

Mensile di progettazione elettronica, attualità scientifica, novità tecnologiche.

€ 6,00

Elettronica In

www.elettronica.in.it

oltre l'elettronica

Anno XXI - n. 197
Luglio / Agosto 2015

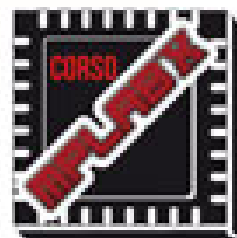
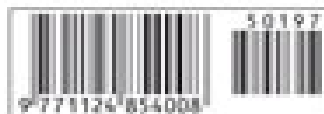
- Microsoft & Open Source: la strana coppia
- Radiocomando 8 canali 433,92 MHz
- GestIC, proviamola con Raspberry Pi

Electronic DRINK MAKER

Arduino + Infrarossi ArdIR



- Domotica OpenSource con i moduli UDOIT
- TDGRemote per TDG133
- Generatore di segnali campione
- Superare i limiti di carica delle porte USB
- Progettare un circuito d'ingresso RF





DRINK MAKER

di BORIS LANDONI
e GABRIELE DAGHETTA

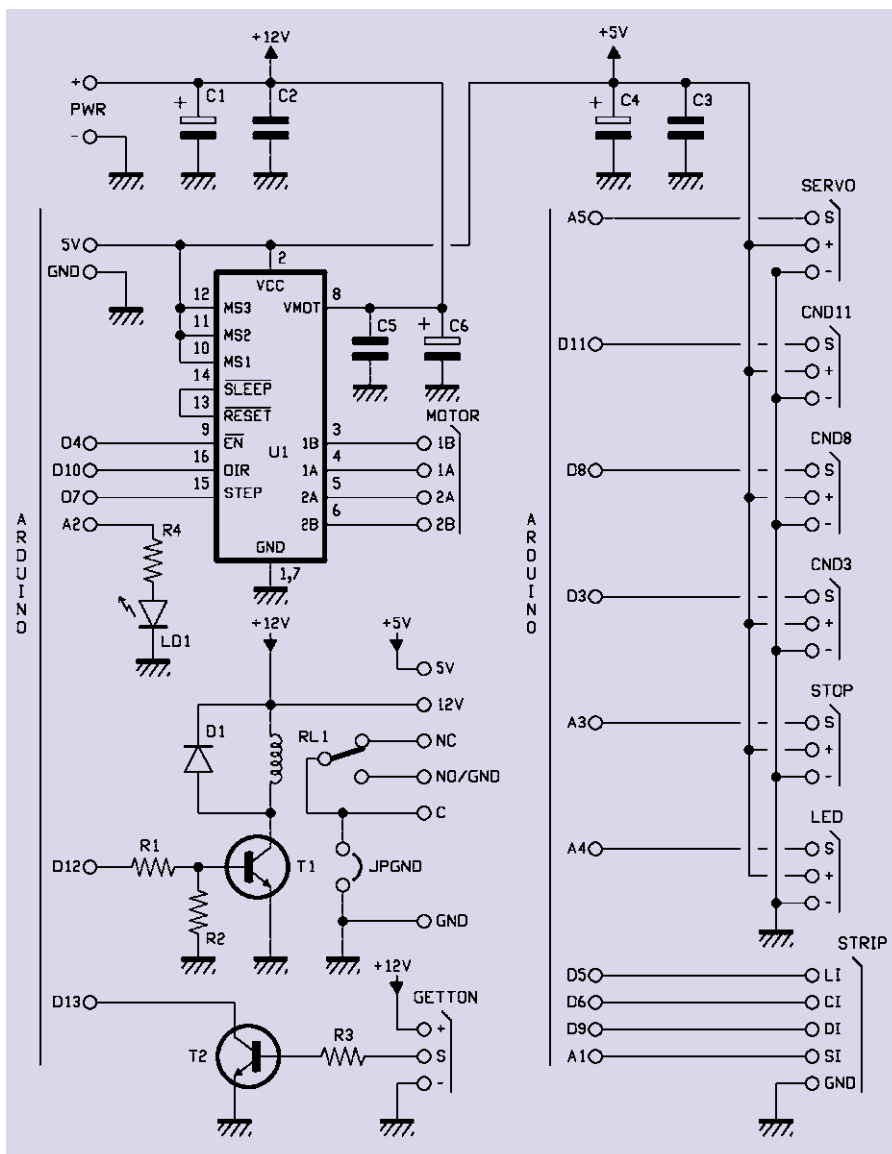


Basata su RandA, prepara cocktail prelevando da appositi dispenser le quantità di bevande previste dalle ricette memorizzate nella pagina web dalla quale si richiede la preparazione. Prima puntata.



Il caldo è arrivato e oltre a un po' d'ombra, nei pomeriggi torridi cerchiamo un po' di rinfresco in una bevanda fredda e rigorosamente analcolica, che ci disseti per un po'. Le sere all'aperto chiamano cocktail che, potendosi rilassare e magari non dovendo mettersi al volante per un po', coniughino il piacere dell'alcol con refrigerio di qualche cubetto di ghiaccio. Se andiamo al bar troviamo il barman che fa il giocoliere con il tumbler per servirci la nostra bevanda preferita mentre se siamo a casa con gli amici, dobbiamo rimboccarci le maniche o delegare a un amico esperto la preparazione delle bevande. Esiste anche una terza possibilità, invero futuristica ma già sperimentata in alcuni locali "invasi" paci-

ficamente dalla robotica: affidarsi a una macchina prepara-cocktail. Cose del genere esistono nel mondo professionale, ma possono essere auto-costruite sfoderando la propria grinta da maker e sfogliando le pagine di Elettronica In, perché infatti, vista all'opera una macchina del genere e conquistati dalla voglia di farci servire un cocktail perfettamente miscelato e con i giusti ingredienti, ci siamo messi all'opera e abbiamo costruito un robot capace di farlo. Ne è nato Drink Maker, un progetto realizzabile con un po' d'iniziativa da chi sa praticare con parti meccaniche e naturalmente con l'elettronica, ma anche disponibile in versione da montare per chi vuole realizzare in casa un proprio angolo bar



tecnologico e stupire gli amici facendo loro "ordinare" sullo smartphone o sul PC il cocktail preferito e vedendolo preparare in diretta dal robot.

IL PROGETTO

La nostra macchina robotizzata prepara-bevande è un telaio composto da profilati di alluminio, sul quale sono applicati in fila dei dispenser con azionamento a pulsante posteriore, premuto da una leva comandata da un servomotore; ciascun dispenser (o erogatore, o spillatore, se preferite) ogni volta che se ne preme

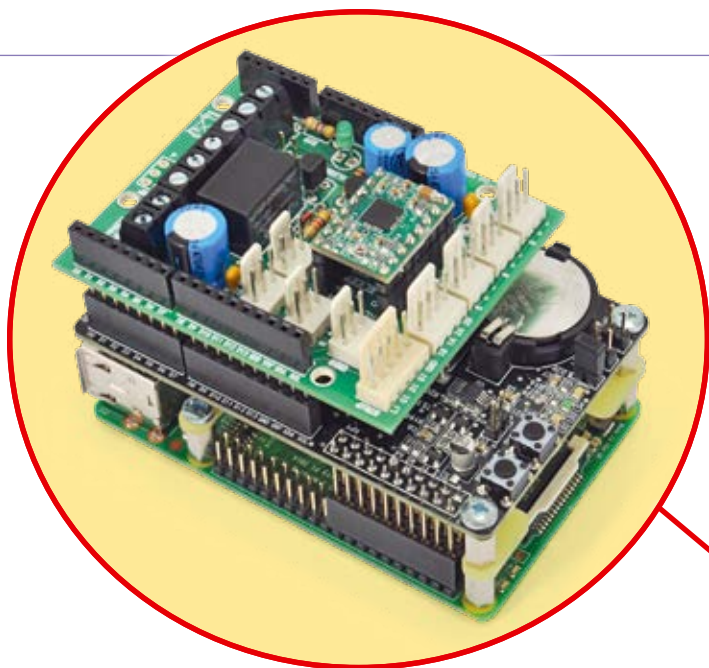
l'azionamento eroga una quantità prefissata di bevanda, che nel nostro caso corrisponde a 2 cc (20 ml). I dispenser sono montati ognuno su un portabottiglia che permette di infilarvi l'ugello a guarnizione nel collo della bottiglia e rendere l'insieme stabile. Il tutto sta sul sopralzo della macchina, fissato alla base sulla quale scorre linearmente sull'asse il carrello portabicchiere, che scorre tramite dei ball caster sulle cave dei profilati d'alluminio più lunghi del telaio; dietro al carrello è fissato anche il servocomando che aziona il dispenser, così ogni

volta che il bicchiere viene portato dal carrello sotto un erogatore, questo viene azionato e versa la propria dose per il numero di volte impostato dal sistema. Per evitare che nel movimento da un dispenser all'altro il cocktail venga agitato tanto da uscire dal bicchiere, abbiamo fatto in modo che il carrello vari progressivamente la propria velocità partendo lento, raggiungendo dopo circa un secondo la velocità massima, quindi rallentando fino a fermarsi, in un altro secondo. Il movimento del carrello è ottenuto mediante una cinghia dentata che ingrana in una puleggia, dentata anch'essa, azionata da uno stepper motor posto nella parte inferiore del telaio; il motore viene gestito da un apposito controller e allineato alla partenza sulla base di un finecorsa posto tutto a sinistra, che viene eccitato da un'apposita sporgenza del carrello. L'inizio della corsa del carrello combacia con il primo dispenser, quindi in fase di costruzione della macchina dovete assicurarvi di posizionare lo switch finecorsa in modo che la sua levetta venga premuta quando il carrello ha il bicchiere centrato rispetto all'ugello del primo dispenser.

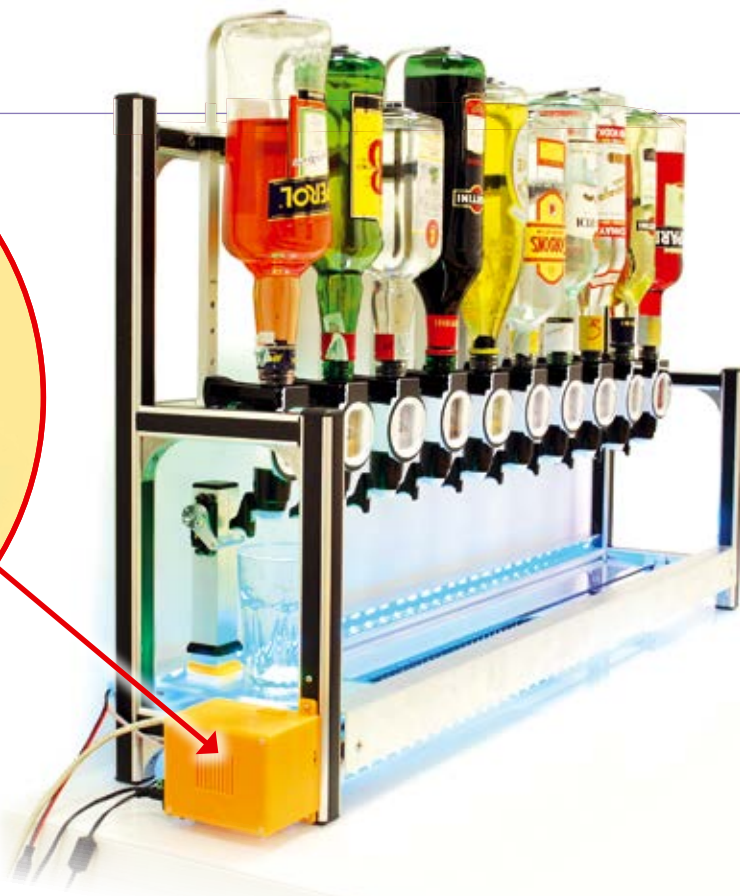
Affinché la meccanica possa compiere quanto appena descrit-



Lo shield realizza l'interfacciamento tra RandA e la parte elettromeccanica della macchina.



L'intera elettronica prende posto nel contenitore stampabile in 3D, da fissare al lato del telaio e da cui escono i cavi, nonché le connessioni di Raspberry Pi.



to, vale a dire portare il carrello col bicchiere sotto le bottiglie necessarie a preparare il cocktail richiesto e a ricevere dai dispenser le quantità di bevanda previste, ci vuole un controllo elettronico abbastanza raffinato, che nel nostro caso è ottenuto combinando Raspberry Pi, la nostra scheda RandA (bridge tra Raspberry Pi e Arduino), uno specifico shield e un driver motori, che poi è lo stesso impiegato nel progetto della stampante 3D 3Drag. Tutta la sequenza di movimenti necessaria a preparare un cocktail e riportare il bicchiere che lo contiene a inizio macchina per essere prelevato, viene decisa sulla base della composizione del cocktail stesso, scelta dall'utente tramite un'interfaccia web personalizzabile. Abbiamo quindi una parte dell'elettronica che permette di scegliere il cocktail in un database (preventivamente preparato dal gestore del sistema e componibile a piacimento), un'altra che traduce gli ingredienti del cocktail stesso in movimenti della meccanica (spostamento carrello, pressione

del pulsante del dispenser...) e un'altra che fisicamente controlla -seguendo le istruzioni corrispondenti- il motore passo-passo e il servocomando per operare le azioni del caso.

In particolare, nella sezione elettronica della macchina:

- Raspberry Pi genera e governa l'interfaccia web d'utente, fa da host per le impostazioni dei cocktail, gestisce funzioni ausiliarie come la verifica della soddisfabilità dell'ordine sulla base della disponibilità degli ingredienti componenti il cocktail (tiene memoria delle quantità erogate, nota la capienza della bottiglia di ogni dispenser), quindi per ogni prodotto prepara la serie di comandi di posizionamento e attivazione del pulsante del dispenser che poi passa alla sezione Arduino;
- RandA, che contiene l'hardware di Arduino Uno e le interconnessioni con Raspberry Pi, riceve da quest'ultima le istruzioni di preparazione del cocktail in corso, vale a dire la posizione richiesta e il coman-

do del pulsante di apertura del dispenser, le converte in comandi diretti al driver motore (impulsi di comando) e del servocomando, quindi impartisce i comandi del caso nella sequenza opportuna;

- lo shield interfaccia RandA con le connessioni dello stepper-motor e del servocomando, ma anche quelle con i dispositivi di segnalazione luminosa (se ne parlerà tra breve), il fincorsa ed altro, oltre ad ospitare il driver motore;
- il driver motore, sulla base degli impulsi ricevuti da RandA, pilota lo stepper motor per farlo avanzare del numero dei passi richiesto.

Più esattamente, RandA ha scritto nel proprio firmware il rapporto tra l'avanzamento necessario per raggiungere ogni dispenser e il corrispondente numero di passi che lo stepper-motor deve compiere; per esempio, nella macchina da noi realizzata per le prove la distanza tra un dispenser e il successivo è 97,7 mm e quella da inizio corsa al primo dispenser è



La strip di LED è dotata di un connettore a passo 2,54 mm e prende l'alimentazione da un jack femmina volante.

zero. Tradotto in step, significa che il primo dispenser corrisponde alla posizione di partenza (quindi per prelevare il contenuto della bottiglia ivi applicata basta solo comandare il servo che aziona il pulsante, senza ordinare avanzamenti del carrello) mentre per raggiungere il secondo dispenser dal primo RandA deve inviare al controller 6.300 impulsi, corrispondenti a 6.300/16 step (perché il driver è impostato per eseguire uno step completo ogni 16 impulsi) che il motore passo-passo dovrà compiere. La distanza tra i dispenser, ossia 97,7 mm, corrisponderà quindi a 6.300/16 passi dello stepper-motor (1 step fa percorrere 0,248 mm). Quindi quando riceve l'ordine di raggiungere il primo dispenser, lo traduce negli step del motore e invia il numero di impulsi corrispondente alle linee di controllo del driver motore affinché questo azioni lo stepper-motor. Completato il numero di passi, RandA impartisce il comando per l'attivazione del servomotore che preme il pulsante del dispenser sotto il quale si è portato il carrello. Terminata la sequenza di posizionamento ed apertura degli spillatori, il carrello viene riportato all'inizio della corsa per servire il cocktail; l'arresto avviene quando si aziona lo switch finecorsa, allorché il sistema conclude la sequenza e si predispone per l'esecuzione di un nuovo ordine. Questo è quanto riguarda l'azionamento; è stata poi prevista una parte -diciamo- coreografica, ed

una di segnalazione ottica in assistenza al "barman" di turno: una strip di LED basata sul controller Philips HL1606 gestibile da bus seriale SPI viene comandata tramite lo shield in modo da fare luci pseudo-casuali per abbellire la macchina, ma all'inizio della sequenza di preparazione del cocktail lampeggia e mostra una barra luminosa che si accorcia in 10 secondi, indicando il tempo rimasto per appoggiare il bicchiere vuoto sul carrello prima che inizi l'operazione di preparazione. Se non si mette il bicchiere in tempo, a meno di non fermare la procedura con il pulsante di reset su RandA, la macchina procede e versa sul fondo della struttura il contenuto prelevato dai dispenser...e non è il massimo perché potrebbe bagnarsi il cablaggio, oppure lo stepper motor o lo switch finecorsa.

SCHEMA ELETTRICO SHIELD

Il circuito che interfaccia RandA alla meccanica della macchina è contenuto in uno shield sul quale si monta il driver motori (per la sua descrizione rimandiamo all'apposito riquadro in queste pagine); lo shield ospita i connettori base per l'applicazione

come quello per connettere lo switch finecorsa (contatti STOP), la strip di LED (contatti STRIP), il servocomando (SERVO) e un eventuale contatto cui subordinare l'ordinazione (si applica ad esempio all'uscita di una gettoniera elettromeccanica con contatto pulito) il cui connettore è siglato GETTON. Sono stati poi allestiti i connettori ausiliari CND3, CND8 e CND11, riservati a eventuali sviluppi che se ve la sentirete potrete fare da voi. C'è anche il connettore LED, per accendere dei LED decorativi da disporre sotto l'alloggiamento del bicchiere per comporre giochi di luce. Lo switch finecorsa può essere NC o NA, collegato tra S e il positivo o tra S e massa, ma nella nostra applicazione è NC, collegato tra S e massa; allo scopo, il firmware di RandA imposta per la linea corrispondente (A3 di Arduino) il resistore interno di pull-up, in modo che quando lo switch è a riposo e risulta chiuso, l'ATmega di RandA riceva lo zero logico, mentre a posizione di riposo raggiunta, A3 sia ad 1 logico. La scelta di usare uno switch NC è dettata dal fatto che la macchina ferma il carrello e la sequenza aprendolo, quindi se si taglia o stacca un filo, siamo certi che la macchina si arresta. L'ingresso della gettoniera è



Per ospitare l'elettronica (RandA, Raspberry Pi e lo shield) abbiamo previsto un contenitore stampabile in 3D, dotato di ventola.

Il modulo driver per stepper motor

Il controllo del motore passo-passo è affidato a un modulo della Futura Elettronica basato sull'integrato A4988 della Allegro, che integra un doppio driver a ponte ad 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 1** 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.

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; 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 sem-

pre 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, 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 µs 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 µs, mentre il modo di decay è Mixed decay sia per correnti in cre-

scita che in calo, tranne che nel modo 1/1 passo;

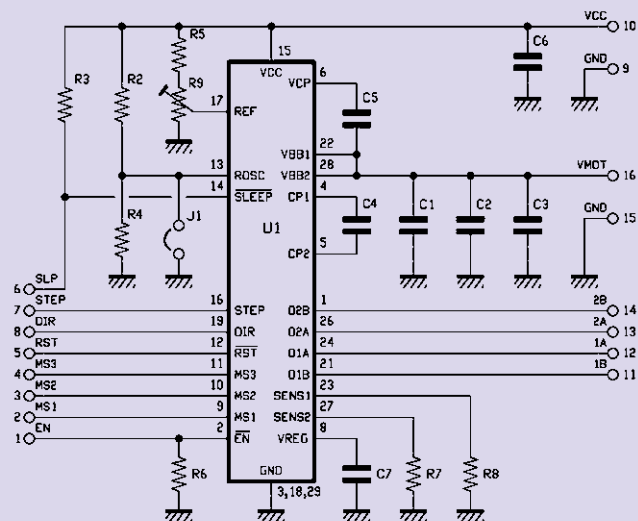
- tensione intermedia= il tempo di off viene definito (in microsecondi) dalla formula $t_{off} \approx R_{sc}/825$.

Queste modalità riguardano l'andamento degli impulsi di comando dei motori, il cui inviluppo 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 regolata dal potenziale di controllo applicato 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).



MS1	MS2	MS3	Risoluzione	Excitation Mode
L	L	L	un passo per impulso	2 Phase
H	L	L	1/2 passo per impulso	1-2 Phase
L	H	L	1/4 di passo per impulso	W1-2 Phase
H	H	L	1/8 di passo per impulso	2W1-2 Phase
H	H	H	1/16 di passo per impulso	4W1-2 Phase

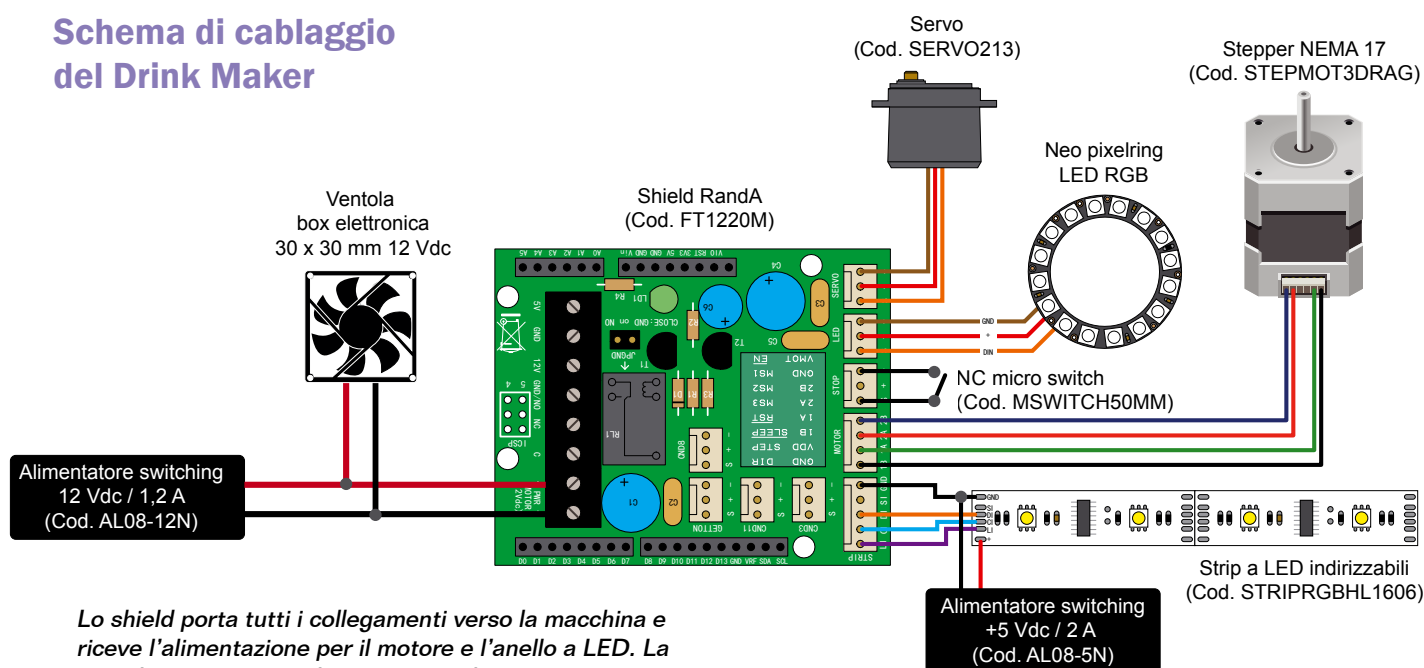
Tabella 1 - Impostazione degli step compiuti dal motore rispetto agli impulsi di comando.

interfacciato a RandA mediante il transistor NPN T1, il cui collettore finisce sulla linea D13, per la quale è opportuno attivare il pull-up interno; il conduttore "caldo" è S, mentre il contatto può essere collegato tra esso e GND (-) o +. Sulla scheda è previsto anche un relé, da utilizzare per varie

funzioni (ad esempio per accendere una luce lampeggiante o un avvisatore acustico quando una bottiglia è vuota), di cui è reso disponibile l'intero scambio e il cui contatto centrale può essere internamente collegato a massa tramite il ponticello JPGND; il relé è comandato dal transistor

NPN T1, a sua volta polarizzato in base alla linea D12 di RandA. Il diodo in parallelo alla bobina di RL1 serve a bloccare le tensioni inverse che ad ogni interdizione del T1 essa genera per reazione e che potrebbero danneggiare la giunzione collettore-base del transistor stesso.

Schema di cablaggio del Drink Maker



Lo shield porta tutti i collegamenti verso la macchina e riceve l'alimentazione per il motore e l'anello a LED. La ventola va montata sul contenitore plastico.

Per quanto riguarda la connessione alla STRIP, è una sorta di bus SPI: la linea CI è il clock della comunicazione seriale, DI il canale dati in uscita da RandA e in ingresso alla strip, SI non è usato (si può evitare) e LI è il segnale di strobe del latch in ingresso all'HL1606. Il protocollo di comunicazione con la strip prevede l'invio in sequenza di bit che permettono di impostare per ciascun LED l'accensione e il colore; il controller indirizza fino a 6 canali con 30 mA di corrente in ciascuno ed è capace di una risoluzione

del colore di 6+1 livelli. Il chip HL1606 comprende, nel set di controlli, le modalità a "sfumatura automatica" per la transizione temporizzata e sincronizzata con alcune sequenze stabilite, da nero a un colore fondamentale, da un fondamentale all'altro o arcobaleno. Nella strip, il controllo seriale è indispensabile perché altrimenti per gestire individualmente i singoli LED che la compongono servirebbero, pur adottando il comando a matrice in multiplex, tantissime linee. Con l'adozione di chip controller come l'HL1606 l'interfaccia si semplifica fino a contare appena tre linee più l'alimentazione (in realtà sono quattro più i 5 volt, ma riusciamo a gestire il controller con tre sole). Il chip HL1606 supporta il collegamento in cascata di altri analoghi, quindi permette di realizzare strip a un numero molto elevato di canali. Ogni uscita del controller del trip pilota i LED con segnali PWM, così da modulare la luminosità nella maniera più efficiente e consentire tutte le possibili sfumature di colore. Quanto al driver per motori passo-passo, è prodotto dalla Futura Elettronica (cod. 3DDRI-

VER) e nello schema dello shield è siglato U1; il 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.

Sebbene 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 (contatti MS1, MS2, MS3 fissi a 1 logico) come mostra lo schema elettrico. Il driver del motore passo-passo viene gestito dalle linee D4, D7, D10 di RandA, che, rispettivamente, abilitano (quando a livello basso) U1, forniscono gli impulsi per l'avanzamento del



Dal lato opposto a quello del motore, la cinghia scorre su un cuscinetto fissato con una squadretta.

Un servocomando, tramite la leva in figura, spinge dal basso il pulsante del dispenser.

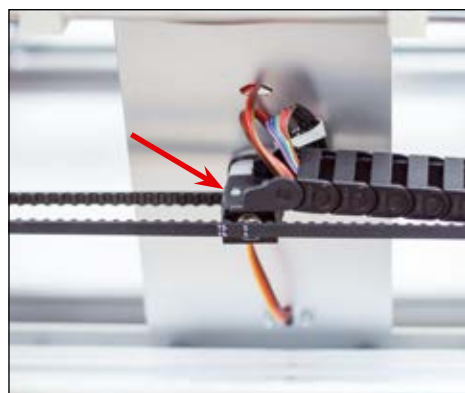
motore (ogni 1 logico su STEP fa compiere al motore uno step, ovvero la frazione impostata) e impostano la direzione di rotazione (DIR). La linea DIR viene definita di volta in volta a seconda che si voglia ottenere la rotazione in senso orario o antiorario e quindi spostare il carrello da una parte o da quella opposta.

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 RandA 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 siccome il motore passo-passo è a 200 step/giro, per far compiere al suo albero un giro il microcontrollore deve inviargli $200 \times 16 = 3.200$ impulsi. 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; questi contatti sono uniti e apparentemente collegati a null'altro, ma in realtà all'interno del driver la linea SLEEP ha un resistore di pull-up, quindi unendo i due contatti si disattivano entrambe le funzioni in maniera permanente. La logica degli ingressi del driver motori 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 $0,3 \times V_{cc}$. L'alimentazione del modulo si ottiene dai 5 volt di RandA, tuttavia la sezione d'uscita, vale a dire quella del doppio ponte, si alimenta con la linea Vmot, cui giungono i 12 V dell'alimentazione principale dello shield. Quanto ai LED da mettere sotto il bicchiere, sono montati in un anello a tecnologia Neopixel (commercializzato da www.adafruit.com con il codice WS2812 5050) a 16 diodi luminosi RGB che possono essere gestiti individualmente, ognuno dei quali può produrre 256 tonalità del proprio colore, determinando un totale di 16.777.216 colori. Neopixel è una soluzione che prevede l'integrazione di un driver e del rispettivo LED RGB in SMD e perciò il comando diretto LED per LED. La frequenza di scansione dei LED è, come per



la strip, di 400 Hz. Il canale dati per la comunicazione è seriale di tipo oneWire e l'alimentazione a 5 volt; la comunicazione avviene a un massimo di 800 kbps. Per l'anello a LED è possibile impostare a piacimento la frequenza di refresh, in modo da rendere impercettibili determinati giochi di luce. Più anelli possono essere collegati in cascata per realizzare vari effetti, ma in questo caso tale configurazione non interessa; sappiate comunque che più anelli si connettono a un unico canale dati, più, fermo restando il massimo data-rate, verrà limitata la frequenza di refresh. Il protocollo di comando dell'anello prevede l'invio di gruppi di



Sotto al carrello si fissa la catena passacavo e il fermo della cinghia, la cui parte più avanzata azionerà la levetta dello switch finecorsa.



Al motore passo-passo va applicato lo switch finecorsa che verrà azionato dal fermo della cinghia posto nella parte inferiore del carrello.

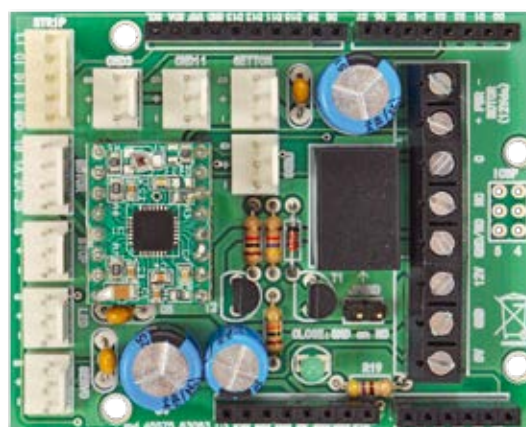
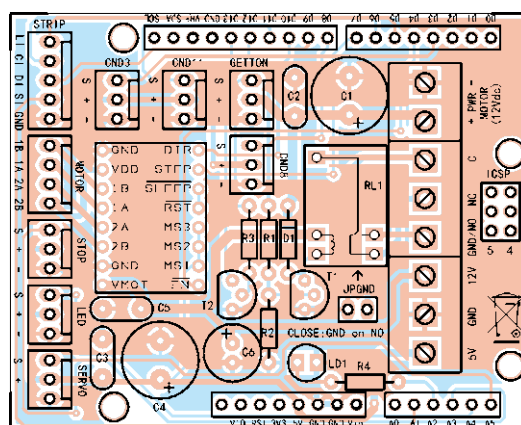
[piano di **MONTAGGIO shield**]

Elenco Componenti:

R1: 4,7 kohm
 R2: 10 kohm
 R3: 1 kohm
 R4: 470 ohm
 C1: 470 μ F 25VL elettrolitico
 C2, C3, C5: 100 nF multistrato
 C4: 470 μ F 25VL elettrolitico
 C6: 220 μ F 25VL elettrolitico
 LD1: LED 3 mm verde
 T1, T2: BC547
 D1: 1N4148
 RL1: Relé 12V singolo scambio
 U1: Modulo driver per step motor (3DDRIVER)

Varie:

- Connettore Board to Wire 5 vie
- Connettore Board to Wire 4 vie
- Connettore Board to Wire 3 vie (7 pz.)
- Strip maschio 2 vie
- Jumper
- Morsetto 2 poli
- Morsetto 3 poli (2 pz.)
- Strip femmina 8 vie (2 pz.)
- Strip Maschio/Femmina 6 vie
- Strip Maschio/Femmina 8 vie (2 pz.)
- Strip Maschio/Femmina 10 vie
- Strip Maschio/Femmina 2x3 vie
- Circuito stampato S1220



tre byte in una stringa di 24 bit, ognuna delle quali contiene lo stato di illuminazione di ciascun colore base (prima gli otto bit del verde, poi quelli del rosso e infine quelli del verde).



Lo spillatore ha un imbocco con guarnizione in gomma multilabbro e un pulsante premendo il quale l'ugello eroga la quantità prefissata.

IL SOFTWARE

Terminata l'analisi dell'hardware possiamo dedicarci al software del progetto, che sostanzialmente prevede il firmware da caricare in RandA e il software vero e proprio che prende posto in Raspberry Pi e girerà su di essa. Questo programma consta di un database, più un'interfaccia utente divisa in due parti:

- un front-end, costituito dalla pagina web che l'utente vede puntando l'indirizzo IP di Raspberry Pi e sulla quale opera per ordinare;
- un back-end (admin) gestito dall'amministratore del sistema e trasparente all'utente.

La parte Admin è divisa a sua volta in tre parti:

- una mostra i cocktail in esecu-

zione e la coda di lavorazione (nome e nome cocktail, più stato attuale, che può essere making –ossia sequenza di preparazione del cocktail avviata– oppure approved, vale a dire



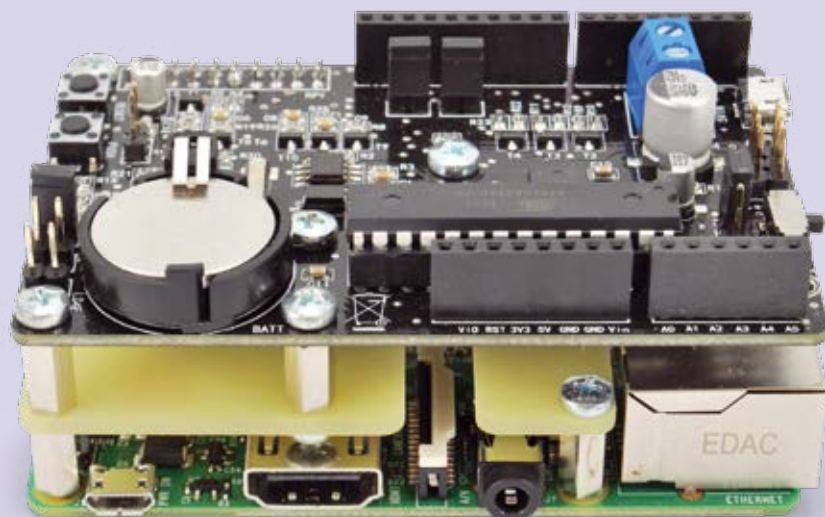
Il pulsante (del dispenser) che fa erogare la bevanda viene azionato dalla leva del servocomando, la quale si solleva e lo preme.

RandA: che cos'è?

L'elettronica che governa la macchina dei cocktail e la interfaccia al computer è basata su Raspberry Pi e su RandA. Per chi non avesse seguito gli articoli in cui l'abbiamo presentata, diciamo che RandA (contrazione dei termini Raspberry and Arduino) è una board per Raspberry Pi con processore Atmel 328, RTC (Real Time Clock), pulsante di accensione e spegnimento programmato, due connettori di alimentazione a 5 Volt (un micro USB e una morsettiere) e connettori laterali compatibili con gli shield Arduino sul lato superiore, e inferiormente un connettore per l'innesto sulla connessione GPIO di Raspberry Pi. Sostanzialmente RandA contiene l'hardware della Arduino Uno privato della sezione di alimentazione e della porta di comunicazione USB, ma con aggiunto l'RTC. RandA consente di sfruttare la dotazione hardware e le enormi potenzialità della piattaforma Linux Raspberry Pi e la quantità di shield e sketch per il mondo Arduino.

Il link tra le due piattaforme è realizzato tramite il connettore di Raspberry Pi, con cui RandA è collegato ma solo per le prime 20 posizioni: questo consente di utilizzare RandA sia su Raspberry Pi B+ che su Raspberry Pi B. La porta seriale è utilizzata sia per la programmazione di Arduino sia per la comunicazione con Raspberry Pi.

Nell'applicazione di queste pagine, Raspberry Pi utilizza Arduino come



periferica configurabile, il cui programma è scelto per gestire motori e sensori della macchina, eseguendo gli ordini della Raspberry Pi e riportandole lo stato. La disponibilità dell'RTC a bordo di RandA permetterebbe di accendere Raspberry Pi e Arduino ad orari programmati, tuttavia in questo caso tale funzione non ci serve; non sfruttiamo neanche la possibilità di accendere Raspberry Pi da RandA, perché la linea preposta (D4) è già destinata allo shield.

La programmazione di RandA può avvenire tramite Raspberry Pi o da rete locale. Su Raspberry Pi è stato installato l'IDE di Arduino per Linux. Tramite l'IDE è possibile programmare la parte Arduino e fare il caricamento (uploading) dello

sketch compilato. L'IDE può essere lanciato da una console linux remota con protocollo SSH oppure collegandosi con tastiera e schermo a Raspberry Pi.

La programmazione di RandA è ottenuta tramite uno script Linux di interfaccia che attiva il reset, permettendo l'uploading degli sketch. La programmazione di RandA in remoto si effettua utilizzando un IDE per PC, modificato per gestire un Arduino collegato in rete simulando una porta remota. Questo IDE lanciato su PC permette di vedere eventuali Arduino remotizzati ed appoggiati su Raspberry Pi. RandA si innesta su Raspberry Pi mediante una struttura intermedia necessaria per ancorare la scheda, come visibile nella foto.

cocktail approvato perché sono disponibili gli ingredienti);

- ingredienti: seleziona quali bottiglie sono caricate, quante parti da 2 cl contiene, posizione dello slot (dispenser) e possibilità di aggiungere ingredienti;
- composizione cocktail (parti, nome, eventuale foto per aiutare l'utente nella scelta, ecc.).

L'utente seleziona il cocktail sull'interfaccia web, gestita da Raspberry Pi. Questa elabora e verifica la possibilità di prepararlo (disponibilità degli ingredienti anche considerando eventuali ordini in coda), quindi se è tutto ok passa ad Arduino la posizione del carrello e quante parti di

bevanda prelevare da ciascuna. Arduino fa solo da controllo dello stepper-motor per spostare il carrello e del servo per azionare lo spillatore.

La barra di LED, iniziata la procedura, indica con una sorta di barra di count-down che abbiamo 10 secondi di tempo per mettere il bicchiere, poi la macchina parte con la propria sequenza.

Per quanto riguarda RandA, il firmware che gira in essa serve semplicemente a interfacciarsi con Raspberry Pi e a convertire le sue richieste in comandi destinati all'attivazione del servocomando e al movimento dello stepper motor.

Per la gestione del servo si usa l'apposita libreria integrata nello

sketch; analogo è il discorso per la gestione della strip, per la quale esiste una libreria Arduino di gestione dell'interfaccia Serial Peripheral Interface (SPI) con o senza linee di Latch e Sync.

La gestione dell'anello a LED è demandata anch'essa a una specifica libreria messa a disposizione da Adafruit, che è il produttore dell'anello.

Nella prossima puntata analizzeremo nel dettaglio il software.

REALIZZAZIONE PRATICA

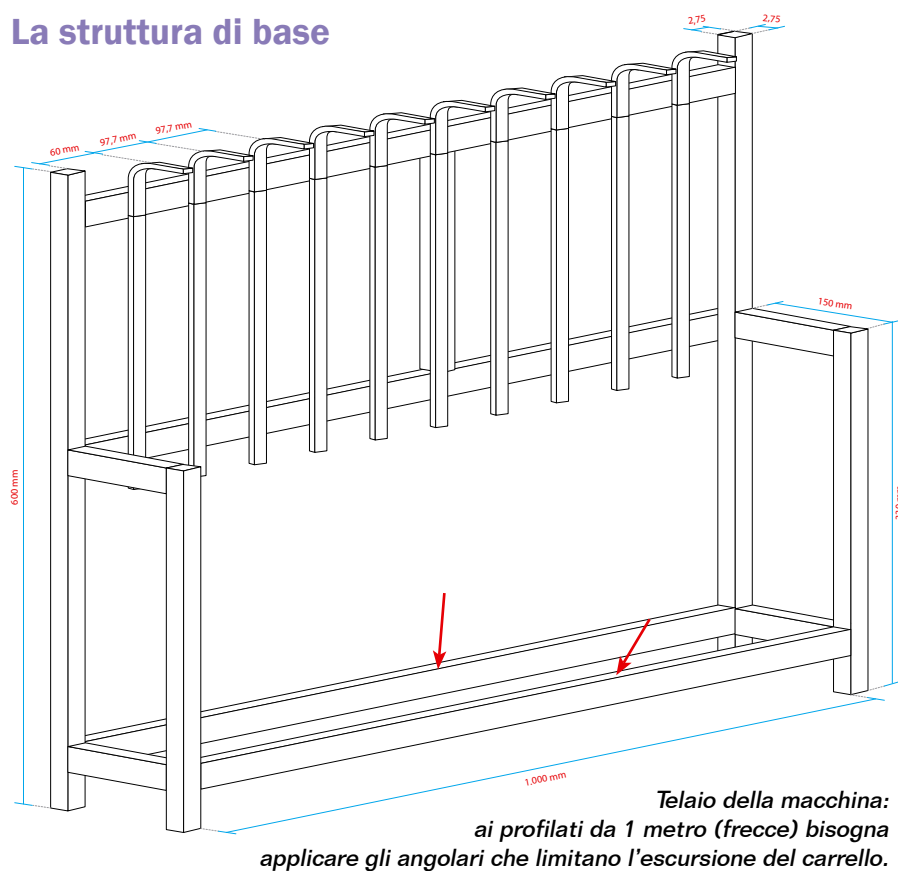
Passiamo adesso alla costruzione della macchina, la cui struttura base si realizza con profilati di alluminio quadro da 27,5x27,5 mm e a L da 29,5x53,6x2,4 mm seguendo il disegno visibile in

La struttura di base

queste pagine; per ragioni di spazio non pubblichiamo tutti i dettagli, ma sul nostro sito www.elettronica.in troverete i file per la lavorazione delle parti con CNC e quelle stampate in 3D che costituiscono alcuni particolari, comunque rimpiazzabili a vostro piacimento. Il telaio vero e proprio consta di due profilati da 27,5x27,5 mm lunghi ognuno 1 metro, uniti da una cornice per lato, fatta con gli stessi profilati ma delle dimensioni di 150 mm (base) per 330 mm (altezza). In realtà il profilato che stanno sul retro delle cornici sono alti 600 mm. L'assemblaggio delle parti si esegue tramite viti, chiudendo ogni estremità con gli appositi tappi da 27,5x27,5 mm. Sulla zona anteriore del sopralzo vanno applicate a vite e a una distanza di 97,7 mm le une dalle altre, le staffe di supporto delle bottiglie, alle basi delle quali bisogna fissare gli erogatori mediante le apposite viti; la staffa più a sinistra deve trovarsi a 60 mm esatti dal lato interno del montante costituito dal profilato a sezione quadra da 27,5x27,5x600 mm. Il carrello è una piastra di alluminio da 192x100 mm spessa 3 mm, lavorata opportunamente con una fresa CNC, cui è appli-



L'anello di LED si applica nel foro sulla parte in plexiglass del carrello.



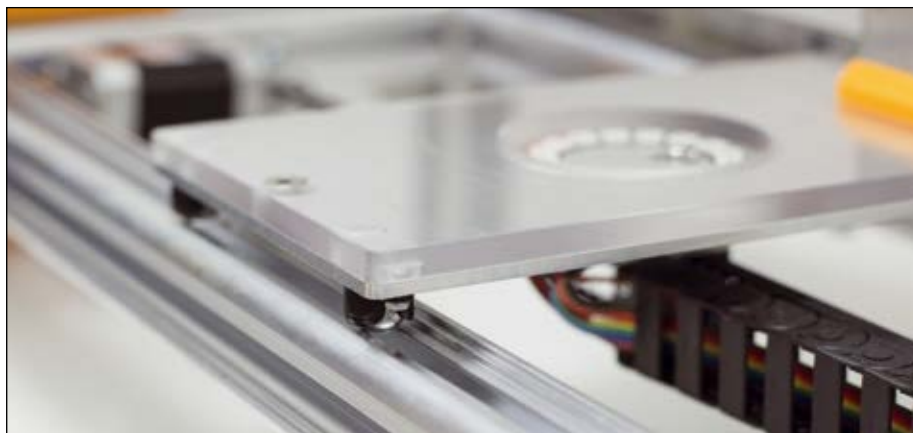
cata una piastra in plexiglass di pari ingombro, lavorata per essere avvitata alla piastra in alluminio sottostante e per ospitare il piattello portabicchiere e l'eventuale anello a LED. Nella parte inferiore del carrello, sui lati corti, vanno applicati due ball-caster con sfera da 10 mm che si possono acquistare su www.futurashop.it con il codice PBCASTER38 e che servono a far scivolare il carrello stesso lungo le cave dei profilati da 1 metro. I ball-caster vanno posizionati a una distanza dai bordi della piastra in alluminio tale da poter scorrere nelle cave. Il movimento del carrello si ottiene fissando la cinghia dentata, tagliata a misura e fatta passare attorno alla puleggia libera del lato destro e a quella dello stepper motor, ad un apposito sostegno da avvitare inferiormente alla piastra di alluminio; lo stesso sostegno, oltre che a fermare la cinghia con una vite, serve a muovere la levetta

dello switch finecorsa; la posizione andrà scelta considerando che lo switch va premuto quando il carrello è tutto all'inizio. Lo switch deve essere fissato, come si vede nelle foto, mediante una delle viti con cui si ferma lo stepper motor. Per evitare la fuoriuscita da sollevamento del carrello, ai profilati da 1 metro si applicano due profilati di alluminio a squadra di pari lunghezza, da 29,5x53,6x2,4 mm come mostrato nelle foto di queste pagine, ovvero con lo spigolo all'esterno. Quanto al cablaggio, seguite lo schema pubblicato in queste pagine, dove trovate il collegamento di tutte le parti con lo shield; per i collegamenti del carrello impiegate del flat-cable dai cui conduttori dovranno passare le eventuali tre connessioni dell'anello a LED, oltre alle connessioni del servocomando. Per evitare che il flat si impigli nella cinghia o penda, nel nostro prototipo

l'abbiamo fatto passare in un passacavo a catena, come vedete nelle foto del prototipo.

L'elettronica di bordo è composta da Raspberry Pi sopra cui è applicata la scheda RandA, che ospita lo shield; il tutto è stato previsto che venga collocato in un contenitore plastico stampato in 3D (con la nostra 3Drag, se l'avete a casa...) nel quale inserirete una piccola ventola a 12 V per il raffreddamento. Se applicate l'anello a LED, fatene passare il cavetto sotto al piatto fissandolo con apposite graffette (sul nostro sito trovate i file per la stampa 3D in PLA), le stesse da usare per fermare la strip a LED sul profilato a L posteriore. Sistemate le varie parti, dovete applicare le alimentazioni, ricordando che serve un alimentatore a 12 Vcc - 2,5 A per lo shield e uno da 5 V, 1,5 A per RandA (che tramite gli strip inferiori alimenta Raspberry Pi); i cavi dei due vanno applicati alle rispettive connessioni a morsettiera. Lo sketch di RandA viene programmato tramite l'IDE modificato, scaricabile dal nostro sito www.elettronica.in, che permette di effettuare il caricamento attraverso la rete, passando da Raspberry Pi, quindi dovete collegare a un PC in rete

quest'ultima. Ricordate che se volete utilizzare anche la strip a LED, dovete prevedere per essa un apposito alimentatore, che è quello venduto a corredo della strip stessa. Invece l'eventuale anello a LED trae alimentazione dalla linea 5V dello shield, alimentata a sua volta da RandA. Prima di applicare lo shield a RandA, dovete aprire il jumper J2 su quest'ultima, che normalmente viene utilizzato per consentire alla logica Arduino on-board di accendere e spegnere Raspberry Pi. In questa applicazione deve essere aperto, anche perché la linea D4 di Arduino è utilizzata attraverso lo shield- per gestire l'enable del controller motori. Notate che prima di utilizzare il sistema potrebbe essere necessario regolare il trimmer miniatura del controller motori; allo scopo assicurarsi che l'alimentatore sia collegato alla scheda e acceso, prendere il multimetro e un cacciavite molto piccolo in ceramica (è possibile utilizzare anche un cacciavite metallico ma in questo caso bisogna stare molto attenti a toccare solamente il trimmer dei driver onde evitare di creare cortocircuiti) quindi impostare sul multimetro la portata 2 V in continua o similare.



