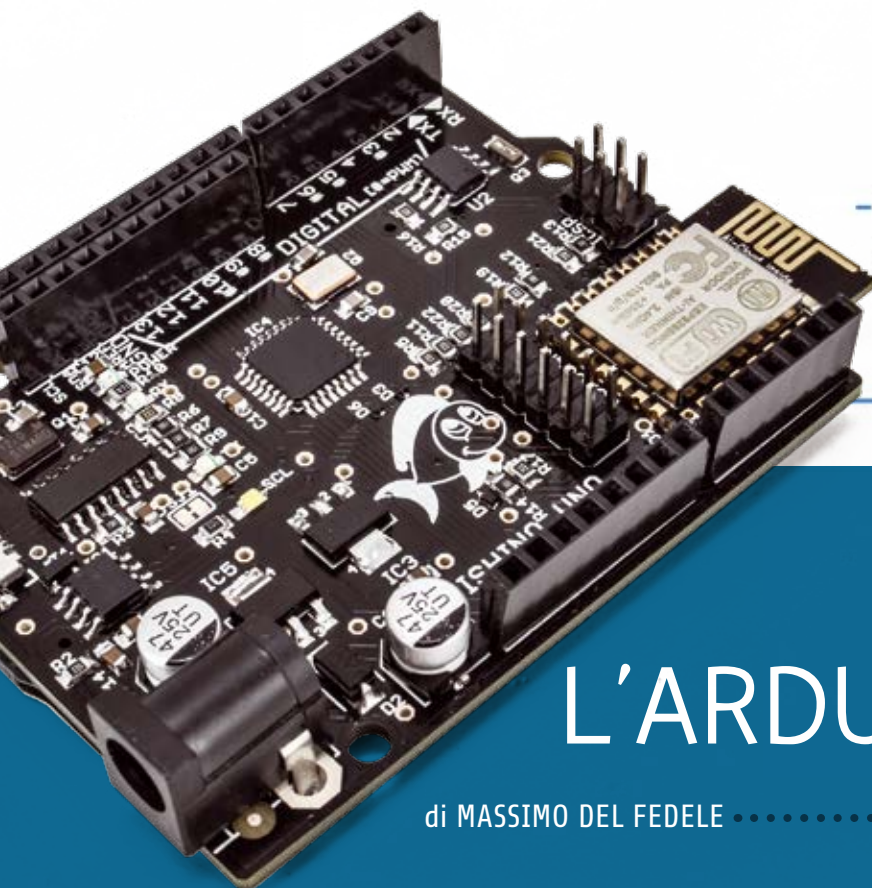


Nel firmamento Arduino brilla una nuova “scheda” simile alla UNO ma dotata di connettività WiFi e lettore per SD. Prima puntata.



FISHINO, L'ARDUINO DIVENTA WIRELESS

di MASSIMO DEL FEDELE

Il nome della scheda descritta in queste pagine è nato da un “pesce d’aprile” fatto dall’autore su un forum dove si presentava una fantomatica scheda denominata “**Fishino Zero**” dotata di caratteristiche mirabolanti, tra le quali un processore “Horata” di ultima generazione, tecnologia WiFi “Poisson”, connessione avanzata WiFi “Fishnet” ed altre improbabili meraviglie. Il simbolo, piazzato tramite un programma di grafica sulla foto di una scheda esistente, era appunto il pesciolino che, finito lo scherzo, è diventato il logo della scheda che proponiamo in queste pagine. Da scherzo, l’idea ha iniziato a prender forma fino alla realizzazione della board definitiva. Il termine **UNO** è poi stato aggiunto sia per indicare la prima di una possibile serie di schede Arduino-compatibili, che per indicare la completa compatibilità con Arduino Uno, sia come connettività che come dimensioni.

UN ALTRO CLONE DI ARDUINO?

Si e no: con questa scheda abbiamo voluto realizzare un prodotto nuovo in grado di abbinare la semplicità d’uso e la sterminata quantità di librerie e shield di Arduino con la connettività Internet, una dotazione praticamente illimitata di memoria grazie alla scheda microSD e, ultimo ma non per importanza, un orologio interno con backup a batteria, il tutto ad una frazione del costo d’acquisto di una Arduino e degli shield relativi alle funzioni implementate, senza occupare prezioso spazio aggiuntivo; la scheda ha infatti un ingombro identico a quello dell’Arduino Uno, salvo la piccola sporgenza dell’antenna WiFi, di soli 7 mm.

L’integrazione delle periferiche descritte, a nostro avviso indispensabili nell’era dell’IoT, permette di realizzare tutta una serie di apparecchi sia controllabili via Internet che in grado di connettersi ad essa a richiesta e trasmettere dati rilevati in precedenza.

CARATTERISTICHE TECNICHE

- **Alimentazione: 12 Vcc o USB**
- **Completamente compatibile con Arduino Uno**
- **Scheda WiFi a bordo, con possibilità di funzionamento in modalità stazione, access point o entrambe contemporaneamente**
- **Interfaccia per schede di memoria MicroSD a bordo**
- **RTC (Real Time Clock) a bordo con batteria al litio di mantenimento**
- **Sezione di alimentazione a 3,3 V potenziata**
- **Connettore aggiuntivo sfalsato in modo da risolvere il problema dell'incompatibilità di Arduino con le schede millefori.**

Tra le realizzazioni possibili ci sono, per esempio:

- sistemi di home automation gestibili via Internet tramite un Web Browser;
- data logger portatili in grado di connettersi e scaricare i dati sulla rete quando si entra nel campo di copertura di una rete WiFi;
- robot controllabili via rete e in grado di trasmettere tramite essa i dati rilevati da sensori.

L'utilizzo di un modulo WiFi a basso costo ma con firmware da noi "hackerato" per ottenere prestazioni elevate e capace di funzionare anche come access point, ovvero senza la necessità di una struttura di rete WiFi esistente, permette il controllo via cellulare in qualsiasi momento, anche in assenza di copertura di rete, rendendo il dispositivo sempre interattivo. La possibilità di eseguire l'aggiornamento degli sketch via Internet, già prevista a livello hardware, permetterà inoltre di avere dispositivi sempre aggiornati senza la necessità di doverli collegare fisicamente ad un computer.

SCHEMA ELETTRICO

Il Fishino Uno, allo stesso modo di Arduino Uno, può essere alimentato tramite sia la porta USB che il connettore per l'alimentazione esterna. L'alimentazione viene automaticamente commutata su quella proveniente dal connettore esterno quando ai capi del medesimo (oppure all'ingresso Vin) viene applicata una tensione sufficiente al funzionamento del regolatore lineare U5.

La tensione giunge dal connettore di alimentazione al regolatore attraverso il diodo Schottky D2, utilizzato come protezione contro l'inversione della polarità; è stato scelto uno Schottky al posto di un più tradizionale diodo in silicio per la più bassa caduta di tensione che presenta: $0,3 \div 0,4$ volt contro gli $0,7$ circa dei diodi in silicio; questo, e l'aver utilizzato un regolatore lineare "low dropout" (a bassa caduta di tensione) permette di alimentare la scheda già con $6,6$ volt (5 necessari ai circuiti + $0,4$ di caduta sul diodo + $1,2$ di caduta massima sul regolatore). La massima tensione di alimentazione ammissibile dipende invece dalla dissipazione del regolatore lineare utilizzato; si sconsiglia di superare i 12 V e, se possibile, di restare intorno ai 9 volt. Il regolatore dispone comunque di protezioni interne che lo disattivano quando la dissipazione risulta eccessiva.

La tensione in ingresso Vin (dopo il diodo di protezione) viene inoltre inviata all'operazionale U1A utilizzato per la commutazione dell'alimentazione tramite porta USB.

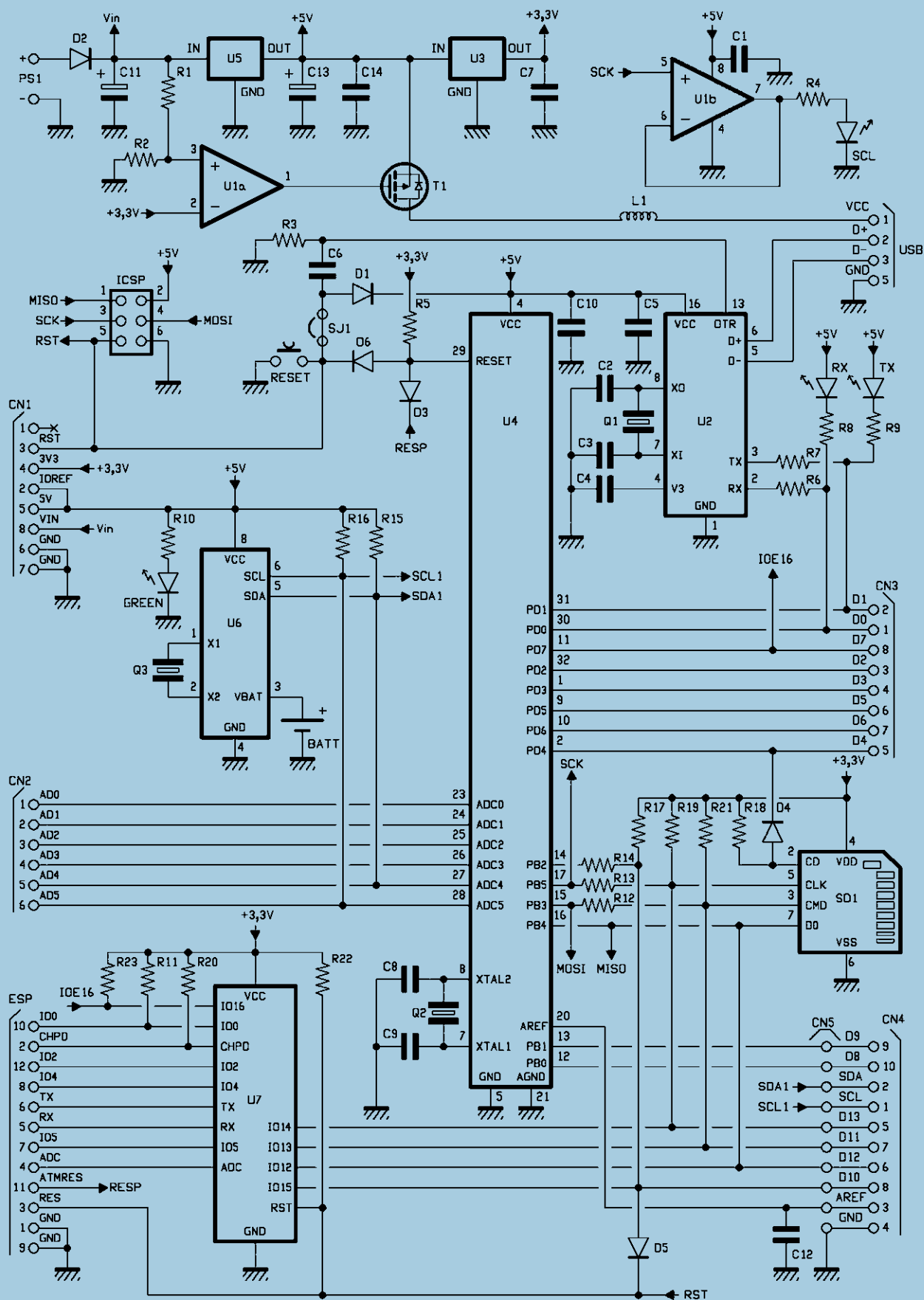
Quando la tensione Vin supera i $6,6$ volt la tensione all'ingresso non invertente dell'operazionale, utilizzato qui come comparatore, dimezzata tramite il partitore co-

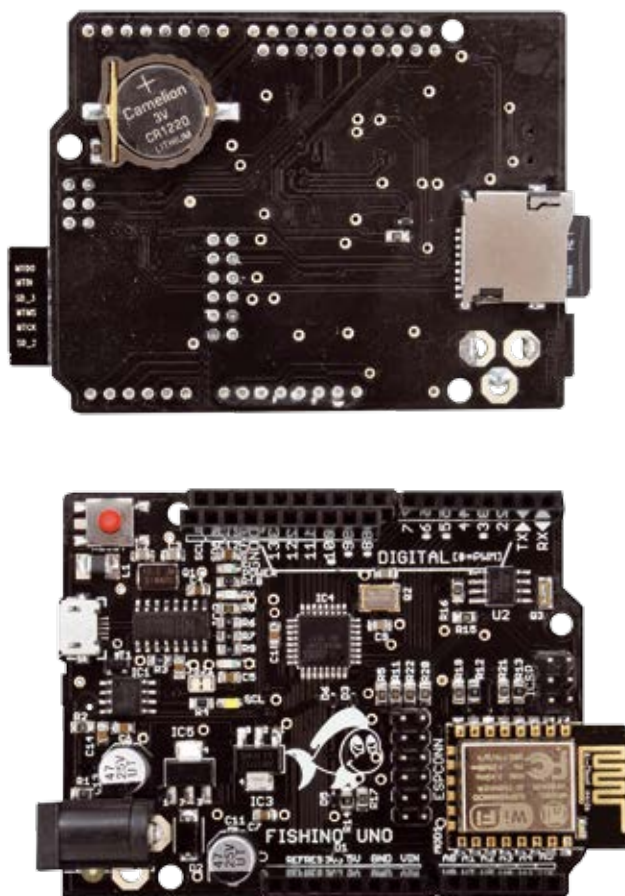
stituito dalle resistenze R1 ed R2, supera quella di riferimento di $3,3$ volt all'ingresso invertente e l'uscita commuta a livello positivo, provocando l'interdizione del MOSFET a canale P siglato T1. Guardando da vicino quest'ultimo, il collegamento è apparentemente strano: l'alimentazione entra dal Drain ed esce dal Source, passando in contemporanea attraverso il diodo interno di clamping; ciò significa che la tensione proveniente dall'ingresso USB passa comunque attraverso il diodo e va ad alimentare il circuito anche se il MOSFET è interdetto.

A che serve quindi il MOSFET? Apparentemente con il solo diodo la commutazione sarebbe assicurata, perché se la tensione al catodo è superiore a quella dell'anodo (USBVCC) il diodo viene interdetto scollegando quindi l'alimentazione USB. Il motivo della presenza del MOSFET (e circuiteria annessa) è da cercarsi sempre nella caduta di tensione provocata dal diodo, che farebbe diminuire l'alimentazione dai 5 volt della linea USB a $4,2 \div 4,6$ volt circa, caduta evitata dal MOSFET stesso una volta entrato in conduzione.

A completare l'alimentazione è presente l'integrato U3, che fornisce in uscita la tensione a $3,3$ V necessaria, tra l'altro, per la scheda SD ed il modulo WiFi. A differenza dell'Arduino originale, che fornisce poche decine di mA sulla linea a $3,3$ volt, nel Fishino sono utilizzabili circa $7 \div 800$ mA a seconda del consumo sulla linea a 5 Volt.

L'interfaccia USB di Fishino è stata realizzata, diversamente dall'Arduino originale, tramite un chip CH340G (siglato U2) in sostituzione al più noto FT232 o altre soluzioni più o meno complicate.





Elenco Componenti:

R1: 10 kohm (0805)	C5: 100 nF ceramico (0805)
R2: 10 kohm (0805)	C6: 1 μ F ceramico (0805)
R3: 1 kohm (0805)	C7: 1 μ F ceramico (0805)
R4: 1 kohm (0805)	C8: 22 pF ceramico (0805)
R5: 10 kohm (0805)	C9: 22 pF ceramico (0805)
R6: 1 kohm (0805)	C10: 100 nF ceramico (0805)
R7: 1 kohm (0805)	C11: 47 μ F 16VL elettrolitico ($\varnothing$ 6mm)
R8: 2,4 kohm (0805)	C12: 100 nF ceramico (0805)
R9: 1 kohm (0805)	C13: 47 μ F 16VL elettrolitico ($\varnothing$ 6mm)
R10: 470 ohm (0805)	C14: 100 nF ceramico (0805)
R11: 10 kohm (0805)	D1: RB521S
R12: 1 kohm (0805)	D2: M7
R13: 1 kohm (0805)	D3: RB521S
R14: 1 kohm (0805)	D4: RB521S
R15: 10 kohm (0805)	D5: CD1206-S01575
R16: 10 kohm (0805)	D6: RB521S
R17: 3,3 kohm (0805)	BATT: Porta batterie CH291-1220LF
R18: 10 kohm (0805)	U1: LMV358L
R19: 3,3 kohm (0805)	U2: CH340G
R20: 10 kohm (0805)	U3: NCP1117ST33T3G
R21: 3,3 kohm (0805)	U4: ATMEGA328P-AU
R22: 10 kohm (0805)	U5: NCP1117ST50T3G
R23: 10 kohm (0805)	U6: DS1307
C1: 100 nF ceramico (0805)	U37: ESP12
C2: 22 pF ceramico (0805)	
C3: 22 pF ceramico (0805)	
C4: 1 μ F ceramico (0805)	

La scelta è stata dettata principalmente da motivi di costo e di semplificazione circuitale a parità di prestazioni. L'integrato per funzionare necessita di pochissimi componenti esterni: un quarzo a 12 MHz (Q1), due condensatori per garantire la stabilità dell'oscillatore (C2 e C3) ed un condensatore di disaccoppiamento per il regolatore interno a 3,3V (C4).

Il circuito può infatti essere alimentato indifferentemente a 3,3 volt (collegando il pin V3 al Vcc) oppure a 5 volt, inserendo un condensatore di disaccoppiamento tra il pin V3 e la massa.

Il componente fornisce in uscita tutti i segnali di un'interfaccia

RS232, ovvero i due segnali di trasmissione/ricezione dati (Rx e Tx) ed i segnali di controllo (CTS, DSR, DXD, DTR e RTS); nel nostro circuito vengono utilizzati solo i segnali dati (RX e TX) e il DTR per generare l'impulso di reset: ogniqualvolta viene aperta la porta seriale, in modo da rendere possibile il caricamento degli sketch senza dover resettare manualmente il dispositivo. Torneremo in seguito sulla circuiteria di reset per spiegare che rispetto all'originale è stata modificata in modo da permettere in futuro la riprogrammazione dell'Atmega via WiFi.

Non disponendo, l'integrato utilizzato, di due uscite separate

per i LED TX ed RX (che segnalano le attività di trasmissione e/o ricezione) questi ultimi sono stati inseriti direttamente sulle due rispettive linee dati, con una scelta accurata dei valori di resistenza in modo da non provocare un'attenuazione eccessiva.

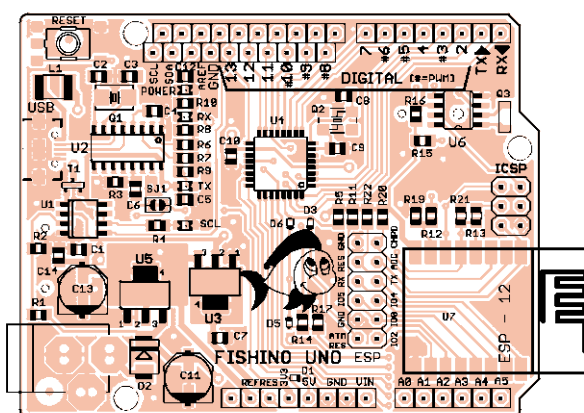
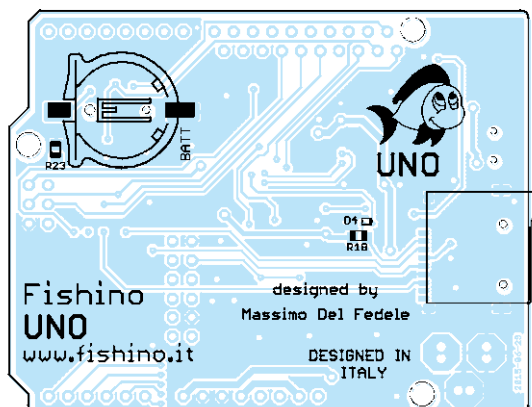
La scelta dei valori è stata inoltre calibrata per permettere alla stessa interfaccia USB/Seriale di riprogrammare il firmware del modulo WiFi senza dover ricorrere ad adattatori esterni.

Per quanto riguarda l'Atmega328P, lo schema di Fishino non si discosta da quello di Arduino Uno: il controller è il medesimo, solo in formato SMD per esigenze di spazio sulla board, corredata

T1: FDN340P
 L1: MF-MSMF050-2
 500mA (1812)
 POWER: LED verde (0805)
 RX: LED giallo (0805)
 TX: LED blu (0805)
 SCL: LED bianco (0805)
 Q1: Quarzo 12 MHz
 Q2: Quarzo 16 MHz
 Q3: Quarzo 32.768 KHz
 RESET: Microswitch TS42
 SD1: Connettore micro-SD

Varie:

- Plug alimentazione
- Connettore micro-USB
- Strip femmina 6 vie (1 pz.)
- Strip femmina 8 vie (2 pz.)
- Strip femmina 10 vie (2 pz.)
- Strip maschio 3 vie (2 pz.)
- Strip maschio 6 vie (2 pz.)
- Circuito stampato S1225



to dall'usuale quarzo a 16 MHz, dai due condensatori sul circuito oscillante e da un certo numero di condensatori di disaccoppiamento sulle linee di alimentazione. Una piccola parentesi su questi ultimi: spesso negli schemi si vedono alcuni (a volte molti) condensatori in parallelo all'alimentazione; qualcuno si chiederà perché al posto dei tanti condensatori non ne abbiamo usato uno solo di capacità equivalente. Ebbene, il motivo è che i moderni circuiti integrati digitali lavorano a frequenze elevate e con segnali impulsivi, che causano grosse variazioni di assorbimento di corrente sui pin di alimentazione dovute alla resistenza delle piste;

per questo è necessario applicare in prossimità dei pin di alimentazione dei condensatori che filtrino i disturbi prima che si propaghino al resto del circuito. Un condensatore solo non servirebbe. Ora una nota sui connettori di I/O: a Fishino è stato aggiunto un piccolo connettore a 10 pin affiancato a quello standard, ma leggermente sfalsato in modo da poter utilizzare una scheda preforata standard per gli shield, risolvendo quindi l'annoso problema del passo incompatibile dei connettori laterali di Arduino. Tutta la circuiteria di Fishino opera a 5 volt, mentre sia le schede MicroSD che il modulo WiFi funzionano a 3,3 volt e, soprattutto,

non sono 5V-tolerant; questo significa che fornendogli segnali TTL a 5 volt si corre il rischio di bruciarli.

Per quanto riguarda il modulo WiFi, da prove effettuate non abbiamo riscontrato alcun problema fornendogli dati TTL a 5 volt, ma, per evitare possibili guasti a medio/lungo termine, abbiamo preferito rispettare le specifiche tecniche ed inserire un circuito di adattamento dei livelli.

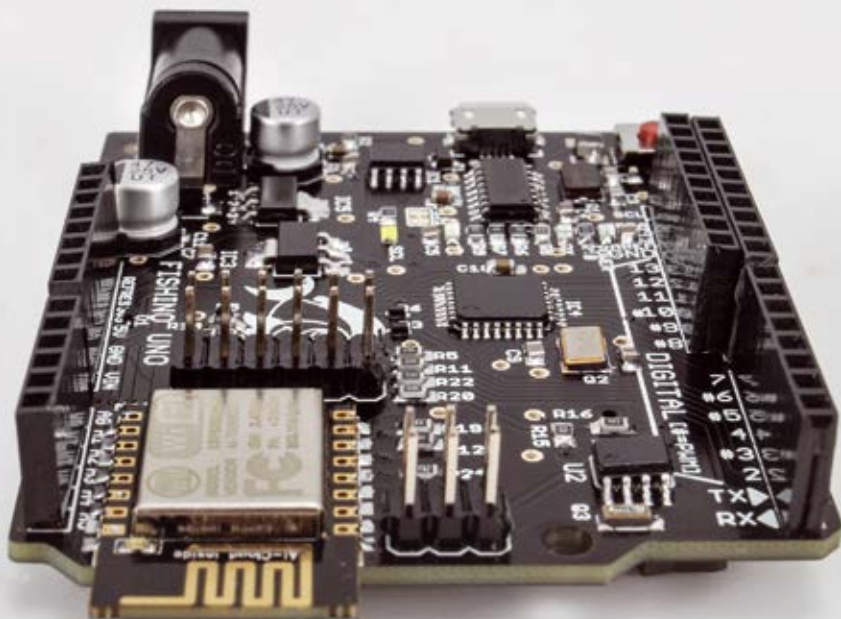
Questo è realizzato molto semplicemente tramite partitori resistivi, per quanto riguarda la direzione Atmega --> SD e WiFi, mentre in direzione opposta si sfrutta il fatto che il valore H dei circuiti a 3,3 volt (da 2,95 volt circa in su) è al limite ma dentro il range compatibile con il livello alto a 5 volt. Si sono quindi risparmiate complesse circuiterie con transistor e/o traslatori integrati.

L'unico "difetto", se così si può dire, dell'approccio utilizzato, è che i partitori resistivi devono sia avere un'impedenza sufficientemente alta per non sovraccaricare le uscite dell'Atmega328P e per non aumentare inutilmente il consumo di corrente, sia avere un'impedenza sufficientemente bassa da non causare perdite di segnali in alta frequenza.

I valori scelti sono un compromesso tra i due requisiti e permettono il buon funzionamento dell'interfaccia anche alla massima velocità. I partitori sono costituiti dalle coppie R13-R21, R12-R19 e R14-R17. Un valore in ingresso a 5 Volt viene convertito a:

$$V_o = 5 \text{ Volt} \cdot 3.3K / (1K + 3.3K) = 3,83 \text{ Volt}$$

valore leggermente superiore ai 3,3 volt teorici ma accettabile secondo le specifiche, essendo ammessi solitamente valori fino a 0,6 Volt superiori all'alimentazione, quindi $3,3 + 0,6 = 3,9 \text{ V}$.



Interfaccia scheda MicroSD

L'interfaccia è semplicissima e rispecchia lo shield SD (o analoghi combinati); funziona attraverso le linee SPI tra cui MOSI (dati dall'Atmega verso la SD, Master Out Slave In), MISO (dati dalla SD all'Atmega, Master In Slave Out) e SCK (clock). I valori verso la scheda SD sono ovviamente ridotti dagli adattatori di livello di cui al paragrafo precedente. La selezione della scheda avviene tramite la linea SDCS, attiva a livello basso. In questo caso l'adattamento di livello viene effettuato tramite una resistenza (R18) verso il positivo ed un diodo (D4) che permette il solo passaggio dei valori negativi. Lo schema scelto consente di avere la scheda in stand-by quando il segnale SDCS (connesso al pin digitale 4 del Fishino) non è utilizzato; col pin in modalità three-state (ovvero ad alta impedenza) il diodo non conduce e sull'ingresso SDCS è presente un valore alto che disattiva la scheda. L'interfaccia è totalmente compatibile con gli shield di Arduino, quindi utilizza le stesse librerie esistenti per il suo funzionamen-

to, come si vedrà negli esempi presentati in seguito.

Modulo WiFi

Se l'Atmega può essere considerato il cervello del Fishino, il modulo WiFi è sicuramente la sua porta d'accesso al mondo esterno, ed è il principale motivo a monte dello sviluppo della scheda. L'idea di creare Fishino è nata infatti dall'esigenza di sviluppare un sistema di Home Automation con possibilità di controllo via Internet. In precedenza per poterla realizzare era necessario usare Arduino con annesso modulo WiFi o ethernet, con costi ed ingombri decisamente più elevati e, nel caso dell'ethernet, la necessità di una connessione cablata alla rete. L'inserimento della connettività WiFi direttamente nel Fishino a costi ragionevoli è stato reso possibile dall'immissione sul mercato dei moduli WiFi ESP8266. Questi moduli contengono un processore a 32 bit, una Flash di programma molto capiente (da 1 a 4 MBit), circa 90 kByte di RAM di cui 32 disponibili per l'utente, un completo stack WiFi e tutta la componentistica di contorno

necessaria al loro funzionamento, fino all'antenna integrata sul PCB. A prima vista, una potenza di calcolo di questo livello (decisamente superiore a quella dello stesso Atmega utilizzato da Arduino), suggerirebbe l'utilizzo del solo modulo programmandolo direttamente, vuoi con il SDK (Software Development Kit) fornito dal produttore (piuttosto complesso da utilizzare), oppure tramite una serie di strumenti che ne permetta la programmazione con l'IDE di Arduino, quasi fosse un nuovo modello del medesimo. Dove sta il problema, allora? Ebbene, per prima cosa le architetture sono molto diverse, Anche se l'IDE di Arduino è stata adattata per compilare e caricare codice direttamente sull'ESP, la compatibilità è ancora troppo limitata. Inoltre si perde la possibilità di utilizzare l'enorme quantità di librerie e shield di cui Arduino è dotato. Infine l'ESP ha un numero ridottissimo di pin di I/O digitale ed un solo input analogico, cosa che ne limita fortemente le possibilità d'impiego. Si è scelto quindi di integrare in una board compatibile con Arduino UNO il modulo WiFi, cercando di renderlo il più simile possibile nell'uso agli analoghi shield WiFi ed Ethernet, grazie ad un'apposita libreria di cui parleremo in seguito. L'utilizzo del modulo ESP è stato inizialmente difficoltoso e fonte di vari tentativi più o meno fallimentari, a causa principalmente del firmware fornito dal costruttore. Questo infatti è realizzato per comunicare attraverso la porta seriale e non, come negli shield originali, attraverso la porta SPI. I vantaggi della porta seriale sono ovviamente la semplicità d'uso (basta collegare un terminale seriale al modulino per

dialogare con esso) e la necessità di due sole linee di dati per la comunicazione.

A fronte di suddetti vantaggi, però, si hanno i seguenti problemi:

- velocità limitata; la porta seriale può viaggiare ad un massimo di 2-300 kbit/secondo, nei casi migliori (velocità più elevate sono possibili ma richiedono connessioni molto corte ed una velocità notevole del controller connesso);
- mancanza di handshake o controllo di flusso dei dati; pur prevedendo alcuni moduli ESP 2 linee di controllo di flusso hardware, il firmware pre-caricato non le utilizza, ed è quindi impossibile bloccare un

trasferimento dati dal modulo verso l'Atmega.

Riguardo a questo secondo problema, immaginiamo di aprire una pagina web contenente da qualche decina a qualche centinaio di kilobyte di dati (cosa usuale per una pagina web), che vengono inviati dal modulo al Fishino, il quale dispone di soli 2 kB di RAM, senza possibilità di comunicare al modulo stesso di attendere l'elaborazione dei dati; ecco, appare evidente cosa accade se non si può controllare il trasferimento dei dati. La porta seriale hardware dal lato Fishino è necessaria per ottenere velocità decenti. Con la seriale software non siamo riusci-

ti ad ottenere velocità affidabili sopra i 57 kbaud.

Dopo svariati tentativi infruttuosi di realizzare una libreria Arduino che permettesse l'uso del modulo ESP con il firmware standard, si è deciso di risolvere il problema a monte riscrivendo un firmware ad hoc, che permettesse l'utilizzo della ben più veloce interfaccia SPI.

A fronte del lungo tempo di sviluppo software del medesimo (la documentazione reperibile è decisamente scarsa) vantaggi sono stati subito evidenti:

- velocità; il bus SPI può viaggiare a 8 MHz, con un clock dell'Atmega di 16 MHz. Anche considerando che il protocollo richiede dei dati di

ElettroExpo

VERONA 28/29 NOVEMBRE 2015

**53[^] FIERA DELL'ELETTRONICA
DELL'INFORMATICA E DEL RADIOAMATORE**

www.elettroexpo.it

organized by



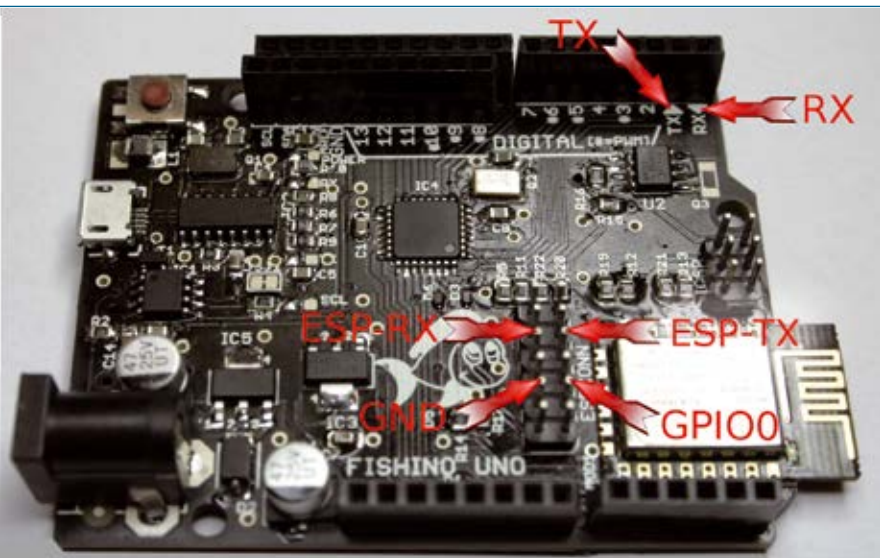


Fig. 1

controllo (cosa peraltro necessaria anche con l'interfaccia seriale) la velocità ottenibile è di almeno un ordine di grandezza superiore a quella della porta seriale, ovvero circa 10 volte tanto;

- controllo di flusso; il protocollo SPI è totalmente gestito dal master (in questo caso l'Atmega). Il controller richiede i dati quando è in grado di elaborarli e non è costretto a "rincorrere" il modulo WiFi;
- possibilità di spostare buona parte dell'impegno sia in termini di elaborazione che di memoria necessari sull'ESP (basti pensare ai 32 K di RAM disponibili contro i 2 dell'Atmega e gli 80/160 MHz di clock contro i 16);
- possibilità di incapsulare tutte le funzionalità del modulo in una libreria dall'uso praticamente identico alle varie WiFi/ethernet standard.

Il firmware realizzato e l'annessa libreria Arduino permettono di ottenere velocità di trasferimento dati (misurata da browser a SD card) di 60÷70 kB/s, quindi leggermente superiore alla velocità ottenibile con la scheda ethernet originale via cavo e decisamente superiori (2÷4 volte tanto) rispetto allo shield WiFi originale; il

software ha comunque ulteriori margini di ottimizzazione che verranno sfruttati in seguito. Futuri aggiornamenti del firmware aggiungeranno ulteriori funzionalità, tra cui l'utilizzo della porta seriale del modulino come seriale hardware aggiuntiva, la possibilità di programmare l'Atmega direttamente via WiFi ed altri.

Passiamo ora alla descrizione dello schema elettrico della sezione WiFi, che contiene alcune particolarità degne di nota dovute principalmente alle problematiche hardware del modulo ESP. Come si nota dal simbolo sullo schema, il modulo contiene varie connessioni denominate GPIO (General Purpose Input Output) utilizzabili per vari compiti a seconda della programmazione e/o dello stato del modulo stesso. I pin sono infatti utilizzabili come gli I/O digitali (e un input analogico) di Arduino, salvo che quasi tutti hanno funzionalità aggiuntive, alcune delle quali utilizzate all'avvio dell'ESP che ne rendono difficoltoso l'utilizzo. Di seguito una breve descrizione di tali pin.

- GPIO0 e GPIO15 : oltre ad essere utilizzabili come input/output digitale, servono per selezionare la modalità

d'avvio al boot del modulo. Quest'ultimo può infatti essere avviato da flash interna (funzionamento normale, GPIO15 a 0 e GPIO0 a 1), da interfaccia seriale, utilizzato per la riprogrammazione del firmware (GPIO15 a 0, GPIO0 a 0) e boot da scheda SD esterna (non l'abbiamo mai utilizzato, GPIO15 a 1, GPIO0 influente). A complicare le cose, il pin GPIO15 è anche utilizzato per selezionare il modulo nella modalità SPI slave, con valore attivo basso. Il modulo deve quindi partire con GPIO15 a 0 e, brevemente dopo l'avvio, questo dev'essere portato a 1 per liberare la porta SPI necessaria anche, ad esempio, per la scheda SD.

- GPIO12 (MISO), GPIO13(MOSI) e GPIO14(SCK), oltre ad essere degli I/O generici, insieme al suddetto GPIO15 sono utilizzati dall'interfaccia SPI.
- GPIO16, I/O generico ma utilizzato per il "risveglio" del modulo dalla funzione di deep sleep. Non lo utilizziamo come tale ma solo come pin di handshake verso l'Atmel. Su questo pin all'avvio del modulo sono presenti degli impulsi di "risveglio" da scaricare prima che acquisti la sua funzionalità definitiva.
- GPIO2, GPIO4 e GPIO5 sono disponibili per l'uso come pins digitali, e saranno sfruttabili in una prossima versione del firmware come fossero estensioni dei pins digitali di Arduino.
- Rx e Tx costituiscono la porta seriale hardware del modulo e sono utilizzati anche in fase di programmazione del firmware. Anche qui, una prossima estensione del firmware ne permetterà l'uso come porta

seriale aggiuntiva che consentirà a Fishino di raddoppiare la sua connettività seriale.

- CH_PD è il pin di abilitazione del modulo. Portandolo a livello alto il modulo risulta abilitato (default), mentre un livello basso mette in stand-by l'ESP riducendone i consumi praticamente a zero.
- RESET è il reset hardware dell'ESP, attivo a livello basso
- ADC è l'ingresso analogico dell'ESP, diretto verso un convertitore A/D da 10 bit (1024 valori possibili).

Volendo utilizzare questi pin bisogna ricordare che non gli vanno applicati più di 3,3 V perché non sono 5V-tolerant. Dello schema elettrico notiamo alcuni particolari, come il diodo D5, che serve quando Fishino (e quindi anche il modulo ESP, vedere di seguito la circuiteria di RESET) viene resettato, a portare a livello 0 il pin GPIO15 forzando quindi l'avvio dalla Flash interna; senza questo accorgimento un valore casuale alto sul GPIO15 impedirebbe l'avvio del modulo. La resistenza R23 a massa sul GPIO16 serve ad indicare che il modulo è impegnato nell'avvio (e quindi il GPIO16 non è ancora stato impostato come handshake); la libreria sfrutta questo segnale per rilevare se il modulo è pronto ad operare dopo il boot. Notate che sono state utilizzate le stesse linee digitali dell'Atmega sfruttate dagli shield WiFi ed ethernet, garantendo quindi la totale compatibilità con eventuali shield aggiuntivi.

Connettore ESP

Su questo connettore vengono riportati alcuni GPIO liberi del modulo ESP (per l'uso come porte) ed è presente lo spazio per alcuni ponticelli di configurazione.

- PIN 1-2: chiudendoli, il modulo ESP viene disattivato completamente (utile nel caso non sia necessario oppure si voglia montare uno shield che va in conflitto con gli I/O utilizzati). È anche possibile connettere il pin 2 ad un'uscita digitale di Fishino e controllare così via software se e quando accendere il modulo WiFi.
- PIN 3: RESET. Un livello zero su questo pin provoca il reset sia del modulo WiFi che dell'Atmega 328.
- PIN 4: ADC. Input del convertitore A/D a 10 bit dell'ESP.
- PIN 5 e 6: ESP-RX ed ESP-TX, rispettivamente (interfaccia seriale hardware del modulo WiFi). Utilizzati anche per riprogrammare il firmware.
- PIN 7-8: GPIO5 e GPIO4. Utilizzabili come pin di I/O digitale.
- PIN 9-10: chiudendoli con un jumper e premendo RESET, il modulo entra in modalità flash, per l'aggiornamento del firmware (vedremo i dettagli più avanti). Il PIN 10

corrisponde anche al GPIO0, utilizzabile come linea di I/O digitale.

- PIN 11-12: chiudendoli con un jumper si abilita la riprogrammazione dell'Atmega tramite WiFi. Al momento non è stato implementato il firmware necessario, ma sarà fatto nelle future versioni scaricabili dal nostro sito web. Il PIN12 corrisponde anche al GPIO2, utilizzabile anche come linea di I/O digitale.

Circuiteria di RESET

La sezione del RESET risulta più complicata rispetto a quella di Arduino originale per i seguenti motivi:

- occorre resettare sia l'Atmega che l'ESP alla pressione del tasto di reset, all'avvio e alla richiesta di programmazione da parte dell'IDE;
- per poter eseguire la programmazione dell'Atmega tramite WiFi, il modulo ESP dev'essere in grado di resettare l'Atmega stesso senza a sua volta auto-resettarsi.

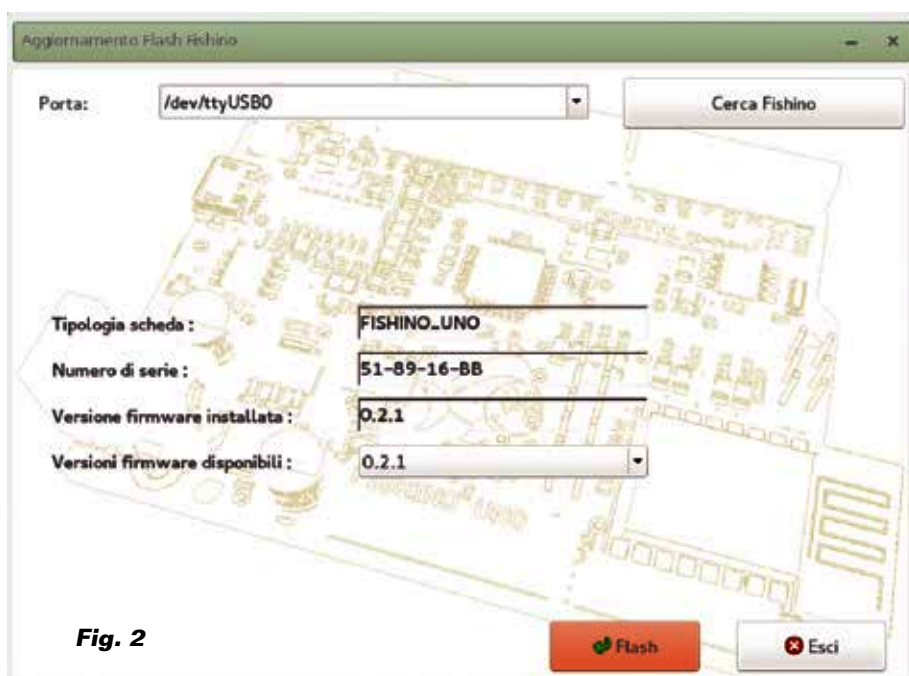


Fig. 2

Listato 1

```
////////////////////////////////////  
// CONFIGURATION DATA          -- ADAPT TO YOUR NETWORK !!!  
// DATI DI CONFIGURAZIONE -- ADATTARE ALLA PROPRIA RETE WiFi !!!  
  
// here put SSID of your network  
// inserire qui lo SSID della rete WiFi  
#define SSID      ""  
  
// here put PASSWORD of your network. Use "" if none  
// inserire qui la PASSWORD della rete WiFi -- Usare "" se la rete non è protetta  
#define PASS      ""  
  
// here put required IP address of your Fishino  
// comment out this line if you want AUTO IP (dhcp)  
// NOTE : if you use auto IP you must find it somehow !  
// inserire qui l'IP desiderato per il fishino  
// commentare la linea sotto se si vuole l'IP automatico  
// nota : se si utilizza l'IP automatico, occorre un metodo per trovarlo !  
#define IPADDR    192, 168, 1, 251  
  
// NOTE : for prototype green version owners, set SD_CS to 3 !!!  
// NOTA : per i possessori del prototipo verde di Fishino, impostare SD_CS a 3 !!!  
const int SD_CS = 4;  
  
// END OF CONFIGURATION DATA  
// FINE DATI DI CONFIGURAZIONE  
////////////////////////////////////
```

Iniziamo dal segnale DTR che esce dall'interfaccia USB/Seriale (U2, CH340G): esso, come anticipato, viene posto a livello basso quando la porta seriale viene aperta. Attraverso il condensatore C6 (scelto da 1 μ F ceramico, contro i 100 nF dell'originale, per allungare l'impulso di reset) viene generato un breve impulso che, passato attraverso il jumper SMD SJ1 (tagliando il quale è possibile disattivare l'auto-reset), raggiunge la linea di "reset esterno", alla quale sono connessi anche il pulsante di reset ed il pin 5 sul connettore di programmazione (ICSP). A differenza del circuito originale, nel Fishino è presente un diodo (D6) tra la linea di RESET ed il pin dell'Atmega. Lo scopo di questo diodo (e del diodo D3 che vedremo in seguito) è di poter resettare solo l'Atmega senza peraltro veicolare il segnale anche all'ESP. Premendo il pulsante RESET, o connettendo la seriale, l'impulso di reset raggiunge sia l'Atmega (attraverso D6) che l'ESP (direttamente), resettandoli entrambi. Un segnale sulla linea ATRES-ESP, generato dall'ESP (nel caso si sia abilitata la riprogrammazione attraverso il WiFi) rag-

giunge attraverso D3 la linea di reset dell'Atmega ma, a causa di D6, non può propagarsi all'ESP stesso. Tramite questo sistema abbiamo quindi dato la possibilità al modulo WiFi di controllare la linea di reset dell'Atmega che, unitamente all'interfaccia SPI, ne permette la riprogrammazione senza nemmeno la necessità di un bootloader precaricato. In pratica, una volta completato lo sviluppo nel firmware, sarà possibile non solo riprogrammare via WiFi l'Atmega ma farlo utilizzando anche lo spazio normalmente riservato al bootloader.

Modulo RTC

Concludiamo lo schema elettrico con il modulo RTC (Real Time Clock), costituito da un classico DS1307 della Maxim, un quarzo a 32 kHz, una batteria di backup ed un paio di resistenze sulla linea I²C. Lo schema è quanto di più classico esista ed è completamente compatibile con le librerie di Arduino esistenti; tutte le funzioni sono gestite tramite linea I²C (SDA/SCL).

Driver USB

Per il converter USB/seriale Fishino richiede driver specifici,

almeno per Windows fino alla versione 7 (si scaricano dal sito www.fishino.it o www.fishino.com, nella sezione Download); pare che dalla 8 in poi i driver siano inclusi. Per l'ambiente Linux, per contro, i driver non sono necessari, essendo già presenti nel kernel.

AGGIORNAMENTO FIRMWARE DEL MODULO WIFI

Fishino viene fornito con la versione del firmware disponibile al momento dell'assemblaggio. Essendo in fase di continuo sviluppo, conviene aggiornare periodicamente il firmware. Le librerie di Arduino disponibili sono infatti aggiornate continuamente in base alle nuove possibilità offerte dal firmware. La procedura di aggiornamento è semplificata da un programma apposito, disponibile sia per Windows che per Linux, che esegue l'operazione in modo completamente automatico e a prova di errore. I passi per l'aggiornamento sono i seguenti.

1. Caricare uno sketch che non utilizzi la porta seriale, come ad esempio BLINK (quello che fa lampeggiare il LED sulla scheda); ciò serve ad evitare interferenze tra lo sketch caricato ed il collegamento seriale tramite l'Atmega e l'ESP. Se il programma di Flash non rileva il Fishino, al 99% il problema è che avete caricato uno sketch sbagliato.
2. Connettere il TX di Fishino con la porta ESP-TX sul connettore ESP, e l'RX di Fishino con la porta ESP-RX sul connettore ESP (vedere Fig. 1).
3. Connettere GPIO0 a massa tramite un cavetto o un ponticello sempre sul connettore ESP (Fig. 1). Collegare Fishino al PC (o premere il pulsante di RESET se già connesso);
4. Lanciare il programma Fishi-

Fig. 3 - Schermata della demo.

noFlasher, assicurandosi che il PC sia connesso ad Internet.

Se i collegamenti sono stati eseguiti correttamente, il programma rileverà la porta a cui è connesso Fishino, determinerà il modello e la versione del firmware attualmente installata, si collegherà ad un server remoto e scaricherà la lista dei firmware disponibili, mostrando l'ultimo e permettendo comunque la selezione delle versioni precedenti nel caso si voglia fare un downgrade, come in Fig. 2. Premendo il pulsante "Flash" verrà avviata la procedura di aggiornamento alla fine della quale apparirà un messaggio di conferma. Per terminare il programma occorre premere il pulsante "Esci".

