che prevede l'impiego della classe di sistema *Timer* e del suo metodo *scheduleAtFixedRate()*,

Listato 1

```
package com.kiand.demowidget;

import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.ComponentName;
import android.content.Context;
import android.util.Log;
import android.widget.RemoteViews;

public class DemoWidget extends AppWidgetProvider
{
    private static final String TAG = "WIDGET";

    private static int _counter = 0;

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)
    {
        Log.v(TAG, "onUpdate");

        ComponentName thisWidget = new ComponentName(context, DemoWidget.class);
        RemoteViews remoteViews = new RemoteViews(context.getPackageName(), R.layout.widgetmain);

        remoteViews.setTextViewText(R.id.widget_textview, "TIME = " + _counter++);

        appWidgetManager.updateAppWidget(thisWidget, remoteViews);
    }
}
```


Listato 2

```

package com.kiand.demowidget;

import java.util.Timer;
import java.util.TimerTask;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.util.Log;
import android.widget.RemoteViews;
import android.app.PendingIntent;

public class DemoWidget extends AppWidgetProvider
{
    private static final String TAG = "WIDGET";

    private static Timer _timer = null;

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)
    {
        Log.v(TAG, "onUpdate");

        _timer = new Timer();
        Counter count = new Counter(context, appWidgetManager);
        _timer.scheduleAtFixedRate(count, 1, 1000);

        RemoteViews remoteViews = new RemoteViews(context.getPackageName(), R.layout.widgetmain);

        // create intent for send a call
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:" + "3331234567"));

        // link action to button1
        PendingIntent callPendingIntent = PendingIntent.getActivity(context, 0, callIntent, 0);
        remoteViews.setOnClickPendingIntent(R.id.button1, callPendingIntent);

        // update controls
        appWidgetManager.updateAppWidget(appWidgetIds, remoteViews);
    }

    @Override
    public void onDeleted(Context context, int[] appWidgetIds )
    {
        Log.v(TAG, "Deleted");

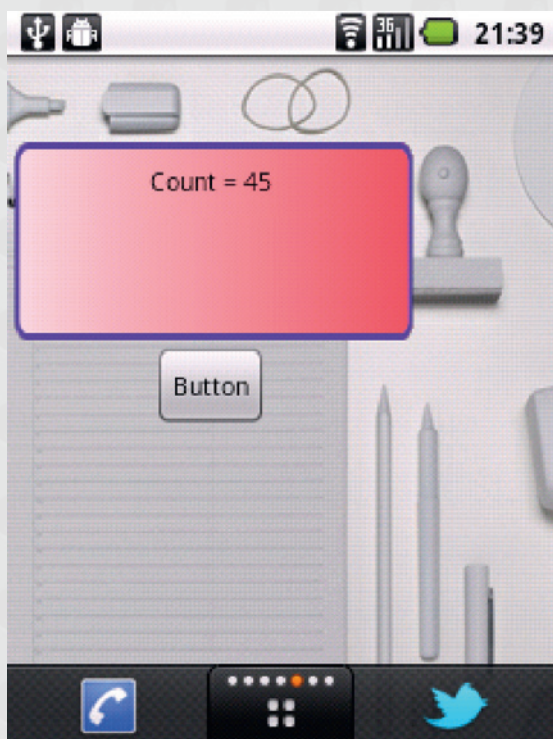
        if(_timer != null) {
            _timer.cancel();
            _timer.purge();
            _timer = null;
        }
    }
}

```

come mostrato nel **Listato 2**, attraverso il quale siamo in grado di eseguire ad ogni intervallo di tempo ciò che implementeremo nel metodo *run()* di una nostra classe estesa di *TimerTask* e che abbiamo chiamato *Counter*. Basterà passare al metodo *scheduleAtFixedRate()* il nostro oggetto della nuova classe *Counter*, insieme al periodo di tempo espresso in millisecondi, ed aggiornare il controllo del Widget all'interno del metodo

run() di questa classe. Ovviamente dovremo implementare il costruttore della classe *Counter* in modo da avere a disposizione tutti i dati necessari per accedere e modificare i controlli grafici del Widget che sono *Context* e *AppWidgetManager*. Sempre nel **Listato 2** noterete che sono stati implementati anche i metodi *onDeleted()* e *onDisabled()* della classe *DemoWidget* e che in essi è stato inserito il codice per la distruzione