Il carrello scorre nelle cave dei profilati da 1m grazie a ball-caster.

Guarda Drink Maker in azione su YouTube!



Vuoi vedere in azione Drink Maker mentre prepara i cocktail come un vero barman?



Nulla di più semplice: collegati alla pagina <https://youtu.be/AXpxWlxJLmQ>. Potrai anche vedere i video di moltissimi altri nostri progetti proposti in passato.

Tenere il puntale nero in contatto con il morsetto negativo di alimentazione della scheda e il puntale rosso sulla piazzola posizionata sopra al chip del driver. Con un cacciavite ceramico regolate il trimmer del driver fino a leggere sul multimetro una tensione di 0,425 V. Fatto ciò potete togliere il tester e staccare l'alimentazione, quindi chiudere il box contenente l'elettronica. ■



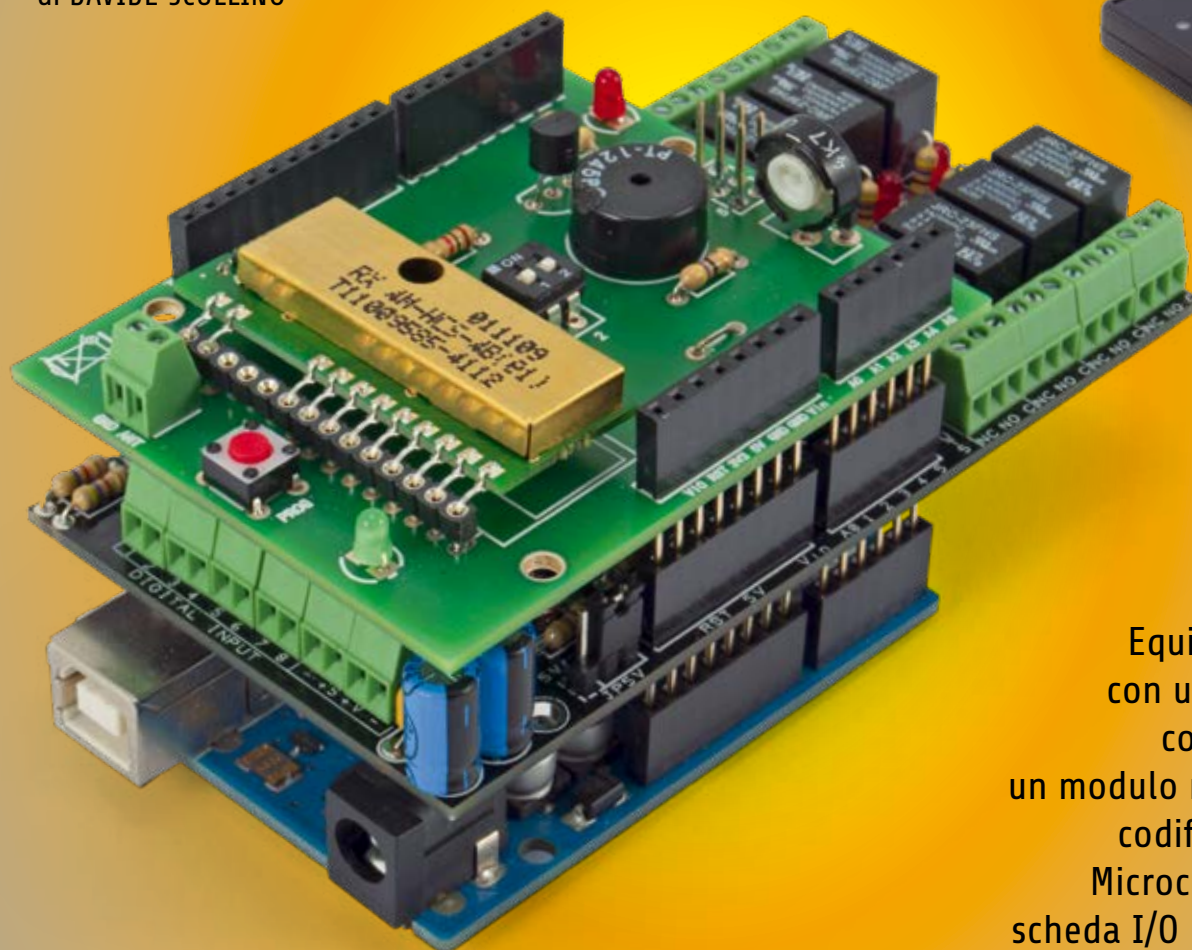
per il MATERIALE

Il kit completo per realizzare la Drinkmaker con 5 dosatori è disponibile presso Futura Elettronica (cod. DRINKMAKER) al prezzo di 899,00 Euro. Il drink shield (cod. FT1220K - Euro 23,00), la board RandA (cod. RANDA - Euro 39,00) e Raspberry Pi 2 Tipo B (cod. 8326274RS - Euro 44,00) sono acquistabili anche separatamente. 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>

RADIOCOMANDO 8 CANALI 433,92 MHZ

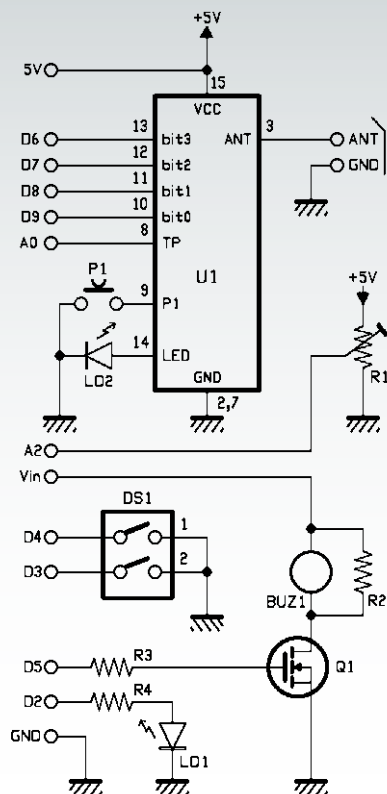
di DAVIDE SCULLINO



Equipaggiato con uno shield contenente un modulo ricevitore codificato HCS Microchip e una scheda I/O Expander I²C-Bus, consente di realizzare radiocomandi a 8 canali basati su Arduino, estendibili a 12.

La gran quantità di dispositivi radiocomandabili presenti ormai nelle case e non solo, pone il problema di dover avere a che fare con svariati trasmettitori e di fare i conti con ciò che ne deriva, primo fra tutti ricordarsi a cosa

serve uno, a cosa l'altro, dove sia l'uno e dove l'altro. Se ci si deve limitare ad azionare un'elettroserratura, comandare un paio di tapparelle motorizzate o un apricancello, accendere e spegnere una luce, può bastare un trasmettitore a due o quat-



tro canali, ma se i sistemi sono di più, è necessario affidarsi a radiocomandi dotati di molti canali. Siccome non è facile trovare in commercio radiocomandi a più di 4 canali, abbiamo pensato di progettarne uno scalabile da 8 a 12 canali, dotato di uscite a relé; a gestirlo, l'ormai immanicabile Arduino, che si avvale di uno shield RF basato su un ricevitore ibrido con integrata una decodifica HCS Microchip e di un'espansione 8 ingressi e 8 uscite a relé gestita da I²C-Bus.

L'unità basata su Arduino è il ricevitore; il trasmettitore non c'è bisogno di costruirlo, perché è un TX palmare già pronto all'uso ed è possibile sceglierlo tra le versioni ad un canale (codice HCS-TX-1), a due canali (codice HCS-TX-2), a tre canali (codice HCS-TX-3), a quattro canali (codice TX-4M-HCS) a sei canali (codice TX-6M-HCS) o a dodici canali (codice TX-12CH), tutti prodotti dall'Aurel e che si acquistano già montati presso Futura Elettronica (www.futurashop.it). Ci concentreremo sul trasmettitore a dodici canali, in quanto permette la gestione di tutti i relé presenti sulla scheda di espansione. Il TX-12CH contiene un modulo trasmettente RF operante in UHF alla frequenza di 433,92 MHz, in modulazione d'ampiezza, con una potenza di 1 milliwatt, sufficiente, con un ricevitore radio come quello impiegato nel nostro sistema, a coprire alcune decine di metri di distanza anche in presenza di ostacoli (purché non si tratti di pareti in cemento armato). Della trasmettente non descriviamo l'elettronica, in quanto si tratta di un prodotto già pronto, che, per la tecnologia impiegata, è più facile comperare che costruirsi; ci limitiamo a dire che contiene un encoder HCS301 attivato da una tastiera a matrice di 4 righe per 3 colonne. Il sistema è

codificato mediante una codifica ad elevata sicurezza, perché variabile: si tratta dell'HCS della Microchip, un sistema rolling-code che già avete conosciuto negli anni passati perché impiegato in numerosi radiocomandi e capace di garantire sia l'esclusività del comando anche in ambienti dove operano molti radiocomandi basati sullo stesso sistema, sia la tutela contro l'effrazione del codice.

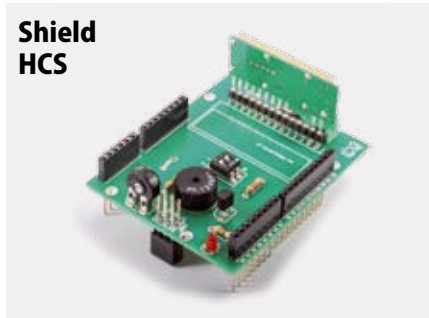
LO SHIELD HCS

L'unità ricevente del nostro radiocomando è implementata in uno shield chiamato RF Shield che si basa su un modulo Aurel anch'esso: si tratta dell'RX 4MHCS-4B, un ibrido supereterodina contenente un ricevitore radio UHF sintonizzato sui 433,92 MHz, provvisto di demodulatore d'ampiezza OOK e squadratore degli impulsi in uscita, completato da una decodifica che gli permette di tradurre i comandi inviati dal trasmettitore. La decodifica è realizzata da un microcontrollore che implementa il protocollo HCS e riconosce i comandi inviati dal trasmettitore; in pratica il decoder ha quattro linee d'uscita che esprimono in formato binario, su una word di 4 bit il numero del canale, ovvero il numero del tasto che è stato premuto sul TX. Ciascuna delle uscite è

**Scheda
Arduino**



**Shield
HCS**



**Shield I/O
Expander**



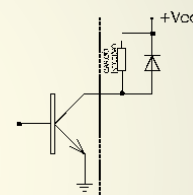
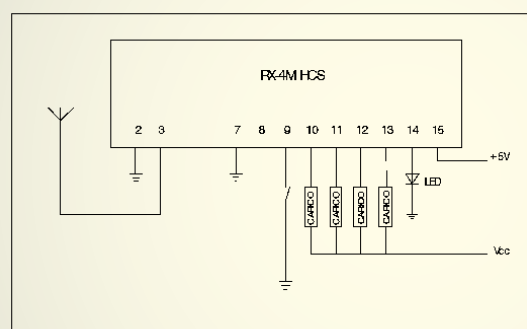
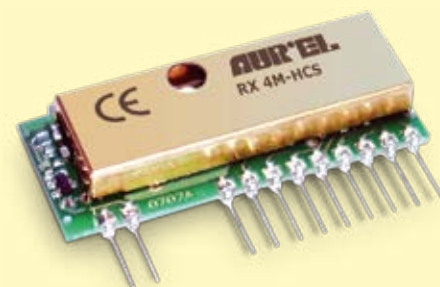
Il ricevitore RX 4MHCS-4B

interfacciata con un transistor NPN, quindi è a logica invertita, ossia attiva a livello logico basso; i transistor delle uscite sono tutti configurati come open-collector. Il modulo dispone di un ingresso ed un'uscita rispettivamente per un pulsante e per connettere un LED, entrambi necessari alla procedura di autoapprendimento; infatti, essendo un rolling-code, il radiocomando richiede che il ricevitore venga abbinato al trasmettitore, ovvero che apprenda il codice base e quant'altro serve per sincronizzarsi con esso. Della procedura di abbinamento ci occuperemo più avanti.

Chiaramente per decifrare i comandi serve una logica, discreta o programmabile, che funzioni da decoder 4 a 16, ovvero, nel nostro caso, 4 a 12, dal momento che abbiamo dodici pulsanti sul trasmettitore. Per semplificare il circuito, abbiamo lasciato il compito ad Arduino, il quale mediante l'apposito sketch legge i dati in arrivo dall'ibrido Aurel e li elabora in modo da usarli per generare i comandi I²C-Bus con i quali gestisce lo shield a relé.

A tale riguardo va precisato che lo sketch contempla la gestione dei codici binari corrispondenti alle prime 8 combinazioni, vale a dire che risponde ai primi otto tasti del ricevitore TX-12CH con comandi I²C-Bus diretti allo shield I/O Expander, mentre riconosce i codici binari dei rimanenti tasti (9, 0, Shift ed Enter) del trasmettitore come comandi di reset delle uscite a relé quando queste funzionano in modalità bistabile. Naturalmente se ve la sentite di mettere mano allo sketch, potete riassegnare i codici dei tasti arrivando così a gestire dodici uscite. Nello schema elettrico, il modu-

Il modulo impiegato nell'unità ricevente del nostro radiocomando è un ibrido operante in UHF (a 433,92 MHz) completo di demodulatore AM, squadratore del segnale dati in uscita e decodificatore per l'algoritmo Keeloq Microchip usato dai trasmettitori con encoder HCS30x.



La decodifica restituisce su 4 bit, corrispondenti ai piedini 10 (Uscita Ch1), 11 (Uscita Ch2), 12 (Uscita Ch3) e 13 (Uscita Ch4) il numero del pulsante premuto sul trasmettitore a 12 canali espresso in binario. La tabella mostra lo stato logico dei piedini corrispondenti alle quattro uscite dati in corrispondenza della pressione di uno dei dodici tasti del trasmettitore. Quando il modulo non riceve alcun segnale dal trasmettitore, le uscite si presentano tutte a livello alto (open).

Pulsante del TX	10	11	12	13
1	0	1	1	1
2	1	0	1	1
3	0	0	1	1
4	1	1	0	1
5	0	1	0	1
6	1	0	0	1
7	0	0	0	1
8	1	1	1	0
9	0	1	1	0
0	1	0	1	0
SHIFT	0	0	1	0
ENTER	1	1	0	0

lo RF è siglato U1 e, come vedete, viene alimentato tramite il 5V di Arduino ed è contornato dal pulsante P1 e dal LED LD2; i contatti d'antenna finiscono su una morsettiera miniatura che permette di connettere l'antenna ricevente, sia essa uno stilo lungo 17 cm (in questo caso si collega al solo morsetto ANT) o un'antenna esterna (nel qual caso alla morsettiera si collega il cavo schermato, con lo schermo a GND e il polo caldo ad ANT). I quattro bit d'uscita e il piedino TP dell'ibrido vengono portati

ad altrettanti pin di Arduino: per l'esattezza, i bit 0, 1, 2, 3 dell'uscita binaria vanno rispettivamente ai pin digitali D9, D8, D7, D6, mentre TP è connesso al pin analogico A0 di Arduino. Quest'ultimo viene usato per rilevare l'intensità del segnale RF, giocando sul fatto che l'ibrido Aurel fornisce sul piedino 8 una tensione proporzionale all'intensità del segnale di media frequenza, dalla quale si desume l'ampiezza del segnale RF in arrivo dall'antenna. Lo sketch per Arduino attualmente si li-

Tabella 1 - Significato dei dip-switch.

SW1	FUNZIONAMENTO USCITE
	Monostabile: l'uscita segue il relativo pulsante del trasmettitore.
	Monostabile: l'uscita si attiva alla pressione del relativo pulsante e torna a riposo trascorso il tempo definito dal trimmer dal rilascio del pulsante.
	bistabile (in questa modalità la posizione del dip2 è irrilevante).

mita a leggere questo valore tramite l'ADC e a stamparlo sulla seriale in modo che sia visibile con Serial Monitor dell'IDE di Arduino, tuttavia se volete modificarlo potete implementarla, ad esempio per rilevare l'affidabilità delle informazioni fornite dall'U1, ma anche per capire se il radiocomando, quando non riceve dei comandi, non lo fa perché è sottoposto a jamming (vale a dire che è "accecato" da un segnale RF a 433,92 MHz che impedisce la decodifica dei dati); questa situazione può essere rivelata dal fatto che il segnale sul piedino 8 è forte ma su bit0÷bit3 non c'è alcun cambiamento dei valori binari.

Lo shield ospita, oltre ad U1, un trimmer (alimentato dagli stessi 5 volt che alimentano l'ibrido), di cui Arduino legge il cursore tramite la linea analogica A2 (allo scopo di impostare il tempo di ritardo in rilascio nella modalità monostabile ritardata) e un LED (LD1) che viene acceso dalla linea D2 di Arduino quando viene ricevuto un codice riconosciuto valido; ancora, a bordo dello shield si trovano il cicalino BUZ1 (pilotato dalla linea D5 di Arduino mediante il MOSFET Q1) usato semplicemente per emettere una nota acustica all'accensione del sistema (modificando lo sketch potete assegnarlo ad ulteriori funzioni) e il dip switch a 2 poli con il quale si impostano le modalità di funzionamento del ricevitore.

Con riferimento alla **Tabella 1**, vediamo il significato dei dip: il dip 1 è quello che definisce se le uscite a relé dello shield debbano funzionare in modo bistabile o astabile: quando è OFF (aperto) tutti i canali, ogniquale riceve un comando, cambia-

no la loro condizione rispetto all'istante precedente, perciò se un relé è a riposo si eccita e se è eccitato torna a riposo. Chiudendo il dip 1 (ON) tutte le uscite lavorano ad impulso. A questo punto, chi decide come debba essere gestita la funzionalità astabile è il dip 2: se è ON (chiuso) ogni relé segue il tasto del trasmettitore; ciò significa che se sul TX premiamo il tasto 2, scatta RL2 e resta eccitato fin quando non rilasciamo detto pulsante. In realtà, mantenendo premuto un tasto, il TX trasmette al massimo per 25 secondi consecutivi. In questa modalità si può attivare un solo relé alla volta. Se invece si pone il dip 2 in OFF (aperto) le uscite a relé vengono attivate appena si preme il tasto corrispondente sul TX, ma rilasciando questo non tornano a riposo subito e lo fanno, invece, trascorso l'intervallo di tempo impostato con il trimmer R1.

Il tempo di ritardo di rilascio del relé è pari al valore letto dall'A/D converter di Arduino (0 a zero volt e 1.023 a 5 volt) dal cursore del trimmer, espresso in millisecondi, per 5, quindi con il cursore tutto verso l'estremo collegato a massa il ritardo è nullo (il relé ricade al rilascio del pulsante del trasmettitore) mentre con il cursore a metà corsa (2,5 volt) siccome la lettura dell'ADC è teoricamente 512, il ritardo è 2,56 secondi; se il cursore viene portato tutto verso l'estremo collegato al +5V, il ritardo è il massimo, vale a dire 5,12 secondi.

LO SHIELD I/O EXPANDER

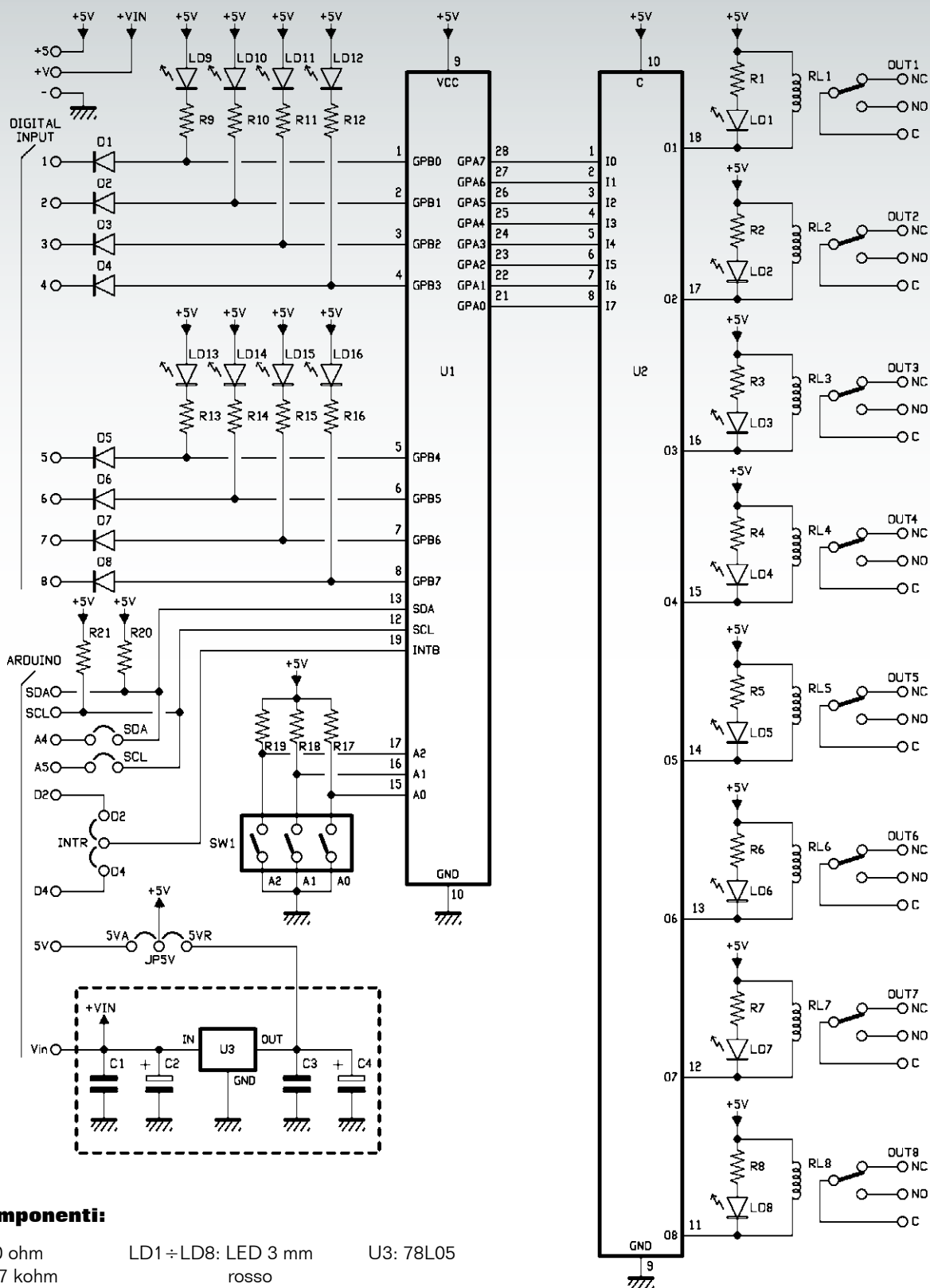
Detto ciò, passiamo ad analizzare rapidamente lo shield I²C I/O Expander, circuito che abbiamo già impiegato in altri progetti e del quale riportia-

Il trasmettitore TX-12CH

Il trasmettitore RF da noi utilizzato per comandare il ricevitore è un palmare prodotto dall'Aurel, dotato di encoder a codice dinamico (HCS301) il cui segnale d'uscita seriale modula in ampiezza un oscillatore SAW a 433,92 MHz operante con una potenza di 1 milliwatt. Le caratteristiche del dispositivo sono qui elencate.

- Frequenza operativa: 433,92 MHz
- Potenza irradiata: 1 mW
- Modulazione: OOK (AM)
- Alimentazione: 2 pile a bottone
- Assorbimento: 10 mA
- Autonomia: 3÷5 anni
- Dimensioni 54 x 85,5 x 10,4 mm





[schema elettrico dello **SHIELD I/O EXPANDER**]

Elenco Componenti:

R1 ÷ R16: 470 ohm
 R17 ÷ R21: 4,7 kohm
 C1, C2: 100 nF
 C3: 100 μ F 16 VL
 elettrolitico
 C4: 100 μ F 16 VL
 elettrolitico
 D1 ÷ D8: 1N4148

LD1 ÷ LD8: LED 3 mm
 rosso
 LD9 ÷ LD16: LED 3 mm
 verde
 SW1: Dip-Switch 3 vie
 RL1 ÷ RL8: Relé
 miniatura 5V
 U1: MCP23017-E/SP
 U2: ULN2803

U3: 78L05

Varie:

- Zoccolo 14+14
- Zoccolo 9+9
- Strip maschio/femmina
 3 poli (2 pz.)
- Strip maschio/femmina
 6 poli (1 pz.)

- Strip maschio/femmina
 8 poli (2 pz.)
- Strip maschio/femmina
 10 poli (1 pz.)
- Strip maschio 3 poli
 (2 pz.)

- Jumper (2 pz.)
- Morsetto 2 poli passo
 2,54 mm (6 pz.)
- Morsetto 3 poli passo
 2,54 mm (8 pz.)
- Circuito stampato

mo il solo schema elettrico; lo shield si basa sull'integrato MCP23017 (siglato U1) prodotto da Microchip: si tratta di un I/O Expander che dispone in totale di 16 pin GPIO (suddivisi in due porte chiamate A e B) configurabili singolarmente come ingresso o uscita.

Nel nostro schema abbiamo deciso di settare tutti gli 8 pin della porta A come uscite e gli 8 pin della porta B come ingressi. Le uscite finiscono sull'integrato U2 ULN2803, un componente progettato appositamente come driver per relé (al suo interno dispone di diversi stadi a transistor che permettono di pilotare

senza problemi le bobine dei relé). Gli ingressi, invece, (porta B) sono collegati ad apposite sezioni circuitali che realizzano gli input digitali; facciamo notare che nell'applicazione descritta in questo articolo non vengono utilizzati, ma per chiarezza li analizziamo lo stesso in quanto sicuramente saranno utili in progetti futuri. Ogni ingresso dispone di un diodo di protezione, una resistenza di pull-up ed un LED. Quando l'ingresso è a livello basso (a massa) il relativo LED risulta acceso, mentre l'I/O Expander segnala che l'ingresso è a "0"; viceversa quando l'ingresso è

nello stato logico alto, il relativo LED è spento e l'I/O Expander indica che l'ingresso è a "1".

La gestione dell'MCP23017 avviene tramite protocollo I²C Bus e scrittura/lettura di appositi registri interni.

Come noto, il bus I²C è realizzato dalle linee SDA (ossia dati) ed SCL (clock); sull'integrato sono disponibili sui pin 13 e 12 e, come da schema, vengono portate sui corrispondenti pin della scheda Arduino.

L'MCP23017 offre la possibilità di generare un interrupt a fronte della variazione di almeno un pin programmato come input; nel nostro caso non abbiamo sfruttato questa funzione, che però potrebbe essere utile per gestire da Arduino gli input in modalità interrupt e non polling (il che assicura più velocità senza appesantire il bus I²C). Chiaramente, per sfruttare questa caratteristica è necessario scrivere opportunamente il firmware.

Nello shield abbiamo previsto la possibilità di portare (tramite ponticello) il pin di interrupt dell'MCP23017 (INTB, pin 19) in ingresso su due differenti pin di Arduino (D2 o D4) in modo da adattarlo il più possibile al reale utilizzo.

L'indirizzo I²C-Bus dell'MCP23017 viene selezionato tramite dip-switch SW1; in esso sono presenti 3 dip, pertanto risulta possibile collegare fino a 8 diversi shield senza che interferiscano tra di loro.

Infine è presente una sezione di alimentazione composta dal regolatore U3 (integrato L78L05) che trasforma la tensione di ingresso Vin ai 5 volt necessari per far funzionare tutto il circuito. Segnaliamo anche che tramite il ponticello JP5V è possibile scegliere di bypas-

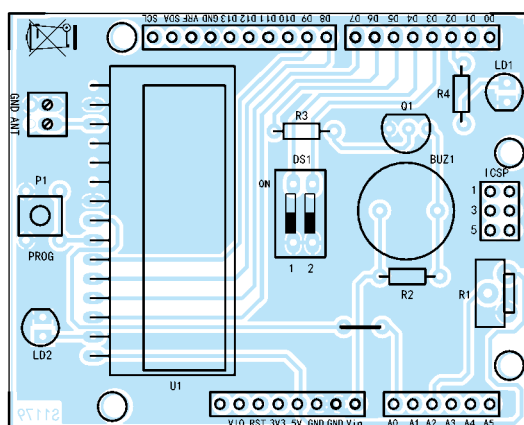
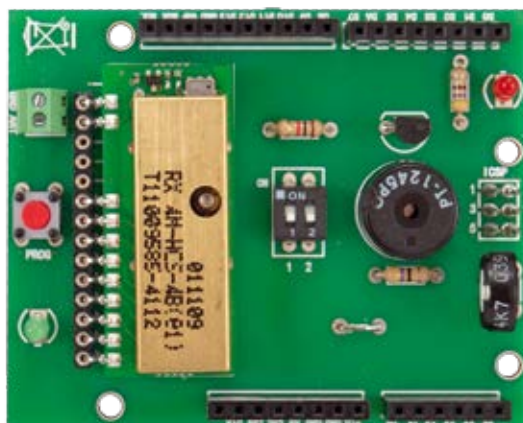
[piano di montaggio **SHIELD HCS**]

Elenco Componenti:

R1: Trimmer 4,7 kohm MV
R2: 47 ohm
R3: 1 kohm
R4: 470 ohm
U1: RX-4MHCS-4B
Q1: BS170
LD1: LED rosso 3 mm
LD2: LED verde 3 mm
P1: Microswitch
BUZ1: Buzzer 12V senza elettronica
DS1: Dip switch 2 vie

Varie:

- Morsettiera 2 poli passo 2,54mm
- Strip Maschio/Femmina 3 vie (2 pz.)
- Strip Maschio/Femmina 6 vie (1 pz.)
- Strip Maschio/Femmina 8 vie (2 pz.)
- Strip Maschio/Femmina 10 vie (1 pz.)
- Spezzone filo 17cm
- Circuito stampato



sare il regolatore ma utilizzare direttamente i 5V forniti dalla scheda Arduino.

LIBRERIE ARDUINO

Per gestire lo shield I/O Expander è prevista una libreria Arduino che fornisce tutte le routine necessarie alla rilevazione degli shield collegati e alla relativa gestione degli I/O (potete scaricare tale libreria dal nostro sito www.elettronica.in.it). La comunicazione con l'MCP23017 è basata sull'I²C-Bus, pertanto la libreria utilizza "Wire.h" di Arduino; è presente una funzione "begin(int i2cAddress)" per inizializzare il singolo shield identificato tramite indirizzo I²C, ed anche una "init()" per programmare correttamente i registri interni del chip secondo le nostre necessità, ed infine una "pinMode(int pin, int mode)" per indicare se i singoli pin di I/O sono input o output.

Oltre alle funzioni di inizializzazione, le istruzioni che ci interessano sono le "digitalRead(int pin)", "digitalWrite(int pin, int val)", "digitalWordRead()" e "digitalWordWrite(word w)" che, rispettivamente, permettono di leggere lo stato di un singolo pin di input, scrivere un singolo pin di output, leggere lo stato di tutti i pin di input (otto, nel nostro caso) ed infine scrivere lo stato di tutti i pin di output (sempre otto).

Come anticipato, la gestione degli input può avvenire tramite interrupt; è pertanto presente un'apposita funzione "pinDisableINT(int pin)" che permette di configurare il singolo pin come generatore di interruzioni o meno (in particolari applicazioni potrebbe essere utile avere la possibilità di non impostare tutti i pin di input

La codifica Keeloq è una delle più utilizzate nei radiocomandi ad elevata sicurezza perché implementa un algoritmo di variazione del codice praticamente non ripetibile e invariabile; costruire il decoder è abbastanza semplice, perché la Microchip rende disponibile il software da implementare nel micro che svolge questa funzione. Sebbene venga utilizzato da gran parte dei fabbricanti di radiocomandi, l'HCS301 garantisce un'elevatissima sicurezza d'uso perché permette di realizzare prodotti personalizzati: infatti ogni costruttore ha la possibilità di personalizzare la propria produzione con un Manufacturer Code a 64 bit, che rappresenta uno dei dati di partenza per generare la chiave di crittografia con la quale viene generata la parte variabile del treno d'impulsi a 66 bit prodotto dall'encoder HCS301. Ogni singolo chip viene inoltre identificato da un codice univoco a 28 bit denominato, appunto,

Serial Number. Anche questo dato viene memorizzato in fase di programmazione del chip da parte del costruttore del trasmettitore. Per poter funzionare, un Encoder e un Decoder devono avere lo stesso Manufacturer Code, altrimenti il codice variabile che il ricevitore (il decodificatore) si aspetta sarà sempre diverso da quello generato dall'encoder. Il Serial Number è un codice progressivo a 28 bit col quale vengono differenziati i chip HCS in fase di programmazione. Il Serial Number consente dunque di distinguere un trasmettitore da un altro. In fase di accoppiamento tra TX e RX, questo codice viene appreso dal ricevitore; durante il normale funzionamento, l'encoder verifica come prima cosa che il Serial Number del TX che trasmette sia tra quelli memorizzati e solo in questo caso procede a decodificare e verificare gli altri dati contenuti nella stringa in arrivo.

Formato stringa di dati generata dal chip HCS301

Key Info (6 bit)	Serial Number (28 bit)	Campo variabile (32 bit)
---------------------	---------------------------	-----------------------------

come generatori di interrupt, ma farlo selettivamente). Quanto allo shield HCS, non richiede librerie ma viene gestito direttamente dallo sketch.

REALIZZAZIONE PRATICA

Descritto il funzionamento dell'hardware, passiamo alla realizzazione delle schede occorrenti: per quanto riguarda lo shield I/O Expander, è possibile acquistarlo da montare presso la Futura Elettronica (www.futurashop.it) oppure autocostruirlo; in tal caso, avendone già descritto il montaggio nel fascicolo n° 181 (novembre 2013) rimandiamo ad esso per tutte le informazioni del caso. In queste pagine ci concentriamo,

piuttosto, sulla realizzazione dello shield HCS; la costruzione dello shield ricevitore è molto semplice perché richiede componenti a montaggio tradizionale e la preparazione di un circuito stampato a singola lamina, il quale si costruisce per fotoincisione a partire dalla traccia lato rame che potete scaricare gratuitamente dal nostro sito www.elettronica.in.it.

In possesso dello stampato, iniziate il montaggio con le resistenze e i LED, rammentando che il catodo è il terminale situato dal lato smussato del contenitore, quindi disponete il dip-switch, il pulsante P1, il cicalino piezo (è del tipo senza elettronica, quindi non ha

Listato 1

```
#include <Wire.h>
#include <IOExpanderMCP23017.h>

const int bit0 = 9;
const int bit1 = 8;
const int bit2 = 7;
const int bit3 = 6;
const int rssi = A0;
const int buzz = 5;
const int dip1 = 4;
const int dip2 = 3;
const int time = A2;
const int led = 2;

byte puls=0;
byte rele=0;
boolean found=0;

IOExpanderMCP23017 IOExpander;
char IOExpanderAddress = 0;

void setup() {
  Serial.begin(9600);
  pinMode(buzz, OUTPUT);
  pinMode(led, OUTPUT);
  pinMode(bit0, INPUT);
  pinMode(bit1, INPUT);
  pinMode(bit2, INPUT);
  pinMode(bit3, INPUT);
  pinMode(rssi, INPUT);
  pinMode(dip1, INPUT);
  pinMode(dip2, INPUT);
  pinMode(time, INPUT);
  digitalWrite(bit0, HIGH);
  digitalWrite(bit1, HIGH);
  digitalWrite(bit2, HIGH);
  digitalWrite(bit3, HIGH);
  digitalWrite(dip1, HIGH);
  digitalWrite(dip2, HIGH);
  tone(buzz, 440);
  delay(500);
  noTone(buzz);

  Wire.begin();

  // Avvio libreria MCP23017
  IOExpander.begin(0);
  if (IOExpander.init() == true){
    Serial.println("Indirizzo 0 Ok");
  }
  else
  {
    Serial.println("Indirizzo 0 Errore");
  }

  Serial.println("RF Shield");
}

void loop(){

  int valrssi=analogRead(rssi);
  Serial.print("RSSI: "); Serial.println(valrssi);

  int valtime=(analogRead(time)*5);
  Serial.print("Time: "); Serial.println(valtime);

  // Serial.print("bit0 "); Serial.println(digitalRead(bit0));
  // Serial.print("bit1 "); Serial.println(digitalRead(bit1));
  // Serial.print("bit2 "); Serial.println(digitalRead(bit2));
  // Serial.print("bit3 "); Serial.println(digitalRead(bit3));
```

polarità) e il MOSFET e, infine, il modulo ibrido U1. Per tutti i componenti polarizzati seguite l'orientamento indicato nel piano di montaggio; quanto all'ibrido RX 4MHCS-4B, entra nello stampato solo nel verso giusto, quindi non ve ne dovete preoccupare. Se decidete di montarlo su uno zoccolo, inserite e saldate nelle apposite piazzole un pin-strip a 15 contatti del quale taglierete il terzo, quarto e quinto piedino. Siccome in questo caso è possibile invertire il modulo ibrido, ricordate che la posizione corretta dell'RX 4MHCS-4B è con lo schermo metallico rivolto all'esterno dello shield.

Per completare lo shield ricevitore, montate i connettori femmina a passo 2,54 mm (con piedini lunghi) per l'inserzione in Arduino: ve ne servono uno da 6 poli, due da 8 poli ed uno da 10. Notate che sullo shield è previsto di montare sei punte (2 file da tre poli) lunghe 20 mm per riportare le connessioni ICSP dello shield sottostante o di Arduino al di sopra dello shield. Terminate con la morsetteria miniatura 2,54 mm per collegare i punti ANT cui applicherete un'antenna, che può essere un semplice spezzone di filo di rame rigido lungo 17 cm o un'antenna accordata: se usate lo spezzone di filo stagnatelo nella piazzola che porta al piedino 3 dell'ibrido, mentre se pensate di adoperare un'antenna ground-plane o una direttiva, connettetela mediante cavo schermato coassiale RG58, la cui calza metallica andrà al terminale di massa.

In possesso degli shield e di Arduino, dovete montare sopra quest'ultima scheda prima l'I/O Expander e poi lo shield HCS: avete così realizzato il ricevi-

tore. Impostate i dip-switch in base al funzionamento desiderato e fornite alimentazione al sistema mediante il connettore plug di Arduino, utilizzando un alimentatore capace di erogare 12 volt in continua (meglio se stabilizzati) e una corrente di 400 mA.

COME SI USA

Bene, giunti a questo punto non resta che collaudare e programmare il radiocomando: una volta in possesso del trasmettitore, alimentate Arduino, quindi verificate che tutti i relé dello shield I/O Expander siano a riposo e che i LED siano spenti. Affinché il ricevitore risponda al trasmettitore di cui siete in possesso, dovete abbinare i due, fermo restando che TX ed RX devono avere lo stesso manufacture code in quanto altrimenti cambia la parte basilare del codice e l'abbinamento non riesce; acquistando trasmettitore e ricevitore dalla ditta Futura Elettronica, non ci saranno problemi. Per l'abbinamento bisogna premere e rilasciare il pulsante di apprendimento (P1) posto sullo shield HCS; fatto ciò, il led LD2 lampeggia rapidamente per 10 secondi. Durante questo tempo, bisogna premere uno qualsiasi dei dodici tasti del trasmettitore, ovviamente restando a una distanza che permetta la ricezione ottimale del segnale (di solito l'apprendimento si fa sul posto). L'invio del comando viene segnalato dall'accensione del LED posto sul pannello frontale del TX.

L'avvenuta memorizzazione del codice base e del sincronismo del trasmettitore vengono indicate dall'LD2 dello shield HCS con un lampeggio lungo. Notate che non serve ripetere l'operazione con gli altri tasti, perché basta trasmettere con un pulsante per essere certi che tutti gli altri vengano rico-

Listato 1

```
puls=0;
bitWrite(puls,0,digitalRead(bit0));
bitWrite(puls,1,digitalRead(bit1));
bitWrite(puls,2,digitalRead(bit2));
bitWrite(puls,3,digitalRead(bit3));
puls=puls^B00001111;
// Serial.print("Puls "); Serial.println(puls,BIN);
Serial.print("Puls "); Serial.println(puls);
Serial.print("dip1 "); Serial.println(digitalRead(dip1));
if (puls==0)
{
    if (found==1){
        digitalWrite(led, LOW);
        found=0;
        if (digitalRead(dip1)==0){
            if (digitalRead(dip2)==0){ //se il dip2 è su on i relé tornano _
a riposo appena viene rilasciato il pulsante del telecomando
                rele=0;
            }
            else
            {
                delay(valtime);
                rele=0;
            }
        }
    }
}
else
{
    if (puls>=1 && puls<=8){
        digitalWrite(led, HIGH);

        Serial.print("Button ");Serial.println(puls);
        if (found==0){
            if (digitalRead(dip1)==0){ //se il dip1 è su off i relé _
rimangono attivati fino a quando il pulsante del telecomando è premuto
                bitSet(rele,puls-1);
            }
            else
            {
                if (bitRead(rele,puls-1)==0)
                {bitSet(rele,puls-1);}
                else
                {bitClear(rele,puls-1);}
            }
        }
        found=1;
    }
    else
    {
        rele=0;
    }
}

Serial.print("rele "); Serial.println(rele,BIN);
gestrele();

}

void gestrele(){
    for (int i=0; i<=7; i++)
    {
        int staterele=bitRead(rele, i);
        IOExpander.digitalWrite((15-i),staterele);
    }
}
```



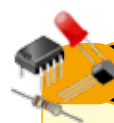
Il ricevitore può essere abbinato a TX palmari Aurel ad un canale (codice HCS-TX-1), a due canali (codice HCS-TX-2), a tre canali (codice HCS-TX-3), a quattro canali (codice TX-4M-HCS), a sei canali (codice TX-6M-HCS) o a dodici canali (codice TX-12CH); tutti si acquistano già montati presso Futura Elettronica (www.futurashop.it)

nosciuti; infatti l'apprendimento acquisisce il codice base, che è unico per il trasmettitore, mentre la parte che descrive i pulsanti è quella variabile. È possibile ripetere questa procedura per fare apprendere al ricevitore RX 4MHCS-4B fino ad un massimo 10 trasmettitori TX-12CH. Potete memorizzare anche trasmettitori ad uno, due, tre o sei canali, ma ricordate che il pulsante su ogni telecomando attiverà sempre lo stesso relè, ovvero non è possibile memorizzare otto trasmettitori monocanale e abbinarli a 8 uscite differenti.

Per rimuovere dalla memoria del ricevitore i trasmettitori abbinati, bisogna operare la cancellazione totale della memoria (non è possibile cancellare un solo codice); questa procedura si svolge premendo e rilasciando il pulsante P1, quindi, appena il LED LD2 inizia a lampeggiare, premendo nuovamente lo stesso P1 (ciò provoca l'accensione del led a luce fissa) e tenendolo premuto fin quando il led non si spegne. Rilasciato P1, LD1 deve emettere 5 lampeggi per indicare che la memoria dei

codici è stata cancellata. Questa operazione va eseguita dopo aver montato il ricevitore, prima di fare l'abbinamento col trasmettitore, in modo da rimuovere eventuali dati casuali che possono trovarsi nella EEPROM dell'ibrido. Se avete impostato per le uscite a relè la modalità monostabile con rilascio ritardato, dovete posizionare il cursore del trimmer posto sullo shield HCS in modo da ottenere il ritardo desiderato; allo scopo, ricordate che ruotando il cursore in senso orario (guardando il trimmer R1 di fronte) il tempo si accorcia mentre in senso antiorario aumenta. Ricordate altresì che l'impostazione della modalità e quella del tempo riguardano tutte le uscite (non è prevista l'impostazione differenziata relè per relè). Come già detto, sebbene lo shield I/O Expander disponga di 8 ingressi in questa applicazione ci interessano le sole uscite a relè; dato che Arduino non li considera perché non interroga lo shield per conoscerne lo stato, non dovete preoccuparvi di fare alcunché, quindi non serve cortocircuitarli. Il progetto presentato in queste

pagine è stato ideato supponendo di utilizzare un solo shield I/O Expander e pertanto di gestire al massimo 8 canali. Tuttavia, dal momento che il sistema supporta fino ad un massimo di 8 shield Expander uno sopra l'altro (ognuno va identificato da un indirizzo I²C univoco) in teoria si possono gestire fino a 64 uscite. Nella pratica, potendo l'ibrido Aurel ricevere i comandi di 12 canali, basta un secondo shield, del quale l'indirizzo dev'essere diverso da quello del primo. Per chi volesse effettuare delle modifiche alla firmware, riportiamo nel Listato 1 lo sketch completo, dove si vede l'include per la libreria wire, utilizzata per la gestione del bus I²C e quello per la libreria che gestisce lo shield I/O Expander. Volendo aggiungere un secondo shield occorre inserire nello sketch una seconda struttura e naturalmente mettendo tra parentesi in IOExpander.begin() l'indirizzo assegnato al secondo shield. ■

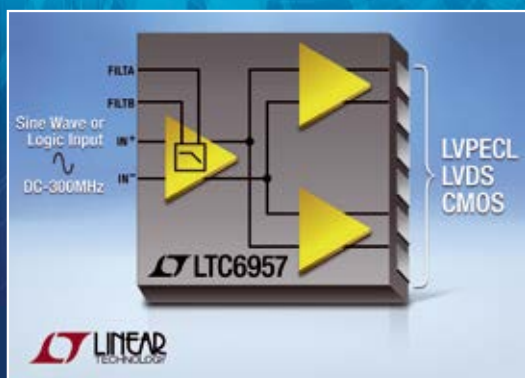


per il MATERIALE

Il materiale utilizzato in questo progetto può essere acquistato presso Futura Elettronica. La board Arduino Uno REV3 (Cod. ARDUINOUNO-REV3) costa 24,50 Euro, il trasmettitore palmare a 12 canali (Cod. TX-12CH) è disponibile a 52,00 Euro, lo shield Arduino I/O expander (Cod. FT1079K) a 34,00 Euro, mentre lo shield ricevitore HCS (Cod. FT1179K) è in vendita a 28,00 Euro.

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

PROGETTARE UN CIRCUITO D'INGRESSO RF



Progettare il circuito d'ingresso di riferimento di un sistema RF può essere difficile. Uno dei problemi a cui far fronte è mantenere le prestazioni del clock d'ingresso in relazione al rumore di fase, al tempo stesso rispondendo ai requisiti di protezione, buffer e distribuzione relativi al clock. Il presente articolo spiega come progettare un circuito d'ingresso di riferimento a 10MHz e ottimizzarne le prestazioni.

di MICHEL AZARIAN*

Gli strumenti RF e i transceiver wireless spesso presentano un ingresso per un clock di riferimento esterno, come la porta d'ingresso di riferimento a 10MHz di cui sono dotati tutti gli strumenti RF. Molti di questi stessi sistemi sono configurati in modo da distribuire il clock di riferimento nell'intero sistema. La Fig. 1 mostra uno schema comune, in cui il clock di riferimento applica l'ingresso di riferimento a due anelli ad aggancio di fase (PLL) distinti. Un ingresso robusto, ben progettato accetterebbe segnali sia sinusoidali che a onda quadra in un'ampia gamma di ampiezze, mantenendo inoltre un segnale di comando costante agli ingressi dei PLL finali all'interno del sistema, anche in presenza di ingressi vari. La porta d'ingresso di riferimento accessibile dall'esterno deve essere dotata di circuito di protezione contro sovratensioni/sovrappotenza. E soprattutto, occorre ridurre al minimo l'inevitabile peggioramento delle prestazioni in relazione al rumore di fase.

IMPLEMENTAZIONE DEL PROGETTO

L'LTC6957 è un dispositivo di traslazione di livelli logici e un buffer con clock a doppia uscita con rumore di fase (o jitter) aggiuntivo bassissimo. L'ingresso dell'LTC6957 accetta un'onda sinusoidale o quadra in un'ampia gamma di ampiezze e pilota carichi ad ampiezza costante. L'LTC6957 offre varie opzioni per il segnale logico di uscita: PECL, LVDS e CMOS (in fase e complementare), per cui può pilotare un'ampia gamma di carichi. La Fig.

2 mostra un circuito d'ingresso di riferimento a 10MHz che utilizza l'LTC6957-3, che produce due uscite CMOS in fase.

Il trasformatore WBC16-1T illustrato nella Fig. 2 svolge numerose funzioni. Anzitutto, congiuntamente ai diodi Schottky inseriti a valle del secondario, offre protezione contro sovratensioni/sovrappotenza all'ingresso. I diodi limitano la tensione CA applicata all'LTC6957-3. Il trasformatore può gestire fino a 0,25W di potenza (3,5VRMS su 50Ω); inoltre isola la massa del connettore, che in genere è collegata allo chassis del sistema RF, dalla massa analogica interna del sistema.