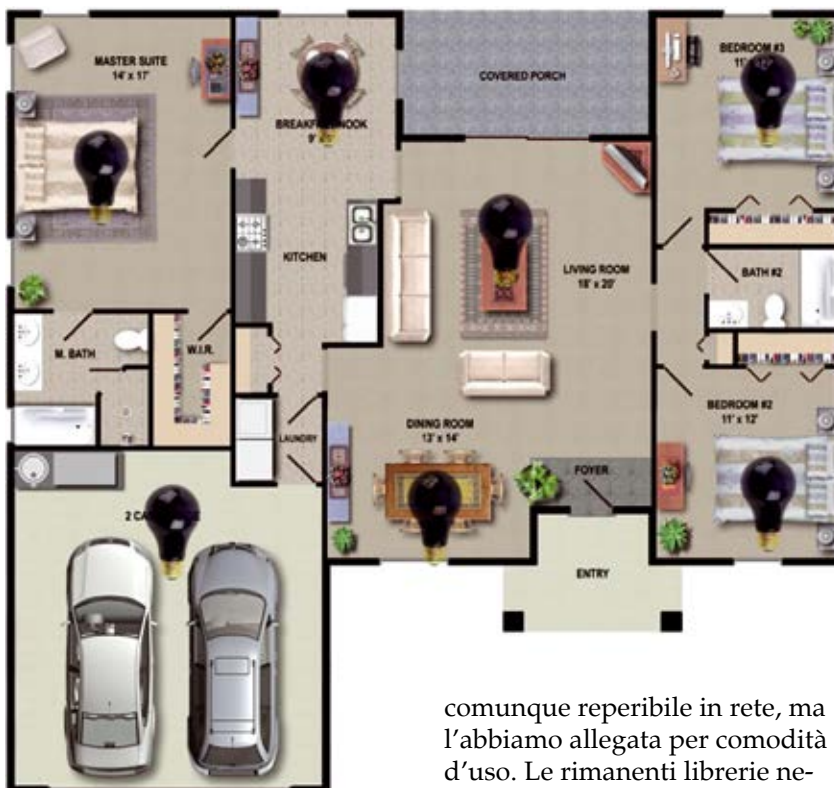
Nel caso Fishino non venga rilevato automaticamente, è possibile provare a selezionare la porta manualmente. È comunque probabile che siano stati commessi degli errori nei collegamenti. La selezione manuale risulta indispensabile nel raro caso in cui più di un Fishino sia connesso contemporaneamente al PC, nel qual caso il primo viene rilevato automaticamente ma resta la possibilità di sceglierne un altro. Una volta terminata la procedura è sufficiente eliminare i tre collegamenti e Fishino sarà pronto per l'uso con il nuovo firmware.

LIBRERIE DISPONIBILI

Attualmente sono disponibili due librerie per la gestione del Fishino: la *Fishino* e la *Fishino WebServer*.

La libreria Fishino è l'esatto equivalente della libreria Ethernet o WiFi di Arduino.

In questa libreria vengono definite le classi FishinoClass (gestione a basso livello, analoga alla EthernetClass o WiFiClass



di Arduino), e le rispettive classi FishinoClient (l'analogo della EthernetClient e WiFiClient) e FishinoServer (EthernetServer e WiFiServer).

Queste classi sono utilizzate in modo pressoché identico alle originali, quindi negli sketch esistenti che utilizzano l'Ethernet o il WiFi basta cambiare il tipo delle varie variabili per ottenerne il funzionamento con il WiFi di Fishino. Le uniche (leggere) differenze sono sull'inizializzazione, essendo il modulo WiFi di Fishino dotato di caratteristiche aggiuntive rispetto al WiFi originale. La libreria FishinoWebServer è il porting su Fishino della nota TinyWebServer; consente la creazione di un completo server web sulla scheda.

È stata inoltre inserita la libreria Flash nel pacchetto di download visto che è utilizzata dalla FishinoWebServer per spostare le costanti nella memoria Flash liberando così la poca RAM a disposizione. Questa libreria è

comunque reperibile in rete, ma l'abbiamo allegata per comodità d'uso. Le rimanenti librerie necessarie sono già disponibili nei vari download dell'IDE di Arduino; sono in particolare necessarie la SD (per la gestione delle schede MicroSD) e la RTCLib per la gestione del modulo RTC con il DS1307, ed altre librerie di sistema. Le librerie sono in continuo sviluppo e presto verranno corredate di funzionalità aggiuntive, in particolar modo per la gestione dei PIN di I/O e della porta seriale aggiuntiva disponibili sul modulo WiFi ed altro.

DEMO HOME AUTO

Per concludere questo articolo presentiamo la demo FishinoHomeAuto che mostra alcune delle caratteristiche più interessanti di Fishino all'opera. Si tratta di un piccolo server web che consente tramite browser la gestione dei pin di I/O digitali sulla board. Ci limiteremo a descriverne brevemente il funzionamento e l'utilizzo affinché possiate provarne la funzionalità immediatamente. Premettiamo che la demo non vuol essere un applicativo completo di home automation ma la

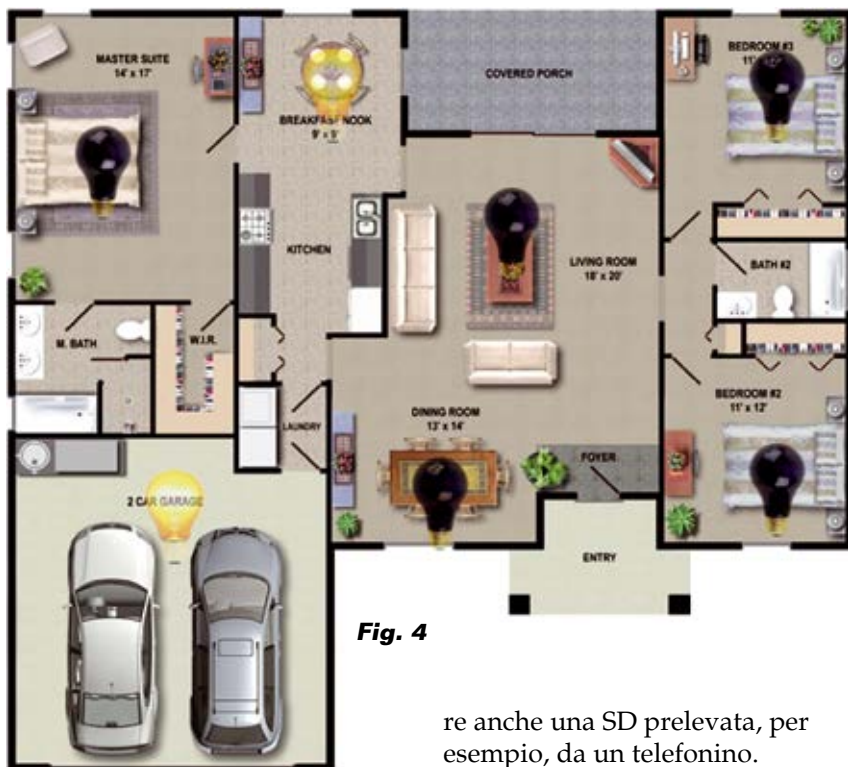


Fig. 4

base per scriverne uno; in particolare, sono gestiti solo output digitali (rappresentati dalle lampadine alle figure seguenti). Il software è stato comunque pensato con l'estensibilità in mente, quindi successivamente verranno implementate le funzioni di input e quelle analogiche. Per prima cosa dovete decomprimere il file *FishinoLibs.zip* nella cartella "libraries". Una volta eseguito avrete tre nuove librerie: Fishino, FishinoWebServer e Flash. Ora decomprimete il file FishinoHomeAuto nella cartella "sketches": apparirà una cartella chiamata FishinoHomeAuto e, dentro questa, alcuni file e la sottocartella **STATIC**. Copiate tutto il contenuto della cartella **STATIC** (non la cartella, ma solo i files in essa contenuti) nella directory radice di una scheda MicroSD. Non occorre cancellare quel che c'è dentro; basta copiare i file nella cartella principale. Lo sketch non altera i dati nella scheda, quindi potrete utilizza-

re anche una SD prelevata, per esempio, da un telefonino. Adesso lanciate l'IDE ed aprite lo sketch FishinoHomeAuto. All'inizio di questo è presente una parte di configurazione, da modificare per adattarla alla propria rete WiFi (**Listato 1**). Leggere i commenti e modificare le impostazioni come indicato. Salvare lo sketch e caricarlo sul Fishino.

A questo punto:

- inserire la MicroSD nel fishino;
- se si desidera vedere cosa succede, aprire il monitor seriale sull' IDE;
- volendo vedere i LED accendersi e spegnersi seguendo i comandi da web, collegare uno o più LED (con le relative resistenze in serie) ai pin di Fishino 2, 5, 6, 8, 9, 14 e 15 (queste sono le uscite digitali previste dalla demo, ognuna delle quali è associata ad una "stanza" nell'immagine che apparirà sul browser);
- premere il pulsante RESET;
- dal PC lanciare il browser e inserire nella barra degli indiriz-

zi l'IP impostato nello sketch. Se avrete optato per l'IP dinamico le cose si complicano, visto che occorre determinare quale IP è stato assegnato a Fishino. Sugeriamo l'impostazione di un IP statico per le prime prove.

Se tutto è andato a buon fine, sul browser verrà visualizzata la pagina con la **Fig. 3**. La pagina è totalmente configurabile tramite i file presenti nella cartella **STATIC** e poi copiati sulla SD card. Cliccando su una delle lampadine spente (nere) l'immagine cambierà in una lampadina accesa (gialla) e contemporaneamente l'eventuale led connesso al Fishino si illuminerà. Cliccando nuovamente sulla lampadina accesa, questa tornerà nera ed il LED si spegnerà (**Fig. 4**). Pur essendo una demo, il programma è sufficientemente configurabile: è possibile per esempio cambiare l'immagine della "casa", la posizione e le immagini delle lampadine, eccetera. Stiamo preparando una versione che preveda la lettura/scrittura di valori analogici, per esempio di temperatura o per regolare un termostato. ■

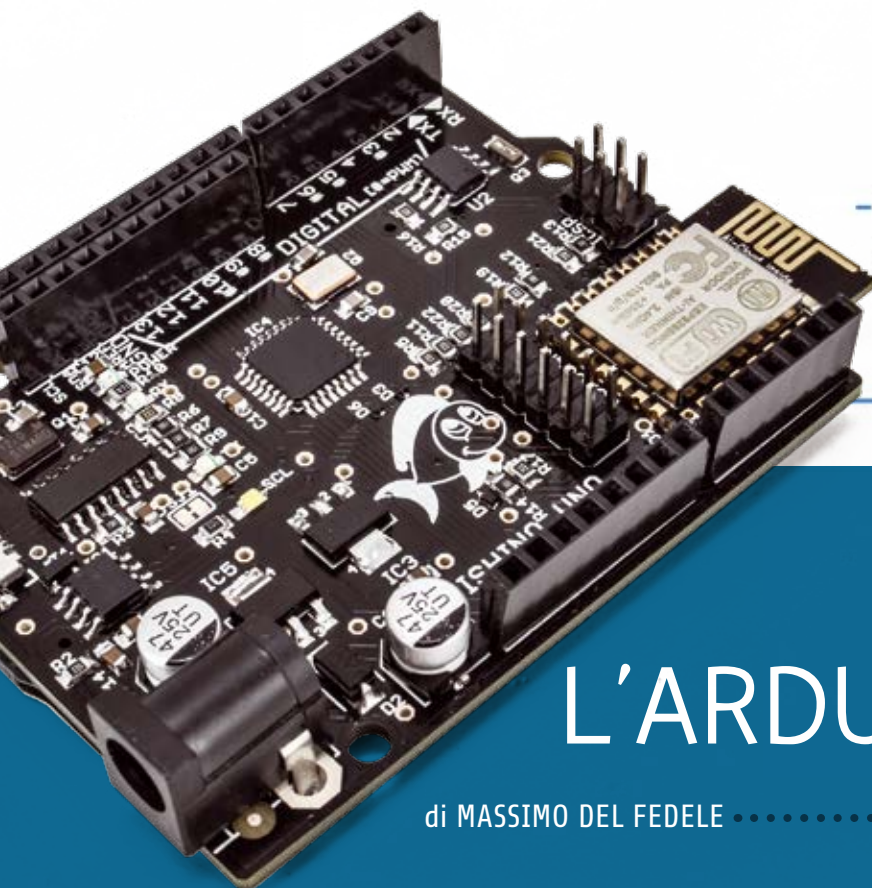


per il MATERIALE

La board Fishino (cod. FISHINOUNO) viene fornita montata e collaudata. Può essere acquistata presso Futura Elettronica al prezzo di Euro 36,00. Il prezzo si intende 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>

Continuamo la presentazione della board Fishino, mostrando le principali funzioni delle librerie e degli esempi d'uso. Seconda puntata.



FISHINO, L'ARDUINO DIVENTA WIRELESS

di MASSIMO DEL FEDELE

Nel numero scorso abbiamo presentato la scheda **Fishino UNO**, una board compatibile con la diffusissima Arduino UNO dotata però di connettività WiFi, slot per microSD ed RTC incorporati.

In questo secondo articolo iniziamo la descrizione delle librerie software disponibili, mostrando le principali funzioni con alcuni semplici esempi d'uso.

Come anticipato, sia il firmware della scheda che le librerie software sono in continua fase di sviluppo, quindi consigliamo di eseguire spesso l'aggiornamento di entrambi.

LE LIBRERIE

Per poter sfruttare tutte le caratteristiche di **Fishino** occorre ovviamente disporre di una serie di librerie software che ne gestiscano tutti i componenti aggiuntivi. Se per la scheda SD card e il Real Time Clock (RTC) esistono già nella suite di Arduino le corrispondenti librerie, questo non vale per il modulo WiFi ESP12, per il quale ne abbiamo sviluppate di apposite.

Inizieremo quindi da queste ultime, fornendo comunque successivamente anche qualche dettaglio su quelle già disponibili nell'IDE.

Le librerie fornite e liberamente scaricabili dal sito sono:

- Libreria '**Fishino**'
- Libreria '**FishinoWebServer**'
- Libreria '**Flash**'

quest'ultima libreria, che abbiamo inserito per comodità nel download pur essendo reperibile in rete, è necessaria per il funzionamento delle due precedenti.

LIBRERIA '**FISHINO**'

Partiamo con la descrizione della libreria (che potete scaricare dal sito della rivista www.elettronica.in.it) che contiene tutta la gestione a basso e medio livello del modulo

WiFi di Fishino.

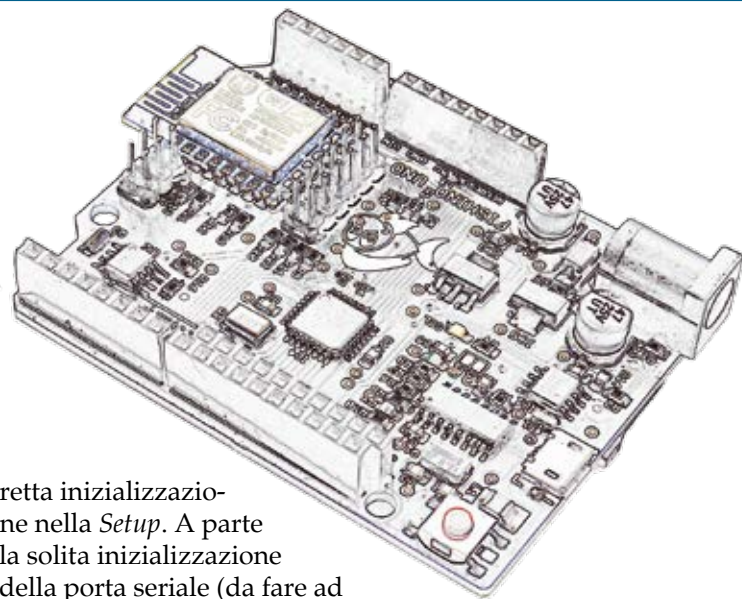
Questa definisce 3 classi:

- **FishinoClass** e la relativa variabile globale **Fishino**
- **FishinoClient**
- **FishinoServer**

Il progetto è in continua evoluzione e ben supportato da una comunità molto attiva anche su FaceBook (<https://www.facebook.com/groups/fishino>); anche grazie a questa in seguito verranno aggiunte la **FishinoUdp**, **FishinoAnalog**, **FishinoDigital** e **FishinoSerial** per gestire rispettivamente le comunicazioni internet tramite socket Udp, l'ingresso analogico, gli I/O digitali e la porta seriale hardware aggiuntiva presenti sul modulo WiFi.

Iniziamo la descrizione della classe **FishinoClass** (istanza singola nella variabile globale **Fishino**), con degli esempi pratici di utilizzo delle varie funzioni. Con **bool Fishino.reset()** si inizializza il modulo WiFi inviandogli un reset software. Obbligatorio ad inizio sketch per garantire un avvio corretto del modulo. Ritorna TRUE se il modulo è stato correttamente inizializzato, FALSE altrimenti.

La funzione di reset esegue inoltre un controllo sulla versione del firmware installata. In caso di versione troppo datata viene inviato un messaggio di errore alla porta seriale ed il programma viene bloccato. Nel **Listato 1** vediamo un esempio di cor-



retta inizializzazione nella *Setup*. A parte la solita inizializzazione della porta seriale (da fare ad inizio setup), si notano:

- l'inizializzazione dell'interfaccia SPI
- il **Fishino.reset()** dell'inizializzazione del modulo

La prima è stata lasciata volutamente manuale per poter cambiare la velocità di comunicazione, nel caso siano presenti altri shields che utilizzano la stessa interfaccia. In questo caso si è impostata la massima velocità disponibile.

La sezione contenente la chiamata **Fishino.reset()** inizializza il modulo e visualizza un messaggio di corretta inizializzazione sulla seriale o, in caso di problemi, visualizza l'errore e blocca lo sketch. Attenzione, il modulo WiFi **NON** parte senza que-

Listato 1

```
void setup()
{
  // apre la porta seriale e ne attende l'apertura
  // consigliabile da eseguire come primo comando per poter visualizzare
  // eventuali messaggi di errore sul monitor seriale
  Serial.begin(115200);

  // attende l'apertura della porta seriale.
  // Necessario solo per le boards Leonardo
  while (!Serial)
    ;

  // inizializza il modulo SPI
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV2);

  // resetta e testa il modulo WiFi
  if(Fishino.reset())
    Serial.println("Fishino WiFi RESET OK");
  else
  {
    Serial.println("Fishino RESET FAILED");

    // attende per sempre
    while(true)
      ;
  }

  Serial.println("Fishino WiFi web server");

  .....<resto dello sketch>...
```

Listato 2

```
...<parte precedente dello sketch>...

Fishino.setMode(STATION_MODE);

...<resto dello sketch>...
```

Listato 3

```
...<parte precedente dello sketch>...
while(true)
{
  if(Fishino.begin("MIO_SSID", "MIA_PASSWORD")) {
    Serial.println("Connected to MIO_SSID");
    break;
  }
  else {
    Serial.println("Failed connecting to MIO_SSID");
    Serial.println("Retrying.....");
  }
}
...<resto dello sketch>...
```

sto comando.

Le funzioni **bool Fishino.setMode(uint8_t mode)** e **uint8_t Fishino.getMode(void)** impostano (o leggono) la modalità di funzionamento del modulo (Listato 2), che può essere una delle seguenti:

- **STATION_MODE**
modalità stazione. Richiede la presenza di un router WiFi a cui connettersi. È la modalità normale.
- **SOFTAP_MODE**
Permette la creazione di un access point a cui connettersi. Utile in mancanza di un'infrastruttura di rete esistente.
- **STATIONAP_MODE**
Modalità doppia, il modulo funziona sia da stazione, collegandosi ad un router esistente, che da access point.

Per eseguire la connessione all'access point/router si utilizza **bool Fishino.begin(SSID, PASSWORD)**, dove al posto di SSID va inserito il punto di accesso e al posto di PASSWORD la chiave per accedervi (quest'ultima può essere una stringa vuota se non è richiesta).

Per controllare se la board Fishino è correttamente connessa il comando è **uint8_t Fishino.status()**. La funzione ritorna TRUE se la connessione ha avuto successo, FALSE altrimenti. Nello spezzone di codice presente nel Listato 3 viene tentata la connes-

CARATTERISTICHE TECNICHE

- Alimentazione: 12 Vcc o USB
- Completamente compatibile con Arduino Uno
- Scheda WiFi a bordo, con possibilità di funzionamento in modalità stazione, access point o entrambe contemporaneamente
- Interfaccia per schede di memoria MicroSD a bordo
- RTC (Real Time Clock) a bordo con batteria al litio di mantenimento
- Sezione di alimentazione a 3,3 V potenziata
- Connettore aggiuntivo sfalsato in modo da risolvere il problema dell'incompatibilità di Arduino con le schede millefori.

Listato 4

```
uint32_t connectTime;
void setup()
{
    ....<parte precedente dello sketch>....
    connectTime = millis();
}

void loop()
{
    // controlla la connessione ogni 10 secondi
    if(millis() - connectTime > 10000) {

        // resetta il tempo
        connectTime = millis();

        // controlla se connesso
        uint8_t stat = Fishino.status();

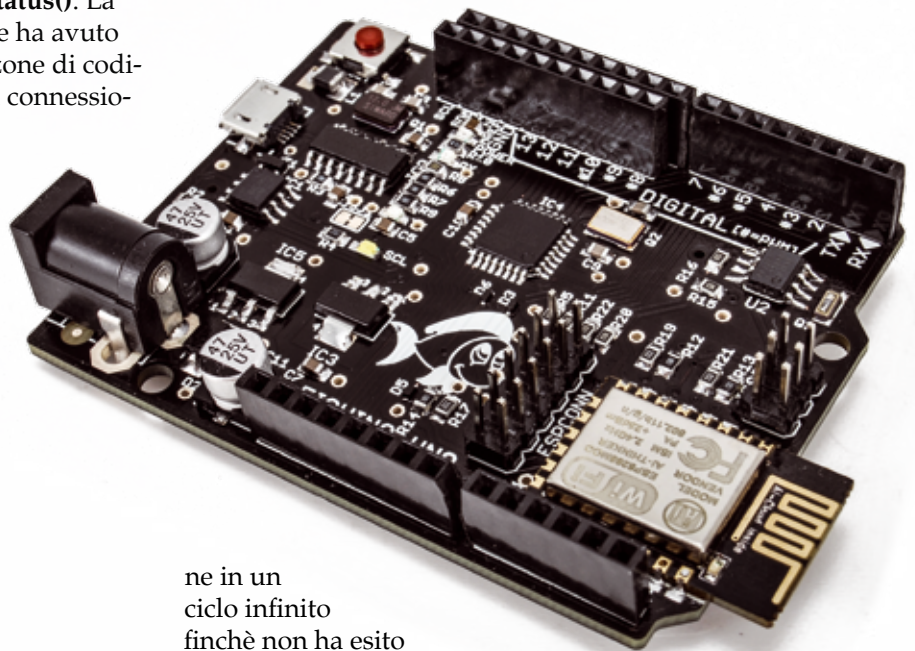
        // se non connesso, tenta la riconnessione
        if(stat != STATION_GOT_IP) {
            if(Fishino.begin("MIO_SSID", "MIA_PASSWORD"))
                stat = STATION_GOT_IP;
        }

        // se connesso, salva i dati sul server
        if(stat == STATION_GOT_IP) {
            salvaDatiSulServer(); // funzione da definire!!!
        }

        // qui legge i sensori e li memorizza sulla SD
        leggiSensoriEMemorizza(); // funzione da definire!!!
    }
}
```

Listato 5

```
Fishino.config(IPAddress(192, 168, 1, 251));
```



ne in un ciclo infinito finché non ha esito positivo.

Questo tipo di connessione (eseguita nella *Setup*) è adatto ad una postazione fissa, ovviamente. Nel caso si utilizzi il Fishino in mobilità, è consigliabile spostare la connessione nel *loop()* e tentarla ogni tanto mentre si fanno altre attività.

In questo modo è possibile, ad esempio, raccogliere

Listato 6

```
Serial.print("Il mio IP è : ");  
Serial.println(Fishino.localIP());
```

dei dati da un sensore, memorizzarli sulla scheda SD e, quando viene rilevata una connessione funzionante, inviarli ad un computer remoto (**Listato 4**). In questo esempio (volutamente abbreviato), nella *setup()* viene letto il tempo corrente e salvato nella variabile *connectTime*; successivamente nel *loop()* viene controllato il tempo passato (*millis()* - *connectTime*) e quando questo supera i 10 secondi viene eseguito un test sulla connessione; se non connesso si tenta la connessione al server e, in caso di successo viene eseguita una funzione (da definire) che salva in rete i dati letti in precedenza.

Il loop continua successivamente tramite un'altra funzione (anch'essa da definire) che legge qualche sensore e memorizza i dati localmente, ad esempio su una scheda SD.

Con uno sketch simile è possibile quindi realizzare un semplice datalogger che non solo legge e memorizza su scheda SD i dati ma che, in presenza di una connessione di rete, è in grado di salvarli in modo totalmente automatico ad intervalli di tempo predefiniti.

Per configurare un IP statico ed eventualmente i servers DNS, il gateway e la subnet della rete locale si utilizzano queste funzioni:

```
bool Fishino.config(IPAddress local_ip)  
bool Fishino.config(IPAddress local_ip, IPAddress  
dns_server)  
bool Fishino.config(IPAddress local_ip, IPAddress  
dns_server, IPAddress gateway)  
bool Fishino.config(IPAddress local_ip, IPAd-  
dress dns_server, IPAddress gateway, IPAddress  
subnet)
```

In pratica, la prima è adoperata per impostare un IP statico, se necessario.

Nel **Listato 5** vediamo come è possibile impostare un IP statico su 192.168.1.251

Se non utilizzata l'IP sarà richiesto dinamicamente al router.

E' ovviamente possibile anche disconnettersi dalla rete WiFi. Il comando per eseguire questa operazione è **bool Fishino.disconnect(void)**.

Qui di seguito, invece, alcune funzioni utilizzate per controllare i parametri della connessione, in particolare per leggere il MAC del modulo WiFi la funzione è **const uint8_t* Fishino.macAddress(void)**

Mentre per la lettura dell'IP acquisito dal modulo WiFi (utile nel caso si sia impostato un IP dinamico) il comando da richiamare è **IPAddress Fishino.localIP()** come mostrato ad esempio nel **Listato 6**. Per leggere la maschera della sottorete e dell'indirizzo IP del gateway potete richiamare queste funzioni:

IPAddress Fishino.subnetMask()

IPAddress Fishino.gatewayIP()

Le funzioni indicate sopra sono state nominate in modo assolutamente simile a quelle delle analoghe funzioni delle librerie Ethernet e WiFi di Arduino,

Listato 8

```
Serial.print("Sono connesso a : ");  
Serial.println(Fishino.SSID());
```

per poter semplificare il porting del codice esistente. Tuttavia le potenzialità superiori di **Fishino**, ed in particolar modo la possibilità di funzionare anche in modalità Access Point senza bisogno di un'infrastruttura esistente, hanno reso necessario studiare nuove funzioni per quanto riguarda la modalità **Stazione** tra cui:

```
bool Fishino.setStaIP(IPAddress ip)  
bool Fishino.setStaMAC(uint8_t const *mac)  
bool Fishino.setStaGateway(IPAddress gw)  
bool Fishino.setStaNetMask(IPAddress nm)
```

Mentre per la modalità **Access Point** sono state create:

```
bool Fishino.setApIP(IPAddress ip)  
bool Fishino.setApMAC(uint8_t const *mac)  
bool Fishino.setApGateway(IPAddress gw)  
bool Fishino.setApNetMask(IPAddress nm)  
bool Fishino.setApIPInfo(IPAddress ip, IPAd-  
dress gateway, IPAddress netmask)
```

In particolare, l'ultima permette di impostare tutti i parametri IP del **Fishino** utilizzato come router

Listato 7

```
Fishino.setApIPInfo(  
    IPAddress(192, 168, 100, 1), // IP del Fishino  
    IPAddress(192, 168, 100, 1), // Gateway del Fishino, solitamente come l'IP  
    IPAddress(255, 255, 255, 0) // Netmask (maschera di sottorete) del Fishino  
);
```

Listato 9

```
uint8_t n = Fishino.scanNetworks();
if(n){
    Serial.print("Trovate ");
    Serial.print(n);
    Serial.println(" reti wifi:");
    for(int i = 0; i < n; i++) {
        Serial.print("Rete #");
        Serial.print(i);
        Serial.print(" : ");
        Serial.println(Fishino.SSID(i));
    }
}
else
    Serial.println("Nessuna rete WiFi trovata");
```

WiFi in un comando singolo (**Listato 7**).

Vedremo come usarle a fine articolo con un esempio completo. Per poter leggere i dati della connessione WiFi, quali lo SSID del router a cui ci si è connessi, il MAC del medesimo (BSSID), la potenza in dBm del segnale (RSSI) ed il tipo di protezione della rete, potete utilizzare queste funzioni:

const char* Fishino.SSID()

const uint8_t* Fishino.BSSID()

int32_t Fishino.RSSI()

uint8_t Fishino.encryptionType()

Come mostrato ad esempio nel **Listato 8**.

Esistono poi alcune funzioni utilizzate per eseguire una lista delle reti WiFi disponibili con le loro caratteristiche:

uint8_t Fishino.scanNetworks()

Questa operazione esegue una scansione delle reti WiFi disponibili e ritorna il numero di reti trovate. Una volta eseguita la *scanNetworks*, è possibile utilizzare le seguenti funzioni, che hanno come parametro il numero della rete da esaminare (numero di reti ritornate da *scanNetworks()* - 1).

La funzione **const char* Fishino.SSID(uint8_t net-**

workItem) ritorna invece lo SSID, ovvero il nome della rete richiesta come mostrato nel **Listato 9**.

Questo esempio stampa sulla seriale un'elenco delle reti wireless trovate.

Per sapere il tipo di protezione della rete il comando da usare è **uint8_t Fishino.encryptionType(uint8_t networkItem)**.

E' possibile anche sapere la potenza del segnale della rete richiesta con **int32_t Fishino.RSSI(uint8_t networkItem)**.

Nella classe **FishinoClass** sono presenti altre funzioni meno utilizzate che tralasciamo per brevità. Il codice della libreria è comunque ben commentato e di facile interpretazione.

CLASSI FISHINOCIENT E FISHINOSERVER

Queste due classi sono l'equivalente delle EthernetClient/WiFiClient ed EthernetServer/WiFiServer delle shield ethernet e WiFi di Arduino, e l'uso è praticamente identico.

Ad esempio, per inviare una richiesta ad una pagina web e stampare sulla seriale la risposta vediamo il **Listato 10**.

Listato 10

```
// tenta la connessione al server
FishinoClient client;
if (client.connect("www.google.com", 80)) {

    Serial.println("connected to server");

    // esegue un request Http
    client.println("GET /search?q=arduino HTTP/1.1");
    client.println("Host: www.google.com");
    client.println("Connection: close");
    client.println();

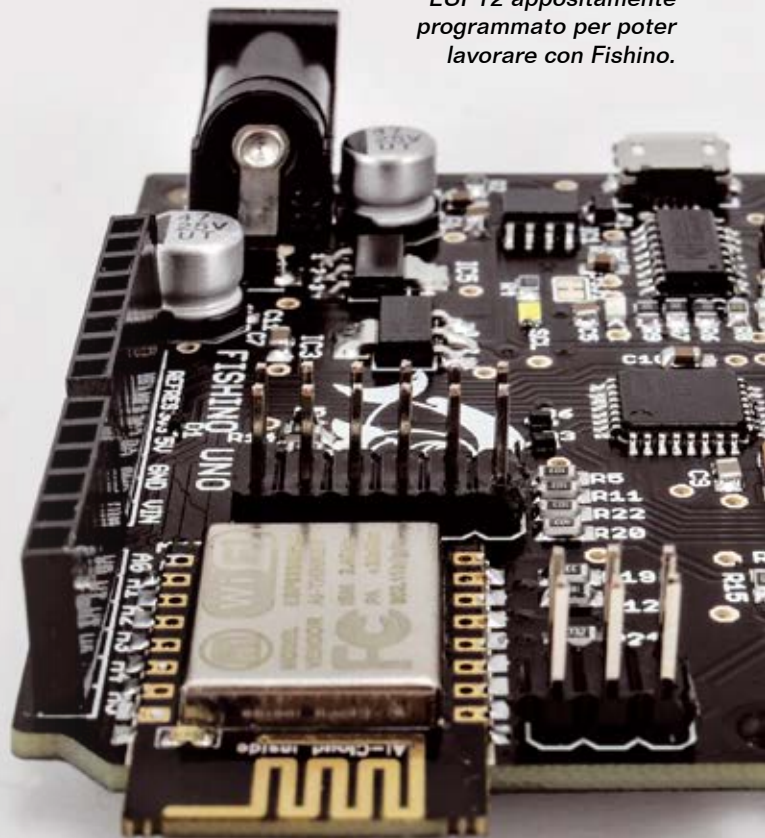
    // legge la risposta finchè il client resta connesso
    do {
        // finchè ci sono bytes in arrivo.....
        while (client.available()) {

            // legge un carattere dal server
            char c = client.read();

            // .... e lo invia alla seriale
            Serial.write(c);

        }
        while(client.connected());
        Serial.println("Client disconnected");
    }
}
```

*Dettaglio del modulo WiFi
ESP12 appositamente
programmato per poter
lavorare con Fishino.*



```
#include <Flash.h>
#include <FishinoUdp.h>
#include <FishinoSockBuf.h>
#include <Fishino.h>
#include <SPI.h>

// CONFIGURAZIONE SKETCH -- ADATTARE ALLA PROPRIA RETE WiFi //
// WiFi SSID e PASSWORD
// potete cambiarle entrambe, verranno utilizzate
// per la creazione dell'infrastruttura WiFi
#define My_SSID "FISHINO"
#define My_PASS ""
// FINE CONFIGURAZIONE

// crea un server in ascolto sulla porta 80 (HTTP standard)
FishinoServer server(80);

void setup()
{
    // apre la porta seriale
    Serial.begin(115200);

    // attende l'apertura della porta seriale.
    // Necessario solo per le boards Leonardo
    while (!Serial)
        ;

    // inizializza il modulo SPI
    SPI.begin();
    SPI.setClockDivider(SPI_CLOCK_DIV2);

    // resetta e testa il modulo WiFi
    if(Fishino.reset())
        Serial << F("Fishino WiFi RESET OK\r\n");
    else
    {
        Serial << F("Fishino RESET FAILED\r\n");

        // attende per sempre in caso di errore
        while(true)
            ;
    }

    Serial << F("Fishino WiFi AP web server\r\n");

    // imposta la modalità SOFT AP (crea una rete autonoma)
    Fishino.setMode(SOFTAP_MODE);

    // ferma il server DHCP, necessario per impostare l'IP della rete
    Fishino.softApStopDHCPserver();

    // imposta i parametri IP dell'access point
    // in questo caso la rete viene creata su 192.168.100.0-255
    // ed il Fishino assume l'IP 192, 168, 100, 1
    Fishino.setApIPInfo(
        IPAddress(192, 168, 100, 1), // IP
        IPAddress(192, 168, 100, 1), // gateway
        IPAddress(255, 255, 255, 0) // netmask
    );

    // imposta i parametri di connessione WiFi, ovvero nome della rete(SSID)
    // e password. Se non avete modificato l'esempio, la rete sarà chiamata FISHINO
    // e sarà una rete aperta, senza password
    Fishino.softApConfig(My_SSID, My_PASS, 1, false);

    // riavvia il server DHCP in modo da poter fornire gli indirizzi
    // in automatico a tutte le stazioni che si connettono
    Fishino.softApStartDHCPserver();

    // inizia l'attesa delle connessioni
    server.begin();
}

void loop()
{
    // attende nuovi clienti
    FishinoClient client = server.available();

    if (client)
    {
        Serial.println("new client");

        // ogni richiesta http termina con una linea vuota
        boolean currentLineIsBlank = true;
        while (client.connected())
        {
            if (client.available())
            {
                char c = client.read();
                Serial.write(c);
            }
        }
    }
}
```

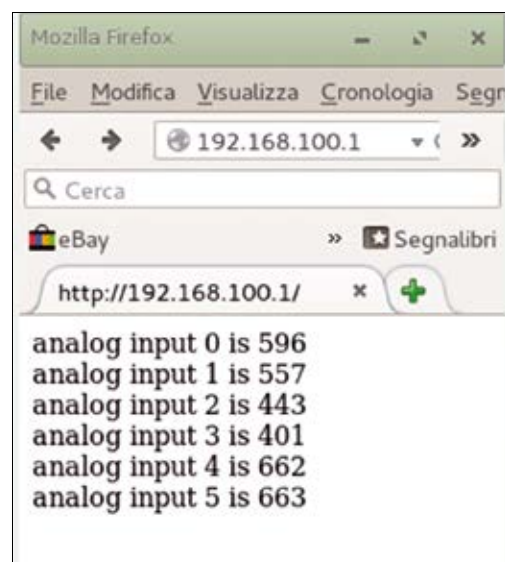
A conclusione dell'articolo, presentiamo un esempio completo che mostra una delle caratteristiche più interessanti di **Fishino**, ovvero la possibilità di creare una propria infrastruttura di rete senza

bisogno di un router esterno, alla quale connettersi in mobilità, per esempio con un cellulare. Un'applicazione simile potrebbe essere usata, ad esempio, per monitorare alcuni sensori all'aperto

Fig. 1



Fig. 2




```

// se si è arrivati a fine linea (carattere 'newline' ricevuto
// e la linea è vuota, la richiesta http è terminata
// quindi è possibile inviare una risposta
if (c == '\n' && currentLineIsBlank)
{
    // invia uno header standard http
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println("Connection: close"); // la connessione verrà chiusa automaticamente una volta inviata la risposta
    client.println("Refresh: 5"); // aggiorna la pagina automaticamente ogni 5 secondi
    client.println();
    client.println("<!DOCTYPE HTML>");
    client.println("<html>");

    // invia il valore di tutti i pins analogici
    for (int analogChannel = 0; analogChannel < 6; analogChannel++)
    {
        int sensorReading = analogRead(analogChannel);
        client.print("analog input ");
        client.print(analogChannel);
        client.print(" is ");
        client.print(sensorReading);
        client.println("<br />");
    }
    client.println("</html>");
    break;
}
if (c == '\n')
{
    // inizio di una nuova linea
    currentLineIsBlank = true;
}
else if (c != '\r')
{
    // sono stati ricevuti dei caratteri nella linea corrente
    currentLineIsBlank = false;
}
}

// lascia tempo al browser per ricevere i dati
delay(1);

// chiudi la connessione
client.stop();
Serial.println("client disconnected");
}
}

```

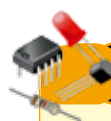
tramite un cellulare da una certa distanza, realizzando così dispositivi completamente portatili. Un'altra interessante applicazione potrebbe essere un comando remoto via WiFi sempre tramite browser web sul cellulare.

L'esempio, nel **Listato 11** crea una rete WiFi "volante", con nome (SSID) **FISHINO**, senza password (rete open) ed avvia un piccolo server che su richiesta fornisce una lettura dei sei ingressi analogici di Fishino. Una volta lanciato lo sketch, occorre selezionare la rete wireless **FISHINO** tra le reti disponibili (**Fig. 1**) e aprendo l'indirizzo **192.168.100.1** sul browser si ottiene il risultato visualizzato in **Fig. 2**. Gli esempi qui riportati sono comunque contenuti, insieme ad altri, nella libreria **Fishino**.

Un'ultima nota sugli I/O occupati dalle estensioni, e che non vanno utilizzati come I/O quando sono attivi i componenti aggiuntivi. Il modulo **WiFi** utilizza i seguenti pins: **7, 10, 11, 12 e 13**. È disattivabile completamente con un ponticello tra il pin **CH_PH** del connettore **ESP** e la massa. La scheda **microSD** utilizza i seguenti pins: **4, 11, 12 e 13** ed impone che il pin **7** sia impostato ad output digitale. Per libe-

rare i ports usati basta non inserire alcuna scheda nel connettore. Il modulo **RTC** comunica via i2c utilizzando i pins **SCL** e **SDA**, abbinati nell'UNO ai ports analogici **A4** ed **A5**.

Continueremo in un prossimo articolo con la descrizione della libreria **FishinoWebServer** che permette la realizzazione di un piccolo ma completo server web, che è la base dell'esempio di Home Automation mostrato in breve nel numero scorso. ■



per il MATERIALE

La board Fishino (cod. FISHINOUNO) viene fornita montata e collaudata. Può essere acquistata presso Futura Elettronica al prezzo di Euro 36,00. Il prezzo si intende 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>

FISH'N TWEET

Controlliamo le uscite di uno shield di espansione Arduino montato sulla nostra Fishino Uno, tramite Twitter.

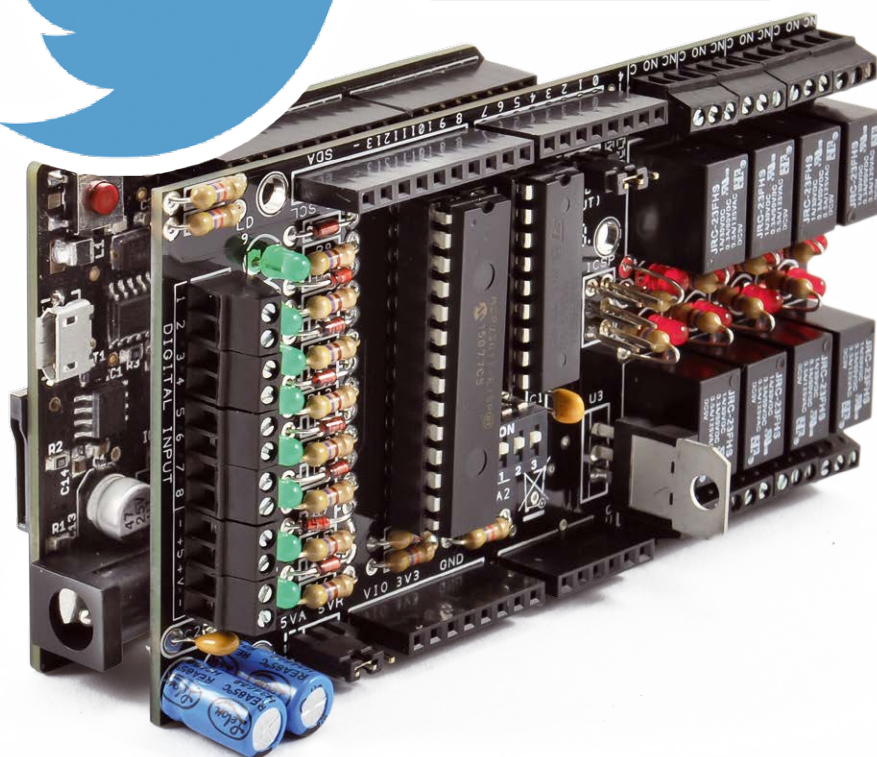


Elettronica In
@elettronica_in

#fishino 24 ON

Reply Elimina Favorite Altro

21 ottobre 2015 12:36



di MASSIMO DEL FEDELE

Avete conosciuto Fishino, il nostro Arduino dotato di connettività WiFi ed RTC, nel fascicolo n° 199; ora è venuto il momento di vedere un'applicazione pratica della scheda, che permette, in modo molto economico e semplice, la gestione remota di apparecchiature elettriche quali lampadine, caldaie, condizionatori, antifurto, eccetera. L'unicità del sistema è costituita dal fatto di non richiedere apparecchiature di rete o server remoti esterni, ma solo una scheda Fishino UNO ed eventualmente uno o più shield per la gestione dei carichi applicati, che nel nostro caso

Ricerca avanzata

Fig. 1

Parole

Tutte queste parole

Questa frase esatta

Una di queste parole

Nessuna di queste parole

Questi hashtag

Scritto in

Persone

Da questi utenti

Fig. 2



sono I/O Expander basati su MCP23017 (descritti nel fascicolo n° 181, cui rimandiamo per gli approfondimenti del caso). Questo progetto, non realizzabile con un semplice Arduino nemmeno se dotato di shield WiFi o Ethernet Shield, è stato reso possibile grazie alla capacità di Fishino di effettuare connessioni sicure con il protocollo https, disponibili solitamente su apparecchi di livello superiore e decisamente più costosi; restare in tema Arduino, attualmente per realizzare un progetto equivalente occorrono una YUN e un software decisamente più complesso. Prima di entrare nel merito del progetto vediamo i vari modi per controllare apparecchiature a distanza.

Controllo via SMS

Vantaggi:

- il ricevitore può essere reso portatile ed indipendente da infrastrutture di rete;
- risposta pressoché istantanea;
- gestibile da qualsiasi cellulare, anche non recente;
- portata limitata solo dalla copertura radiomobile.

Svantaggi:

- richiede una SIM sull'apparecchio controllato, che solitamente scade se non usata;
- richiede un dispositivo GSM, non economicissimo, per la ricezione;
- richiede una copertura GSM, cosa non sempre scontata.

Controllo via moduli radio

Vantaggi:

- il ricevitore può essere reso portatile e indipendente da infrastrutture di rete;
- risposta pressoché istantanea;
- molto economico.

Svantaggi:

- portata molto limitata e fortemente influenzata dalla presenza di eventuali ostacoli.

Controllo via Internet

Vantaggi:

- può essere reso portatile al costo di un ricevitore GPRS;
- può essere effettuato pressoché da qualsiasi PC o telefono cellulare;
- molto economico, se presso la destinazione è presente un'infrastruttura di rete;
- può diventare bidirezionale senza costi aggiuntivi e senza particolare difficoltà.

Svantaggi:

- richiede la presenza di un'infrastruttura di rete in ricezione o un costo aggiuntivo per il GPRS;
- la stabilità dipende dalla connessione Internet;
- i tempi di accesso possono essere variabili a seconda della connessione e/o del metodo utilizzato;
- spesso richiede ricevitori complessi e/o un computer di appoggio;
- richiede un IP pubblico sull'apparecchio ricevente o l'uso di un servizio intermedio di terze parti;
- possono esserci problemi di sicurezza che richiedono accorgimenti particolari per evitare utilizzi non autorizzati.

Con questo progetto abbiamo cercato di unire i vantaggi del controllo via Internet, eliminando alcuni degli svantaggi di tale tecnologia, sfruttando un servizio ben noto a tutti: Twitter. In pratica, i dispositivi vengono azionati inviando un semplice tweet con un formato particolare, dall'utente preimpostato (o più utenti, con qualche piccola modifica al software). Con questo sistema risolviamo le seguenti problematiche del controllo via Internet:

- la stabilità della connessione; nel caso la connessione sia momentaneamente assente, il comando viene solo posticipato ed eseguito al suo ripristino;
- il ricevitore risulta largamente semplificato, dal lato sia hardware che software;
- non è richiesto alcun IP pubblico né un servizio di rete intermedio;
- la sicurezza è garantita dall'infrastruttura di Twitter, che lavora con connessioni sicure.

Come per tutte le cose, anche questo sistema non è esente da svantaggi, che sono principalmente la non immediatezza dell'attuazione (il tempo di propagazione di un tweet è di qualche secondo fino ad un paio di decine di secondi) ed alcune limitazioni (aggirabili) sull'invio di tweet duplicati.

Vediamo innanzitutto l'utilizzo, per poi passare alla descrizione del sistema.