Fig. 6



dell'oggetto `_timer`; questo è necessario per evitare che il codice contenuto nel metodo `onUpdate()` continui ad essere eseguito anche dopo aver disabilitato il Widget.

A questo punto possiamo eseguire il nostro primo Widget su emulatore o direttamente su dispositivo; il file eseguibile generato avrà sempre l'estensione `.apk`, ma questa volta, non trattandosi di una applicazione vera e propria, non verrà eseguito immediatamente dal sistema operativo, ma sarà semplicemente aggiunto alla lista dei Widget disponibili e potremo verificarne il funzionamento andando a selezionarlo nel solito modo. Il risultato che ci apparirà sarà come in Fig. 6. Al pari di ogni altro Widget, potrà essere trascinato a piacimento nelle zone dello schermo o anche spostato negli schermi laterali a disposizione. Per personalizzarlo maggiormente potremmo creare un'icona a nostro

Listato 3

```
#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();

      digitalWrite(13, HIGH);

      break;

  }

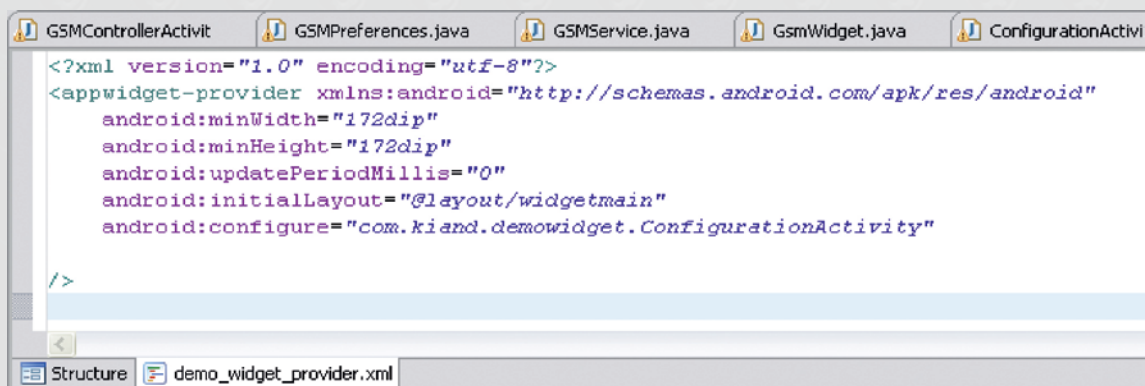
  delay(1000);
}
```

piacimento come già visto nelle puntate precedenti e la potremo vedere comparire accanto al nome del Widget nella lista interna di tutti gli altri Widget.

AGGIUNGIAMO I BOTTONI

Abbiamo, in realtà, già aggiunto un bottone a livello di layout grafico come risulta visibile in Fig. 6, ma non abbiamo ancora associato ad

Fig. 7



Listato 4

```

package com.kiand.demowidget;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.ComponentName;
import android.content.Context;
import android.util.Log;
import android.widget.RemoteViews;
import android.widget.Button;
import android.content.Intent;

public class ConfigurationActivity extends Activity
{
    private int widgetID;

    Button _buttonOk;
    int mAppWidgetId = AppWidgetManager.INVALID_APPWIDGET_ID;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.configuration);

        widgetID = AppWidgetManager.INVALID_APPWIDGET_ID;
        Intent intent = getIntent();
        Bundle extras = intent.getExtras();
        if (extras != null)
        {
            widgetID = extras.getInt(AppWidgetManager.EXTRA_APPWIDGET_ID, AppWidgetManager.INVALID_APPWIDGET_ID);
        }

        _buttonOk = (Button)findViewById(R.id.buttonApply);
        _buttonOk.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent resultValue = new Intent();
                resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, widgetID);
                setResult(RESULT_OK, resultValue);
                finish();
            }
        });
    }
}