Oltre a ciò, il trasformatore applica un guadagno di tensione al segnale d'ingresso, rendendo più ripidi i fronti dei segnali applicati all'LTC6957-3; ciò contribuisce a ridurre la conversione del rumore da AM a PM, il che a sua volta limita il peggioramento del rumore di fase, specialmente con piccoli segnali d'ingresso. Il WBC16 1T presenta un guadagno di tensione pari a 4 ed è possibile utilizzare tale guadagno, anziché dipendere dal guadagno di potenza ideale e massimo pari a 1, poiché l'LTC6957-3 presenta un carico ad alta impedenza al trasformatore.

R1 e R2 possono essere regolate in combinazione per adattare la porta d'ingresso a 50Ω. Per piccoli segnali d'ingresso, i diodi sono in interdizione e il carico del trasformatore nella Fig. 2 è pari a 804Ω. Il carico viene riportato all'ingresso a un valore di circa 50Ω a causa del

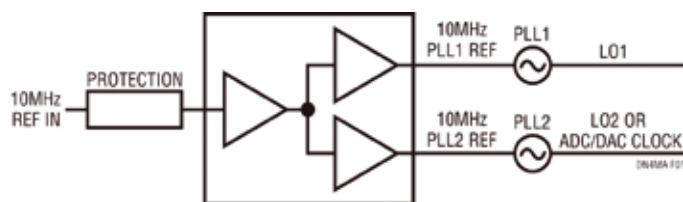
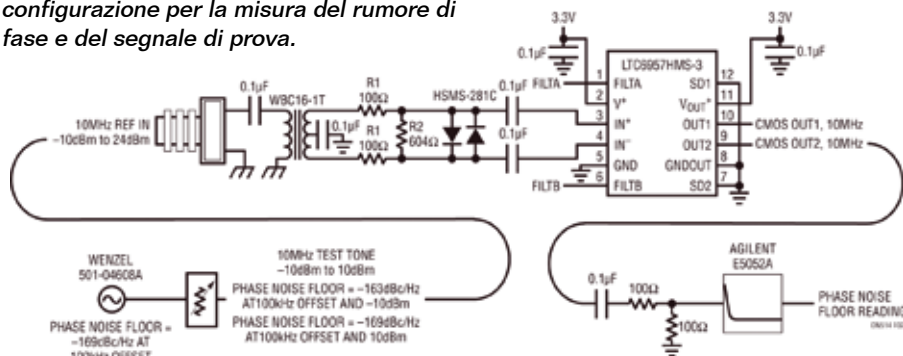


Fig. 1 - Diagramma a blocchi di un circuito di distribuzione di riferimento e ingresso di riferimento comune per un sistema RF.

rapporto d'impedenza fra primario e secondario del trasformatore, pari a 16. Per segnali d'ingresso maggiori, i diodi Schottky vanno in conduzione, riducendo la resistenza di 604Ω quasi a un cortocircuito. Questo peggiora l'attenuazione di ritorno all'ingresso di riferimento, un problema che può essere evitato regolando i valori di R1 e R2, ma ne derivano compromessi. Per elevati segnali d'ingresso, l'attenuazione di ritorno all'ingresso può essere migliorata aumentando R1 e riducendo R2, in modo tale che la resistenza in serie di R1 e R2 rimanga uguale a circa 800Ω. Tuttavia, poiché R1 è in serie con il segnale, vi aggiunge rumore. Una R1 maggiore è in combinazione con una R2 minore, così che all'ingresso dell'LTC6957-3 viene applicata una parte minore dell'ingresso, peggiorando ulteriormente le prestazioni in relazione al rumore di fase. Ossia, il progettista può attuare un compromesso tra le prestazioni in relazione al rumore di fase e l'attenuazione di ritorno all'ingresso regolando R1 e R2. I valori illustrati nella Figura 2 consentono di ottenere un equilibrio complessivo fra questi due parametri prestazionali.

Fig. 2 - Circuito d'ingresso di riferimento a 10MHz che impiega l'LTC6957-3 con protezione front-end, mostrato nella configurazione per la misura del rumore di fase e del segnale di prova.



Il condensatore di accoppiamento CA che nella Fig. 2 separa il connettore dal trasformatore offre protezione dell'ingresso dai generatori CC. L'LTC6957-3 è dotato di filtri passa basso interni selezionabili mediante i pin FILTA e FILTB. Questa opzione limita strategicamente la larghezza di banda del primo stadio amplificatore dell'LTC6957 e quindi il rumore di fase aggiuntivo del circuito, specialmente quando il segnale d'ingresso è debole, come illustrato di seguito.

PRESTAZIONI

Un oscillatore al quarzo termostato (OCXO) da 10MHz è collegato all'ingresso del circuito attraverso un attenuatore regolabile, come illustrato nella Fig. 2. Il segnale d'ingresso di riferimento viene fatto variare tra -10dBm e 10dBm mentre si misura il livello base del rumore di fase all'uscita dell'LTC6957-3 con differenti impostazioni dei filtri d'ingresso impiegando l'analizzatore di generatore di segnale Agilent E5052A. La Fig. 3 mostra il livello base del rumore di fase dell'uscita di clock del CMOS a 10MHz dell'LTC6957-3 misurato a un offset di 100kHz. Se l'ampiezza del segnale di riferimento a 10MHz applicato esternamente è ignota, portando il pin FILTA al livello basso e il pin

FILTB al livello alto si ottengono buone prestazioni complessive in relazione al rumore di fase, come illustrato nella Fig. 3. Ciò nonostante, è possibile ottimizzare le prestazioni se si misura il livello del segnale applicato all'ingresso e si applicano le appropriate impostazioni per i filtri. I valori R1 e R2 scelti nella Fig. 2 causano un'attenuazione di ritorno all'ingresso di -9dB quando la potenza dell'ingresso di riferimento è uguale a 0dBm su 50Ω. L'attenuazione di ritorno all'ingresso è migliore a potenze d'ingresso inferiori e

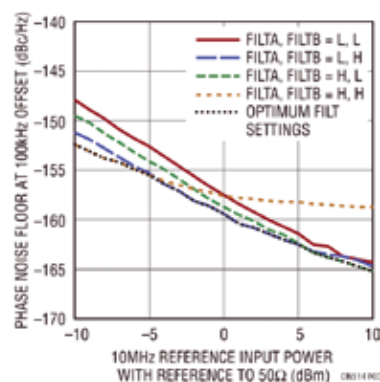


Fig. 3 - Livello base del rumore di fase con offset di 100kHz all'uscita dell'LTC6957-3 in funzione del livello di potenza d'ingresso di riferimento a 10MHz per varie impostazioni dei filtri dell'LTC6957.

peggiore a potenze superiori. L'LTC6957-3 semplifica le fasi di progettazione e consente di ottenere prestazioni complessive eccellenti. Le caratteristiche sono un'ampia gamma di tipologie di segnale d'ingresso e compatibilità di livelli, distribuzione del clock e protezione con limitato peggioramento del rumore di fase. L'attenuazione di ritorno all'ingresso e il rumore di fase del circuito sono stati valutati e ottimizzati. Tutto ciò consente di costruire un robusto circuito d'ingresso di riferimento a 10MHz con prestazioni di tutto rispetto.

* Michel Azarian è Senior Applications Engineer presso Linear Technology.



Telecomando universale a infrarossi gestibile anche da Internet, basato su RandA dotata di un apposito shield e su Raspberry Pi2.

di OSVALDO SANDOLETTI

ARDUINO + INFRAROSSI = ARDIR

Nonostante la rivoluzione “smart” della domotica, intesa come automazione e automatizzazione completa ed integrata dei dispositivi presenti nelle nostre abitazioni, venga ancora sfruttata da pochi, va detto che le applicazioni in tal senso crescono giorno dopo giorno ed offrono soluzioni a molti problemi grandi e piccoli della vita quotidiana. Il progetto **ArdIR** presentato in queste pagine è un’applicazione domotica molto particolare, perché simula il telecomando di TV, elettrodomestici e condizionatori d’aria, trasmettendo gli stessi dati all’apparecchio desiderato, ma è interamente programmabile e gestibile da remoto, essendo “visibile

in rete” come un Web Server che espone le proprie pagine cui connettersi per impartire i comandi. Con queste premesse, appare chiaro che il suo utilizzo di “elezione” sia il controllo del climatizzatore domestico: chi si trova fuori casa può collegarsi via Internet per accenderlo e dargli tempo di portare la temperatura al livello desiderato prima di rientrare. Ma anche stando in casa, ci può essere la necessità di controllare un apparecchio, come il televisore o lo stereo, da stanze diverse senza doversi spostare. Infine, ArdIR può essere utilizzato come semplice sostituto del telecomando, nel caso quest’ultimo andasse fuori uso o per superarne i limiti di copertura.

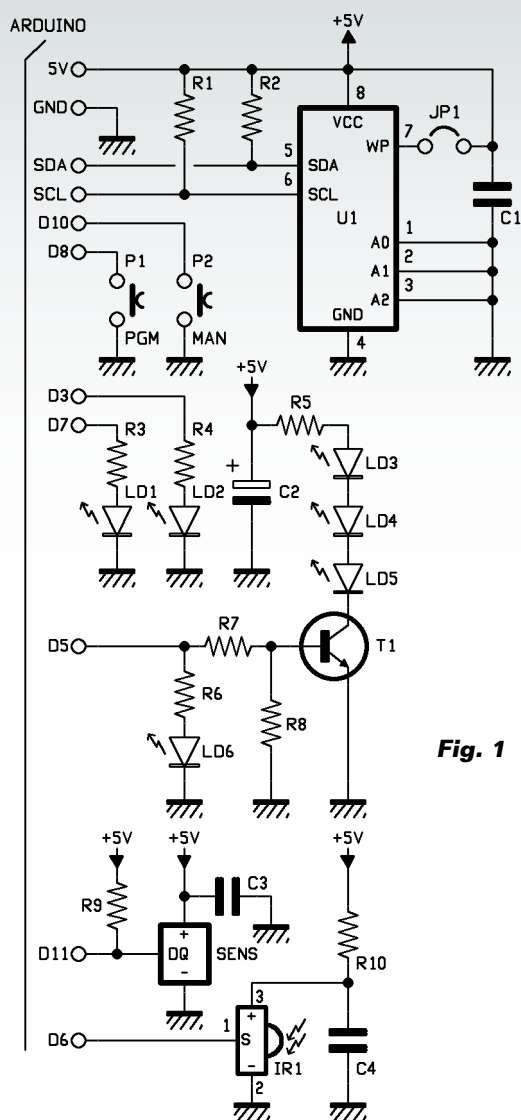


Fig. 1

PRINCIPIO DI FUNZIONAMENTO

Il funzionamento alla base di ArdIR è quello di un normale telecomando "universale": inizialmente necessita una fase di apprendimento per memorizzare i codici del (dei) telecomando (i) e in seguito, quando richiesto, tali codici vengono trasmessi all'apparecchio da gestire. Il *plus* sta nel fatto che ArdIR non si pilota tramite una tastiera fisica, ma con quella "virtuale" presentata nella pagina HTML/JS esposta dal suo Web Server con protocollo HTTP, quindi visibile

collegandosi da smartphone o PC tramite un browser.

A livello pratico, il sistema hardware di principio potrebbe semplicemente essere composto da Arduino collegato ad un apposito shield dotato di ricevitore (per la fase di apprendimento) e LED emittenti nell'infrarosso. Per aggiungere la connettività alla rete, però, Arduino non basta: la nostra scelta è pertanto ricaduta sull'utilizzo della board **RandA**, presentata pochi numeri fa su questa stessa rivista, che alla semplicità di Arduino aggiunge la potenza di calcolo e la quantità di memoria offerte da Raspberry Pi, che qui sono necessarie per gestire le funzionalità del Web Server. Rispetto all'utilizzo di Raspberry Pi e Arduino come schede "separate", RandA aggiunge diversi vantaggi:

- l'installazione del software della Raspberry Pi include diverse librerie per la comunicazione con Arduino per facilitare l'integrazione tra i due sistemi; ad esempio, Arduino può direttamente trasmettere comandi Linux alla Raspberry Pi e scrivervi/leggerli file;
- la programmazione di Arduino può avvenire da Raspberry Pi, il cui software a corredo comprende anche l'IDE; ma è disponibile anche un IDE per PC "modificato" in grado di connettersi ad Arduino da remoto, nel momento in cui la Raspberry Pi sia collegata ad una rete;
- nel pacchetto software viene già incluso, configurato e funzionante, il server web con a corredo diversi applicativi e delle pagine web di esempio utilizzabili come base per sviluppare le vostre applicazioni;
- è stato aggiunto un orologio in tempo reale (RTC) completo di batteria tampone, utile se si

vuole -ad esempio- programmare delle attivazioni degli apparecchi basate sull'orario;

- dal punto di vista hardware, il sistema complessivo risulta più compatto: può venire alimentato da un'unica sorgente e non vi sono cablaggi "volanti" di collegamento tra una scheda e l'altra.

Con questa "dotazione", l'unica scheda hardware ancora da sviluppare è lo shield con le interfacce necessarie alla comunicazione all'infrarosso, che adesso descriviamo nel dettaglio. Per tutti i dettagli del funzionamento e per conoscere le potenzialità del sistema RandA, consigliamo di consultare i relativi articoli apparsi sui numeri 190, 191 e 192 di Elettronica In.

SCHEMA ELETRICO DELLO SHIELD

In Fig. 1 è mostrato lo schema elettrico dello shield ArdIR, che risulta decisamente semplice, sia concettualmente che come numero di componenti necessari. Partendo dal basso, notiamo il ricevitore ad infrarosso IR1, la cui uscita viene letta direttamente dalla linea D6 di Arduino/RandA, durante la fase di apprendimento dal telecomando "originale". Su D11 è collegato il sensore di temperatura, l'arcinoto DS18B20, per restituire all'utente la temperatura ambientale. Salendo ancora si arriva al transistor bipolare T1, utilizzato come interruttore elettronico controllato dalla linea D5 (configurata come uscita), la quale viene programmata dallo sketch per applicare il segnale, modulato, relativo al canale che vogliamo trasmettere. La scelta di D5 non è casuale, in quanto corrisponde all'uscita del timer TCCR0 utilizzato per ricavare la frequenza

della modulazione.

I diodi all'infrarosso LD3÷LD5, pilotati con questo segnale, trasmettono i dati sotto forma di impulsi di luce all'infrarosso: con l'alimentazione a 5V ed un resistore serie (R5) da 22Ω, la corrente diretta nella serie di diodi si attesta sui 50mA, valore che permette di trasmettere a una distanza di almeno due-tre metri (l'esatta portata dipende anche dall'allineamento tra i fotodiodi ed il ricevitore, e dalla sensibilità di quest'ultimo). Per aumentare ulteriormente la portata bisogna incrementare la corrente, il che si può fare (abbassando il valore di resistenza e/o aumentando la tensione) a patto di ridurre il *duty-cycle* del segnale applicato a T1 (diminuendo il rapporto tra tempo di accensione e di spegnimento) per non danneggiare i diodi stessi. Sulla base di T1 è collegato anche un LED giallo (LD6), utilizzato per monitorare l'esecuzione dei cicli di trasmissione (lampeggerà velocemente durante la trasmissione).

I LED LD1 e LD2 servono a visualizzare lo stato del sistema durante l'esecuzione delle varie fasi, come spiegato più avanti, mentre P1 e P2 servono a impartire i comandi, soprattutto relativi alla fase di acquisizione, che saranno pure dettagliati in seguito.

Infine, U1 è una memoria EEPROM gestita tramite bus I²C, necessaria a memorizzare i codici dei canali acquisiti: utilizzando una memoria tipo 24LC256 (32 kB) si possono memorizzare sino a 127 codici differenti. Abbiamo dovuto ricorrere a una memoria esterna perché la EEPROM interna al microcontrollore di Arduino ha una capacità limitata (1 kB) che permette di memorizzare non oltre quattro canali. Passiamo adesso dall'hardware

al software, andando a vedere come viene gestito lo shield appena descritto.

LO SKETCH PER ARDUINO

Il programma per l'Arduino residente in RandA è stato sviluppato come una macchina a stati, il cui diagramma è riassunto in Fig. 2. Dallo stato di attesa, con entrambi i LED spenti, si rimane in attesa che venga premuto un pulsante oppure che arrivi un comando da remoto (da pagina web, che -come vedremo più avanti- dal punto di vista di Arduino corrisponde alla ricezione di alcuni caratteri sulla seriale). Premendo P1 si entra in modalità di acquisizione da telecomando: il LED rosso lampeggia, segnalando che sulla tastiera virtuale della pagina web l'utente deve cliccare sul tasto che desidera associare al codice del telecomando che sarà acquisito. Se l'operazione non viene compiuta

entro dieci secondi la fase viene abortita ed il sistema torna nello stato di attesa. Viceversa, il LED si accende fisso a segnalare che bisogna attivare il tastino desiderato sul telecomando (dopo averlo "puntato" verso il ricevitore a infrarossi IR1 a circa 10 cm di distanza) per trasmettere il codice da acquisire. Ciò fatto, dovrebbe accendersi il LED verde per cinque secondi a segnalare il corretto apprendimento del nuovo codice. Anche in questo caso, è presente un *time-out*: se entro breve tempo il segnale infrarosso non arriva, il LED rosso lampeggia tre volte a segnalare l'insuccesso dell'operazione, quindi si torna nello stato di attesa. Notare che la fase di acquisizione si può attivare solo premendo il tastino P1 e non dalla pagina web, e questo è stato voluto principalmente per evitare che attivandola erroneamente da remoto venga sovrascritto qual-

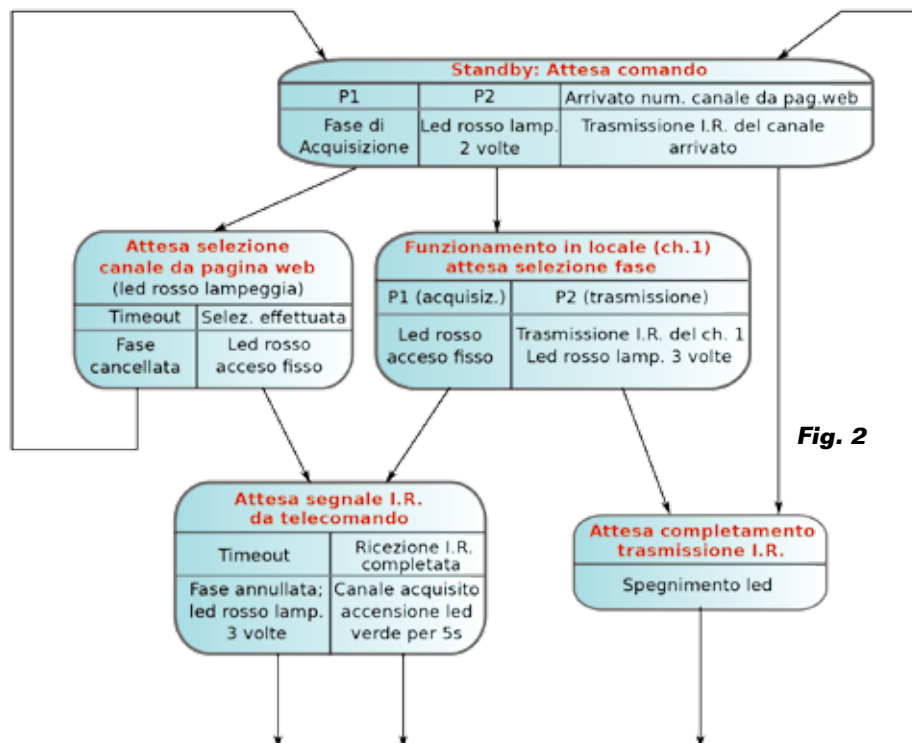


Fig. 2

che canale. Comunque, il dover premere il pulsante non costituisce un problema perché bisogna essere vicini alla scheda per poter “puntarvi contro” il telecomando da cui apprendere i codici.

Tornando al diagramma, se dallo stato di attesa si preme P2, il LED rosso lampeggia due volte ad indicare che si è attivata la *modalità locale*, nel senso che le operazioni di acquisizione e trasmissione si possono compiere agendo sui due tastini, quindi senza doversi collegare “da remoto” tramite pagina web. Questa opzione permette di utilizzare il sistema in modalità “stand-alone”, ad esempio per fare un test rapido sul funzionamento anche quando si è lontani da un PC.

L'unica limitazione è che non si ha a disposizione la tastiera virtuale, e dunque si è scelto di fare apprendere e/o trasmettere solo il primo canale. Il funzionamento è evidente dal diagramma: entrati nel funzionamento “in locale”, la pressione di P1 inserisce la fase di acquisizione, con il LED rosso acceso fisso in attesa di ricevere un codice dal telecomando (che poi verrà salvato, come già detto, nel canale nume-

ro uno). Premendo invece P2, il LED rosso lampeggia tre volte e viene poi attivata la trasmissione IR.

Infine, se dallo stato di attesa si riceve un comando da remoto, si attiva la trasmissione IR del canale il cui numero è specificato dal comando stesso (che evidentemente corrisponde al relativo pulsante cliccato sulla pagina web). È poi stato aggiunto un comando, identificato da (**numero di canali + 1**), che richiede ad Arduino di leggere la temperatura dal sensore DS18B20 e poi ritrasmetterla alla pagina web.

LETTURA DEI DATI DEL TELECOMANDO

I normali telecomandi a infrarossi (di televisori, decoder, videoregistratori, ecc.) lavorano associando ciascun tasto pigiato ad un codice binario che viene poi trasmesso serialmente (un bit per volta) utilizzando diverse codifiche (OOK, FSK,...) e modulando la luce emessa dal trasmettitore infrarosso con una data *frequenza portante* (tipicamente 36÷38 kHz) per minimizzare la possibilità di essere disturbati da sorgenti di luce ambientali che essendo “rumore” hanno invece componenti

continue o a frequenze “casuali”. La modulazione di gran lunga più utilizzata in questi casi è sicuramente la OOK (*On-Off Keying*, di cui si trova numerosa documentazione in Rete). Per ciascun bit da trasmettere, il modulatore/trasmettitore controlla il suo valore: se è 1 viene trasmesso un treno di impulsi all'infrarosso alla frequenza della portante; se è 0, invece, il trasmettitore rimane spento (o viceversa).

Il ricevitore, dal canto suo, è “sintonizzato” per riconoscere questo treno di impulsi solo se ha una frequenza corrispondente a quella della portante, e trasmettere in uscita il valore ‘1’ (o ‘0’) se li rileva, e ‘0’ (oppure ‘1’) altrimenti: in pratica, all'uscita del ricevitore si ritrova il segnale di partenza, quindi “ripulito” della portante (pertanto detto *segnale demodulato*).

Nel nostro shield questo segnale demodulato lo ritroviamo sull'uscita di IR1, che è un ricevitore progettato per lavorare con segnali modulati a 38 kHz. In Fig. 3 è mostrato l'oscillogramma del segnale rilevato (sull'uscita di IR1) da un telecomando per televisore, ed una “zoommata” per rilevare la durata dei singoli bit.

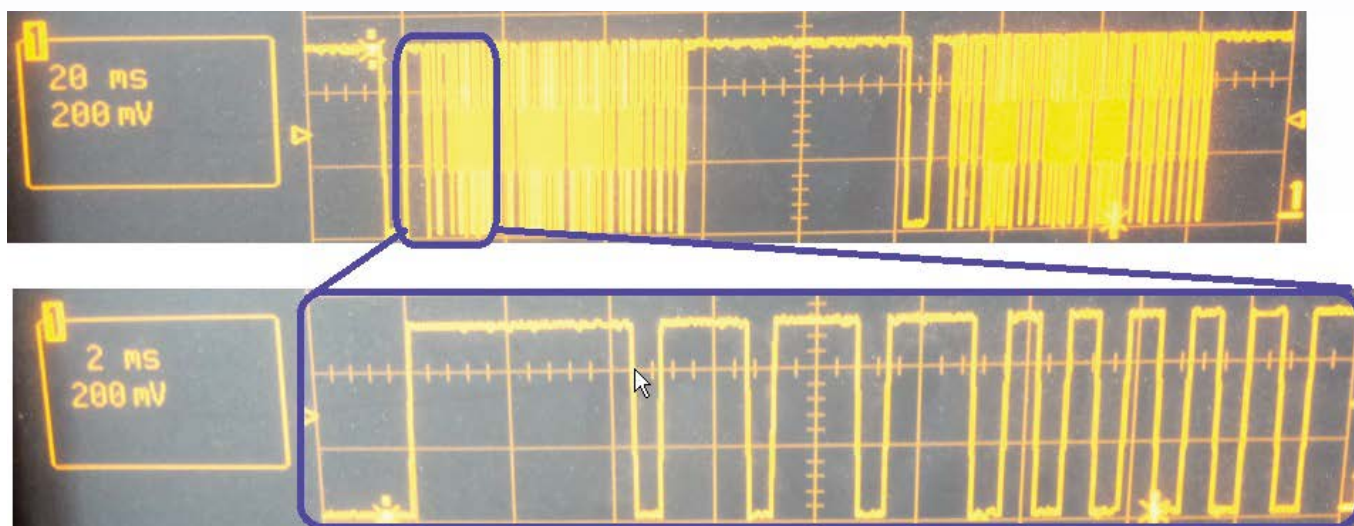
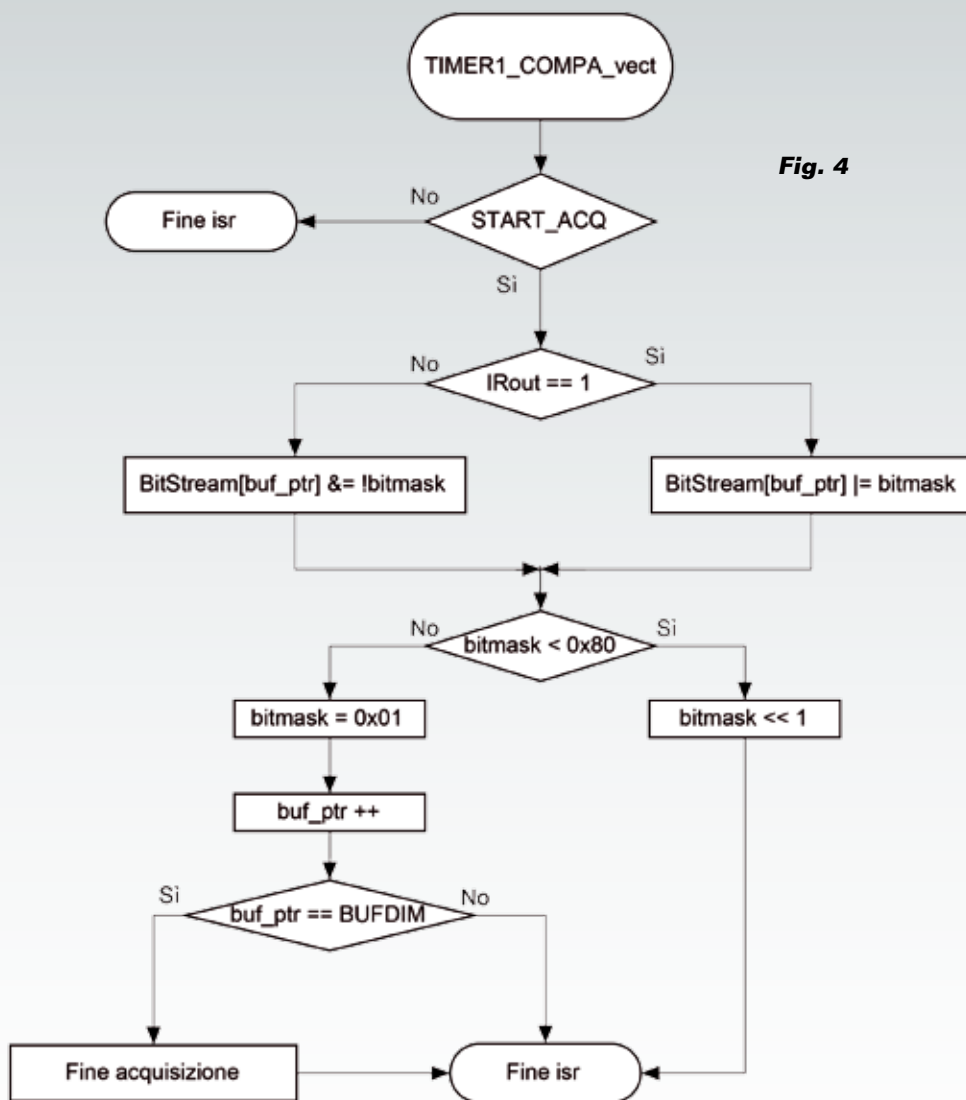


Fig. 3

A livello software, per poter acquisire questi segnali lo sketch contiene, oltre all'implementazione della macchina a stati appena descritta, una sezione in "real time", ovvero, che deve essere eseguita ad intervalli di tempo molto stretti e regolari, per garantire che il segnale possa venire campionato e riprodotto correttamente e senza perdere informazioni. Per tale motivo si è racchiuso tale codice in una *Interrupt Service Routine* chiamata dal Timer1 del microcontrollore, il cui diagramma semplificato e ridotto alla parte di acquisizione è visibile in Fig. 4.

L'intervallo di tempo è stato scelto sufficientemente breve da poter campionare qualsiasi segnale possa arrivare dal telecomando, senza "perdere" dei bit di informazione; la teoria insegna che la frequenza (minima) di campionamento dev'essere pari al doppio del segnale in ingresso (nel caso di un segnale digitale questo vale però se il *duty-cycle* rimane intorno al 50%). Basandosi sulle forme d'onda della Fig. 3, si è deciso di adottare un periodo di campionamento di 100 µs, decisamente inferiore al periodo dei dati analizzati. Durante la fase di apprendimento (flag 'START_ACQ' attivo), ad ogni campionamento si copia lo stato della linea d'ingresso facente capo a IR1 e la si memorizza in un bit; ogni otto campionamenti si forma quindi un nuovo byte di dato del vettore BitStream[], che è 'lungo' BUFDIM (costante dichiarata nel programma) byte. Pertanto, il numero di bit che possiamo memorizzare è pari a BUFDIM * 8 e la durata del segnale in ingresso può arrivare a 100µs * BUFDIM * 8; attualmente abbiamo dichiarata un vettore di 256 (BUFDIM) elementi, quindi il segnale utile del telecomando può durare fino



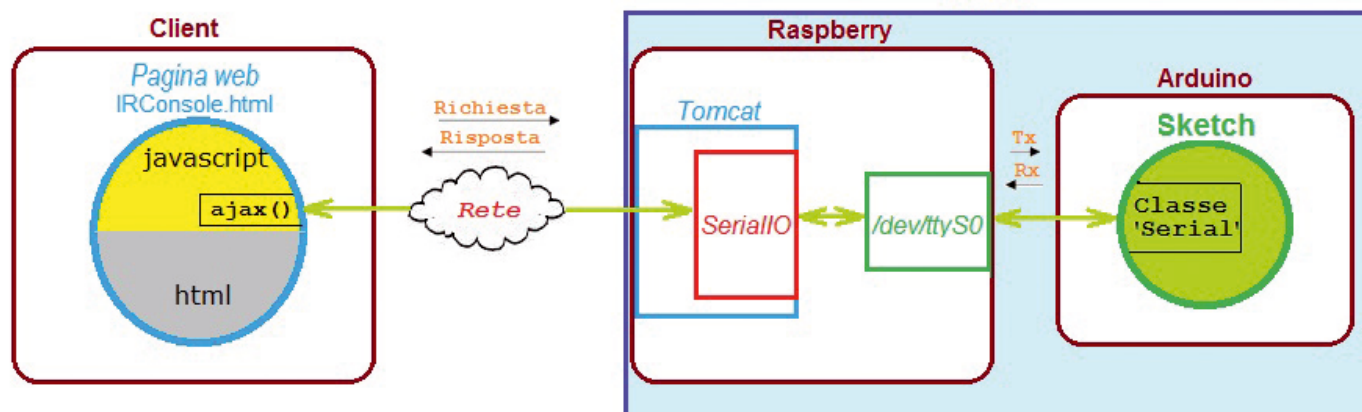
a 200ms, più che sufficienti dato che bastano poche decine di ms per trasmettere un comando. Il codice del telecomando illustrato in Fig. 3, ad esempio, viene trasmesso in soli 60ms, e poi successivamente ripetuto (finché si tiene premuto il tasto). In fase di trasmissione invece (flag 'SEND_IR' attivo) il vettore BitStream[] viene letto bit per bit, nella stessa sequenza con cui era stato scritto ed alla stessa cadenza (100µs) per cui il 'flusso di bit' riprodotto risulterà necessariamente identico a quello precedentemente letto. Questi bit servono per accendere-spegnere

l'uscita PWM del timer T0 (D5 di Arduino) che è programmato per riprodurre un'onda quadra a 38 kHz e dunque ci ritroveremo la sequenza di bit modulata a tale frequenza, atta a pilotare tramite T1 i diodi all'infrarosso dello shield.

INTERFACCIA WEB

Per comunicare con lo sketch dall'esterno del "regno" di Arduino, come detto, abbiamo sfruttato l'"altra metà" di RandA, ovvero la Raspberry Pi e gli strumenti software che il mondo Linux mette a disposizione, in particolare il programma Apache

RandA

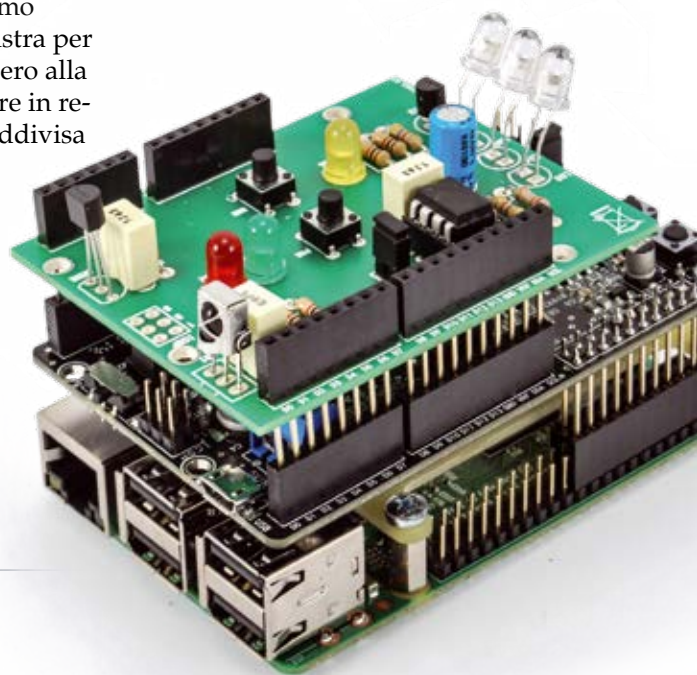


naturalmente editare il sorgente della servlet (`SerialIO.class`) che si trova nella cartella `/home/apache-tomcat-7.0.47/webapps/RandA/WEB-INF/classes/ArduIO`. Tenete solo presente che, se volesse modificarla a piacimento, bisogna rifarne il *deploy* su Tomcat (al proposito rimandiamo alla consultazione della guida su <https://tomcat.apache.org/tomcat-7.0-doc/appdev/deployment.html>).

In **Fig. 5** vediamo la schematizzazione complessiva del flusso dei dati e degli attori che ne prendono parte, dalla pagina web al micro di Arduino, in cui si nota la funzione di interfaccia della servlet `SerialIO` tra il server web e la porta seriale (descritta da `/dev/ttyS0`) della Raspberry Pi. Nella stessa figura possiamo adesso spostarci sulla sinistra per arrivare al lato client, ovvero alla pagina web da visualizzare in remoto. Questa pagina è suddivisa in due parti: una globale in html, utilizzata più che altro per costruire la presentazione (ovvero la grafica), ed una sezione di codice *javascript* che si occupa della gestione degli eventi (in pratica i "clic" sui tasti) attivati dall'utente.

Il meccanismo utilizzato è molto semplice: ad

ogni tasto dell'interfaccia grafica si associa una funzione javascript che sarà invocata ogni volta che il tasto viene attivato. Questa funzione può effettuare elaborazioni in locale (sulla macchina *client*) oppure, nel nostro caso in cui vogliamo trasmettere i comandi alla scheda RandA, chiamare a sua volta la funzione `ajax()` che, con opportuni parametri, instaura un colloquio col server, o meglio con la *servlet* SerialIO. Più specificamente il client invia una richiesta al server, che come visto attiverà una funzione sulla servlet che a sua volta trasformerà in una comunicazione seriale ad Arduino; la risposta di quest'ultimo, seguendo il percorso inverso, viene



poi intercettata e restituita dalla funzione `ajax()` stessa (sotto forma di stringa di caratteri), per essere eventualmente visualizzata e dare all'utente il risultato richiesto o almeno un responso sull'esito dell'operazione.

PREPARAZIONE DEL SISTEMA

Passando al lato pratico, la prima cosa da fare è realizzare lo shield ArdIR, saldando sull'apposito circuito stampato i pochi componenti necessari; non vi sono criticità particolari (tutti i componenti, di facile reperibilità, sono in formato "tradizionale") quindi le raccomandazioni rimangono sempre le solite riguardanti il montaggio dei componenti elettronici (soprattutto fate attenzione alla polarità!) e non staremo qui a ripeterle ulteriormente. L'unico suggerimento è quello di non saldare a filo scheda i diodi all'infrarosso LD3, LD4, LD5 ma di lasciarne i reofori lunghi almeno 1-2 cm in modo da poterne ottimizzare l'orientamento verso l'apparecchio da gestire. Per quanto riguarda la programmazione dello sketch ArdIR sul micro di Arduino, dopo avere alimentato e collegato RandA alla rete locale in modo da potervi accedere tramite TCP/IP, possiamo scegliere di procedere in due modi:

- installare su PC la IDE di Arduino modificata (inclusa nella distribuzione di RandA), che permette di "vedere" la scheda Arduino di RandA al relativo indirizzo IP (verificare che sia selezionato sul menu Strumenti → Porta seriale); in seguito è sufficiente caricare lo sketch e programmare Arduino come di consueto;
- utilizzare l'IDE di Arduino per Raspberry Pi: collegarsi a RandA tramite `ssh` (ad esempio utilizzando MobaXterm) e

Il *Linux Infrared Remote Control* (**LIRC**) è un pacchetto di programmi software open source (che naturalmente "gira" su piattaforma Linux, ma esiste anche **WinLIRC**, la versione per Windows) sviluppato per leggere i segnali in arrivo da un telecomando a infrarossi e trasformarli in azioni da eseguire sul PC su cui è installato come, ad esempio, lanciare applicazioni, muovere il puntatore del mouse, controllare il volume della scheda audio, ecc.

Chi utilizza il PC come sistema multimediale, troverà comodo poterlo collegare allo schermo televisivo e controllarlo a distanza tramite telecomando.

A livello hardware, l'unico problema è sempre e solo quello di permettere al PC di acquisire il segnale a infrarosso: LIRC permette di leggerlo dalle porte seriale, parallela e persino dall'ingresso del microfono (audio); esternamente bisogna dunque solo procurarsi (oppure costruirsi) una piccola scheda di interfaccia, che normalmente utilizza un ricevitore a infrarosso del tipo utilizzato nel nostro progetto (IR38DM oppure la serie TSOP17xx), la cui uscita si interfaccerà ad una delle porte del PC summenzionate. Naturalmente la portante del ricevitore dev'essere la stessa del telecomando che volete utilizzare; come già detto, comunque, le frequenze più utilizzate sono 36 e 38 kHz.

A livello software, una volta scaricato ed installato il pacchetto, bisognerà attivare il modulo del dispositivo d'ingresso utilizzato (es. `modprobe lirc_serial` per la porta seriale), ed attivare il demone `lircd` che, in background, si occupa di decodificare i segnali ricevuti dal telecomando. Questo processo trasmetterà quindi i codici dei tasti pigiati sul telecomando, tramite un apposito socket, agli applicativi (*LIRC client* oppure programmi che hanno già il supporto a LIRC come il *media player* VLC). Nel caso dei client LIRC, si può anche editare un file di configurazione (`lircrc`) per associare una funzione specifica a piacere a ciascun tasto premuto.

LIRC supporta una lista di telecomandi a infrarossi ben nutrita, è comunque possibile attivare una procedura di acquisizione per farlo funzionare anche con quelli non contemplati.

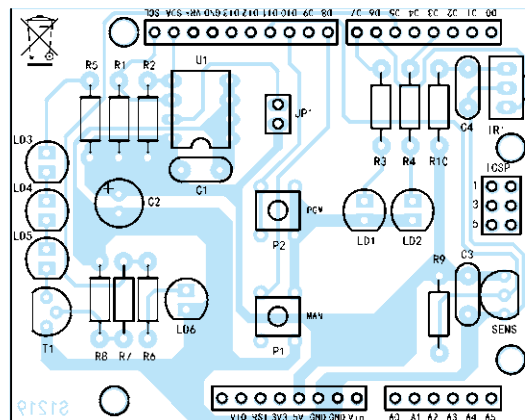
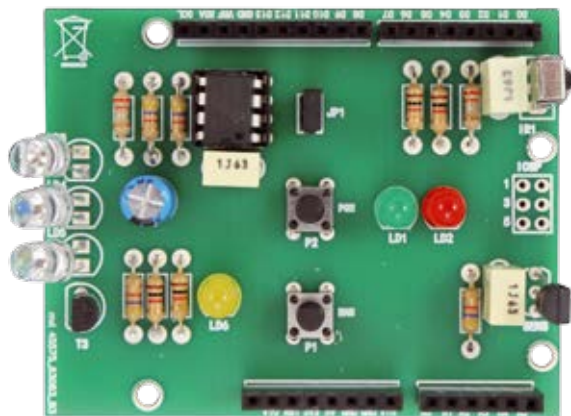
Per chi volesse approfondire l'argomento su LIRC, il sito di partenza da consultare è ovviamente quello ufficiale del progetto (<http://www.lirc.org/>), da cui si possono scaricare anche gli schematici per costruirsi l'hardware dell'interfaccia. su <http://lnx.manoweb.com/lirc/?partType=section&partName=lirc> sono descritti i passi per installare ed utilizzare rapidamente il software, tralasciando dettagli che si trovano invece sul sito del progetto.

Infine, una panoramica più ampia sui progetti che utilizzano Linux nell'ambito della comunicazione a infrarossi (soprattutto per quanto riguarda il protocollo IrDA) si può consultare su <http://www.tldp.org/HOWTO/text/Infrared-HOWTO>

trasferirvi lo sketch sul filesystem (ad esempio nella cartella `home/pi/sketchbook/`); in seguito lo si può aprire e compilare tramite la IDE di Arduino "locale".

In quest'ultimo caso, Arduino viene "visto" sulla porta seriale / `dev/ttyS0` (l'unica disponibile). In tutti i casi, se non l'avete mai fatto dovrete anche scaricare la libreria OneWire (necessaria per la gestione del sensore di temperatura DS18B20) e copiarla sotto la cartella `libraries` d'installazione dell'IDE (se in Windows) oppure in `/usr/share/arduino/`

`libraries` (se su Raspberry Pi). Le altre librerie del progetto sono invece già comprese con l'IDE per entrambe le piattaforme. Notare che l'IDE di Arduino di Raspberry Pi dovrebbe essere incluso se vi procurate una SD-card con il sistema RandA già installato; altrimenti, accertatevi che la scheda sia collegata in Internet durante l'installazione del pacchetto RandA dato che gli strumenti di Arduino vengono scaricati on-line. Per verificarlo, potete digitare su una console della Raspberry Pi il comando: `ping 8.8.8.8` ove 8.8.8.8 è l'indirizzo IPv4 del



Elenco Componenti:

R1: 4,7 kohm
 R2: 4,7 kohm
 R3: 1 kohm
 R4: 1 kohm
 R5: 22 ohm
 R6: 1 kohm
 R7: 1 kohm
 R8: 4,7 kohm
 R9: 4,7 kohm
 R10: 22 ohm
 C1: 1 μ F 63 VL poliestere
 C2: 47 μ F 16 VL elettrolitico

C3: 1 μ F 63 VL poliestere
 C4: 1 μ F 63 VL poliestere
 LD1: LED verde 5 mm
 LD2: LED rosso 5 mm
 LD3: LED IR 5 mm
 LD4: LED IR 5 mm
 LD5: LED IR 5 mm
 LD6: LED giallo 5 mm
 IR1: IR38DM
 SENS: DS18B20+
 T1: BC337
 U1: 24LC256

P1: Microswitch
 P2: Microswitch

Varie:

- Strip M/F 6 vie
- Strip M/F 8 vie (2 pz.)
- Strip M/F 10 vie
- Strip M/F 2x3 vie
- Strip maschio 2 vie
- Jumper
- Zoccolo 4+4
- Circuito stampato S1219

server DNS primario di Google. Se l'esito è positivo, la scheda Raspberry Pi "vede" Internet. Da alcune prove eseguite, consigliamo di utilizzare il primo sistema: la IDE su Raspberry è decisamente più lenta, soprattutto nella compilazione, ma d'altra parte la potenza di calcolo non può essere paragonata a quella di un PC!

Programmato lo sketch Ar-dIR ed inserito il suo shield su RandA, si può effettuare subito un primo collaudo del sistema. Innanzitutto si può verificare che, dopo ogni reset (oppure ogni programmazione) il LED rosso lampeggia tre volte, ad indicare che lo sketch è caricato e pronto all'uso. Quindi pigiare P2 sino a veder lampeggiare due volte il

LED rosso per entrare nella modalità locale di funzionamento (come descritto nello schema di Fig. 2). Successivamente pigiamo P1 per entrare in modalità di acquisizione, ed il LED rosso si accenderà fisso: puntiamo quindi il telecomando verso il ricevitore IR1 ad una distanza di 5-10cm e ne azioniamo il comando che desideriamo sia appreso dalla nostra scheda (esempio, cambio canale del televisore). Il LED rosso dovrebbe spegnersi, ed accendersi invece quello verde a conferma dell'avvenuta operazione. In caso contrario, ripetere l'operazione. Se nuovamente non andasse a buon fine bisogna verificare che il telecomando funzioni e che la frequenza della portante sia di 36-38

kHz; nel dubbio potete provare con un altro telecomando. Ammesso che l'operazione sia andata a buon fine, dopo cinque secondi il LED verde si spegnerà e torniamo nello stato di attesa: proviamo quindi a trasmettere il comando appena acquisito verso l'apparecchio relativo al telecomando che avevamo utilizzato, avendo cura che i LED all'infrarosso LD3-5 siano rivolti verso l'apparecchio ad una distanza possibilmente non superiore al metro (almeno per la prima prova). Pigiamo quindi P2 sino a vedere i due lampeggi del LED rosso, e quindi pigiamo nuovamente P2: il LED rosso produrrà ulteriori tre lampeggi al termine dei quali verrà effettuata la trasmissione verso l'apparec-

chio. Notare che contestualmente il LED blu dovrà accendersi brevemente. Se l'apparecchio "di prova" ha eseguito il comando che avevamo fatto acquisire, il collaudo può ritenersi concluso.

SPOSTIAMO IL CONTROLLO SUL WEB

L'ultimo passo da implementare è quello di dare l'accesso alla nostra scheda da remoto: come già visto, sfruttiamo il web server di RandA (Tomcat) per rendere visibili le pagine HTML che vogliamo rendere pubblicabili sui client remoti. Di conseguenza, a livello di installazione bisogna solo preoccuparsi (oltre che di crearle!) di depositarle nella cartella:

`/home/apache-tomcat-7.0.47/webapps/RandA/`

da dove Tomcat la preleverà per trasferirla al client che la richiede. La pagina creata appositamente per questa applicazione, che abbiamo chiamato *IRConsole.html*, è visibile in Fig. 6: lo stile grafico è stato impostato sulla falsa riga delle altre pagine web già realizzate per il controllo remoto della board RandA; inoltre abbiamo aggiunto il suo riferimento anche su *index.html* (che permette di aprire le altre pagine con le applicazioni per la gestione di RandA da remoto) in modo che ospiti anche il collegamento alla nostra *IRConsole.html* (Fig. 7). Una volta caricate le pagine nella cartella sopra citata, possiamo verificarne subito il corretto funzionamento con un PC connesso alla rete locale di RandA, digitando nella barra degli indirizzi del browser: `http://<Indirizzo IP RandA>/RandA/IRConsole.html`, dove `<Indirizzo IP>` è l'indirizzo IPv4 della scheda RandA nella LAN. Dovrebbe quindi venire caricata la pagina di Fig. 6, che come già detto corrisponde al file

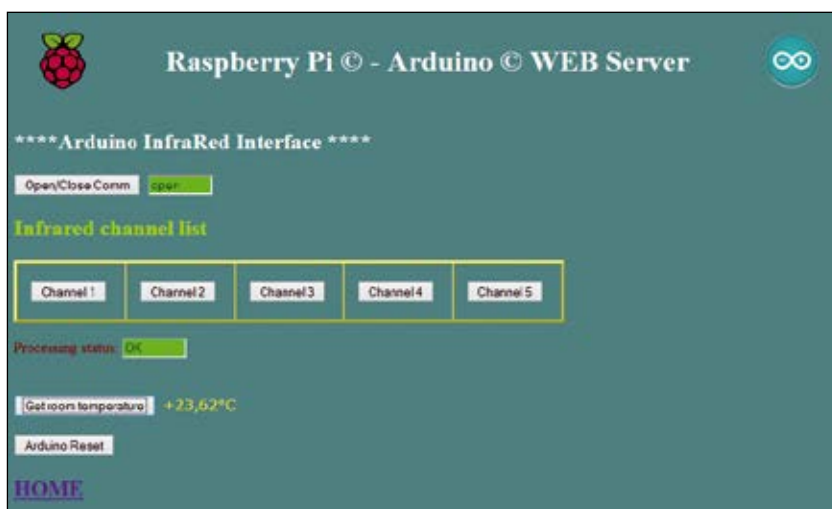


Fig. 6 - La pagina web della nostra applicazione.