Il controllo avviene tramite l'invio di un Tweet con il seguente formato:

#hashtag porta ON/OFF

ove **'#hashtag'** è un hashtag impostabile nel software (nel nostro caso abbiamo preimpostato **#fishino**), **'porta'** è un numero di porta da attivare (vedremo i dettagli in seguito) e **'ON'** o **'OFF'** sono i comandi di accensione/spengimento della medesima. Esempio:

#fishino 24 ON accende l'uscita 24
#fishino 10 OFF spegne l'uscita 10

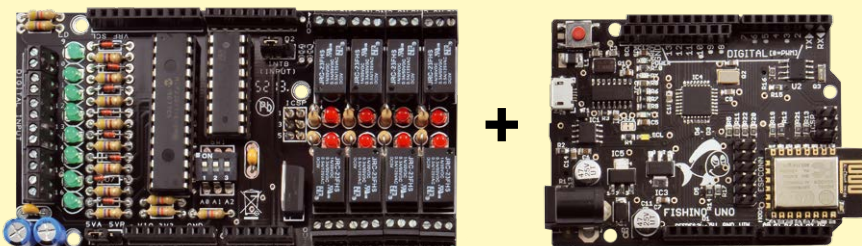
Come detto sopra, tra l'invio del tweet e l'esecuzione del comando passa un tempo stimabile in 5-20 secondi, dipendente dalla congestione della rete di Twitter. L'altro svantaggio di cui parliamo sopra è l'impossibilità di

Il progetto si propone di controllare utilizzatori elettrici o gestibili elettricamente tramite l'invio di tweet contenenti le istruzioni per governare i relé di schede di I/O applicabili a Fishino: si tratta di shield FT1079 della Futura Elettronica, sviluppati intorno all'I/O Expander MCP23017 della Microchip. Questo integrato dispone in totale di 16 pin (suddivisi a livello logico in 2 diverse porte A e B) configurabili individualmente come ingresso o uscita. Nello shield gli 8 pin della porta A sono impostati come uscite e gli 8 pin della porta B come ingressi. Le uscite pilotano gli ingressi di un line-driver ULN2803, progettato appositamente per pilotare carichi induttivi come le bobine dei relé (infatti al suo interno dispone di diversi stadi a transistor dotati di diodi di protezione della giunzione base-collettore). Gli ingressi sono invece collegati ad apposite sezioni circuitali che realizzano gli input digitali. Ogni ingresso dispone di un

diodo di protezione, una resistenza di pull-up e un LED che ne segnala lo stato. Comunque in questa applicazione non usiamo gli input.

La gestione dell'MCP23017 avviene da I²C Bus, realizzato con i piedini 12 (SCL) e 13 (SDA), i quali sono collegati ai relativi pin di Fishino. 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 perché non ci serve acquisire lo stato degli ingressi. L'indirizzo I²C dell'MCP23017 viene selezionato tramite il dip-switch SW1.

Lo shield si alimenta tramite il Vin di Fishino, grazie ad un regolatore interno, basato su un L78L05, che ricava 5 V stabilizzati; per attivare questa opzione, il jumper JP5V va chiuso su 5VR. Chiudendolo su 5VA, lo shield riceve i 5 V direttamente dalla linea +5V di Fishino; ciò è possibile solo se il numero di schede è limitato a due.



inviare tweet duplicati in breve sequenza; per esempio:

#fishino 24 ON accende l'uscita 24
#fishino 24 OFF spegne l'uscita 24
#fishino 24 ON accende l'uscita 24

Il terzo tweet non viene accettato ma segnalato con un "hai già inviato questo tweet". A ciò si può ovviare molto semplicemente aggiungendo qualche carattere a caso dopo la scritta ON:

#fishino 24 ON 123456 accende l'uscita 24

I caratteri supplementari (123456) vengono trascurati dal ricevitore.

FUNZIONAMENTO

Partiamo dall'invio del tweet, che viene memorizzato e propagato

da Twitter. Chiunque può vedere, ricevere e cercare il tweet nella rete da qualsiasi parte del mondo. Utilizzando un PC o un telefono cellulare si può effettuare la ricerca, per esempio, impostando come parametri il mittente del tweet e l'hashtag (**Fig. 1**).

Il risultato della ricerca è qualcosa di simile alla **Fig. 2**.

Come si nota, vengono mostrati i tweet corrispondenti al criterio di ricerca, dall'ultimo arrivato a quelli precedenti.

Analizziamo un po' più in dettaglio un singolo tweet; ogni messaggio contiene:

- un mittente (@mdelfede in questo caso);
- nessuno, uno o più hashtag

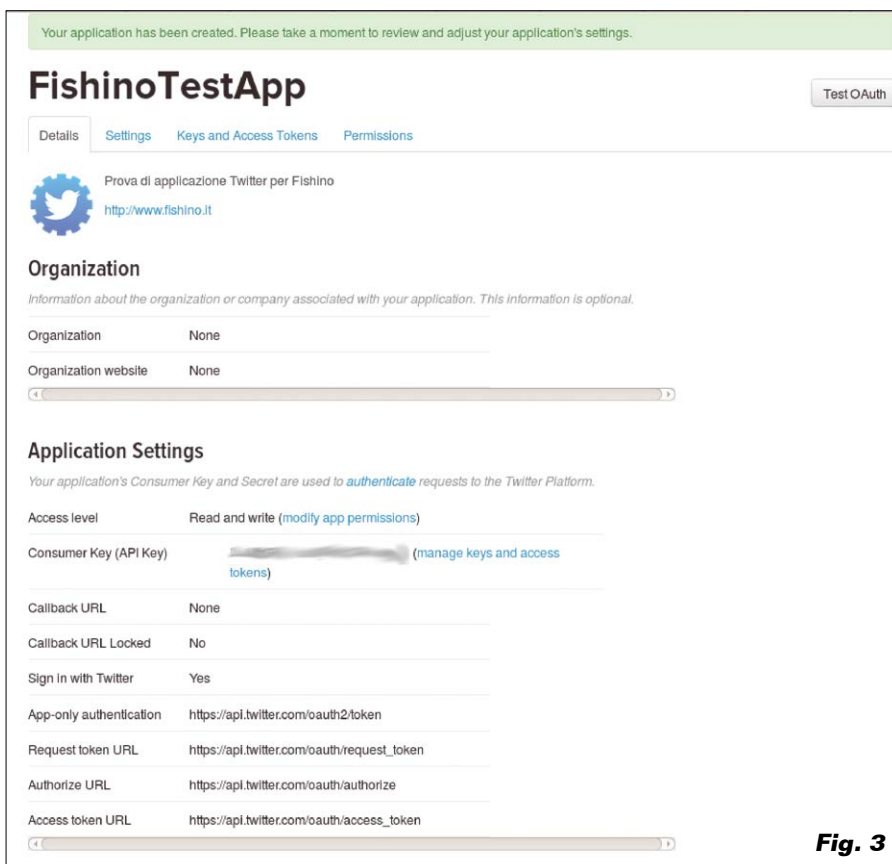


Fig. 3

(#fishino in questo caso);

- un messaggio di testo, comprendente gli hashtag e altro;
- un timestamp, ovvero data ed ora di invio.

Il tweet può anche contenere riferimenti ad altre persone, immagini, URL, eccetera.

Nel nostro sistema sono importanti il **mittente**, utilizzato per consentire o negare l'accesso, gli **hashtag**, utilizzati per differenziare un tweet normale da uno di comando al sistema, il testo, che contiene il comando vero e proprio, ed infine il **timestamp**, o data e ora di invio, che serve per discriminare i comandi già ese-

guiti da quelli ancora da eseguire. Ovviamente non possiamo utilizzare una ricerca simile tramite un PC per controllare i nostri dispositivi, ma abbiamo bisogno di qualcosa di più "digeribile" da un sistema automatico, ed il più semplice possibile.

Per questo ci vengono in aiuto le API (Application Programming Interfaces) fornite da Twitter che permettono di effettuare, tra le altre cose, delle ricerche sui tweet.

LE API DI TWITTER

Le API in questione sono molto estese, permettendo di effettuare moltissime operazioni, sia di ricerca che di invio di tweet tra-

mite software. Attualmente, per il nostro controller ci interessano solo le API di ricerca.

Il funzionamento di tutte le API è comunque abbastanza omogeneo e, una volta capiti i meccanismi di un loro sottoinsieme, risulta facilissimo utilizzarle.

Iniziamo con il superare il primo scoglio incontrato durante lo sviluppo, e che è lo scoglio più grosso che ha obbligato molti progetti analoghi ad utilizzare sistemi molto più complessi di Fishino e/o PC di appoggio per la ricerca dei tweet: la sicurezza. Twitter utilizza, per rendere sicuro il suo sistema, principalmente due meccanismi i quali prevedono che:

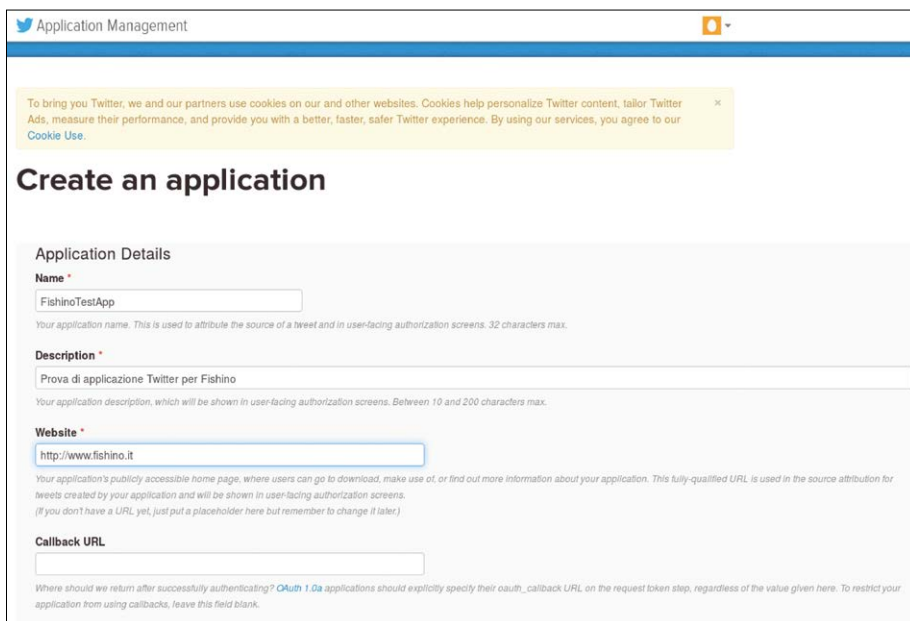
- la connessione al server delle API debba avvenire con protocollo sicuro https;
- l'utente delle API debba autenticarsi in uno dei due modi che vedremo in seguito.

Se il secondo requisito è relativamente semplice da soddisfare, il primo impone -per contro- l'utilizzo di un sistema in grado di effettuare connessioni con protocollo sicuro HTTPS. Questo esclude automaticamente gli Ethernet Shield e WiFi Shield di Arduino, per esempio, che non sono in grado di utilizzarlo. Invece Fishino è in grado di eseguire connessioni https anche se col limite di una connessione per volta, vista la grande richiesta di risorse di memoria necessaria per la crittografia.

Vediamo, nel **Listato 1**, una

Listato 1

```
GET
/1.1/search/tweets.json?count=1&q=%23fishino+from%3Amdelfede&result_type=recent&max_id=647068135569387520&since_id=646038066440896517 HTTP/1.1
Host: api.twitter.com
User-Agent: Fishino Twitter Client
Authorization: Bearer AAAAAAAAAAAAAAAAAA(altri caratteri della chiave di autorizzazione)
```



Application Management

To bring you Twitter, we and our partners use cookies on our and other websites. Cookies help personalize Twitter content, tailor Twitter Ads, measure their performance, and provide you with a better, faster, safer Twitter experience. By using our services, you agree to our [Cookie Use](#).

Create an application

Application Details

Name *
FishinoTestApp
Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

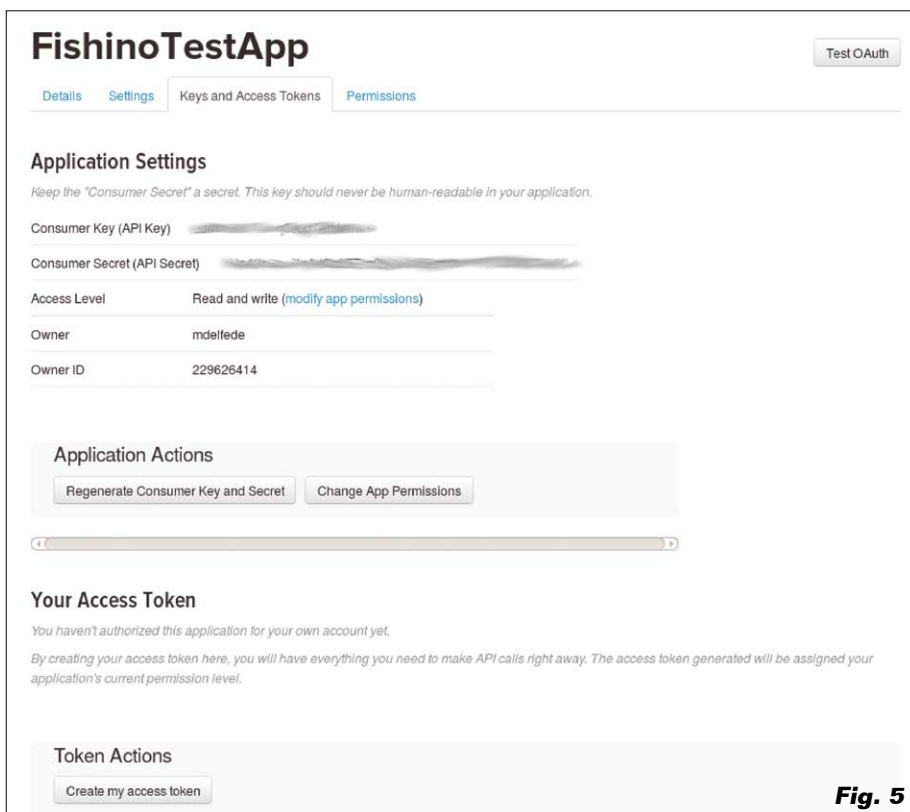
Description *
Prova di applicazione Twitter per Fishino
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *
<http://www.fishino.it>
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Fig. 4



FishinoTestApp

Details Settings Keys and Access Tokens Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key) [REDACTED]

Consumer Secret (API Secret) [REDACTED]

Access Level: Read and write (modify app permissions)

Owner: mdelfede

Owner ID: 229626414

Application Actions

Regenerate Consumer Key and Secret Change App Permissions

Your Access Token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token Actions

Create my access token

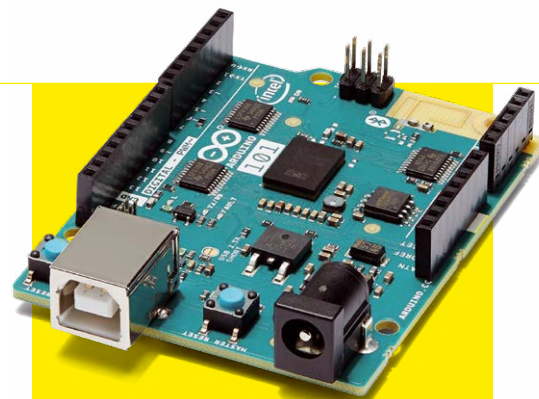
Fig. 5

richiesta https tipica alle API di Twitter, che andremo a esaminare in dettaglio. La prima riga è la richiesta vera e propria:

- **GET /1.1/search/tweets.json?** è la richiesta alle API di ricerca dei tweet;
- **count=1:** è il numero di risul-

tati richiesti; ne vogliamo uno per volta;

- **q= %23fishino+from %3Amdelfede** significa che cerchiamo l'hashtag (%23, #) fishino proveniente (%3A,@) da mdelfede;
- **result_type=recent** significa



Arduino sposa Intel

All'ultimo Rome Maker Faire, Intel e Massimo Banzi hanno presentato Genuino 101, una scheda di sviluppo pensata per la scuola, il mondo dei maker e per quanti si avvicinano per la prima volta alla programmazione. Genuino 101 è basata su Intel Curie, un SoC a basso consumo dalle dimensioni ridottissime, nato per applicazioni wearable; esso implementa un modulo Bluetooth LE e un accelerometro giroscopio a 6 assi. Il core è un Quark x86 a 32 bit con clock a 32 MHz nel quale gira un sistema operativo RTOS ViperOS. La board utilizza anche un core ARC (Argonaut RISC Core) a 32 bit anch'esso con clock a 32 MHz. Per il resto la basetta ha lo stesso form-factor di Arduino UNO e dispone di 14 pin digitali di I/O (di cui 4 possono essere utilizzati come uscite PWM), 6 ingressi analogici, un connettore USB per la comunicazione seriale e per il caricamento degli sketch, un jack di alimentazione e un connettore ICSP con SPI e I²C. Le linee di I/O funzionano a 3,3V ma sono protette contro le sovratensioni fino a 5 V. Genuino 101 sarà adottata da Creative Technologies in the Classroom, programma di physical computing attualmente implementato in oltre 300 scuole.

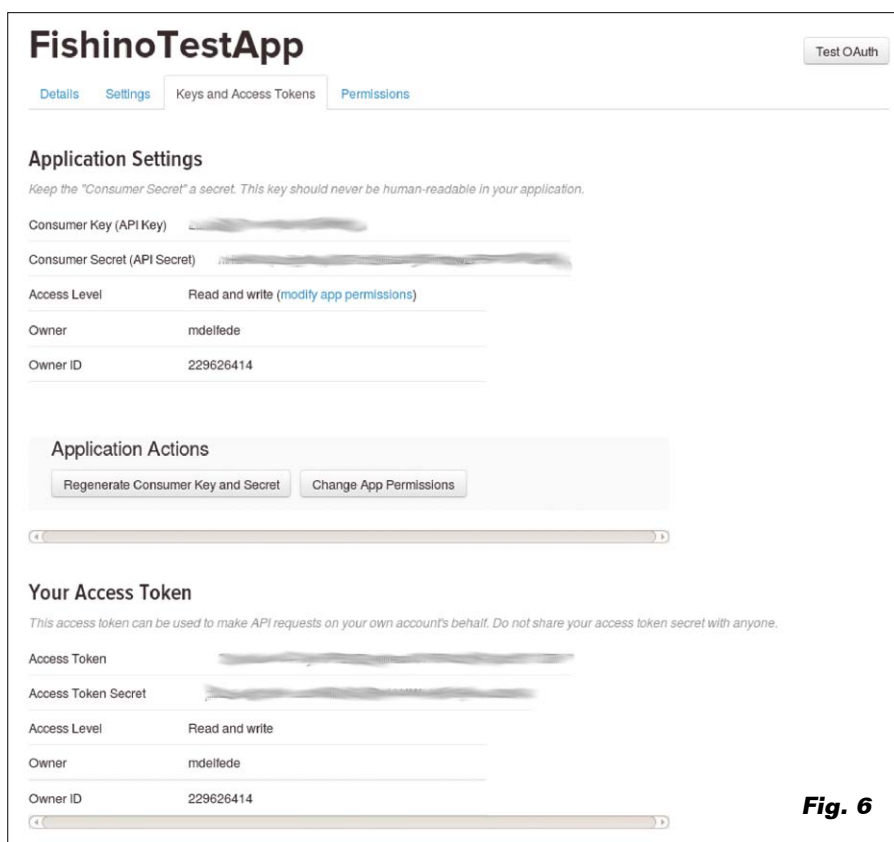


Fig. 6

che vogliamo i risultati in ordine di tempo e non di popolarità;

- **max_id= 647068135569387520;** gli ID sono numeri di sequenza contenuti in ogni messaggio Twitter, e che forniscono la sequenza temporale dei medesimi; in questo caso vogliamo cercare un messaggio non successivo al numero impostato;
- **since_id= 46038066440896517**

vuol dire che vogliamo un messaggio successivo al numero impostato;

- **HTTP/1.1** chiude la query http specificando il formato del protocollo.

La seconda e la terza riga indicano rispettivamente l'host cui viene inviata la richiesta (*api.twitter.com*) e l'applicazione che la

sta effettuando (Fishino Twitter Client); quest'ultima è arbitraria e serve solo ad identificare l'applicazione che utilizza le API. La quarta riga è molto importante, perché è quella utilizzata per l'autenticazione della richiesta sulla rete twitter:

Authorization:
Bearer AAAAAAAAAAAAAAAAAAAAAA
 (altri caratteri della chiave di autorizzazione)

Come già detto Twitter utilizza due metodi per rafforzare la sicurezza delle connessioni: il protocollo https (che impedisce gli attacchi tipo "man in the middle") e delle chiavi di autenticazione. Queste ultime sono di due tipi: personali o di applicazione (application-only); con il primo tipo di autenticazione il programma accede all'API come se fosse un utente specifico di Twitter ed agisce in suo nome; può, per esempio, inviare tweet come se fossero stati inviati da un PC. Il secondo tipo di autenticazione è invece legato all'applicazione, quindi parzialmente anonimo; non permette l'invio di messaggi ma solo alcune operazioni: per esempio la ricerca dei tweet. Nel nostro progetto abbiamo utilizzato il secondo metodo, che da un lato è più limitato perché Twitter pone dei limiti temporali all'utilizzo delle API: non è possibile eseguire oltre 180 operazioni ogni 15 minuti come utente o come applicazione. Quindi, se eseguiamo 180 richieste in 15 minuti come applicazione ce ne rimangono altrettante a disposizione come utente. 180 richieste ogni 15 minuti equivalgono ad un intervallo di 5 secondi tra 2 richieste.

Superare questi limiti vuol dire rischiare di farsi bloccare l'account Twitter, quindi è opportuno evitare richieste troppo frequenti.

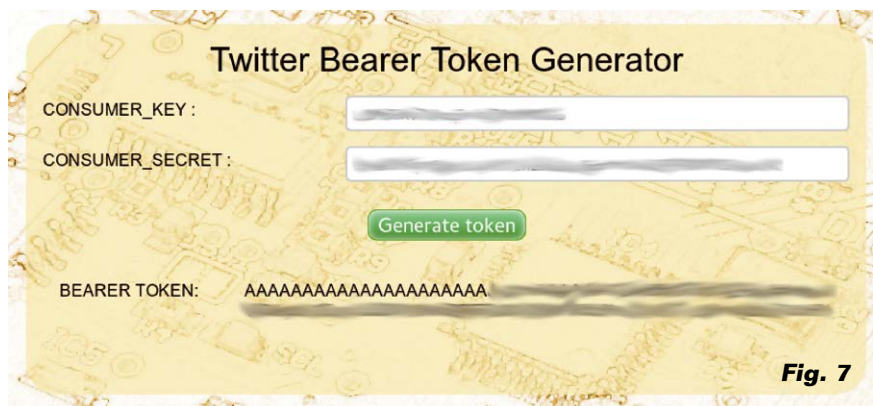


Fig. 7

Listato 2

```
////////////////////////////////////
////////////////////////////////////
// CONFIGURATION DATA          -- ADAPT TO YOUR NETWORK !!!
// DATI DI CONFIGURAZIONE -- ADATTARE ALLA PROPRIA RETE WiFi !!!

////////////////////////////////////
////////////////////////////////////
// CONFIGURATION DATA          -- ADAPT TO YOUR NETWORK !!!
// DATI DI CONFIGURAZIONE -- ADATTARE ALLA PROPRIA RETE WiFi !!!

// here put SSID of your network
// inserire qui lo SSID della rete WiFi
#define MY_SSID    "CASA"

// here put PASSWORD of your network. Use "" if none
// inserire qui la PASSWORD della rete WiFi -- Usare "" se la rete non è protetta
#define MY_PASS    "unapassword"

// here put required IP address of your Fishino
// comment out this line if you want AUTO IP (dhcp)
// NOTE : if you use auto IP you must find it somehow !
// inserire qui l'IP desiderato per il fishino
// commentare la linea sotto se si vuole l'IP automatico
// nota : se si utilizza l'IP automatico, occorre un metodo per trovarlo !
#define IPADDR     192, 168, 1, 251

// here put your application Bearer authorization key
// see https://dev.twitter.com/oauth/application-only for details
// inserire qui la chiave "bearer" di autenticazione dell'applicazione
// vedere https://dev.twitter.com/oauth/application-only per dettagli
#define BEARER     "AAAAAAAAAAAAAAAAAAAAA....."

// here put your Twitter user name; if empty the application will react to ANY user
// inserire qui il nome utente ai cui tweets il programma si attiva; se vuoto, il programma si attiva con TUTTI gli utenti
#define TWITTER_USER "pippo"

// here put twitter hashtag needed to trigger the event (without the # char!)
// inserire qui l'hashtag necessario ad avviare l'applicazione (senza il carattere #!)
#define TWITTER_HASHTAG "fishino"

// here put the delay in milliseconds between requests
// there is a limit of about 180 requests every 15 minutes, so don't put a too short time here
// inserire qui il ritardo tra due richieste successive
// c'è un limite di circa 180 richieste ogni 15 minuti, quindi non inserire un tempo troppo breve
#define REQUESTS_INTERVAL 5000
```

OTTENERE UNA CHIAVE 'BEARER' DI AUTORIZZAZIONE

Vediamo come creare un'applicazione Twitter e ottenere la relativa chiave di autorizzazione. Innanzitutto occorre essere registrati a Twitter e aprire l'indirizzo <https://apps.twitter.com/>.

Facendo clic sul pulsante Create New App appare la schermata di Fig. 3. Qui occorre riempire alcuni campi obbligatori quali:

- **Nome applicazione (Name);** in questo caso, **FishinoTestApp**;
- **Descrizione applicazione**

(Description); in questo caso "Prova di applicazione Twitter per Fishino";

- **Sito web (Website)** è un sito che dovrebbe contenere i dettagli dell'applicazione; è obbligatorio, ma può essere anche solo un segnaposto; in questo caso è www.fishino.it;
- **Callback URL**; non necessaria per i nostri scopi (lasciare in bianco).

In fondo alla pagina ci sono le usuali condizioni del servizio da accettare (selezionare la

checkbox) e il pulsante per creare l'applicazione (Create your Twitter application); cliccando appare la pagina di Fig. 4.

L'applicazione è stata creata e viene fornita (qui l'abbiamo oscurata) la 'consumer API key', che è una delle chiavi necessarie all'utilizzo delle API. Poiché vogliamo usare la 'Bearer token', ovvero la chiave necessaria in modalità 'Application only', serve un ulteriore passo: in alto cliccate sul tab 'Keys and access tokens'; vi apparirà la schermata in Fig. 5, che informa che l'applicazione

Listato 3

```
if (!client.connected() && !client.available())
{
    Serial.println("\nStarting connection to server...");
    if(client.connect("api.twitter.com", 443))
    {
        Serial.println("connected to server");

        // run the query looking from given hashtag and user
        doQuery(client, "%23" TWITTER_HASHTAG);
    }
}
```

Listato 4

```
// run a query on twitter
// esegue la query a Twitter
void doQuery(FishinoSecureClient &client, const char *query)
{
    // if walkback is finished (nextID == -1) we reset sinceID to next found maximum ID
    // otherwise we let it as it is and continue the walk back
    // se il walkback è finito (nextID == -1) re-impostiamo il sinceID a dopo
    // il massimo ID trovato
    // altrimenti lasciamo tutto come sta e continuiamo col walkback
    if(nextID == (uint64_t)-1)
        sinceID = maxID;

    client.print("GET /1.1/search/tweets.json?count=1&q=");
    client.print(query);
    if(strlen(TWITTER_USER))
    {
        client.print("&from%3A");
        client.print(TWITTER_USER);
    }
    client.print("&result_type=recent");
    if(nextID != (uint64_t)-1)
    {
        client.print("&max_id=");
        client << nextID;
    }
    client.print("&since_id=");
    client << sinceID;
    client.println(" HTTP/1.1");

    client.println("Host: api.twitter.com");
    client.println("User-Agent: Fishino Twitter Client");

    // send auth key to server
    // invia la chiave di autorizzazione al server
    client.println("Authorization: Bearer " BEARER);
    client.println();

    // reset nextID - it will be filled by parser if more results are pending
    // azzerà il nextID - verrà riempito dal parser se ci sono altri risultati da leggere
    nextID = -1;
}
```

non è ancora stata autorizzata (voce sotto **'Your Access Token'**). Facendo clic sul pulsante **'Create my access token'** in basso si ottiene la schermata di **Fig. 6**. Sotto la voce **'Your Access Token'** appaiono ora due chiavi che vanno utilizzate per l'uso dell'applicazione in modalità normale (**non Application-only**), vale a dire come se foste voi ad eseguire

i comandi. L'ultimo passo per ottenere la chiave **Application-only**, che è poi quella che useremo nella nostra applicazione; sarebbe il più complicato, visto che nelle pagine della documentazione delle API viene spiegato come fare ma, purtroppo, non viene fornito un sistema per farlo in modo semplice. Abbiamo quindi provveduto a realizzare

una pagina all'indirizzo *www.fishino.it/bearer.php* che permette di ottenere in modo facilissimo la cosiddetta **'Bearer token'**. È sufficiente inserire la **'consumer_key'** e la **'consumer_secret'** dell'applicazione, ricavate in precedenza, e premere il pulsante **'Generate token'** per ottenere la chiave da copiare ed incollare nello sketch. Ultima nota: la chiave appare sullo schermo in due linee per motivi di larghezza di pagina, ma è da copiare e incollare in una sola linea.

LO SKETCH PER FISHINO

Passiamo ora ad esaminare lo sketch (**Listato 2**) partendo dall'inizio del file, dove abbiamo scelto di configurare pochi elementi per dare modo agli "impazienti" di provare subito il sistema. Le impostazioni sono poche e semplici; chi già conosce le applicazioni dimostrative di Fishino noterà che lo stile è sempre lo stesso e i contenuti variano di poco; in particolare, verso la fine del gruppo di impostazioni si notano quelle rilevanti per Twitter, ovvero la Bearer Token di cui abbiamo parlato al paragrafo precedente (**BEARER**), il nome utente (**TWITTER_USER**) da cui devono provenire i tweet, l'hashtag che attiva il programma (**TWITTER_HASHTAG**), da inserire senza il carattere di cancelletto (#) e l'intervallo tra due richieste al sistema. Per quanto riguarda quest'ultimo, per rispettare il limite di 180 richieste in 15 minuti abbiamo impostato una richiesta ogni 5 secondi circa (#define REQUESTS_INTERVAL 5000). Sugeriamo di non scendere sotto il valore 5.000 (millisecondi) se non per brevi prove.

DESCRIZIONE DELLO SKETCH

Una buona parte dello sketch si occupa dell'inizializzazione della

Listato 5

```
// analyze data and act on devices
// analizza i dati in arrivo da Twitter
void parserCallback(uint8_t filter, uint8_t level, const char *name, const char *value)
{
    // check for user name
    // controlla il nome utente
    if(level == 3 && !strcmp(name, "screen_name") && !strcmp(value, "\"" TWITTER_USER "\""))
        gotUser = true;

    // check twitter message - MUST contain the requested hashtag, a port number and word ON or OFF
    // controlla il testo del messaggio - DEVE contenere l'hashtag richiesto, un numero di porta e le parole ON o OFF
    else if(level == 2 && !strcmp(name, "text") && value && value[0] && !strnicmp(value + 1, "#" TWITTER_HASHTAG, strlen(TWITTER_HASHTAG) + 1))
    {
        // hashtag found, skip it
        // trovato l'hashtag, lo salta
        const char *p = value + strlen(TWITTER_HASHTAG) + 3;

        // skip spaces after hashtag
        // salta gli spazi dopo l'hashtag
        while(*p && isspace(*p))
            p++;

        // read port number - sets it to 0xff if not found
        // legge il numero di porta - lo imposta a 0xff se non trovato
        if(isdigit(*p))
        {
            port = 0;
            while(isdigit(*p))
                port = port * 10 + *p++ - '0';
        }
        else
            port = 0xff;

        // if port not found, just leave
        // se non trova il numero di port ritorna
        if(port == 0xff)
            return;

        // skip spaces after port number
        // salta gli spazi dopo il numero di porta
        while(*p && isspace(*p))
            p++;

        // check for ON or OFF keywords - if not found sets port to 0xff and leave
        // cerca le parole chiave ON o OFF - se non trovate imposta la porta a 0xff ed esce
        if(!strnicmp(p, "on", 2))
            command = true;
        else if(!strnicmp(p, "off", 3))
            command = false;
        else
            port = 0xff;
    }

    // look if there are previous tweets to handle
    // cerca se deve elaborare tweets precedenti
    else if(level == 1 && !strcmp(name, "next_results"))
    {
        // get next tweet id to handle
        const char *s = strstr(value, "\"?max_id=");
        if(s)
        {
            s += 9;
            nextID = readID(s);
        }
    }

    // get max id of search, needet to go further after walk back
    // cerca il massimo ID corrispondente alla ricerca, necessario per proseguire dopo la ricerca all'indietro
    else if(level == 2 && !strcmp(name, "id"))
    {
        uint64_t id = readID(value);
        if(id > maxID)
            maxID = id;
    }
}
```

Listato 6

```
// waits for data from twitter and parse them - return true on success, false if no data
bool doReceive(FishinoSecureClient &client, JSONStreamingParser &parser)
{
    // wait 1 second max for data from server
    // attende al massimo 1 secondo i dati dal server
    unsigned long tim = millis() + 1000;
    while(!client.available() && millis() < tim)
        ;

    // if no data is available, close the connection
    // se non ci sono dati, chiude la connessione
    if(!client.available())
        return false;

    // skip data up to 2 consecutive cr
    bool cr = false;
    while(client.available())
    {
        if(cr)
        {
            if(client.read() == 0x0d && client.read() == 0x0a)
                break;
            else
                cr = false;
        }
        else if(client.read() == 0x0d && client.read() == 0x0a)
            cr = true;
    }

    while(client.available())
    {
        char c = client.read();
        parser.feed(c);
    }
    return true;
}
```

scheda, della connessione alla rete WiFi, eccetera.

Per brevità rimandiamo agli articoli di presentazione di Fishino; qui vediamo in dettaglio solo le parti più interessanti dello sketch, ovvero la gestione della connessione sicura alle API di Twitter e l'utilizzo delle schede di espansione I/O.

Per quanto riguarda la connessione alle API, notiamo subito una differenza rispetto agli sketch visti in precedenza:

```
// The web secure client used
to access twitter API
// il client https usato per
l'accesso all'API di Twitter
FishinoSecureClient client;
```

Qui si nota l'utilizzo del FishinoSecureClient al posto del precedente FishinoClient. Questo client viene utilizzato per le connessioni sicure https.

La connessione alle API avviene tramite lo spezzone di codice contenuto nel **Listato 3**: prima viene controllato se il client è ancora connesso dal loop precedente e/o se ha ancora dei dati da leggere, visto che i dati sono mantenuti in un buffer nel client anche dopo la chiusura dello stesso. In caso contrario viene aperta la connessione al sito *api.twitter.com* sulla porta 443, che è quella utilizzata nelle connessioni https.

Se la connessione ha esito positivo, viene inviata la richiesta vera e propria tramite la funzione `doQuery()`, che ha come parametri il client e l'hashtag da cercare. La funzione non fa altro che inviare i dati come visto durante la spiegazione delle API (**Listato 4**). La particolarità di questa funzione, a parte l'invio dei dati di ricerca veri e propri, sta nella gestione dei campi **since_id** e **max_id** che consentono di restringere la ricerca in ben precisi limiti temporali.

Listato 7

```
// if port has been set, change its output
// se la porta è stata impostata, ne modifica l'output
if(port != 0xff)
{
    // if no expanders found, check if pin is free on Fishino
    // and set its output (must be changed on Mega!)
    if(!numExpanders)
    {
        if(port == 4 || port == 7 || port >= 10)
            Serial << F("INVALID PORT #") << port << F(" -- COMMAND DISCARDED\n");
        else
        {
            Serial << "\n\nSETTING PORT " << port << " TO " << (command ? "HIGH" : "LOW") << "\n";
            pinMode(port, OUTPUT);
            digitalWrite(port, command ? HIGH : LOW);
        }
    }
    else
    {
        uint8_t board = port / 8;
        if(board >= numExpanders)
            Serial << F("INVALID PORT #") << port << F(" -- COMMAND DISCARDED\n");
        else
        {
            Serial << "\n\nSETTING PORT " << port << " TO " << (command ? "HIGH" : "LOW") << "\n";
            port %= 8;
            expanders[board]->digitalWrite(15-port, command ? HIGH : LOW);
        }
    }
}
```

Per motivi di spazio e velocità, ad ogni richiesta scegliamo infatti di ricevere un solo tweet, che è l'ultimo presente in rete tra i due ID forniti; giocando su questi ultimi è possibile andare a ritroso richiedendo i tweet precedenti, fino all'esaurimento dei medesimi. Finita la ricerca all'indietro, viene reimpostato il **since_id** in modo che nelle richieste successive vengano selezionati solo i tweet più nuovi.

Una volta inviata la richiesta, occorre ricevere ed analizzare i dati. Le API di Twitter rispondono con una serie di dati in formato JSON e di dimensioni molto variabili, anche piuttosto grandi. Vista la limitatezza delle risorse di memoria della scheda, una memorizzazione di questi dati è impensabile; occorre quindi analizzarli man mano che arrivano. Per far questo abbiamo scritto una libreria completa per il parsing (scansione) dei dati in arrivo in modalità streaming, ovvero carattere per carattere, senza memorizzazioni intermedie.

Questa libreria, **JSONStreamingParser**, è diversa dalla maggior parte delle librerie analoghe presenti in rete visto che, oltre all'analisi dei dati in tempo reale, non memorizza nemmeno i risultati, se non quello corrente.

In pratica, i dati JSON hanno un formato simile a questo:

```
{
  nome1 : dato1,
  nome2 : {nome3 : dato3, nome4 : dato4},
  nome5 : [dato6, dato7, dato8]
}
```

Ovvero, i dati possono contenere valori semplici (stringhe, numeri, eccetera) oppure liste di dati oppure array di dati. Il formato, arrivando ai dati semplici, è comunque sempre lo stesso:

```
nome : dato
```

Listato 8

```
// find all available IOExpander boards
void findIOExpanders(void)
{
  numExpanders = 0;
  for(uint8_t addr = 0; addr < 8; addr++)
  {
    IOExpanderMCP23017 *exp = new IOExpanderMCP23017;

    // set expander address
    exp->begin(addr);

    // this is because expander board hasn't reset connected to arduino one
    // and it checks register 0 upon init to see if board is present
    // (register should be 0xff, all A port as input)
    for(int i = 0; i < 16; i++)
      exp->pinMode(i, INPUT);

    if(exp->init())
    {
      // expander found, add to list and setup pins
      expanders[numExpanders++] = exp;

      // set pin modes
      for(int i = 0; i < 8; i++)
        exp->pinMode(15-i, OUTPUT);
    }
    else
      // not found, delete expander object
      delete exp;
  }
  if(numExpanders)
    Serial << F("Found ") << numExpanders << F(" I/O expander boards\n");
  else
    Serial << F("No expander boards found - using Fishino's I/O pins\n");
}
```

Il parser, quando determina un nome, esegue la scansione del dato; se questo è un tipo semplice, chiama una funzione predefinita con i seguenti parametri:

```
void callback(uint8_t filter,
uint8_t level, const char *name,
const char *value
```

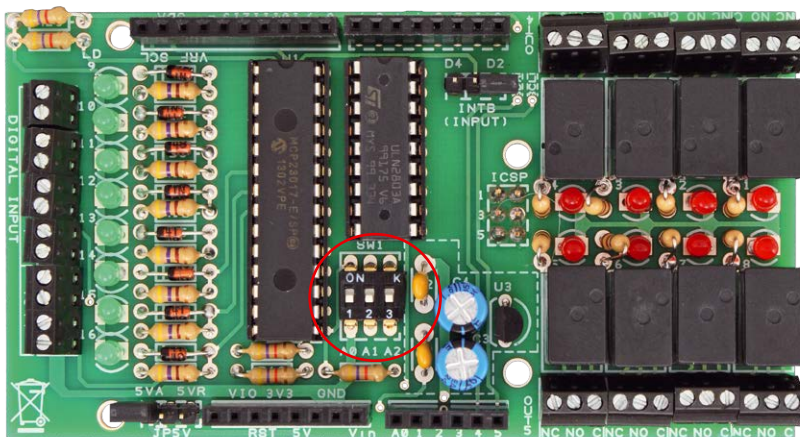
dove il parametro 'filter' è previsto per future espansioni, il parametro 'level' indica il livello di nidificazione del dato ed i parametri 'name' e 'value' indicano nome e valore del dato. Se il dato non è semplice ma costituito da una lista o da un array, il campo 'value' assume il valore "<LIST>" o "<ARRAY>" e la scansione prosegue al suo interno. Creando opportune funzioni di **callback** è quindi possibile analizzare al volo il contenuto del **JSON** ricevuto ed impostare le

variabili opportune nello sketch. L'attuale funzione di callback è contenuta nel **Listato 5**.

La funzione sembra complessa, ma in realtà non fa altro che sfruttare i parametri 'level' e 'name' per determinare se sono presenti o meno i campi richiesti nel tweet e, nel caso, ne estrae ed analizza i valori.

Qui si nota che viene controllato l'utente che ha inviato il tweet, gli hashtag per vedere se c'è quello impostato, il parametro 'text' per leggere i comandi inviati e gli ID utilizzati per controllare la sequenza temporale dei tweet.

Come dicevamo, la funzione viene richiamata dal **JSONStreamingParser** ogni volta che questo legge un valore dallo stream. Il parser, a sua volta viene utilizzato dalla funzione **doReceive()**



Con SW1 si imposta l'address I²C dello shield I/O Expander: notate la corrispondenza tra A0, A1, A2 e i singoli dip.

come mostrato nel **Listato 6**.

Questa funzione non fa altro che leggere i dati dal client, saltare il preambolo HTTP (la cui fine è indicata da una linea vuota) ed inviare i restanti caratteri, uno alla volta, al parser, per l'analisi. Una volta completata l'analisi dello stream di dati, in caso di comando valido la variabile globale 'port' conterrà il numero di porta da modificare e la variabile globale 'command' (booleana) sarà impostata a true se vorremo portare l'uscita a, valore HIGH, oppure false se la vorremo a LOW.

Vediamo la parte che attiva l'uscita, interessante per il fatto che prevede la possibilità di applicare gli I/O Expander a Fishino (**Listato 7**). Qui si nota per prima cosa il controllo sulla validità della porta (port); se è a 0xff il comando è da scartare, altrimenti viene proseguita l'analisi. Successivamente il codice agisce in modo differente se sono presenti o meno le espansioni. Nel primo caso, determina il numero di scheda di espansione (port / 8) e la porta sulla medesima (port % 8). Per esempio, un valore di port pari a 33 corrisponde alla scheda n° 4 (33/8) ed alla sua porta n° 1 (33 % 8 = 33 - 4 •

8 = 1). L'operatore '%' fornisce infatti il resto della divisione. Nel caso non siano presenti le espansioni, lo sketch utilizza il numero di porta direttamente come I/O digitale del Fishino; controlla che sia un valore valido e che non vada a modificare pin utilizzati per altri scopi ed agisce su quello.

UTILIZZO DELLE SCHEDE DI ESPANSIONE I/O

Lo sketch prevede, in modo completamente automatico, il riconoscimento di eventuali schede I/O Expander anche connesse in cascata, fino ad un massimo di 8. L'importante è avere l'accortezza di fornire indirizzi I²C differenti ad ogni scheda.

All'avvio lo sketch controlla, nel setup(), se sono presenti le schede e le numera in base all'indirizzo I²C. Se ne trova, assegna i port di uscita in sequenza: da 0 a 7 per la prima scheda, da 8 a 15 per la seconda e così via.

Nel caso non venga rilevata alcuna scheda di espansione, lo sketch va in modalità standalone utilizzando tutti gli I/O liberi di Fishino, e quindi le uscite 0, 1, 2, 3, 5, 6, 8 e 9, essendo le altre riservate al funzionamento delle periferiche incluse.

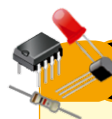
In questo caso il numero di porta corrisponde al numero dell'I/O digitale di **Fishino**.

La numerazione delle schede avviene tramite lo spezzone di codice contenuto nel setup (**Listato 8**).

Si nota il ciclo che prova tutti gli indirizzi I²C possibili (da 0 a 7); per ogni indirizzo crea una variabile **IOExpanderMCP23017** (in modo dinamico, come puntatore), ne esegue il test di presenza e, nel caso funzioni la memorizza in un array di puntatori. Se la scheda non è presente elimina la variabile e riparte con la successiva. In questo modo le schede vengono rilevate in ordine di indirizzo I²C.

Con questo abbiamo concluso la descrizione dell'applicazione **Fish'n Tweets!**

In un prossimo articolo vedremo come fare l'operazione inversa, ovvero far inviare dei tweet alla scheda quando alcuni pin di ingresso cambiano di stato, cosa utilissima per esempio per realizzare degli allarmi o monitorare eventi in un luogo remoto. ■



per il MATERIALE

Tutto il materiale necessario per realizzare questo progetto è disponibile presso Futura Elettronica. Lo shield Arduino I/O expander in kit (cod. FT1079K) è disponibile a Euro 34,00. La board Fishino (cod. FISHINOUNO) viene fornita montata e collaudata al prezzo di Euro 36,00. Software e sketch sono scaricabili gratuitamente sul sito www.elettronica.in.it. 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>

L'ACQUARIO CONNESSO

di SIMONE DALL'ASTA

Rendiamo smart un acquario esistente e inseriamolo nel mondo dell'IoT grazie alla scheda Fishino.

La tecnologia si è fatta strada ormai anche nella nostra vita quotidiana e la domotica diviene ogni giorno più familiare: è comodo pensare, ad esempio, ad un sistema di riscaldamento che ci avvisa di aver dimenticato aperta una finestra per evitare di disperdere il calore; ma è anche suggestivo vederla impiegata in situazioni meno canoniche, come la

gestione di un acquario proposta in queste pagine. Il progetto che vi presentiamo consente infatti di gestire un acquario senza però modificarne la struttura o l'aspetto elettrico. L'idea è quindi quella di passare da un acquario tradizionale ad una versione più "smart" senza la paura di non poter tornare indietro nella propria scelta.

Nello sviluppo del progetto abbiamo trovato un valido alleato in Fishino UNO, la scheda basata sull'ATmega328 di Arduino UNO, distribuita da Futura Group e che è stata presentata sulla rivista numero 198; la scel-



CARATTERISTICHE TECNICHE

- Alimentazione: 5 Vcc
- Basato su Fishino UNO (WiFi, RTC e lettore di microSD a bordo)
- Connessione con un database MySQL
- Acquisizione della temperatura dell'acqua tramite la sonda DS18B20
- Possibilità di monitorare la temperatura dell'acquario in un grafico ovunque ci si trovi tramite qualsiasi dispositivo con connessione a Internet
- Invio automatico di e-mail ad un indirizzo impostato in caso di temperature critiche
- Personalizzazione tramite microSD
- Attivazione o disattivazione automatica di aeratore, diffusore di anidride carbonica, plafoniere di illuminazione (se già presenti nell'acquario) tramite due relé

ta è stata dettata da specifiche funzionalità di cui la scheda dispone: una connessione WiFi per permettere di monitorare a distanza la temperatura dell'acqua utilizzando qualsiasi dispositivo dotato di connessione Internet e un orologio interno (RTC, Real Time Clock) con backup a batteria per temporizzare le fasi di funzionamento al posto dei soliti timer analogici (che in caso di blackout dovrebbero essere rimessi in passo). Infine, tramite una scheda microSD, è possibile definire con semplicità vari aspetti legati alla vita nell'acquario, come le ore di luce, quanta anidride carbonica erogare e se attivare o meno un diffusore di ossigeno. Per implementare le stesse funzioni con una classica Arduino UNO avremmo dovuto

dotarla di shield RTC, SD e WiFi, con evidente aumento dei costi e degli ingombri.

La scelta di Fishino UNO è poi naturale, in quanto il nome ...si addice al compito assegnatole!