```

esso l'implementazione del codice. Immaginiamo di voler eseguire una chiamata telefonica verso lo shield GSM di Arduino al quale sarà associata una determinata azione; per gestire la telefonata useremo lo stesso codice visto nella puntata precedente con cui creiamo l'Intent ACTION_CALL e dal quale creeremo un PendingIntent (pending perché usato in un altro processo) che passeremo al metodo *setOnClickPendingIntent()* della classe *RemoteView* vista prima e che rappresenta tutti i controlli del Widget. Assieme al *PendingIntent* passeremo anche l'ID della risorsa che è appunto il nostro bottone (*R.id.button1*). Tutto questo è visibile sempre nel **Listato 2**. Ora, senza dimenticarci di aggiungere gli import necessari e la riga di permission (*android.permission.CALL_PHONE*) nel solito file *AndroidManifest.xml* possiamo ve-

rificare il funzionamento scaricando nuovamente l'applicazione sul dispositivo. Ricordiamo che per cancellare il nostro Widget ci comporteremo come per qualsiasi altro Widget, tenendo premuto sul controllo per qualche secondo e trascinandolo sull'icona del cestino che comparirà sulla parte bassa (o alta) dello schermo. Anche se molto simile a quello della puntata precedentemente, possiamo dare un'occhiata al **Listato 3** nel quale è presentato il codice Arduino che gestisce la chiamata in arrivo attivando l'uscita del pin 13.

AGGIUNGIAMO LA CONFIGURAZIONE

Avrete notato come nel **Listato 2** il numero di telefono verso cui chiamare sia cablato nel codice; potremmo invece voler cambiare dinamicamente questo numero ogni volta che il Widget

Fig. 8

```

public class GsmWidget extends AppWidgetProvider
{
    private static final String TAG = "WIDGET";

    public static String ACTION_START = "ActionReceiverStart";
    public static String ACTION_STOP = "ActionReceiverStop";
    public static String ACTION_OPTION = "ActionReceiverOption";

    private static Timer _timer = null;

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)
    {
        Log.v(TAG, "onUpdate");

        Intent serviceIntentStart = new Intent(context, GSMService.class);
        serviceIntentStart.setAction(ACTION_START);
        serviceIntentStart.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS, appWidgetIds);

        Intent serviceIntentStop = new Intent(context, GSMService.class);
        serviceIntentStop.setAction(ACTION_STOP);
        serviceIntentStop.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS, appWidgetIds);

        Intent activityOptionIntent = new Intent(context, GSMPreferences.class);
        activityOptionIntent.setAction(ACTION_OPTION);

        RemoteViews remoteViews = new RemoteViews(context.getPackageName(), R.layout.widgetmain);

        PendingIntent callPendingIntentStart = PendingIntent.getService(context, 0, serviceIntentStart, PendingIntent.FLAG_UPDATE_CURRENT);
        remoteViews.setOnClickPendingIntent(R.id.buttonStart, callPendingIntentStart);

        PendingIntent callPendingIntentStop = PendingIntent.getService(context, 1, serviceIntentStop, PendingIntent.FLAG_UPDATE_CURRENT);
        remoteViews.setOnClickPendingIntent(R.id.buttonStop, callPendingIntentStop);

        PendingIntent callPendingIntentOpt = PendingIntent.getActivity(context, 2, activityOptionIntent, PendingIntent.FLAG_UPDATE_CURRENT);
        remoteViews.setOnClickPendingIntent(R.id.buttonOption, callPendingIntentOpt);

        remoteViews.setTextViewText(R.id.widget_textviewStatus, "Status service:");
        remoteViews.setTextViewText(R.id.widget_textviewPosition, "Position:");

        appWidgetManager.updateAppWidget(appWidgetIds, remoteViews);
    }
}