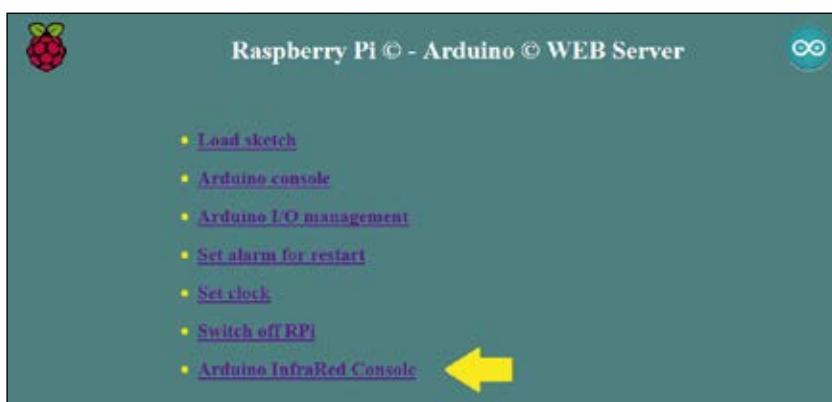


Fig. 7 - La pagina principale.

IRConsole.html precedentemente caricato.

Su questa pagina si può notare subito il primo tasto, relativo alla comunicazione seriale: esso serve ad attivare il canale di comunicazione visto in Fig. 5, dalla servlet alla seriale `/dev/ttyS0` (e quindi ad Arduino). Pertanto, è la prima cosa da attivare ed attendere che la finestra a fianco riporti la scritta "open". Vale la pena ricordare che la seriale di Arduino viene anche utilizzata per la programmazione, quindi se abbiamo attivato la comunicazione con la pagina web non possiamo programmare la scheda, e viceversa. Allo scopo di non lasciare la seriale inutilmente impegnata sarebbe anche buona norma, prima di lasciare la pagina, ricordarsi

di spegnere l'interfaccia agendo nuovamente sul tastino.

Subito sotto abbiamo collocato la lista dei tasti relativi ai vari canali del nostro telecomando virtuale; a tale proposito va detto che è possibile aggiungerne altri (fino a 127) e magari personalizzare i loro nomi, in modo che descrivano la funzionalità associata (ad esempio, "Accendi climatizzatore" al posto di "Channel 1"). Per fare ciò bisogna editare il file *IRConsole.html*, e modificare il codice compreso tra i commenti `<!--Channel list begin -->` e `<!-- Channel list end -->`. Per cambiarne il nome, è sufficiente cambiare il relativo campo "value". La procedura necessaria ad aggiungere dei pulsanti viene per chiarezza descritta a parte in

Come **aggiungere i canali** al nostro **telecomando**

Il progetto è predisposto per acquisire fino a 127 differenti canali, quindi più di quello che servirà nella pratica; potete comunque utilizzare e configurare solo quelli di cui avete bisogno.

Per modificare il numero di pulsanti mostrati nella pagina web, bisogna editare il file `IRConsole.html` ed agire sul codice compreso tra i commenti `<!-- Channel list begin -->` e `<!-- Channel list end -->`. Ad esempio, per incrementare il numero dei pulsanti in fila a quelli presenti, si può aggiungere il seguente codice prima del tag `</tr>`:

[illegible]

Invece per creare un'altra fila sottostante bisogna aggiungere una riga nella tabella che li contiene: è sufficiente ricopiare tutto il codice tra i tags `<tr>` e `</tr>`, compresi.

Se però non vi sentite sicuri di quello che fate, consultate prima una guida su HTML e, nello specifico, la parte riguardante la disposizione degli elementi su tabelle (es. <http://www.html.it/pag/16055/tabella-struttura-di-base>): farete prima ed il risultato sarà migliore. I più esigenti vorranno magari alterare anche la grafica per creare un layout corrispondente ad un telecomando “vero”...

In ogni caso, per ogni pulsante che inserite dovreste:

- 1) assegnare un identificatore ed un nome univoci (campi "id" e "name" rispettivamente); in particolare, l'identificatore dev'essere nel formato ChX, ove X è un numero progressivo (es. Ch6, Ch7, Ch8,...).
- 2) aggiungere la seguente riga nella sezione di codice javascript (in alto nel file), nella funzione window.onload, dopo le altre righe simili:

`el=document.getElementById("<ChX>"); el.onclick=setPin;`
dove `<chX>` è l'identificatore del nuovo pulsante come descritto al punto 1

Come ultima cosa, sempre nella sezione javascript, dovrete aggiornare la variabile NCHANNELS con il numero totale di pulsanti dei canali:

```
var NCHANNELS = <numero totale pulsanti canali>;
```

Dal "lato Arduino", bisogna infine ricordarsi di aggiornare con tale numero anche l'omonima costante dichiarata nello shield:

```
#define NCHANNELS <numero totale pulsanti canali>
```

e quindi riprogrammare la scheda attraverso l'IDE.

questo stesso articolo (nel riquadro "Come aggiungere i canali al nostro telecomando").

Dopo la lista dei pulsanti-canale, vi è la casellina "Processing status" che riporta l'esito della comunicazione con RandA: per qualsiasi comando impartito, visualizza "wait" durante l'attesa di risposta, "fail" se il comando non è stato eseguito oppure se Arduino non ha risposto entro un certo periodo (vi è un timeout nella servlet SerialIO), oppure "OK" se il comando è stato eseguito con successo. Scendendo nella pagina incontriamo i due pulsanti descritti qui di seguito.

- *Get room temperature*: ad ogni “click” aggiorna il valore di temperatura rilevato dal sensore DS18B20 presente nello shield; questa informazione può essere utile se si utilizza ArdIR per il controllo del climatizzatore domestico. Tenere presente che alla prima interrogazione, non avendo ancora conversioni valide, potrebbe visualizzare il valore di default (85°C): non spaventatevi, ma cliccate di nuovo.
- *Arduino reset*: forza un reset del micro di Arduino. In caso di (improbabile) necessità, permette di fare “ripartire” lo sketch.

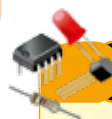
Il link “HOME”, infine, mutuato dalle altre pagine web di RandA, consente di tornare alla pagina iniziale di **Fig. 7**.

CONCLUSIONI

Replicare il codice di un telecomando ad infrarossi è relativamente semplice, utilizzando pochi componenti ed una scheda Arduino; usando invece RandA, si aggiungono altre potenzialità quali l'accesso al Web ed il controllo a distanza.

La scelta dei componenti utilizzati (ricevitore IR1 a 38 kHz) e dei tempi di campionamento del segnale è stata effettuata analizzando alcuni telecomandi a nostra disposizione.

Non è però escluso, seppure improbabile, di incappare in telecomandi sintonizzati su frequenze diverse o con segnali con tempistiche pure diverse; in tal caso, bisognerà identificare la frequenza della portante e sostituire il ricevitore con un componente idoneo a tale valore, ed eventualmente modificare lo sketch per adeguare il tempo di campionamento.



per il MATERIALE

Tutto il materiale necessario per realizzare questo progetto è disponibile presso Futura Elettronica. La scheda ArdIR shield (cod. FT1219K) viene venduta in kit ed è disponibile al prezzo di Euro 18,00. La board RandA (cod. RANDA) costa Euro 39,00 e Raspberry Pi 2 Tipo B (cod. 8326274RS) è in vendita al prezzo di Euro 44,00. 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>



Windows 10



Un tempo antagonista e icona del software chiuso, la casa di Redmond si è avveduta dell'inarrestabile avanzata di Arduino e Raspberry Pi, rendendo compatibile con esse il proprio sistema operativo.

MICROSOFT & OPEN SOURCE: LA STRANA COPPIA

di FABRIZIO GUGLIELMINO

Fino a qualche tempo fa accostare Microsoft all'open source era impensabile, perché Microsoft ha sempre rappresentato l'approccio commerciale al software e spesso ha intrapreso battaglie legali (a dire il vero mai troppo efficaci) contro il movimento open source. Con la fine dell'era Ballmer (ex amministratore delegato di Microsoft) ha però iniziato a "cambiare pelle", segno evidente che il modello di business che l'ha fatta tanto crescere, basato sulla posizione monopolistica di Windows e Office, inizia a perdere colpi. L'attuale CEO, Satya Nadella, sembra aver ben chiara questa situazione e, seppur da poco alla guida della compagnia, ha intrapreso evidenti cambi di direzione. La strategia di base sembra orientata all'apertura per migrare da un'azienda focalizzata sui prodotti verso una rivolta ai servizi. Azure, la piattaforma cloud targata MS, dimostra chiaramente questa apertura supportando ufficialmente Linux e tecnologie non particolarmente comuni nell'ambito Microsoft, come Python, Node.js e molti altri.

La nuova strategia sta gradualmente sconfinando anche verso il mondo dei Maker, che Microsoft sembra voler corteggiare con particolare attenzione, come spiegheremo in questo articolo valutando le opportunità che si prospettano.

WINDOWS 10: LA PIATTAFORMA UNICA

La nuova strategia non abbandona il sistema operativo, che da sempre identifica l'azienda, ma gli cambia pelle. Windows 10 non è più un semplice sistema operativo per PC ma una piattaforma unica per vari dispositivi.

La release ufficiale è attesa per questi mesi con un grosso carico di novità: l'unificazione renderà Windows 10 accessibile dal PC allo smartphone passando per tablet e console, fino a giungere ai sistemi embedded. Proprio quest'ultimi sono quelli che più ci interessano, ed in particolare Arduino e Raspberry Pi.

L'offerta di Microsoft non si limita in realtà ai soli dispositivi citati, perché già da qualche tempo Microsoft ha manifestato il proprio interesse per

l'IoT, sia supportando diverse piattaforme embedded, sia iniziando ad offrire servizi specifici nella propria piattaforma di Cloud: Azure. Entrando un po' nel dettaglio, Windows 10 sarà il primo sistema operativo certificato per Arduino.

Ma all'atto pratico cosa significa? Semplificando, Windows 10 introduce quella che viene definita la Universal Windows Platform (UWP), ovvero un set di API (Application Program Interface) che poggiano su Windows Core per rendere una serie di funzionalità del sistema operativo accessibili da diversi contesti. Riprendendo uno degli esempi presentati dall'azienda, se un Maker realizza una videocamera di sicurezza con motori per il brandeggio, controllata da Arduino, può usufruire delle funzionalità della UWP, per esempio per mandare le immagini sul cloud, implementare il riconoscimento dei volti o realizzare un'interfaccia di controllo. Tutto questo, avvalendosi di funzionalità nativamente presenti nel sistema.

LE NOVITÀ PER ARDUINO

I progetti che si basano su Arduino sono innumerevoli e buona parte di essi permette di interagire con smartphone e computer. Analizzandoli, però, salta subito all'occhio che vengono utilizzati tanti metodi di interfacciamento quanti sono i progetti stessi. In alcuni casi vengono scritti protocolli proprietari, così una funzionalità banale, come eccitare un relé collegato ad Arduino da uno smartphone, richiede l'implementazione del protocollo anche per il telefono.

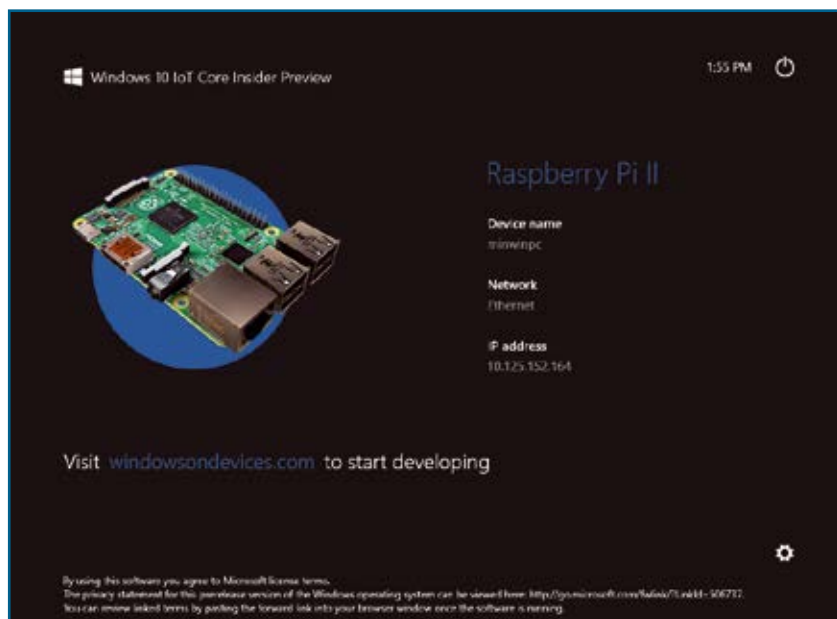
In altri casi vengono usate librerie più o meno standard che implementano meccanismi di comunicazione tra Arduino e il computer; questa condizione non agevola la condivisione dei progetti tra i membri della community. Integrare progetti diversi tra loro implica un costo, spesso non trascurabile, per capire e adattare i meccanismi di comunicazione implementati.

Microsoft ha compreso perfettamente questo scenario ed ha intuito che offrendo una soluzione standard coprirebbe quest'area

grigia con un ritorno di immagine e diffusione non trascurabile. Ma all'atto pratico, in cosa consiste la proposta Microsoft? Al momento in due progetti chiamati **Windows Remote Arduino** e **Windows Virtual Shield**. Vediamo di cosa si tratta, iniziando proprio da Windows Remote Arduino (WRA): si tratta di un'implementazione del protocollo "Firmata" per Windows; tale protocollo permette di comunicare direttamente con il processore della scheda da uno dei device su cui gira Windows 10.

In pratica risulta estremamente semplice creare applicazioni che girano, per esempio, su uno smartphone e comunicano con Arduino. Per vedere quanto può essere semplice effettuare questo tipo di operazioni, nel **Listato 1** proponiamo un pezzetto di codice (uno snippet, come si suol dire). Il codice è scritto in C# e viene eseguito sul dispositivo Windows 10 (per esempio, il cellulare). Non è un esempio completo, ma analizzandolo ci accorgiamo subito di una certa familiarità. Dopo le prime righe, che inizializzano gli oggetti necessari alla comunicazione, saltano all'occhio le similitudini con il codice che tipicamente si scrive nell'IDE di Arduino. Il setup dei due pin in output e input rispettivamente, il set allo stato alto del primo e la lettura di un valore analogico risultano comprensibili anche a chi non ha mai scritto codice per Windows, ma l'ha fatto su Arduino. La potenza di WRA è evidente, con meno di 10 righe di codice si è in grado di interagire con gli input/output della scheda in modo semplice ed immediato.

Non è da meno Windows Virtual Shield (WVS), anzi, se possibile, è ancora più interessante. In questo caso il tipo di interazione



Listato 1

```
using namespace Microsoft.Maker.RemoteWiring;
using namespace Microsoft.Maker.Serial;

// Connessione bluetooth e dell'interfaccia di gestione di Arduino
IStream bt = new BluetoothSerial();
RemoteDevice arduino = new RemoteDevice( bt );

// Avvio della connessione
bt.begin(115200, 0);

// Setup di due pin
arduino.pinMode(7, PinMode.OUTPUT);
arduino.pinMode(9, PinMode.INPUT);
// Set del pin 7
arduino.digitalWrite(7, PinState.HIGH);
// Lettura di un valore analogico dal pin 9
UInt16 val = arduino.analogRead(9);
```

è inverso rispetto a quello visto con WRA, in quanto il sistema trasforma il dispositivo Windows 10 in uno shield virtuale utilizzato da Arduino.

Uno dei punti di forza di Arduino sono proprio gli shield, che offrono la possibilità di estendere le funzionalità della scheda con dei componenti hardware standard; per questo attorno ad Arduino è sorto un ecosistema di accessori che copre praticamente ogni possibilità: esiste lo shield GPS per progetti che necessitano di localizzazione, lo shield WiFi per avere la connettività wireless, lo shield con il display per avere un dispositivo di output grafico o testuale ecc.

Con WVS il nostro device Windows 10 viene visto da Arduino come uno shield che rende disponibili tutti i sensori di cui dispone. A livello pratico la cosa è gestita da alcune librerie da usare negli sketch che eseguiamo su Arduino e da un'App da installare in Windows 10. Con queste due componenti, collegando la scheda a Windows via Bluetooth o direttamente via cavo, Arduino sarà in grado di accedere ai sensori, usare il telefono cellulare come gateway verso Internet o mostrare informazioni sul display.

Come per la gran parte degli shield ad oggi disponibili, l'utilizzo di WVS è mediato da una serie di funzionalità rese accessibili dalle librerie fornite da Microsoft; queste offrono le funzioni specifiche per interagire con i sensori del dispositivo Windows 10, quindi, per esempio, è disponibile una classe Text che permette di visualizzare del testo, per esempio sul telefono. Così come la classe Camera per interagire con la fotocamera, Speech per usufruire del sistema di sintesi vocale, o ancora Sms

per inviare messaggi su rete mobile e così via, praticamente per ogni componente che l'hardware su cui è installato Windows può offrire.

UTILIZZI PRATICI

Abbiamo visto, con un rapido excursus, alcune delle componenti di Windows 10 che lo rendono il primo sistema operativo certificato per Arduino. Con esso possiamo fare un'infinità di cose: con WRA per esempio possiamo creare l'interfaccia di controllo di un sistema domotico direttamente sul cellulare Windows Phone. Con WVS invece potremmo creare un apricancello collocato in auto, che leggendo le coordinate GPS dal cellulare invia il segnale di apertura solo quando siamo in sua prossimità.

Questi due semplici esempi dimostrano il potenziale di questi oggetti; va detto che le funzionalità di WRA e WVS non sono novità assolute; già oggi esistono tool e librerie che permettono di fare le stesse cose anche su sistemi operativi non Windows. La vera novità apportata da Windows 10 è la possibilità di rendere queste funzioni parti integranti del sistema, così da creare una sorta di ecosistema sul quale è altamente probabile che nascerà una miriade di applicazioni pratiche.

CONCLUSIONI

Abbiamo citato ad inizio articolo anche Raspberry Pi, ma la questione meriterebbe intere pagine. Per darvi un'anteprima diciamo che, a differenza di quanto visto per Arduino, è stata realizzata una versione di Windows 10 da installare su Raspberry Pi (solo sulla versione 2). Si tratta di una versione "ridotta" di Windows 10, definita Windows 10 IoT Core Insider, priva dell'interfaccia grafica ma con le funzionalità basilari (il core, appunto); va vista come un'alternativa alla Raspbian, che è ormai la distribuzione Linux di riferimento per Raspberry. Inoltre va detto che Windows 10 IoT Core per Raspberry Pi è gratuito il che rende la cosa ancora più interessante. Quello che abbiamo visto rappresenta solo la punta dell'iceberg, perché il grosso delle novità deve ancora palesarsi e i prossimi mesi saranno decisivi. Il fatto che un'azienda come Microsoft punti i riflettori su realtà come Arduino e Raspberry Pi dimostra come il movimento dei Maker stia facendosi sempre più strada in quello che viene definito il mondo "Enterprise". Non passerà molto tempo prima che le competenze su questi sistemi inizieranno ad essere richieste in ambito professionale, dando una spinta ancor maggiore all'innovazione. ■

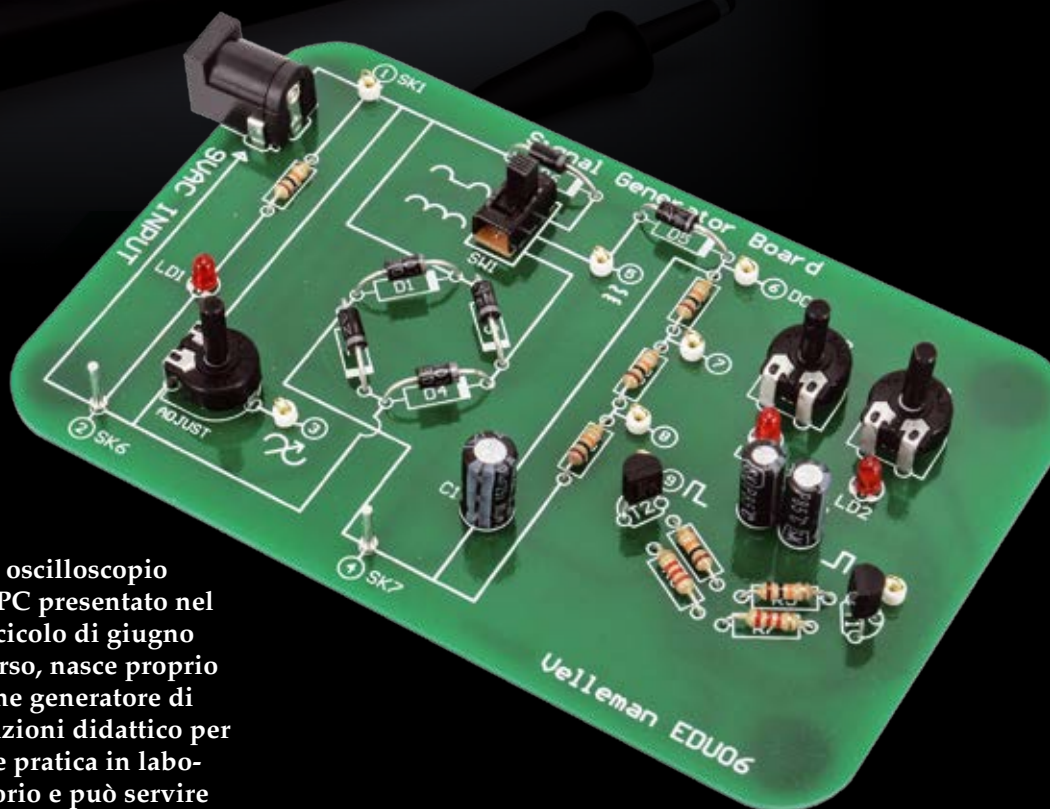
GENERATORE DI SEGNALI CAMPIONE

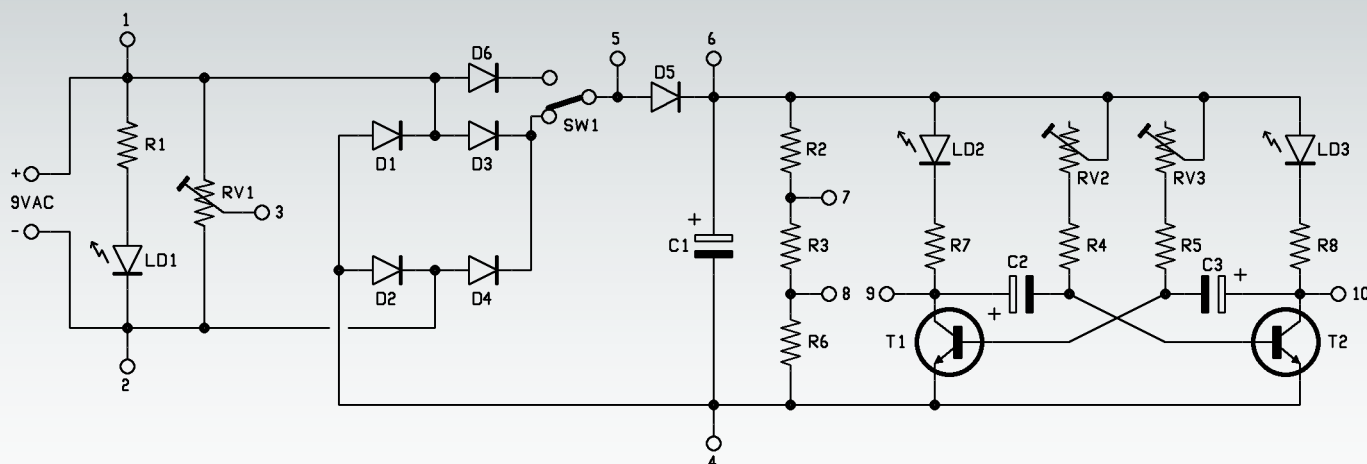
di DAVIDE SCULLINO

Genera tensioni costanti e variabili con le quali si può fare pratica con l'utilizzo delle funzioni di base dello oscilloscopio; ideale per gli studenti o per chi si approccia per la prima volta a tale preziosissimo strumento.

Per gli studenti che iniziano ad utilizzare la strumentazione da laboratorio, l'oscilloscopio potrebbe rappresentare uno scoglio difficilmente superabile; per questo può essere utile disporre di un generatore di segnali da utilizzare per esercitarsi a compiere le misure principali e ad utilizzare correttamente i comandi di base dei tempi, ampiezza, trigger, accoppiamento dell'ingresso. Il progetto qui descritto, naturale complemento del digitalizzatore

per oscilloscopio da PC presentato nel fascicolo di giugno scorso, nasce proprio come generatore di funzioni didattico per fare pratica in laboratorio e può servire anche a chi ha comperato il suo primo

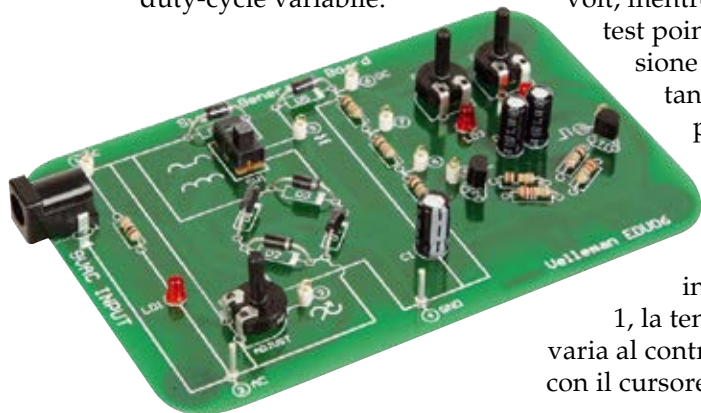




oscilloscopio, di qualunque tipo sia, per vedere come risponde ai vari segnali.

Il circuito che vi proponiamo di realizzare non è il classico generatore di funzioni che produce tensioni sinusoidali, quadre, triangolari ecc, ma eroga tensioni semplici, due delle quali a frequenza variabile. Per l'esattezza possiamo contare su:

- una componente alternata di ampiezza regolabile tra zero e un massimo;
- una componente sinusoidale di ampiezza fissa;
- una componente raddrizzata a doppia semionda di ampiezza fissa;
- una componente raddrizzata a singola semionda di ampiezza fissa;
- una componente continua di varie ampiezze;
- un segnale rettangolare a frequenza variabile;
- un segnale rettangolare a duty-cycle variabile.



Per capire da dove nascono queste tensioni campione, diamo adesso uno sguardo allo schema elettrico del generatore.

SCHEMA ELETTRICO

Il circuito è molto semplice e consta di un raddrizzatore a diodi componibile mediante l'inserzione del deviatore SW1, di un partitore a scala di resistenze, di un potenziometro e di un multivibratore astabile a frequenza e duty-cycle variabili. Il tutto deve essere alimentato con un trasformatore da rete avente il secondario da 9 Veff, ovvero un alimentatore in alternata provvisto di plug adatto a quello della scheda. L'alimentazione si applica al plug di alimentazione e raggiunge il bipolo R1/LD1, il cui LED ne segnala la presenza, e il potenziometro RV1; tra il punto 1 e il punto 2 si preleva l'alternata sinusoidale così com'è, dell'ampiezza di 9 volt, mentre tra il punto 1 e il test point 3 si legge una tensione variabile di valore tanto maggiore quanto più il cursore del trimmer si avvicina al test point 2. Riferendosi a quest'ultima invece che al punto 1, la tensione sinusoidale varia al contrario, ovvero è zero con il cursore verso il punto 2 e

massima verso il punto 1. L'alternata raggiunge poi il raddrizzatore configurabile formato dal ponte di Graetz composto dai diodi D1, D2, D3, D4, oltre che D6 e dal deviatore SW1; quest'ultimo permette di far raddrizzare l'alternata a doppia semionda -tramite il ponte- o a singola semionda (da D6). Quando SW1 è chiuso su D3/D4, si inserisce il ponte di Graetz, che funziona così: nella semionda in cui la polarità è positiva sul punto 1, D3 conduce, la corrente fluisce in D5 (che conduce anch'esso) carica l'elettrolitico C1 e percorre il partitore multiplo R2, R3, R6, oltre ad alimentare il multivibratore astabile che segue. La corrente esce dal punto 4, entra in D2 (D1 rimane interdetto perché il suo catodo ha un potenziale maggiore di quello dell'anodo) ed esce dal suo catodo per tornare al punto 2 e al plug. D4 rimane interdetto perché il suo catodo ha un potenziale maggiore di quello dell'anodo.

Nella semionda opposta, ovvero quando la polarità è positiva sul punto 2 e negativa sul punto 1, la corrente entra in D4 (D2 rimane interdetto) e prosegue attraverso il solito D5, passando nel resto del circuito e uscendo dal punto 4, da dove ritorna tramite D1 (D3 risulta interdetto) verso il punto 1 e il plug di alimentazione.

La sonda dell'oscilloscopio

Nel digitalizzatore per oscilloscopio digitale cui facciamo riferimento in questo articolo è stata proposta una sonda semplificata, tuttavia si può utilizzare la tradizionale sonda da oscilloscopio mostrata in questo riquadro, che è opportuno conoscere in quanto equipaggia gli strumenti professionali.



1. Punta del puntale; corrisponde al polo positivo dell'ingresso dello strumento. È isolato opportunamente dall'impugnatura per evitare di prendere la scossa quando si eseguono misure su circuiti a tensione relativamente alta.
2. Connettore BNC; si inserisce nella presa dell'oscilloscopio ruotandone la ghiera per fissarlo.
3. Coccodrillo di massa; stabilisce il contatto con la massa di riferimento dell'ingresso (è il negativo della sonda).
4. Clip della sonda; è collegata al polo positivo e si estrae tirando l'astuccio che la contiene in modo da agganciarsi a test point e fili, terminali di componenti e altri punti dove si deve effettuare la misura ma non è possibile o conveniente tenere il puntale con la mano.
5. Selettore x1/x10; permette di lasciar passare la tensione inalterata o di ridurla di 10 volte. Quando è su x1 il segnale raggiunge l'ingresso dell'oscilloscopio senza attenuazione, mentre su x10 il segnale viene attenuato 10 volte, il che permette di misurare tensioni relativamente elevate senza danneggiare l'ingresso dello strumento. Notare che nel partitore di tensione è inserito un condensatore variabile che va tarato, con un cacciavite, per limitare la deformazione dei fronti d'onda adattando la capacità della sonda a quella d'ingresso dell'oscilloscopio; per la taratura si può usare il segnale CAL normalmente prelevabile su un test point collocato sul pannello frontale dell'oscilloscopio.

Notate che il ponte di Graetz rende unidirezionali tensione e corrente tra il test point 5 ed il punto 4, qualunque sia il verso della componente applicata al jack d'ingresso; tra il punto 5 ed il punto 4 con l'oscilloscopio si possono visualizzare impulsi sinusoidali positivi su 5, alla frequenza di 100 Hz perché è il doppio di quella della tensione proveniente dal jack, derivata dall'alternata di rete.

Ora immaginiamo di chiudere SW1 sull'estremo opposto, in modo da isolare D3/D4: l'alternata viene raddrizzata a singola semionda da D6, che conduce quando la tensione fornita dal trasformatore ha polarità positiva sul punto 1, come fa anche D5; la corrente carica C1, scorre nel partitore a scala R2, R3, R6 e alimenta il multivibratore astabile, quindi ritorna attraverso il punto 4, stavolta collegato direttamente al punto 2 e al jack di alimentazione. Nella semionda opposta D5 blocca la corrente e lo stesso fa D6. Tra il test point 5 e il punto 4 questa volta con l'oscilloscopio troviamo una tensione pulsante sinusoidale alla frequenza di 50 Hz, giacché il raddrizzatore elimina la metà negativa della sinusoide.

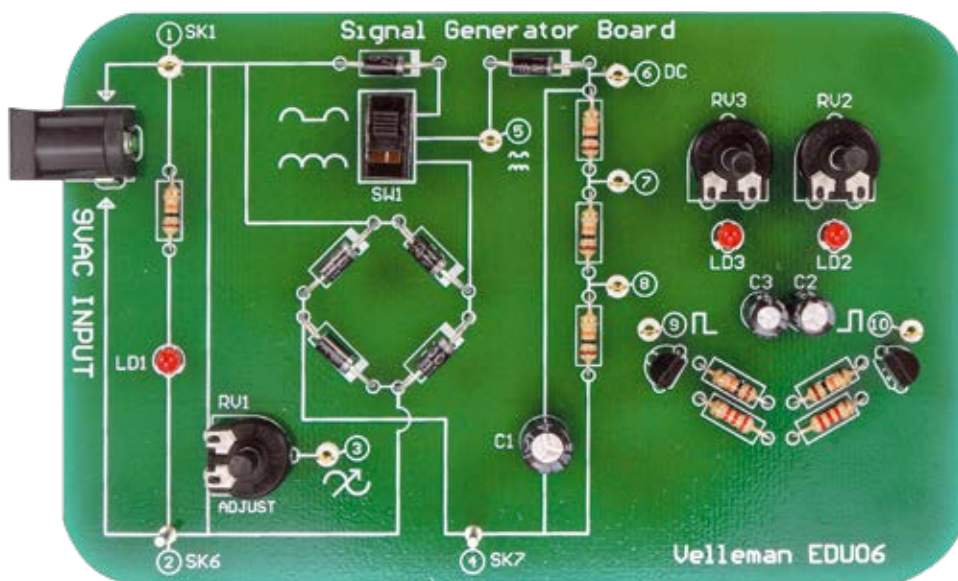
Indipendentemente dalla posizione dell'SW1, tra il test point 6 ed il punto 4 si misura una tensione continua, con un minimo di ripple (ondulazione residua), il cui valore è pari a $1,414 \times 9 \text{ V}$ diminuito della caduta su due diodi (1,2V) se raddrizziamo a singola semionda e su tre se inseriamo il ponte di Graetz. Quindi alimentando il circuito a 9 Vca efficaci, si ottengono circa 11,5 volt nel primo caso e circa 10,9 V nel secondo.

Tramite il partitore è possibile prelevare 1/3 della tensione presente ai capi di C1 su ciascuna

delle resistenze che lo compongono, ovvero 1/3 tra il punto 4 ed il test point 8, 2/3 tra il punto 4 ed il test point 7, 1/3 tra 8 e 7 e 1/3 tra 7 e 6 (si rileva 2/3 della tensione sull'elettrolitico anche misurando tra 8 e 6).

Concludiamo l'analisi dello schema elettrico con il multivibratore astabile, che è la parte facente capo ai due transistor:

si tratta di un generatore di impulsi rettangolari dotato di due uscite -9 e 10- che forniscono due onde rettangolari in opposizione di fase tra loro (vale a dire che quando una è a livello basso l'altra è ad 1 logico e viceversa); grazie ai trimmer RV2 ed RV3, possiamo regolare la durata dei singoli impulsi e quindi sia la frequenza delle onde, sia il loro



Elenco Componenti:

- R1: 10 kohm
- R2: 10 kohm
- R3: 10 kohm
- R4: 10 kohm
- R5: 10 kohm
- R6: 10 kohm
- R7: 1 kohm
- R8: 1 kohm
- RV1: Trimmer 10 kohm MO
- RV2: Trimmer 250 kohm MO
- RV3: Trimmer 250 kohm MO
- C1: 100 μ F 25 VL elettrolitico
- C2: 10 μ F 63 VL elettrolitico
- C3: 10 μ F 63 VL elettrolitico
- D1: 1N4007

scende a un potenziale di qualche decimo di volt e scarica C2, la cui armatura negativa porta la base del T2 a un potenziale inferiore a quello che servirebbe a innescare la conduzione; lo stato di saturazione fa scorrere la corrente di collettore del T1 nel LED LD2 (che si illumina) e nella resistenza R7 che ne limita la corrente.

Nel frattempo siccome T2 è interdetto, attraverso R4 ed RV2 il condensatore C2 si carica con polarità positiva sul negativo, fin quando la giunzione base-emettitore del T2 non raggiunge una differenza di potenziale superiore a quella di soglia; a questo punto il transistor va in saturazione e il suo collettore trascina a circa 0 volt il positivo dell'elettrolitico C3 e determinando due condizioni: la prima è che C3 ora viene scaricato e determina un impulso negativo sulla base del T1, tale da far interdire questo transistor e spegnere LD2. La seconda è che la corrente di collettore del T2 porta a pochi decimi di volt la

resistenza R8, facendo illuminare LD3.

Adesso C2 viene scaricato attraverso R4, RV2, R7 e LD2, per poi caricarsi con polarità positiva sull'armatura positiva, attraverso LD2 ed R7; quando la carica è sufficiente a far scendere la tensione base-emettitore del T2 sotto gli 0,6 volt, tale transistor passa in interdizione e ricomincia il ciclo descritto finora. Tale ciclo determina le tensioni rettangolari sui collettori di T1 e T2. Va notato che siccome i tempi di carica/scarica degli elettrolitici C2 e C3 dipendono in buona parte dalla resistenza inserita dai trimmer, rispettivamente RV2 ed RV3, regolando tali componenti è possibile variare la durata dello stato logico alto di un'onda e, di riflesso, quello della pausa dell'altra e viceversa, quindi si cambia anche il duty-cycle delle onde rettangolari generate. Più esattamente, da RV2 dipende la durata dello stato logico alto sul test point 10 e da RV2 quello dell'1 logico sul test point 9.

Ogni volta che si ritocca la posizione di un trimmer si cambia inevitabilmente, oltre al duty-cycle, anche la frequenza delle onde rettangolari, dato che si va a variare la durata del periodo.

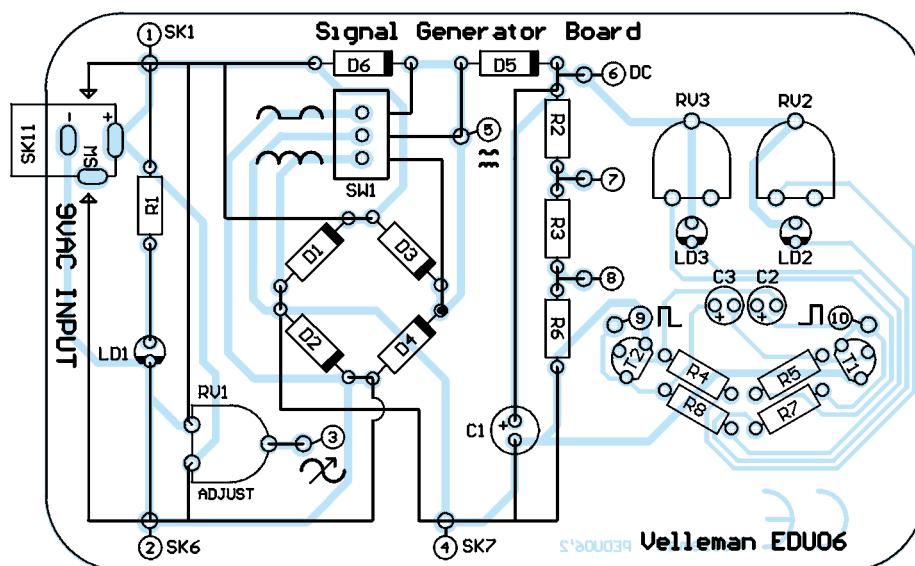
REALIZZAZIONE PRATICA

Spiegato come funziona il generatore di segnali, passiamo adesso alle note costruttive, con la premessa che, vista la semplicità, il circuito può essere realizzato su un pezzetto di basetta millefori. Se invece volete montare i componenti su circuito stampato come abbiamo previsto, la prima cosa da fare è preparare la basetta seguendo la traccia rame scaricabile dal nostro sito www.elettronicaain.it insieme agli altri file del progetto. Seguendo la natura didattica del circuito, lo stampato è di facile costruzione in quanto è a singola ramatura. Inciso e forato il c.s., montate per prime le resistenze e poi i diodi al silicio, quindi posizionate i trimmer, il deviatore a slitta e i transistor; proseguite con i LED, gli elettrolitici e la presa

D2: 1N4007
 D3: 1N4007
 D4: 1N4007
 D5: 1N4007
 D6: 1N4007
 T1: BC547
 T2: BC547
 LD1: LED 3 mm rosso
 LD2: LED 3 mm rosso
 LD3: LED 3 mm rosso
 SW1: Deviatore slitta

Varie:

- Plug alimentazione
- Alberino per trimmer (2 pz.)
- Circuito stampato PEDU06



jack da stampato. Nelle piazzole al punto 2 e al punto 7 (sono le "masse di riferimento") inserite e saldate dei capocorda, mentre in tutte le altre piazzole (punto 1, 3, 5, 6, 7, 8, 9, 10) dovete montare dei test point, ovvero altrettanti capocorda. Per l'orientamento dei diodi al silicio, dei LED e dei transistor, oltre che dei condensatori elettrolitici, riferitevi al piano di montaggio visibile in queste pagine. Completate le saldature, portate i cursori dei trimmer tutti a metà corsa e avete terminato.

LE MISURE

Bene, detto questo vediamo come utilizzare il generatore di segnale per fare le vostre prove: per prima cosa alimentate il circuito con un trasformatore dotato di spinotto plug adatto alla presa montata sullo stampato (non esiste polarità, quindi non vi preoccupate...) capace di erogare 9 volt e una corrente anche solo di 50 milliampere; verificate che una volta fornita l'alimentazione si accenda LD1.

Le prove che proponiamo sono le sei seguenti; se utilizzate l'oscilloscopio presentato il mese scorso presumiamo che abbiate collegato la scheda al computer ed avviato in quest'ultimo l'apposito software.

Misura di una tensione alternata

La prima consiste nel visualizzare nell'oscilloscopio la forma d'onda sinusoidale fornita dal trasformatore: allo scopo portate la massa della sonda dell'oscil-

loscopio sul test point 2 e il positivo (puntale rosso) sul punto 1, quindi dovete:

- impostare "AC coupling" (nel nostro oscilloscopio basta fare clic su AC);
- impostare time/div a 10 ms/div.;
- impostare volts/div su 10 V/div.

In alternativa potete utilizzare la funzione Autoset (impostazione automatica, che nel nostro

Fig. 1

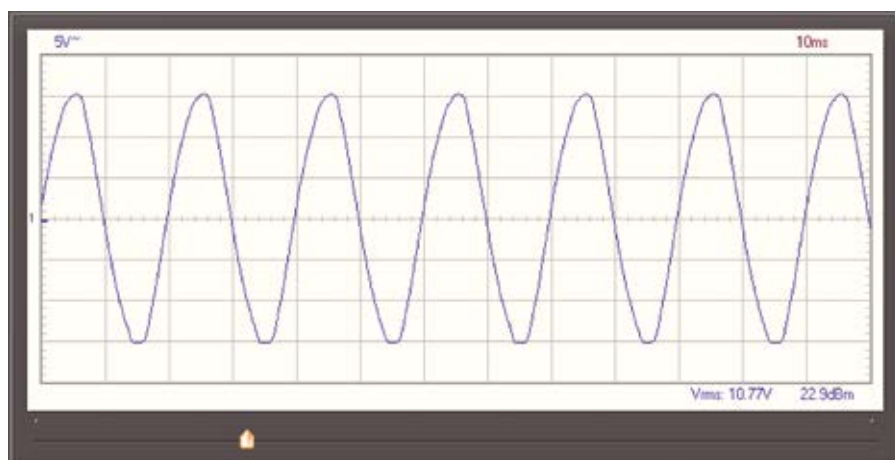




Fig. 2

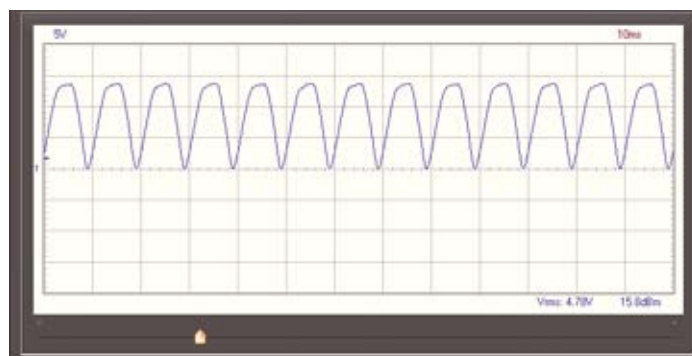


Fig. 3

oscilloscopio si attiva con un clic sul pulsante omonimo). Lo schermo dell'oscilloscopio mostrerà un'onda sinusoidale il cui periodo occupa due divisioni e l'ampiezza picco-picco è circa 6 quadretti (Fig. 1).

Notate che in alto a destra dello schermo il nostro oscilloscopio mostra la base tempi impostata e in basso il valore efficace della tensione alternata (Vrms).

Misura di una tensione alternata di ampiezza variabile

Adesso spostate il puntale positivo della sonda sul test point 3, corrispondente al cursore del potenziometro RV1, lasciando le stesse impostazioni della misura precedente. Ruotando il cursore del trimmer vedrete variare l'ampiezza della sinusoide, ma il periodo rimane di larghezza invariata.

Notate che anche in questo caso in alto a destra dello schermo il

nostro oscilloscopio mostra la base tempi impostata e in basso il valore efficace della tensione alternata (Vrms).

Analisi delle tensioni raddrizzate

Spostate il puntale positivo della sonda sul test point 5 e il negativo sul punto 4, quindi:

- impostate "DC coupling" (nel nostro oscilloscopio basta fare clic su DC);
- impostate time/div a 10 ms/div.;
- impostate volts/div su 10 V/div.

Ora lo schermo dell'oscilloscopio deve mostrare la situazione di Fig. 2 se SW1 è spostato verso D6 e quella di Fig. 3 se lo stesso è commutato su D3 e D4.

In entrambi i casi vediamo impulsi sinusoidali, la cui durata è una divisione in orizzontale (l'ampiezza è di poco superiore alle 2,5 divisioni).

Siccome la base dei tempi è impostata a 10 ms/div. ciascun impulso dura 10 millisecondi. Con il raddrizzamento a singola semionda (SW1 sul catodo del D6) rileviamo un periodo ($T = \text{impulso} + \text{pausa}$) di 20 millisecondi, il che ci conferma che la rete è a 50 Hz (T vale $1/f$, dove f è la frequenza).

Notate che con SW1 nella posizione opposta la frequenza raddoppia perché il ponte di Graetz ribalta sul piano positivo le semionde negative; questo spiega perché il raddrizzatore a doppia semionda offre una tensione continua più pulita, dato che ricarica prima il condensatore di filtro in virtù della raddoppiata frequenza di ricarica.

Analisi della tensione raddrizzata e filtrata

Lasciando il puntale negativo sul punto 4, spostate il positivo sul contatto 6: mantenendo inalterate, rispetto al caso precedente, le

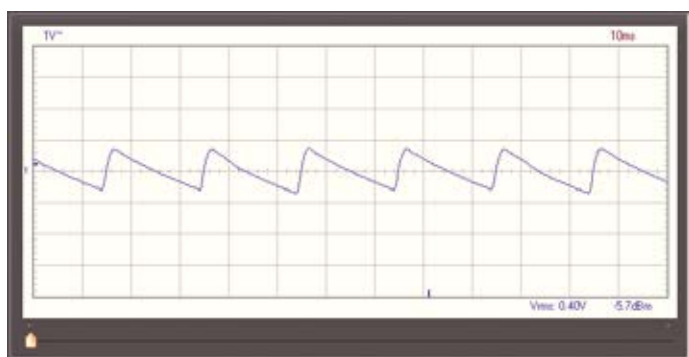


Fig. 4

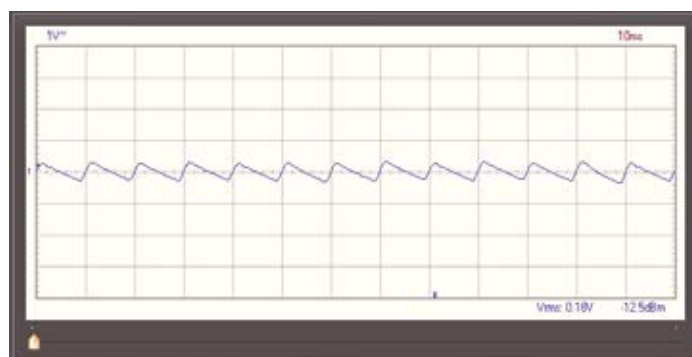


Fig. 5

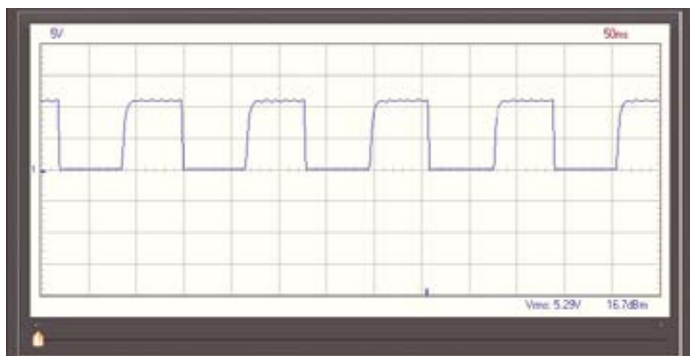


Fig. 6



Fig. 7

impostazioni dell'oscilloscopio, vedrete una tensione continua, indipendentemente da dove spostate SW1; tuttavia su D6 vedrete un'ondulazione residua maggiore di quella analizzabile spostando il deviatore sull'uscita del ponte di Graetz (raddrizzamento a doppia semionda). Per vedere meglio l'ondulazione, impostate l'accoppiamento della sonda su AC Coupling, in modo che l'ondulazione (ripple) della tensione dovuta alla ciclica carica/scarica del condensatore sia centrata rispetto all'asse orizzontale e alzate la sensibilità selezionando 1V/div.