IL SISTEMA

Vediamo dunque di cosa si tratta riferendoci alla Fig. 1; in essa vedete che il cuore pulsante del sistema è Fishino UNO, grazie al quale è possibile dotare l'acquario delle caratteristiche che lo rendono "smart" e che quindi lo distinguono da un acquario

tradizionale. Il funzionamento del dispositivo è semplice: Fishino UNO "apprende" gli orari di accensione e spegnimento delle plafoniere di illuminazione, dell'erogatore di anidride carbonica e dell'aeratore tramite i file contenuti nella scheda microSD. Dopo questa operazione sarà in grado di attivare o disattivare i dispositivi a seconda dell'orario. Il tempo è scandito dal modulo RTC (Real Time Clock) innestato in Fishino UNO e, anche in caso di alimentazione assente, la batteria di backup non fa perdere l'orario programmato. Grazie alla sonda impermeabile contenente un DS18B20 della Maxim è possibile rilevare la temperatura dell'acqua e, ogni mezz'ora, questo dato viene inviato in automatico tramite WiFi per essere poi memorizzato su un database MySQL (vedremo come). Successivamente sarà possibile, tramite un qualsiasi dispositivo dotato di connessione ad Internet, prendere visione

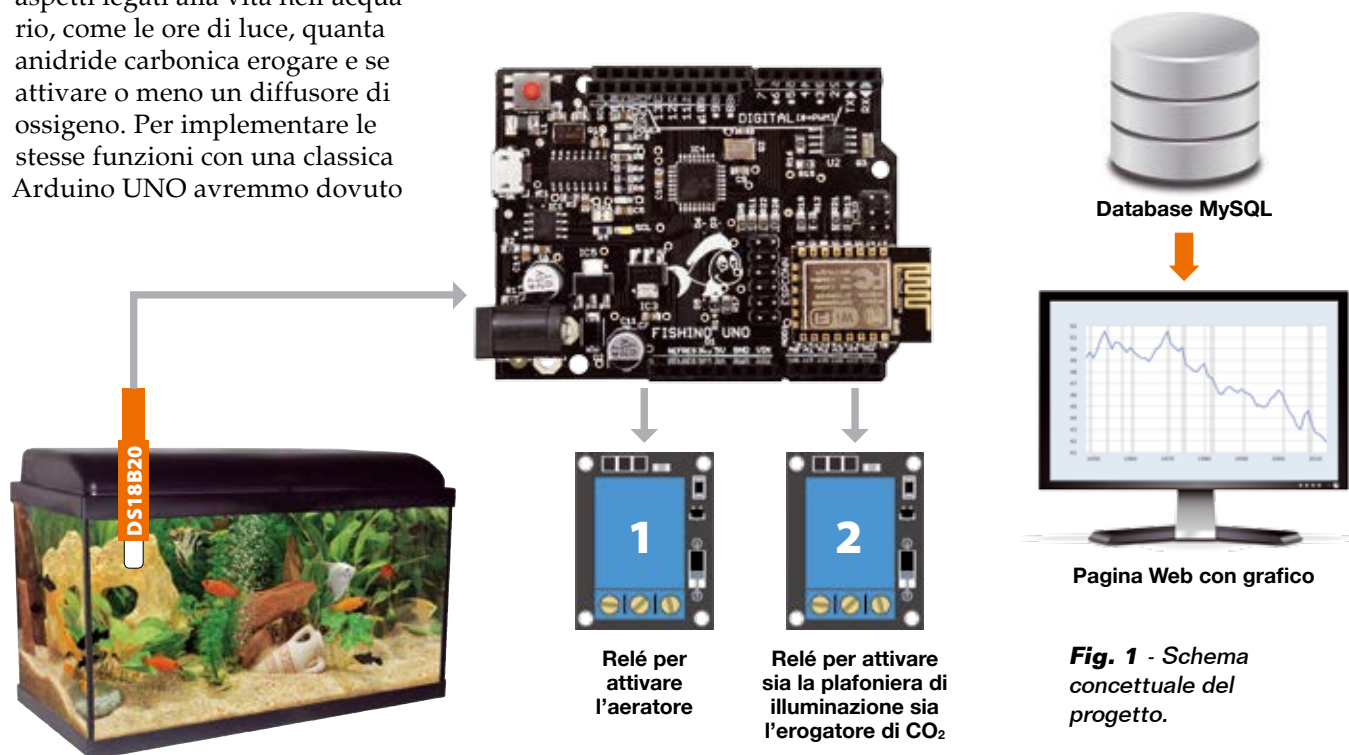


Fig. 1 - Schema concettuale del progetto.

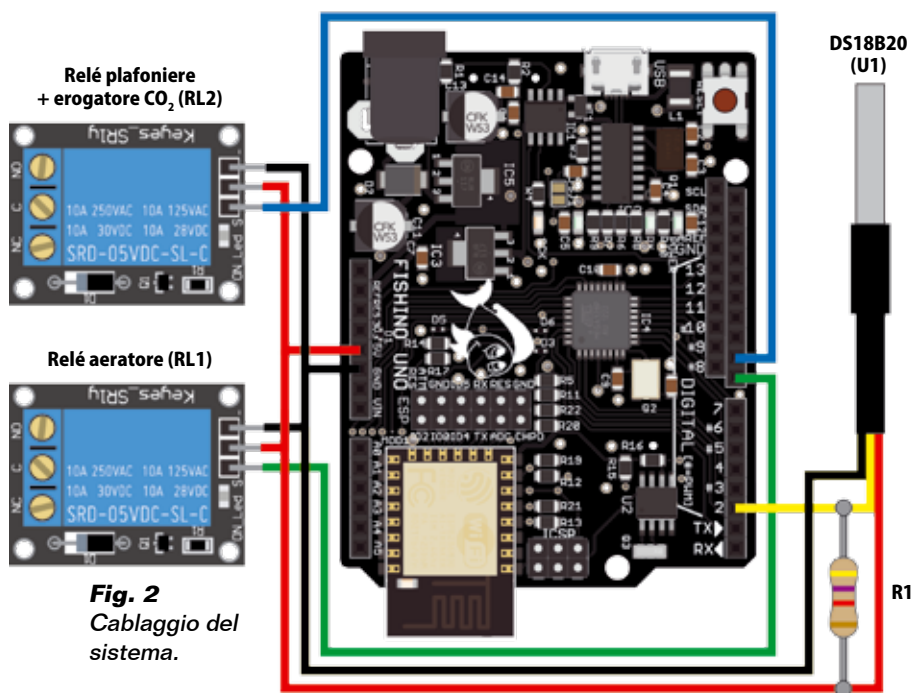


Fig. 2
Cablaggio del sistema.

della temperatura su un comodo grafico per sapere, ad esempio, la minima o la massima temperatura raggiunta dall'acqua. La temperatura rilevata, oltre ad esser salvata sul database, è usata anche per inviare, in caso di temperature critiche (troppo caldo o troppo freddo), una e-mail ad un indirizzo impostato così da essere avvisati e poter agire per tempo.

SCHEMA ELETTRICO

Nella rivista numero 198 abbiamo spiegato l'hardware di Fishino UNO, quindi qui ci limiteremo a spiegare la parte di circuito aggiuntiva richiesta dal progetto, il cui schema di cablaggio è mostrato dalla Fig. 2. Vediamo che la temperatura dell'acqua è acquisita da Fishino tramite una sonda di temperatura waterproof in acciaio inossidabile, realizzata con il sensore digitale di temperatura DS18B20 della Maxim (U1). U1 consente di misurare con precisione la temperatura in ambienti umidi, non solo se immerso in liquidi, con un range che va da -55°C a +125°C; naturalmente per un acquario questi

limiti sono più di quel che serve. Ogni sensore è dotato di un codice seriale a 64 bit memorizzato in una ROM al suo interno; ciò, unito al fatto che è richiesto un solo pin di comunicazione (1-Wire), permette di collegare più di un sensore a bus, in modo da misurare la temperatura in più punti dell'acquario ed ottenere così indicazioni più precise sulle condizioni dell'ambiente. La temperatura è misurata con una precisione di $\pm 0,5^\circ\text{C}$ (nel campo tra -10°C e $+85^\circ\text{C}$). La sonda si alimenta con una tensione che va da 3V a 5,5V.

La connessione della sonda a Fishino UNO si fa così:

- il filo nero va al pin GND;
- il filo rosso va al pin 5V;
- il filo giallo (canale dati) va al pin digitale 2 e gli si deve collegare una resistenza (R1) da 4,7 k Ω verso il 5V (pull-up).

Per quanto riguarda RL1 e RL2, abbiamo utilizzato due schede dotate di relé monoscambio a 240V/10A, pilotabili con segnali TTL; ciascun modulo relé richiede un'alimentazione di 5 Vcc, prelevata da Fishino UNO (cui si connette anche la massa, con il filo nero), e rende accessibili i contatti N.A. (Normalmente Aperto - NO), COM (Comune) e N.C. (Normalmente Chiuso). I moduli con RL1 e RL2 si pilotano con Fishino UNO tramite i pin digitali 8 e 9: il livello logico alto corrisponde a relé eccitato e lo zero a relé a riposo. Lo scambio dei relé va usato da interruttore, quindi C ed NO interromperanno un filo dell'alimentazione del carico per poi chiuderlo quando il relé viene attivato. Abbiamo previsto di collegare ad RL1 l'aeratore e di gestire con RL2 delle plafoniere di illuminazione e l'erogatore di CO₂. Per quanto riguarda l'apporto di anidride carbonica e di ossigeno siete liberi di usare il metodo con il quale siete abituati: potete, ad esempio, collegare al relé un'elettrovalvola normalmente chiusa che apre e chiude il tubo dentro il quale passa la CO₂ dopo aver attraversato il riduttore di pressione posto sopra la bombola. I carichi di plafoniere ed erogatore di anidride carbonica possono essere gestiti distintamente usando un relé aggiuntivo, ma, a nostro avviso, è più semplice usarne uno soltanto poiché, come ben saprete, l'apporto di CO₂ è necessario soprattutto durante i

Listato 1

```
CREATE TABLE acquario (
  Temperatura FLOAT NOT NULL,
  Timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP PRIMARY KEY
)
```


Fig. 3 - Percorso logico del dato on-line.



periodi di luce, giacché concorre al processo di fotosintesi clorofilliana, durante il quale la flora dell’acquario produce ossigeno assorbendo anidride carbonica grazie alla luce.

SOFTWARE

Visto l’hardware, passiamo alla parte software, che si divide sostanzialmente in due parti: la prima riguarda la programmazione di Fishino UNO, mentre la seconda è quella che permette il salvataggio dei dati su un

database relazionale MySQL, che ne permette poi il recupero per costruire un grafico sulla base della temperatura rilevata ogni mezz’ora. Potremmo vedere le due parti seguendo il percorso logico che seguirà il dato della temperatura dal momento dell’acquisizione a quello del

salvataggio fino ad arrivare all’output nel grafico. Conviene però cominciare dal fondo, ovvero da come avviene il salvataggio del dato su database, perché ciò vi aiuterà a capire il funzionamento dello sketch.

PAGINA WEB E DATABASE RELAZIONALE MYSQL

Per tenere monitorata la temperatura dell’acqua usando qualsiasi dispositivo dotato di connessione a Internet, è necessario rendere disponibile al mondo esterno ciò che Fishino UNO rileva; il modo più semplice per farlo è usare un sito web personale attraverso cui sarà possibile avere una sorta di finestra condivisa tra voi e Fishino. In Fig. 3 potete osservare il percorso effettuato dal dato della temperatura on-line: una prima pagina web si occupa della ricezione del dato e del salvataggio su database, mentre una seconda servirà per prelevare i dati dal database e mostrarli nel grafico.

Crea tabella

Nome tabella:

Struttura

Campo	Temperatura	Timestamp
Tipo	FLOAT	TIMESTAMP
Lunghezza/Set*1		
Predefinito ²	Nessuno	CURRENT_TIMESTAMP
Collation		
Attributi		
Null	☐	☐
Indice	***	PRIMARY
AUTO_INCREMENT	☐	☐
Commenti		

#	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra	Azione
1	Temperatura	float			No	Nessuno		☐ ☐ Più ▼
2	Timestamp	timestamp			No	CURRENT_TIMESTAMP		☐ ☐ Più ▼

Fig. 4
Campi della tabella "acquario".

Fig. 5 - Contenuto della tabella dopo la creazione.

Fig. 6 - Il contenuto della tabella "acquario" dopo aver inserito sei valori della temperatura.

			Temperatura	Timestamp
☐			27	2015-11-08 11:51:01
☐			27.9	2015-11-08 11:51:05
☐			27.91	2015-11-08 11:51:08
☐			27.31	2015-11-08 11:51:13
☐			26.31	2015-11-08 11:51:18
☐			25.61	2015-11-08 11:51:25

Per prima cosa va quindi creato il sito web, se già non ne avete uno.

Per fare questo vi potete affidare ai tanti servizi di hosting disponibili a pagamento o gratuitamente; l'importante è che nel servizio sia inclusa la possibilità di disporre anche di un database MySQL. Una volta creato il sito web e attivata la base di dati, recatevi nella sezione nella quale sarà possibile gestire il vostro database (se ne avete più di uno sceglietene uno qualsiasi).

Una volta dentro, create una tabella (è comodo usare lo strumento di creazione sicuramente offerto) dal nome acquario con i seguenti campi (**Fig. 4**) rispettando maiuscole e minuscole:

- Temperatura
 - Tipo: FLOAT;
- Timestamp
 - Tipo: TIMESTAMP;
 - Predefinito: CURRENT_TIMESTAMP;
 - Indice: PRIMARY.

In questo modo avete creato una tabella con due parametri dove *Timestamp* (che, se non definito diversamente, è impostato automaticamente alla data e all'ora del server su cui è ospitato il database) è la chiave primaria, ovvero non potranno mai esserci due righe della tabella con lo stesso Timestamp. Se non volete usare lo strumento per la creazione delle tabelle, potete usare

la console (di solito la si trova sotto la voce *SQL*) e inserire il codice del **Listato 1**.

Poi cliccate su *Esegui* oppure *Salva*, a seconda dei casi. Il risultato sarà come quello mostrato in **Fig. 5**.

Per quanto riguarda la pagina web che verrà usata da Fishino per caricare la temperatura sul database, può essere utilizzato uno script PHP. La scelta è ricaduta sul PHP poiché, essendo un linguaggio di scripting server-side, si presta perfettamente al compito da svolgere: Fishino opera come client, si connette al server, invia la temperatura che viene elaborata e salvata su database. Per scrivere la pagina è sufficiente un editor di testo come il Blocco Note di Windows. Il file dovrà essere salvato, ad esempio come *in.php* (vi consigliamo di cambiare nome ma non sceglietene uno troppo lungo), e successivamente caricato sul vostro spazio web (preferibilmente nella directory radice) usando, ad esempio il pannello utente (o gestore file) del servizio di hosting. Vediamo insieme le parti di cui è composto lo script nel file *in.php*. Cominciate

aprendo il tag `<?php` (ricordatevi di chiuderlo con `?>` alla fine); da qui in poi seguirà il vostro script. Dovrete poi riportare (**Listato 2**) i parametri di connessione al vostro database (li trovate solitamente nel pannello di controllo): `$username`, `$password`, `$db`, `$tablename`, `$host`. I nomi delle variabili (ricordiamo che nel linguaggio PHP si indicano con il simbolo `$` davanti) possono essere anche cambiati, ricordandosi però di mantenere il nome scelto per tutto lo script. Alcuni servizi di hosting consigliano di lasciare vuoti i campi di username, password e host per due motivi: innanzitutto sono recuperati automaticamente, poi lo svantaggio del PHP è che spesso, in caso di errori di sintassi o di altro genere, viene stampato a schermo tutto il codice, compresi questi parametri personali. In ogni caso, se non fate errori, non c'è alcun rischio. La variabile `$tablename` rappresenta il nome della tabella; noi la abbiamo chiamata "acquario" ma se avete scelto un nome differente allora dovrete sostituirlo con quello da voi impostato.

Per fare interpretare al server i dati che gli verranno passati da Fishino UNO, usiamo il metodo GET (avremmo potuto usare anche il metodo POST) con il quale si fanno richieste ad un Web Server. Il metodo GET consiste nell'accodare i dati che ci interessano all'indirizzo della pagina richiesta. Per farlo si deve aggiungere, dopo l'indirizzo della pagina, un punto in-



Fig. 7 - Esempio di e-mail inviata a seguito di una temperatura critica.

Listato 2

```
<?php
//-----
// ----- TODO -----
$username="username";
$password="password";
$db="database";
$tablename = "acquario";
$host="host";
//-----
```

Listato 3

```
$temperatura_acquario = $_GET['t'];

echo "Temperatura acquario = " . $temperatura_acquario;
```

Listato 4

```
if ($con = mysqli_connect($host, $username, $password))
{
    mysqli_select_db($con,$db) or die ("Unable to select database");
    $query = "INSERT INTO $tablename VALUES ($temperatura_acquario, now())";
    $result = mysqli_query($con,$query);
}
else
{
    echo('Unable to connect to database.');
```

terrogativo seguito dalle coppie *nome = valore* dei dati di interesse. Se vi è più di una coppia, esse vanno separate dal simbolo &. Il risultato sarà una variabile globale `$_GET` contenente l'array di tutte le coppie campo=valore che potremo usare nello script. Nel nostro caso avremo solo una coppia formata dalla temperatura con relativo valore. Vorremmo quindi passa-

re, ad esempio, la coppia *temperatura=25.6* (con il punto, non la virgola) al Web Server in modo tale da farglielo interpretare e, successivamente, caricare sul database. Per fare questo si dovrà assegnare alla variabile `$temperatura_acquario` il valore del campo *temperatura* passato con l'array `$_GET`. Se, ad esempio, chiamate il campo della temperatura con la

lettera t, allora si potrà selezionare questo specifico elemento dell'array rappresentato dalla variabile `$_GET` scrivendo `$temperatura_acquario = $_GET['t']` (ricordando gli apici prima e dopo t). In questo modo, digitando nella barra degli indirizzi del browser `www.miosito.it/in.php?t=25.6`, come risultato avrete che la variabile `$temperatura_acquario` sarà pari a 25,6. Se poi voleste visualizzare nel browser il dato appena inserito, basterà mostrarlo in output con la funzione `echo` (o `print`, o una funzione con lo stesso compito). È da notare che se vi dovessero essere errori di sintassi, di battitura o di altra natura nella query MySQL, questi non verrebbero notificati ma, semplicemente, non verrebbe caricato alcun dato sul database. In caso di errori vedreste semplicemente la temperatura in output ma, controllando nei dati contenuti nel database, vedreste una tabella vuota. Per verificare di non aver commesso errori, dopo aver fatto la prova appena descritta (inserire la temperatura di 25,6), controllate che il dato sia stato effettivamente salvato nel database. Se tutto è avvenuto senza errori allora la tabella non sarà vuota ma conterrà i dati appena inseri-

Listato 5

```
//Invio mail per temperature critiche
if ($temperatura_acquario < 23)
{
    $messaggio = "La temperatura dell'acquario e' troppo BASSA! Rilevata temperatura di " .
    $temperatura_acquario . "°C in data: " . date("Y-m-d H:i:s");
    mail('mail@mail.com','Temperatura acquario CRITICA',$messaggio);
}
if ($temperatura_acquario > 29)
{
    $messaggio = "La temperatura dell'acquario e' troppo ALTA! Rilevata temperatura di " .
    $temperatura_acquario . "°C in data: " . date("Y-m-d H:i:s");
    mail('mail@mail.com','Temperatura acquario CRITICA',$messaggio);
}

?>
```


ti. Vi consigliamo di provare ad inserire più di un dato, così da popolare la tabella come mostrato in **Fig. 6**.

Questo tornerà utile quando dovrete testare il corretto funzionamento dello script che mostrerà il grafico (ne parleremo più avanti). È giunto il momento di salvare i dati nel database (**Listato 4**).

A questo scopo, dovrete innanzitutto usare la funzione `mysqli` per far connettere la vostra applicazione Web al Database Manager MySQL e, successivamente, selezionare il database di interesse. Con `if ($con = mysqli_connect($host, $username, $password))` viene tentata la connessione e, in caso di fallimento, viene mostrato l'errore *Unable to connect to database*.

Se invece va a buon fine, viene tentato il collegamento al database con `mysqli_select_db($con,$db)`; anche qui è previsto un messaggio di errore in caso di fallimento.

La funzione `mysqli` accetta anche il nome del database come quarto parametro; potreste quindi usare un'unica funzione

Listato 6

```
<html>
<head>
<title>Acquario</title>
</head>
<body>
<h1 align="center">ACQUARIO </h1>
<p align="center">- based on Fishino UNO - </p>

<?php

    $max_value = 100; //Quanti valori mostra al massimo nel grafico

    //----- TODO -----
    // -----
    $username="username";
    $password="password";
    $db="database";
    $tablename = "acquario";
    $host="host";
    //-----
```

Listato 7

```
$con = mysqli_connect($host, $username, $password);
mysqli_select_db($con, $db) or die('Errore connessione...');
```

per far connettere l'applicazione al Database Manager MySQL e selezionare un database: `mysqli_connect($host, $username, $password, $db)`.

La variabile `$con` è l'identificativo di connessione che servirà per comunicare con il database. Una volta avvenuta la connessione verrà mandata una query (una interrogazione di un data-

base) al database con la funzione `mysqli_query(identificativo, query)`, dove l'identificativo di connessione è `$con` (se mancante, verrà scelto l'ultimo identificativo usato) e la query è indicata come stringa in `$query`. Con questa query specifichiamo di inserire nella tabella `$tablename` (nel nostro caso, `acquario`) i valori di temperatura e orario nei campi

Listato 8

```
$q6 = "SELECT Temperatura, Timestamp FROM $tablename ORDER BY Timestamp DESC";

$i = 0;

$r6 = mysqli_query($con,$q6);
$n_rilevazioni = mysqli_num_rows($r6);
while((list($temp_water[$i], $temp_time[$i]) = mysqli_fetch_row($r6)) and $i<$max_value)
    $i = $i+1;

$i = $i - 1;

?>
```

Listato 9

```
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
```

```

<script type="text/javascript">
    google.load('visualization', '1', {packages: ['corechart', 'line']});
    google.setOnLoadCallback(drawBackgroundColor);

    function drawBackgroundColor() {
        var data = new google.visualization.DataTable();
        data.addColumn('date', 'Time');
        data.addColumn('number', 'Temp');

        data.addRows([
            <?php
                $j = 0;
                while($i >= 0 and $temp_water[$i] != 0)
                {
                    echo "[";
                    echo "new Date(";
                    $temp_date = date("Y,m-l,d,H,i,s", strtotime($temp_time[$i]));
                    echo $temp_date;

                    echo ")";
                    echo ",";
                    echo $temp_water[$i];
                    echo "]";
                    if ($i > 0)
                    {
                        echo ",";
                    }
                    $j = $j + 1;
                    $i = $i - 1;
                }

            ?>

        ]);

        var options = {
            hAxis: {
                title: 't'
            },
            vAxis: {
                title: 'gradi'
            },
            focusTarget: 'category',
            curveType: 'function',
            lineWidth: 3,
            title: 'Temperatura acquario degli ultimi giorni',
            backgroundColor: 'white'
        };

        var chart = new google.visualization.LineChart(document.getElementById('chart_div'));
        chart.draw(data, options);
    }

</script>

<!-- Portions of this code are modifications based on work created and shared by Google and used according to
terms described in the Creative Commons 3.0 Attribution License. Code samples are licensed under the Apache
2.0 License. -->

```

del database. Con la funzione **now()** viene ritornato l'orario attuale del Web server nel formato YYYY-MM-DD HH:MM:SS. Siccome nella definizione della tabella del database avete impostato per il campo Timestamp **CURRENT_TIMESTAMP** come valore predefinito, anche passando **NULL** al posto di **now()**

non sorgerebbero problemi. Potreste aggiungere la possibilità di ricevere una e-mail in caso di temperature critiche (un esempio è in **Fig. 7**). Questo può essere fatto con la funzione mail, la cui sintassi è: *bool mail (string \$a , string \$oggetto , string \$messaggio [, string \$header_addizionali [, string \$parametri_addizionali*

]]). Tralasciando i parametri opzionali (tra parentesi quadre) è evidente come questa funzione permetta di inviare automaticamente il messaggio *\$messaggio* all'indirizzo *\$a* con oggetto della e-mail *\$oggetto*. La funzione può essere chiamata quando la temperatura è fuori da un certo range (nell'esempio, sotto i 23°C

Listato 11

```
<p align="right"><?php echo "Dati rilevati: #" . $n_rilevazioni; ?> </p>
<p>&nbsp;</p>

<?php
    echo "<br /><br /><strong>Ultima temperatura rilevata: " . $temp_water[0] . "&deg;C in data: " .
    $temp_time[0] . "</strong>";
?>

</body>
</html>
```

e sopra i 29°C).

GRAFICO DELLA TEMPERATURA

Per il grafico sul quale visualizzare le ultime temperature rilevate, la soluzione più semplice è sfruttare le molte librerie reperibili sul web; quella da noi utilizzata è la soluzione offerta da Google, che mette a disposizione gratuitamente un gran numero di grafici ottimizzati per i vari dispositivi, personalizzabili in molti aspetti e facili da utilizzare. I *Google charts*, la cui documentazione è reperibile all'indirizzo web <https://developers.google.com/chart/>, sono liberamente utilizzabili e forniti sotto licenza Creative Commons Attribution 3.0, mentre gli esempi di codice sono concessi sotto licenza Apache 2.0. Il grafico che vi mostriamo nel nostro esempio è scelto dalla

categoria "Line Charts", con sfondo bianco, linea del grafico blu e con andamento curvilineo. Sull'asse delle ascisse rappresentiamo l'andamento del tempo, mentre sull'asse delle ordinate vi sarà la temperatura. Ogni singolo dato avrà poi un'etichetta, con data e temperatura in formato testuale, che comparirà al passaggio del cursore del mouse (o toccando lo schermo, per i dispositivi touch). Anche per visualizzare il grafico dovrete realizzare uno script PHP, chiamando il file *acquario_grafico.php*. Iniziate (**Listato 6**) con i tag di apertura di una normale pagina HTML (html, head, title, body) e aprite poi il tag PHP con *<?php*. Con la variabile *\$max_value* definite quanti dati mostrare nel grafico: sceglietene il valore considerando che verrà inviato e caricato un dato ogni

30 minuti quindi; per vedere il grafico di un giorno intero, il valore della variabile dovrà essere almeno pari a 48. Nel nostro esempio poniamo questo valore pari a 100, tanto da coprire poco più di 2 giorni.

Infine inserite i parametri di connessione al database come già avete fatto per il file *in.php*. Allo stesso modo facciamo collegare l'applicazione web al database: ciò si può fare come nel file *in.php* oppure in modo più abbreviato, come mostrato nel **Listato 7**.

Per prelevare le temperature dal database dovremo impostare la query *\$q6*; stavolta specifichiamo che vogliamo recuperare le temperature con relative date dalla tabella *acquario*, ordinate a partire dalla più recente. Dopo aver interrogato il database con la funzione *mysql_query*

File	Dimensione	Data
 <i>acquario_grafico.php</i>	3 KB	08/11/2015
 <i>in.php</i>	1 KB	08/11/2015

Fig. 8 - Contenuto del "Gestore file" del sito web.

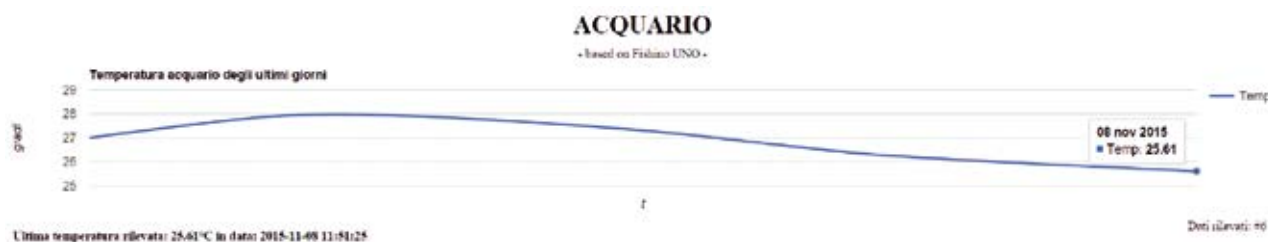


Fig. 9 - Aspetto del grafico con focus su uno dei dati.

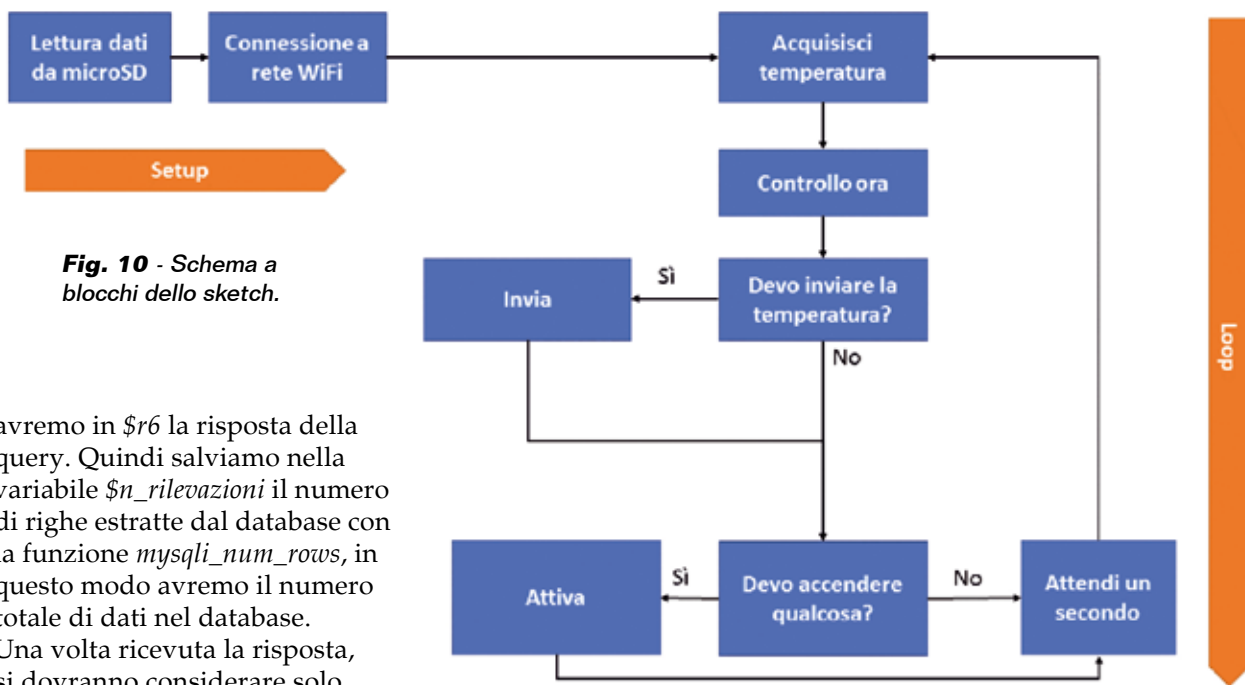


Fig. 10 - Schema a blocchi dello sketch.

avremo in `$r6` la risposta della query. Quindi salviamo nella variabile `$n_rilevazioni` il numero di righe estratte dal database con la funzione `mysql_num_rows`, in questo modo avremo il numero totale di dati nel database. Una volta ricevuta la risposta, si dovranno considerare solo le prime 100 righe di ciò che il database ha restituito. Per fare ciò, usate la funzione `mysql_fetch_row(risultato)` che ha come parametro il risultato dell'interrogazione del database (la variabile `$r6` nel nostro caso). Questa funzione carica in un array una riga del risultato associato all'identificativo specificato, oppure restituisce NULL se non ci sono più righe.

Ciascun elemento di questo array sarà formato dalla coppia temperatura-data. Usate il costrutto `list()` per assegnare il valore dei due campi a due variabili rappresentate ciascuna da una cella di un array (`$temp_water[$i]` e `$temp_time[$i]`, dove `$i` è una variabile inizializzata a 0 e incrementata di 1 ad ogni ci-

clo). Questo costrutto è usato per assegnare valori a una lista di variabili in una sola operazione. Con il ciclo `while` nel **Listato 8** viene specificato che, per ciascuna riga restituita dal database, si deve salvare la temperatura e la data nei due array indicati rappresentanti rispettivamente il vettore delle temperature e

quello delle date. Se il numero di righe restituite è maggiore del massimo degli elementi voluti, il ciclo termina quando si sono esauriti gli elementi; se invece il numero di righe è maggiore, allora il ciclo termina quando la variabile contatore `$i` raggiunge il valore massimo. Una volta fuori dal ciclo `while`,

Listato 12

```

File myFile;

//Pin DATA di DS18B20: pin 2
OneWire ds(2);

// Initialize the Fishino client library
FishinoClient client;

//Credenziali rete WiFi
#define MY_SSID    "your_network"
#define MY_PASS    "your_password"

/* Definisco i pin di attivazione dei relé.
 * BOLLE (Aeratore) = Pin 8
 * PLAFO (Plafoniera + CO2) = Pin 9
 */
#define BOLLE 8
#define PLAFO 9

//Numero massimo di fasce orarie nel file contenuto nella microSD
#define MAX_VAL 4

RTC_DS1307 RTC;

//Matrice usata per memorizzare gli orari letti da microSD
int orari_bolle_plaf[MAX_VAL][7];

```

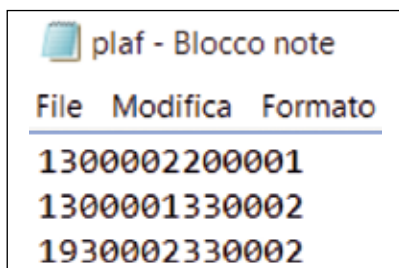


Fig. 11 - Esempio di file `plaf.txt`.

Listato 13

```
void setup() {
  Serial.begin(115200);

  //Inizializzo gli elementi della matrice a 0
  for (int t = 0; t < MAX_VAL; t++)
    for (int j = 0; j < 7; j++)
      orari_bolle_plaf[t][j] = 0;

  //Leggo il file nella microSD e inserisco gli orari nella matrice
  leggi_SD();

  //Relé non eccitati
  pinMode(PLAFO, OUTPUT);
  pinMode(BOLLE, OUTPUT);
  digitalWrite(PLAFO, LOW);
  digitalWrite(BOLLE, LOW);

  // only for Leonardo needed
  // necessario solo per la Leonardo
  while (!Serial)
    ;
  // initialize SPI
  // inizializza il modulo SPI
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV2);

  // reset and test WiFi module
  // resettata e testa il modulo WiFi
  while (!Fishino.reset())
    Serial << F("Fishino RESET FAILED, RETRYING...\n");
  Serial << F("Fishino WiFi RESET OK\n");

  // go into station mode
  // imposta la modalità stazione
  Fishino.setMode(STATION_MODE);

  // Initialize the Wifi connection.
  // Inizializza la connessione WiFi
  Serial << F("Setting up the Wifi connection...\n");

  // try forever to connect to AP
  // tenta la connessione finché non riesce
  while (!Fishino.begin(MY_SSID, MY_PASS) == STATION_GOT_IP)
    Serial << F("ERROR CONNECTING TO AP, RETRYING...\n");

  Serial << F("SUCCEFULLY CONNECTED TO AP\n");

#ifdef IPADDR
  Fishino.config(ip);
#else
  Fishino.startDHCP();
#endif
  delay(1000);
}
```

sarà necessario decrementare di 1 la variabile *\$i* in modo da avere l'indice dell'array *\$temp_water* riferito all'ultimo valore inserito (se le righe restituite sono più di 100, ovviamente il valore di *\$i* sarà 100). In questo modo avete due array: uno contenente tutte le temperature da rappresentare nel grafico, l'altro con le date relative all'acquisizione. È il momento di inserire questi valori in un grafico; per sfruttare quello

offerto da Google aggiungete le librerie necessarie (**Listato 9**). Successivamente basta impostare qualche parametro legato all'aspetto del grafico e fornire i dati da rappresentare. L'aspetto più interessante è il passaggio dei dati, che avviene sotto forma di tabella dove le colonne costituiscono i campi da rappresentare nell'ordine ascisse e ordinate, mentre le righe sono i dati veri e propri.

Nel **Listato 10** vediamo infatti la variabile *data*, rappresentante i dati, che permette di aggiungere colonne con il metodo *addColumn(tipo, nome)* e righe con il metodo *addRows*.

Per evitare di dover aggiungere manualmente una riga per ogni elemento si può sfruttare un ciclo che aggiunga iterativamente righe separate da virgola fino a che non arriva al penultimo elemento, dopo il quale la virgola non è più necessaria. Alla fine del ciclo si avrà un risultato del tipo: *data.addRows([[data, temperatura], [data, temperatura], [data, temperatura]])*. Associando il grafico al contenitore *chart_div*, sarà possibile mostrarlo in output:

```
<div id="chart_div"></div>
```

Prima di vedere il risultato finale vi mostriamo come sia possibile visualizzare in output il numero totale di dati nel database e l'ultima temperatura rilevata con il relativo orario (**Listato 11**).

Una volta salvato il file (ricordatevi di chiudere tutti i tag aperti), caricatelo sul vostro sito web tramite il gestore file (**Fig. 8**). Il grafico sarà poi visibile (**Fig. 9**) digitando ad esempio www.mio-sito.it/acquario_temperatura.php.

LO SKETCH

Passiamo ora a schematizzare in pochi passi la logica dello sketch e lo facciamo con l'aiuto della **Fig. 10**. Inizialmente si includono le necessarie librerie:

```
#include <Wire.h>
#include <RTCLib.h>
#include <Flash.h>
#include <FishinoUdp.h>
#include <FishinoSockBuf.h>
#include <Fishino.h>
#include <SPI.h>
#include <SD.h>
```

Listato 14

```

/* Funzione che legge il file plaf.txt nella microSD
 * e ne estrae gli orari inserendoli nella matrice
 * degli orari.
 * CS - pin 4*/
void leggi_SD()
{
    String orari = "";
    if (!SD.begin(4)) {
        return;
    }
    int i;
    char c;
    myFile = SD.open("plaf.txt");
    if (myFile) {
        i = 0; //Numero di righe trovate nel file
        while (myFile.available()) {
            c = myFile.read();
            orari += c;
            if (c == '\n') //C'è ancora una riga
                ++i;
        }
        myFile.close();
    }
    int x = 0;
    int cont = 0;
    for (int j = 0; j <= i; j++)
    {
        while (orari[cont] != '\n')
        {
            switch (x)
            {
                case 0:
                    orari_bolle_plaf[j][0] += (orari[cont] - 48) * 10; //ORE_ON
                    break;
                case 1:
                    orari_bolle_plaf[j][0] += (orari[cont] - 48);
                    break;
                case 2:
                    orari_bolle_plaf[j][1] += (orari[cont] - 48) * 10; //MINUTI_ON
                    break;
                case 3:
                    orari_bolle_plaf[j][1] += (orari[cont] - 48);
                    break;
                case 4:
                    orari_bolle_plaf[j][2] += (orari[cont] - 48) * 10; //SECONDI_ON
                    break;
                case 5:
                    orari_bolle_plaf[j][2] += (orari[cont] - 48);
                    break;
                case 6:
                    orari_bolle_plaf[j][3] += (orari[cont] - 48) * 10; //ORE_OFF
                    break;
                case 7:
                    orari_bolle_plaf[j][3] += (orari[cont] - 48);
                    break;
                case 8:
                    orari_bolle_plaf[j][4] += (orari[cont] - 48) * 10; //MINUTI_OFF
                    break;
                case 9:
                    orari_bolle_plaf[j][4] += (orari[cont] - 48);
                    break;
                case 10:
                    orari_bolle_plaf[j][5] += (orari[cont] - 48) * 10; //SECONDI_OFF
                    break;
                case 11:
                    orari_bolle_plaf[j][5] += (orari[cont] - 48);
                    break;
                case 12:
                    orari_bolle_plaf[j][6] += (orari[cont] - 48); //1 (plafoniere) o 2 (bolle)
                    break;
            }
            ++x;
            ++cont;
        }
        ++cont;
        x = 0;
    }
    orari = "";
}

```

```

#include <OneWire.h>
#include <avr/io.h>
#include <avr/wdt.h>

```

Molte di esse saranno già presenti se avete scaricato l'IDE di Arduino mentre altre, come quelle di Fishino (scaricabili dal sito www.fishino.it o www.fishino.com), RTCLib.h per usare il modulo Real Time Clock (scaricabile da <https://github.com/adafruit/RTCLib>) e OneWire (scaricabile ad esempio da <http://playground.arduino.cc/Learning/OneWire>) dovranno essere aggiunte nella directory *Libraries* di Arduino.

Successivamente andranno definite alcune variabili (**Listato 12**): va dichiarato, ad esempio, il pin al quale è collegata la sonda DS18B20 (nel nostro caso il 2), va inizializzata la libreria client di Fishino e va definito il nome della rete WiFi con relativa password (*MY_SSID* e *MY_PASS*). Si dovranno poi definire i pin con i quali si attiveranno e disattiveranno i due relé. Con *MAX_VAL* potete specificare quante fasce orarie massime volete considerare, non è consigliabile inserire un valore alto per poi inserire pochi orari. Ad esempio se vorrete accendere una sola volta la plafoniera di illuminazione e, magari, due volte l'aeratore, allora potrete mettere come *MAX_VAL* un valore poco più grande di 3 (ad esempio 4 o 5) così da non sprecare memoria e da poter aggiungere in futuro qualche fascia oraria senza ricompilare lo sketch ma semplicemente modificando il file sulla microSD.

Infine, sarà usata una matrice *orari_bolle_plaf[MAX_VAL][7]* per poter salvare gli orari contenuti nella microSD in modo da non doverla aprire ad ogni

Listato 15

```
// Funzione per calcolare la temperatura dell'acqua in gradi Celsius tramite il DS18B20
float getTemp() {
    byte data[12];
    byte addr[8];

    if ( !ds.search(addr)) {
        //no more sensors on chain, reset search
        ds.reset_search();
        return -1000;
    }

    if ( OneWire::crc8( addr, 7) != addr[7]) {
        return -1000;
    }

    if ( addr[0] != 0x10 && addr[0] != 0x28) {
        return -1000;
    }

    ds.reset();
    ds.select(addr);
    ds.write(0x44, 1); // start conversion, with parasite power on at the end

    byte present = ds.reset();
    ds.select(addr);
    ds.write(0xBE); // Read Scratchpad

    for (int i = 0; i < 9; i++) { // we need 9 bytes
        data[i] = ds.read();
    }

    ds.reset_search();

    byte MSB = data[1];
    byte LSB = data[0];

    float tempRead = (MSB << 8) | LSB); //using two's compliment
    float TemperatureSum = tempRead / 16;

    return TemperatureSum;
}
```

Listato 16

```
/* Funzione per decidere se il tempo attuale sia in un intervallo o meno.
 * Accetta 6 parametri:
 * - Prima ora
 * - Primo minuto
 * - Seconda ora
 * - Secondo minuto
 * - Ora attuale
 * - Minuto attuale
 * Restituisce true o false se il calcolo ha esito affermativo o meno*/
boolean ora_giusta(int ora1, int minuto1, int ora2, int minuto2, int ora_ adesso, int minuto_ adesso)
{
    //Considero subito una risposta affermativa
    boolean risposta = true;

    //Vado a cercare se sono fuori dall'intervallo
    if (ora_ adesso < ora1 || ora_ adesso > ora2)
        risposta = false;
    if (ora1 == ora2 && minuto1 == minuto2)
        risposta = false;
    if (ora_ adesso == ora1 && minuto_ adesso < minuto1)
        risposta = false;
    if (ora_ adesso == ora2 && minuto_ adesso >= minuto2)
        risposta = false;
    if (ora1 > ora2 || (ora1 == ora2 && minuto1 > minuto2))
        risposta = false;

    return risposta;
}
```

ciclo, ma solo una volta sola. Questa matrice avrà tante righe quante sono le fasce orarie (le restanti saranno poste a zero) e le colonne indicheranno rispettivamente:

1. ora di accensione;
2. minuto di accensione;
3. secondo di accensione;
4. ora di spegnimento;
5. minuto di spegnimento;
6. secondo di spegnimento;
7. carattere per distinguere tra illuminazione ed erogatore di anidride carbonica o aeratore.

SETUP()

Nel **Listato 13** vediamo l'inizializzazione di Fishino nel *Setup*: si può notare come Fishino sia in modalità stazione. La matrice degli orari è inizialmente riempita di zeri ma, tramite la funzione **leggi_SD()**, viene progressivamente popolata dagli orari definiti da microSD.

LEGGI_SD()

La microSD è letta tramite la funzione **leggi_SD()** che non necessita di alcun parametro e si limita ad aprire il file *plaf.txt* (**Fig. 11**) contenuto nella microSD. All'interno di questo file saranno contenuti gli orari di accensione e spegnimento dei due relé nel formato *HHMMSSHHMMSSn* dove *n* è un carattere 1 o 2 a seconda che si tratti di un orario riferito al relé dell'illuminazione e dell'anidride carbonica oppure di quello dell'aeratore. Se, ad esempio, volessimo accendere l'aeratore tra le 10 e le 11 del mattino, dovremmo scrivere: *1000001100001*. Aggiungete una nuova riga di testo se e solo se volete aggiungere un nuovo orario, non lasciate altrimenti righe vuote. Una volta letti tutti i caratteri del file *plaf.txt* e averli inseriti nella stringa *orari*, questa viene analizzata e, in base alla posizio-


```

void loop() {
  //Imposto il WatchDog a 8 secondi
  wdt_enable(WDTO_8S);
  // server address:
  char server[] = "www.miosito.it";
  float temperatura = 0.0;
  while (temperatura < 1 || temperatura > 84) //Evita di stampare dati impossibili
    temperatura = getTemp();
  //Inizializza stringa da inviare al server
  String temperature = "";
  //Data attuale
  DateTime now = RTC.now();
  //Queste variabili non sono necessarie, sono usate solo per comodità
  int ore = now.hour();
  int minuti = now.minute();
  int secondi = now.second();
  int anno = now.year();
  //Ad ogni inizio ora e ogni mezza...
  if ((minuti == 0 || minuti == 30) && (secondi == 0 || secondi == 1))
  {
    // close any connection before send a new request.
    // This will free the socket on the WiFi module
    // chiude ogni ventuale connessione prima di inviare una nuova richiesta
    // quest libera il socket sull modulo WiFi
    client.stop();
    delay(1000);
    //Preparo la stringa da inviare al server
    temperature = "GET /in.php?t=";
    temperature += temperatura;
    temperature += " HTTP/1.1";
    // if you get a connection, report back via serial:
    // se ottiene una connessione, stampa i dati sulla seriale
    if (client.connect(server, 80))
    {
      //Serial.println("connected to server");
      // Make a HTTP request:
      client.println(temperature);
      //TODO
      client.println("Host: www.miosito.it");
      client.println("User-Agent: ArduinoWiFi/1.1");
      client.println("Connection: close");
      client.println();
    }
  }
  /* Variabili per decidere se accendere o spegnere i relé:
  * false = spento
  * true = acceso
  * Inizialmente li considero spenti*/
  boolean bolle_on = false, plaf_on = false;
  /*Scorro la matrice degli orari per controllare se deve essere acceso qualcosa
  * (nel caso si trovi qualcosa da accendere pongo a true le variabili bolle_on o plaf_on)*/
  for (int k = 0; k < MAX_VAL; k++)
  {
    if (orari_bolle_plaf[k][0] + orari_bolle_plaf[k][1] + orari_bolle_plaf[k][3] + orari_bolle_plaf[k][4] != 0)
    {
      if (ora_giusta(orari_bolle_plaf[k][0], orari_bolle_plaf[k][1], orari_bolle_plaf[k][3], orari_bolle_plaf[k][4], ore, minuti) == true)
      {
        switch (orari_bolle_plaf[k][6])
        {
          case 1:
            plaf_on = true;
            break;
          case 2:
            bolle_on = true;
            break;
        }
      }
    }
  }
  //Accesi (true) o spenti (false)?
  if (plaf_on == false)
    digitalWrite(PLAFO, LOW);
  else
    digitalWrite(PLAFO, HIGH);
  if (bolle_on == false)
    digitalWrite(BOLLE, LOW);
  else
    digitalWrite(BOLLE, HIGH);
  //Resetto il Fishino una volta al giorno
  if (ore == 23 && minuti == 59 && (secondi >= 0 || secondi <= 5))
    delay(15000); //Lascio scadere gli 8 secondi del WatchDog
  //Attendo un secondo
  delay (1000);
  //Resetto il contatore del WatchDog
  wdt_reset();
}

```

ne del carattere in esame, viene inserito in una cella della matrice il valore calcolato differenziando tra decine e unità (**Listato 14**). Ad esempio la stringa "34" viene trasformata nel numero 34 moltiplicando il 3 per 10, aggiungendolo al 4, il tutto dopo aver convertito da *char* a *intero*.