```

viene avviato. Sappiamo ora come creare un altro bottone a cui associare un'Activity di configurazione, ma anche in questo caso Android ci viene incontro offrendoci un metodo più semplice: basterà aggiungere un nuovo campo, *android:configure*, nel file di risorsa *demo_widget_provider.xml* visto in precedenza e passargli il percorso del file java in cui scriveremo la nostra Activity di configurazione, come in Fig. 7. Nel

nostro caso il file sarà *ConfigurationActivity.java* e conterrà una *TextView* in cui inserire il numero di telefono di destinazione ed un bottone con cui validarlo ed avviare il Widget.

Non sarà, in questo modo, necessario aggiungere un ulteriore pulsante al Widget per la sua configurazione perché la nostra *ConfigurationActivity* verrà automaticamente creata al momento del lancio del Widget e si occuperà lei stessa di

Fig. 9

```

@Override
public void onStart(Intent intent, int startid)
{
    Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
    Log.d(TAG, "onStart Service");

    _statusService = true;

    // update widget
    _appWidgetManager = AppWidgetManager.getInstance(this.getApplicationContext());
    _thisWidget = new ComponentName(getApplicationContext(), GsmWidget.class);
    _remoteViews = new RemoteViews(this.getApplicationContext().getPackageName(), R.layout.widgetmain);

    String strAction = intent.getAction();
    if (strAction != null)
    {
        if (strAction.equals(GsmWidget.ACTION_STOP))
        {
            _remoteViews.setTextViewText(R.id.widget_textviewStatus, "Status service: Stopped");
            _appWidgetManager.updateAppWidget(_thisWidget, _remoteViews);

            Log.d(TAG, "auto stop service");
            stopSelf();
            return;
        }
    }

    _remoteViews.setTextViewText(R.id.widget_textviewStatus, "Status service: Started");
    _appWidgetManager.updateAppWidget(_thisWidget, _remoteViews);

    ///////////////
}

```


farlo partire, sulla pressione del bottone *buttonApply* come mostrato nel **Listato 4**.

Anche in questo caso dovremo aggiornare il file *AndroidManifest.xml* informandolo sull'utilizzo di questa nuova Activity e per passare l'informazione acquisita nella TextBox possiamo ricorrere alle *SharedPreferences* viste nella puntata precedente; il numero di telefono verrà memorizzato nell'Activity per poi essere recuperato ed utilizzato nel metodo *onUpdate()* del Widget stesso.

INTEGRAZIONE CON GSMCONTROLLER

Vediamo ora come aggiungere un Widget personalizzato alla nostra applicazione *GSM-Controller*.

L'obiettivo è quello di aumentare visibilità al nostro servizio dotandolo di un controllo maggiore senza dover necessariamente riaprire ogni volta l'applicazione in questione. Possiamo ad esempio fare in modo che nel Widget vi sia un bottone per attivare direttamente il servizio, un altro per stopparlo e un terzo per accedere alle opzioni. Allo stesso modo possiamo fare in modo che il servizio dell'applicazione *GSMController* invii, esso stesso, determinate informazioni al Widget, come la distanza dal punto geografico prestabilito - aggiornata di volta in volta in tempo reale - o il numero di telefono corrente al quale verrà diretta la chiamata in caso di scuotimento ripetuto del dispositivo oltre allo stato del servizio ed altri dati utili.