Poi spostate SW1 su D6: questa impostazione, corrispondente al raddrizzamento a singola semionda, determina la visualizzazione di **Fig. 4**, dove il ripple è a 50 Hz. Spostate quindi SW1 dall'altra parte e vedrete che il ripple raddoppia la propria frequenza arrivando a 100 Hz; però l'ampiezza dell'ondulazione è inferiore (**Fig. 5**) a dimostrazione del fatto che ricaricare il condensatore di filtro C12 (scaricato dalla successiva sezione del circuito) con una frequenza raddoppiata minimizza il residuo d'alternata.

Misura di tensioni continue

Ritornate all'accoppiamento DC e misurate, con i quadretti, il valore della tensione raddrizzata e filtrata; prima di farlo, però:

- impostate "DC coupling" (nel

nostro oscilloscopio basta fare clic sul pulsante DC);

- impostate time/div a 10 ms/div.;
- impostate volts/div su 5 V/div.

La traccia dovrebbe essere una linea orizzontale leggermente ondulata, al valore corrispondente, vale a dire circa 2,5 quadretti in altezza.

Spostate adesso il puntale positivo su 7 e verificate che sia all'altezza di 2/3, ossia poco sopra 1,5 quadretti; infine portate il puntale positivo su 8 e accertate che l'ampiezza della linea orizzontale sia poco meno di un quadretto.

Misura delle tensioni rettangolari

Mantenendo l'accoppiamento DC, portate il puntale positivo sul test point 9, quindi:

- impostate "DC coupling" (nel nostro oscilloscopio basta fare clic su DC);
- impostate time/div a 10 ms/div.;
- impostate volts/div su 5 V/div.

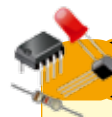
Ora lo schermo dell'oscilloscopio deve mostrare un'onda rettangolare come in **Fig. 6**, la cui durata del periodo (impulso+pausa a zero volt) varierà in funzione delle impostazioni dei trimmer RV2 e RV3. Provate a variare la posizione

del cursore di RV2 e vedrete stringersi gli impulsi (**Fig. 7**) ma anche accorciarsi il periodo.

Se spostate ora il positivo della sonda dell'oscilloscopio sul test point 10 vedrete una situazione speculare, ovvero gli impulsi lunghi come la pausa di poco fa e le pause lunghe come gli impulsi del caso precedente; muovete ancora il cursore dell'RV2 e vedrete cosa accade.

Giocando sull'RV3 ora otterrete una situazione analoga a quella vista con RV2 quando la sonda era sul test point 9.

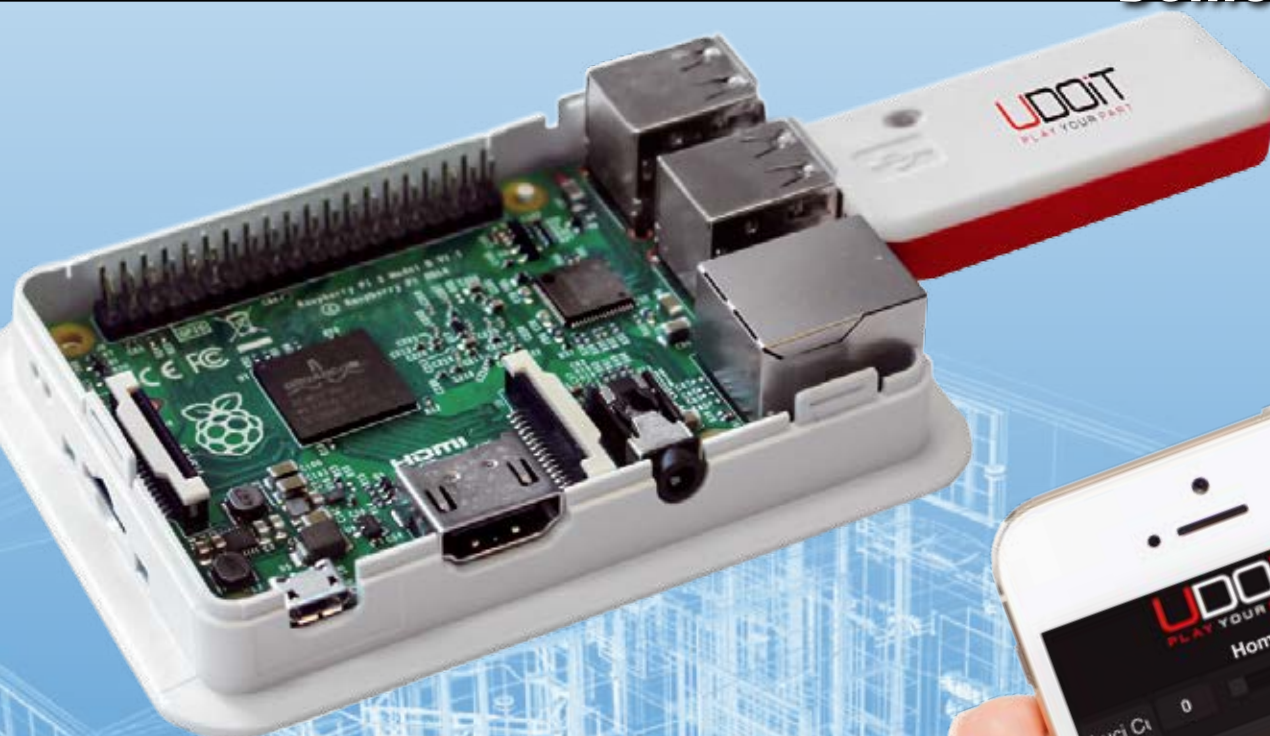
Per variare la frequenza dei segnali rettangolari senza modificare il rapporto tra la durata di pause e impulsi, dovete registrare insieme le posizioni dei trimmer RV2 e RV3.



per il MATERIALE

Questo progetto viene prodotto in scatola di montaggio dalla società Velleman e distribuito in Italia da Futura Elettronica. Il kit è completo di tutti i componenti, minuterie comprese. Il microcontrollore è già programmato, la basetta è forata e serigrafata. È disponibile al prezzo di 24,00 Euro IVA compresa (cod. EDU06).

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



DOMOTICA OPEN SOURCE con i moduli UDOiT

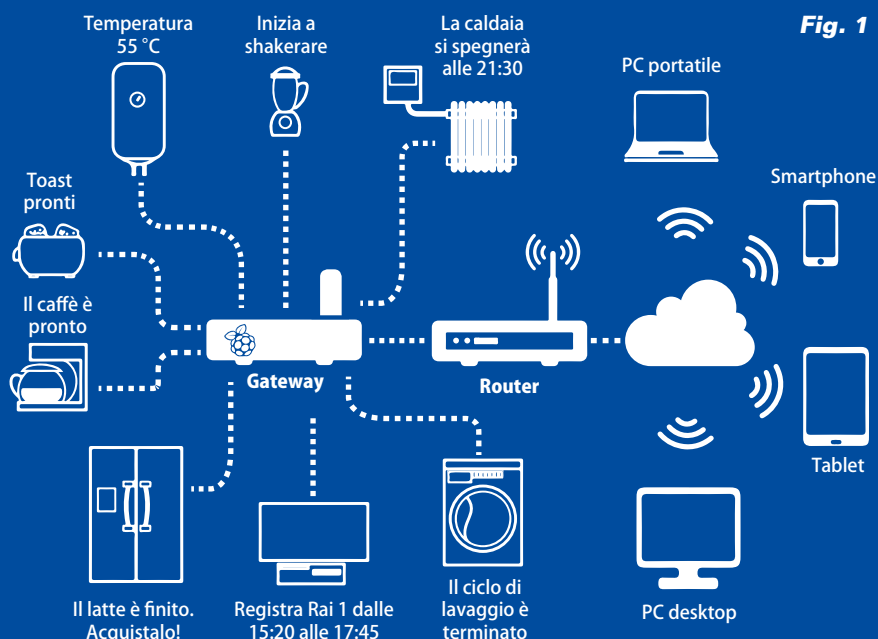
Si parla sempre più di Internet delle cose e delle sue applicazioni. Presentiamo una serie di articoli su una soluzione Open Source di UDOiT basata sui moduli RFTide.

..... di MARCO MAGAGNIN

Da un po' di tempo a questa parte si stanno diffondendo sempre più termini come Iot (Internet of Things), domotica, building automation, e simili. Di conseguenza, come sempre accade, molte aziende stanno orientando le loro proposte commerciali in questa direzione. Esempi si possono trovare nell'impiantistica elettrica, negli elettrodomestici, nei sistemi di supervisione e controllo di ambienti e di persone, soprattutto se bisognose di assistenza, senza trascurare il mondo dell'automobile, dell'intrattenimento, delle comunicazioni,

per finire all'agricoltura, alla meteorologia ed alla gestione degli animali domestici e non. Data la nostra natura di "curiosi" tecnologici ci stiamo interessando, da un po' di tempo, all'evoluzione di questo settore, sia dal punto di vista degli "oggetti" che si sono resi via via disponibili sul mercato, che degli aspetti "filosofici" che caratterizzano le diverse proposte. Parallelamente a questo lavoro di analisi delle soluzioni possibili abbiamo "provato" a costruirci una "mappa" di riferimento basata sulla nostra percezione personale del fenomeno in

Fig. 1

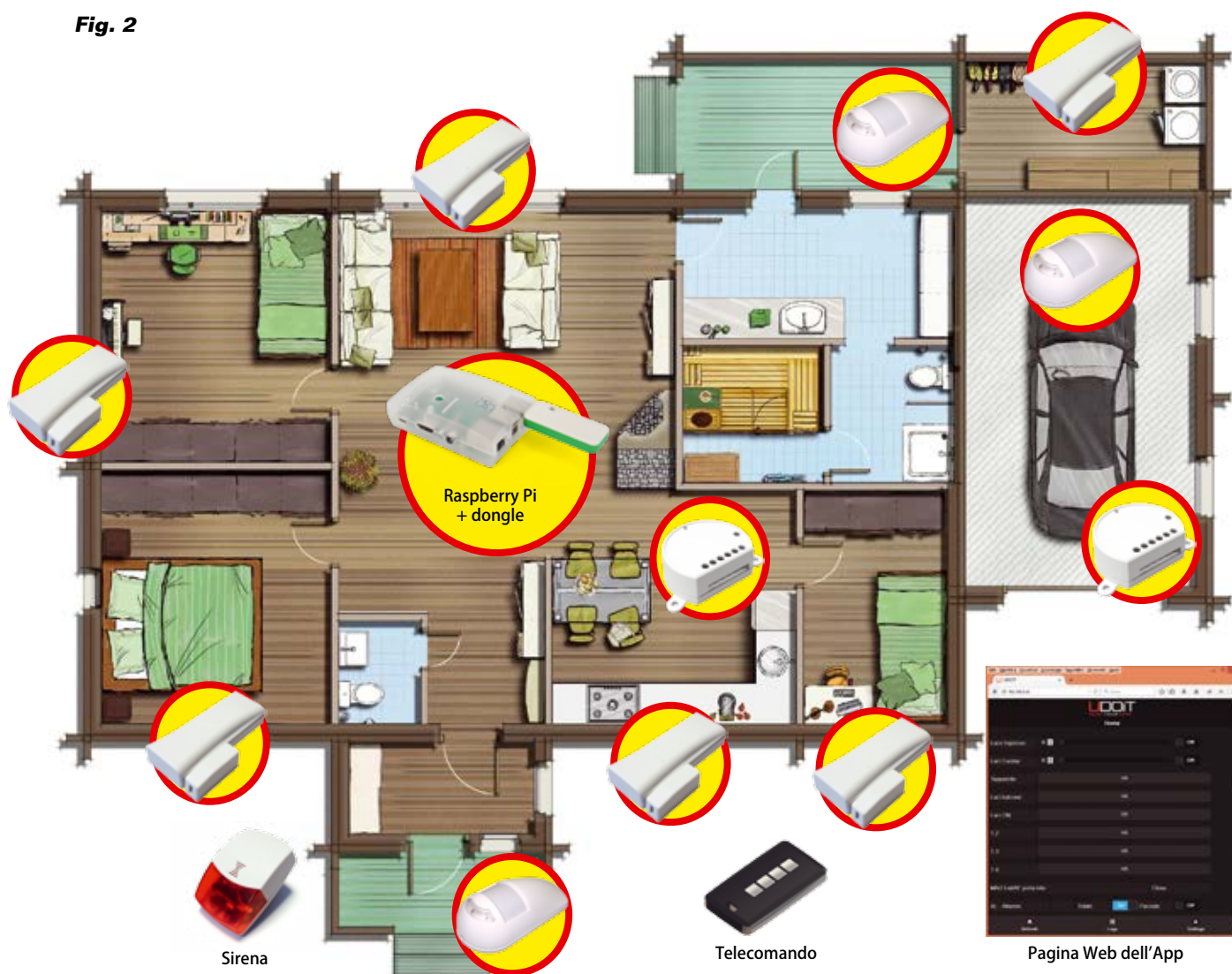


questione. In particolare pensiamo che il motore principale di questo “movimento” sia il desiderio di ridurre sempre di più l’intermediazione umana nella gestione di “contesti” dove la tecnologia è già entrata o si sta apprestando ad entrare. Prendiamo per esempio una casa riscaldata da un impianto a termosifoni, dove i termosifoni hanno delle valvole tradizionali, da operare manualmente. Gli abitanti della casa, quando hanno freddo, aprono le valvole dei termosifoni, che in questo modo trasferiscono calore all’ambiente, mentre quando hanno caldo chiudono le valvole riducendo il trasferimento di calore. Per automatizzare il processo di regolazione della temperatura ambiente si potrebbe utilizzare un “termostato”. Ciò richiede un intervento sull’impianto per sostituire gli “attuatori” manuali, con altri in grado di essere operati automaticamente. Si può intervenire a livello centrale, sulla caldaia, oppure sostituendo tutte le valvole dei termosifoni con altre, operabili automaticamente, in modo da poter regolare singolarmente la temperatura di ogni stanza. Poi, per poter “automatizzare” il processo è necessario “concre-

tizzare” il concetto di caldo e di freddo in termini di grandezze numeriche, per esempio in “gradi centigradi”, per poter regolare, per esempio, un termostato. A questo punto il controllo umano si riduce a “regolare” il termostato centrale o quelli delle singole stanze. Ma perché riscaldare un’abitazione allo stesso modo sia quando è abitata sia quando tutti gli abitanti sono fuori e la casa è disabitata? Ogni volta che si esce di casa si dovrebbe regolare il o i termostati, e lo stesso al ritorno. È vero, esistono termostati con regolazioni multiple basate su timer. Ma ancora, perché riscaldare a tutta potenza quando apriamo le finestre? Una soluzione potrebbe essere quella di collegare un sistema di gestione di presenza, come i controlli di sicurezza periferici dell’antifurto, con il sistema di controllo termico, in modo da farli interagire in base ad un insieme di regole di comportamento finalizzate a gestire correttamente le diverse condizioni che si possono presentare. Bene, avete mai provato a collegare un antifurto ad un termostato? O la centralina meteo all’impianto di irrigazione del giardino? O ancora il citofono al televisore ed al

suo telecomando? E tutto quanto insieme gestibile dallo smartphone quando siete in vacanza alle Maldive? Magari per aprire la porta di casa a qualcuno di fiducia dopo averlo riconosciuto dalle telecamere integrate nello spioncino? Per concludere, secondo noi la chiave di successo di tutto quello che abbiamo descritto nella nostra specie di sogno ad occhi aperti si chiama “interoperabilità” dei sistemi. Che significa? Significa che al momento stiamo assistendo alla proliferazione sul mercato di sistemi, diciamo così, di gestione ambientale, di tipo proprietario che operano in base a specifiche ed a protocolli propri e, per lo più, non documentati, e quindi chiusi rispetto alla possibilità di interfacciarsi ed interagire con sistemi di produttori diversi. Ciò accade, per ironia, anche quando i sistemi proprietari di produttori diversi adottano gli stessi standard di comunicazione tra centralina di controllo, sensori ed attuatori. Questo non giova certo a realizzare la visione di un mondo interconnesso e dotato della capacità, entro certi limiti, di autogestirsi. Per “interoperabilità” intendiamo l’esatto contrario di quanto abbiamo appena descritto. Devo poter scegliere il sistema di gestione dell’impianto termico che preferisco, quello per il condizionatore, quello per la gestione energetica, quello per le comunicazioni ed uno o più per gli elettrodomestici senza dovermi ritrovare con la le pareti tappezzate di centraline che non si parlano tra di loro. Il sistema di gestione di tutti gli impianti deve essere unico ed in grado di gestire tutti gli altri sistemi “periferici”. La soluzione? A nostro parere non è così a portata di mano come potrebbe

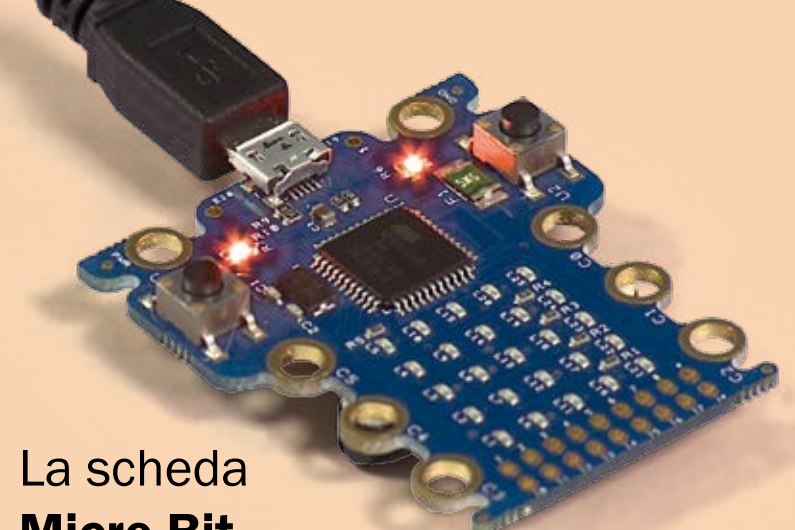
Fig. 2



apparire ad un'analisi superficiale, anche se qualcosa si sta muovendo. I produttori dovrebbero "aprire" i loro sistemi, o quantomeno dotarli di interfacce di comunicazione (API, librerie, web-service e simili) per mettere in grado altri produttori di realizzare sistemi in grado di interconnettere i singoli dispositivi e gestirli in base a regole unificate. Una prima risposta, anche se probabilmente non quella definitiva, sta arrivando dal mondo "open source", con soluzioni realizzate da comunità di sviluppatori finanziati dai produttori di hardware (centraline, sensori ed attuatori), in modo da garantire la possibilità

di integrare tra di loro in modo uniforme e normalizzato le soluzioni proposte dai diversi produttori. Utopia? Non sappiamo, di fatto è la modalità con la quale si sono imposti nel mondo le reti internet (TCP/IP), il web e GNU/Linux, dopo essere sopravvissuti alle infinite schiere di "competitori" con soluzioni proprietarie, divenendo i dominatori incontrastati dei rispettivi mercati di riferimento. In fondo stiamo parlando di Internet of Things, vero? Per coerenza sarebbe bene accettarne le regole. Che poi si parli di "internet" per reti di sensori ed attuatori basate su protocolli di comunicazione totalmente diversi da quelli di

internet, potrebbe essere un buon argomento di discussione. Quanto meno ci limitiamo a prendere atto che al momento si tratta più di uno slogan di marketing che la definizione di una precisa tecnologia, ancora lontana da essere disciplinata in termini di standard e metodologie di progettazione e sviluppo. In quest'ottica, affrontiamo l'argomento presentandovi una soluzione open source, tra le molte esistenti, per la gestione di un ambiente domestico, basata sui sensori, gli attuatori e le periferiche commercializzate da UDOIT, una start-up italiana che realizza prodotti elettronici per il mondo dei makers. Oltre



La scheda **Micro Bit** della BBC

Così come nel passato per aumentare e diffondere la vendita dei libri è stato necessario che i potenziali clienti imparassero a “leggere”, oggi è altrettanto necessario che, gradualmente, gli utilizzatori di sistemi di controllo evoluti, imparino a conoscere, configurare e personalizzare i loro sistemi, utilizzando strumenti di codifica di regole, questo sì, i più semplici possibili. Negli stati più attenti a questi fenomeni in divenire non solo per quanto riguarda l'IoT, ma pressoché in tutti gli aspetti della vita umana, si stanno introducen-

do nuove discipline nelle scuole di tutti i gradi, fin dalle primarie, come i corsi di “coding”, mentre le discipline informatiche stanno cessando di essere considerate materie a sé e stanno divenendo competenze “trasversali” in tutte le discipline oggetto di studio.

Citiamo solo due fatti che hanno caratterizzato il mondo della scuola inglese. Il “regalo” di 15'000 Raspberry Pi da parte di Google alle scuole inglesi e quello della BBC di un milione di schede Micro Bit agli studenti delle scuole secondarie.

alla piattaforma Raspberry Pi (utilizzata nel progetto della domotica qui descritto), sviluppa shield Arduino dedicate al mondo dell'IoT e del M2M con moduli radio in tecnologia LoRa e dispositivi wireless long range per la terminazione volo dei droni, come richiesto dalla recente normativa ENAC. Maggiori informazioni sono disponibili sul sito web www.udoit.eu il tutto 100% made in Italy. Nella **Fig. 1** è rappresentato lo schema di riferimento generale di quello che noi crediamo essere il dominio definibile come Iot, all'interno del quale si può collocare la soluzione hardware e software che vi presentiamo.

Prima di descrivere lo schema di riferimento ci permettiamo alcune doverose precisazioni riguardo l'applicazione software che vi

presentiamo, valide del resto per tutte le applicazioni open source in genere ed anche, a leggere bene le clausole di licenza, anche per le applicazioni con licenza a pagamento. Le applicazioni rilasciate con licenze open source come la Creative Common, ed altre, senza oneri economici da parte degli utilizzatori, in altre parole gratuita, è bene sottolineare che vengono rilasciate “AS IS” ovvero nello stato in cui si trovano, e non è prevista né può essere pretesa alcuna forma di garanzia o di assistenza al loro utilizzo. L'installazione, la personalizzazione e la gestione dell'applicazione è a totale carico dell'utilizzatore e non è previsto alcun supporto in caso di malfunzionamento, richieste di configurazioni, personalizzazioni, modifiche o nuove funzio-

nalità. La disponibilità dei codici sorgenti permette, a chi ha le conoscenze adatte, di risolvere in autonomia eventuali situazioni anomale o di apportare le modifiche desiderate per personalizzare la propria installazione. A nostro avviso è necessario che gli utilizzatori di queste applicazioni abbiano già acquisito un certo grado di conoscenza e dimestichezza con la gestione, amministrazione e programmazione di sistemi GNU/Linux. Al di là della disponibilità di soluzioni Open Source l'argomento IoT introduce una complessità più elevata rispetto a quella tipica associata alle soluzioni di controllo di impianti singoli. Un sistema di riscaldamento gestito da un controllo a termostato richiede funzionalità di gestione limitate ed adatte a soddisfare pressoché ogni ambito dove il controllo viene installato. Un sistema nella filosofia IoT richiede di integrare la gestione di diversi impianti in base ad un insieme di “regole” che devono essere adattate, in base alla tipologia degli impianti ed alle esigenze degli utilizzatori che, tra l'altro sono soggette a continue variazioni nel tempo. È anche vero che le tendenze commerciali spingono a far passare per “semplice” tutto ciò che si vuole abbia ampia diffusione sul mercato. Nel nostro caso, oltre che di una rivoluzione tecnologica, siamo in presenza di una rivoluzione culturale, per la quale viene richiesto agli utilizzatori di acquisire nuove competenze per usufruire al meglio dei nuovi servizi. La soluzione di questo “problema” sta nel formare gradualmente gli utilizzatori ad usufruire delle nuove tecnologie. Vedi al riguardo il box “La scheda MicroBit della BBC”. Tornando al tema più specifico dell'IoT vediamo di descriverne gli

aspetti essenziali, facendo riferimento sempre allo schema di **Fig. 1** Possiamo vedere l'universo IoT come un insieme di discipline e tecnologie che collaborano per interconnettere gli aspetti più svariati del mondo reale con applicazioni in grado di gestirli per conto degli "umani", sollevandoli dal doversene occupare di continuo. Come contropartita, agli "umani" spetta il compito di "insegnare" ai sistemi di comunicazione, coordinamento e controllo, le regole in base alle quali si desidera che i sistemi stessi si comportino.

Per prima cosa vediamo l'esistenza di "insiemi" di sensori che fanno capo ad uno o più "concentratori" e "gateway". Consideriamo un concentratore come un dispositivo che coordina un insieme di sensori ed attuatori della stessa famiglia, finalizzati a presidiare un luogo circoscritto, come un appartamento od una fabbrica. In genere queste "famiglie" di sensori ed attuatori utilizzano per comunicare diverse tipologie di "mezzi": bus su connessioni cablate, sistemi wireless su frequenze dedicate, tecnologie e protocolli di varia natura. Raramente vengono utilizzate tecnologie

e protocolli propri della rete internet. Un gateway, invece, si occupa delle "relazioni esterne", ovvero di far comunicare i sensori con entità di controllo distribuite, in genere messe a disposizione sfruttando la tecnologia cloud e dislocate ovunque nel mondo, dalla stessa stanza agli antipodi del pianeta. Nel caso che vi presentiamo concentratore e gateway "alloggiano" nello stesso dispositivo. Nello stesso dispositivo, che d'ora in poi chiameremo "gateway", alloggiano anche i motori delle regole di gestione "locali". Non necessariamente per accendere una lampada quando entriamo in una stanza dobbiamo coinvolgere un servizio su un server che sta in Australia. Il servizio che sta in Australia ci può tornare utile per gestire il nostro "mondo" quando siamo in vacanza oppure per unificare la gestione di regole da applicare a diversi ambienti in modo coordinato. Per esempio quando torno verso la casa di città ad una certa distanza di chilometri alzo il termostato, oppure faccio accendere il forno quando esco dall'ufficio. Per la parte di gestione assumono estrema importanza le tecnologie di interazione tra "umani" e sistemi. In pratica applicazioni interattive sviluppate in base a molteplici tecnologie in grado di integrare dispositivi di diversa natura come telecomandi, applicazioni web, tablet, smartphone, video-lavagne, smart-watch, smart-car ecc. L'applicazione Open Source che vi presentiamo, si configura come un ponte tra una o più reti di moduli di acquisizione e controllo e la rete internet che permette la remotizzazione della visibilità e dell'archiviazione dei dati acquisiti oltre che il controllo, sempre da remoto delle peri-

Fig. 4



feriche gestite. Di fatto lo scopo dell'applicazione nel suo insieme è di garantire la trasmissione e la gestione delle informazioni dal mondo fisico dei sensori, degli attuatori e delle eventuali periferiche ad essi collegati. Da un lato il gateway gestisce la trasmissione delle informazioni mediante la rete mesh costituita dai moduli RF presenti su ciascun sensore ed attuatore e dall'altro, contemporaneamente, sta la capacità di elaborare i dati ricevuti in base

Fig. 3



Fig. 5

Fig. 6

a regole prestabilite, permettere la gestione in remoto da parte dell'utente fornendo un'apposita interfaccia grafica in tecnologia web. In **Fig. 2** è visibile lo schema d'insieme della soluzione proposta da UDOiT.

I moduli RF di cui sono dotati i sensori, gli attuatori e gli altri componenti del sistema sono i moduli RFTide di Aurel, lavorano in trasmissione e ricezione sulla banda degli 868 Mhz ed i dati scambiati sono codificati sulla base di un protocollo unificato del quale sono rilasciate le specifiche. Un dongle USB dotato anch'esso di un modulo RFTide, permette ai vari sensori ed attuatori di comunicare con l'unità centrale di elaborazione, il gateway vero e proprio, che utilizza un Raspberry Pi B 2. L'unità centrale di elaborazione raggruppa in sé diverse funzionalità che vengono svolte in multiprogrammazione, grazie al sistema operativo GNU/Linux ed alla nuova CPU a quattro processori che forniscono funzionalità e capacità di elaborazione ottimali per l'applicazione di controllo. I sensori e gli attuatori

messi a disposizione da UDOiT permettono di realizzare una rete "mista" di controllo ambientale. Il modulo "indispensabile" per la gestione della rete, che deve essere fisicamente installato su Raspberry Pi è il "dongle" USB, visibile nella **Fig. 3**

Nel protocollo adottato dai moduli UDOiT, che descriveremo nel dettaglio nei prossimi articoli, ciascuna "rete" di sensori ed attuatori è riconosciuta da un codice "network" univoco, in modo che non possano verificarsi "intrusioni" tra moduli di reti adiacenti. Il dongle USB nasce codificato con un identificativo di "network" univoco e non modificabile. Sulla base di questo codice, utilizzando le funzionalità di "learning" previste nella applicazione di controllo, vengono "personalizzati" ed "associati" alla rete tutti i successivi moduli aggiuntivi. L'applicazione di controllo presenta un'interfaccia utente grafica fruibile da un browser web sia utilizzando un PC che un qualsiasi dispositivo mobile, smartphone, tablet, notebook, ecc. Una volta creata la rete, installati ed associati tutti i moduli desiderati, i comandi possono essere impartiti dall'interfaccia di "comando" dell'applicazione, che si configura "dinamicamente" in base ai moduli associati. Un'alternativa è di "associare" alla rete uno o più telecomandi, del tipo visibile in **Fig. 4**.

Ciascun telecomando è dotato di quattro tasti, a ciascuno dei quali, come vedremo in dettaglio nei prossimi articoli, può essere assegnato il compito di comandare uno o più attuatori ma anche di eseguire azioni (programmi) sul gateway, anche predisposti dall'utilizzatore. I diversi telecomandi possono svolgere tutti le stesse funzioni oppure, ciascuno,

**Fig. 7**

funzioni diverse, a scelta dell'utilizzatore. Le funzioni associate ai "tasti" dei telecomandi, tutte o in parte, possono essere rese disponibili anche sull'interfaccia grafica "web" di comando. Il sensore "PIR", visibile in **Fig. 5**, permette di rilevare una presenza o, più precisamente, una sorgente di calore, nell'ambiente antistante il suo angolo di visione. A fronte della rilevazione di una presenza può essere attivata un procedura di segnalazione dell'evento, per

Fig. 8

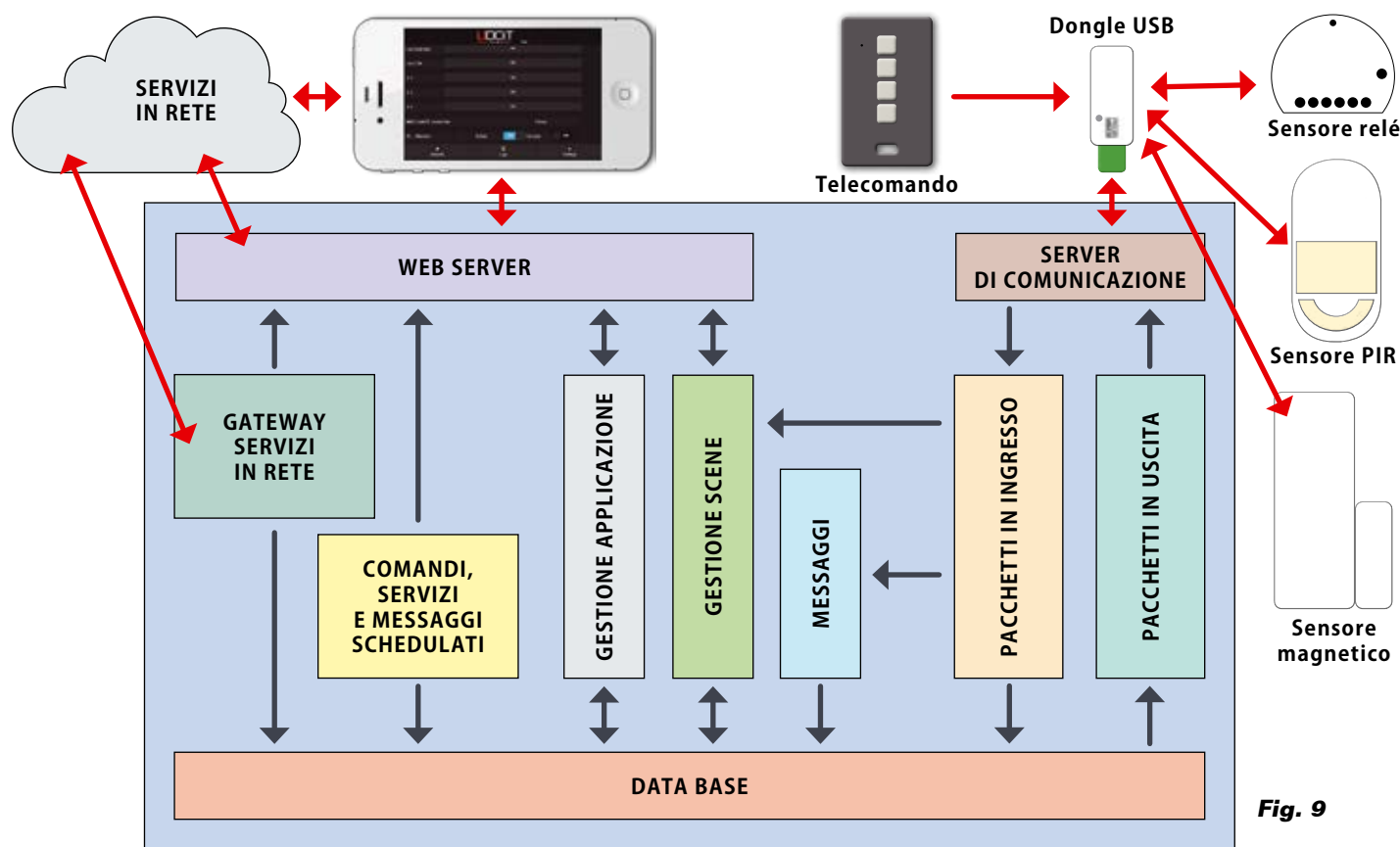
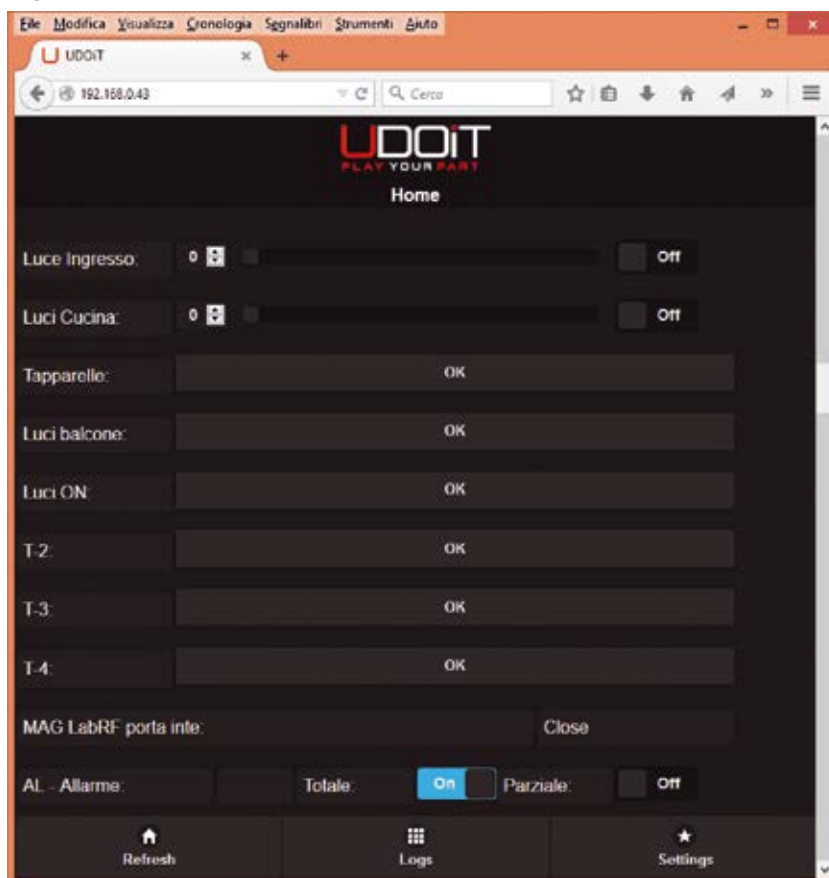


Fig. 9

Fig. 10

esempio attivando una sirena per un certo tempo ed inviando un messaggio e-mail o su un altro sistema di messaggistica, oppure attivare un'utenza, come per esempio accendere una o più lampade. Ovviamente le diverse funzioni possono essere combinate. Il sensore PIR segnala anche i tentativi di manomissione e lo stato di carica della batteria. Queste segnalazioni vengono visualizzate sull'interfaccia grafica dell'applicazione di controllo. In **Fig. 6** è visibile il sensore magnetico, sempre collegato in radio frequenza, che si utilizza per rilevare l'apertura e la chiusura di porte e finestre, ma anche di qualsiasi oggetto dotato di meccanismi di apertura come cassetti o bauli. Può essere utilizzato anche per rilevare se viene asportato un oggetto semplicemente appoggiato ad un piano. Anche questo sensore può far parte di una "sottorete" di sensori dedicati al controllo



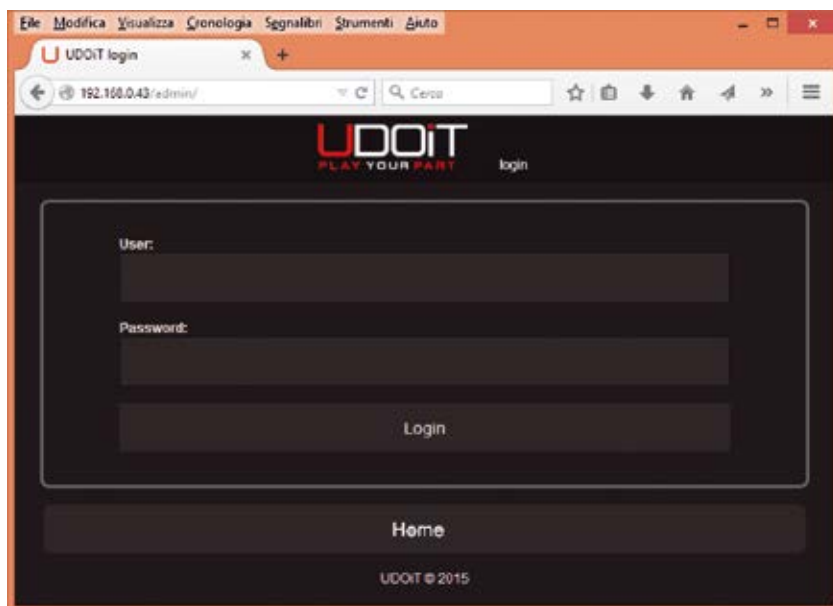


Fig. 11

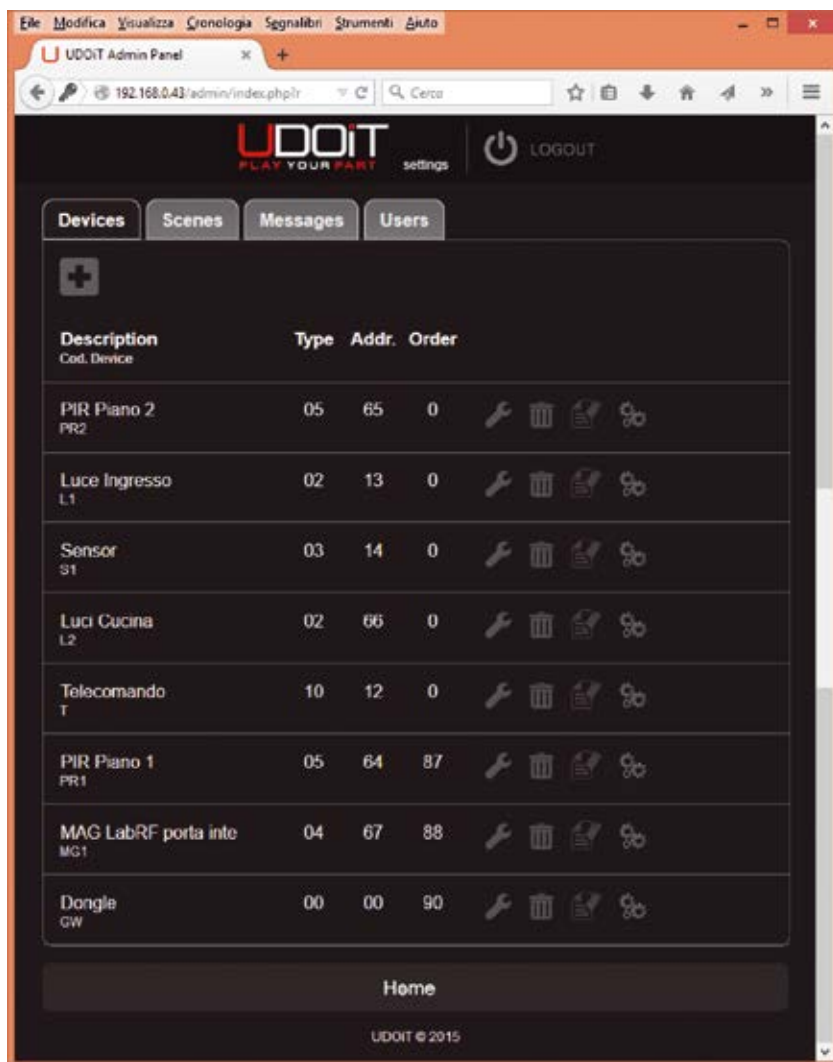


Fig. 12

ambientale, oppure utilizzato per innescare un comando verso uno o più attuatori od anche inviare messaggi o scrivere loghi di accessi, per esempio corredati da fotografie di chi o cosa ha aperto o chiuso un accesso. Anche questo sensore segnala i tentativi di manomissione e lo stato della batteria. In Fig. 7 è possibile vedere il modulo relè. In realtà il modulo relè funziona contemporaneamente sia come attuttore che come sensore. Il modulo relè si utilizza per alimentare, sia da remoto che con intervento manuale, un carico a 220V fino ad un massimo di 10 Ampere e può essere operato sia da remoto, tramite la rete senza fili e l'applicazione di gestione, sia manualmente come un comune interruttore. La funzione sensore permette al modulo di trasmettere il suo stato, in modo da informare il sistema di gestione, anche quando viene operato manualmente. Il modulo relè è stato progettato per essere alloggiato nelle comuni cassette dei frutti degli impianti elettrici tradizionali. Il modulo dimmer è in fase di sviluppo. La sua funzione è di accendere una fonte luminosa permettendone di regolare la luminosità, sia utilizzando un modulo telecomando che mediante l'interfaccia grafica dell'applicazione di controllo. Il modulo sirena è anch'esso in fase di sviluppo e permetterà di realizzare un sistema di controllo ambientale. In Fig. 8 è visibile un'ipotesi del prototipo, ma non è detto che corrisponda alla versione definitiva. Saremo più precisi nei prossimi articoli. In caso di segnalazione di intrusione il sistema può essere programmato, oltre che a segnalare l'evento sull'interfaccia grafica dell'applicazione di controllo o con un messaggio e-mail o

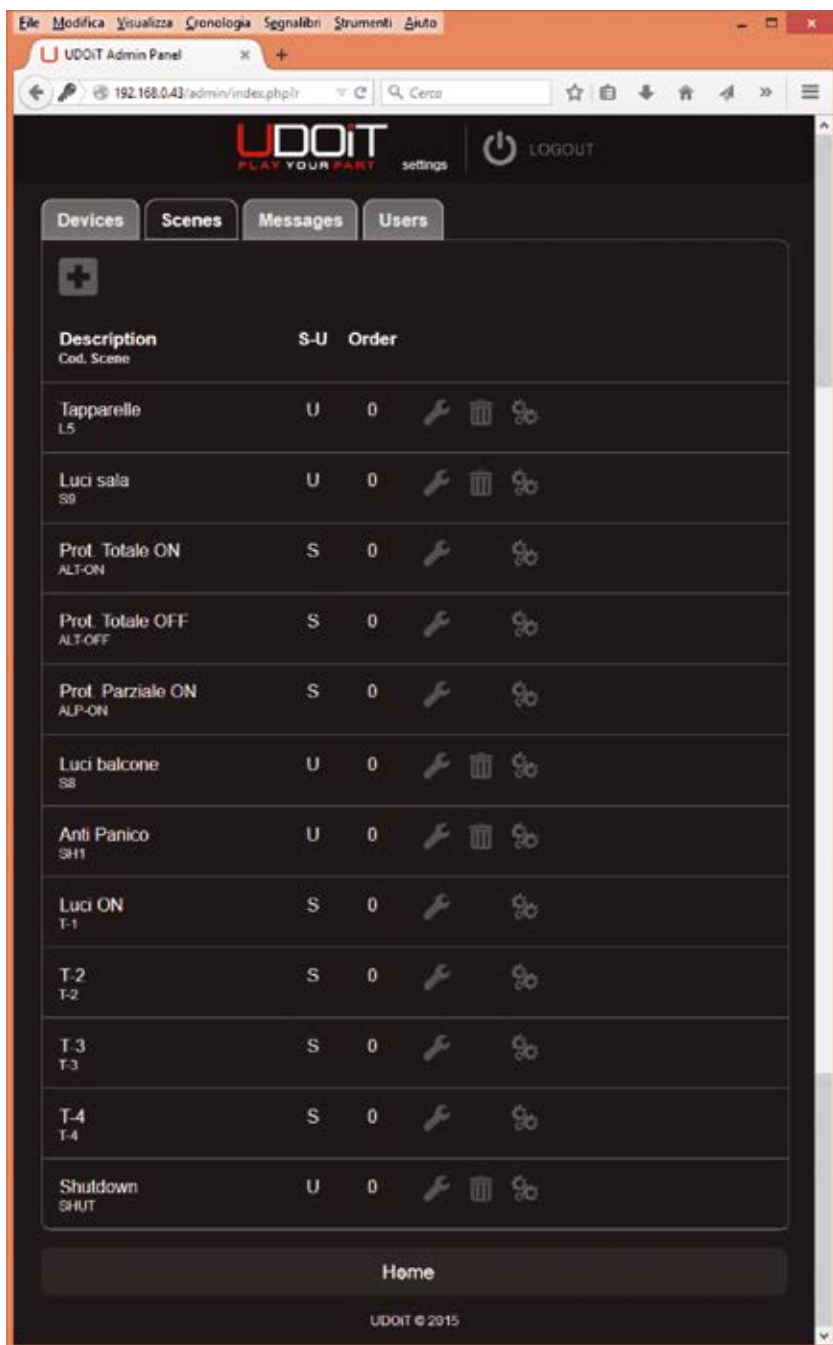


Fig. 13

cimali). I pacchetti in uscita dal sistema possono essere generati da azioni sul telecomando, sull'interfaccia grafica, da eventi del sistema, da "regole" di comportamento in base allo stato generale del sistema o da altre fonti. I pacchetti di comando in uscita vengono instradati verso il "dongle" USB, compatibilmente con la disponibilità di quest'ultimo a riceverli. Il "servizio" che gestisce i pacchetti in ingresso, analizza la provenienza ed il contenuto di informazioni di ciascun pacchetto. In base ai dati contenuti aggiorna i valori dei sensori nelle tabelle di stato ed in base alle regole di comportamento codificate nel sistema, esegue i comandi ed attiva i processi previsti. Il database di supporto all'applicazione è costituito da numerose tabelle, che approfondiremo nei prossimi articoli, che contengono le caratteristiche dei moduli fisici (sensori, attuatori e periferiche in genere), lo stato degli stessi e le regole di comportamento a fronte delle diverse situazioni che si vengono a creare. Altre tabelle raccolgono i comandi, provenienti dalle diverse origini, che devono essere immessi nella rete verso i moduli esterni e gli stati di ritorno dai sensori e dai moduli in genere.