GETTEMP()

La funzione **getTemp()**, nel **Listato 15**, non è stata modificata dalla funzione che si trova anche nella libreria OneWire. Il suo scopo è quello di calcolare la temperatura in base a quanto misurato dal sensore DS18B20. La funzione restituisce un float rappresentante la temperatura in gradi Celsius.

ORA_GIUSTA

La funzione **ora_justa** (**Listato 16**) accetta come parametri tre orari formati da ore e minuti e controlla se il terzo orario si trova tra i primi due. La funzione non controlla se i due orari sono invertiti ma si potrebbe aggiungere tale opzione. In ogni caso è sufficiente rispettare l'ordine con il quale li si inserisce. Inizialmente, nella funzione, la variabile *risposta* è posta a true. Successivamente si verifica se l'orario in esame si trovi fuori dall'intervallo analizzato, in caso affermativo la variabile *risposta* è posta a false. Alla fine dei confronti viene restituito il valore di *risposta*. Se tale valore è rimasto a true allora significa che l'orario analizzato si trova tra gli altri due.

LOOP()

Passiamo ora ad analizzare il corpo principale dello sketch, la funzione **Loop()** (**Listato 17**). All'inizio del ciclo viene impostato il WatchDog a 8 secondi così

da resettare Fishino in caso di blocchi o cicli infiniti causati da errori imprevedibili.

Viene poi definito il server al quale connettersi, basta indicare in *server[]* l'indirizzo del vostro sito Web sul quale avete caricato il file *in.php*.

In seguito viene letto il valore della temperatura dalla sonda, evitando errori di misura con un ciclo while.

Ogni 30 minuti deve avvenire la connessione al server per inviare la temperatura, quindi si chiudono le eventuali connessioni ancora aperte e si prepara la stringa da inviare.

La stringa sarà composta dal GET, che abbiamo già trattato, e dal percorso del file *in.php* sul sito Web (se lo avete messo nella directory radice sarà sufficiente inserire */in.php*) seguito dal campo *t* (quello che verrà usato dallo script PHP per recuperare il valore della temperatura) con relativo valore. Infine viene effettuata la connessione al Web server e inviata la temperatura con il metodo GET.

Ricordatevi di cambiare anche l'indirizzo host in *client.println("Host: www.miosito.it")* inserendo, ancora una volta, l'indirizzo del server.

Successivamente si scorre la matrice degli orari per decidere se i relé devono essere accesi o spenti e, per far questo, si usano due variabili *bolle_on* e *plaf_on* inizialmente poste a false che, se la funzione *ora_justa* ritorna true, allora vengono poste a true. Alla fine del ciclo vengono eccitati solo i relé la cui variabile booleana relativa è posta a true, altrimenti vengono disattivati. Per sicurezza, una volta al giorno (alle 23:59) Fishino è resettato, così da azzerare tutti i contatori e non incorrere in blocchi inutili. Il reset è ottenuto facendo attende-

re 15 secondi così da far scadere il WatchDog.

Questa parte può ovviamente essere evitata.

Viene poi resettato il contatore del WatchDog dopo aver atteso un secondo. Infine viene rieseguito il ciclo di *loop()*.

Se non avete mai impostato l'orario dell'RTC di Fishino potete farlo facilmente caricando lo sketch *Softrtc* che trovate negli esempi della libreria *RTCLib*.

SVILUPPI FUTURI

Il sistema presentato svolge solo alcune delle funzioni che potrebbe compiere nel settore dell'acquariofilia; con qualche sensore in più e alcune modifiche al software, potreste ad esempio utilizzare il valore della temperatura per far gestire a Fishino l'aspetto del raffreddamento e del riscaldamento. Sulla base del file *plaf.txt*, potreste inserire un ulteriore file nella microSD e usarlo per definire il colore di strisce LED RGB per dare un tono allegro all'ambiente. ■

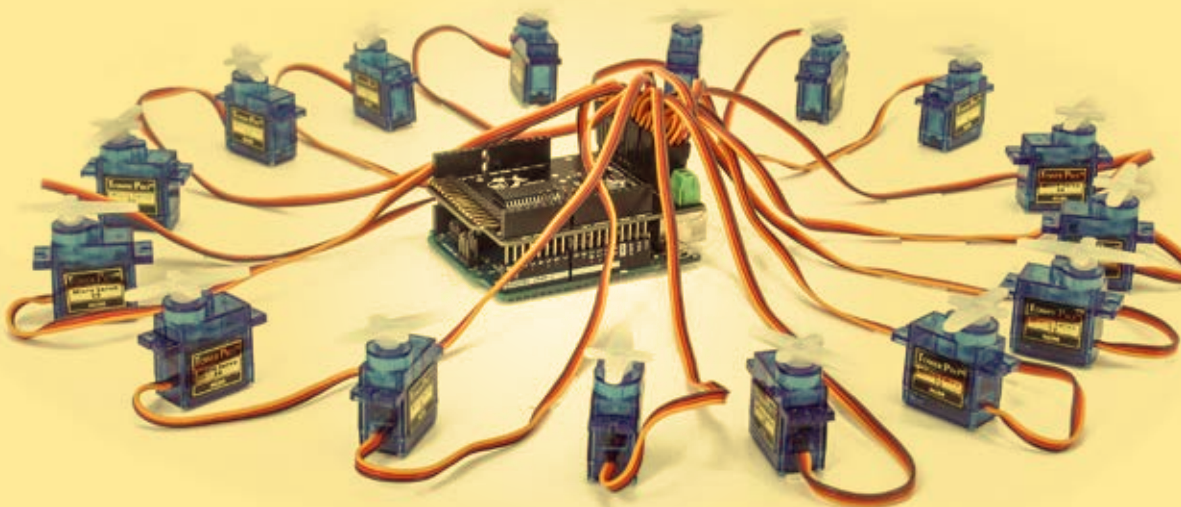


per il MATERIALE

Tutto il materiale necessario per realizzare questo progetto è disponibile presso Futura Elettronica. La board Fishino (cod. FISHINOONO) viene fornita montata e collaudata al prezzo di Euro 36,00. La sonda di temperatura waterproof con DS18B20 (cod. SONDADS18B20) è in vendita a Euro 7,00. Il modulo 1 relé 5 Vdc 10A viene fornito montato al prezzo di Euro 5,00. I prezzi si intendono IVA compresa.

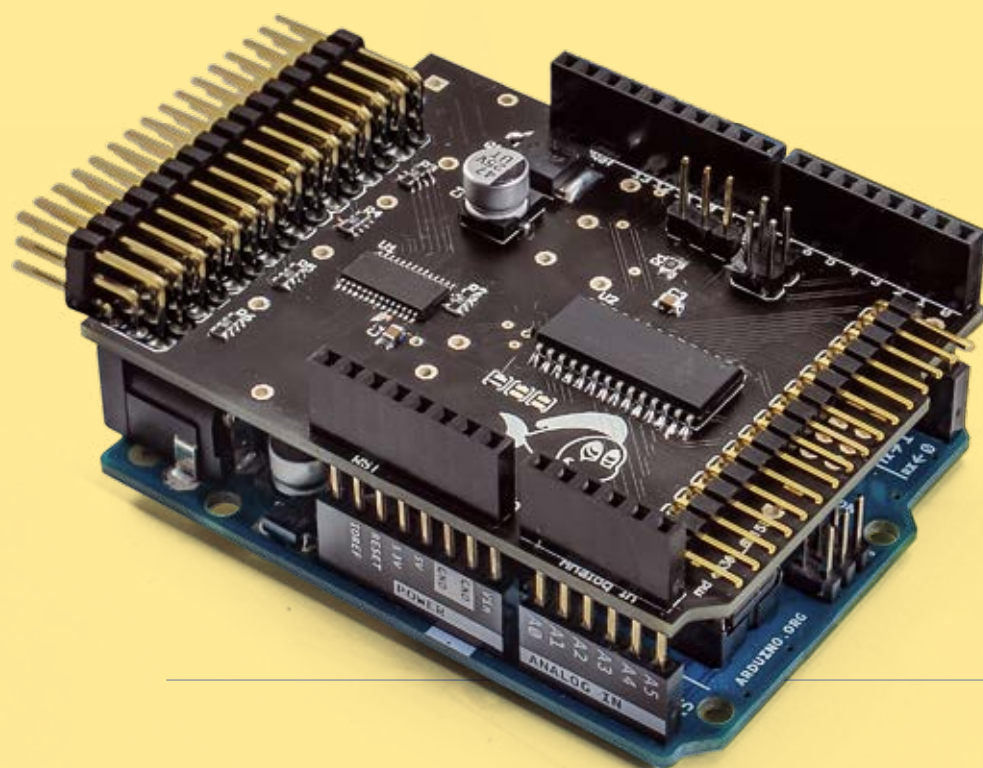
Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287
<http://www.futurashop.it>

Shield che rende disponibili 16 I/O digitali o altrettanti PWM impiegando un solo canale I²C-Bus di Arduino. Ideale per gestire LED multicolore o servomotori.



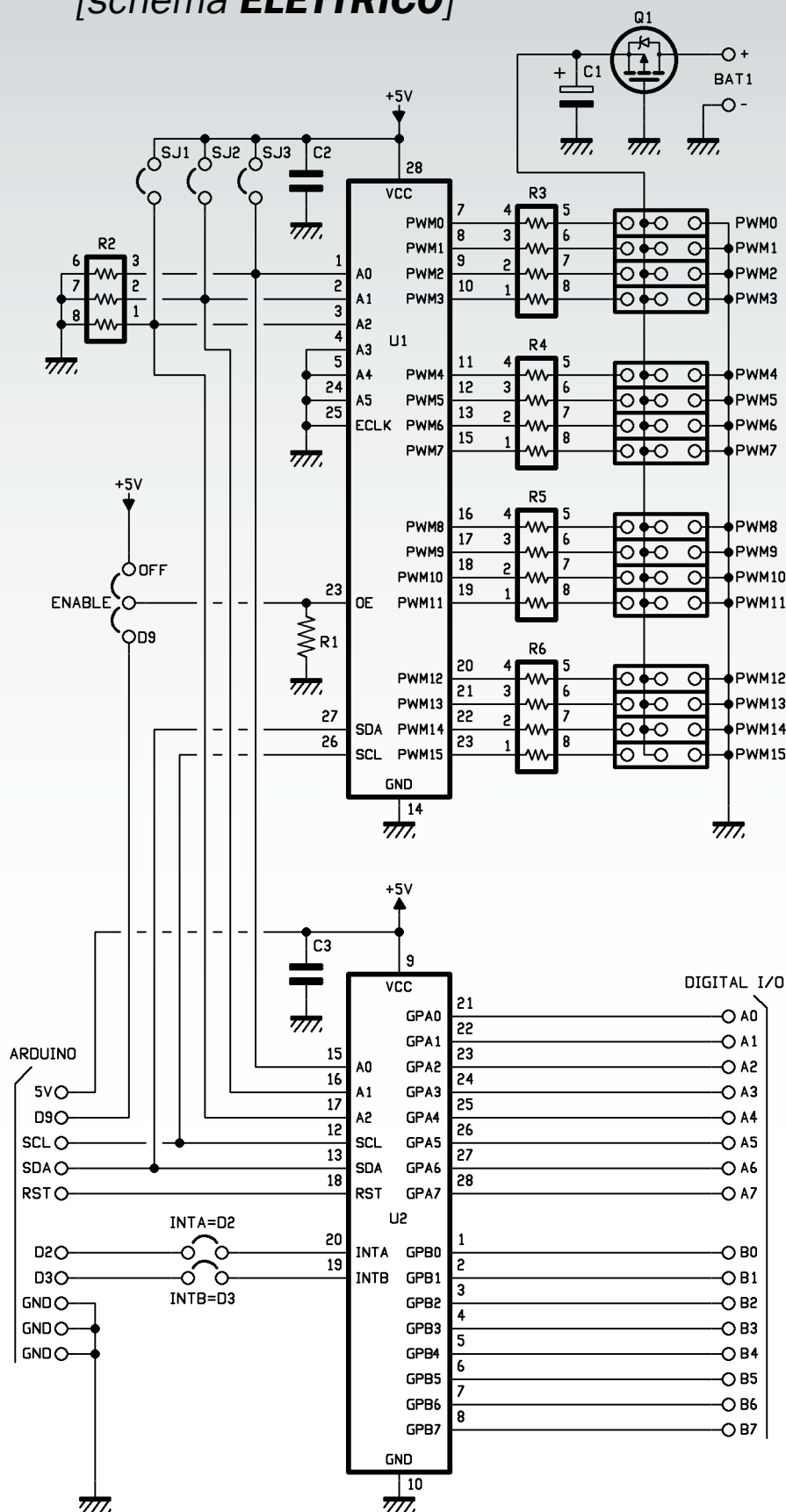
OCTOPUS, ESPANSIONE “TENTACOLARE” PER ARDUINO/FISHINO

..... di MASSIMO DEL FEDELE



Per quanto pratiche e capaci di realizzare innumerevoli applicazioni, le schede Arduino e compatibili hanno due limiti: la memoria di programma relativamente ridotta e la ridotta quantità di uscite disponibili, specie di I/O cui si può assegnare un segnale PWM. Per esempio, una Arduino/Fishino UNO dispone di sole sei uscite PWM e, a meno di non generare i relativi segnali via software (con notevole impegno del processore), permette il pilotaggio di un solo driver e quindi un solo LED RGBW di potenza, o in alternativa di sei led monocromatici.

[schema **ELETTRICO**]



Lo stesso limite emerge quando si vogliono pilotare più di 6 servomotori con le stesse schede; la realizzazione di un robot tipo "hexapod", che richiede ben 12 servi, risulta problematica, se non impossibile. Anche gli ingressi e le uscite digitali sono limitati; sempre parlando delle schede Arduino, abbiamo un totale di 13 I/O digitali e 6 ingressi analogici, utilizzabili anch'essi in digitale; sembrerebbero anche abbondanti, se non fosse che molti di questi vengono utilizzati per le periferiche a bordo o dagli shield di espansione. In pratica, realizzando un progetto con uno shield Ethernet/WiFi, una memoria SD e che necessita dell'uscita seriale e di qualche ingresso analogico, restano a disposizione solo sei I/O digitali che sono spesso insufficienti per progetti di media complessità.

Per tutti questi motivi abbiamo pensato di progettare uno shield di espansione, compatibile con le varie schede Arduino e con la nostra Fishino UNO (e la Fishino Mega che vi presenteremo a breve) che, senza praticamente impegnare risorse hardware, permette di avere a disposizione ben 16 uscite in PWM e 16 ingressi/uscite digitali aggiuntivi. Non solo, le schede sono sovrapponibili fino ad un massimo di 8, consentendo di gestire con Arduino fino a 128 I/O digitali e 128 uscite PWM aggiuntive; il tutto reso completamente trasparente all'utilizzatore tramite una libreria realizzata ad-hoc.

SCHEMA ELETTRICO

Lo shield qui proposto è composto da tre blocchi:

- espansione I/O digitali;
- espansione PWM;
- circuito di protezione dall'inversione di polarità sull'alimentazione esterna.

ESPANSIONE I/O DIGITALI

Questa sezione si basa sul ben noto integrato MCP23017, nell'abituale versione controllata tramite interfaccia I²C-Bus, quindi tramite due soli fili, il clock (SCL) ed il filo di data (SDA).

Esiste anche una versione controllabile tramite interfaccia SPI, più veloce ma che impegna quattro linee della CPU e che quindi non abbiamo utilizzato.

Come tutti i dispositivi I²C, prevede un indirizzamento tramite una sequenza di bit inviati sulle due linee, che attivano l'integrato solo se l'indirizzo corrisponde.

Nel caso dell' MCP23017 l'indirizzo è composto da una parte fissa, indicata come MCP23017_I2C_BASE_ADDRESS nella nostra libreria, pari al numero esadecimale 0x20, i cui ultimi TRE bit sono però impostabili mediante tre pin esterni (A0, A1 ed A2) ai valori binari da 000 (0 decimale) a 111 (7 decimale), permettendo quindi di variare l'indirizzo completo tra 0x20 e 0x27 inclusi. Questo indirizzamento consente il collegamento allo stesso I²C-Bus di otto MCP23017, a patto che per ogni dispositivo venga scelto un indirizzo differente tra gli otto disponibili.

L'integrato è dotato di due porte I/O digitali ad 8 bit (GPA0÷GPA7 e GPB0÷GPB7), per un totale di 16 linee complessive, impostabili singolarmente come ingresso o uscita, con o senza resistenze di pullup interne, tramite la libreria software di cui parleremo in seguito.

Per ogni canale ad 8 bit il chip è in grado di generare un segnale di interrupt, sulle uscite **INTA** e **INTB**, al verificarsi di ogni cambio di stato degli ingressi, sia separatamente per ogni gruppo di 8 porte (permettendo quindi di distinguere gli interrupt a livello hardware tra i 2 canali) oppure congiuntamente, un solo segnale di interrupt per tutti gli ingressi.

La generazione degli interrupt al cambio di stato degli ingressi è un grande vantaggio quando è necessario ottenere risposte immediate al verificarsi di eventi esterni, e consente di compensare la limitazione delle schede come Arduino/Fishino UNO, dotate di due soli ingressi di interrupt.

Le uscite **INTA** ed **INTB** sono collegabili tramite due ponticelli distinti agli indirizzi, rispettivamente 2 e 3 di Arduino, gli unici in grado di rilevare un cambio di stato; tramite software sarà poi possibile scegliere se utilizzare entrambi, solo uno o nessuno. Completa la descrizione di questa sezione il condensatore di disaccoppiamento C3, situato in prossimità del chip per evitare che disturbi sulle linee

di alimentazione ne influenzino il funzionamento, e la linea RST che, connessa al reset di Arduino, consente di ripristinare il funzionamento all'accensione e/o alla pressione del tasto **RESET**.

SEZIONE PWM

Questa sezione si basa sul meno noto ma non per questo meno versatile PCA9685, un integrato in grado di generare 16 segnali PWM quasi indipendenti (vedremo nella descrizione delle librerie che la frequenza deve essere la stessa per tutte le uscite). Questo integrato consente un indirizzamento di ben sei bit, tramite gli ingressi A0..A5, che permetterebbe la sovrapposizione di 64 schede diverse; essendo però il limite imposto dall' MCP di 8 schede (e risultando 64 schede connesse ad un singolo bus decisamente troppe) abbiamo sfruttato i soli bits A0÷A2, ottenendo così anche qui la possibilità di connettere otto dispositivi diversi, collegando i rimanenti pins di indirizzamento direttamente a massa. La frequenza del PWM è impostabile tramite un registro interno (prescaler) con la seguente formula:

$$f_{PWM} = \frac{f_{clock}}{4096 \cdot (prescale + 1)}$$

L'integrato è in grado sia di generare la frequenza di clock internamente, tramite un oscillatore da 25 MHz che consente di ottenere frequenze PWM da 24 a 1.526 Hz, sia di operare con un clock esterno massimo di 50 MHz, che però non abbiamo utilizzato nella nostra scheda.

A differenza dell' MCP23017, questo integrato non dispone di un ingresso di reset; la relativa funzionalità va quindi realizzata tramite software (è previsto un indirizzo I²C speciale per il reset) oppure semplicemente ripristinando via software il contenuto dei registri (cosa da noi effettuata tramite la libreria). È invece disponibile un ingresso OE in grado di disabilitare tutte le uscite contemporaneamente, che abbiamo ritenuto opportuno portare all'esterno dando la possibilità, tramite un ponticello, di collegarlo all'I/O D9 di arduino nel caso se ne presenti la necessità.

Una possibilità interessante di questo integrato è di poter operare sulle uscite sia in modalità totem-pole, con una coppia di MOSFET complementari, in grado di fornire una corrente sia positiva (source) che negativa (sink), con valori massimi rispettivamente di 10 mA e 25 mA a 5 volt, che in modalità open-drain (o collettore aperto, in italiano), in grado di fornire solo una corrente negativa fino a 25 mA, modalità questa utile in alcuni casi.

Dalle caratteristiche (e da altre che qui tralasciamo per brevità) si evince che l'integrato è nato per pilotare in PWM dei LED, connessi direttamente alle uscite tramite resistenze; ciò non toglie che, come vedremo, è possibile utilizzarlo per innumerevoli altre applicazioni. Sullo schema potrete notare, infatti, una serie di resistenze in serie alle uscite (inserite per proteggere le medesime da sovracorrenti) che portano ad un connettore a tre pin per uscita sui quali, oltre al segnale PWM vengono riportate la massa ed una tensione positiva della quale parleremo più avanti. Questo connettore, realizzato ad angolo di 90°, permette di connettere direttamente alle uscite 16 servocontrolli senza impedire la sovrapposizione di ulteriori schede Octopus.

PROTEZIONE DALL'INVERSIONE DI POLARITÀ

Come abbiamo accennato nelle pagine precedenti, la scheda è stata pensata per poter pilotare direttamente dei servocomandi, collegandoli al connettore angolato. Questi componenti assorbono una notevole corrente di picco ed è impensabile poterli alimentare con i 5 volt di Arduino; è stato quindi previsto un connettore per fornire alla scheda una tensione esterna con corrente adeguata. Per evitare che un'inversione di polarità distrugga scheda e servocontrolli, abbiamo inserito in serie all'alimentazione il Mosfet Q1, a canale P, in grado di lasciar passare la corrente solo nella direzione corretta. Perché un mosfet e non un semplice diodo? Il motivo è che il diodo ha una caduta di tensione fissa ai suoi capi (leggermente variabile con la corrente) che, se risulta influente per tensioni medio-elevate e basse correnti, causa perdite e surriscaldamenti notevoli nel caso opposto di elevate correnti e basse tensioni.

Ad esempio, scegliendo un diodo Schottky con una caduta di tensione intorno a 0,5 V e facendo scorrere una corrente di 4 ampere, esso si trova a dover dissipare ben 2 W: e un diodo da 2 watt ha dimensioni non trascurabili e scalda molto.

Per contro, il MOSFET selezionato ha una resistenza di conduzione particolarmente bassa, pari a 51 mOhm (milliohm) alla stessa corrente di drain. La dissipazione diviene quindi, sempre nelle condizioni di cui sopra, pari a:

$$P = I^2 \cdot R = 4 \text{ Ampere} \cdot 0.051 \text{ ohm} = 0.816 \text{ Watt}$$

Una potenza quindi decisamente inferiore, con conseguente minor surriscaldamento e maggior durata di eventuali batterie. Il condensatore elettrolitico C1 filtra la tensione in ingresso da eventuali disturbi.

Il valore scelto, 100 μ F, è sufficiente fino a correnti di qualche ampere; nel caso di correnti più elevate e/o di assorbimenti molto impulsivi consigliamo di inserire un condensatore aggiuntivo di capacità elevata esternamente sulla linea di alimentazione.

REALIZZAZIONE PRATICA

Lo shield è realizzato su un circuito stampato a doppia faccia con fori metallizzati e fa uso di componenti in SMD; trattandosi di un montaggio superficiale, occorre la consueta attenzione, in particolar modo per i componenti più piccoli e quelli con i pin molto ravvicinati.

La maggiore attenzione è richiesta dall'integrato PCA9685, fornito in package TSSOP28 a 28 pin con spaziatura di soli 0,65 mm, e dalle reti resistive R3÷R6, che hanno pin piccoli e molto ravvicinati; attenzione anche al condensatore elettrolitico C1 che, pur non essendo di dimensioni particolarmente ridotte, ha i pin sotto il corpo che sporgono solo pochissimo dal medesimo.

Con un minimo di pratica, una lente d'ingrandimento e una pinzetta, un buon saldatore a punta fine (da non più di 20 W di potenza) ed un ottimo flussante (che aiuta a fondere e a distribuire lo stagno su piedini e piazzole) risulta comunque abbastanza semplice montare il circuito; nel caso coli troppa lega saldante e faccia cortocircuiti tra i pin degli integrati è possibile rimediare cospargendoli di flussante, pulendo molto bene la punta del saldatore e scaldandoli in modo che lo stagno in eccesso risalga per sulla punta liberandoli. Solitamente con 2-3 passaggi anche i cortocircuiti più ostici si risolvono in questo modo.

Occorre fare solo attenzione alla temperatura del chip, evitando di soffermarsi troppo tempo col saldatore e, nel caso occorra ripetere il procedimento, lasciando raffreddare per qualche decina di secondi il componente.

Chi non se la sentirà di realizzare lo shield potrà acquistarlo (presso la Futura Elettronica, www.futurashop.it) in versione montata e collaudata, con i connettori laterali e frontali non montati; ciò per offrire la possibilità di cambiare tipologia di connettore, per avere i contatti delle uscite (tipo servo, quindi con PWM, +5V e massa in linea) accessibili in verticale o frontalmente.

LA LIBRERIA

Come già accennato, per questa scheda abbiamo realizzato un'apposita libreria software, denominata **Octopus**, dotata di alcune particolarità che ne rendono semplicissimo l'utilizzo.

La prima particolarità interessante della libreria si può notare dalle linee dell'include file (**Octopus.h**):

```
#define Octopus __octopus()
OctopusClass &__octopus();
```

e dalle linee del file sorgente (**Octopus.cpp**):

```
OctopusClass &__octopus()
{
    static OctopusClass octo;
    return octo;
}
```

Questa modalità apparentemente strana di utilizzo della variabile **Octopus** permette di ovviare ad uno dei problemi del C++, ovvero dell'inizializzazione delle variabili globali che non avviene in un ordine predeterminato ma è casuale, e viene effettuata al caricamento dello sketch. Nel nostro caso, dovendo inizializzare l'interfaccia I²C tramite le istruzioni:

```
Wire.begin();
Wire.setClock(400000);
```

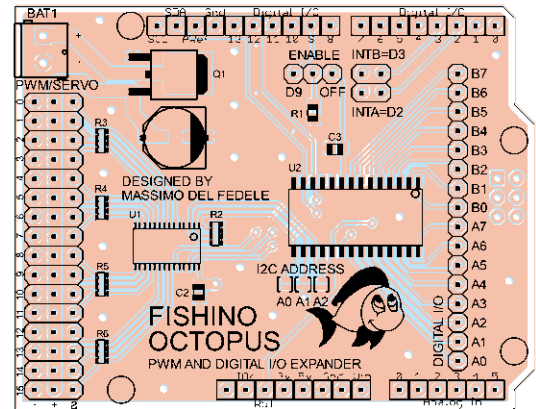
prima dell'utilizzo della libreria, risulta impossibile creare la variabile statica **Octopus** al momento del caricamento del programma, visto che l'interfaccia **Wire** in quel momento non è ancora stata inizializzata. La soluzione prescelta permette per contro di ottenere la creazione della variabile al primo utilizzo della medesima, e quindi dopo aver inizializzato correttamente l'interfaccia I²C-Bus; questo ci ha permesso di realizzare un codice che, senza nessuna riga di programma aggiuntiva, è in grado di contare ed inizializzare correttamente tutti gli shield Octopus connessi e di numerarne automaticamente le uscite in ordine di indirizzo I²C. Ad esempio, se alla nostra Arduino applichiamo due shield, avremo a disposizione 32 I/O digitali, numerati da 0 a 31, e 32 PWM, numerati anch'essi da 0 a 31.

La libreria fornisce due funzioni che permettono di conoscere il numero di schede connesse ed il numero di I/O e PWM disponibili:

```
// return number of boards found
uint8_t getNumBoards(void) const;

// return number of available I/O
uint8_t getNumIO(void) const;
```

Come detto in precedenza, la frequenza del PWM è unica per ogni scheda, quindi per ogni gruppo di 16 uscite PWM; è impostabile tramite le due funzioni seguenti, la prima scheda per scheda e



Elenco Componenti:

R1: 10 kohm (0805)
 R2: Array 4x10 kohm (0603)
 R3÷R6: Array 4x220 ohm (0603)
 C1: 100 µF 16 VL elettrolitico (Ø6mm)
 C2: 1 µF ceramico (0805)
 C3: 1 µF ceramico (0805)
 Q1: NTD25P03L (DPAK)
 U1: PCA9685PW
 U2: MCP23017-E/SO

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 3x16 vie 90°
- Strip Maschio 16 vie 90°
- Jumper (3 pz.)
- Strip Maschio 3 vie
- Strip Maschio 2 vie (2 pz.)
- Morsetto 2 poli passo 3mm
- Circuito stampato S1226

Listato 1

```
// Progetto : OctopusTest
#include <Wire.h>
#include <Octopus.h>

uint16_t i = 1;
uint16_t valueTable[16];

// Codice di inizializzazionecode
void setup(void)
{
    Serial.begin(115200);
    Serial.println("OCTOPUS TEST APP");

    Wire.begin();
    Wire.setClock(400000);

    delay(500);
    Serial.print("Found #");
    Serial.print((int)Octopus.getNumBoards());
    Serial.println(" OCTOPUS boards");
    delay(500);

    Octopus.setPWMFreq(1526);

    for(int k = 0; k < 16; k++)
        valueTable[k] = (- (double)k * k / 64 +
0.25 * k) * 4096;
}

// ciclo infinito
void loop(void)
{
    for(uint8_t p = 0; p < 16; p++)
        Octopus.analogWrite(p, valueTable[(i +
p) % 16]);
    i++;
    delay(50);
}
```

la seconda per tutte le schede connesse in un solo comando:

```
// set pwm frequency for a single connected board
// valid values 24 Hz...1526 Hz
void setPWMFreq(uint8_t board, uint16_t freq);

// set pwm frequency for ALL connected boards
void setPWMFreq(uint16_t freq);
```

Nella prima occorre indicare il numero di scheda (che va da 0 a *Octopus.getNumBoards()*) e la frequenza di PWM, da 24 Hz a 1.526 Hz; nella seconda è sufficiente indicare la frequenza e tutte le schede verranno impostate su quella. All'accensione, la frequenza preimpostata è di 200 Hz, adatta ai servocontrolli ma anche ai LED. Il valore delle uscite PWM è impostabile, analogamente alle librerie di Arduino, tramite la funzione seguente:

```
// pwm output
void analogWrite(uint8_t port, uint16_t val, bool invert = false);
```

Ad esempio, per impostare l'uscita 30 (la terzultima della seconda scheda connessa) al 50% del valore massimo, occorre scrivere

```
Octopus.analogWrite(30, 2048);
```

Il terzo parametro opzionale, **invert**, è utile nel caso si connettano dei led in uscita sfruttando le uscite in modalità open collector e collegandone gli anodi al positivo; in questo caso si ha bisogno di un'uscita inversa (più tempo resta alta, meno corrente scorre nel led) ed è quindi necessario impostare il parametro a true per ottenere una luminosità crescente con il valore **val**.

Come avrete sicuramente notato, il valore del PWM a differenza delle uscite di Arduino è a 16 bit, dei quali ne vengono utilizzati 12, permettendo quindi una variazione di intensità a 4.096 livelli al posto dei 256 di Arduino; questo consente, per esempio, un controllo molto più graduale dell'intensità della luce emessa da LED connessi alla scheda.

Per contro occorre ricordarsi di questo quando si imposta il valore, visto che un numero pari a 255, che su Arduino corrisponde all'intensità massima, qui corrisponde ad un valore piuttosto basso (255/4.096 del massimo).

Per la gestione degli I/O digitali, la libreria fornisce le seguenti funzioni, praticamente identiche a quelle delle librerie standard, se non per il fatto di poter utilizzare un numero anche molto grande di porta:

```
// digital I/O
void pinMode(uint8_t port, uint8_t mode);
bool digitalRead(uint8_t port);
void digitalWrite(uint8_t port, bool value);

// read/write all digital pins of given board at once
uint16_t digitalReadAll(uint8_t board);
void digitalWriteAll(uint8_t board, uint16_t val);
```

Le ultime due funzioni permettono di leggere e scrivere tutti i ports digitali di una scheda in un colpo solo, tramite una variabile a 16 bit; questo risulta molto comodo nel caso si abbia bisogno di modificare o leggere molto rapidamente le porte digitali. Al momento della pubblicazione di questo articolo la libreria (scaricabile dal nostro sito www.elettronica.it insieme ai file del progetto) è in fase di completamento, e le funzioni degli interrupt e della modifica di modalità delle uscite PWM (open drain/totem pole) devono essere completate.

UNO SKETCH DI PROVA

Per concludere, presentiamo un semplice sketch che permette di visualizzare una "coda" luminosa utiliz-

zando 16 LED connessi alle uscite PWM (**Listato 1**). Il codice inizializza l'interfaccia seriale, stampa un messaggio, inizializza l'IPC, stampa il numero di schede rilevate e, utilizzando la prima di esse (uscite PWM da 0 a 15) crea una sorta di 'serpente' luminoso sfruttando 16 led.

I valori di luminosità del 'serpente' sono preventivamente calcolati nella setup ed inseriti in una tabella contenente 16 valori; a seconda del punto di partenza della tabella (variabile 'i' nel loop) la 'testa del serpente' si trova in un punto differente, creando quindi l'effetto visivo voluto.

Come si nota, a parte il dover inserire 'Octopus.' davanti ai comandi **analogWrite()** l'utilizzo è praticamente identico alla libreria n. attiva di **Arduino**.

L'unica cosa degna di nota è il calcolo dei valori di luminosità, effettuato qui con un polinomio del secondo ordine in modo da creare un effetto di onda; sono possibili altri metodi con funzioni trigonometriche o semplicemente con una variazione lineare su cui potrete sperimentare; ad esempio:

```
for(int k = 0; k < 16; k++)
    sinTable[k] = 4096 * sin(M_PI / 16 * k);
```

per un andamento sinusoidale, oppure

```
for(int k = 0; k < 16; k++)
    if(k <= 8)
        sinTable[k] = 4096 * (double)k / 8;
    else
        sinTable[k] = 4096 * (double)(15 - k) / 8;
```

per un andamento bilineare.

Concludiamo qui la descrizione della nostra scheda **Octopus**; potete ora sperimentare i diversi effetti luminosi utilizzando ad esempio il driver Colibrì presentato nel numero scorso di Elettronica In. ■



per il MATERIALE

Lo shield Octopus (cod. OCTOPUS) è disponibile presso Futura Elettronica nella versione con la componentistica SMD premontata a 24,00 Euro (con i connettori laterali e frontali da montare). Software e sketch sono scaricabili gratuitamente sul sito www.elettronica.in. Il prezzo si intende IVA compresa.

Il materiale va richiesto a:

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

Moduli low cost per controlli remoti tramite rete GSM

Per controllare, attivare e verificare in modalità remota mediante cellulare o smartphone il funzionamento di qualsiasi apparecchiatura o sistema elettrico/elettronico. Modulo quadriband GSM/GPRS integrato. Tutti i dispositivi sono Made in Italy e certificati CE - R&TTE.

cod. TDG145
Combinatore telefonico GSM vocale con funzione TTS

cod. TDG138
Teleallarme GSM con anti-Jammer

cod. TDG139
Termostato con controllo GSM

cod. TDG140
Telecontrollo con comandi DTMF

cod. TDG133
Telecontrollo 2 Ingressi / 2 uscite

cod. TDG134
Apricancello per max. 200 utenti

Disponibile separatamente il modulo USB seriale (cod. FT782M) per il collegamento dei telecontrolli al PC.

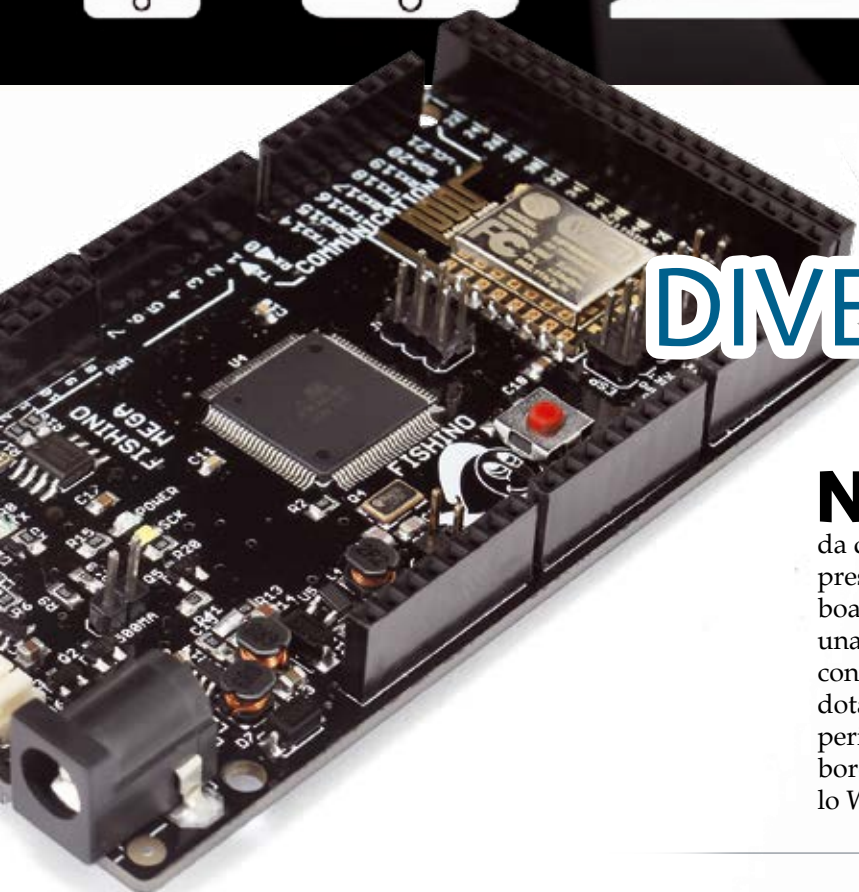
Documentazione tecnica e acquisti on-line su:
www.futurashop.it

FUTURA ELETTRONICA

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



La nostra versione della popolare e potente Arduino MEGA, basata su un hardware rivisitato specialmente nella sezione di alimentazione.

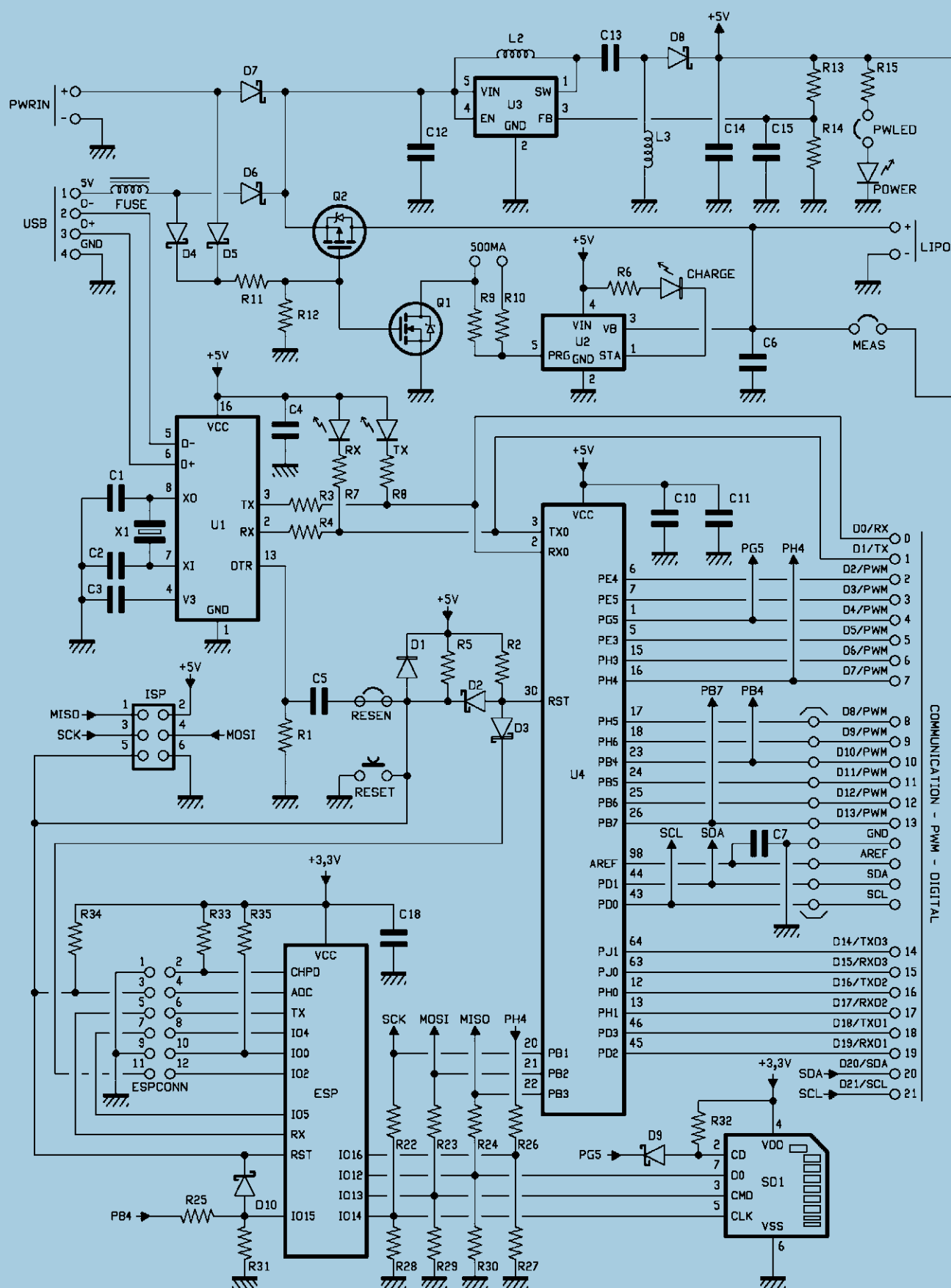


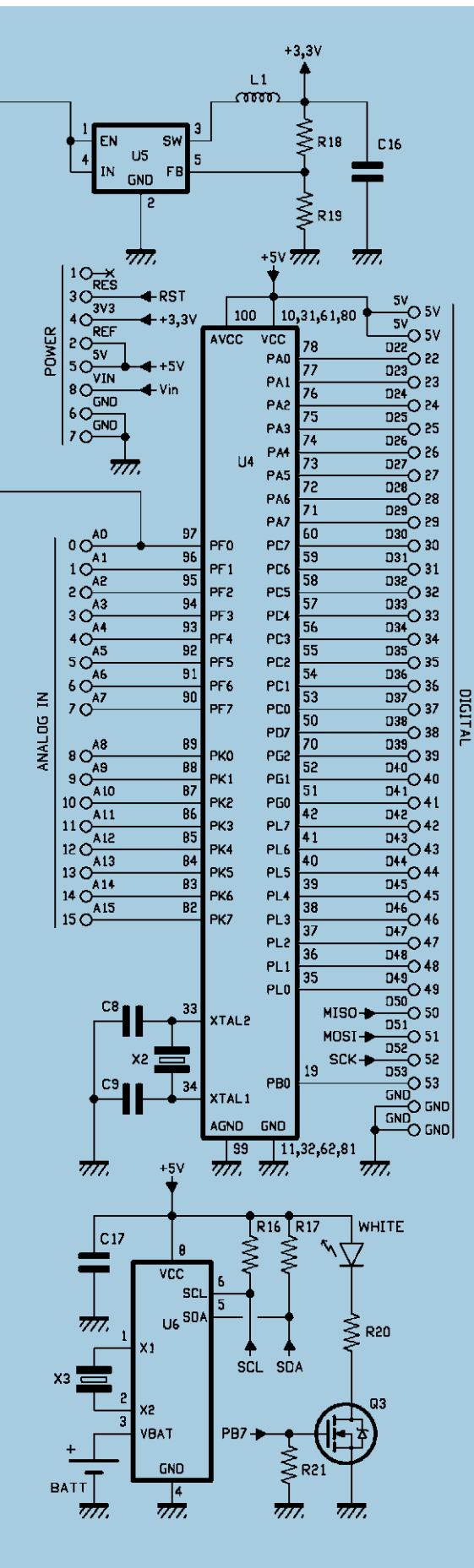
FISHINO DIVENTA MEGA

..... di MASSIMO DEL FEDELE

Non è passato neanche tanto tempo da quando abbiamo presentato la nostra board Fishino UNO, una scheda compatibile con Arduino UNO ma dotata di interessanti periferiche aggiuntive a bordo, quali un modulo WiFi, un lettore di

schede microSD ed un RTC (Real Time Clock, orologio in tempo reale). Come anticipato allora, la board non voleva essere un "pezzo unico" ma la prima di una serie completa di schede Arduino-compatibili con prestazioni e funzionalità estese, di cui si





sentiva la mancanza. In questo articolo presentiamo quindi la seconda scheda della serie: la Fishino MEGA. Si tratta di una board, compatibile con Arduino Mega, nel cui hardware abbiamo implementato una significativa novità: la possibilità di ottenere l'alimentazione a batteria.

Vediamo innanzitutto le caratteristiche complete della Fishino MEGA:

- compatibile al 100% con Arduino MEGA;
- modulo WiFi a bordo, con dotazione completa di librerie da noi sviluppate;
- lettore di microSD a bordo;
- modulo RTC a bordo;
- alimentazione switching integrale, in grado di fornire 800 mA-1A circa (complessivi tra i 3 ed i 5 volt), a partire da una tensione in ingresso che può variare da 3 e 20 volt;
- connettore per batteria LiPo a cella singola e caricabatteria a bordo;
- connettore laterale aggiuntivo per risolvere il noto bug di disallineamento della serie Arduino, che ne rende impossibile l'utilizzo con schede millefori e breadboard.

SCHEMA ELETTRICO

Il "pezzo forte" di questa nuova scheda è la sezione di alimentazione, che ha richiesto un lungo tempo di sviluppo ed ha anche portato alla realizzazione di un alimentatore multiuso separato la cui descrizione è iniziata in Marzo e si conclude in questo stesso fascicolo: il Torpedo. Per una descrizione dettagliata del convertitore SEPIC utilizzato rimandiamo quindi al relativo articolo, mentre qui ci limiteremo a farne una descrizione sommaria e ad analizzare la circuiteria di contorno in dettaglio. Iniziamo dagli ingressi di alimen-

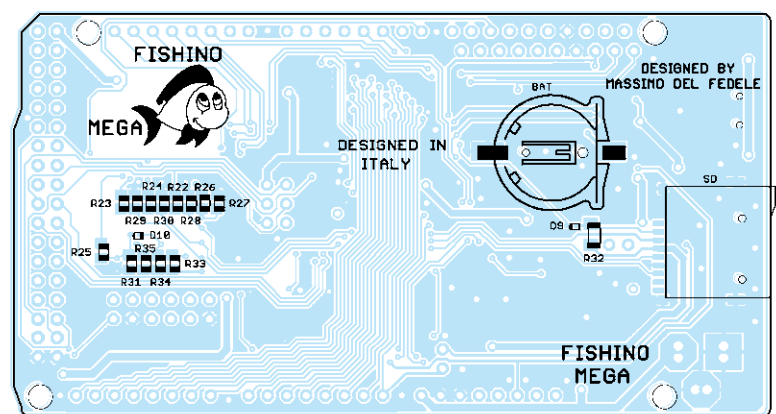
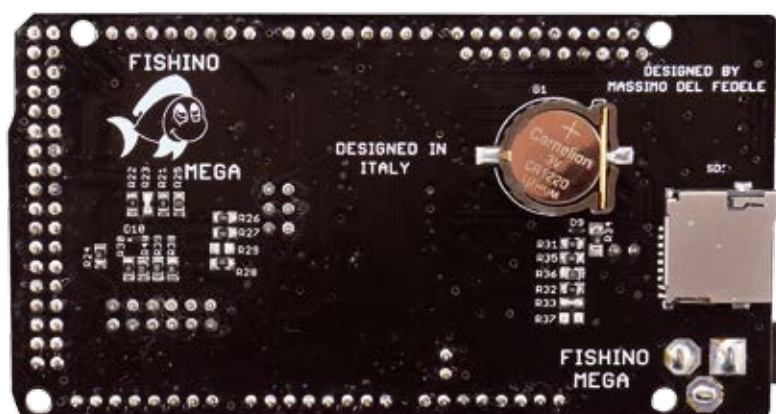
tazione, che portano tre tensioni:

- VLiPo, che fa capo al connettore per la batteria ricaricabile ai polimeri di litio (LiPo);
- VUSB, che fa capo al connettore microUSB;
- VIN, che fa capo al plug di alimentazione PWRIN.