Le modifiche da apportare sono relativamente poche, potremmo anche utilizzare parte del codice già scritto per il *DemoWidget* visto sopra, semplicemente copiando il file *DemoWidget.java* all'interno della cartella *src* del progetto, rinominandolo in *GsmWidget.java* e aggiungendo due bottoni al layout grafico (*widgetmain.xml*). Per la visualizzazione delle informazioni potremo usare lo stesso controllo *TextView*

Listato 5

```
@Override
public void onReceive(Context context, Intent intent)
{
    String strAction = intent.getAction();
    Log.v(TAG, "onReceive:" + strAction);

    if (intent.getAction().equals(ACTION_START))
    {
        Log.v(TAG, "ACTION_START");
    }
    else if (intent.getAction().equals(ACTION_STOP))
    {
        Log.v(TAG, "ACTION_STOP");
    }
    else if (intent.getAction().equals(ACTION_OPTION))
    {
        Log.v(TAG, "ACTION_OPTION");
    }
    else {
        super.onReceive(context, intent);
    }
}
```

utilizzato in precedenza, aggiungendone altri per visualizzare eventualmente altri dati con maggior dettaglio. Questa volta rinomineremo il file con le informazioni del Widget in *gsm_widget_provider.xml*, aggiorneremo opportunamente il file *AndroidManifest.xml* sulla base delle modifiche viste prima, ma senza utilizzare l'Activity di configurazione. Possiamo adesso compilare tutto e scaricare sul dispositivo l'applicazione che partirà normalmente, ma rispetto a prima, adesso abbiamo a disposizione un nuovo Widget che verrà identificato con la stessa icona dell'applicazione.

La parte più interessante da analizzare è invece l'interazione tra il nostro Widget e l'applicazione, perché vogliamo essere in grado di poterla controllare tramite il Widget in primo piano, ma anche di permettere all'applicazione stessa di aggiornare il Widget sul suo stato attuale. Partiamo inizialmente a modificare il file *GsmWidget.java* ed in particolare il metodo *onUpdate()* che modificheremo come in **Fig. 8**. Non utilizzeremo più la classe *Timer* per aggiornare periodicamente il Widget perché in questo caso non sarà necessario, anzi, basterà che il metodo *onUpdate()* sia chiamato una volta sola, al momento del lancio del Widget e per esserne

```
</activity>

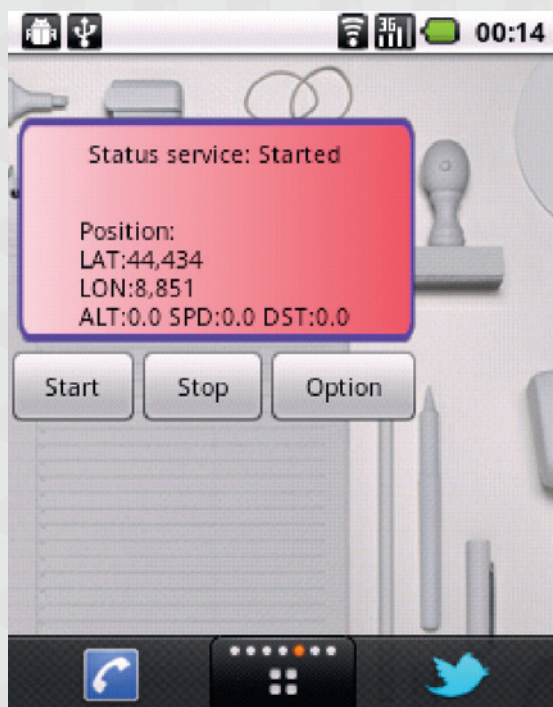
<receiver android:name="com.kiand.gsm.GsmWidget"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE"/>

        <action android:name="com.kiand.gsm.GsmWidget.ACTION_START"/>
        <action android:name="com.kiand.gsm.GsmWidget.ACTION_STOP"/>
        <action android:name="com.kiand.gsm.GsmWidget.ACTION_OPTION"/>

    </intent-filter>
    <meta-data android:name="android.appwidget.provider" android:resource="@xml/gsm_widget_provider" />
</receiver>
```

Fig. 10

Fig. 11



sicuri imposteremo il valore *updatePeriodMillis* a zero. Dovremo però associare ai tre bottoni le azioni desiderate e quindi creeremo due Intent per attivare e stoppare il servizio GSMController ed un Intent per chiamare l'Activity che gestirà le opzioni dell'applicazione principale (la PreferenceActivity *GsmPreferences*) già scritta ed analizzata nella puntata precedente.