Il comportamento del sistema può essere personalizzato con programmi e script che possono essere scritti o personalizzati direttamente dall'utilizzatore. Il "collante" del sistema è costituito dall'applicazione grafica per configurare ed operare l'intero sistema (la cui videata di "comando" è visibile in Fig. 10). Le videate di "comando" sono ad accesso libero, mentre le funzionalità di gestione dell'applicazione sono protette da user id e password (Fig. 11) e permettono di:

simile, anche con l'azionamento del modulo sirena per un certo periodo. Lo stesso modulo può essere utilizzato come segnalatore in un sistema antipanico. Nello schema di Fig. 9 è visibile la struttura logica dell'applicazione che realizza il gateway. Il modulo "server di comunicazione" è realizzato in logica "event driven", e gestisce i pacchetti in ingresso ed uscita dal dongle

USB. Il modulo di gestione dei pacchetti è composto da due sezioni principali, in pratica due server all'interno dello stesso modulo software, che gestiscono i due "rami" della comunicazione seriale, uno per la gestione dei "pacchetti" in uscita ed uno per quelli in ingresso dai moduli. Il protocollo di comunicazione è composto da pacchetti di undici byte (ventidue caratteri esade-

- Aggiungere e far riconoscere al sistema sensori, attuatori e moduli in genere, come i telecomandi, assegnando loro un nome di riconoscimento, una descrizione, un indirizzo e le modalità di comportamento (Fig. 12);
- Eseguire il "learning" di ciascun modulo periferico, in modo da "associarlo" al network di riferimento;
- Aggiungere al sistema le "scene", ovvero sequenze di operazioni preconfigurate che possono essere innescate ed eseguite a fronte di un evento

"automatico" o di un comando intenzionale dell'utilizzatore, via telecomando od interfaccia web (Fig. 13);

- Ciascun modulo associato al sistema che produce e trasmette "dati" ed informazioni, oltre ad innescare l'aggiornamento della tabella che contiene gli stati del sistema genera messaggi automatici che sono evidenziati nella relativa pagina web. È possibile personalizzare la descrizione dei messaggi e la loro posizione all'interno della pagina dedicata. È anche possibile

inserire messaggi personalizzati che vengono alimentati da programmi scritti dall'utilizzatore (Fig. 14);

- Le pagine web di controllo e di configurazione vengono composte automaticamente in base ai sensori alle scene ed ai messaggi configurati. È possibile personalizzare ulteriormente l'interfaccia web nascondendo eventuali elementi indesiderati ed ordinando i rimanenti nella sequenza desiderata (Fig. 10 e Fig. 15);
- In caso di "attivazione" di un sensore o di esecuzione di una scena è possibile comandare l'esecuzione di un processo (script) preparato dall'utilizzatore. Questa possibilità rende il sistema estremamente flessibile ed adattabile ad un gran numero di esigenze pratiche. È possibile inviare mail o messaggi su ciascun tipo di social network, alimentare database e sistemi remoti in genere, inviare sms gestire messaggi vocali, alimentare file di log, scattare foto ed archivarle od inviarle ad uno o più destinatari. Il limite all'espandibilità ed alla personalizzazione è dato dalla fantasia e dalle funzionalità disponibili nel sistema operativo di riferimento;
- Un sottosistema già preconfigurato, permette di allestire un sottoinsieme di sensori (o anche tutti) come facenti parte di un sistema di controllo ambientale che, anche se non è possibile definirlo come un vero e proprio antifurto, una volta attivato permette di comandare attuatori o processi di avviso a fronte di intrusioni non desiderate nella zona "sorvegliata" dai sensori. È possibile distinguere ed attivare in alternativa un

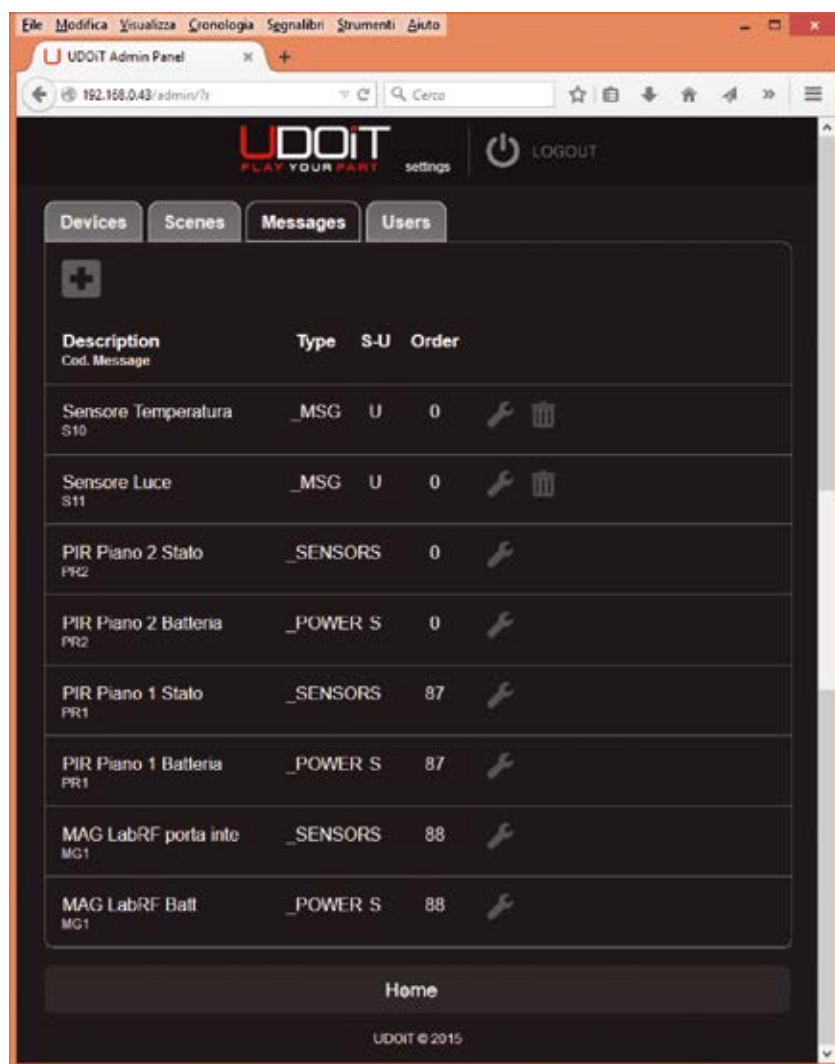


Fig. 14

Fig. 15



The screenshot shows a web browser window with the address bar displaying '192.168.0.43/message.php'. The page title is 'UDOiT' with the tagline 'PLAY YOUR PART'. Below the title is a section titled 'Logs' containing a table of sensor data. At the bottom of the page are three buttons: 'Refresh', 'Home', and 'Settings'.

Sensore	Valore	Data e Ora
Sensore Luce:	220 Lux	11-05-2015 23:50:
Sensore Temperatura:	26,5 C	11-05-2015 23:50:
PIR Piano 1 Batteria:	Low	11-05-2015 23:54:
PIR Piano 1 Stato:	OK	11-05-2015 23:53:
MAG LabRF Batt:	OK	10-05-2015 15:16:
PIR Piano 2 Stato:	OK	11-05-2015 23:52:
PIR Piano 2 Batteria:	OK	11-05-2015 23:53:
MAG LabRF porta inte:	Tamper Mano.	10-05-2015 15:15:

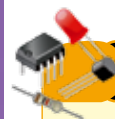
sistema di supervision parziale o periferico oppure globale;

- Esiste ovviamente la possibilità di integrare l'applicazione con programmi da eseguire a tempo od a scadenza, lanciati dallo schedatore "cron" di GNU/Linux. In questo modo è possibile, tra le mille altre possibilità, realizzare un impianto di irrigazione in grado di modificare il suo intervento in base ai dati acquisiti dai sensori di temperatura, umidità e pioggia se non addirittura dopo aver consultato le previsioni meteo;
- La possibilità di integrare e di adattare il comportamento dell'applicazione anche grazie

ad informazioni provenienti da fonti remote, come sensori controllati da sistemi long range, sms o via rete internet, cablata o wifi ne rendono possibile l'utilizzo per il controllo e la gestione anche di ambienti diversi dalle mura domestiche, pensiamo a campi coltivati, parcheggi, centraline meteo collocate in luoghi poco accessibili, ecc.

Nei prossimi articoli descriveremo i singoli moduli, sensori, attuatori, dongle e telecomandi che compongono la famiglia UDOiT, ciascuno con le funzionalità che lo caratterizzano ed il corrispondente protocollo di comunicazione.

ne. Descriveremo le funzionalità dell'applicazione e le modalità di installazione e configurazione su Raspberry Pi o altri microcomputer. Infine descriveremo l'utilizzo dell'interfaccia web dell'applicazione che permette di personalizzare la rete di sensori e degli altri moduli ed il loro utilizzo pratico. Sempre mediante l'interfaccia web è possibile definire scene di comportamento con le diverse possibilità di attivazione. Così come è possibile espandere le funzionalità dell'applicazione stessa mediante il meccanismo degli script e come integrare le funzionalità dell'applicazione con altre applicazioni o sistemi già esistenti, come per esempio l'invio di mail o comunicazioni su svariati sistemi come i social network o l'integrazione di sensori diversi da quelli della famiglia UDOiT.



per il MATERIALE

I prodotti presentati in questo progetto possono essere acquistati presso Futura Elettronica. Il dongle USB (cod. RFT868USB) costa 39,00 Euro, il sensore magnetico (cod. RFT868MAG) è disponibile a 49,00 Euro, il sensore di movimento (cod. RFT868PIR) è in vendita a 55,00 Euro, il modulo a relé (cod. RFT868REL) costa 55,00 Euro e il modulo a dimmer (cod. RFT868DIM) è disponibile a Euro 55,00.

Il radiocomando a 4 canali (cod. RFT8684CH) è in vendita a Euro 39,00 Euro, e la Raspberry Pi 2 Tipo B (cod. 8326274RS) a Euro 44,00.

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



Abbiniamo l'elettrodo per il riconoscimento di gesture 3D a Raspberry Pi, per realizzare un'applicazione in cui con i gesti facciamo scorrere su uno schermo HDMI delle immagini.

di MATTEO DESTRO

GESTIC: PROVIAMOLA CON RASPBERRY PI

Nel fascicolo di giugno scorso abbiamo descritto un nuovo elettrodo GestIC pensato e realizzato per l'interfacciamento con le board Arduino Uno Rev. 3, Arduino Leonardo Rev. 3 e Raspberry Pi B+/2.0, e vi abbiamo proposto un'applicazione con

Arduino, promettendovi di proporre a breve l'abbinamento a Raspberry Pi; ebbene, è giunto il momento di vedere anche l'interfacciamento con questa scheda e analizzare l'implementazione della libreria necessaria a gestire l'integrato MGC3130. La libreria è



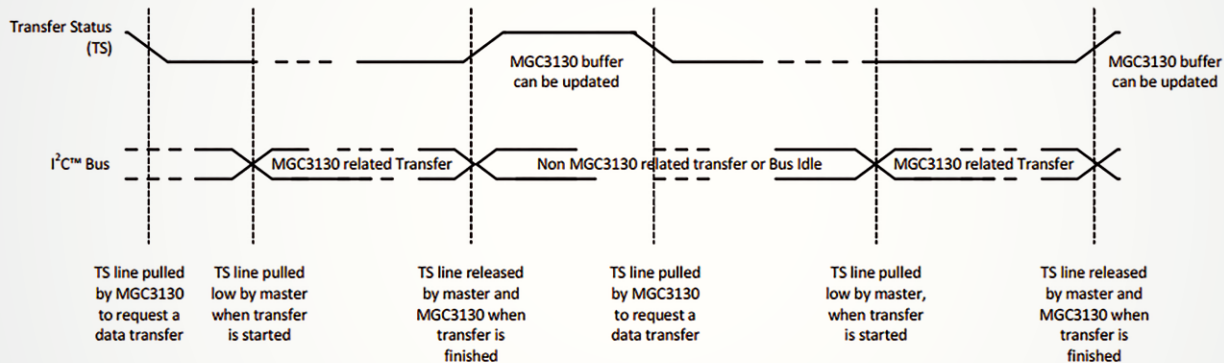


Fig. 1

realizzata in Python, ovvero un linguaggio di programmazione dinamico orientato agli oggetti utilizzabile per lo sviluppo software di qualsiasi tipologia. Inoltre permette lo sviluppo di codice di qualità e di facile manutenibilità, offre un forte supporto all'integrazione con altri linguaggi e programmi, è fornito di un'estesa libreria standard e può essere appreso in pochi giorni nelle sue funzioni di base.

BREVE PANORAMICA SULL'HARDWARE

Abbiamo già affrontato nella precedente puntata la descrizione dello schema elettrico del nostro nuovo elettrodo, quindi non ci ripetiamo (rinvandovi per gli approfondimenti del caso al fascicolo di giugno) e ci soffermeremo solo su alcuni brevi punti senza andare nei dettagli.

Come si può osservare dallo schema elettrico, della scheda Raspberry Pi vengono usati pochi pin tra cui i segnali SDA e SCL collegati direttamente all'integrato MGC3130, il quale lavora con una logica che ha lo stesso livello di tensione di quello della Raspberry Pi e quindi non necessita di adattatori di livello, diversamente da quanto invece è richiesto da Arduino. Stesso discorso per le linee di I/O, che vengono portate direttamente all'integrato e in particolare le ormai note linee TS e RESET. Brevemente ricordiamo che la linea TS serve per indicare alla Raspberry Pi che sono pronti dei dati nel buffer di MGC3130 come conseguenza del riconoscimento di una gesture. Non appena Raspberry Pi si "accorge" che la linea TS è andata a livello logico basso, provvede ad occupare tale linea configurandola come uscita e portando anch'esso la linea a livello logico basso; a questo punto può iniziare la lettura dei dati presenti nel buffer dell'MGC3130 sfruttando l'interfaccia di comunicazione I²C. Conclusa la lettura dei dati, la linea viene riportata a livello logico alto e riconfigurata come ingresso in attesa di un prossimo evento. La Fig. 1 mette in evidenza l'andamento della linea TS rispetto al bus I²C.

La linea RESET, come suggerisce il nome, serve per resettare l'integrato MGC3130 in caso di necessità. Solitamente si tende a mantenere la linea di RESET a livello logico basso durante il power-up dell'integrato e a rilasciare tale linea solo ad alimentazione stabilizzata (linea RESET riportata a livello logico alto). Infine alla scheda Raspberry Pi sono stati collegati tre pulsanti P1, P2 e P3 più un jumper J1. Solitamente per questa tipologia di componenti si prevede sempre una resistenza di pull-up da collegare sull'ingresso del microcontrollore che ne legge lo stato. Nel nostro caso si è preferito sfruttare la resistenza di pull-up interna messa a disposizione dal SOC (System On chip) presente su Raspberry Pi (Broadcom BCM2835 per Raspberry B+ e Broadcom BCM2836 per Raspberry 2). Il SOC permette di configurare gli ingressi sia con una resistenza di pull-up che di pull-down oppure nessuna delle due (non consigliato). Nella nostra applicazione, diversamente da quanto fatto per Arduino, abbiamo sfruttato solo i tre pulsanti P1, P2 e P3, tralasciando il jumper J1.

```
pi@RaspTest ~ $ find /usr | grep -i gpio
/usr/lib/pyshared/python2.7/RPi/GPIO.so
/usr/lib/pyshared/python2.6/RPi/GPIO.so
/usr/lib/python3/dist-packages/RPi/GPIO.cpython-32mu.so
/usr/lib/python3/dist-packages/RPi.GPIO-0.5.11.egg-info
/usr/lib/python2.7/dist-packages/RPi.GPIO-0.5.11.egg-info
/usr/lib/python2.7/dist-packages/RPi.GPIO-0.5.11.egg-info
/usr/lib/python2.6/dist-packages/RPi.GPIO-0.5.11.egg-info
/usr/lib/python2.6/dist-packages/RPi.GPIO-0.5.11.egg-info
/usr/share/pyshared/RPi.GPIO-0.5.11.egg-info
/usr/share/doc/python3-rpi.gpio
/usr/share/doc/python3-rpi.gpio/changelog.Debian.gz
/usr/share/doc/python3-rpi.gpio/changelog.gz
/usr/share/doc/python3-rpi.gpio/copyright
/usr/share/doc/python-rpi.gpio
/usr/share/doc/python-rpi.gpio/changelog.Debian.gz
/usr/share/doc/python-rpi.gpio/changelog.gz
/usr/share/doc/python-rpi.gpio/copyright
pi@RaspTest ~ $
```

Fig. 2

Il led LD8, anch'esso collegato a un pin di I/O, serve a indicare l'avvenuto riconoscimento di una gesture; la gestione del LED LD8 non dipende dalla libreria ma dal codice della demo.

Ricordiamo infine che con il nuovo elettrodo si è deciso di sfruttare gli I/O estesi EIO2, EIO3, EIO6 e EIO7, messi a disposizione da MGC3130, ai quali abbiamo collegato dei LED per segnalare alcune delle gesture riconosciute. Durante la parametrizzazione è quindi possibile decidere quale gesture deve essere monitorata e riportata su una delle possibili uscite o una combinazione di esse. Di seguito l'elenco:

- *Flick West to East;*
- *Flick East to West;*
- *Flick North to South;*
- *Flick South to North;*
- *Single Tap North;*
- *Single Tap South;*
- *Single Tap West;*
- *Single Tap East;*
- *Single Tap Centre;*
- *Clock Wise;*
- *Counter Clock wise.*

CONFIGURARE RASPBERRY PI

Prima di addentrarci nella descrizione della libreria Python è necessario eseguire dei passi intermedi di configurazione di Raspberry Pi. Vediamo passo passo come procedere: per prima cosa è necessario installare la libreria Python per la gestione degli I/O di Raspberry Pi e, se già installata, provvedere ad aggiornarla all'ultima release disponibile che attualmente si attesta alla 0.5.11.

Per verificare l'attuale versione della libreria GPIO eseguire il seguente comando:

```
find /usr | grep -i gpio
```

La Fig. 2 evidenzia il feedback del comando inviato; in questo caso la libreria risulta aggiornata all'ultima release. Se la release della libreria dovesse essere inferiore alla 0.5.11 provvedere ad un suo aggiornamento usando i seguenti comandi:

```
sudo apt-get update
sudo apt-get upgrade
```

oppure utilizzare i seguenti:

```
sudo apt-get install python-rpi.gpio
sudo apt-get install python3-rpi.gpio
```

Passiamo ora alla configurazione della periferica I²C messa a disposizione da Raspberry Pi; procediamo

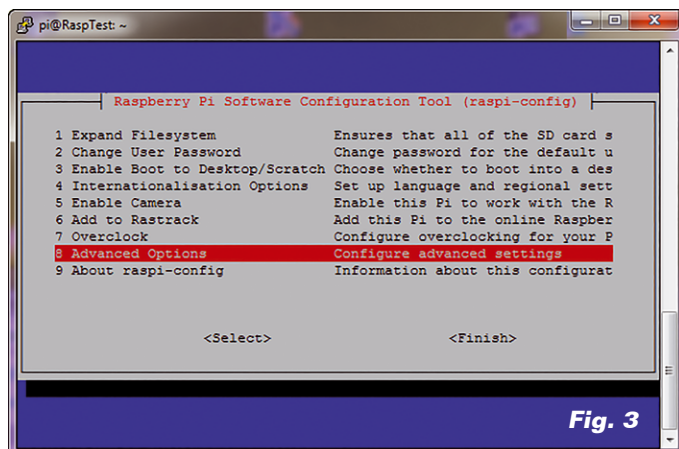


Fig. 3

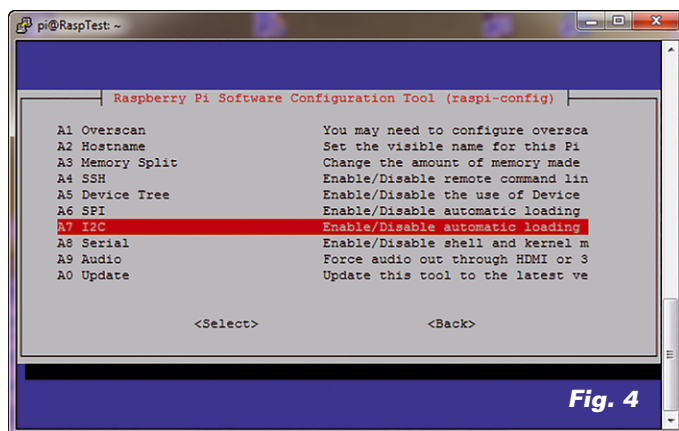


Fig. 4

con l'installazione della libreria Python "smbus" per la gestione del bus I²C:

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

seguita dai tools I²C:

```
sudo apt-get install i2c-tools
```

Dobbiamo poi abilitare il supporto del bus I²C da parte del kernel; per fare ciò è necessario richiamare la finestra di configurazione, digitando il seguente comando:

```
sudo raspi-config
```

Come visibile in Fig. 3, selezioniamo la voce "Advanced Options", quindi:

- nella nuova schermata selezioniamo la voce "A7 I²C Enable/Disable automatic loading", come in Fig. 4;
- nelle seguenti schermate dobbiamo rispondere in sequenza "Yes", "Ok", "Yes" e infine ancora "Ok";
- usciamo dalla finestra di configurazione selezionando la voce "Finish".

```

pi@RaspTest: ~
GNU nano 2.2.6 File: /etc/modules

/etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with '#' are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
i2c-bcm2708
i2c-dev

```

Fig. 5

```

pi@RaspTest: ~
pi@RaspTest ~$ sudo i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- 42 -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@RaspTest ~$

```

Fig. 6

Adesso eseguiamo il reboot del sistema eseguendo il comando:

```
sudo reboot
```

Dopo avere riavviato Raspberry Pi apriamo con un editor di testo generico il file “modules” mediante il comando:

```
sudo nano /etc/modules
```

A questo punto dobbiamo aggiungere, se mancanti, le seguenti due righe (Fig. 5):

```
i2c-bcm2708
i2c-dev
```

Per ultimo bisogna verificare se nella vostra distribuzione è presente il seguente file:

```
/etc/modprobe.d/raspi-blacklist.conf
```

In caso affermativo apritelo con un editor di testo usando il comando:

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

e, se presenti, commentate (utilizzando il carattere “#”) le seguenti istruzioni:

```
blacklist spi-bcm2708
```

```
blacklist i2c-bcm2708
```

A questo punto non rimane che riavviare nuovamente la Raspberry Pi e verificare che la comunicazione I²C sia attiva; per fare ciò eseguite il comando seguente:

```
sudo i2cdetect -y 1
```

Se tutto è andato a buon fine, il comando inviato ritorna una tabella come quella mostrata in Fig. 6. In pratica il comando analizza il bus I²C in cerca di periferiche e nel nostro caso ha individuato l’integrato MGC3130 con indirizzo 0x42.

La configurazione dell’hardware I²C, e relativa libreria di gestione, è conclusa; ora si rende necessario scaricare e installare un modulo Python che simula la pressione dei tasti della tastiera, cosicché si possa associare una gesture a un tasto, ad esempio i tasti funzione F1, F2 ecc.

Il modulo da scaricare è reperibile al seguente link:

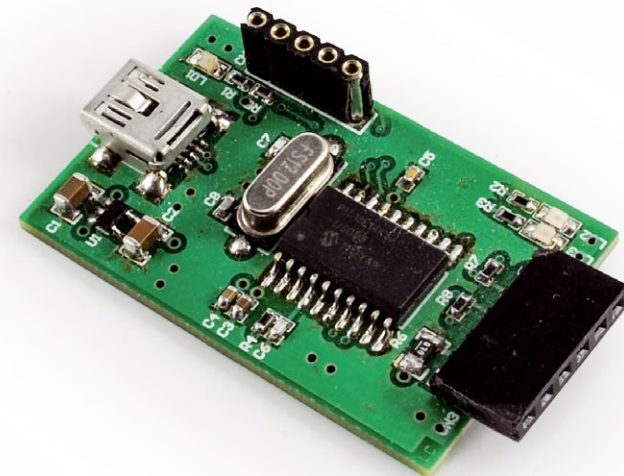
<https://github.com/SavinaRoja/PyUserInput>

Scaricate il file .zip e scompattatelo in una directory: ad esempio la “home” o, se preferite, in una directory da voi creata per l’occasione.

Tra i file scompattati ne troverete uno con nome “Setup.py” che vi servirà per installare il modulo nella vostra distribuzione e renderla disponibile nel path Python, cosicché la si possa includere nei propri progetti.

Digitate quindi il comando:

```
python setup.py install
```



Se tutte le dipendenze sono verificate, il modulo viene installato e reso subito disponibile per essere integrato nei vostri progetti. Tuttavia potrebbe succedere che non siano rispettate tutte le dipendenze e che si renda necessario installare il modulo aggiuntivo “pi3d” scaricabile dal seguente link:

<https://github.com/tipam/pi3d>

Come per il precedente scaricare, scompattare e infine installare il seguente comando:

```
python setup.py install
```

Infine potrebbe essere necessaria la libreria “xlib” per Python la quale può essere installata con il seguente comando:

```
sudo apt-get install python-xlib
```

Bene, arrivati a questo punto abbiamo tutto quello che ci serve per introdurre e studiare la libreria di gestione dell’integrato MGC3130 sviluppata in Python.

MGC3130 E LA LIBRERIA PER RASPBERRY PI

Parliamo ora della libreria scritta in Python per gestire l’integrato MGC3130, la quale si rifà sulla falsa riga di quella già presentata per Arduino. Cominciamo con una puntualizzazione sulla gestione delle strutture dati in Python necessarie a gestire il flusso dati proveniente dall’integrato MGC3130. In Python le strutture dati vengono costruite in modo leggermente differente rispetto al linguaggio di programmazione C, quindi di seguito riportiamo un confronto tra la struttura dati scritta in C per Arduino rispetto alla nuova libreria in Python per Raspberry Pi. Come vi ricorderete dalla precedente puntata, la struttura dati usata con Arduino è composta come mostrato nel **Listato 1**.

Ora mostriamo l’equivalente scritta in Python (**Listato 2**): notate che l’impostazione è differente ma la funzione svolta è la medesima.

Anche se sembrano diverse in realtà sono uguali e svolgono lo stesso compito. Nel nostro caso per accedere a questa struttura dati è necessario dichiara-

Listato 1

```
union GestureOutput {
    uint32_t Gesture;
    uint8_t GestArray[4];
    struct {
        uint8_t Byte_0;
        uint8_t Byte_1;
        uint8_t Byte_2;
        uint8_t Byte_3;
    } GestureByte;
    struct {
        uint8_t TouchSouth      :1; // GESTURE_TOUCH_SOUTH      0x00000001
        uint8_t TouchWest       :1; // GESTURE_TOUCH_WEST       0x00000002
        uint8_t TouchNorth      :1; // GESTURE_TOUCH_NORTH      0x00000004
        uint8_t TouchEast       :1; // GESTURE_TOUCH_EAST       0x00000008
        uint8_t TouchCentre     :1; // GESTURE_TOUCH_CENTRE     0x00000010
        uint8_t TapSouth        :1; // GESTURE_TAP_SOUTH        0x00000020
        uint8_t TapWest         :1; // GESTURE_TAP_WEST         0x00000040
        uint8_t TapNorth        :1; // GESTURE_TAP_NORTH        0x00000080
        uint8_t TapEast         :1; // GESTURE_TAP_EAST         0x00000100
        uint8_t TapCentre       :1; // GESTURE_TAP_CENTRE       0x00000200
        uint8_t DoubleTapSouth   :1; // GESTURE_DOUBLE_TAP_SOUTH 0x00000400
        uint8_t DoubleTapWest   :1; // GESTURE_DOUBLE_TAP_WEST  0x00000800
        uint8_t DoubleTapNorth   :1; // GESTURE_DOUBLE_TAP_NORTH 0x00001000
        uint8_t DoubleTapEast    :1; // GESTURE_DOUBLE_TAP_EAST  0x00002000
        uint8_t DoubleTapCentre  :1; // GESTURE_DOUBLE_TAP_CENTRE 0x00004000
        uint8_t GestWestEast     :1; // GESTURE_WEST_EAST        0x00008000
        uint8_t GestEastWest     :1; // GESTURE_EAST_WEST        0x00010000
        uint8_t GestSouthNorth   :1; // GESTURE_SOUTH_NORTH     0x00020000
        uint8_t GestNorthSouth   :1; // GESTURE_NORTH_SOUTH     0x00040000
        uint8_t EdgeGestWestEast :1; // GESTURE_EDGE_WEST_EAST   0x00080000
        uint8_t EdgeGestEastWest :1; // GESTURE_EDGE_EAST_WEST   0x00100000
        uint8_t EdgeGestSouthNorth :1; // GESTURE_EDGE_SOUTH_NORTH 0x00200000
        uint8_t EdgeGestNorthSouth :1; // GESTURE_EDGE_NORTH_SOUTH 0x00400000
        uint8_t GestClockWise    :1; // GESTURE_CLOCK_WISE       0x00800000
        uint8_t GestCounterClockWise :1; // GESTURE_COUNTER_CLOCK_WISE 0x01000000
        uint8_t FreeBit          :7;

    } Bit;
} GestureOutput;
```

Listato 2

```
class GestureBit (Structure):
    _fields_ = [("TouchSouth", c_uint32, 1),
                ("TouchWest", c_uint32, 1),
                ("TouchNorth", c_uint32, 1),
                ("TouchEast", c_uint32, 1),
                ("TouchCentre", c_uint32, 1),
                ("TapSouth", c_uint32, 1),
                ("TapWest", c_uint32, 1),
                ("TapNorth", c_uint32, 1),
                ("TapEast", c_uint32, 1),
                ("TapCentre", c_uint32, 1),
                ("DoubleTapSouth", c_uint32, 1),
                ("DoubleTapWest", c_uint32, 1),
                ("DoubleTapNorth", c_uint32, 1),
                ("DoubleTapEast", c_uint32, 1),
                ("DoubleTapCentre", c_uint32, 1),
                ("GestWestEast", c_uint32, 1),
                ("GestEastWest", c_uint32, 1),
                ("GestSouthNorth", c_uint32, 1),
                ("GestNorthSouth", c_uint32, 1),
                ("EdgeGestWestEast", c_uint32, 1),
                ("EdgeGestEastWest", c_uint32, 1),
                ("EdgeGestSouthNorth", c_uint32, 1),
                ("EdgeGestNorthSouth", c_uint32, 1),
                ("GestClockWise", c_uint32, 1),
                ("GestCounterClockWise", c_uint32, 1),
                ("Free", c_uint32, 7)]

class GestureByte (Structure):
    _fields_ = [("Byte0", c_uint8),
                ("Byte1", c_uint8),
                ("Byte2", c_uint8),
                ("Byte3", c_uint8)]

class Gesture (Union):
    _fields_ = [("Gesture32Bit", GestureBit),
                ("GestureByte", GestureByte),
                ("GestureLong", c_uint32),
                ("GestArray", c_ubyte * 4)]
```

rare una variabile nel file Python che richiama tale struttura, ad esempio:

```
_GestureOutput = Gesture()
```

Tale dichiarazione può essere fatta come tipo globale e quindi visibile in ogni funzione dichiarata nel file di libreria, a patto che in ogni funzione si dichiari la seguente:

```
global _GestureOutput
```

così facendo si indica all'interprete Python che la variabile è di tipo globale. Per accedere al valore, ad esempio, della gesture Tap su elettrodo Sud si deve scrivere la seguente riga di codice:

```
if (_GestureOutput.Gesture32Bit.TapSouth):
```

Se invece si vuole impostare il valore di tale bit si deve usare questa sintassi:

```
_GestureOutput.Gesture32Bit.TapSouth = 0
```

Oltre a livello di bit è possibile accedere ai dati a livello di singolo byte per un totale di quattro byte (32 bit totali di cui sette liberi per sviluppi futuri) oppure direttamente come double word agendo in un solo colpo su tutti i 32 bit della struttura. Quindi l'accesso diventerebbe:

```
_GestureOutput.GestureByte.Byte0 = 0x00
_GestureOutput.GestureLong = 0x00000000
```

Dopo questa premessa analizziamo le funzioni messe a disposizione dalla nostra libreria; in particolare, abbiamo le seguenti funzioni pubbliche:

```
def MGC3130_SetAdd(Addr)
def MGC3130_ResetDevice(GPIO, Rst):
def MGC3130_ExitResetDevice(GPIO, Rst):
def MGC3130_Begin(GPIO, Ts, Rst):
def MGC3130_GetTsLineStatus(GPIO, Ts):
def MGC3130_ReleaseTsLine(GPIO, Ts):
def MGC3130_GetEvent():
def MGC3130_DecodeGesture():
```

La funzione "MGC3130_SetAdd" serve per impostare l'indirizzo hardware occupato sul bus I²C da MGC3130, nel nostro caso è sempre e solo 0x42. L'unico parametro da passare a questa funzione è appunto l'indirizzo hardware.

La funzione "MGC3130_ResetDevice", come suggerisce il nome, serve per mantenere resettato l'integrato MGC3130 fintantoché non viene rimossa tale condizione. I parametri da passarle sono due: il primo è la libreria GPIO necessaria per gestire le linee di I/O generiche, mentre il secondo è il pin associato alla linea di RESET.

La funzione opposta è "MGC3130_ExitResetDevice" la quale rimuove la condizione di RESET; i parametri da passarle sono i medesimi della funzione precedente.

La funzione "MGC3130_Begin" serve per inizializzare le linee di comunicazione tra la Raspberry Pi e l'MGC3130. I parametri da passare alla funzione sono la libreria GPIO, il numero di pin per la linea TS e il numero di pin per la linea RESET.

Il pin TS viene impostato come ingresso con resistenza di pull-up interna abilitata, mentre la linea RESET

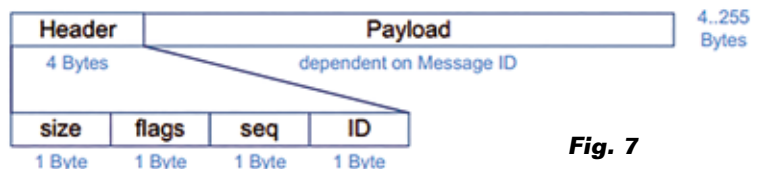


Fig. 7

Listato 3

```
GPIO.setmode(GPIO.BCM) # set board mode to Broadcom
GPIO.setup(JUMPER_J1, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Jumper
GPIO.setup(PULSE_P1, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Pulsante P1
GPIO.setup(PULSE_P2, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Pulsante P2
GPIO.setup(PULSE_P3, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Pulsante P3
GPIO.setup(LED_DIAGNOSTIC, GPIO.OUT, initial=GPIO.HIGH) # Led

GPIO.add_event_detect(PULSE_P1, GPIO.FALLING, bouncetime=100)
GPIO.add_event_detect(PULSE_P2, GPIO.FALLING, bouncetime=100)
GPIO.add_event_detect(PULSE_P3, GPIO.FALLING, bouncetime=100)
```

viene impostata come uscita con valore logico iniziale alto. L'inizializzazione prevede di mantenere in RESET forzato l'integrato per un periodo di 250mSec portandolo così alle condizioni iniziali come se fosse avvenuto un power-up.

La funzione "MGC3130_GetTsLineStatus" serve per monitorare la linea TS e intercettare quando l'integrato MGC3130 la impegna per avvisare che è stata riconosciuta una gesture. Non appena il sistema si accorge che la linea TS è stata impegnata, cambia lo stato del pin andando lui stesso ad occupare la linea TS, potendo così iniziare la procedura di lettura dei dati contenuti nel buffer dell'integrato MGC3130. Conclusa la lettura dei dati si può richiamare la funzione "MGC3130_ReleaseTsLine" rilasciando così la linea TS.

Per entrambe le funzioni appena esposte bisogna passare due parametri: il primo è la libreria GPIO, mentre il secondo è il pin cui è collegata la linea TS. La funzione "MGC3130_GetEvent" serve per leggere i dati memorizzati nel buffer dell'integrato MGC3130 e salvarli in apposite strutture dati private utilizzate dalla libreria. Sarà poi una successiva funzione a provvedere alla decodifica dei dati ricevuti e a renderli disponibili all'utente finale per la propria applicazione. Vi ricordiamo, come già ampiamente detto nelle precedenti puntate, che il pacchetto dati utilizzato per la lettura dei dati caricati nel buffer dell'integrato MGC3130 è del tipo raffigurato in Fig. 7. L'indirizzo contenuto nel pacchetto è quello hardware dell'integrato dove il bit meno significativo indica se si sta eseguendo una scrittura ("0") oppure una lettura ("1"). La sezione "MGC3130 message"

può essere espansa come indicato in Fig. 8 identificando due sezioni "Header" e "Payload". Espandendo ancora la sezione "Header", come in Fig. 9, si nota che nei primi quattro byte ritroviamo la lunghezza del pacchetto dati ricevuto e, cosa molto importante, l'ID che identifica il pacchetto dati. Nel caso di una lettura dati dopo il riconoscimento di una gesture l'ID del pacchetto sarà 0x91. Per l'elenco dei codici ID disponibili vi invitiamo a rileggere la precedente puntata.

La libreria gestisce anche l'ID 0x83 che identifica la revisione di libreria caricata nell'integrato. Questa informazione viene letta allo start-up dopo avere tolto la condizione di RESET. Attualmente c'è una limitazione nel numero di byte che si possono leggere dal bus I²C sfruttando la libreria "smbus", ovvero la lunghezza massima del buffer di ricezione della libreria è impostato a 32 byte il che è un attimino limitante. Questo comporta che il pacchetto dati con ID 0x83 letto dal buffer di MGC3130 è troncato. Per chi volesse cimentarsi è possibile scaricare il sorgente della libreria "smbus", cambiarne la lunghezza del buffer, ricompilare e re-installare la libreria nella propria distribuzione.

La funzione "MGC3130_GetEvent" applica dei filtri ai dati letti dal buffer di MGC3130 per eliminare le parti non necessarie ai nostri scopi. I dati fondamentali sono le informazioni inerenti le "GestureInfo" e le "TouchInfo" alle quali si applicano delle maschere di filtro per eliminare i bit indesiderati (la "MASK_GESTURE_RAW" viene applicata alle "GestureInfo" mentre la "MASK_TOUCH_RAW" viene applicata a "TouchInfo"). Questi valori, se non per validi motivi, non devono mai essere modificati dall'utente.

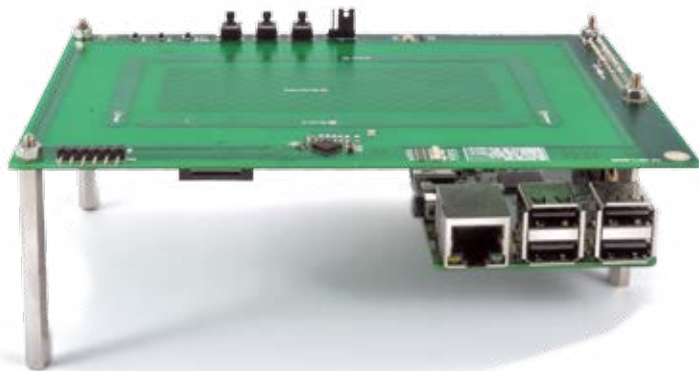
L'ultima funzione "MGC3130_DecodeGesture" decodifica i dati letti dalla precedente funzione e li ren-



Fig. 8



Fig. 9



de disponibili attraverso una struttura dati pubblica che l'utente può utilizzare nella propria applicazione (vedere la struttura dati descritta precedentemente). La libreria si accorge delle variazioni intercettate dall'integrato MGC3130 e le riporta sui rispettivi bit della struttura dati pubblica. La gesture è riconosciuta quando il bit corrispondente va a livello logico "1". Volendo, ma non è necessario, è possibile filtrare le gesture che non si vogliono utilizzare nel proprio progetto; infatti si può configurare una costante di filtro, presente nel file `MGC3130_DefVar.py`, che permette di filtrare le gesture non volute. La costante si chiama `"MASK_FILTER_GESTURE"` ed è un valore a 32 bit configurabile bit a bit. Se il bit è a "1" logico significa che si desidera filtrare la gesture viceversa la si vuole mantenere.

Per favorire lo studio della libreria ci sono delle parti di codice che possono essere attivate/disattivate attraverso delle variabili booleane appositamente dichiarate. Se la variabile è `"True"` attiva la parte di codice interessata, viceversa la disattiva. Le variabili sono le seguenti:

```
1) EnablePrintMGC3130RawFirmwareInfo = False
2) EnablePrintMGC3130RawData          = False
3) EnablePrintMGC3130Gesture          = False
4) EnablePrintMGC3130xyz              = False
```

Per impostazione predefinita, le variabili sono tutte settate al valore `"False"`; se attivate, vedrete a schermo la stampa dei dati letti dall'integrato ed eventualmente le relative decodifiche, in particolare in ordine abbiamo:

```
1) Stampa in formato RAW dei dati inerenti la revisione
   FW caricata nell'integrato
2) Stampa in formato RAW dei dati inerenti la gesture
   riconosciuta
3) Stampa della gesture riconosciuta dopo la decodifica
4) Stampa delle coordinate x, y e z
```

Ad esempio, se la variabile booleana `"EnablePrintMGC3130RawData"` è settata a `True` l'intercettazione della gesture `"Flick East to West"` genera

una tabella dati così formattata:

```
#####
Row data from MGC3130
Header: 1A081191
Payload: 1F01 | 86 | 80 | 0073 | 03100000 | 00000000
| 0000 | 0000000000000
#####
```

Per aumentarne leggibilità e interpretazione, il pacchetto dati viene diviso tra `"Header"` e `"Payload"`; in più il `"Payload"` viene suddiviso in sotto-pacchetti (vedere il datasheet della libreria GestIC).

MGC3130 DEMO

Descriviamo ora il nostro programma demo, sempre scritto in Python, che ci dà la possibilità di gestire un programma di visualizzazione delle immagini direttamente con le gesture riconosciute da MGC3130. Il programma si chiama `"Eye Of Gnome"` e può essere installato sulla propria Raspberry Pi semplicemente eseguendo il seguente comando:

```
sudo apt-get install eog
```

Analizziamo ora la demo realizzata per Raspberry Pi, il file si chiama `"DemoGestic.py"` e come per tutti i file scritti in Python in testa trova le dichiarazioni di importazione dei moduli necessari al suo corretto funzionamento, tra questi ritroviamo la nostra libreria per la gestione dell'integrato MGC3130 (`import MGC3130`), la libreria per la gestione degli I/O (`import Rpi.GPIO`), una libreria per la gestione dei Thread (`import threading`), una libreria per la gestione dei tempi (`import time`) e infine la libreria di emulazione dei pulsanti della tastiera (`from pykeyboard import PyKeyboard`).

Per semplicità di gestione del codice si dichiarano delle variabili a valore costante che identificano i pin di I/O della Raspberry Pi; in particolare, abbiamo le linee TS e RESET nonché le linee SDA e SCL, i pulsanti, il jumper e il LED di diagnostica. Tutte queste definizioni vengono utili nel momento in cui si vanno a configurare i pin di I/O da utilizzare con Raspberry Pi e tutte le volte che se ne vuole testare il valore o modificarne lo stato.

Come si può osservare nel **Listato 3**, con la funzione `"setup"` si definisce se il pin di interesse è un ingresso o una uscita e, nel caso dei pin di ingresso, si attivano anche le resistenze di pull-up interne. Inoltre per i pulsanti di ingresso P1, P2 e P3 si definisce un ulteriore parametro che introduce la gestione dell'evento pressione pulsante `"add_event_detect"` il

quale interviene sui fronti di discesa con un tempo di debouncing di 100mSec. Nel ciclo infinito del main ritroviamo quindi la gestione dei tre pulsanti che sono in attesa che si scateni l'evento sopra configurato, ad esempio:

```
if GPIO.event_detected(PULSE_P1):
```

Questa istruzione condizionale è in attesa che venga intercettato l'evento di pressione del pulsante P1 per eseguire la propria funzionalità. In particolare abbiamo definito che il pulsante P1 attiva la demo e quindi permette l'associazione di una gesture con un tasto della tastiera, il pulsante P2 disabilita quanto appena detto, mentre il pulsante P3 chiude la demo liberando le porte di I/O dalle configurazioni fatte. Prima di entrare in un ciclo "while" infinito il codice esegue una serie di configurazioni tra cui l'indirizzo hardware che occupa l'integrato MGC3130 sul bus I²C, nonché l'inizializzazione delle linee di I/O necessarie a gestire lo stesso. In questo secondo caso, alla funzione di configurazione viene passata come parametro anche la libreria GPIO; infatti questa libreria è bene che sia dichiarata in un solo punto e poi eventualmente passata alle funzioni che ne necessitano. La sezione di codice che si occupa di rilevare la variazione sulla linea TS e quindi provvedere alla lettura del buffer dell'integrato MGC3130 è la seguente:

```
if (MGC3130.MGC3130_GetTsLineStatus(GPIO, MGC3130_TS_LINE) == True):
    MGC3130.MGC3130_GetEvent()
    MGC3130.MGC3130_DecodeGesture()
    MGC3130.MGC3130_ReleaseTsLine(GPIO, MGC3130_TS_LINE)
```

Utilizzando la funzione di libreria:

```
def MGC3130_GetTsLineStatus(GPIO, Ts):
```

si testa la linea TS in attesa che vada a livello logico basso per poi impegnarla e iniziare la procedura di lettura dei dati tramite la funzione di libreria:

```
def MGC3130_GetEvent():
```

Seguono le funzioni di decodifica dei dati letti per estrapolare le gesture riconosciute e la funzione per il rilascio della linea TS:

```
def MGC3130_DecodeGesture():
def MGC3130_ReleaseTsLine(GPIO, Ts):
```

Descriviamo ora l'interazione tra le gesture riconosciute e il visualizzatore di immagini "Eye Of Gnome". Una volta avviato il software e selezionato

il direttorio con le immagini da guardare è possibile eseguire le seguenti operazioni:

- 1) sfogliare le immagini verso destra, gesture Flick West to East; simula il tasto "freccia destra";
- 2) sfogliare le immagini verso sinistra, gesture Flick East to West; simula il tasto "freccia sinistra";
- 3) ingrandire l'immagine (Zoom in), gesture Touch North Electrode; simula il tasto "+";
- 4) rimpicciolire l'immagine (Zoom out), gesture Touch South Electrode; simula il tasto "-";
- 5) visualizzare a schermo intero l'immagine, gesture Touch Centre; simula il tasto "F5";
- 6) ruotare l'immagine in senso orario, gesture Clockwise; simula la combinazione "CTRL + R";
- 7) ruotare l'immagine in senso antiorario, gesture Counter Clockwise; simula tasto "SHIFT + CTRL + R";
- 8) quando l'immagine è ingrandita è possibile spostare la stessa sullo schermo usando le gesture Flick West to East, Flick East to West, Flick North to South e Flick South to North.

Ma come fa il nostro codice a emulare i pulsanti della tastiera? Lo fa grazie alla libreria "PyUserInput". Una volta importata questa, con il comando:

```
import PyKeyboard
```

si deve creare l'oggetto:

```
KeyPressed = PyKeyboard()
```

L'oggetto creato può gestire i seguenti eventi:

- `press_key('h')`; l'esempio emula la pressione del tasto "h";
- `release_key('h')`; l'esempio emula il rilascio del tasto "h". Ad un "press_key" segue sempre obbligatoriamente un "release_key" (a meno che la situazione non richieda di mantenere premuto un tasto per un determinato lasso di tempo);
- `tap_key('h')`; l'esempio emula il tocco di un tasto della tastiera;
- `type_string('hello')`; l'esempio emula l'invio di una stringa.