Le due tensioni solitamente più elevate, ossia VUSB e VIN (in realtà al connettore PWRIN possiamo applicare una tensione minore, fino a 3,2 volt, sebbene sia sconsigliabile perché comporta un calo di efficienza), vengono fatte passare attraverso i due diodi Schottky di potenza D6 e D7, mentre la tensione "più bassa", per motivi di efficienza, viene instradata verso il MOSFET a canale P siglato Q2.

Tralasciamo per un istante il funzionamento di Q2, sul quale ritorneremo a breve, e consideriamo solo il diodo interno al medesimo; i tre diodi (D6, D7 ed il diodo interno al MOSFET) realizzano una porta OR di potenza, la quale permette di ottenere in uscita la tensione maggiore tra quelle presenti agli ingressi; supponiamo ad esempio di avere solo la tensione VLiPo, di circa 3,7V: in questo caso il diodo interno al MOSFET risulta polarizzato direttamente portando verso il convertitore la medesima tensione, che non può ritornare verso gli altri due ingressi a causa della presenza dei diodi D6 e D7 polarizzati inversamente.

Ora, se colleghiamo il connettore microUSB, la tensione VUSB di 5 volt polarizza direttamente il diodo D6 e raggiunge il convertitore mentre il diodo interno al MOSFET si trova con una tensione di 4 volt all'anodo e di 5 volt al catodo e risulta quindi polarizzato inversamente, interdicendosi. Se, infine, alimentiamo anche l'ingresso PWRIN con una



Elenco Componenti:

R1: 1 kohm (0805)	R33: 10 kohm (0805)
R2: 10 kohm (0805)	R34: 10 kohm (0805)
R3, R4: 1 kohm (0805)	R35: 10 kohm (0805)
R5: 10 kohm (0805)	C1: 22 pF ceramico (0805)
R6: 470 ohm (0805)	C2: 22 pF ceramico (0805)
R7: 2,4 kohm (0805)	C3: 1 µF ceramico (0805)
R8: 1 kohm (0805)	C4: 100 nF ceramico (0805)
R9: 10 kohm (0805)	C5: 1 µF ceramico (0805)
R10: 2,7 kohm (0805)	C6: 4,7 µF ceramico (0805)
R11: 2,2 kohm (0805)	C7: 100 nF ceramico (0805)
R12: 220 kohm (0805)	C8: 22 pF ceramico (0805)
R13: 97,6 kohm (0805)	C9: 22 pF ceramico (0805)
R14: 13,3 kohm (0805)	C10: 100 nF ceramico (0805)
R15: 470 ohm (0805)	C11: 100 nF ceramico (0805)
R16, R17: 10 kohm (0805)	C12: 22 µF 25 VL tantalio (0805)
R17: 10 kohm (0805)	C13: 4,7 µF 25 VL tantalio (0805)
R18: 475 kohm (0805)	C14: 22 µF ceramico
R19: 105 kohm (0805)	
R20: 1 kohm (0805)	
R21: 220 kohm (0805)	
R22 ÷ R25: 1 kohm (0805)	
R26: 3,3 kohm (0805)	
R27: 10 kohm (0805)	
R28 ÷ R31: 3,3 kohm (0805)	
R32: 10 kohm (1206)	

tensione superiore ai 5 volt, ad esempio di 15 volt, il diodo D7 risulterà polarizzato direttamente, portando la tensione VIN al convertitore, mentre i diodi D6 e quello interno al MOSFET verranno polarizzati inversamente interdicensi.

Abbiamo quindi ottenuto lo scopo prefissato, ovvero una commutazione totalmente automatica della sorgente di alimentazione. Torniamo ora al MOSFET inserito sulla linea positiva del connettore LiPo; apparentemente si tratta di un componente superfluo, essendo sufficiente il suo diodo interno per realizzare la commutazione; purtroppo i diodi, anche gli Schottky, hanno una caduta di tensione ai loro capi, che, per quanto piccola sia, causa perdite di potenza tali da risulta-

re rilevanti ad alte correnti. Facciamo un esempio pratico: supponiamo che l'utilizzatore (dopo il convertitore di cui parleremo in seguito) assorba 800 mA a 5 volt di tensione e che il convertitore abbia un'ottima efficienza (il 90% circa) cosa vicina a quella reale del nostro circuito. Ipotizziamo altresì che i diodi Schottky presentino una caduta di tensione di circa 0,5 volt. Alimentando il tutto attraverso il plug PWRIN con una tensione di 15 volt, otterremo un assorbimento di corrente pari a:

$$I_{IN} = 800 \text{ mA} \cdot \frac{5 \text{ V}}{90\% \cdot (15 \text{ V} - 0.5 \text{ V})} = 306.5 \text{ mA}$$

Da subito si nota il grosso vantaggio di utilizzare un alimentatore switching: la corrente assor-

bita diminuisce con l'aumentare della tensione in ingresso, a differenza dei regolatori lineari utilizzati sulla serie Arduino, nei quali la corrente assorbita resta identica a quella fornita e, moltiplicata per la caduta di tensione entrata-uscita, causa una rilevante dissipazione termica da parte del regolatore.

Nella formula, la caduta di tensione del diodo è stata considerata (0,5 volt), cosa che porta l'efficienza complessiva a:

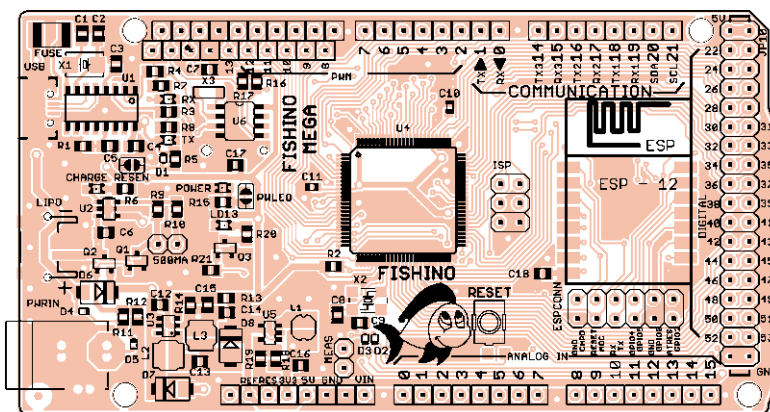
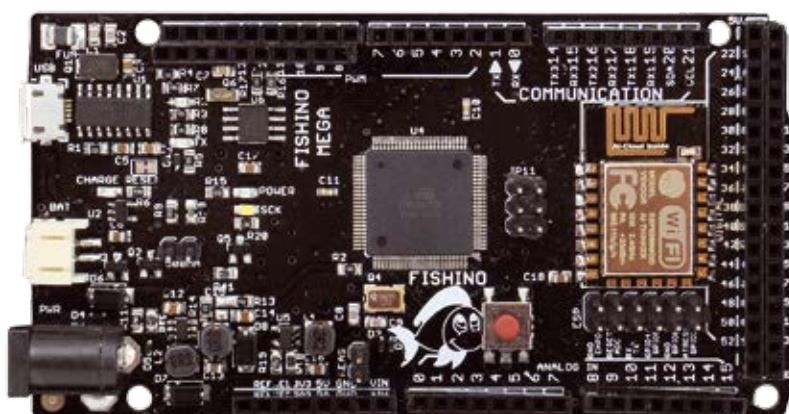
$$E = \frac{V_{OUT} \cdot I_{OUT}}{V_{IN} \cdot I_{IN}} = \frac{5 \text{ V} \cdot 800 \text{ mA}}{15 \text{ V} \cdot 306.5 \text{ mA}} = 87\%$$

contro il 90% teorico dato dal convertitore. Il diodo causa quindi una perdita di efficienza di un 3%, che possiamo considerare trascurabile. Dalla formula si nota

(0805)
 C15: 100 pF ceramico (0805)
 C16: 10 μ F ceramico (0805)
 C17: 100 nF ceramico (0805)
 C18: 1 μ F ceramico (0805)
 CHARGE: LED rosso (0805)
 WHITE: LED bianco (0805)
 RX: LED giallo (0805)
 TX: LED blu (0805)
 POWER: LED verde (0805)
 D1÷D5: RB521S-30TE61
 D6÷D8: SS34
 D9: RB521S-30TE61
 D10: RB521S-30TE61
 Q1: 2N7002
 Q2: NTR4171P
 Q3: 2N7002
 U1: CH340G
 U2: MCP73831T-2
 U3: SX1308
 U4: ATMEGA2560-16AU (MF1257)
 U5: LC3406
 U6: DS1307Z+

ESP: Modulo ESP8266
 X1: Quarzo 12 MHz
 X2: Quarzo 16 MHz
 X3: Quarzo 32.768 kHz
 L1: 2,2 μ H
 L2: 6,8 μ H
 L3: 6,8 μ H
 BAT: Porta Batteria CR1220 da CS
 FUSE: 500mA (1210)
 RESET: Microswitch
 SD: Connettore micro-SD
 USB: Connettore micro-USB

Varie:
 - Plug alimentazione
 - Connettore JST 2 vie passo 2 mm
 - Strip maschio 2x3 vie
 - Strip maschio 2 vie (2 pz.)
 - Strip femmina 8 vie (5 pz.)
 - Strip femmina 10 vie (2 pz.)
 - Strip femmina 2x18 vie
 - Batteria CR1220
 - Circuito stampato S1257



che l'incidenza della caduta sul diodo aumenta col diminuire della tensione in ingresso, il che è logico perché lo switching trasferisce la potenza convertendone tensione e corrente: crescendo la tensione cala la corrente e viceversa.

Facciamo ora lo stesso calcolo utilizzando una tensione in ingresso di 3,6 volt (un valore prossimo a quello della batteria LiPo):

$$I_{LiPo} = 800mA \cdot \frac{5V}{90\% \cdot (3.6V - 0.5V)} = 1433.7mA$$

Vediamo innanzitutto che per poter ottenere 800 mA in uscita dobbiamo prelevare dall'ingresso ben 1.433,7 mA; l'efficienza reale diventa quindi:

$$E = \frac{V_{OUT} \cdot I_{OUT}}{V_{LiPo} \cdot I_{LiPo}} = \frac{5V \cdot 800mA}{3.6V \cdot 1433.7mA} = 77.5\%$$

In questo caso la caduta di tensione sul diodo ha portato ad una perdita di efficienza di ben il 12,5%, proprio dove servirebbe invece un'efficienza maggiore, visto che alimentiamo la scheda a batteria.

Oltretutto, una caduta di 0,5 volt sul diodo con una corrente di 1,4 ampere corrisponde ad una potenza dissipata di 0,7 W, che su un componente di piccole dimensioni porta ad un notevole incremento di temperatura.

Il MOSFET, per contro, non ha una caduta fissa ai suoi capi ma presenta, in conduzione, una resistenza che dipende dal componente utilizzato; nel nostro caso vale al massimo 100 milliohm.

Il calcolo si complica un pochino:

$$I_{LiPo} \cdot (V_{LiPo} - I_{LiPo} \cdot R_{DS(ON)}) \cdot 90\% = V_O \cdot I_O$$

che è un'espressione di secondo grado in I_{LiPo} ; inserendo i valori e risolvendola, si ottiene:

$$I_{LiPo} = 1280mA$$

Una corrente ben inferiore ai 1.433,7 mA del caso precedente. In tali condizioni l'efficienza diventa:

$$E = \frac{V_{OUT} \cdot I_{OUT}}{V_{LiPo} \cdot I_{LiPo}} = \frac{5V \cdot 800mA}{3.6V \cdot 1280mA} = 86.8\%$$

Quindi il MOSFET causa una perdita di solo il 3,2% contro il 12% del diodo.

Stabilita l'utilità del MOSFET, resta da trovare il modo per portarlo in conduzione quando serve. Essendo un canale P, per ottenere ciò occorre polarizzarne

il gate con una tensione negativa rispetto al source; trattandosi di un MOSFET a bassa tensione di gate, per ottenere la resistenza di conduzione richiesta (R_{dsON} inferiore ai 100 mΩ) sono sufficienti circa 2,5 volt. A questo provvede la resistenza R12, che polarizza negativamente il gate. Per contro i diodi Schottky per piccoli segnali D4 e D5, uniti alla resistenza R11 (quest'ultima per evitare che sul gate arrivi una tensione superiore a quella ammissibile, fungendo da partitore insieme alla R12) si occupano di portare all'interdizione il MOSFET quando è presente una tensione agli ingressi USB e/o PWRIN.

La tensione prescelta raggiunge i piedini 4 e 5 dell'U3, che è la base del convertitore SEPIC. Il convertitore è stato ampiamente descritto nei relativi articoli, quindi ne daremo solo un breve cenno; il vantaggio di questo schema è principalmente la possibilità di ottenere in uscita una tensione sia inferiore che superiore a quella in ingresso, unendo così i vantaggi di un convertitore Buck e di un Boost, ma utilizzando un solo integrato switching.

La tensione in uscita è regolata dal partitore costituito da R13 ed R14, che forniscono una tensione di riferimento di 0,6 volt all'ingresso FB dell'integrato quando in uscita sono presenti 5 volt.

All'uscita del convertitore SEPIC troviamo quindi una tensione continua di 5 volt. Il LED verde POWER funge da spia di accensione ed è disattivabile tagliando il ponticello SMD PWLED, in modo da ridurre i consumi all'osso se si vuole alimentare il Fishino MEGA a batteria, cosa che ci era stata richiesta dagli utenti di Fishino UNO.

La tensione di 5 volt in uscita dal SEPIC entra in un ulteriore convertitore switching, questa

volta un semplice Buck (o step-down, abbassatore) converter che, per motivi di efficienza, è stato realizzato tramite un convertitore sincrono, ovvero dotato all'interno di un secondo MOSFET al posto del diodo Schottky utilizzato normalmente. Da questo convertitore escono i 3,3 volt necessari ad alimentare il modulo WiFi ed il lettore di microSD; come anticipato, la corrente disponibile si aggira sugli 800 mA - 1 ampere complessivi tra i 5 volt ed i 3 volt, cosa che permette di alimentare parecchi moduli esterni senza che l'alimentazione vada in crisi come succede spesso con Arduino ed i suoi regolatori lineari.

Una piccola parentesi: potrete notare la dimensione estremamente ridotta delle tre induttanze e dei condensatori di filtro dell'alimentazione; questo è stato reso possibile da un lato dall'elevata frequenza di lavoro degli integrati switching (sopra il MHz) e dall'altro dalla disponibilità di condensatori ceramici ad alta capacità.

Stadio carica batteria

Il cuore di questo stadio è l'integrato U2, un ben noto MCP73831, che contiene all'interno tutti i circuiti necessari alla carica ed al mantenimento della stessa di una batteria ai polimeri di litio a cella singola. L'integrato viene alimentato tramite una tensione a 5 volt applicata al piedino VIN, mentre la batteria da caricare viene connessa al piedino VBAT disaccoppiandola con un condensatore ceramico da 4,7 µF, che garantisce la stabilità dell'integrato. Il chip integra un sistema per rilevare la presenza della batteria, sistema a dire il vero piuttosto critico che a volte ne segnala la presenza anche quando non è connessa, in particolar modo a basse tensioni di alimentazione

della Fishino MEGA, ma ciò non ne inficia il funzionamento; l'unico inconveniente è l'illuminazione del LED di carica in quei casi. Il piedino PROG viene utilizzato per impostare la corrente di carica, che nel nostro circuito può essere scelta tra due valori: circa 100 mA con il ponticello siglato 500 MA aperto (viene inserita la sola resistenza R9 da 10 kohm) oppure circa 500 mA con il ponticello chiuso, il che corrisponde a inserire in parallelo alla R9 la R10 da 2,7 kohm. La formula per impostare la corrente di carica, nel caso si ritenesse necessario modificarla, è la seguente:

$$I_{reg} = \frac{1000}{R_{PROG}}$$

dove RPROG è la resistenza applicata al piedino PROG, in KOhm, ed IREG è la corrente di carica, in mA. Lo stesso piedino PROG se connesso al positivo dell'alimentazione o se lasciato fluttuante (disconnesso) serve a disabilitare la carica, disattivando i circuiti interni dell'integrato e riducendone il consumo praticamente a zero; questo viene utilizzato nel nostro schema inserendo il MOSFET Q1, un comune 2N7002 a canale N che, quando il gate risulta polarizzato negativamente, scollega di fatto le 2 resistenze di programmazione rendendo l'ingresso PROG fluttuante.

Il MOSFET è pilotato dallo stesso segnale utilizzato per pilotare l'interruttore della batteria Q2, anche se in modalità inversa: una tensione agli ingressi VUSB o VIN lo porta in conduzione attivando la carica, mentre in assenza (quando il circuito risulta alimentato dalla sola batteria) il MOSFET viene interdetto e la carica disattivata.

Per concludere la descrizione,

si noti il jumper siglato MEAS (measure, misura) che permette di collegare l'ingresso analogico ADC0 del controller alla batteria per eseguire il controllo dello stato della carica.

Interfaccia USB

L'interfaccia USB è la medesima utilizzata per il Fishino UNO, quindi ci limiteremo ad un piccolo accenno, rimandando al succitato articolo per una descrizione approfondita. Lo stadio gravita attorno all'ormai noto CH340G, un'alternativa estremamente valida al più noto FT232 o ad altre soluzioni con microcontrollori; il chip è stato scelto sia per motivi di economicità che di semplicità circuitale, a parità di prestazioni. L'integrato fornisce in uscita tutti i segnali di un'interfaccia RS232 standard, dei quali utilizziamo solo quelli di trasmissione/ricezione dati (Rx e Tx) ed il segnale DTR utilizzato per il reset automatico in fase di programmazione, come nell'Arduino originale, cosa che permette il caricamento degli sketch senza dover premere pulsanti o azionare interruttori. Torneremo in seguito sulla circuiteria di Reset perchè rispetto all'originale è stata modificata in modo da permettere in futuro la riprogrammazione dell'ATmega via WiFi.

ATmega 2560

In questa sezione lo schema di Fishino MEGA non si discosta da quello di Arduino MEGA; il controller è il medesimo, corredato dall'usuale quarzo a 16 MHz, dai 2 condensatori sul circuito oscillante e da un certo numero di condensatori di disaccoppiamento sulle linee di alimentazione, necessari per evitare che disturbi sulle linee di alimentazione possano influenzare il funzionamento del chip. Una

nota sui connettori di I/O : come si può notare dalle immagini è stato aggiunto un piccolo connettore a 10 pins affiancato a quello standard ma leggermente sfalsato in modo da poter utilizzare una scheda preforata standard per gli shields, risolvendo quindi l'annoso problema del passo errato dei connettori di Arduino senza peraltro inficiarne la compatibilità con gli shields esistenti.

Interfaccia SPI Adattatori di livello

Anche questa sezione risulta praticamente identica a quella già vista nel Fishino UNO; serve ad adattare i livelli delle logiche a 5 volt dell'ATMEGA con quelle a 3.3 volt del modulo WiFi e della scheda microSD. L'adattamento viene realizzato semplicemente tramite partitori resistivi (resistenze da R22 a R31) nella direzione 5V -> 3.3V, mentre nella direzione inversa viene sfruttato il fatto che le logiche a 5 volt accettano come segnali alti valori ben inferiori a 3 volt, risultando quindi compatibili con le logiche a 3.3 volt. Il segnale MISO (Master In Slave Out) in teoria non richiederebbe un adattamento, visto che la direzione va dalla logica a 3.3 volt verso quella a 5 volt; l'abbiamo inserito comunque in previsione della futura estensione del firmware del modulo WiFi che permetterà il caricamento degli sketch attraverso il medesimo, cosa che richiede uno scambio di ruoli, diventando in quel caso il modulo WiFi il master e l'ATmega lo slave.

Interfaccia scheda microSD

L'interfaccia è la medesima che abbiamo descritto nel Fishino UNO, e rispecchia lo shield SD (o analoghi combinati); funziona attraverso le linee SPI tra cui

Librerie disponibili

Attualmente sono state realizzate due librerie per la gestione del Fishino, di seguito descritte.

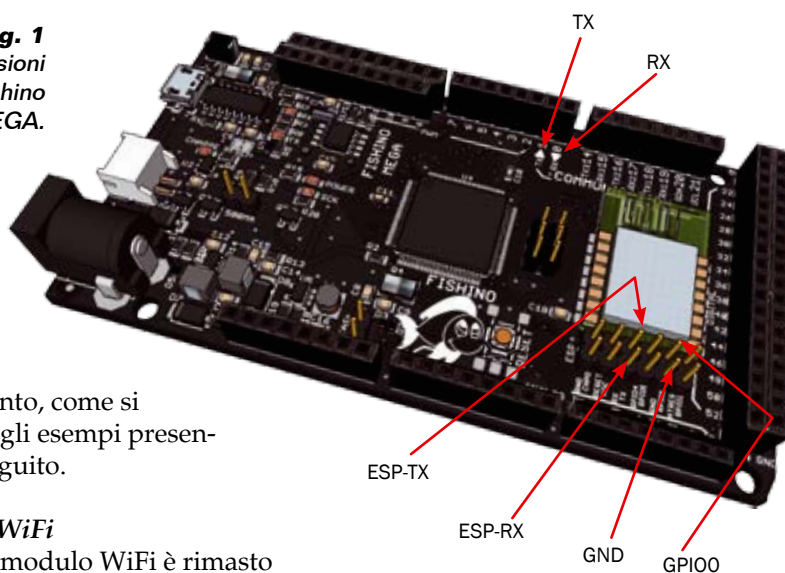
- **Libreria Fishino:** è l'esatto equivalente della libreria Ethernet o WiFi di Arduino. In questa libreria vengono definite le classi FishinoClass (gestione a basso livello, analoga alla EthernetClass o WiFiClass di Arduino), e le rispettive classi FishinoClient (l'analogo della EthernetClient e WiFiClient), FishinoSecureClient (non presente nelle librerie originali, permette connessioni a server sicuri SSL/HTTPS) e FishinoServer (EthernetServer e WiFiServer). Queste classi sono utilizzate in modo pressochè identico alle originali, quindi negli sketch esistenti che utilizzano l'Ethernet o il WiFi basta cambiare il tipo delle varie variabili per ottenerne il funzionamento con il WiFi di Fishino. Le uniche (leggere) differenze sono sull'inizializzazione, essendo il modulo WiFi di Fishino dotato di caratteristiche aggiuntive rispetto al WiFi originale.
- **Libreria FishinoWebServer:** è il porting su Fishino della nota TinyWebServer; consente la creazione di un completo server web sulla scheda. È stata inoltre inserita la libreria Flash nel pacchetto di download visto che è utilizzata dalla FishinoWebServer per spostare le costanti nella memoria Flash liberando così la poca RAM a disposizione. Questa libreria è comunque reperibile in rete, ma l'abbiamo allegata per comodità d'uso. Le rimanenti librerie necessarie sono già disponibili nei vari download dell'IDE di Arduino; sono in particolare necessarie la SD (per la gestione delle schede MicroSD) e la RTClib per la gestione del modulo RTC con il DS1307, ed altre librerie di sistema. Le librerie sono in continuo sviluppo e presto verranno corredate di funzionalità aggiuntive; è già stata recentemente implementata la gestione degli I/O digitali aggiuntivi presenti sul connettore ESPCONN, mentre è prevista a breve la gestione della seriale, dell'input analogico e delle uscite PWM sempre sul connettore ESPCONN.

Nel pacchetto di download sono presenti altre librerie, utilizzate per lo più nelle demo che sono state sviluppate e/o verranno sviluppate in seguito; consigliamo di installare tutto il contenuto nell'apposita cartella di Arduino.

MOSI (dati dall'ATmega verso la SD, Master Out Slave In), MISO (dati dalla SD all'ATmega, Master In Slave Out) e SCK (clock). I valori verso la scheda SD sono ovviamente ridotti dagli adattatori di livello di cui al paragrafo precedente.

La selezione della scheda avviene tramite la linea SDCS, attiva a livello basso. In questo caso l'adattamento di livello viene effettuato tramite una resistenza (R23) verso il positivo e un diodo (D9), che permette il solo passaggio delle sole correnti negative. Lo schema scelto consente di avere la scheda in standby quando il segnale SDCS (connesso al pin digitale 4 del Fishino MEGA) non è utilizzato; col pin in modalità threestate (ovvero ad alta impedenza) il diodo non conduce e sull'ingresso SDCS è presente un valore alto che disattiva la scheda. L'interfaccia è totalmente compatibile con gli shield di Arduino, quindi utilizza le stesse librerie esistenti per il suo fun-

Fig. 1
Connessioni
della Fishino
MEGA.



zionamento, come si vedrà negli esempi presentati in seguito.

Sezione WiFi

Anche il modulo WiFi è rimasto invariato rispetto a quello della scheda Fishino UNO; rimandiamo perciò all'articolo, pubblicato nel fascicolo n° 199, dove è stato descritto in maniera esauriente. Riportiamo quindi solo alcune note importanti, ovvero il tipo di comunicazione con l'atmega, realizzata tramite interfaccia SPI grazie ad un firmware da noi appositamente sviluppato, ed alcuni accorgimenti circuitali quali il diodo D10 utilizzato per forzare a livello basso il pin GPIO15 del

modulo al reset, senza il quale il modulo stesso si avvierebbe nella modalità "caricamento da SD" che lo renderebbe inutilizzabile. Questo risulta necessario poichè la linea GPIO15 ha anche funzione di Slave Select (SS) del modulo e non può quindi essere collegata direttamente a massa. Tutti i pin utili del modulo sono portati su un connettore (ESPCONN) come riportato qui di seguito.

- GPIO0: oltre ad essere utilizzabile come input/output digitale, serve per selezionare la modalità d'avvio al boot del modulo. Quest'ultimo può infatti essere avviato da Flash interna (funzionamento normale, GPIO0 a 1) o da interfaccia seriale, utilizzato per la riprogrammazione del firmware (GPIO0 a 0).
- GPIO2, GPIO4 e GPIO5 sono disponibili per l'uso come pin digitali, e sono sfruttabili tramite le apposite funzioni di libreria come fossero estensioni dei pin digitali di Arduino. Rx e Tx costituiscono la porta seriale hardware del modulo e sono utilizzati anche in fase di programmazione del firmware. Una prossima estensione del firmware ne permetterà l'uso come porta seriale aggiunti-



Silica apre la strada all'IoT

Il nome è già un programma: Visible Things, ossia "cose visibili" (ad evidenziare la possibilità di creare applicazioni concrete) è la piattaforma per lo sviluppo di sistemi e applicazioni IoT di Silica (www.avnetmemec-silica.com), che offre risorse hardware e software integrate per collegare sensori intelligenti e dispositivi embedded direttamente alle applicazioni cloud tramite funzioni di connettività a corto raggio verso unità gateway e connessioni WiFi, 3G e 4G. Supporta anche le reti IoT in tecnologia SIGFOX e LoRaWAN. Il produttore offre già tre starter kit basati su microcontrollori ARM Cortex.

va che consentirà a Fishino MEGA di avere un'ulteriore porta seriale.

- CH_PD è il pin di abilitazione del modulo. Portandolo a livello alto il modulo risulta abilitato (impostazione predefinita), mentre un livello basso mette in standby l'ESP riducendone i consumi praticamente a zero.
- RESET è il reset hardware dell'ESP, attivo a livello basso.
- ADC è l'ingresso analogico dell'ESP, diretto verso un convertitore A/D da 10 bit (1.024 valori possibili).

Circuiteria di RESET

La sezione di reset della Fishino MEGA è uguale a quella della Fishino UNO che già conoscerete; come accennato in precedenza, risulta più complicata di quella dell'Arduino MEGA per i seguenti motivi:

- occorre resettare sia l'Atmega che l'ESP alla pressione del tasto di reset, all'avvio e alla richiesta di programmazione da parte dell'IDE.
- per poter eseguire la programmazione dell'Atmega tramite WiFi, il modulo ESP dev'essere in grado di resettare l'Atmega stesso senza a sua volta autoresettersi.

Iniziamo dal segnale DTR che esce dall'interfaccia USB/Seriale (U1/CH340G); questo, come anticipato, viene posto a livello basso quando la porta seriale viene aperta. Attraverso il condensatore C5 (1 μ F ceramico, contro i 100 nF dell'originale per allungare l'impulso di reset) viene generato un breve impulso che, passato attraverso il jumper SMD RESEN (tagliando il quale è possibile disattivare l'autoreset), raggiunge la linea di "reset esterno", alla quale sono connessi

anche il pulsante di reset ed il pin 5 sul connettore di programmazione (ICSP).

A differenza del circuito originale, nelle schede Fishino è inserito un diodo (D2) tra la linea di RESET ed il pin dell'Atmega. Lo scopo di questo diodo (e del diodo D3 che vedremo in seguito) è di poter resettare solo l'Atmega, senza peraltro veicolare il segnale anche all'ESP.

In sintesi:

- premendo il pulsante RESET, o connettendo la seriale, l'impulso di reset raggiunge sia l'Atmega (attraverso D2) che l'ESP (direttamente), resettandoli entrambi;
- un segnale sulla linea ATRES-ESP, generato dall'ESP (nel caso si sia abilitata la programmazione attraverso il WiFi) raggiunge attraverso D3 la linea di reset dell'Atmega ma, a causa di D2, non può propagarsi all'ESP stesso.

Tramite questo sistema abbiamo quindi dato la possibilità al mo-

dulo WiFi di controllare la linea di reset dell'Atmega che, unitamente all'interfaccia SPI, ne permette la riprogrammazione senza nemmeno la necessità di un bootloader precaricato. In pratica, una volta completato lo sviluppo nel firmware, sarà possibile non solo riprogrammare via WiFi l'Atmega, ma farlo utilizzando anche lo spazio normalmente riservato al bootloader.

Modulo RTC

Concludiamo lo schema elettrico con il modulo RTC (Real Time Clock), costituito da un classico DS1307 della Maxim, un quarzo a 32 kHz, una batteria di backup e un paio di resistenze sulla linea I²C. Lo schema è quanto di più classico esista ed è completamente compatibile con le librerie di Arduino esistenti; tutte le funzioni sono gestite tramite linea I²C (SDA/SCL).

DRIVER USB

Fishino MEGA utilizza un convertitore USB/Seriale del tipo

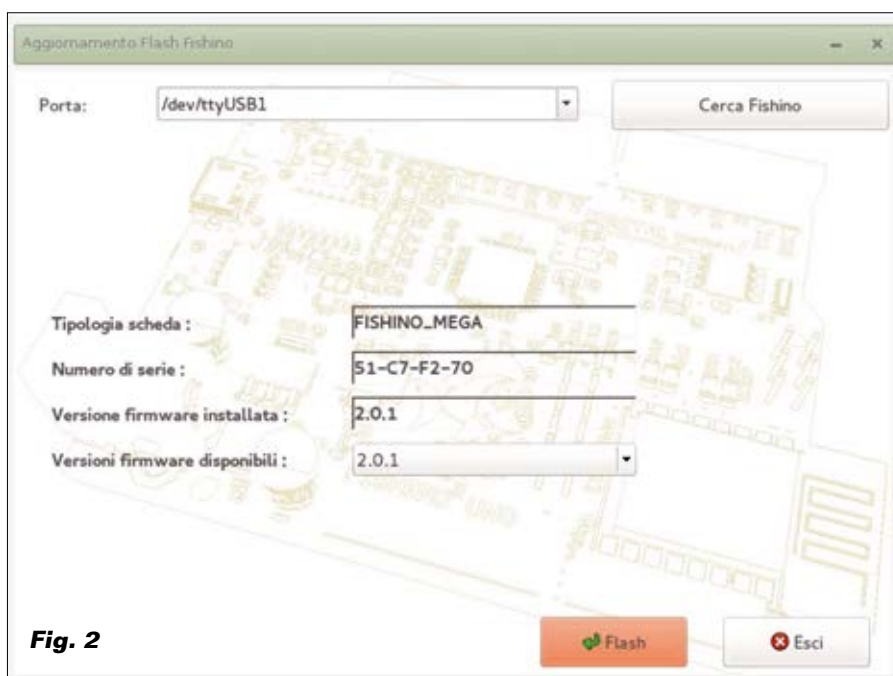
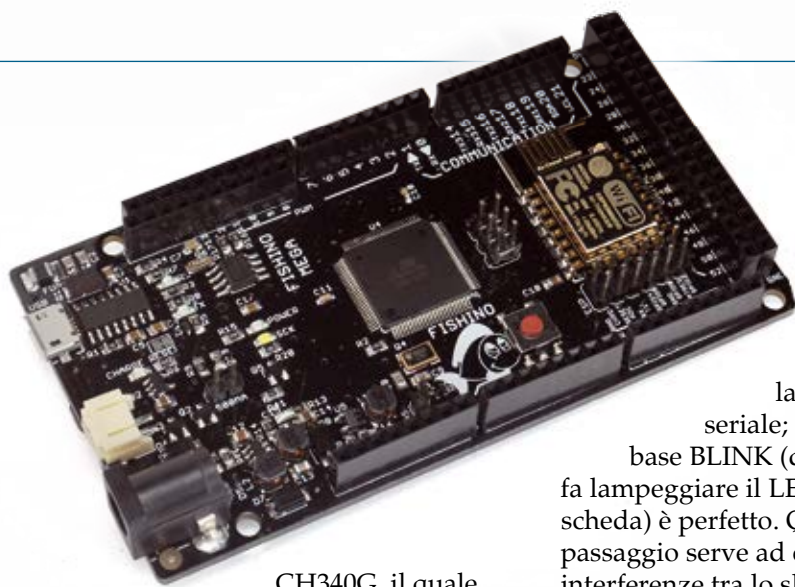


Fig. 2



CH340G, il quale necessita di driver appositi, almeno per Windows (fino alla versione 7 inclusa, pare che dalla 8 in poi i driver siano già presenti) e per Mac. Per l'ambiente Linux, per contro, i driver non sono necessari essendo questi già presenti nel kernel. I driver per Windows e per Mac si possono scaricare dal sito www.fishino.it (o www.fishino.com per la versione internazionale in inglese), sezione Download.

AGGIORNAMENTO FIRMWARE DEL MODULO WIFI

Fishino MEGA viene fornito con la versione del firmware disponibile al momento dell'assemblaggio. Essendo questo in fase di continuo sviluppo, conviene sicuramente eseguire un aggiornamento immediato ed è consigliato ripeterlo periodicamente. Le librerie di Arduino disponibili sono infatti aggiornate continuamente in base alle nuove possibilità offerte dal firmware. La procedura di aggiornamento è semplificata da un programma apposito, disponibile sia per la piattaforma Windows che Linux, che esegue l'operazione in modo completamente automatico ed a prova di errore. È prevista in un prossimo futuro anche una versione del flasher per Mac. I passi per l'aggiornamento sono i seguenti.

1) Caricare uno sketch che NON

utilizzi la porta seriale; l'esempio base BLINK (quello che fa lampeggiare il LED sulla scheda) è perfetto. Questo passaggio serve ad evitare interferenze tra lo sketch caricato ed il collegamento seriale tramite l'Atmega e l'ESP. Se il programma di flashing non rileva il Fishino, al 99% il problema è uno sketch sbagliato caricato.

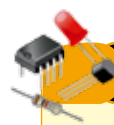
- 2) Connettere la porta TX di Fishino con la porta ESP-TX sul connettore ESPCONN, e la porta RX di Fishino con la porta ESP-RX sul connettore ESPCONN (Fig. 1).
- Connettere la porta GPIO0 a massa tramite un cavetto o un ponticello sempre sul connettore ESPCONN (Fig. 1).
- Collegare il Fishino al PC (o premere il pulsante di RESET se già connesso).
- Lanciare il programma FishinoFlasher, assicurandosi che il PC sia connesso ad Internet.

Se i collegamenti sono stati eseguiti correttamente, il programma rileverà la porta a cui è connesso Fishino, determinerà il modello e la versione del firmware attualmente installata, si collegherà ad un server remoto e scaricherà la lista dei firmware disponibili, mostrando l'ultimo e permettendo comunque la selezione delle versioni precedenti nel caso si voglia fare un downgrade (Fig. 2). Facendo clic sul pulsante "Flash" verrà avviata la procedura di aggiornamento, alla fine della quale apparirà un messaggio di con-

ferma. Per terminare il programma occorre premere il pulsante "Esci". Nel caso Fishino non venga rilevato automaticamente, è possibile provare a selezionare la porta manualmente. È comunque probabile che siano stati commessi degli errori nei collegamenti. La selezione manuale risulta indispensabile nel raro caso in cui più di un Fishino sia connesso contemporaneamente al PC, nel qual caso il primo viene rilevato automaticamente ma resta la possibilità di sceglierne un altro. Una volta terminata la procedura è sufficiente eliminare i tre collegamenti e Fishino sarà pronto per l'uso con il nuovo firmware.

CONCLUSIONI

Terminiamo qui l'esposizione della seconda scheda della serie Fishino, preannunciandovi l'imminente pubblicazione della prossima board, che sarà la Fishino Guppy; questa board, compatibile con la Arduino Nano, si distinguerà da essa perché dotata di WiFi, microSD e l'alimentazione switching con batteria che contraddistinguono il nostro "pesciolino" tecnologico. ■



per il MATERIALE

La board Fishino MEGA (cod. FISHINOMEGA) viene fornita montata e collaudata. Può essere acquistata presso Futura Elettronica al prezzo di Euro 49,90. Il prezzo si intende 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>

Rivisitazione della popolare Arduino Nano, rispetto alla quale implementa in più l'architettura della nostra Fishino con tanto di connettività WiFi, caricabatteria ecc.

GUPPY: FISHINO SI FA NANO

di MASSIMO DEL FEDELE

Come vi abbiamo anticipato nell'articolo in cui abbiamo presentato la nostra Fishino MEGA (scheda compatibile con Arduino MEGA, contraddistinta da un gran numero di I/O e molta memoria di programma e/o RAM rispetto alla versione UNO) abbiamo sviluppato una board che può essere considerata l'altra faccia della medaglia, in quanto è la versione più essenziale della "saga" Fishino. Si tratta della Fishino Guppy, il cui nome deriva da un pesciolino ben conosciuto agli appassionati di acquari, che abbiamo voluto assegnarle in virtù delle ridottissime dimensioni della scheda, che è la più piccola Fishino. Malgrado sia la versione più miniaturizzata (come la Nano è per Arduino) la nostra piccola Guppy presenta funzionalità e caratteristiche di tutto rispetto (come potete leggere riquadro a pagina seguente). Le periferiche incluse nella board sono quelle delle Fishino UNO e MEGA, eccetto che per il modulo RTC, che fisicamente non

trova posto nel PCB che è stato voluto, in sintonia con la filosofia della Arduino Nano, il più piccolo possibile. Rispetto alla Fishino UNO, però, abbiamo aggiunto tutto lo stadio d'alimentazione in tecnologia switching e la possibilità di alimentare la board con una batteria ai polimeri di litio, cosa estremamente interessante perché si presta a realizzare progetti portatili, se non, addirittura, indossabili.

SCHEMA ELETTRICO

Passiamo ora alla descrizione dello schema elettrico, che si può considerare diviso nei seguenti blocchi:

- stadio di alimentazione;
- stadio caricabatteria;
- interfaccia USB;
- controller Atmega328p;
- adattatori di livello;
- modulo WiFi;
- connettore per microSD-Card.

CARATTERISTICHE TECNICHE

- 100% compatibile con Arduino Nano
- Scheda WiFi a bordo
- Connettore per microSD a bordo
- Tripla alimentazione: batteria LiPo, porta USB e connettore di alimentazione esterno (6,5÷20 volt)
- Stadio di alimentazione switching ad elevata efficienza, con step-up per ottenere i 5 volt dalla batteria e step-down per ricavarli dall'ingresso esterno,
- Sezione 3,3 volt potenziata per gestire il modulo WiFi, la microSD ed eventuali ulteriori componenti esterni
- Caricabatteria a bordo con indicazione dello stato di carica

SCHEMA ELETTRICO

Siccome, eccezion fatta per i primi due blocchi (lo stadio di alimentazione e il caricabatteria), lo schema elettrico della Guppy è molto simile, quasi identico a quello già presentato per la board Fishino UNO, ci limiteremo a descrivere in dettaglio solo i primi due stadi e daremo una spiegazione sommaria dei restanti, evidenziando le differenze rispetto alle precedenti schede. Per gli approfondimenti del caso rimandiamo all'articolo dove abbiamo descritto Fishino UNO (fascicolo n° 198 di Elettronica In). Lo schema elettrico cui faremo riferimento in questa descrizione è illustrato nella pagina qui accanto.

L'alimentazione

A differenza di quanto fatto nella Fishino Mega, nella Guppy non abbiamo utilizzato uno stadio Sepic, il quale, pur permettendoci di risparmiare un componente e di poter alimentare il circuito sul pin VIN con una tensione minima di 3,5 volt, si è rivelato troppo instabile a causa delle dimensioni ridotte della scheda.

Nel primissimo prototipo avevamo infatti utilizzato tale tipologia, unita ad un PCB a due layer; il tutto funzionava perfettamente finché non veniva utilizzato il modulo WiFi, dopodiché i disturbi provocati da questo "rientravano" nel feedback dell'alimentazione provocando instabilità. Siamo quindi ritornati sui nostri

passi optando per un doppio converter: uno step-up per l'ingresso batteria ed uno step-down per l'alimentazione dall'ingresso VIN. Per evitare l'instabilità possibile comunque anche con queste tipologie, abbiamo poi optato per un PCB a quattro layer, con i piani di massa ed alimentazione all'interno, in modo da creare una barriera che impedisce ai forti segnali impulsivi di trasmettere le proprie interferenze attraverso il circuito stampato stesso.

Con questi accorgimenti la nostra Guppy è quindi diventata estremamente stabile, anche se il prezzo da pagare è che non è possibile autocostruire la scheda con metodi artigianali; pubblichiamo comunque schema elettrico e piano di montaggio per consentirvi di conoscere a fondo l'hardware della board.

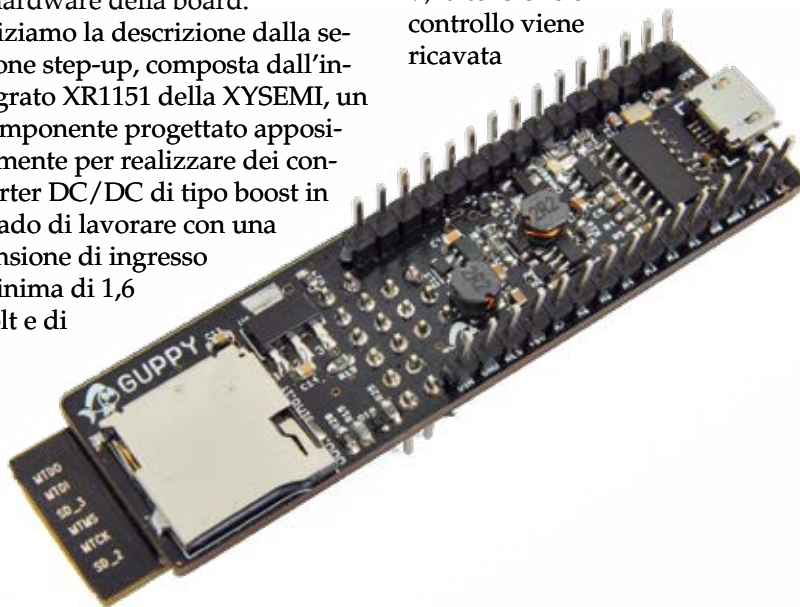
Iniziamo la descrizione dalla sezione step-up, composta dall'integrato XR1151 della XYSEMI, un componente progettato appositamente per realizzare dei converter DC/DC di tipo boost in grado di lavorare con una tensione di ingresso minima di 1,6 volt e di

fornire in uscita 5 volt con 800 mA di corrente, se alimentato con 3,3 volt in ingresso. È quindi ottimale per accettare l'alimentazione da una comune batteria LiPo 1s (3,7 volt, quindi a singola cella).

L'integrato contiene al suo interno tutti gli stadi necessari: un generatore PWM, un interruttore CMOS con una resistenza $R_{ds(on)}$ estremamente bassa (tipicamente 100 milliohm), la necessaria circuiteria di retroazione (feedback) per mantenere stabile la tensione in uscita ed un sistema di avvio progressivo (soft-start) per evitare picchi di corrente eccessivi nei componenti demandati alla commutazione della corrente d'uscita.

L'integrato lavora ad una frequenza tipica di 1,2 MHz e quindi consente l'utilizzo di componenti (induttanza e condensatori) di dimensioni estremamente contenuti. Opera in modo asincrono, quindi richiede un diodo esterno di commutazione (D12) che nel nostro caso è uno Schottky con bassissima tensione diretta per non penalizzare l'efficienza.

Il circuito di retroazione è calibrato per una tensione di 0,6 V; la tensione di controllo viene ricavata





dal partitore costituito dalle resistenze di precisione R29 ed R30 ed applicata nell'ingresso FB. La differenza di potenziale in uscita viene stabilizzata sul valore dato dalla formula:

$$V_{out} = 0.6V \cdot \left(1 + \frac{R_{29}}{R_{30}}\right) = 0.6V \cdot \left(1 + \frac{97.6K}{13.3K}\right) = 5.003V$$

La tensione viene riportata direttamente alla linea dei 5V senza alcun diodo aggiuntivo, essendo già presente il diodo D8 e l'interruttore interno all'integrato, disattivabile tramite l'ingresso EN dello stesso, sul quale ritorneremo più avanti.

Una nota: l'XR1151 è fatto per lavorare con tensioni molto basse, e quindi si danneggia se all'ingresso vengono applicati più di 6 volt; questo lo differenzia dal componente utilizzato nello stadio SEPIC della Fishino Mega, che in ingresso accettava tensioni ben più elevate. È quindi assolutamente da evitare l'applicazione di tensioni superiori ai 5÷5,5 V all'ingresso batteria, pena la distruzione del componente e di ciò che viene alimentato da esso. Proseguiamo con la sezione step-down, che ruota attorno all'integrato AOZ1280 dell'Alpha&Omega Semiconductor; questo componente accetta in ingresso tensioni da 3 volt fino a 26 volt ed è in grado di fornire un'uscita minima di 0,8 volt. La frequenza di lavoro è elevatissima, 1,5 MHz, quindi anche qui possiamo utilizzare componenti di dimensioni estremamente ridotte. Come nel caso visto per lo step-up, l'AOZ1280 contiene tutti i componenti necessari per realizzare un buck converter asincrono, ma richiede un diodo esterno di blocco, siglato D11 nello schema elettrico.

L'integrato dispone di un ingresso di compensazione che, tramite il condensatore C12 da 10 nF, permette di stabilizzare la rete di

retroazione in tutte le condizioni di carico.

L'ingresso EN (enable) in questo caso non viene utilizzato, ragion per cui lo abbiamo collegato direttamente alla tensione in ingresso, il che abilita il convertitore non appena viene alimentato.