Come già fatto per il Widget di esempio, ricaviamo i PendingIntent dai corrispondenti Intent appena creati e li associamo ai bottoni tramite il metodo *setOnClickPendingIntent()*, infine aggiorniamo le stringhe di testo nelle due TextView tramite *setTextViewText()* e i controlli per rendere valide le modifiche con *updateAppWidget()*. Da notare come tutti gli Intent vengano creati passando il nome della classe corrispondente (*GSMService* per il servizio e *GSMPreferences* per l'Activity) mentre per gli Intent che gestiscono lo start e lo stop del servizio si debba usare *PendingIntent.getService()* e per l'Activity occorra usare *PendingIntent.getActivity()*. Infine, non dimentichiamo di usare un requestcode (il secondo parametro di *getService* e *getActivity*) diverso per la creazione dei vari PendingIntent, altrimenti tutti i bottoni avranno una stessa azione associata. Noterete anche come ogni Intent abbia associata una stringa diversa e personalizzabile, tramite il metodo *setAction()*; nel nostro caso sono ACTION_START, ACTION_STOP e ACTION_OPTION e servono per identificare l'Intent corrispondente. Ogni volta che viene premuto uno dei tre pulsanti, viene chiamato

il metodo *onStart()* del servizio *GSMService* collegato all'Intent dal quale possiamo ricavare l'azione associata tramite il metodo *getAction()* ed essere in grado di capire quale pulsante sia stato premuto. Questo è quanto succede nella Fig. 9 in cui ci troviamo già nel codice del servizio; in particolare se l'azione ricevuta è la ACTION_STOP interrompiamo il servizio nel quale già ci troviamo con la funzione di sistema *stopSelf()*. Nulla ci avrebbe vietato di usare il meccanismo di passaggio parametri dell'Intent tramite i metodi *putExtra()* e *getExtra()* visti nella scorsa puntata ed anzi, a seconda dei casi, si possono utilizzare entrambe le strade, ma l'impiego di queste Action ci offre anche il vantaggio di gestire la scelta dei pulsanti anche all'interno della classe Widget grazie al metodo *onReceive()* (nel file *GsmWidget.java*) che è visibile nel Listato 5 in cui possiamo implementare ulteriore codice di controllo. Per far sì che la *onReceive()* sia in grado di 'trappare' queste Action occorre registrarle nel file *AndroidManifest.xml* come mostrato in Fig. 10, all'interno del tag *<receiver>* e pertanto occorre che le Action siano dichiarate pubbliche nella classe del Widget. Infine, una volta che nel servizio della nostra applicazione abbiamo ricavato l'oggetto *_remoteView* riferito ai controlli presenti nel Widget, come in Fig. 9, possiamo utilizzarlo per aggiornare il Widget con i dati che ci vengono forniti nel tempo dal servizio; ad esempio se volessimo visualizzare sulla TextView del Widget la locazione corrente o la distanza dal punto prefissato in cui partirà la chiamata verso lo Shield GSM di Arduino, potremmo farlo tranquillamente al solito modo, usando il metodo *setTextViewText()* seguito da *updateAppWidget()* ogni qual volta venga chiamata la *_locationListener* che è in attesa di una notifica dal sistema operativo di cambiamento di locazione. Il risultato finale sarà come in Fig. 11.

CONCLUSIONI

Con questa puntata concludiamo il corso di programmazione di Android, consapevoli di non aver approfondito alcuni argomenti di carattere prettamente informatico, talvolta senza scendere troppo nei dettagli, per prediligere il lato "hardware" dell'opera; speriamo vivamente di aver fornito comunque le principali basi e qualche spunto per i vostri futuri progetti elettronici nel mondo Android. ■