Grazie a questi oggetti si possono anche simulare le combinazioni di tasti; ad esempio, per simulare "shift" + "ctrl" + "M" bisogna scrivere il codice seguente:

```
KeyPressed.press_key(KeyPressed.shift_key)
KeyPressed.press_key(KeyPressed.control_key)
KeyPressed.tap_key('M')
KeyPressed.release_key(KeyPressed.control_key)
KeyPressed.release_key(KeyPressed.shift_key)
```

Dopo questa premessa possiamo descrivere come abbiamo usato questa libreria per i nostri scopi. Ad esempio, il riconoscimento della gesture "West to East" viene associato al tasto freccia verso destra:

```
if (MGC3130._GestOutput.Gesture32Bit.GestWestEast == 1):
    MGC3130._GestOutput.Gesture32Bit.GestWestEast = 0
    GPIO.output(LED_DIAGNOSTIC, GPIO.LOW)
    BaseTimeTimer = threading.Timer(1, SlowBaseTime)
    BaseTimeTimer.start()
    KeyPressed.tap_key(KeyPressed.right_key)
```

Simulando il tasto freccia destro le immagini vengono scrollate verso destra. L'equivalente per lo scrolling delle immagini verso sinistra sarà:

```
if (MGC3130._GestOutput.Gesture32Bit.GestEastWest == 1):
    MGC3130._GestOutput.Gesture32Bit.GestEastWest = 0
    GPIO.output(LED_DIAGNOSTIC, GPIO.LOW)
    BaseTimeTimer = threading.Timer(1, SlowBaseTime)
    BaseTimeTimer.start()
    KeyPressed.tap_key(KeyPressed.left_key)
```

Quando una gesture viene riconosciuta viene fatto lampeggiare per un secondo il led LD8. Si è quindi creato un Thread che viene attivato e richiamato dopo un periodo di un secondo dopo ogni riconoscimento di una gesture. Più in dettaglio quando la gesture viene riconosciuta si accende il led LD8 sfruttando la linea di codice:

```
GPIO.output(LED_DIAGNOSTIC, GPIO.LOW)
```

Subito dopo viene attivato il Thread che si occuperà di spegnere il led LD8 dopo un secondo. Più precisamente l'attivazione del Thread viene fatta eseguendo il codice:

```
BaseTimeTimer = threading.Timer(1, SlowBaseTime)
BaseTimeTimer.start()
```

La funzione richiamata dal Thread è la seguente:

```
def SlowBaseTime():
    global BaseTimeTimer

    GPIO.output(LED_DIAGNOSTIC, GPIO.HIGH)
    BaseTimeTimer.cancel()
```

Questa funzione spegne il led LD8 e cancella il Thread richiamato cosicché sia possibile richiamarlo al prossimo evento.

Con questo abbiamo terminato la descrizione della nostra demo di gestione dell'elettrodo per MGC3130 abbinato a una scheda Raspberry Pi B+ o 2.0. Ricordatevi che per attivare l'interazione tra le gesture e la libreria che emula i tasti della tastiera dovete premere il pulsante P1, viceversa per disattivarla premere P2.

DUE PAROLE SU "EOG" E LA NOSTRA DEMO

Per avviare il visualizzatore di immagini "EOG" precedentemente installato bisogna, prima di tutto, avviare il server "X" con il comando "startx".

Dopodiché cliccare sulla voce "Menu → Graphics → Image Viewer" per aprire il programma "EOG". Selezionare la cartella contenente le foto su cui si vuole lavorare cliccando sulla voce di menu "Image → Open". Conclusa questa fase possiamo avviare la nostra demo Python utilizzando "LXTerminal" e al prompt digitare "python DemoGestic.py" per avviare la demo.

Ancora una volta vi ricordiamo che per attivare l'interazione tra la demo e il visualizzatore di immagini dovete premere sul pulsante P1, per disattivare l'interazione premere su P2 e per uscire dalla demo e liberare le porte di I/O premere su P3.

CONCLUSIONI

In questo quarto ed ultimo articolo dedicato alla tecnologia GestIC della Microchip abbiamo imparato ad utilizzare il nuovo elettrodo già descritto -sul piano hardware- nel fascicolo scorso (giugno 2015) con la scheda Raspberry Pi B+/2.0, abbiamo studiato la libreria Python di gestione dell'integrato MGC3130 e mostrato un esempio pratico di utilizzo dell'elettrodo.

Ora dopo questa rassegna di progetti sul riconoscimento delle gesture, possiamo dire di avervi dato le basi per poter sviluppare i vostri progetti integrando così il riconoscimento gestuale in una moltitudine di diverse applicazioni, come la visualizzazione di pagine di menu, di immagini, ma anche l'apertura e la chiusura di tapparelle e tende motorizzate, l'accensione e lo spegnimento di luci e via di seguito.



per il MATERIALE

Il software Aurea di Microchip può essere scaricato dal sito www.microchip.com/gestic.

Il sensore di prossimità (cod. MGC3130) è reperibile presso Futura Elettronica a Euro 9,80, così come la scheda Gesture (cod. FT1214M) che viene fornita assemblata al prezzo di Euro 58,00. Tutti i prezzi sono 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>



SUPERARE I LIMITI DI CARICA DELLE PORTE USB



Rendere più efficiente la carica di dispositivi portatili tramite USB superando i limiti di queste porte.

di BORIS LANDONI

La *Universal Serial Bus* (o USB) è l'interfaccia per computer più utilizzata al mondo. Inizialmente era solo un bus di espansione per personal computer, ma il suo utilizzo si è diffuso rapidamente a causa della sua flessibilità, delle prestazioni e della sua capacità *hot plug*. La maggior parte dei dispositivi elettronici portatili che richiedono connettività verso il PC utilizza l'USB per il trasferimento dei file. Tra questi troviamo riproduttori MP3, fotocamere digitali, cellulari e tablet. Dato che la porta *downstream* del bus USB standard può erogare corrente per almeno 500mA (USB 2.0) o 900mA (USB 3.0), tutti hanno iniziato ad utilizzare questo mezzo per ricaricare le batterie dei dispositivi connessi. Potendo superare questo limite di corrente, la ricarica potrebbe essere molto più efficiente. A tale scopo è possibile utilizzare l'Hub Controller USB2534 di Microchip con RapidCharge. Ove la corrente richiesta superi tali limiti, sia il dispositivo di ricarica che la porta di ricarica devono adottare un protocollo particolare per consentire la ricarica della batteria. Una porta del caricabatterie ha la responsabilità di fornire il corretto *handshake* segnalando al caricatore di indicare quando questo è collegato ad una porta di ricarica, nonché se è in grado di trasferire corrente oltre il limite della USB standard. La segnalazione corretta varia a seconda del dispositivo portatile. Alcuni di questi seguono i protocolli BC1.2 USB-IF, ma ci sono alcuni dispositivi che utilizzano protocolli di *handshake* proprietari, noti anche come

"modalità *legacy*", per la ricarica delle batterie. I dispositivi *legacy* supportano diverse forme di rilevamento dei caricabatterie destinati a caricatori dedicati. Alcuni di questi caricabatterie effettuano un corto circuito tra D+ e D- o li collegano attraverso una resistenza posta in serie. Per il rilevamento del caricatore, alcuni dispositivi *legacy* attivano una tensione su D+ collegando un *pull-up* per poi rilevare la tensione su D-. Se viene rilevata una tensione positiva, il dispositivo può supporre che è collegato ad un caricabatterie dedicato e non ad una porta USB standard. Altri dispositivi realizzano un *pull down* di una linea dati e, allo stesso tempo un *pull up* dell'altra.

Una volta che il dispositivo ha rilevato un caricatore per la presenza di una tensione su D-, è in grado di avviare la carica dalla connessione Vbus ad un livello di corrente che supera le specifiche USB.

Altri dispositivi *legacy* si affidano al caricatore per pilotare tensioni fisse (più di 1V) sulle linee dati D- e D+. Questi sono indicati come caricabatterie SE1. Se tali tensioni vengono rilevate dal dispositivo di carica, il sistema presuppone di essere collegato ad un caricabatterie dedicato e avvia la ricarica. Una porta USB *downstream* standard, infatti, non avrebbe mai presentato queste tensioni fisse sulle linee D+ e D-.

RILEVAMENTO DEL CARICATORE

Il dispositivo portatile è responsabile del rilevamento

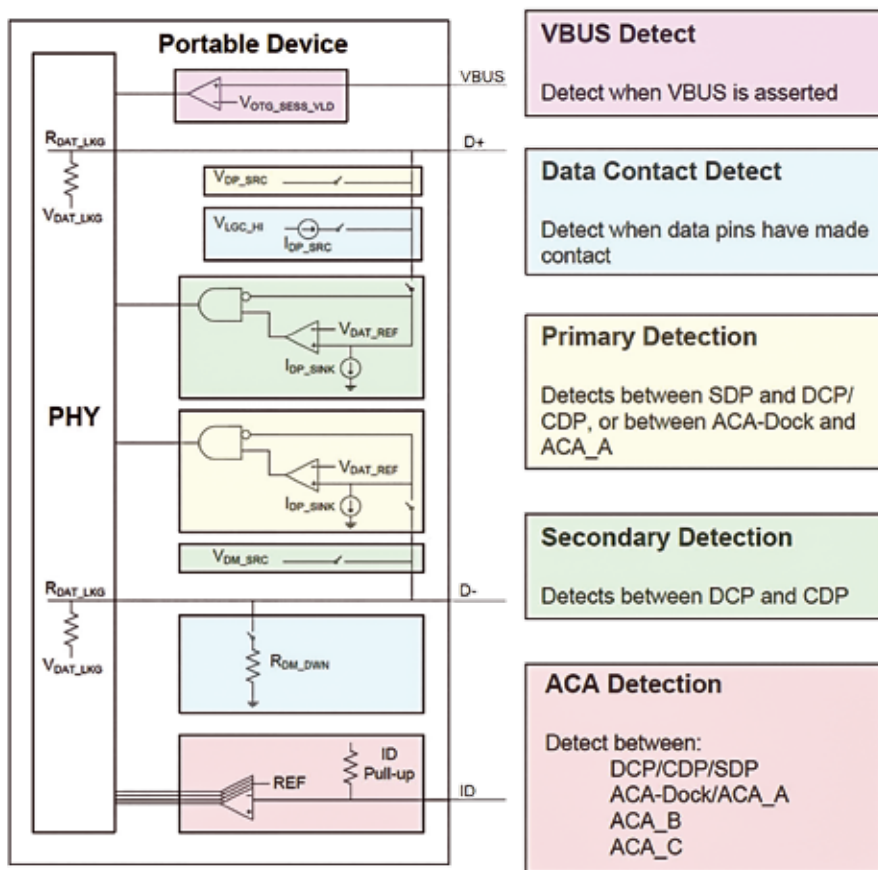


Fig. 1 - Hardware per il rilevamento del dispositivo di carica.

del caricabatterie; la Fig. 1 mostra l'hardware necessario. Ci sono cinque blocchi funzionali indicati in Fig. 1: rilevamento Vbus, data contact detect, rilevamento primario, rilevamento secondario e rilevamento ACA. Un dispositivo portatile sfrutta anche un session valid comparator. Il Vbus deve presentare un valore superiore alla tensione di soglia prima che il rilevamento del caricatore abbia inizio. Ciò è indicato sullo schema come VOTG_SESS_VLD. Il data contact detect è un blocco opzionale usato per confermare che le linee dati abbiano effettivamente fatto contatto durante il collegamento. A tale scopo, viene fornita corrente sul terminale D+ e collegata una resistenza di pull-down su D-. Se la linea D+ commuta su low significa che le linee dati sono collegate ad una porta di ricarica o a una

porta standard e la sequenza avvia il rilevamento primario. È richiesto un circuito di timeout per assicurare che il rilevamento primario parta dopo un tempo prefissato dal collegamento, nel caso in cui il contatto non venga rilevato o il data contact detect block non sia presente. È necessario anche un dispositivo portatile per implementare il rilevamento primario, utilizzato per distinguere tra una standard downstream port (SDP) e una charging port. La Fig. 2 mostra cosa accade quando il dispositivo è connesso ad una dedicated charging port (DCP), la Fig. 3 quando è connesso ad una charging downstream port (CDP) e la Fig. 4 quando è connesso ad una standard downstream port (SDP). Il rilevamento secondario viene utilizzato per distinguere tra un DCP ed un CDP. Se un dispositivo por-

tatile è pronto per l'enumerazione all'interno di un tempo prefissato dal rilevamento Vbus, questo può bypassare il rilevamento secondario, altrimenti deve implementarlo. Solo i dispositivi portatili con un connettore USB Micro-AB possono supportare il rilevamento ACA, e di conseguenza questo è facoltativo. Il rilevamento viene eseguito misurando la resistenza del pin ID.

CARICA DELLE BATTERIE

L'hub controller USB2534 di Microchip utilizza la tecnologia RapidCharge per fornire l'adeguato *handshaking* ai dispositivi portatili sulle porte downstream segnalando loro di consentire la carica delle batterie. Possiede anche la capacità di rilevamento su USB upstream del caricatore. Per caricare la maggior parte dei dispositivi portatili è necessario fornire gli adeguati segnali *handshake* per i caricatori legacy, caricatori SE1, caricatori conformi con le specifiche per caricabatterie YD/T 1591-2009 Chinese Telecommunications Industry e dispositivi conformi con BC1.2. L'hub controller raccoglie in sé tutti questi protocolli per implementare una carica delle batterie completa e in grado di supportare dispositivi Apple, Samsung e altri. Se una porta USB downstream viene configurata per fornire carica a delle batterie, la porta sarà una CDP, se questa può rilevare il dispositivo, oppure DCP in caso contrario. Se la porta non è configurata per la ricarica di batterie, la porta è una SDP. Le porte downstream possono essere attivate per la carica di batterie aggiungendo una resistenza di pull-up (10kΩ) sullo strap di configurazione del caricabatterie per la porta corrispondente. Questi strap sono rilevati al reset e, se vengono rilevati come high, la corrispondente porta verrà abilitata alla carica della batteria. La funzione di ricarica può anche

essere attivata attraverso l'uso dei registri di configurazione per la carica di batterie, che risiedono a bordo del chip USB2534. Questi registri di configurazione sono utilizzati da firmware ROM interno per configurare le funzionalità di caricabatteria su ogni porta. Gli stessi registri possono essere modificati da una configurazione programmata nella memoria *one time programmable* (OTP) utilizzando il tool di programmazione *ProTouch*.

Il ProTouch è uno strumento di sviluppo Microchip per la configurazione e la programmazione dell'hub controller USB2534. Può essere utilizzato per lo sviluppo e prototipazione dove una singola parte viene programmata, o per più parti in un ambiente produttivo.

Dove non vi fosse alcun *upstream* Vbus, e di conseguenza nessun host USB connesso alla porta *upstream*, le porte *downstream* abilitate per la carica delle batterie funzioneranno come porte DCP. Le porte abilitate come caricabatteria usciranno da questa modalità nel caso in cui la porta *upstream* abbia una connessione host. La modalità DCP verrà attivata se l'USB2534 viene sospeso e il *wake-up* remoto disabilitato. Nel *battery charging* in modalità DCP, la porta tenterà l'*handshake* ed identificherà il dispositivo *BC-capable*. Nella modalità DCP il dispositivo si avvia sempre in modalità SE1. Questa non è in grado di rilevare se un dispositivo SE1 è collegato, ma può rilevare se un dispositivo non-SE1 è collegato quando commuta su DM o DP. Non appena entra nella modalità RapidCharge, l'USB2534 passa alla modalità di carica SE1 e la porta offre i livelli di tensione SE1. Se un dispositivo SE1 viene collegato, rileverà passivamente i livelli SE1 ed inizierà la carica. Il DCP non sarà in grado di rilevare la presenza del dispositivo SE1. La porta permarrà nella modalità di carica SE1 mentre il SE1 PD carica.

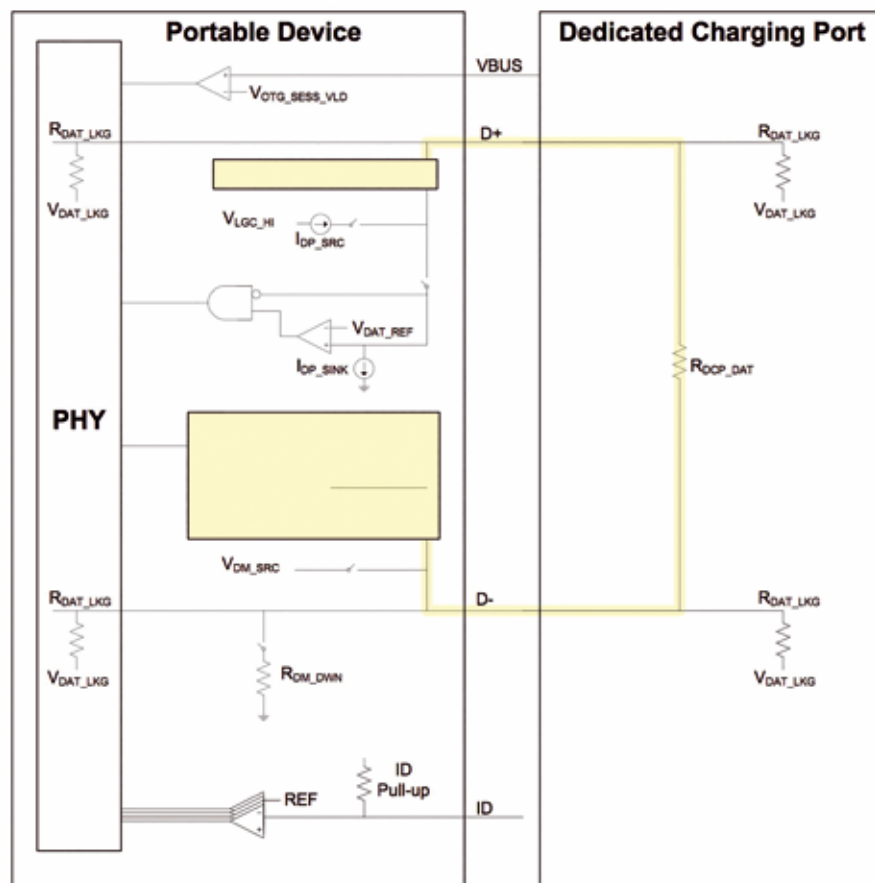


Fig. 2 - Rilevamento primario con una dedicated charging port.

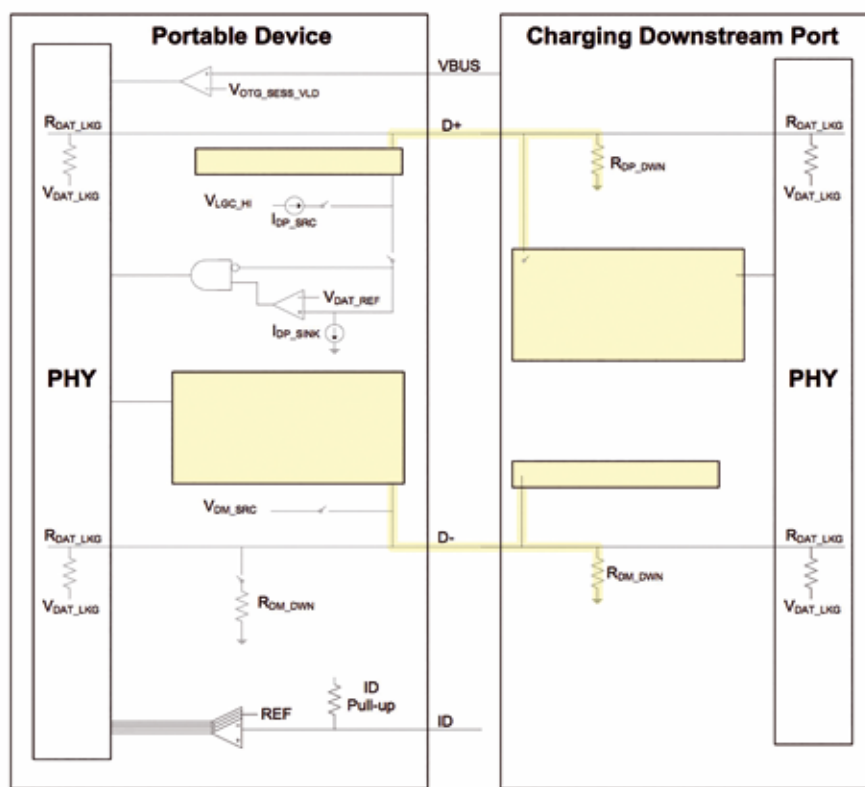


Fig. 3 - Rilevamento primario con una charging downstream port.

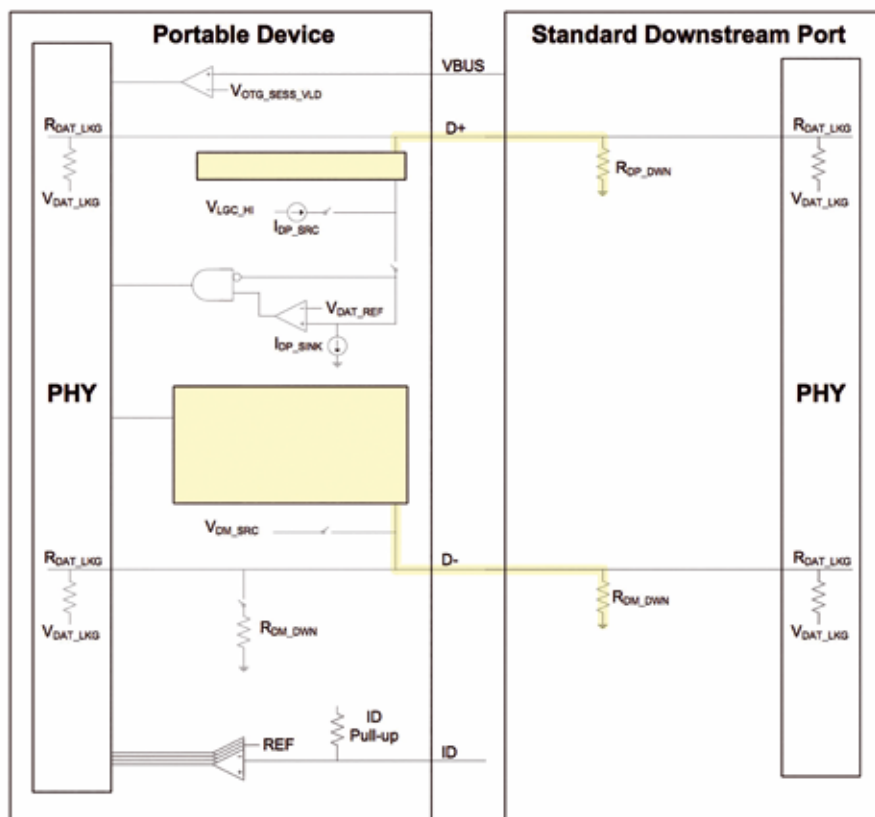


Fig. 4 - Rilevamento primario con una standard downstream port.

Se viene collegato un dispositivo BC 1.2, la sua corrente sarà abbastanza forte da portare la linea D- su *low*. Allo stesso modo, è stato notato che i dispositivi *legacy* di carica portano le linee D- su *low* quando vengono collegati. Per risolvere questo problema, la porta *downstream* passa alla modalità di carica *legacy* se lo stato della linea D- classica è rilevata come *low*. Lo stato della linea D- viene *de-bounced* per evitare rilevamenti falsati dai plug-in del dispositivo. La modalità di carica *legacy* viene mostrata in **Fig. 5**. Le porte abilitate per la carica delle batterie usciranno dalla modalità DCP per entrare in quella CDP se la porta *upstream* rileva una connessione host. Al rilevamento dell'*address command* preimpostato dall'USB host, ogni porta BC abilitata verrà spenta per almeno 250ms prima che possa essere riattivata per consentire all'alimentazione della porta di *decadere*. Se l'host invia il comando di riaccensione dell'ali-

mentazione della porta, il comando viene opportunamente ritardato. Se il comando venisse invece ricevuto dopo l'azzeramento del timer, verrebbe eseguito immediatamente. Un MCU può escludere la sequenza di rilevamento automatico del caricatore modificando i registri runtime SMBus per caricabatterie. Dato che l'avvio del rilevamento del caricatore è impostato perché si verifichi come default, l'MCU deve scrivere sul registro di controllo del carica batteria o sul registro di *configuration interlock* per disabilitare la sequenza automatica prima che la stessa sia avviata. Se la sequenza automatica è disabilitata, l'MCU può sempre avviarla manualmente. La ricarica delle batterie con USB offre un meccanismo conveniente per ricaricare batterie o dispositivi portatili quali cellulari e tablet. USB-IF ha pubblicato le *BC1.2 battery charging specification* per aiutare nella standardizzazione dei protocolli utilizzati tra caricatori e

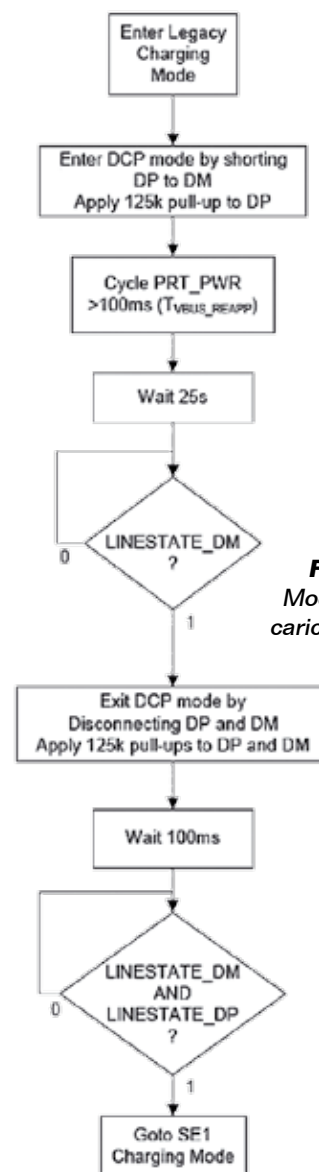


Fig. 5
Modalità di
carica legacy.

dispositivi da caricare perché il procedimento di carica sia effettuato in sicurezza. L'hub controller USB2534 di Microchip con RapidCharge fornisce protocolli per la ricarica di batterie che includono *legacy*, SE1, Chinese Telecommunications Industry YD/T 1591-2009 e USB-IF BC1.2, per implementare processi di ricarica di batterie che supportino dispositivi Apple, Samsung, e la quasi totalità degli altri. L'hub controller consente anche il rilevamento di caricabatteria per utilizzi con dispositivi portatili che richiedano capacità di rilevamento di caricabatterie USB.

Una nuova app Android comoda ed intuitiva per gestire al meglio il nostro telecomando GSM TDG133.

Nelle pagine di questa rivista viene spesso utilizzato il modulo GSM Sim900 per applicazioni di telecomando remoto; è il caso degli ormai noti telecomandi TDG133, TDG134 e TDG140. Si tratta di veri e propri apparati domotici che permettono sia di rilevare allarmi notificandoli tramite SMS, sia di controllare determinati dispositivi a distanza come elettroserrature, impianti di irrigazione, cancelli motorizzati, tramite un semplice

messaggio SMS o anche una chiamata via GSM. Per i telecomandi succitati abbiamo sviluppato un'app Android che ci permetterà di controllare i nostri apparati remoti e di ricevere i relativi allarmi tramite un'interfaccia grafica che faciliterà l'intera gestione di questi comodi dispositivi. Questa app si chiama TDGRemote per TDG133, è scaricabile gratuitamente dallo store di Google, ed è stata scritta basandoci sul telecomando TDG133 che gestisce 2 ingressi e 2 uscite.



TDGRemote PER TDG133

di ANDREA CHIAPPORI



Fig. 1
Schermata
settings.

Il codice è scritto in maniera modulare, quindi nel caso avessimo a disposizione altri telecomandi che gestiscono più I/O basterà eseguire le modifiche per adattarlo alle proprie esigenze, anche perchè i comandi aggiuntivi sono molto simili e il codice dell'applicazione è disponibile online, quindi è possibile implementare nuove funzioni a piacimento.

L'idea di sviluppare un'app nasce perché i nostri telecomandi si comandano usando SMS, i quali, dovendo essere corti, costringono a utilizzare una sintassi che può apparire criptica e difficile da ricordare. Utilizzando l'app non dovremo più preoccuparci di memorizzare i particolari comandi, ma il tutto sarà più intuitivo tramite l'interfaccia grafica. Per attivare un'uscita del telecomando, ad esempio, sarà sufficiente premere il relativo pulsante senza preoccuparci di digitare il comando corretto, perchè a tutto penserà il codice associato. Allo stesso tempo questa app eseguirà anche un controllo sulla stringa da inviare in modo che sia sintatticamente corretta, così da evitare di sprecare SMS, qualora, per dimenticanza o errori di battitura, questa non sia giusta. Anche l'inserimento di parametri di configurazione che andrebbero fuori dai margini consentiti viene segnalato da finestre di errore in modo da evitare un erroneo funzionamento dell'apparato.

PREPARAZIONE DEL TELECONTROLLO

Prima di provare l'app, assicuriamoci che nel nostro telecomando sia presente la scheda SIM e che non sia protetta dal codice di sblocco, altrimenti non potrà rispondere ai nostri

messaggi.

Una volta alimentata la scheda, il LED verde lampeggerà ogni secondo durante la ricerca della rete GSM, per poi cambiare frequenza lampeggiando una volta ogni 3 secondi a segnalare il corretto funzionamento. A questo punto possiamo utilizzare la nostra app, ma non prima di averla configurata correttamente. Analizziamo ora le varie schermate dell'app per evidenziare come gestire il nostro telecomando TDG133; ci servirà anche per un breve ripasso sui vari comandi a disposizione e per sfruttare al meglio questi dispositivi di telecomando.

SCHEMATA DI CONFIGURAZIONE

Prima di analizzare la schermata principale, occorre una breve descrizione della pagine di configurazione, in cui, dopo la prima installazione, dovremo impostare necessariamente alcuni campi. Per accedere a tale schermata basterà premere il tasto Menu di Android e selezionare l'icona *Settings*; verrà visualizzata una schermata come quella visualizzata per intero in **Fig.1**. Ovviamente potremo accedere in qualsiasi momento a tale schermata per modificare alcune di queste impostazioni. Nella prima casella, ossia **Remote Device Name**, dobbiamo scrivere il nome del nostro dispositivo remoto in modo da poterlo distinguere da eventuali altri in uso.

Remote Sim Number sarà il vero e proprio numero di destinazione, ovvero il numero della SIM alloggiata nel TDG133. Per aggiornare subito questo campo basterà premere il pulsante sottostante (*Set*).

La casella **password** deve essere modificata qualora si voglia modificare la password del telecomando e comunicarla

al TDG premendo il pulsante *Set* sulla destra; la pressione di *Set* invierà il comando al dispositivo tramite un SMS e successivamente riceveremo un messaggio di avvenuto cambiamento. La password predefinita è 12345.

Sempre nel riquadro riguardante la password, la checkbox **Request always password** servirà per costringere o meno l'utente a digitare ogni volta tale password prima di inviare il messaggio di comando, aumentando la sicurezza dell'applicazione ed evitando attivazioni erronee di qualche uscita.

La casella **Startup Message** viene usata per gestire il messaggio che invierà il telecomando al nostro smartphone ad ogni accensione e può essere disabilitata togliendo la spunta da **Use Startup Message**; la pressione del pulsante *Set* sottostante richiederà l'invio di un messaggio di comando. La casella **Forward no-std sms** serve per decidere a quale numero di telefono inviare i messaggi che vengono ricevuti dal telecomando, ma che non hanno una sintassi standard, ovvero non sono veri e propri comandi ma SMS come, ad esempio, avvisi del gestore di telefonia per aggiornamenti sul credito residuo. Questo numero di telefono sarà scelto (tramite una combobox) tra una lista di numeri presenti nella memoria del telecomando che viene gestita anch'essa dalla nostra app tramite una schermata (*WhiteList*) che vedremo in seguito. I due pulsanti sottostanti servono per richiedere l'aggiornamento di questa lista (*Request Update*) ed inviare le informazioni al telecomando (*Set*) sotto forma di messaggi SMS.

La casella **Option Misc.** presenta due checkbox che non richiedono invio di SMS, ma che rappresentano opzioni utilizzate per uso interno; in particolare,

con **Disable Multi Answer** si impedisce al telecomando di inviare più messaggi di risposta nel caso di invio multiplo, mentre spuntando l'opzione **Confirm before send message**, prima di ogni invio di un comando apparirà una finestra di conferma; in questo modo si eviterà che per errore o distrazione vengano inviati più messaggi dei necessari, con conseguente spreco di denaro. La casella **Get Quality Signal** presenta un pulsante col quale sarà possibile richiedere la qualità del segnale d'antenna del nostro telecomando (richiede sempre l'invio di un messaggio SMS); dopo la ricezione del messaggio, l'informazione verrà segnalata tramite la relativa progressbar e nella parte sottostante comparirà la data e l'ora dell'ultima richiesta effettuata.

La casella **Factory Reset** contiene il pulsante che invierà un messaggio di comando col quale riporteremo il nostro telecomando alle impostazioni originali.

Infine la casella **Version** permette di visualizzare, dopo la ricezione del messaggio di risposta, il nome e la versione del firmware del telecomando. Nel caso della **Fig. 1** è *TDG133 V 2.8*.

Chiude la schermata, il pulsante per salvare tutte queste impostazioni e tornare alla schermata principale.

SCHERMATA PRINCIPALE

Dalla schermata principale in **Fig. 2** possiamo tenere sotto controllo lo stato degli ingressi IN1 e IN2 (nella parte superiore) tramite il colore dei LED circolari e quello delle uscite OUT1 e OUT2 (nella parte inferiore) tramite i relativi pulsanti. In particolare, il LED sarà di colore verde quando il rispettivo ingresso è attivo e rimarrà grigio



Fig. 2
Schermata principale



Fig. 3
Schermata configurazione ingressi.

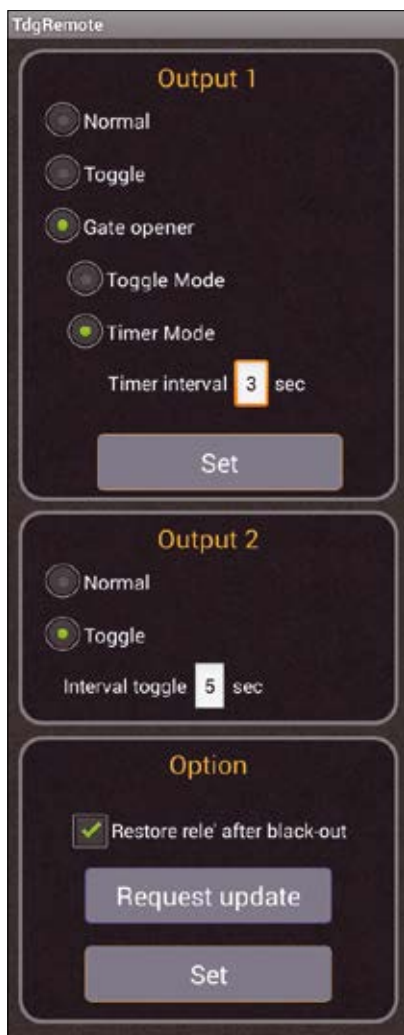


Fig. 4 - Schermata configurazione uscite.

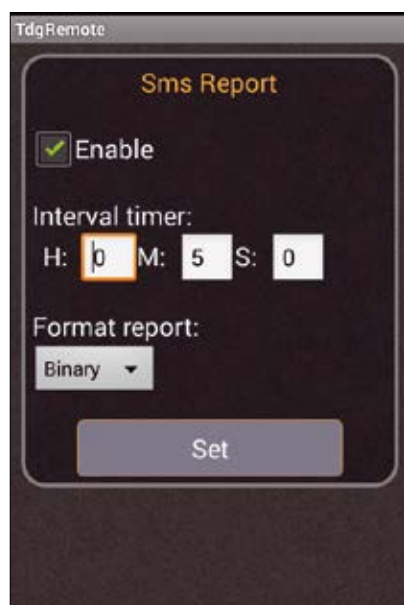


Fig. 5 - Configurazione Report.

in caso di inattività. Per quanto riguarda le uscite, le scritte *on* o *off* sui pulsanti indicheranno lo stato attuale dei relé sul telecomando.

Siccome alla prima installazione non sapremo quale sarà lo stato attuale del telecomando, sarà necessario inviare un comando di richiesta *update* premendo sulla stringa sottostante alle caselle nella sezione **Update I/O Status**; questo invierà un messaggio in seguito al quale il nostro telecomando risponderà con una stringa formattata in modo particolare e che l'app analizzerà aggiornando i controlli e tutte le configurazioni correnti.

Per verificare la validità delle informazioni che avremo in questa schermata, nella parte sottostante compariranno la data e l'ora in cui è stata fatta l'ultima richiesta di aggiornamento.

CONFIGURAZIONE INGRESSI

Ogni singolo ingresso può essere configurato, dalla pagina principale, premendo il pulsante sotto al relativo LED, allorché comparirà una schermata come quella in Fig. 3. In essa, la casella **Alarm in level** permetterà di scegliere se attivare l'allarme selezionato sul livello (alto o basso) o sul cambiamento del segnale in ingresso. La casella

Listato 1

```
public boolean callNumber(String simToCall)
{
    try
    {
        Intent callIntent = new Intent(Intent.ACTION_CALL);
        callIntent.setData(Uri.parse("tel:" + simToCall));
        callIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

        _context.startActivity(callIntent);
        return true;
    }

    catch (ActivityNotFoundException e)
    {
        Log.e("GSM Call Example", "Call failed", e);
        return false;
    }
}

...

public boolean sendMessage(String message)
{
    String messageInfo = "sending command to " + _simToCall + ": " + message;
    Log.v(TAG, messageInfo);

    Toast msg = Toast.makeText(_context, messageInfo, Toast.LENGTH_LONG);
    msg.show();

    if (!dbgMode)
    {
        SmsManager smsManager = SmsManager.getDefault();
        smsManager.sendTextMessage(_simToCall, null, message, null, null);
    }

    return true;
}
```

Listato 2

```
public class IncomingSms extends BroadcastReceiver
{
    public static final SmsManager sms = SmsManager.getDefault();
    public boolean parseMessage(String message, String callingNumber)
    {
        ...

        return true;
    }

    public void onReceive(Context context, Intent intent)
    {
        final Bundle bundle = intent.getExtras();

        try
        {
            if (bundle != null)
            {
                final Object[] pdusObj = (Object[]) bundle.get("pdus");
                for (int i = 0; i < pdusObj.length; i++)
                {
                    SmsMessage currentMessage = SmsMessage.createFromPdu((byte[]) pdusObj[i]);
                    String phoneNumber = currentMessage.getDisplayOriginatingAddress();

                    String remoteNumber = TdgRemoteActivity._remoteSimNumber;

                    String senderNum = phoneNumber;
                    String message = currentMessage.getDisplayMessageBody();

                    Log.i("SmsReceiver", "senderNum: " + senderNum + "; message: " + message);

                    parseMessage(message, senderNum);
                }
            }
        }
        catch (Exception e)
        {
            Log.e("SmsReceiver", "Exception smsReceiver" +e);
        }
    }
}
```

sottostante, **Reset inhibition time**, ci permette di impostare un periodo di tempo, successivo all'evento che ha causato l'allarme durante il quale un ulteriore allarme non verrà preso in considerazione. Questa funzione può essere anche ignorata (ricezione continua di messaggi ad ogni allarme) disattivando la checkbox corrispondente. La casella **Observation time pre-alarm** permette di definire

un intervallo di tempo in cui la condizione che fa scattare l'allarme deve rimanere valida. Questo permette di prevenire l'invio di messaggi di falso allarme dovuti a disturbi elettrici all'ingresso corrispondente. Infine la casella **Enable alarm message** permette di scegliere il testo nel messaggio che riceveremo sul nostro smartphone quando l'ingresso corrispondente sarà alto o basso. Dalla stessa casella è possibile

definire il numero massimo di messaggi che verranno inviati tramite la casella **Max number of SMS-Alarms**. L'intervallo di invio tra un SMS e l'altro dipenderà dal tempo di inibizione (*inhibition time*) definito precedentemente. Con il pulsante *Set* presente in ciascuna sezione della schermata invieremo un messaggio di comando per impostare la nostra scelta. Anche in questa schermata è presente la possibilità di

Listato 3

```
<manifest>

<uses-permission android:name="android.permission.SEND_SMS"></uses-permission>
<uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
<uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>
<uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>

<uses-permission android:name="android.permission.RECEIVE_SMS"></uses-permission>
<uses-permission android:name="android.permission.READ_SMS" />

<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <activity
        android:label="@string/app_name"

        android:screenOrientation="portrait"
        android:name=".TdgRemoteActivity" >
        <intent-filter >
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name=".IncomingSms">
        <intent-filter android:priority="1000">
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>

    ....

</application>

</manifest>
```

richiedere un aggiornamento dal telecontrollo sullo stato di queste configurazioni, premendo l'icona in basso a sinistra; riceveremo un messaggio particolare e comparirà la data e l'ora dell'ultimo aggiornamento.

CONFIGURAZIONE USCITE

Dalla schermata principale, premendo sull'icona o sulla scritta **Output settings** si accederà alla configurazione delle uscite. Questa volta possiamo configurare OUT1 e OUT2 dalla stessa schermata visibile in **Fig. 4**.

Possiamo notare come entrambe le uscite abbiano due possibilità di configurazione *Normal* e *Toggle*; con la prima, ad ogni pressione del relativo pulsante attiveremo o meno l'uscita collegata, mentre con la seconda verrà invertito lo stato attuale per un intervallo di tempo definibile

nell'apposita casella: è, pertanto, equivalente a generare un impulso (alto o basso) di durata impostabile.

Inoltre questo tipo di settaggio non necessita di invio del messaggio di comando. Come si può vedere dalla **Fig. 4**, per quanto riguarda l'uscita OUT1 è prevista anche la modalità apricancello **Gate opener**, grazie alla quale verrà associata al pulsante OUT1 una chiamata telefonica che attiverà il relé per un intervallo di tempo impostabile dall'apposita casella numerica. Se invece volessimo un'attivazione dell'apricancello in modalità bistabile (senza nessun intervallo di attivazione, ma solo apertura/chiusura) basterà selezionare la modalità *Toggle Mode*. Per quanto riguarda la modalità apricancello, occorrerà inviare il messaggio di comando tramite il tasto *Set*

qualora si voglia cambiare tale configurazione.

Va precisato che l'impostazione apricancello consentirà ad un massimo di 200 persone di poter aprire e chiudere un cancello motorizzato semplicemente effettuando una chiamata telefonica al numero della SIM telecontrollo, senza necessariamente aver installato questa app sullo smartphone. Invece l'app permette di leggere o scrivere la lista di questi 200 numeri abilitati nella memoria interna del telecontrollo, come vedremo tra poco.

In base alle scelte fatte sulla configurazione delle uscite, nella schermata principale appariranno alcune scritte sotto i pulsanti relativi alle uscite che ci permetteranno di capire la modalità di funzionamento.

CONFIGURAZIONE REPORT

Per impostazione predefinita questa funzione non è abilitata, ma il telecontrollo TDG ha anche la possibilità di monitorare costantemente lo stato di ingressi e uscite tramite un report personalizzato e configurabile tramite la schermata **Report**, accessibile tramite menu dalla schermata principale. Questa schermata è visualizzata in **Fig.5**, dove possiamo notare come sia possibile definire l'intervallo tra un messaggio e l'altro (ora:minuti:secondi) ed il formato del messaggio (binario o testuale). Anche in questo caso, per attivare le modifiche sarà necessario premere il tasto *Set*, che invierà il messaggio di comando.

GESTIONE LISTE

Nella memoria interna del telecontrollo possono essere memorizzate due liste particolari. Nella nostra app abbiamo

chiamato **White List** la lista di un massimo di 8 numeri di telefono cui verranno inviati messaggi di testo o chiamate vocali ogni qual volta venga rilevata la condizione che ha fatto scattare l'allarme. Tramite questa lista, visualizzata in **Fig. 6**, è possibile decidere, spuntando le apposite caselle **SMS** e **CALL**, se al singolo numero di telefono verranno inviati solo SMS, solo chiamate o entrambi. Inoltre i numeri in questa lista hanno la possibilità di inviare comandi al telecontrollo senza necessità di inserire la password. Il discorso è riferito a telefoni senza l'app installata, perché usando l'app, tale password verrà inserita automaticamente.

Prima di aggiungere un nuovo numero alla lista è consigliabile eseguire preventivamente una lettura dalla memoria del telecontrollo tramite la pressione del pulsante *Read from Device* posizionato in fondo alla schermata, che invierà un messaggio di comando e che permetterà di avere visione della configurazione corrente. A questo punto, tenendo premuto per due secondi il dito su una casella di testo vuota, si aprirà la rubrica dello smartphone e potremo selezionare il numero di telefono desiderato che andrà ad aggiungersi alla lista. Una volta completata, sarà possibile aggiornare la memoria interna del telecontrollo premendo il pulsante in basso *Send selecteds to Device*.

Il telecontrollo TDG gestisce anche la lista di tutti i numeri (massimo 200) che sono in grado di azionare l'apricancello eseguendo una chiamata al numero della SIM. Per accedere a questa lista occorre premere il pulsante *List* dal menu di

android e successivamente il tab *Gate List* situato a destra di quello per la *White List*; si presenterà un'interfaccia come quella mostrata in **Fig. 7**, dalla quale potremo aggiungere un nuovo numero tramite pulsante *Add* o cancellarne uno (o tutti) selezionando la relativa checkbox e premento sull'opportuno pulsante *Delete*.

IL CODICE

Terminata questa carrellata sulle funzioni della nostra app, ci limitiamo ad una breve analisi del codice che la realizza; esaminarlo tutto sarebbe troppo dispersivo, anche perché molti elementi di programmazione sono già stati esaminati nel corso Android di questa rivista ed altri meriterebbero una trattazione a parte.

È bene, però, spendere due parole su alcune funzioni che permettono di gestire la telefonia di Android a run-time. Nel **Listato 1** sono presenti due metodi definiti nella classe *telephony.java* che permettono, in modo molto semplice, di comporre una chiamata verso il numero *simToCall(callNumber)* e di inviare un messaggio di testo (*sendMessage*). Allo stesso modo, restando in ambito telefonia, possiamo gestire un SMS in ingresso e analizzarlo a seconda delle esigenze, tramite la classe *IncomingSMS* presente, seppure non totalmente, nel **Listato 2**. È una classe che deriva da *BroadcastReceiver* e che nel metodo *onReceive()* ricava, tramite brevi passaggi, il corpo del messaggio ed il numero di telefono del mittente.

Ovviamente non dovremo dimenticarci di aggiungere i relativi permessi nel file *AndroidManifest.xml* assieme alla registrazione della classe



Fig. 6 - White List.

IncomingSms nel *receiver*, come appaiono nel **Listato 3**. Il codice completo è comunque disponibile per una visione più dettagliata, ma anche per apportare le proprie modifiche o implementare nuove caratteristiche, sul nostro sito www.elettronica.in insieme agli altri file di questo articolo. ■

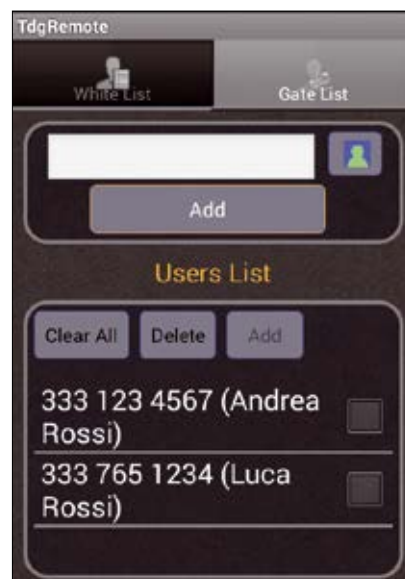


Fig. 7 - Lista apri-cancello.

CORSO

MPLAB X

8

di FRANCESCO FICILI
e VINCENZO GERMANO

Continuiamo il nostro viaggio alla scoperta di MPLab X, il nuovo ambiente di sviluppo integrato prodotto e distribuito da Microchip per sostituire l'MPLab IDE. In queste pagine realizziamo applicazioni -come il Web Server- che impiegano lo stack TCP/IP di Microchip. Ottava puntata.

Nelle ultime due puntate di questo corso abbiamo focalizzato la nostra attenzione sugli stack forniti a supporto degli sviluppatori che intendono creare applicazioni con i PIC32 e con la toolchain di sviluppo open source di Microchip; in particolare, abbiamo analizzato lo stack USB, che si compone di una parte device e di una parte host. In questa ottava puntata analizziamo un altro potente e complesso stack open source di Microchip: lo stack TCP/IP. Come esempio pratico realizzeremo un semplice Web Server embedded.