L'ingresso di retroazione (FB, ossia feedback) è in questo caso calcolato per una tensione di riferimento interna pari a 0,8 volt; la tensione in uscita risulta quindi pari a:

$$V_{out} = 0.8V \cdot \left(1 + \frac{R_{31}}{R_{32}}\right) = 0.8V \cdot \left(1 + \frac{68.1K}{12.1K}\right) = 5.3V$$

La Vout è stata volutamente scelta leggermente superiore ai 5 volt per compensare la caduta sul diodo Schottky in uscita D10, che si aggira sugli 0,2÷0,3 volt; tale diodo è indispensabile per evitare che la tensione proveniente dallo step-up rientri nel componente e vada ad influire sull'ingresso VIN.

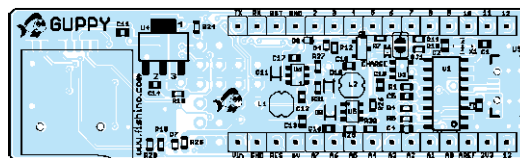
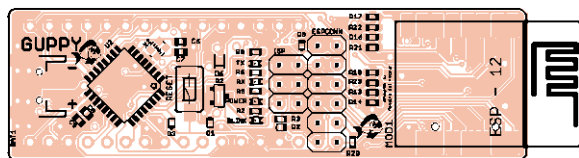
Detto ciò, andiamo ad analizzare la sezione di commutazione degli ingressi di alimentazione, differente rispetto allo schema delle altre Fishino; essa ruota attorno ai due diodi Schottky D4 e D5, al MOSFET a canale N Q1 e alle resistenze R12 e R26. In assenza di tensione sugli ingressi VIN o USBVCC, il gate di Q1 viene mantenuto a massa tramite la resistenza R12, spegnendo quindi il componente. La sua uscita si trova quindi libera di fluttuare ed è mantenuta a livello prossimo all'alimentazione dall'integrato caricabatteria (che vedremo in seguito) U3; l'ingresso EN (enable) dell'XR1151 verrà quindi a trovarsi a livello alto (tramite la resistenza R26) e quindi entrerà in funzione. L'uscita a 5 volt non potrà rientrare sui pin VIN o USBVCC a causa della presenza dei diodi D6 e D10.

Applicando, per contro, una tensione su uno degli ingressi VIN o USBVCC, il MOSFET Q1 verrà polarizzato positivamente attraverso il diodo D4 o D5 e quindi entrerà in conduzione, portando a massa l'ingresso EN dell'XR1151 spegnendolo.

Se la tensione all'ingresso VIN è minore di circa 6 volt, all'uscita del convertitore Step-Down sarà presente una tensione inferiore ai 5 volt, cosa che farà entrare in conduzione il diodo D6 attivando quindi l'alimentazione via USB; per contro, se sui VIN applichiamo una tensione sufficiente, l'uscita del buck converter AOZ1280 sarà sufficiente a portare il diodo D6 in interdizione e quindi disattivare anche l'alimentazione dalla porta USB. Abbiamo quindi ottenuto una commutazione completamente automatica delle tensioni di alimentazione, in questo modo: se è presente una tensione su VIN o sul plug USB, la batteria verrà sconnessa dal circuito e ne verrà contemporaneamente attivata la carica, come vedremo in seguito.

Se non è presente alcuna tensione esterna, entrerà in azione la batteria e, contemporaneamente, verrà disattivato il circuito di carica.

L'ultima parte che analizziamo, ma non ultima per importanza, è costituita dal regolatore a 3,3 volt che fornisce la tensione ai moduli WiFi e alla scheda microSD. Questa parte circuitale, per esigenze di spazio è stata realizzata tramite un consueto regolatore lineare, ma di capacità superiore a quello della Arduino Nano; è infatti in grado di erogare in uscita continuamente oltre 800 mA di corrente, contro i 500 mA di picco del regolatore di tensione originale della Nano.



R1, R2, R4, R5, R8, R18: 1 kohm (0603)
 R3, R10, R12, R14, R19, R24÷R26, R28: 10 kohm (0503)
 R6: 2,4 kohm (0603)
 R7, R9: 470 ohm (0603)
 R11: 2,7 kohm (0603)
 R13, R23: 3,3 kohm (0603)
 R15÷R17: 680 ohm (0603)
 R20÷R22: 1,5 kohm (0603)
 R27: 12,1 kohm (0603)
 R29: 97,6 kohm (0603)
 R30: 13,3 kohm (0603)
 R31: 68,1 kohm (0603)
 C1, C2, C6, C7: 22 pF ceramico (0603)
 C3, C5: 1 µF ceramico (0603)

C4, C8, C9, C11: 100 nF ceramico (0603)
 C10: 4,7 µF ceramico (0603)
 C12: 10 nF ceramico (0603)
 C13÷C17: 22 µF ceramico (0603)
 D1÷D5, D7, D9: RB521S SOD523
 D6, D8, D10, D11: PMEG3020EJ SOD323
 X1, X2: Quarzo 16 Mhz
 Q1, Q2: 2N7002 SOT23
 L1, L2: Bobina 2,2 µH CD32
 TX: LED Blu (0603)
 RX: LED Giallo (0603)
 BLINK: LED Bianco (0603)
 POWER: LED Verde (0603)
 CHARGE: LED Rosso (0603)
 U1: CH340G SOIC16

U2: ATMEGA328P-AU TQFP32-08
 U3: MCP73831 SOT23-5
 U4: NCP1117ST33 SOT223
 U5: XR1151 SOT23-6L
 U6: AOZ1280 SOT23-6L
 RESET: Microswitch
 SD1: Connettore micro-SD
 ESP: Modulo Wi-Fi ESP8266
 USB1: Connettore micro-USB
 BAT1: Connettore JST 2mm 2 vie

Varie:
 -Strip maschio 2x3 vie
 -Strip maschio 6 vie (2 pz.)
 -Strip maschio 15 vie (2 pz.)
 -Circuito stampato S1270

Stadio caricabatteria

Questo stadio ruota attorno al notissimo MCP73831 (U3) che abbiamo già trattato in vari progetti e che quindi non stiamo a rispiegare. L'unica parte degna di nota è il circuito di attivazione della carica, che fa capo ad alcuni componenti già visti nel paragrafo riguardante la commutazione dell'alimentazione: Q1, D4, D5 ed R12.

Abbiamo infatti già visto che quando è presente una tensione di alimentazione esterna su VIN o USBVCC, il MOSFET Q1 viene portato in conduzione e quindi il suo drain viene a trovarsi collegato a massa. Questo, oltre che spegnere lo step-up XR1151 come visto in precedenza, collega a massa la resistenza R10 (con eventualmente in parallelo la R11 se scelta la carica a 500 mA),

attivando così la ricarica.

Il diodo led rosso CHARGE si accende durante la carica e si spegne quando essa è completa, in assenza di batteria o quando il caricabatteria è disattivato.

INTERFACCIA USB

Lo stadio d'interfaccia USB della Guppy è identico a quello utilizzato a bordo delle schede Fishino UNO e Fishino Mega e gravita attorno all'ormai noto CH340G, un'integrato in grado di realizzare un completo convertitore USB/Seriale con tutti i relativi segnali di controllo, dei quali noi utilizziamo solo i canali dati (RX e TX) ed il segnale DTR per l'autoreset della Fishino Guppy quando viene programmata dal PC mediante l'IDE Arduino.

Per problemi di spazio, qui non

abbiamo previsto la possibilità di disattivare l'autoreset, funzione, peraltro, quasi mai utilizzata. Per una descrizione più approfondita di questo modulo rimandiamo all'articolo relativo dove abbiamo pubblicato il progetto della Fishino UNO dove è stato descritto in dettaglio (fascicolo n° 198).

MICROCONTROLLORE

Questo stadio risulta praticamente identico a quello della scheda Fishino UNO e utilizza lo stesso microcontrollore, ossia l'ATmega328P; differisce per l'utilizzo di due ingressi analogici aggiuntivi, vale a dire A5 ed A6, che non sono presenti nel layout della "UNO". Per il resto non sono state introdotte modifiche, quindi anche in questo caso qui rimaniamo alla descrizione dettagliata

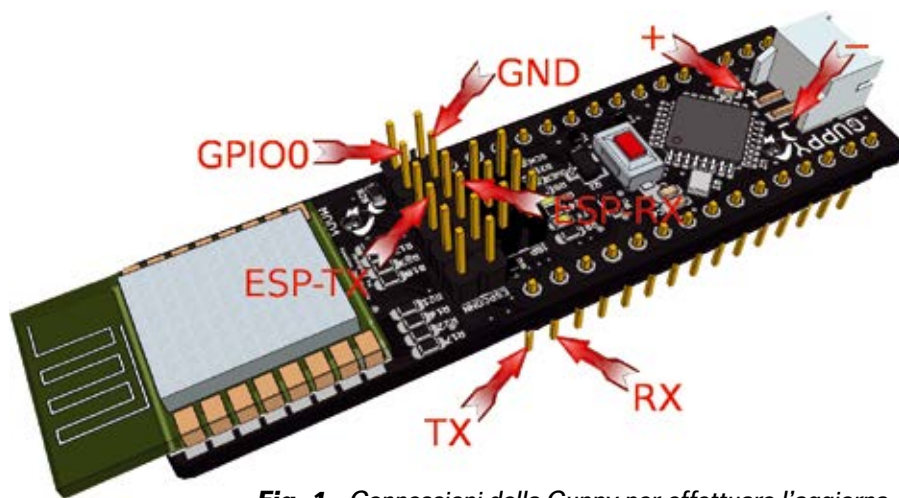


Fig. 1 - Connessioni della Guppy per effettuare l'aggiornamento del firmware.

reperibile nell'articolo descrittivo della Fishino UNO.

Una sola nota riguardante il bootloader: pur essendo questa scheda compatibile con l'Arduino Nano, abbiamo preferito installare il bootloader di Arduino UNO, per il semplice motivo che quest'ultimo occupa 1,5 kByte di Flash in meno e consente quindi di caricare sketch di dimensioni maggiori rispetto a quanto possibile con il bootloader della Nano. Pertanto è necessario, nell'IDE, selezionare come scheda Arduino UNO e non Arduino Nano, altrimenti l'IDE non riconoscerà il prodotto.

Nell'improbabile eventualità che si necessiti la piena compatibilità anche dal lato bootloader è comunque possibile caricare sul Guppy il bootloader originale di Arduino Nano.

Prestate attenzione alla piedinatura del connettore ISP, che nella Guppy è "rovesciata" rispetto a quella della UNO; ciò per seguire la piedinatura originale della Arduino Nano. Vi diciamo questo perché a chi scrive è capitato di "bruciare" una scheda Nano per aver collegato il connettore a rovescio, quindi, in caso di dubbio, vi consigliamo di consultare lo schema elettrico ed il layout del PCB prima di eseguire la programmazione del bootloader!

STADIO ADATTATORE DI LIVELLO

Anche in questo stadio non vi sono modifiche rispetto alle schede Fishino precedenti, fatto salvo il valore delle resistenze (da R13 a R23) che è dimezzato rispetto a quello delle corrispondenti resistenze di Fishino UNO e Fishino MEGA, per garantire una maggior immunità ai disturbi; tale requisito è stato imposto dalle dimensioni estremamente ridotte della scheda. Ad un'impedenza minore, infatti, corrisponde una maggior immunità ai disturbi esterni al costo di un piccolo aumento dei consumi durante il funzionamento del WiFi o della scheda microSD, da considerarsi comunque del tutto trascurabile. Per la descrizione dettagliata del funzionamento vi rimandiamo anche qui all'articolo relativo al Fishino UNO; qui indichiamo soltanto che gli stadi sono costituiti da partitori resistivi

che abbassano il valore massimo della tensione in uscita dall'Atmega dai 5 volt a circa 3,3 Volt, per non danneggiare i componenti a tensione più bassa; in senso inverso non occorre alcuna traslazione, essendo i 3,3 volt in grado di pilotare direttamente le logiche a 5 volt, dato che i pin sono 5V-tolerant.

MODULO WIFI

Come nella Fishino UNO e nella Fishino MEGA, la sezione di interfaccia wireless ruota attorno al modulo ESP12-F (l'ultima uscita della serie 12) basata sul chip WiFi ESP8266; la particolarità del modulo qui utilizzato è che ha un firmware studiato ad-hoc per comunicare con l'Atmega328 tramite interfaccia SPI, ben più performante della seriale che il firmware di fabbrica prevede di utilizzare per l'interfacciamento. A fine articolo riporteremo il procedimento di sostituzione di tale firmware con quello che occorre per il colloquio su SPI (la procedura è comunque presente anche sul sito web di riferimento del progetto, www.fishino.it). Maggiori dettagli sul modulo WiFi li trovate negli articoli dedicati a Fishino UNO e Fishino MEGA.



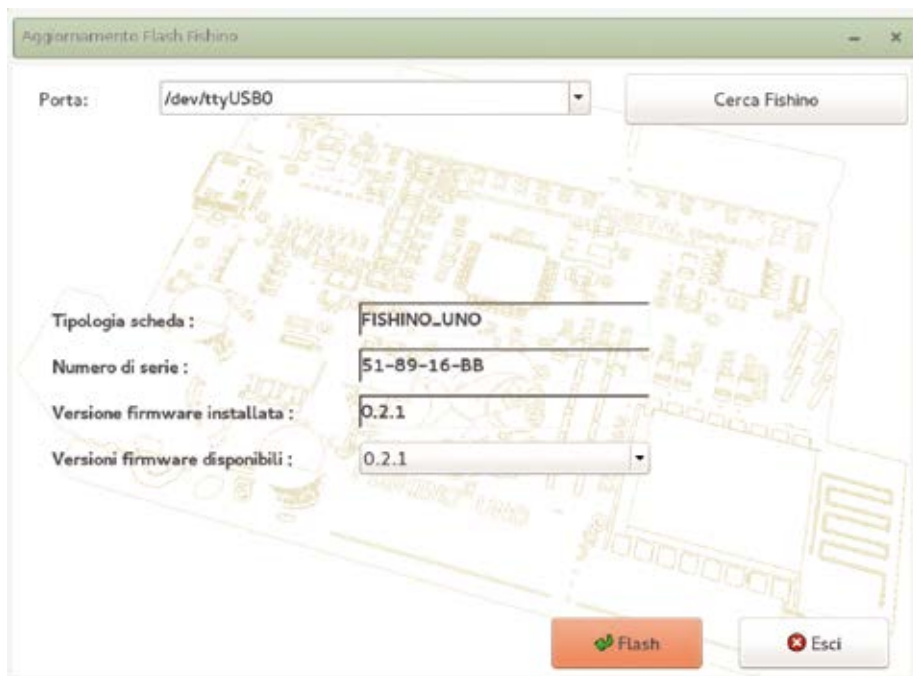


Fig. 2

CONNETTORE PER MICROSD

Anche questo stadio non è stato modificato rispetto alle Fishino MEGA e Fishino UNO, per cui rimandiamo chi volesse approfondirne il funzionamento agli articoli relativi a tali schede.

AGGIORNAMENTO DEL FIRMWARE DEL MODULO WIFI

Come per i modelli precedenti, suggeriamo di eseguire un aggiornamento del firmware al primo utilizzo e comunque periodicamente, perché è un programma in continuo sviluppo ed intimamente connesso con le librerie fornite. La procedura di aggiornamento è praticamente identica a quella dei precedenti modelli di Fishino; differiscono le connessioni da effettuare, che sono quelle illustrate nella **Fig. 1**. La procedura di aggiornamento è semplificata da un programma apposito, disponibile sia per la piattaforma Windows che per Linux, che esegue l'operazione in modo completamente automatico ed a prova di errore. I passi da compiere per l'aggiornamento sono i seguenti.

1. Caricare uno sketch che non

utilizzi la porta seriale. L'esempio base BLINK (quello che fa lampeggiare il led sulla scheda) è perfetto; tale passaggio serve ad evitare interferenze tra lo sketch caricato ed il collegamento seriale tramite l'ATmega e l'ESP; se il programma di flash non rileva Fishino, al 99% il problema è uno sketch sbagliato caricato.

2. Connettere la porta TX di Fishino Guppy con la porta ESP-TX sul connettore ESP, e la porta RX della Fishino Guppy con la porta ESP-RX sul connettore ESP (vedere la **Fig. 1**).
3. Connettere la porta GPIO0 a massa tramite un cavetto o un ponticello sempre sul connettore ESP (vedere figura a fianco).
4. Collegare la Guppy al PC (o premere il pulsante di RESET, se già connessa).
5. Lanciare il programma **FishinoFlasher**, assicurandosi che il PC sia connesso ad Internet.

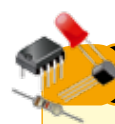
Se i collegamenti sono stati eseguiti correttamente, il programma rileverà la porta a cui è connessa la Guppy, determinerà il modello e la versione del

firmware attualmente installata, si collegherà ad un server remoto e scaricherà la lista dei firmware disponibili, mostrando l'ultimo ma permettendo comunque la selezione delle versioni precedenti (**Fig. 2**) nel caso si voglia fare un downgrade. Facendo clic sul pulsante "Flash" verrà avviata la procedura di aggiornamento alla fine della quale apparirà un messaggio di conferma. Per terminare il programma occorre fare clic sul pulsante "Esci".

Nel caso la Guppy non venga rilevata automaticamente, è possibile provare a selezionare la porta manualmente. È comunque probabile che siano stati commessi degli errori nei collegamenti.

La selezione manuale risulta indispensabile nel raro caso in cui più di una Fishino sia connessa contemporaneamente al PC, nel qual caso il primo viene rilevato automaticamente ma resta la possibilità di sceglierne un altro. Una volta terminata la procedura è sufficiente eliminare i tre collegamenti e **Fishino** sarà pronto per l'uso con il nuovo firmware.

Bene, abbiamo concluso: siete pronti per sviluppare le vostre applicazioni con la Guppy! ■

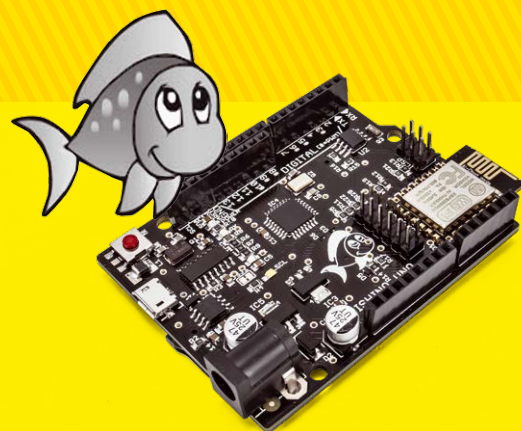


per il MATERIALE

La board Guppy (cod. Guppy) è in vendita presso Futura Elettronica al prezzo di Euro 33,90. Il prezzo si intende 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>

Fishino Contest: I PROGETTI VINCITORI



“FISH AND KILOWATT”

di MAURO SEGAFREDO

Monitoriamo i consumi di elettricità con la Fishino Uno impiegando un circuito non invasivo che legge gli impulsi dei LED del contatore elettronico che ormai tutti abbiamo in casa e al lavoro.

Fishino Contest si è concluso e come anticipato, vogliamo pubblicare in questo e nei prossimi due fascicoli i tre progetti premiati dalla giuria; questo, perché riteniamo giusto dare spazio alle soluzioni proposte e condividerle anche con chi non è assiduo frequentatore del web. In queste pagine vediamo il primo classificato. Il progetto in questione permette di realizzare un monitor dei consumi elettrici che consente di:

1. leggere in tempo reale la potenza istantanea consumata (i dati vengono aggiornati ogni 5 secondi);
2. registrare su file in formato CSV, il log dei consumi orari, giornalieri e mensili;
3. visualizzare in tempo reale, l'andamento dei consumi in

- un grafico aggiornato ogni 5 secondi (modalità *Live View*);
4. scaricare da browser web i file di log;
5. visualizzare su browser web i grafici dei consumi registrati sui file di log;
6. essere avvisati del superamento di una soglia prestabilita di consumo istantaneo, tramite la app *Pushetta* (disponibile gratuitamente sia per dispositivi Android, che IOS);
7. verificare il superamento di una soglia prestabilita da browser, grazie al cambiamento di colore dello sfondo;
8. inviare sull'uscita digitale 2 di Fishino un impulso in corrispondenza di ogni watt consumato.

Le funzionalità 3, 5 e 6, sono

disponibili solo in modalità WiFi. Con il progetto proposto, si potranno quindi mettere sotto osservazione i consumi elettrici dei singoli elettrodomestici di casa conoscendone in ogni momento l'effettiva potenza assorbita; si potrà inoltre, analizzare lo storico dei consumi per individuare, ed eventualmente correggere, l'utilizzo dei vari elettrodomestici e l'incidenza in *stand-by*.

Il principio di funzionamento dell'applicazione si basa sul fatto che ad ogni lampeggio del LED superiore del contatore elettrico corrisponde l'assorbimento di un watt di potenza attiva; ricordiamo che i contatori elettronici riescono a leggere la potenza attiva. Se riusciremo a far riconoscere i lampeggi a Fishino, sarà sufficiente calcolare il tempo che intercorre fra due consecutivi per poter risalire al consumo.

COME FUNZIONA

Stabilito che ad ogni lampeggio del LED superiore del contatore corrisponde l'assorbimento di 1 watt, se ad esempio rilevassimo un lampeggio ogni due secondi avremmo 30 lampeggi al minuto, che corrispondono a $60 \times 30 = 1.800$ lampeggi in un'ora, corrispondenti a un consumo di 1,8 kWattora. Se il contatore lampeggiasse più velocemente, ad esempio alla frequenza di un lampeggio al secondo, avremmo 60 lampeggi al minuto, pari a 3.600 lampeggi in un'ora, che equivalgono a un consumo di 3,6 kWh. Potremmo fare altri calcoli ma il meccanismo ormai dovrebbe essere chiaro. Vedremo più avanti il circuito per interfacciare Fishino al contatore; per ora basta sapere che il rilevamento dei lampeggi del LED viene effettuato puntandogli contro un fotoreistore.

Adesso concentriamoci per un istante sull'aspetto firmware. Utilizzando la funzione **millis()** siamo in grado di rilevare il numero di millisecondi trascorsi da quando la scheda Fishino ha iniziato l'esecuzione del programma. Se salviamo in una variabile il risultato della funzione **millis()** calcolata in corrispondenza di due lampeggi consecutivi del LED, possiamo calcolare per differenza i millisecondi intercorsi per la frequenza corrispondente al consumo di un watt. Nel programma, la variabile si chiama *Gftinterval* e la formula per calcolare i W/h è:

$$(3600 \times 1000) : Gftinterval$$

$$Gftinterval = (GLtflash - GLtoldflash) ;$$

$$GLtoldflash = GLtflash;$$

$$GFPotIstantanea = 3600 / Gftinterval * 1000; //Calcolo la potenza istantanea$$

Se leggiamo un lampeggio ogni due secondi, *Gftinterval* sarà uguale a 2.000 ms. Applicando la formula si ottiene:

$$(3600 \times 1000) : 2000 = 1800 = 1,8 \text{ kW/h}$$

IL CIRCUITO ELETTRICO

Per poter rilevare un lampeggio realizziamo un semplice partitore di tensione collegando la fotoreistenza ai +5V, e chiudendo il circuito a massa tramite una resistenza da 47 Kohm. Il punto centrale

del partitore verrà collegato all'ingresso analogico A0 di Fishino e con la funzione **analogRead(A0)** potremo monitorare la variazione di tensione provocata dal lampeggio del LED.

Negli esperimenti fatti dall'autore il contatore è posizionato in un garage dove sono presenti delle finestre che danno su una strada. Il valore letto con **analogRead(A0)** è stato quindi influenzato sia dall'accensione o meno dell'illuminazione del garage, sia dal variare durante il corso del giorno della quantità di luce che filtra dalle finestre nonché dal passaggio di auto con i fari accesi. Tenete conto di questa esperienza nel realizzare il vostro sistema di captazione

della luce dei LED, ricordando che se l'ambiente è illuminato dovete affacciare i fotoreistori sui corrispondenti LED, cercando di far passare meno luce possibile dall'ambiente.

Nel collocare fisicamente il sistema ottico va considerato che sul pannello frontale del contatore è presente anche un secondo LED che lampeggia ogni volta che viene impegnato un watt di potenza reattiva (che si chiama VAR, ossia Volt-Ampere Reattivo). La potenza reattiva non entra nel calcolo della bolletta e pertanto dobbiamo ignorare tale lettura, tuttavia il lampeggio del relativo LED si propaga tramite la plastica del coperchio del contatore e può interferire con la lettura del LED della potenza attiva. Per risolvere il problema l'autore ha aggiunto una seconda fotoreistenza da posizionare in corrispondenza del LED inferiore; così, invece di rilevare un impulso derivante dalla lettura di un



L'autore del progetto,
il Sig. Mauro Segafredo.

valore assoluto, si opera una lettura differenziale fra le due fotoresistenze. In pratica, tutte le condizioni di disturbo sopra elencate si manifestano in egual misura sulle misurazioni delle due fotoresistenze. Facendo la differenza dei valori rilevati si otterrà un valore positivo se viene acceso il LED superiore, ovvero negativo se sta lampeggiando il LED inferiore, mentre risulterà un valore prossimo a zero se nessuno dei LED si accende e vengono percepite le sole interferenze ottiche succitate. Con questo semplice accorgimento, riduciamo drasticamente le possibilità di errore nel rilevamento.

Detto ciò passiamo alla realizzazione del circuito definitivo, che richiede due fotoresistenze, e due resistenze fisse. In particolare, l'apparato si compone di due fotoresistenze Futura Elettronica (codice 8220-LDR09) e due resistenze fisse da 47 Kohm. La realizzazione è molto semplice e non è previsto alcun PCB.

Le due fotoresistenze devono essere collegate ai +5V, e devono chiudersi a massa tramite una resistenza da 47 Kohm. Il punto centrale del partitore riferito al LED superiore (potenza attiva), viene applicato all'ingresso analogico A0 di Fishino. Il punto centrale del partitore riferito al LED inferiore (potenza reattiva), viene applicato all'ingresso analogico A1 di Fishino. Le fotoresistenze dovranno essere posizionate in corrispondenza dei due LED del contatore e fissate con del comune nastro adesivo (si potrebbe utilizzare della colla, ma è meglio non farlo perché il contatore è proprietà dell'azienda elettrica, che non gradirebbe).

Per il suo prototipo, l'autore ha inserito le due fotoresistenze in due tubicini neri che poi ha alloggiato in una vecchia lampada con braccio snodabile. In questo modo ha potuto posizionare l'apparato in maniera molto accurata senza intaccare in alcun modo il contatore.

Dal momento che i due LED del contatore non saranno più visibili, Fishino farà da ripetitore per quanto riguarda il solo LED superiore (quello relativo alla potenza attiva).

Durante il campionamento della tensione corrispondente a un lampeggio, Fishino porterà al valore *HIGH* l'uscita digitale 2, alla quale sarà possibile collegare un LED e una resistenza da 180 ohm, per ripetere l'accensione del LED del contatore.

Per quanto riguarda l'alimentazione, non ci sono particolari indicazioni in quanto non vi sono carichi esterni da alimentare, o shield aggiuntivi.

Non è richiesto alcun sistema di mantenimento dell'alimentazione in caso di stacco del contatore, in quanto in una tale situazione, non vi sarebbero consumi da monitorare.

DESCRIZIONE DEL SOFTWARE

Entriamo adesso nel vivo della programmazione, analizzando per prima la parte contenente la dichiarazione delle variabili.

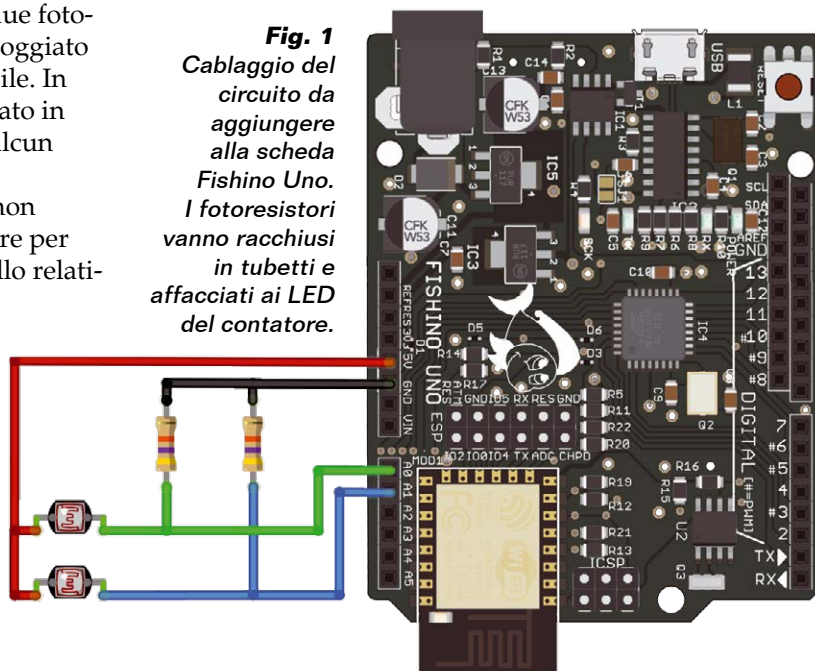
Sezione dichiarativa

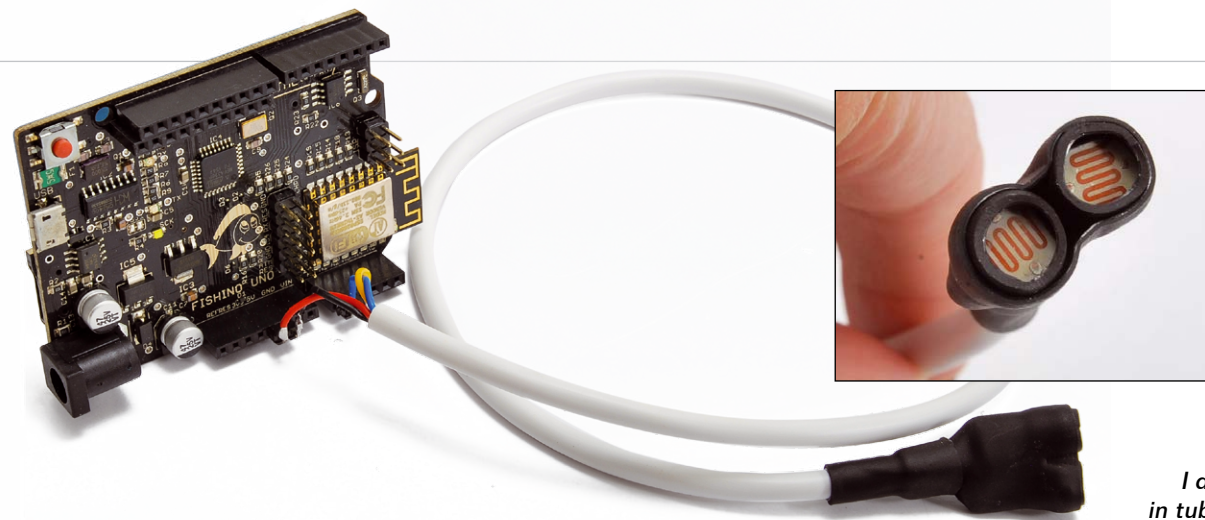
La sezione dichiarativa presenta le variabili e le funzioni di supporto ordinate come di seguito.

- Gestione modulo WiFi (con la possibilità di definire la modalità *standalone* o *normal* come da esempio FishinoWiFiWebServer).
- Gestione dell'interfaccia web server.
- Gestione della lettura potenza istantanea. La variabile *GISoglia*, contiene il valore differenziale che indica al programma l'accensione del LED superiore. Il valore predefinito è stato rilevato in maniera sperimentale ed è impostato a 90. Eventualmente può essere variato. La variabile *GIContaFlash*, indica il numero di campionamenti consecutivi con superamento della soglia differenziale, necessari a rilevare un lampeggio valido. Il valore predefinito è stato rilevato in maniera sperimentale ed è impostato a 150. Eventualmente può essere variato.

La funzione **CheckFlash()** si fa carico di campio-

Fig. 1
Cablaggio del circuito da aggiungere alla scheda Fishino Uno. I fotoresistori vanno racchiusi in tubetti e affacciati ai LED del contatore.





I due fotoreistori racchiusi in tubetti di termorestringente (all'interno del quale si trovano anche i resistori fissi) uniti e collegati alla Fishino.

nare la tensione presente sui due partitori, calcolarne la differenza e confrontarla con il valore *GISoglia*. Vengono poi contate le volte consecutive che si supera tale soglia, e se si raggiunge il valore indicato in *GIContaFlash*, si assume il rilevamento di un lampeggio valido. In questo caso la funzione restituisce al programma principale il valore rilevato dalla chiamata **millis()**, altrimenti restituisce 0.

- Gestione delle notifiche tramite Pushetta (con compilazione condizionale in quanto valido solo in modalità *normal*). È possibile configurare la variabile *GLLimite*, impostando il limite in watt superato il quale viene inviata la notifica Pushetta. Il valore predefinito è 3000, che corrisponde a 3 kW. Troviamo poi la definizione dell'API Key associato al proprio account Pushetta, e la definizione del Channel sul quale far confluire la notifica. La *Content-Length* è impostata a 74 considerando la lunghezza del messaggio che apparirà sulla notifica Pushetta. Se si volesse variare il messaggio predefinito "Attenzione soglia superata", occorrerà ricalcolare tale dato con la formula $48 + \text{lunghezza messaggio}$. Per ulteriori dettagli, fate riferimento all'articolo su Elettronica In n° 194 di aprile 2015 e alla documentazione online (www.pushetta.com/).
- Gestione dell'orario e dei nomi file di log. Non è stata utilizzata la libreria RTCLib; dovendo solo leggere l'orario, l'autore si è interfacciato direttamente ai registri dell'RTC, risparmiando così molta memoria di programma.
- Il naming dei file di log è il seguente:

```
char GLFname_m[] = "aamm.csv"; //File di log mensile
char GLFname_g[] = "aammgg.csv"; //File log giornaliero
char GLFname_h[] = "aammgghh.csv"; //File log orario
```



Fig. 2 - L'app Pushetta notifica su smartphone il superamento della soglia.

Sezione setup()

La sezione *setup()* inizializza il modulo WiFi di Fishino e le interfacce SERIAL, SPI, SD e WIRE.

Procede poi alla lettura dell'ora e all'impostazione delle relative variabili. Vengono anche impostati i nomi dei tre file di log. In ultimo, viene attivata l'interfaccia web.

Ciclo principale loop()

Il loop principale svolge in sequenza le seguenti funzioni:

1. Verifica cambio di minuto, ora o giorno. In corrispondenza dello scadere di un minuto, ora o giorno, vengono registrate le letture di consumo e successivamente aggiornati i

nomi dei file di log:

```
if ( getDateTime() ) {
  if ( GLm != GLmOld ) {
    // scaduto un minuto, scrivi su file GLFname_h
    // il numero di watt (GLwm) consumati nell'ultimo minuto (GLmOld)
    dataFile = SD.open(GLFname_h, FILE_WRITE);
    if (dataFile) {
      dataFile.print(GLmOld);
      dataFile.print(",");
      dataFile.println(GLwm);
      dataFile.close();
    }
    // azzero i watt consumati nel minuto
    GLwm = 0;
    GLmOld = GLm;
  }
}
```

2. Campionamento del LED superiore del contatore per rilevare l'assorbimento in watt e calcolare la potenza istantanea assorbita.

```
// campiono il led della potenza attiva
GLtflash = CheckFlash();
// se rilevato il flash, procedo con il calcolo della potenza istantanea
```

```

if ( GLtflash > 0 )
{
  GFTinterval = (GLtflash - GLtoldflash) ;
  GLtoldflash = GLtflash;
  GFPotIstantanea = 3600/GFTinterval*1000; //Calcolo la potenza istantanea
  //ho consumato un watt, aggiorno i contatori di consumo
  GLwm++;
  GLwh++;
  GLwg++;
}

```

3. Gestione delle chiamate web; la struttura di questa sezione deriva da FishinoWiFiWebServer. Sono state gestite le seguenti risorse:

- "index", per servire la pagina iniziale index.htm. La pagina viene creata dinamicamente e la compilazione condizionale esclude dalla risposta i riferimenti non ammessi in modalità stand-alone.

```

#ifndef STANDALONE_MODE
client << F("<a href='\"liveview.htm'\">Live View</a><BR>");
client << F("<a href='\"chfile.htm'\">Grafici Storici</a><BR>");
#endif

```

- "FileName", per l'invio dei file di log CSV. Sia per lo scarica da browser, che per la creazione dinamica dei grafici.
- "leggivalore", per l'invio in formato json, della potenza istantanea. Questa risorsa è utilizzata nella modalità *Live View*.
- "liveview", per servire la pagina htm corrispondente alla modalità *Live View*: *liveview.htm*.
- "chfile", per servire la pagina htm corrispondente alla modalità Grafici Storici: *chfile.htm*.
- "logfile", per servire la pagina htm corrispondente alla modalità Log Storici: *logfile.htm*.

4. Gestione notifica Pushetta; se viene rilevato il primo superamento della soglia oppure se è passato più di un minuto dall'ultimo superamento della soglia, parte la notifica (Fig. 2).

Anche in questo caso, ci si avvale della compilazione condizionale per escludere o meno tale chiamata.

LE PAGINE HTML

La struttura delle pagine HTML è la seguente: **index.htm** è la pagina principale, che viene creata dinamicamente direttamente dal programma che gira in Fishino (Fig. 3). La pagina ha una clausola di refresh impostata a 5 secondi. Al superamento della

Fish and Kilowatt Potenza Istantanea: 513.77 Wh

[Live View](#)
[Grafici Storici](#)
[File di Log](#)

Fig. 3
La pagina web index.htm.

soglia prestabilita, la pagina cambia colore.

```

if (GFPotIstantanea > GLLimite)
{
  client << F("<body bgcolor='\"#ff0000'\">\n");
}

```

liveview.htm; questa pagina consente di visualizzare l'andamento in tempo reale della potenza istantanea su di un grafico. Il refresh è di 5 secondi, e si utilizza una chiamata ajax con jQuery alla risorsa *leggivalore*. Se si interroga con il browser l'indirizzo della nostra scheda Fishino, seguito dalla stringa *leggivalore*, si ottiene la risposta in formato Json che contiene il tempo in millisecondi dall'avvio della scheda, e il valore della potenza istantanea (Fig. 4).

```

if (StrContains(HTTP_req, "leggivalore")>0) {
  SendHeaderOk();
  client << F("Content-Type: text/json\n\n");
  client << F("(" << millis() << F(",") << GFPotIstantanea << F(")\n");
}

```

Per la gestione dei grafici, l'autore si è avvalso delle librerie highcharts (www.highcharts.com); un

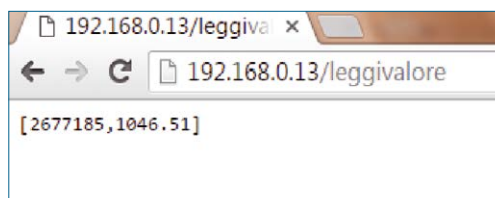


Fig. 4
Risposta in formato Json.



Fig. 5
Grafico in liveview.htm.

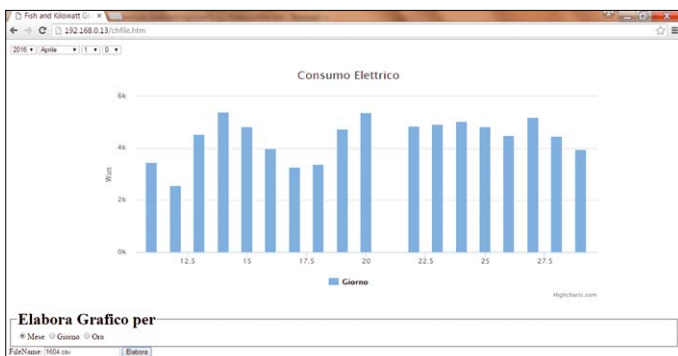


Fig. 6 - Grafico mensile.

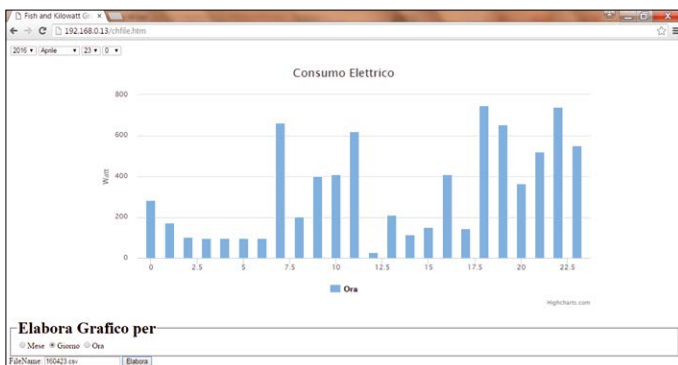


Fig. 7 - Grafico giornaliero.

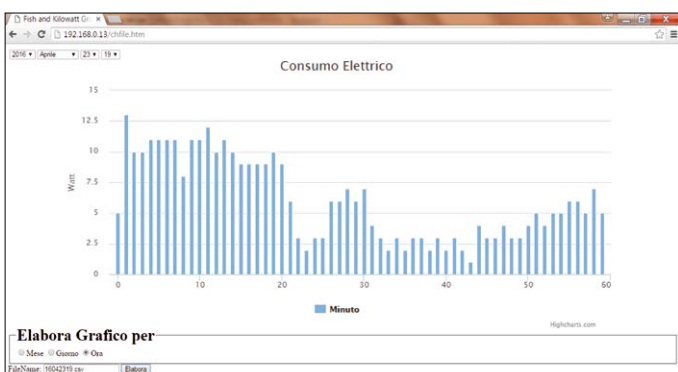


Fig. 8 - Grafico orario.

esempio è nella Fig. 5, che mostra un grafico della potenza all'interno della pagina *index.htm*.

chfile.htm; da questa pagina è possibile visualizzare in formato grafico, i log file salvati su scheda microSD. È necessario selezionare la data, ed eventualmente anche l'ora, per generare i tre tipi di grafici: mensile (Fig. 6), giornaliero (Fig. 7) e orario (Fig. 8). Anche in questo caso ci si avvale delle librerie highcharts e a livello Fishino, si fa riferimento alla risorsa "FileName="

logfile.htm – da questa pagina è possibile scaricare i log file salvati su scheda microSD. L'interfaccia è simile a quella già illustrata per i grafici. Ad esem-

Fig. 9
Parametri per
il file di log.



Fig. 10
Fotoresistori
applicati al
contatore e
collegati con
fili inseriti in un
tubo flessibile.

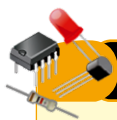
pio, con i parametri mostrati nella Fig. 9 possiamo scaricare il file CSV giornaliero del 24 aprile 2016.

CONSIDERAZIONI FINALI SUL SOFTWARE

Dovendo ottimizzare al massimo l'occupazione del software, l'autore ha tolto quasi tutte le segnalazioni di debug su seriale. Prima di caricare lo sketch, consigliamo quindi di verificare la corretta configurazione dei singoli elementi:

1. rete WiFi o stand-alone (utilizzando lo sketch di esempio FishinoWiFiWebServer);
2. scheda microSD (utilizzando gli esempi della libreria SD);
3. RTC (utilizzando gli esempi della libreria RTCLib; ricordate di impostare l'ora con l'esempio ds13078).

Bene, con questo si conclude la descrizione della prima applicazione premiata dal Fishino Contest; appuntamento al fascicolo di ottobre per la successiva, che consisterà nella realizzazione di un joystick basato su Fishino. Potete trovare tutti i progetti partecipanti alla pagina: blog.elettronica.in/tag/fishino-contest-i-progetti/



per il MATERIALE

Tutti i componenti necessari per realizzare questo progetto possono essere acquistati presso Futura Elettronica. La board Fishino UNO (cod. FISHINOUNO) è in vendita al prezzo di Euro 36,00, mentre la fotoresistenza LDR 20-50K (cod. LDR09) è disponibile a Euro 1,00. I prezzi si intendono IVA inclusa.

Il materiale va richiesto a:

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

Quasi un anno fa abbiamo presentato Fishino UNO, una board Arduino-compatibile nata per l'accesso a Internet e al mondo dell'IOT senza bisogno di shield dedicati. Il riscontro favorevole ci ha spinto a realizzare due ulteriori unità, alimentabili a batteria ricaricabile e quindi utilizzabili in assenza di alimentazione di rete: Fishino Mega (rivista 204), per chi necessita più pin e memoria per le proprie applicazioni, ed il Fishino Guppy (rivista 206), il "pesce" più piccolo del nostro acquario, per applicazioni davvero miniaturizzate.

Che altro potevamo aggiungere? Guardandoci attorno e sviluppando, ci siamo resi conto che le limitazioni imposte dagli 8 bit dei processori Atmega 328p e Atmega 2560 iniziavano a farsi sentire, sia in termini di spazio per le applicazioni, sia in termini di prestazioni. Abbiamo quindi deciso (già da un po', a dire il vero, ma l'impegno sia dal lato hardware che dal lato software ha comportato lunghi tempi di sviluppo) di passare a un processore a 32 bit. Sì, ma quale? La scelta tra ciò che offre il mercato è stata

difficile.

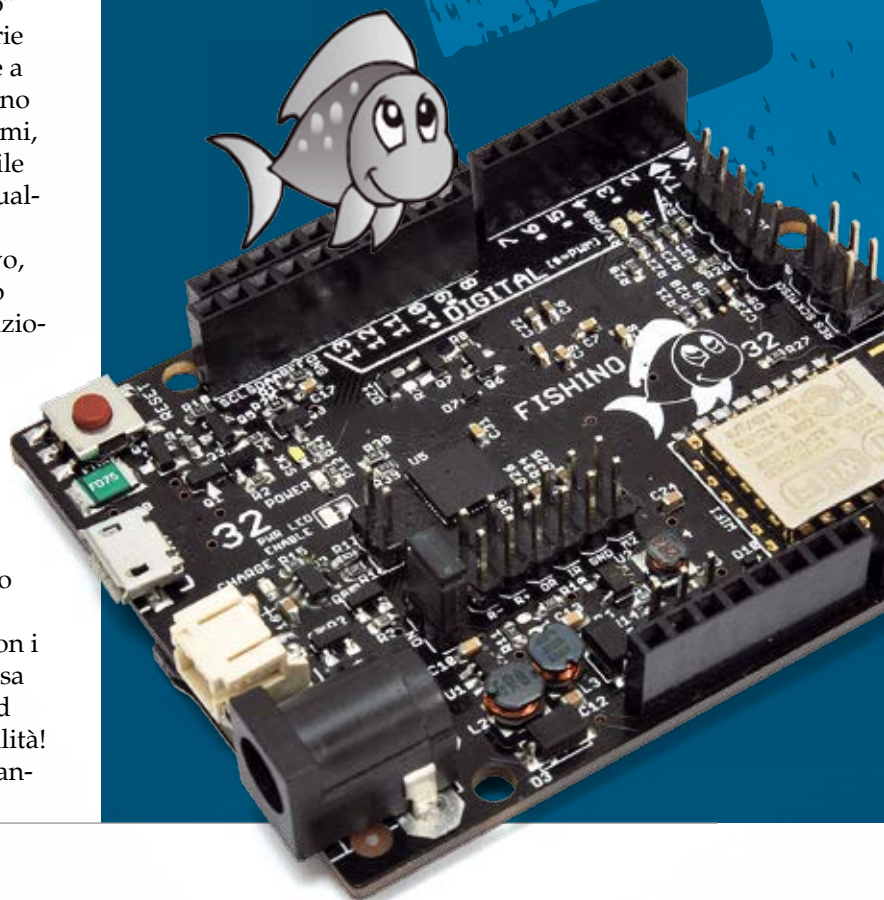
Per ragioni che andremo a spiegare a breve, siamo andati un po' controcorrente, abbandonando gli Atmel (e quindi gli ARM, nel campo dei 32 bit) in favore dei PIC della Microchip, nella versione a 32 bit, in architettura MIPS.