LO STACK TCP/IP DI MICROCHIP

Nel pacchetto Microchip Libraries for Applications (MLA), insieme allo stack USB già utilizzato e ad altri stack e librerie non trattate in questo corso, viene

fornito un completo stack TCP/IP integrabile in progetti che fanno uso di microcontrollori Microchip (PIC18, PIC24/dsPIC e PIC32). Lo stack supporta diverse configurazioni hardware, permettendo di utilizzare sia i moduli interni presenti nei PIC18 e PIC32, sia i controller esterni ENC28J60 (10Base-T) ed ENC624J600 (10/100Base-T) Microchip. La Fig. 1 illustra le soluzioni HW supportate dallo stack, evidenziando le caratteristiche di ognuna. In questa puntata ci occuperemo della soluzione rappresentata in alto a destra nello schema di Fig. 1, ossia quella che sfrutta il controller interno ai PIC32 con MAC integrato (in Fig. 2 è riportato lo schema a blocchi di un PIC32, dove è evidenziato il modulo Ethernet interno, collegato a due canali DMA). In questa configurazione non è previsto il livello fisico all'interno

Ethernet Portfolio

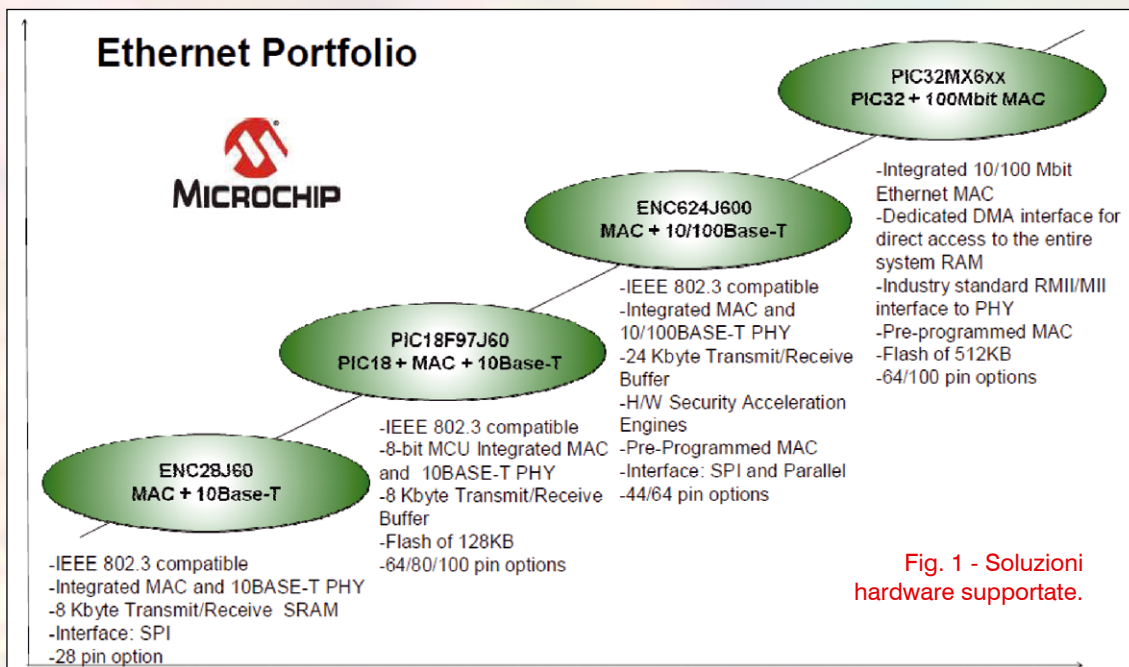


Fig. 1 - Soluzioni hardware supportate.

del micro, che dovrà quindi essere aggiunto esternamente (ad esempio, nella demoboard presentata a supporto del corso è stato utilizzato il chip National Semiconductor DP83848C, connesso al PIC32MX795F512L tramite lo standard RMII).

ARCHITETTURA DELLO STACK TCP/IP

L'architettura dello stack TCP/IP di Microchip segue un approccio a layer, come la teoria implementativa di uno stack di comunicazione suggeri-

sce: il software è diviso in strati sovrapposti l'uno all'altro e ogni strato utilizza servizi provenienti dagli strati sottostanti e fornisce servizi agli strati superiori. Un modello semplificato dello stack TCP/IP è riportato in Fig. 3. Naturalmente, siccome l'implementazione dello stack non è pensata per un ambiente desktop, dotato di risorse virtualmente infinite e di una capacità di calcolo elevata, l'implementazione Microchip racchiude un subset dei vari livelli che possono comporre lo stack. In Fig. 4 è riportata l'implementazione Microchip, comparata con il modello di riferimento del protocollo TCP/IP. L'implementazione dello stack segue il paradigma del multitasking cooperativo e può quindi essere integrata in qualsiasi altro sistema che sfrutti il medesimo approccio, come ad esempio nello scheduler che abbiamo realizzato a supporto del corso (fascicolo n° 194) e che abbiamo già utilizzato per l'integrazione dello stack USB. Come si può vedere dallo schema, l'implementazione dello stack è modulare e ogni modulo implementa un particolare livello o porzione di livello. Nella Tabella 1 sono riportati i principali moduli che compongono lo stack con la relativa descrizione e la lista dei file necessari per includere il modulo nel progetto. Chi volesse approfondire i dettagli dello stack TCP/IP open source di Microchip può leggere un'interessantissima application note sull'argomento: l'AN833 - The Microchip TCP/IP Stack.

CONFIGURAZIONE DELLO STACK

Come abbiamo visto in precedenza, lo stack TCP/IP di Microchip è composto da diversi moduli.

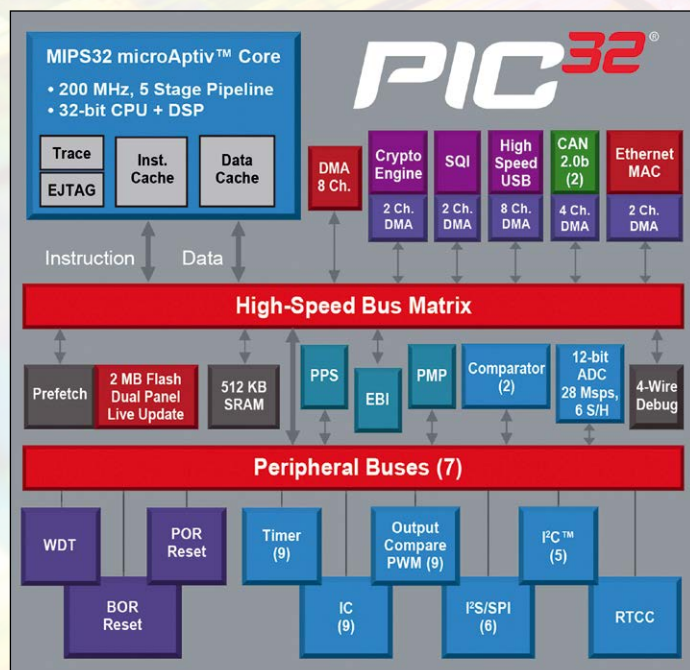


Fig. 2 - Schema a blocchi dell'architettura hardware interna di un PIC32.

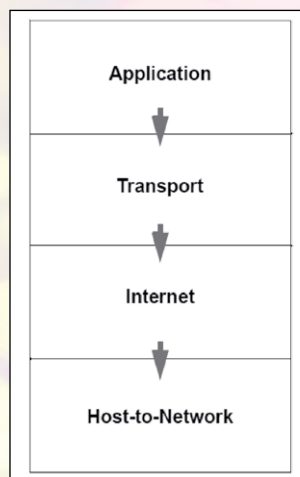


Fig. 3 - Modello semplificato dello stack TCP/IP.

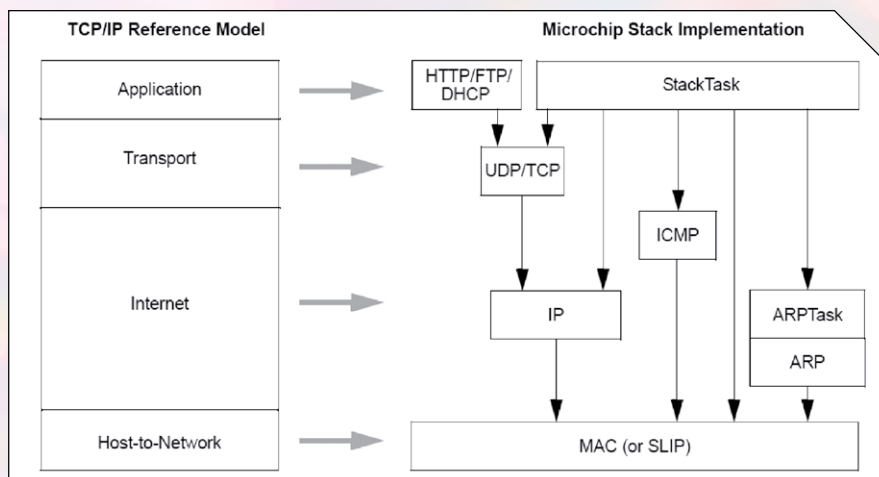


Fig. 4 - Comparazione tra l'implementazione Microchip e il modello di riferimento per il protocollo TCP/IP.

Ogni modulo, inoltre, può essere opportunamente configurato in base all'applicazione specifica e tale configurazione risiede in un particolare file, chiamato *TCPIPConfig.h*, che deve essere incluso nel progetto. L'impostazione delle configurazioni del file *TCPIPConfig.h* può essere un'operazione piuttosto complessa, ecco perché la Microchip ha realizzato un'applicazione, che prende il nome di *TCPIPConfig.exe*, che ci permette di configurare

in maniera semplice questo file attraverso un'interfaccia grafica, esattamente come avevamo fatto per lo stack USB. Vediamo quindi come funziona, utilizzandola per configurare il file *TCPIPConfig.h* dell'esempio pratico proposto in questa puntata. Per prima cosa lanciamo l'applicazione, che, una volta avviata, ci presenterà la schermata di Fig. 5, dove è possibile scegliere il percorso del file *TCPIPConfig.h* da modificare. Nel caso

IL LINGUAGGIO HTML

Il linguaggio HTML, acronimo di HyperTextMarkup Language, è il linguaggio base utilizzato per la realizzazione delle pagine web. È un linguaggio open source, la cui sintassi è stabilita dal World Wide Web Consortium (W3C), e che è derivato da un altro linguaggio avente scopi più generici, l'SGML. È basato sull'utilizzo di Tags che permettono di formattare il testo nel modo desiderato. Non può essere considerato un linguaggio di programmazione perché non possono essere definite alcune variabili e cicli di controllo ma unicamente un linguaggio di formattazione testuale, ed è generalmente indicato come *linguaggio di markup*. Per ovviare a questa limitazione è però possibile inserire all'interno della pagina dei contenuti esterni come immagini, video o script in altri linguaggi di programmazione come ad esempio il JavaScript. I tags sono racchiusi tra parentesi angolari ed ogni volta che un tag viene aperto deve poi anche essere chiuso inserendo prima del nome del tag, la barra. Ad esempio il per inserire il testo "Testo formattato" in grassetto dovremo scrivere:

```
<b>Testo Formattato</b>
```

La struttura base di un file Html è raffigurata in Fig. A. Partendo dalla testa del file, si ha una descrizione

del documento, alcune informazioni di servizio ed il corpo del documento nel quale viene formattato l'intero contenuto della pagina.

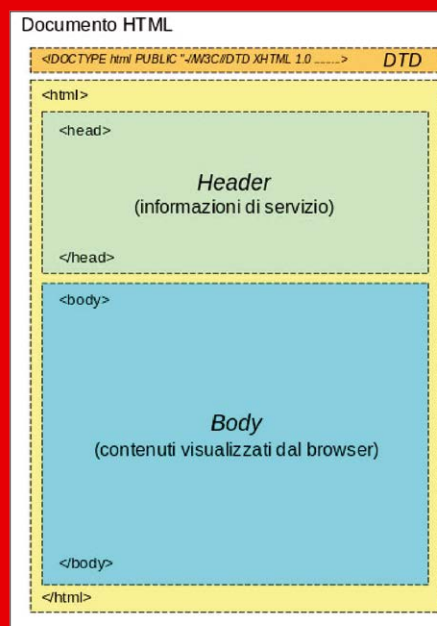


Fig. A - Schermata iniziale del tool TCPIPConfig.

Modulo	File Necessari	Descrizione
Stack Manager	StackTsk.c TCP.c IP.c ICMP.c ARPTsk.c ARP.c MAC.c or SLIP.c Tick.c Helpers.c	Stack Manager ("StackTask"), che coordina l'esecuzione di tutti gli altri moduli dello stack.
MAC	MAC.c Delay.c	Media Access Layer.
SLIP	SLIP.c	Media Access Layer per il protocollo SLIP.
ARP	ARP.c ARPTsk.c MAC.c or SLIP.c Helpers.c	Address Resolution Protocol.
IP	IP.c MAC.c or SLIP.c Helpers.c	Internet Protocol.
ICMP	ICMP.c StackTsk.c IP.c MAC.c or SLIP.c Helpers.c	Internet Control Message Protocol.
TCP	StackTsk.c TCP.c IP.c MAC.c or SLIP.c Helpers.c Tick.c	Transmission Control Protocol.
UDP	StackTsk.c UDP.c IP.c MAC.c or SLIP.c Helpers.c	User Datagram Protocol.
HTTP Server	HTTP.c TCP.c IP.c MAC.c or SLIP.c Helpers.c Tick.c MPFS.c	HyperText Transfer Protocol Server.
FPT Server	FTP.c TCP.c IP.c MAC.c or SLIP.c	File Transfer Protocol Server.
DHCP	DHCP.c UDP.c IP.c MAC.c Helpers.c Tick.c	Dynamic Host Configuration Protocol.

Tabella 1 - Elenco dei moduli che compongono lo stack TCP/IP della Microchip.

si utilizzi anche il WiFi, si deve inserire anche il percorso del file *WF_Config.h*, che contiene le opzioni di configurazione del modulo WiFi. Nella schermata successiva, visibile in **Fig. 6**, è possibile selezionare i moduli base da includere nell'applicazione. Nel nostro caso vogliamo realizzare un Web Server, quindi spuntiamo semplicemente la casella "Web Sever", sotto "Application Protocols" e deseleggiamo il resto. Andando ancora avanti troveremo la schermata che ci permette di

selezionare alcuni moduli di esempio. Nel nostro caso non ci interessa averne, quindi deseleggiamo tutto e procediamo.

La schermata successiva è riportata in **Fig. 7**, e ci permette di selezionare i moduli di supporto. Selezioniamo i seguenti moduli:

- DHCP Client e Server;
- ICMP Client;
- DNS Client;
- NetBIOS Name Server;
- AutoIP Client.

e andiamo avanti. Nella schermata successiva possiamo impostare un po' di indirizzi della scheda. Lasciamo pure inalterati il MAC e l'indirizzo IP, come anche la SubNet Mask, l'indirizzo del gateway e dei server DNS, mentre invece modifichiamo il nome dell'Host, usando quello che preferiamo (nell'esempio abbiamo usato il nome "EINBOARD"). Si consideri che, avendo incluso il modulo DHCP, i valori inseriti per IP Address, Subnet Mask, Gateway Address, e DNS Server, sono solo valori predefiniti, dato che poi ci penserà il modulo DHCP ad inserire dei valori dinamici quando collegheremo la scheda. A questo punto bisogna configurare il Web Server, operazione che possiamo eseguire nelle successive due schermate. Per prima cosa, nella schermata di **Fig. 9**, inseriamo il nome della pagina di default, che è quella che conterrà la schermata iniziale del nostro web server. Poi inseriamo il numero massimo di socket simultanei che possono essere mantenuti (nel nostro esempio 2). Nella sezione "Additional Features" e "Demo Applications" deseleggiamo tutto. Nella schermata successiva (**Fig. 10**) bisogna solo impostare dove verrà memorizzata la pagina (o le pagine) web del Web Server. Ci sono diverse opzioni; in questo caso scegliamo la più semplice, ossia "Internal Program Memory". Con questa opzione la pagina web viene caricata direttamente nella memoria Flash del PIC32 che stiamo utilizzando, e vedremo dopo come questa operazione viene eseguita. Non sempre è possibile seguire questa strada, dato che se le pagine web sono piuttosto pesanti, la Flash del microcontrollore potrebbe non riuscire a contenerle; in questo caso ci sono altre possibilità, come usare memorie EEPROM o Flash esterne o addirittura un supporto FAT (come ad esempio una SD card o una memoria USB). Nella schermata di **Fig. 11** è mostrata la configurazione dei socket TCP; potete lasciare i valori predefiniti,

ma se volete modificare qualcosa consultate -per saperne di più- la sezione relativa al modulo TCP sulla Application Note 833. Una volta completata anche la configurazione dei socket, verrà visualizzata la schermata che permette di completare la configurazione.

MICROCHIP FILE SYSTEM (MPFS)

Come abbiamo visto in precedenza, l'implementazione di Microchip dello stack TCP/IP prevede anche l'implementazione di un Web Server HTTP. Tuttavia, se viene implementato un Web Server ci saranno sicuramente una o più pagine web da visualizzare, che devono essere immagazzinate in una qualche memoria. Nell'im-

mentazione Microchip del Web Server embedded viene utilizzato un semplice tipo di file system (Microchip File System o MPFS) per immagazzinare le pagine web da pubblicare. Come abbiamo già visto prima, tali pagine possono essere memorizzate nella memoria Flash del micro stesso oppure su un supporto esterno (EEPROM esterna, SPI Flash o anche una memoria SD o USB esterna). Il formato di un'immagine MPFS è riportato in Fig. 12: dopo un blocco iniziale riservato, la cui dimensione è fissata dalla macro MPFS_RESERVE_BLOCK, l'immagine contiene una serie di FAT (File Allocation Table) entry, seguite da altrettanti blocchi di dati. Ogni FAT entry ha il formato riportato in Fig. 13. Come si

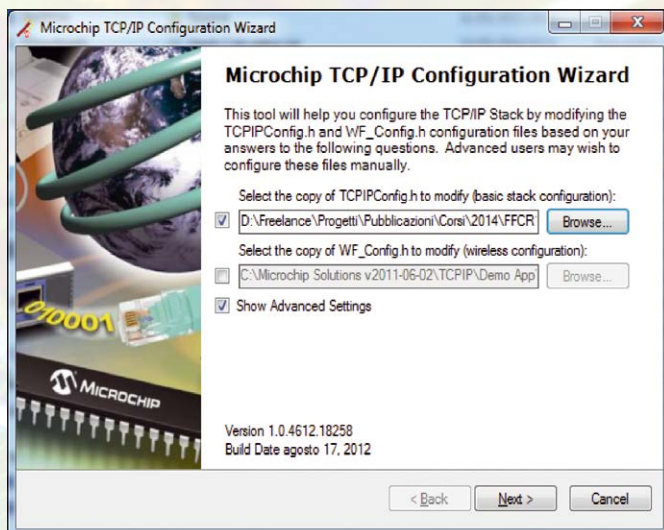


Fig. 5 - Schermata iniziale del tool TCPIPConfig.

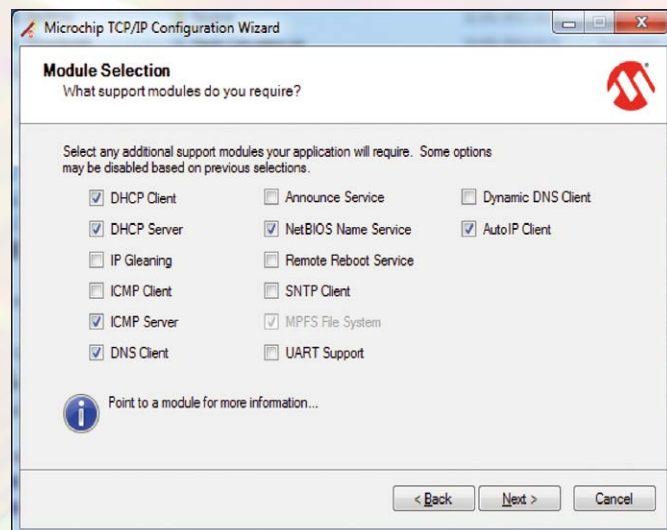


Fig. 7 - Selezione dei moduli di supporto da includere nell'applicazione.

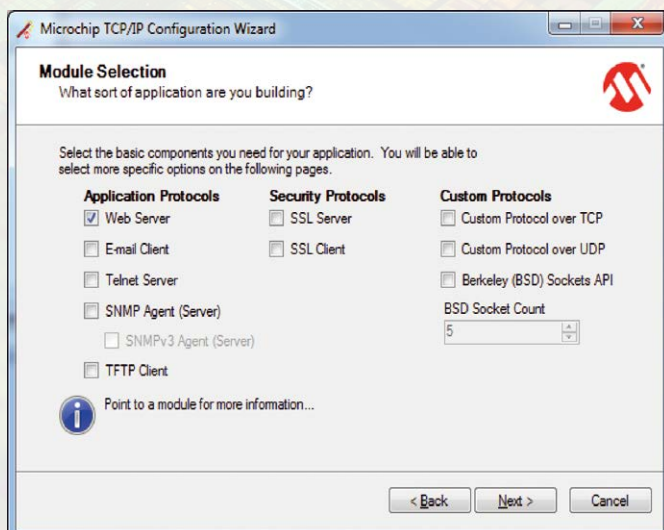


Fig. 6 - Selezione dei moduli base da includere nell'applicazione.

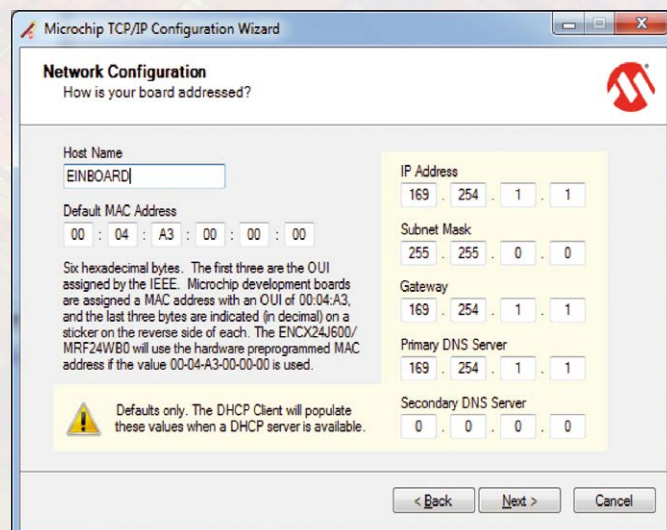


Fig. 8 - Impostazione degli indirizzi.

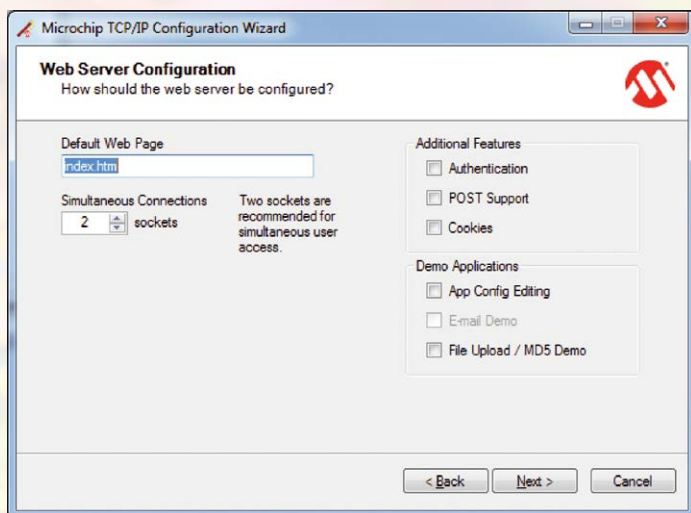


Fig. 9 - Configurazione del WebServer.

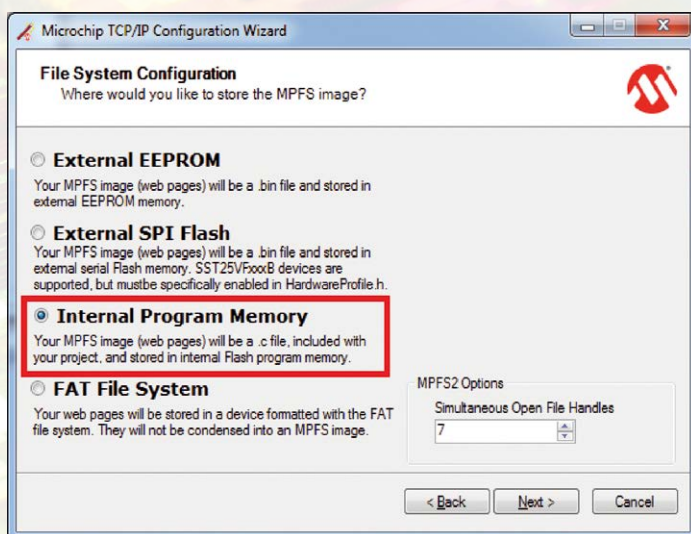


Fig. 10 - Opzioni di store delle pagine web del WebServer.

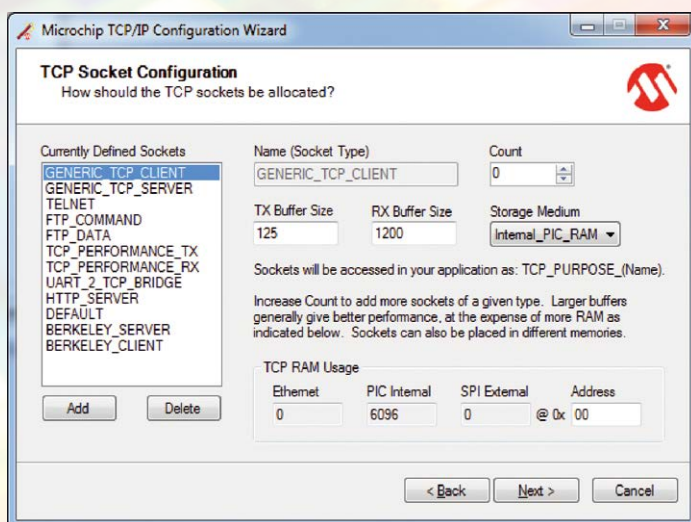


Fig. 11 - Configurazione dei sockets.

può facilmente notare, l'insieme delle FAT entry costituisce una tabella che associa un indirizzo al nome del blocco dati. L'indirizzo può essere a 16 o 24-bit in funzione del supporto utilizzato per la memorizzazione. Ad esempio, se le pagine vengono immagazzinate nella memoria Flash e per compilare un progetto è utilizzato il modello di memoria SMALL, allora gli indirizzi sono a 16-bit. Per quanto riguarda -invece- il file name, è utilizzato il nome di tipo "short", nel formato "8 + 3". L'indirizzo di ogni FAT entry punta al relativo blocco che contiene i dati binari della pagina, il cui formato è riportato in Fig. 14. La generazione di un'immagine di tipo MPFS è indispensabile per la realizzazione di un'applicazione che utilizzi il Web Server (quindi come quella del nostro esempio pratico). Per semplificare il processo, Microchip ha introdotto un'applicazione desktop (chiamata MPF2) che, partendo da una normale pagina web realizzata in HTML, genera la corrispondente immagine MPFS da utilizzare nel nostro progetto embedded. L'applicazione, la cui schermata è riportata in Fig. 15, permette di generare immagini binarie (da immagazzinare su supporti esterni), file .c (nel caso di memorizzazione nella Flash del micro stesso), come anche codice per file system di tipo FAT (SD card o chiavetta USB). Nella nostra applicazione di esempio faremo uso di questo tool per la conversione della nostra pagina web.

VARIABILI DINAMICHE

Un'applicazione che implementa un web server all'interno del nostro PIC32 sarebbe piuttosto inutile se non fosse in grado di interagire in qualche modo con il resto del sistema per scambiare dati. Microchip ha pensato anche a questo e per risolvere il problema ha introdotto il concetto di variabile dinamica. Tali variabili permettono una interazione tra modulo Web Server e il resto dell'applicazione che gira sul microcontrollore, permettendo di avere delle variabili condivise tra il modulo stesso e il resto del codice. Un esempio dell'utilità di questa soluzione è schematizzato in Fig. 16: in questo schema è stata realizzata un'applicazione che al proprio interno implementa un Web Server e che è anche collegata ad un particolare sensore. Utilizzando il concetto di variabile dinamica, le letture del sensore sono combinate all'interno del Web Server in modo da essere visualizzate sulla pagina web e poi venire trasmesse via Internet, da cui sono disponibili per ogni client che ne faccia

richiesta. Le variabili dinamiche possono essere utilizzate, in combinazione con il Web Server, in modi differenti. Noi ne esamineremo due, che sono i due casi utilizzati nel nostro esempio pratico: visualizzazione di dati sulla pagina web ed uso del metodo GET. Nel primo caso abbiamo la possibilità di visualizzare sulla nostra pagina una variabile di sistema (ossia un generica variabile presente nel nostro progetto embedded). Per farlo sono necessari due step: dal lato pagina web, la variabile da visualizzare va racchiusa tra due simboli tilde (~), mentre dal lato embedded deve essere implementata un'opportuna callback, il cui prototipo viene generato dal tool MPFS2 quando la pagina web viene tradotta in immagine binaria. Ad esempio, se volessimo visualizzare una generica variabile "var" presente nel nostro progetto embedded, sulla pagina web non dovremmo fare altro che inserire questa variabile tra i simboli tilde:

```
~var~
```

Poi, dal lato embedded dovremmo implementare la seguente callback:

```
HTTPPrint_var()
```

Al resto penserà lo stack. Vedremo più chiaramente questa procedura nell'esempio pratico. Nel secondo caso, invece, l'esecuzione di un metodo GET lato web fa sì che tutti i dati passati con il metodo vengano accumulati sul buffer *HTTPcurr.data* e che venga chiamata la callback *HTTPExecuteGet()*. Combinando questi due eventi è possibile gestire l'occorrenza di una GET in maniera opportuna lato embedded. Anche questo aspetto risulterà più chiaro con la descrizione del progetto pratico.

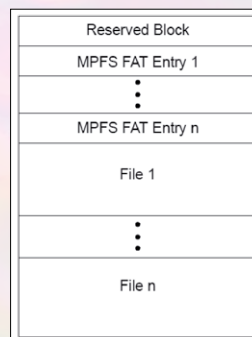


Fig. 12 - Formato di una immagine MPFS.

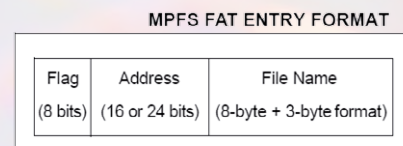


Fig. 13 - Formato della FAT Entry.

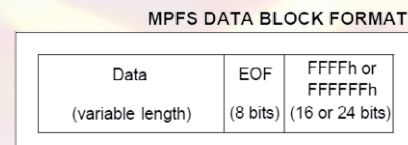


Fig. 14 - Formato del data block.

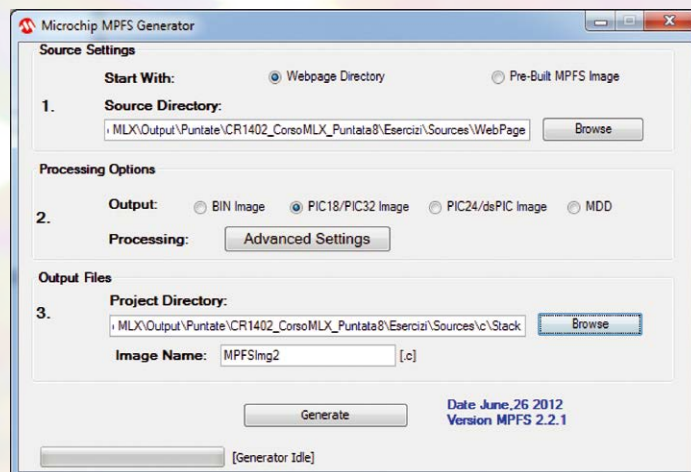


Fig. 15 - Schermata principale dell'applicazione MPFS2.jar.

ESEMPIO PRATICO: WEB SERVER EMBEDDED

Passiamo quindi alla parte pratica, nella quale ci proponiamo di realizzare un semplice Web Server embedded, che ci permetta di controllare da remoto un LED. L'applicazione in sé è molto semplice, ma ci permette di validare in maniera pratica quanto è stato presentato nel resto della puntata. Partiamo quindi dalla solita impostazione architetturale, il cui schema è visibile in Fig. 17. Come si può vedere, in questo



Fig. 16 - Esempio d'uso di una variabile dinamica.

Listato 1

```

/* Maximum number of task */
#define ACTIVE_TASK_NUMBER                ((UINT16) (2))

/* Array of task to call */
void (*TaskArray[ACTIVE_TASK_NUMBER]) (void) =
{
    LedTask,
    TcpIpTask,
};

/* Array of task period timeouts */
UINT16 TaskPeriodTimeoutMs[ACTIVE_TASK_NUMBER] =
{
    LED_TASK_PERIOD_MS,
    TCPIP_TASK_PERIOD_MS,
};

```

Listato 1 - Strutture dati.

Listato 2

```

/*****
* Function:      TcpIpTask
* Input:         None
* Output:        None
* Author:        F.Ficili
* Description:   Manage TCPIP Task.
* Date:          31/05/15
*****/
void TcpIpTask (void)
{
    switch (SystemState)
    {
        /* System Initialization Phase */
        case InitializationState:
            /* Initialize tcpip stack */
            TcpIpInitStack();
            break;

        /* System Normal operation Phase */
        case RunningState:
            /* Call tcpip tasks */
            TcpIpStackTasks();
            break;

        /* Default */
        default:
            break;
    }
}

```

Listato 2 - Implementazione del TcpIpTask

caso utilizziamo solo due task per implementare la nostra applicazione: un task dedicato al TCP/IP e il solito "Led task" che ci permette di avere un minimo di feedback dalla nostra applicazione (nel nostro caso è sempre un lampeggio del LED alla frequenza di 1 Hz). Chiaramente il TcpIpTask si appoggia sullo stack TCP/IP e sull'implementazione del web server HTTP di Microchip.

In Fig. 18 è invece riportato il project manager di MPLab X relativo al progetto pratico (che abbiamo chiamato MLX_Ex11), mentre la Tabella 2 riassume, come di consueto, le periodicità dei due task in esecuzione.

L'infrastruttura utilizzata è sempre la stessa,

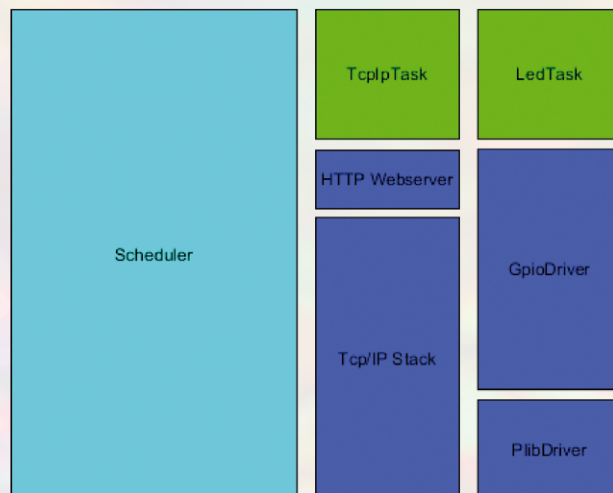


Fig. 17 - Architettura Software.

così come le definizioni delle strutture dati che supportano l'esecuzione dello scheduler, riportate nel Listato 1. Spostiamo ora la nostra analisi sul TcpIpTask, che è il task principale di questo esempio e vediamo come è stato implementato, analizzando il Listato 2: il task si compone di una sezione di inizializzazione, che serve per inizializzare correttamente le strutture dati dei vari task dello stack utilizzati in questa applica-

Task	Periodicità [ms]
LedTask	100
TcpIpTask	10

Tabella 2 - Periodicità dei task

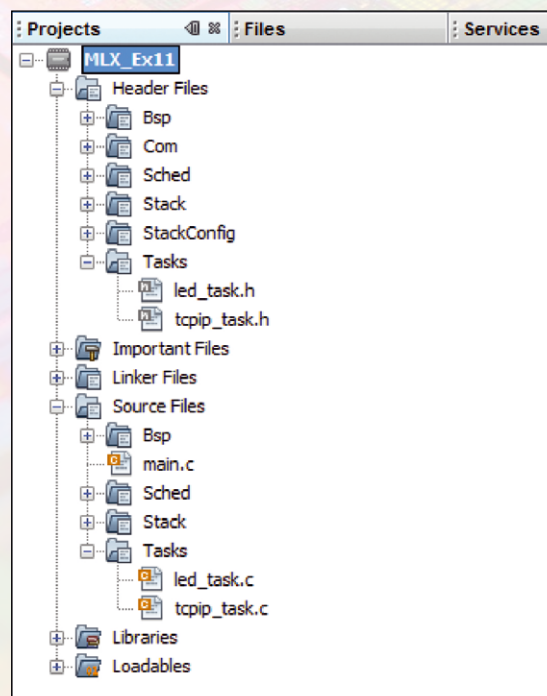


Fig. 18 - Project Manager MPLab X.

zione, e di una sezione "running" che fa girare le macchine a stati dello stack stesso. Analizziamo la struttura della funzione di inizializzazione, il cui codice è riportato all'interno del **Listato 3**. In questa funzione vengono inizializzati in sequenza:

- il componente Tick, che serve a generare tutte le necessarie tempistiche per l'intero stack;
- il componente MPFS, che serve per l'implementazione delle pagine web del server;
- la struttura dati AppConfig.

Vengono inoltre operate le inizializzazioni più generali dello stack; in particolare, l'inizializzazione della struttura dati AppConfig serve a caricare la maggior parte dei dati impostati tramite il tool di configurazione, come ad esempio i valori degli indirizzi, il NetBios Name, ecc...

Passiamo ora all'esame della parte operativa del task, il cui codice è riportato all'interno del **Listato 4**. Sono due i task chiamati in questo caso; il primo, ossia StackTasks, manda in esecuzione tutte le macchine a stati principali dello stack. Si tratta di una serie di servizi generali e temporizzazioni che sono necessari a qualsiasi task di livello applicazione dello stack. Invece il task StackApplications manda in esecuzione una serie di task di livello applicativo e dipende fortemente dalla configurazione dello stack, contenuta in *TCPIPConfig.h*. Nel caso specifico, ad esempio, sono eseguiti i task relativi al web server HTTP, al DHCP ed al NetBIOS Name Server.

Ora il nostro stack TCP/IP è correttamente inizializzato e sta girando, tuttavia se avviamo l'applicazione il nostro Web Server non permetterà di visualizzare nulla, dato che non abbiamo caricato nessuna pagina al suo interno. Per consentire la visualizzazione di una pagina web occorre creare un'immagine MPFS adatta ad essere ospitata sul Web Server http, come è stato descritto precedentemente. La pagina che intendiamo caricare è una normale pagina HTML, il cui codice è riportato nel **Listato 5**.

Analizziamo brevemente il codice HTML della pagina; la sezione iniziale comprende solo testo racchiuso tra differenti headings tag (h1, h2 e h3) e quindi è solo una porzione di HTML che visualizzerà del testo statico (in particolare la scritta "Welcome!" e alcune righe di descrizione). La sezione successiva fa uso del concetto di variabile dinamica per la visualizzazione del NetBios Name associato alla scheda (come abbiamo visto in precedenza, la variabile ~hname~ invocherà

Listato 3

```

/*****
 * Function:      TcpIpInitStack
 * Input:         None
 * Output:        None
 * Author:        F.Ficili
 * Description:   Initialize the TCPIP stack.
 * Date:          31/05/15
 *****/
void TcpIpInitStack (void)
{
    /* Init tick counts */
    TickInit();
    /* Init MPFS */
    MPFSInit();
    /* Init AppConfig */
    InitAppConfig();
    /* Init stack */
    StackInit();
}

```

Listato 3 - Implementazione della funzione TcpIpInitStack.

Listato 4

```

/*****
 * Function:      TcpIpStackTasks
 * Input:         None
 * Output:        None
 * Author:        F.Ficili
 * Description:   Manage the TCPIP necessary tasks.
 * Date:          31/05/15
 *****/
void TcpIpStackTasks (void)
{
    /* Perform Stack Tasks */
    StackTask();

    /* This tasks invokes each of the core stack application tasks */
    StackApplications();
}

```

Listato 4 - Implementazione della funzione TcpIpStackTasks.

Listato 5

```

<h1>Welcome!</h1>
<h1>This is a test webpage!!!</h1>
<h3>This test webpage prove the capability _
of Microchip free TCP/IP stack</h3>

<div id="location">
    Machine NetBios Hostname: ~hname~
</div>

<br>

<form>

    <div id="led_control">
        LED Control:

        <input type="radio" name="led"
            value="on"/> On
        <input type="radio" name="led"
            value="off"/> Off

        <br>
        <input type="submit" class="btn"
            value="Set"/>

    </div>
</form>

```

Listato 5 - Codice HTML della pagina web da caricare sul Web Server.

Listato 6

```

/*****
* Function:      HTTPPrint_hname
* Input:         None
* Output:        None
* Author:        F.Ficili
* Description:   Print hostname callback.
* Date:          31/05/15
*****/
void HTTPPrint_hname(void)
{
    TCPPutString(sktHTTP, AppConfig.NetBIOSName);
}

```

Listato 6 - Implementazione della callback HTTPPrint_hname.

Listato 7

```

/*****
* Function:      HTTPExecuteGet
* Input:         None
* Output:        None
* Author:        F.Ficili
* Description:   Execute get callback.
* Date:          31/05/15
*****/
HTTP_IO_RESULT HTTPExecuteGet(void)
{
    BYTE *DataPtr;

    /* Get data */
    DataPtr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *) "led");

    /* Check if data is present inside the pointer */
    if (DataPtr)
    {
        /* If "on" data has been passed */
        if (strcmp(pgm2ram((char*)DataPtr, (ROM char*) "on")
        == 0)
        {
            /* Send on evt */
            GenerateEvt(&Led2On);
        }
        else
        {
            /* Send off evt */
            GenerateEvt(&Led2Off);
        }
    }
}

```

Listato 7 - Implementazione della callback HTTPExecuteGet.

la callback HTTPPrint_hname, tramite la quale sarà possibile, usando una semplice funzione, far visualizzare al posto della variabile stessa il NetBios Name). Infine, la sezione racchiusa tra i tag <form> e </form> fa uso del metodo GET per generare un input per il web server che permetta di controllare un LED sulla scheda (per permettere di intercettare i dati passati dal metodo GET è necessario utilizzare la callback HTTPExecuteGet). Dal punto di vista grafico, questa sezione genera due radio button contrassegnati rispettivamente con On e Off ed un pulsante (Set). Prima di passare alla realizzazione delle callback, vediamo come generare il codice eseguibile della

pagina web, in maniera che possa essere integrato nel progetto MPLab X. Come abbiamo visto in precedenza, il codice della pagina si genera utilizzando il tool MPFS2. Possiamo sia utilizzare l'interfaccia grafica del tool, o, in alternativa, utilizzare uno script batch, come quello illustrato qui di seguito:

```

java -jar "..\Utilities\MPFS2.jar" /mpfs2 /C18_C32
"\WebPage" "." "MPFSImg2.c"

```

che invoca il tool MPFS2, passando i corretti parametri per la generazione del codice C corrispondente alla pagina.

Il risultato dell'esecuzione dello script è costituito dai seguenti due file:

- MPFSImg2.c = costituisce il codice C che implementa la pagina web da visualizzare;
- HTTPPrint.h = è l'header file che contiene i prototipi delle callback da includere nel progetto.

Entrambi i file vanno inclusi nel progetto MPLab X. La pagina web, come da configurazione impostata tramite il tool TCPConfig, verrà compilata in modo da risiedere direttamente nella memoria Flash del PIC32. Abbiamo quasi finito: non ci resta che implementare le callbacks relative alla variabile dinamica hname ed al metodo GET utilizzato sulla pagina web e compilare il nostro progetto. Iniziamo con la callback per la visualizzazione del NetBios Name. Come abbiamo visto in precedenza, la presenza della variabile dinamica ~hname~ nella pagina web provocherà, al caricamento della pagina, l'invocazione della callback HTTPPrint_hname(); non dobbiamo fare altro che implementare questa callback in modo che visualizzi il NetBios Name della scheda, una volta invocata. Il NetBios Name (il nome dell'host che abbiamo dato in fase di configurazione, nel nostro esempio EINBOARD) è facilmente recuperabile dalla struttura dati AppConfig ed il codice che implementa la callback è riportato nel Listato 6. In questo caso lo stack ci viene molto in aiuto, fornendoci la funzione TCPPutString, che scrive sul socket passato come primo parametro i dati presenti nell'array passato come secondo parametro; ci basta quindi passare alla funzione il socket http (sktHTTP) e i dati da visualizzare (AppConfig.NetBIOSName): molto semplice, come vedete...

Invece per quanto riguarda il controllo del LED, per intercettare il flusso di dati richiesti al server

tramite il metodo GET, lo stack ci fornisce la callback HTTPExecuteGet, che viene appunto invocata all'esecuzione di questo metodo. Inoltre i dati passati vengono automaticamente caricati sul buffer currHTTP.data. Dato che la callback è unica per tutte le istanze del metodo, occorrerà esaminare i dati e, tramite un'opportuna struttura di selezione, effettuare le azioni specifiche. Le implementazioni possibili, nel caso specifico, sono molteplici; noi abbiamo fornito quella illustrata nel **Listato 7**. Per prima cosa, attraverso la funzione HTTPGetROMArg, vengono recuperati i dati associati all'argomento passato come parametro (in questo caso la stringa "led"), ed il risultato viene passato ad un opportuno puntatore. Poi, per prima cosa si controlla se il puntatore ha dei dati validi, e quindi si verifica di fatto che l'operazione precedente abbia fornito un "match"; in caso affermativo, effettuando una comparazione tra la stringa dati e la stringa "on", si determina se il LED va acceso o spento. Si noti che l'effettiva accensione (o lo spegnimento) del LED non viene eseguita direttamente nella callback, ma l'operazione viene delegata al LedTask, tramite l'invio di un opportuno evento (**GenerateEvt(&Led2On)** e **GenerateEvt(&Led2Off)**). Lo snippet di codice del LedTask che intercetta l'evento è riportato all'interno del **Listato 8**. L'operazione potrebbe sembrare un pochino macchinosa, ma delega correttamente i compiti al task specifico, permettendoci di mantenere la corretta modularità nell'implementazione. Se un domani l'effetto del clic sul pulsante "Set" dovesse essere cambiato dal semplice on/off a un'operazione più complessa, come ad esempio un lampeggio, il LedTask sarebbe sicuramente il task adatto cui far compiere questa operazione, che non potrebbe essere implementata correttamente dalla HTTPExecuteGet, il che ci obbligherebbe a modificare lo snippet o aggiungere una funzione specifica.

La nostra applicazione a questo punto è completa; non ci resta che testarla, ad esempio, usando la demoboard a supporto del corso. Dopo aver effettuato il download del codice, aver dato potenza ed aver collegato il cavo ethernet al nostro PC o al nostro access point, dobbiamo semplicemente collegarci, con un qualsiasi browser, all'indirizzo `http://einboard`, e verrà visualizzata la pagina mostrata in **Fig. 19**. Come possiamo vedere, oltre alla stringa statica, la pagina fornisce il NetBIOS name associato alla scheda: EINBOARD. Per controllare invece il LED è sufficiente selezionare

Listato 8

```
/* If a Led on evt i received */
if (ReceiveEvt(&Led2On))
{
    /* Turn on LED */
    LED_2 = LED_ON;
}
else if (ReceiveEvt(&Led2Off))
{
    /* Turn off LED */
    LED_2 = LED_OFF;
}
else
{
    /* Do nothing */
}
```

Listato 8 - Snippet del LedTask che intercetta gli eventi di accensione/spegnimento del LED 2.

il radio button corrispondente all'azione che si desidera eseguire (LED on o off) e fare clic sul pulsante "Set". Il LED si accenderà o si spegnerà di conseguenza. Ricordiamo che, avendo incluso il modulo DHCP non sarà necessario modificare gli attributi della scheda di rete.

CONCLUSIONI

In questa ottava puntata abbiamo analizzato lo stack TCP/IP open source della Microchip, fornito a supporto dell'ambiente di sviluppo MPLab X, spingendoci con questo potente strumento ad una implementazione di una certa complessità: la realizzazione di un Web Server embedded. Questa applicazione evidenzia quanto potente e versatile sia l'ambiente di sviluppo, la piattaforma utilizzata (ossia i microcontrollori PIC32) e gli stack e le infrastrutture software fornite a supporto. A chi fosse interessato ad approfondire tematiche di questo tipo ricordiamo che a breve verrà realizzato un apposito corso da Futura Academy (www.futura.academy).

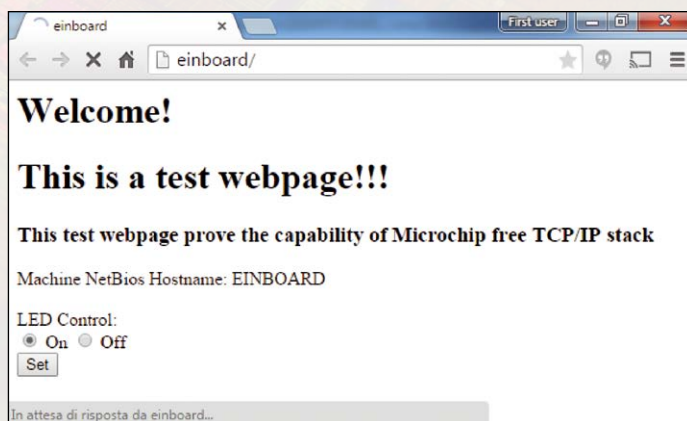


Fig. 19 - Pagina web visualizzata accedendo al Web Server con chrome.