Inizialmente la scelta era caduta sulle "nuovissime" MCU PIC32MZ, la serie più prestante, ma in seguito, a causa dell'immaturità hardware e soprattutto dell'assenza attuale di uno stack USB veramente open source, abbiamo "ripiegato" sulla collaudata serie PIC32MX che, oltre a non soffrire di alcuno dei suddetti problemi, ha il non trascurabile pregio di costare qualcosa in meno.

È un ripiegato relativo, visto che il modello utilizzato ha prestazioni di tutto rispetto, ovvero un clock a 120 MHz, una Flash Memory da 528 kByte ed una memoria RAM da 128 kByte: un abisso rispetto a quanto eravamo abituati con i modelli ad 8 bit, cosa che apre a nuove ed interessanti possibilità! Avendo fatto il "gran-

FISHINO A 32 BIT!

di MASSIMO DEL FEDELE



La nostra scheda di prototipazione guadagna il processore a 32 bit e punta in alto, verso applicazioni impegnative che possono sfruttare a pieno il supporto WiFi, l'RTC e l'SD-Card che distinguono Fishino dalla comune Arduino UNO. Prima puntata.



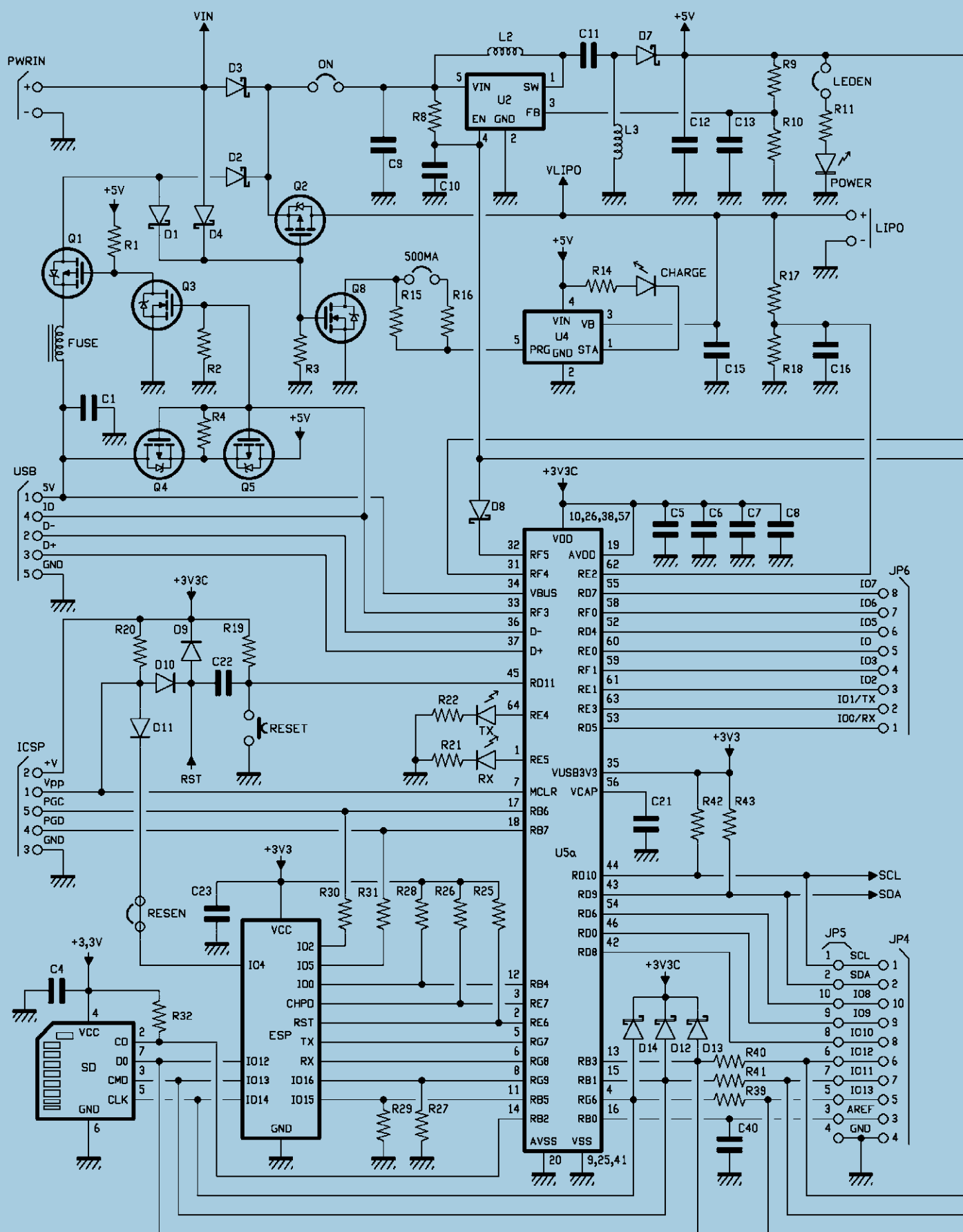
de balzo" nel mondo dei 32 bit, ci siamo poi chiesti se era possibile aggiungere qualcos'altro alla nostra scheda capostipite, che avevamo già ipotizzato nel formato collaudato e molto amato del Fishino UNO. Il PIC ha un'interfaccia USB incorporata (nativa), cosa che ci fa risparmiare lo spazio del convertitore; abbiamo inoltre deciso di sopprimere l'RTC esterno, visto che il controller utilizzato ne ha uno interno, dotato di molte possibilità aggiuntive rispetto al DS1307 utilizzato nel Fishino UNO, come per esempio la capacità di risvegliare il controller dallo standby ad orari o intervalli di tempo prefissati.

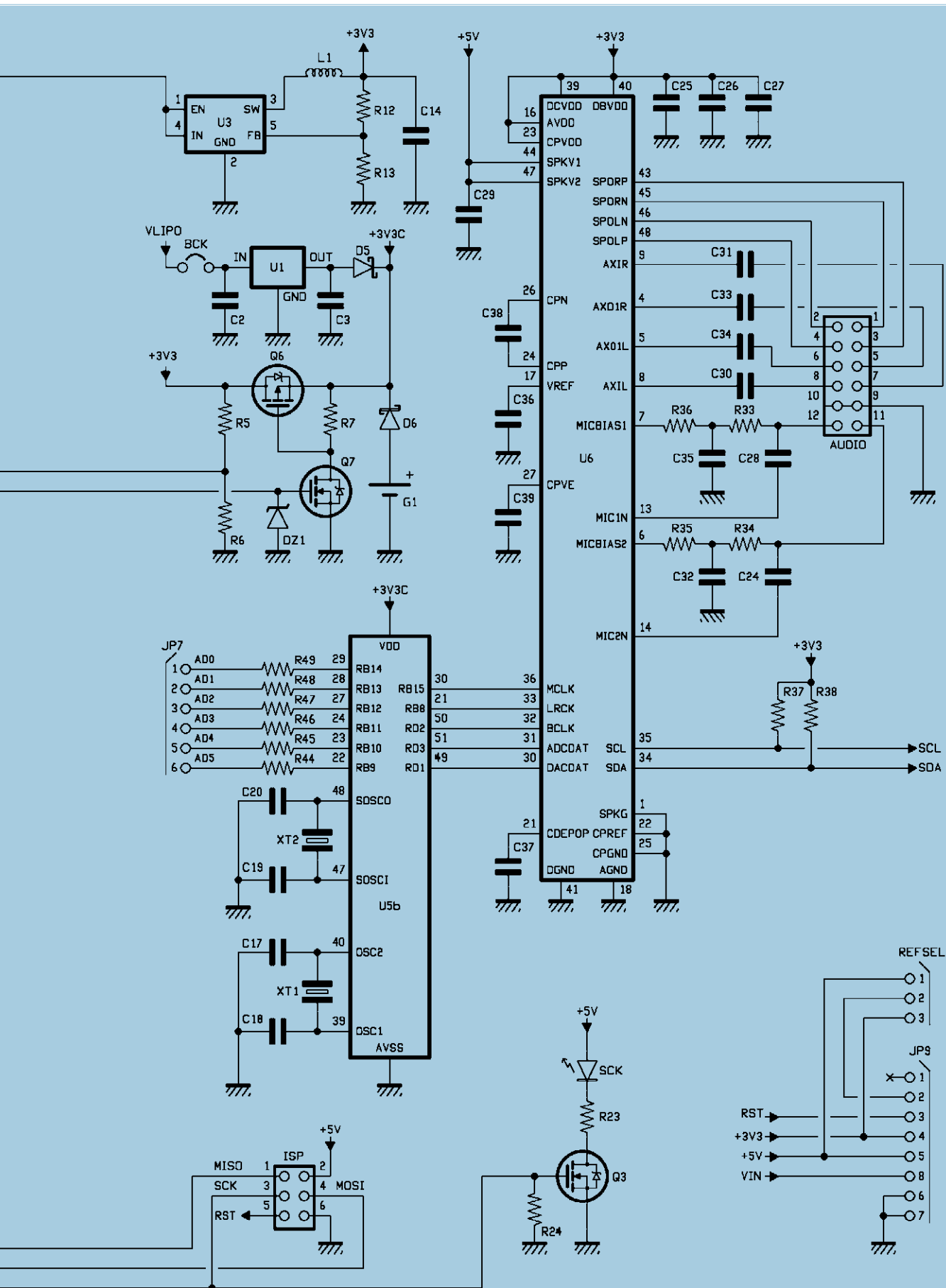
Come utilizzare, quindi, lo spazio aggiuntivo? Per prima cosa abbiamo inserito l'ormai collaudatissimo alimentatore SEPIC, già presente nel Mega e nel Guppy, rendendo quindi possibile un'alimentazione a batteria. Che altro potevamo inserire? Beh, un bel codec audio! Il codec scelto è un collaudatissimo ALC5631Q, le cui caratteristiche complete elencheremo nella sezione a esso dedicata; al momento vi basti sapere che include

un DAC (convertitore digitale/analogico) stereo, un ADC (convertitore analogico/digitale) stereo, un amplificatore in classe D in grado di pilotare direttamente una coppia di altoparlanti, ingressi microfonici, equalizzatore, effetti audio quali pseudostereo e stereo spaziale ...e molto altro. Con questo componente è, per esempio, possibile realizzare, senza bisogno di hardware aggiuntivo, un riproduttore audio, un feedback vocale per i propri progetti (che dire, ad esempio, di far leggere la temperatura di una sonda a voce dal nostro Fishino ?), un registratore audio, un controllo vocale, eccetera. È persino possibile sfruttare il codec per realizzare un sintetizzatore polifonico, cosa che interesserà gli appassionati di musica.

Ecco quindi le caratteristiche del nostro Fishino 32, capostipite di una serie a 32 bit di cui abbiamo già pronto il secondo modello, di caratteristiche inferiori ma con dimensioni estremamente ridotte, che presenteremo a breve:

- processore PIC32MX470F512;
- 512 kB di ROM;
- 128 kB di RAM;
- clock a 120 MHz;





- interfaccia USB nativa, sia Device che Host;
- RTC incorporato;
- supporto schede microSD;
- modulo WiFi;
- codec audio;
- alimentazione a batteria, plug e/o via connettore USB, da 3 volt circa fino a 20 volt;
- ricarica automatica di una batteria LiPo, se presente l'alimentazione esterna;
- possibilità di spegnere via software tutta la scheda, lasciando alimentato solo il processore in standby;
- possibilità di risveglio dallo standby tramite pin esterno o in tempi prefissati, da RTC;
- formato standard Arduino UNO;
- funzionamento interno a 3,3 volt;
- pin DIGITALI 5V-tolerant, quindi non c'è pericolo di bruciare la scheda con shield a 5 volt.

Abbiamo poi aggiunto una "chicca": grazie a un bootloader appositamente studiato, non è più necessario effettuare alcun collegamento esterno per aggiornare il firmware del modulo WiFi; con una pressione prolungata del tasto reset la scheda si predisporrà automaticamente in modalità aggiornamento, sfruttando il connettore USB. Qualcuno sarà probabilmente preoccupato dalla difficoltà di programmare questa nuova scheda, ma possiamo rassicurarlo; un po' grazie al lavoro del team di Arduino che ha reso l'IDE "estensibile" in modo estremamente semplice, un po' per il lavoro del team di Chipkit che abbiamo parzialmente sfruttato (anche se con parecchie modifiche), un po' grazie ad un buon numero di librerie software da noi scritte e/o adattate, la scheda

si programma esattamente come un normalissimo Arduino UNO, tramite l'IDE e con la sintassi cui siamo abituati.

È sufficiente eseguire una piccola procedura nell'IDE stesso, che permette di attivare gli strumenti di sviluppo per le MCU PIC ed il gioco è fatto. Per i componenti aggiuntivi abbiamo predisposto una serie di librerie che permetteranno di sfruttarli appieno; alcune di queste (soprattutto la libreria audio) sono in fase di sviluppo e quindi verranno aggiornate frequentemente. Sperando di aver stimolato il vostro "appetito" di maker, passiamo all'esame dello schema elettrico.

SCHEMA ELETTRICO

Partiamo dallo stadio di alimentazione, che risulta decisamente più complesso rispetto a quello dei modelli precedenti, e questo per le seguenti ragioni:

- la necessità di funzionare sia come host che come device USB;
- la tripla alimentazione;
- la possibilità di spegnimento software di tutto lo stadio di alimentazione;
- la possibilità di attivare esclusivamente l'alimentazione del controller in standby.

Iniziamo dalla sezione di controllo dell'alimentazione USB; questa sezione dev'essere in grado di fornire alimentazione al Fishino dal connettore USB se connesso come device, mentre deve fornire un'adeguata alimentazione a un device eventualmente connesso al Fishino, che in questo caso si comporta da host USB.

Per poter eseguire la commutazione ci viene in aiuto il piedino che fa capo alla linea U_{SBID} , collegata sia al connettore USB che all'apposita porta del controller.

I cavetti microUSB sono costituiti in modo da avere ad un estremo il pin corrispondente all' U_{SBID} connesso a massa (tipo A), mentre all'altro estremo questo pin è scollegato (tipo B); l'apparecchiatura cui è connessa l'estremità di tipo A assume il ruolo iniziale di host e contemporaneamente fornisce l'alimentazione all'altra apparecchiatura (connessa, questa, all'estremo di tipo B) che assume il ruolo iniziale di device. Abbiamo parlato di ruoli iniziali per un motivo ben preciso: in una connessione OTG i due device connessi possono cambiare ruolo; la specifica garantisce soltanto che la linea verrà alimentata sempre dal device connesso all'estremità A.

Torniamo quindi al nostro schema; vediamo che alla linea U_{SBID} è connessa una resistenza di valore elevato ($R2$, 100 kohm) verso massa, il cui scopo è soltanto quello di non lasciare fluttuante la linea in caso di assenza di alimentazione, ed una resistenza, $R4$, di valore decisamente inferiore, (10 kohm) connessa tra i due source dei MOSFET $Q4$ e $Q5$. Come si evince dallo schema, su questi source, grazie ai diodi interni ai MOSFET, è sempre presente una tensione positiva (5V meno la caduta diretta sui diodi) proveniente o dalla presa USB o dall'alimentazione a 5V della scheda Fishino. Abbiamo quindi la certezza che la $R4$ sia connessa ai +5V in qualsiasi condizione di funzionamento. Se la linea U_{SBID} è scollegata da massa (connessione tipo B) questa tensione positiva, attraverso $R4$, porta in conduzione il MOSFET a canale N $Q3$, il quale a sua volta manda in conduzione il MOSFET a canale P $Q1$; contemporaneamente, sempre attraverso $R4$ i due MOSFET $Q4$ e $Q5$ visti precedentemente vengono posti in interdizione e,

essendo collegati in antiserie, non permettono il passaggio dell'alimentazione dai 5V di Fishino al connettore USB.

Sintetizzando, a U_{SBID} fluttuante, quindi quando inseriamo un connettore di tipo B, il fishino NON fornisce alimentazione al connettore USB che, per contro, fornisce alimentazione al Fishino attraverso il MOSFET Q1. Otteniamo quindi la prima parte dello scopo desiderato, ovvero che con un connettore di tipo B Fishino ricava l'alimentazione dalla linea USB comportandosi da device vero e proprio.

Se inseriamo, per contro, un connettore di tipo A, la linea U_{SBID} viene posta a massa, cosa che porta in conduzione Q4 e Q5 e contemporaneamente, sempre attraverso Q3, in interdizione Q1; l'alimentazione fluisce quindi dal Fishino verso il connettore USB e non rientra "dalla porta di servizio" essendo Q1 interdetto.

È da notare, ovviamente, che nel secondo caso (connessione di

tipo A) per poter fornire i 5V alla porta USB il Fishino dev'essere alimentato in qualche altro modo, ovvero dalla batteria o dal plug VIN, come vedremo in seguito.

Proseguendo verso il basso del circuito notiamo la serie di diodi D1+D4 e D2+D3 rispettivamente, i primi due dei quali sono per piccole correnti, mentre i secondi sono diodi di potenza. Vediamo inoltre il MOSFET Q2, al quale giunge l'alimentazione della batteria.

Anche qui il funzionamento è abbastanza sottile; i diodi di potenza (D2 e D3), insieme al MOSFET Q1, costituiscono una porta OR di potenza, che confluisce nella linea V_{OR} dello schema. Il funzionamento dei diodi è abbastanza semplice: la tensione maggiore tra V_{IN} e V_{USB} viene trasmessa su V_{OR} , mentre l'altra, a causa dell'interdizione del diodo corrispondente, che si trova polarizzata inversamente, viene isolata. Ci si potrebbe chiedere perché sulla linea della batteria non abbiamo

inserito semplicemente un diodo di potenza come per le altre due, cosa che effettivamente avrebbe semplificato il circuito, realizzando una porta OR di potenza con tre diodi. Questo è fattibile, ma c'è il solito problema della caduta di tensione sui diodi che, se per tensioni elevate è praticamente influente, su una tensione ridotta come quella della batteria porta a perdite considerevoli: intuitivamente, si può dire che una caduta di tensione di 0,5 volt su 10 volt corrisponde al 5%, mentre una caduta di 0,5 volt su 3,6 volt è pari a circa il 14%, cosa che si riflette sull'assorbimento in corrente come vedremo di seguito, e proprio dove servirebbe una maggior efficienza, ovvero nell'alimentazione a batteria.

Il mosfet Q2 non ha questo problema, visto che non presenta una caduta di tensione fissa come un diodo ma una resistenza di conduzione, particolarmente bassa nel modello selezionato; per contro, un MOSFET non ha il

CARATTERISTICHE TECNICHE

- **Alimentazione:**
 - 5 V tramite connettore USB
 - 5 V sull'ingresso +5V
 - da 3,5 a 20 V sull'ingresso Vin
 - da 3,5 a 20 V sul plug di alimentazione
 - batteria ricaricabile agli ioni di litio, a cella singola da 3,7 V, sul connettore BAT
 - caricabatteria incluso
- **Livelli digitali:**
 - 3,3 V (5 V tollerati sugli ingressi digitali eccetto i due aggiuntivi caratteristici Fishino)
- **Controller:**
 - PIC32MX470F512H
- **Clock:**
 - 120 MHz, riducibile via software
- **Memoria:**
 - 512 kByte Flash
 - 128 kByte RAM
 - EEPROM emulata via software
- **Porte disponibili:**
 - 14 digitali di cui 5 in PWM
 - 6 analogiche
 - 2 digitali a 3,3 V
- **Interfacce disponibili:**
 - 1 x SPI esterna condivisa con WiFi ed SD
 - 1 x SPI/I²S interna dedicata al codec
 - 1 x I²C
- **Moduli aggiuntivi sulla scheda:**
 - WiFi
 - Connettore per schede microSD
 - RTC incorporato nel processore
 - Codec Audio ad alta definizione
 - Controllo software completo dell'alimentazione



II PIC32MX470F512

Il microcontrollore PIC utilizzato nella nostra board appartiene alla famiglia PIC32MX, basata sull'architettura MIPS, che si distingue subito dalla classe di microcontrollori che equipaggiano Arduino, i quali sono Atmel AVR, basati su un core ARM.

Lo schema a blocchi che trovate in questo riquadro mostra l'architettura interna, a partire da un core MK4, in grado di funzionare fino a 120 MHz di clock, con una capacità di elaborazione di ben 198 DMIPS. Il core è dotato di capacità di moltiplicazione/divisione hardware a 32 bit, di un sistema per il debugging del codice e di altre notevoli caratteristiche, oltre ad essere contornato da una ricca dotazione di periferiche incorporate, una generosa dotazione di memoria Flash e RAM. Cosa non trascurabile per lo sviluppo di board di sviluppo e prototipazione, la MCU Microchip da noi impiegata in Fishino 32 offre la possibilità di "spostare" le connessioni alle periferiche incorporate su vari I/O, permettendo quindi una notevole flessibilità nella progettazione.

Il modello da noi scelto è al top delle prestazioni ed è dotato, inoltre, di un modulo USB OTG, che ne permette un

facile interfacciamento sia con device
che con host USB esterni.

Il controller è inoltre dotato di un completo sistema di power management, che permette di spegnere le periferiche non utilizzate, ridurre la velocità di clock per limitare i consumi e “ibernare” tutto il sistema, mantenendo acceso lo stretto indispensabile per poter reagire agli eventi esterni. Risulta quindi possibile, tramite software, “calibrare” il consumo della scheda a seconda delle necessità, fino all’arresto pressoché completo, allo scopo di aumentare fino al limite desiderato l’autonomia, nel funzionamento a batteria.



funzionalità minime, confluisca nei circuiti che si vogliono tenere spenti.

Normalmente, se è presente tensione sui +3,3V, questa confluisce attraverso il diodo interno a Q6, e anche attraverso il Q6 stesso se questo è in conduzione, sull'alimentazione +3V3Core del controller; è questo il funzionamento normale. Se noi invece togliamo l'alimentazione sui +3,3V, e contemporaneamente portiamo a livello basso la linea OFF, attraverso Q7 portiamo il Q6 ad interdizione e quindi impediamo ai +3V3Core di andare ad alimentare il resto della scheda. La linea OFF viene comandata dal controller, come vedremo in seguito, durante lo spegnimento programmato.

Si possono poi notare la pila al litio di mantenimento G1 e il diodo D6, che servono a fornire tensione al controller quando manca qualsiasi altra fonte in grado di alimentarlo. Per questo occorre fare attenzione: se togliamo completamente l'alimentazione al Fishino, questa pila finisce per alimentare tutta la scheda, a meno di non "prendere provvedimenti" attraverso il software portando a livello basso la linea OFF. Quindi, a differenza delle schede ad 8 bit, dove la pila serviva solo per la tensione di mantenimento dell'RTC, e la cosa veniva gestita in modo automatico, qui è necessario un intervento del software a supporto, altrimenti la pila si scaricherebbe in pochi minuti. Consigliamo quindi di non utilizzarla e lasciare il mantenimento alla batteria LiPo, a meno che non si sappia come gestire il risparmio energetico della scheda via software. Per ultima notiamo la linea PWRGOOD, connessa ad un partitore con la massa; questa fornisce un segnale positivo ad

pregio del diodo di condurre solo quando serve, per cui occorre un sistema per mandarlo in conduzione o interdirlo a seconda delle necessità. Per questo utilizziamo i due diodi per piccole correnti D1 e D4; il funzionamento del tutto è il seguente:

- se non è presente un'alimentazione su V_{IN} e nemmeno attraverso V_{USB} , la resistenza $R3$ porta a massa il gate di $Q1$ che quindi conduce e permette il passaggio della tensione della batteria verso V_{OR} ;
- se invece è presente una delle due (V_{IN} o V_{USB} , o entrambe) uno dei diodi $D1$ e $D4$ conduce, polarizzando positivamente $Q2$ che passa in interdizione scollegando quindi la batteria.

In breve, se abbiamo un'alimen-
tazione esterna, che proviene dal

plug V_{IN} o dal connettore USB, la batteria viene scollegata; se invece manca l'alimentazione esterna entra in funzione la batteria.

Si può notare che la linea su cui confluiscono i diodi D1 e D4 ha un nome, BATOFF, perché questo segnale, che è a livello alto se è presente un'alimentazione esterna, verrà utilizzato anche in un'altra sezione del circuito.

Vediamo ora il gruppo di componenti che fanno capo ai MOSFET Q6 e Q7; questi servono per scollegare la linea +3V3, ovvero la linea principale a +3,3 V, dalla linea +3V3Core, destinata ad alimentare solo il controller. A cosa servono? Semplicemente ad evitare che un'alimentazione destinata al solo controller, attivata quando tutto il resto è spento, e che ha lo scopo di mantenere attivo l'RTC interno ed altre

un I/O del controller che indica la presenza di un'alimentazione adeguata sulla scheda, e può venire utilizzato, appunto, per gestire la mancanza di questa e le funzioni di standby. Come anzidetto, si tratta di una gestione piuttosto complessa, per la quale è in preparazione un'apposita libreria software; per il momento, a meno che non sappiate come gestire la cosa, consigliamo di non utilizzare le funzioni di spegnimento software.

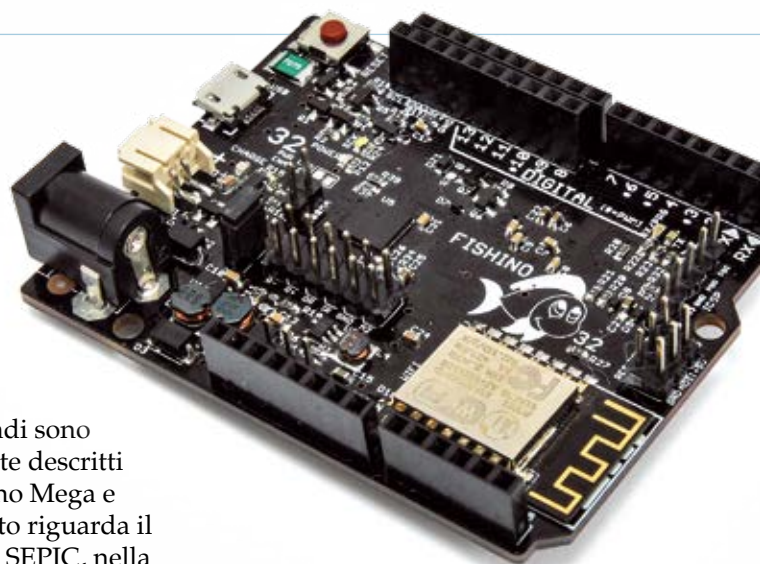
L'ultima parte della sezione in esame è quella che ruota attorno al regolatore lineare U1, un comune regolatore a bassa corrente (max 100 mA circa) e bassissimo consumo, destinato a fornire la tensione di mantenimento al controller tramite la batteria LiPo esterna. Questo ha lo stesso effetto della pila al litio G1, con l'indubbio vantaggio di utilizzare una batteria ricaricabile in grado di fornire un'autonomia estremamente lunga anche in presenza di consumi non proprio trascurabili. Questa sezione è disattivabile tagliando il ponticello BCK, normalmente connesso. Anche qui valgono (anche se in modo meno pressante) i discorsi sul risparmio energetico fatti prima: se spegniamo via software la scheda, il controller prenderà alimentazione dalla batteria LiPo, che quindi tenderà a scaricarsi; è compito del software fare in modo che questo venga posto in standby o comunque in una delle modalità a basso consumo disponibili.

Passiamo adesso alla sezione dello stadio switching, che fa capo agli integrati U2 (SX1308), un convertitore asincrono step-up, utilizzato qui in modalità SEPIC, in grado di fornire in uscita 5V con tensioni in ingresso variabili da circa 3, fino ad oltre 20 volt, ed all'integrato U3, che

è un convertitore sincrono step-down, utilizzato per ricavare i 3,3 volt per l'alimentazione della maggior parte del circuito. Entrambi questi stadi sono già stati ampiamente descritti per le schede Fishino Mega e Guppy e, per quanto riguarda il funzionamento del SEPIC, nella scheda Torpedo, pubblicata sul numero 204 di Elettronica In. Le uniche differenze consistono nella presenza del jumper ON, che permette l'inserimento di un interruttore di spegnimento in grado di togliere l'alimentazione a tutta la scheda (fatto salvo quella di mantenimento eventualmente abilitata, come spiegato nel paragrafo precedente), e la linea OFF, già vista prima, in grado di spegnere, se portata a massa, il convertitore switching. Questa linea è sotto diretto controllo del processore, attraverso un I/O, e permette quindi sia lo spegnimento completo della scheda via software, sia l'isolamento della sezione di alimentazione del solo controller dal resto del circuito, come visto nel punto precedente.

Riassumendo, ponendo la linea OFF a massa viene tolta tutta l'alimentazione alla scheda; ovviamente questa linea deve essere mantenuta a massa dal controller, che quindi deve rimanere operativo attraverso una delle due sorgenti di mantenimento (LiPo o pila al litio); in assenza di queste, portando la linea OFF a massa otterremo solo un reset della scheda ed il conseguente riavvio del tutto.

Quanto al caricabatteria, fa capo all'onnipresente MCP73831, un completo regolatore di carica integrato per LiPo a singola cella;



è lo stesso della Fishino Mega, quindi rimandiamo al relativo articolo per i dettagli. L'unico punto degno di nota è costituito dal MOSFET Q9, un canale N, utilizzato per abilitare o meno la funzione di carica della batteria. Se ricordiamo la descrizione del primo stadio (controllo alimentazioni), la linea BATOFF è a livello alto quando è presente l'alimentazione esterna; in questo caso il MOSFET Q9 va in conduzione, portando a massa la resistenza R15 (ed eventualmente la R16 se abbiamo selezionato la carica a 500 mA) attivando quindi la carica. In assenza di tensioni esterne (alimentazione a batteria) la linea BATOFF va a livello basso, disattivando la carica.

Notate le resistenze R17 ed R18, che costituiscono un partitore di tensione in grado di fornire metà della tensione della batteria LiPo, misurabile attraverso una linea del controller per poter segnalare lo stato di carica della stessa. Abbiamo utilizzato un partitore perché la tensione della batteria LiPo, a piena carica, supera i 3,3 V accettati dagli ingressi analogici del controller; occorre tenerne quindi conto nella misura.

STADIO CONTROLLER

Arriviamo finalmente al PIC32MX470F512, del quale al momento ci limiteremo ad

esaminare alcune connessioni, lasciando il resto agli altri stadi per i dettagli. Notiamo innanzitutto due quarzi, uno da 20 MHz e l'altro da 32.768 kHz; servono perché il primo fornisce il clock principale, attraverso un PLL interno, che lo moltiplica fino a 120 MHz. Si tratta di un circuito non proprio a basso consumo, quindi da evitare nelle condizioni di risparmio energetico; per contro, il quarzo da 32.768 kHz serve esclusivamente per il real time controller (RTC) interno alla PIC, che è un circuito studiato appositamente per avere un bassissimo assorbimento e che può quindi essere lasciato in funzione anche in condizioni di standby.

Tornando al primo quarzo, accettando una precisione minore sulla frequenza questo avrebbe potuto essere rimosso; la PIC ha infatti un oscillatore RC piuttosto preciso al suo interno in grado di farne quasi completamente le veci. "Quasi", perché l'oscillatore interno ha due difetti:

- non ha la precisione del quarzo (questo ha un'importanza relativa);
- non è in grado di portare la PIC alla sua massima frequenza di 120 MHz, a causa di vincoli del PLL interno, che sarebbe quindi limitata a 96 MHz.

Il primo punto ha importanza solo se utilizziamo la scheda per misure precise di tempi brevi, ad esempio come frequenzimetro o periodometro; per i tempi lunghi basterebbe l'altro quarzo.

Il secondo punto è vincolante se vogliamo "spremere" il controller al massimo delle sue possibilità, e quindi l'abbiamo inserito!

Si notino, tra le svariate connessioni sul controller, le tre linee viste in precedenza, ovvero la OFF, che gli permette di togliere

l'alimentazione principale, la PWRGOOD, che consente il controllo dello stato della medesima, e la linea MBAT che permette la misura della tensione della batteria.

Diamo adesso uno sguardo alla sezione reset, interessante perché ci ha permesso di utilizzare un solo pulsante di reset per svariate funzioni, in abbinamento con un apposito bootloader sul controller. Notiamo subito tre linee che fanno capo allo stadio di reset:

- MCLR;
- REMOTERESET;
- LONGRESET.

La linea MCLR è quella di reset vero e proprio del controller; portandola a massa il medesimo viene riavviato.

Notiamo subito che questo può avvenire sia premendo il pulsante di reset, che attraverso il condensatore C22 invia un breve impulso negativo alla linea MCLR, sia portando a massa la linea REMOTERESET, proveniente dal modulo WiFi e che consente quindi a questo di resettare il controller.

La linea LONGRESET, per contro, viene a trovarsi a massa per tutto il tempo in cui il pulsante di reset viene mantenuto premuto, consentendo quindi al bootloader di conoscere le nostre intenzioni.

In sintesi:

- una breve pressione del pulsante reset (inferiore ai 2 secondi) o un segnale negativo sulla linea REMOTERESET riavviano la scheda, come d'abitudine;
- mantenendo premuto il pulsante di reset per un tempo tra 2 e 4 secondi viene avviato il bootloader (cosa evidenziata dal lampeggio del led arancione), mettendo quindi la scheda in modalità caricamento sketch;

- mantenendo premuto il pulsante per un tempo superiore ai 4 secondi si avvia la procedura di upgrade del firmware del modulo WiFi, cosa evidenziata dal lampeggio del LED blu.

È comunque possibile, una volta entrati nelle 2 modalità del bootloader, riavviare semplicemente la scheda ed eseguire lo sketch precaricato ripremendo brevemente il pulsante di reset. Per quanto riguarda la sezione led, si tratta semplicemente di due led connessi ad altrettante porte del controller, utilizzabili per qualsiasi compito (indicati sulla scheda come RX/PRG e TX, rispettivamente) e del solito LED connesso alla porta 13 che viene utilizzato nei test della scheda (programma BLINK), questo attraverso il MOSFET Q9 per non caricare l'uscita D13, e che tra l'altro riflette lo stato della linea di clock (SCK) dell'interfaccia SPI, cosa che si nota utilizzando la scheda microSD o il modulo WiFi.

Passiamo adesso alla sezione WiFi, simile a quella delle Fishino precedenti, perché il modulo ha sempre un firmware da noi realizzato, in grado di comunicare ad alta velocità tramite la porta SPI e non la porta seriale.

In questa scheda è stato eliminato il connettore ESPCONN al quale eravamo abituati nelle board precedenti, ed al quale facevano capo alcuni I/O e pin di controllo del modulo sfruttati sia per programmarlo che, per alcuni di essi, come I/O aggiuntivi. Questo perché nel Fishino32 abbiamo già liberato alcuni degli I/O che venivano usati per comunicare con il modulo e con la scheda SD, che erano il 4, 7 e 10; abbiamo inoltre separato il bus I²C dai due ingressi analogici A4 e A5, ottenendo quindi 3 porte

digitali e due porte analogiche in più a disposizione.

Non solo, ma due porte sul connettore ICSP sono sfruttabili come porte digitali aggiuntive (ne parleremo in dettaglio descrivendo il software), quindi abbiamo preferito sfruttare le connessioni con il modulo ESP in altro modo. Infatti, tutte le linee dell'ESP (salvo l'ingresso analogico) sono ora sotto controllo del processore principale, cosa che ci permette di effettuare le seguenti operazioni direttamente da software, senza impegnare alcun I/O:

- resettare il modulo da software;
- spegnere il modulo da software (cosa che in precedenza richiedeva l'uso di un I/O del Fishino);
- caricare il firmware sul modulo senza dover effettuare collegamenti strani né caricare sketch particolari;
- ridirigere l'output seriale del modulo sulla seriale USB o su qualsiasi altra seriale, sia software che hardware, disponibile sul Fishino.

Per il resto, nulla cambia rispetto alle schede precedenti; la comunicazione avviene attraverso la porta SPI, che fa capo anche alla scheda SD ed ai pin digitali D11, D12 e D13 sul connettore esterno. Visto che la PIC utilizzata ha due interfacce SPI incorporate, verrebbe da chiedersi perché non è stata dedicata una di queste ai moduli inclusi (WiFi e SD) liberando così anche gli I/O 11, 12 e 13 completamente; questo è stato fatto perché la seconda interfaccia SPI è dedicata, in modalità I²S, al codec audio che quando è in funzione la impegna a tempo pieno. In un primo tempo avevamo scelto quella strada, salvo accorgerci in seguito che risultava impossibile, ad esempio,

Poiché una trattazione completa di tutte le librerie disponibili richiederebbe troppo spazio, la rimandiamo ad articoli successivi; al momento vi proponiamo uno sketch dimostrativo, ma perfettamente funzionante e piuttosto piacevole, di un audio player.

Per poterlo utilizzare è necessario seguire i seguenti passi :

- Preparare una scheda microSD con al suo interno alcuni files in formato .wav. Attualmente il player non supporta il formato .mp3 (in preparazione!), quindi eventuali .mp3 vanno convertiti in .wav utilizzando, ad esempio, il programma open source Audacity, reperibile in rete. Nella conversione occorre scegliere il formato Microsoft WAV, contenente audio PCM non compresso, indifferentemente a 8, 16, 24 o 32 bit. Per un'ottima qualità consigliamo 16 o ancora meglio 24 bit.
- Una volta preparata la scheda con i file musicali, inseritela nel lettore sul Fishino, collegate una coppia di altoparlanti alle uscite L+ e L- (altoparlante sinistro) e R+ ed R- (altoparlante destro) sul connettore AUDIO, caricate lo sketch *FishinoPlayerDemo* e... godetevi un po' di musica!

Il player esegue tutti i brani in formato .wav presenti sulla scheda in sequenza, ripetendoli all'infinito.

Una nota: l'amplificatore audio sulla scheda dispone di una potenza non trascurabile (ovviamente rapportata alle dimensioni della scheda!), che può arrivare a 2,3W per canale. Questa potenza è eccessiva per l'alimentazione tramite USB, e rischia di resettare la scheda per insufficienza di corrente. In questo caso, le soluzioni sono tre:

- utilizzare un solo altoparlante;
- ridurre il volume audio;
- utilizzare un'alimentazione esterna, anche tramite la batteria LiPo; quest'ultima può tranquillamente essere usata insieme all'USB e provvederà a fornire i picchi di corrente necessari all'audio quando l'USB non ce la fa.

Un ultimissima ma importante nota: fate molta attenzione alla polarità della batteria LiPo, sulla scheda è indicata chiaramente e, nel caso la vostra batteria abbia una diversa polarità occorre invertire il connettore.

Per motivi di efficienza l'ingresso di alimentazione a batteria non è protetto da inversioni di polarità, quindi collegando la LiPo in marriera errata danneggerete irrimediabilmente la scheda!

riprodurre un file audio letto da scheda SD, visto che durante la riproduzione l'interfaccia era totalmente impegnata con il codec senza possibilità di accedere alla SD. Va detto che nelle schede in formato Arduino UNO i 3 I/O suddetti sono comunque dedicati alla porta SPI, e comunque se necessario abbiamo a disposizione i 2 I/O sulla porta ICSP di cui parlavamo prima, ed eventualmente tutti gli I/O analogici che sono comunque utilizzabili anche in digitale. L'interfaccia verso la microSD è identica a quella delle schede già viste; sfrutta l'interfaccia SPI, come il modulo WiFi, mentre anche qui per la selezione

della scheda viene utilizzato un I/O interno, lasciando libero quindi l'I/O D4 che prima era necessario per questo scopo. L'unica grossa differenza rispetto alle schede precedenti è l'assenza degli stadi di conversione livelli (da 5 a 3,3 V) visto che il PIC utilizza i 3,3 V come alimentazione, rendendo quindi possibile (e più veloce!) un collegamento diretto. Unico accorgimento da tenere, nell'esecuzione di sketch esistenti: il pin di selezione scheda (CS) non è più il 4 (o il 10 in alcune versioni di Arduino) ma un pin interno denominato SDCS; è sufficiente quindi sostituire tutti i riferimenti all'I/O 4 con l'I/O

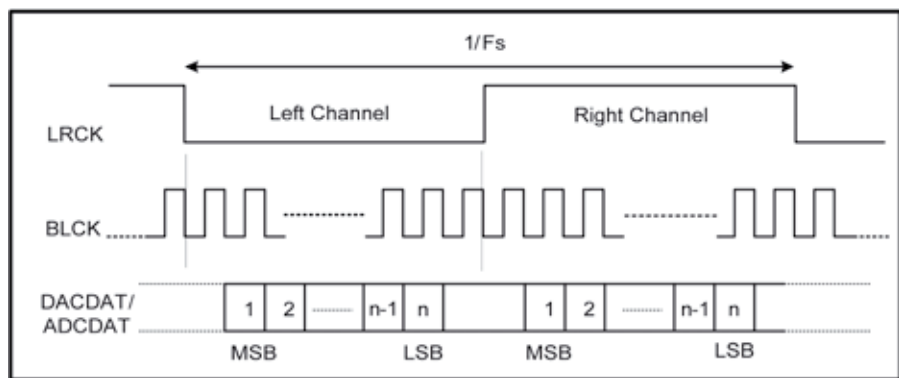


Fig. 1 - Segnali dell'I2S correlati.

SDCS (scritto proprio così, visto che può cambiare fisicamente da scheda a scheda).

STADIO CODEC AUDIO

Ed eccoci arrivati alla succosa novità presente nel Fishino32: il codec audio! Si basa su un chip, l'ALC5631Q, utilizzato in molti prodotti consumer (tablet, riproduttori audio, eccetera) e quindi dotato di prestazioni notevolissime pur conservando un prezzo più che accettabile. Il codec ha le seguenti caratteristiche:

- un convertitore D/A (DAC) stereo a 24 bit con un rapporto segnale/disturbo di 100 dB ed una distorsione totale a -90 dB;
- un convertitore A/D (ADC) stereo a 24 bit con un rapporto segnale/disturbo di 93 dB ed una distorsione totale a -88 dB;
- un amplificatore stereo in classe D, in grado di erogare 650 mW per canale se alimentato a 3,3 V su un'impedenza di 8 Ohm, e fino a 2,8 W per canale se alimentato (come nel nostro caso) a 5 volt e su un'impedenza di 4 ohm;
- un amplificatore per cuffie stereo, in grado di erogare 45 mW su 16 ohm, che purtroppo non utilizzeremo nella nostra scheda per motivi di spazio;
- 3 ingressi analogici differenziali o un ingresso analogico stereo (noi utilizziamo quest'ultimo);

- due ingressi microfonici analogici preamplificati, sia in modalità differenziale (da noi non usata) sia in modalità single-ended;
- un ingresso per microfono stereo digitale (anche questo, non utilizzato sulla nostra scheda sempre per motivi di spazio);
- un'uscita analogica di linea, alla quale è possibile connettere un amplificatore esterno;
- un completo sistema di power management che permette di spegnere i singoli componenti del codec se non utilizzati;
- un equalizzatore digitale a 6 bande audio;
- effetti di pseudo-stereo e di stereo spaziale;
- circuiti di rimozione dei disturbi di accensione e spegnimento negli altoparlanti;
- gestione automatica di livelli (ALC) sia in riproduzione che in registrazione;
- interfaccia standard I²S ad alta velocità (24 bit/da 8 a 192 kHz sul DAC in riproduzione e 24 bit/da 8 a 96 kHz sull'ADC in registrazione).

Come si può notare, una serie di caratteristiche impressionanti, che permetteranno di realizzare, senza alcun hardware aggiuntivo che non sia un altoparlante o un microfono, moltissimi progetti che coinvolgono l'audio digitale; per esempio:

- player audio (WAV e MP3);
- registratore audio;
- riconoscimento vocale;
- sintesi musicale;
- sintesi vocale.

Come vedrete in seguito, abbiamo già implementato un player per files .wav, ed è in realizzazione un completo lettore MP3; per la registrazione e le altre caratteristiche scriveremo le opportune librerie software in seguito.

Come già accennato parlando del modulo WiFi, il codec sfrutta un'interfaccia SPI dedicata, utilizzata in modalità I²S, per la gestione della quale abbiamo scritto un'apposita libreria software.

Il codec ha inoltre bisogno di una porta I2C (non necessariamente dedicata solo a lui, ed infatti abbiamo utilizzato la porta I2C standard del Fishino) per controllare tutte le varie funzioni disponibili. Anche qui abbiamo creato un'apposita libreria in grado di controllare il funzionamento del codec; tramite questa si può, per esempio, decidere se registrare o riprodurre, modificare il volume audio, regolare l'equalizzatore, impostare l'ALC, eccetera.

Come potrete ben immaginare i controlli sono numerosissimi, quindi la libreria verrà estesa man mano che se ne presenterà la necessità.

Sullo schema elettrico c'è pochissimo da aggiungere; come potete vedere, oltre al codec, qualche condensatore di disaccoppiamento sulle alimentazioni e sugli I/O, un condensatore per un convertitore interno destinato ad aumentare la tensione per l'amplificatore delle cuffie (C38) ed un paio di resistenze per la tensione di bias (polarizzazione) dei microfoni, non c'è altro!

Dicevamo dell'interfaccia I2S: questa è un'interfaccia basata sull'I2S prevista espressamente per

la trasmissione di flussi audio stereo ad alta velocità; a differenza della SPI, ed a causa del fatto che i segnali che vi transitano devono essere in tempo reale, è praticamente impossibile utilizzarla in contemporanea su più apparecchiature.

L'interfaccia utilizza gli usuali pin SDI (MISO), SDO (MOSI) e SCK (clock) per i dati, mentre il pin di selezione viene utilizzato in modo differente per selezionare il canale audio tra i 2 trasmettitori; è inoltre presente una linea addizionale di clock usata come riferimento.

Quindi abbiamo, nominando i pin dal lato master, ovvero dal punto di vista del controller:

- MISO o SDI > ADCDAT, ovvero i dati in ingresso dal convertitore A/D al controller;
- MOSI o SDO > DACDAT, ovvero i dati in uscita dal convertitore D/A verso la PIC;
- SCK > BLCK, ovvero il clock per i singoli bit trasmessi;
- SS > LRCK, ovvero il clock per differenziare canale sinistro e destro.
- MCLK > Clock di riferimento, abitualmente quattro volte l'SCK.

I segnali che transitano sono visibili nella **Fig. 1**, dove manca il clock di riferimento (MCLK) che è comunque un clock fisso di frequenza quadrupla rispetto al BLCK.

Lasciamo per il momento la descrizione del codec, che riprenderemo quando parleremo delle librerie software.

CONNETTORI D'ESPANSIONE

In questa sezione sono raggruppati tutti i connettori che portano i segnali verso l'esterno della scheda, quindi gli header laterali, il connettore ISP (che su Fishino32 non è utilizzato per pro-

grammare la scheda, ma inserito per compatibilità con gli shield esistenti), il connettore ICSP (In Circuit Serial Programming) che serve per programmare la scheda senza passare per il bootloader (ad esempio per programmare il bootloader stesso), e sul quale sono disponibili due I/O digitali aggiuntivi, ed il connettore USB. Come accennato parlando delle caratteristiche della scheda, la PIC ha molti, ma non tutti, i pin I/O 5V tolerant, ovvero in grado di accettare 5V in ingresso anche se il controller è alimentato a 3,3 V senza danneggiarsi. Dove possibile abbiamo sfruttato quei pin portandoli all'esterno come I/O digitali; purtroppo in alcuni casi questo non è stato possibile, ed abbiamo quindi inserito una protezione in ingresso tramite resistenze e diodi; è questo il caso dei segnali sulla porta SPI (I/O da 11 a 13) che, essendo necessari anche ai moduli interni, avremmo dovuto limitare comunque. La protezione consiste in una resistenza da 220 Ohm in serie all'ingresso ed un diodo in grado di scaricare l'eccedenza sulla linea a 3,3 V, limitando quindi la tensione in ingresso a tale valore. Sui pin analogici, che non sono 5V-tolerant (proprio a causa della circuiteria analogica interna al controller) abbiamo inserito solamente delle resistenze in serie per limitare un eventuale sovracorrente nel caso vi si inseriscano per errore dei valori superiori ai 3,3 V.

Attenzione, questo non significa che è possibile inserire 5 volt su quel connettore! Le resistenze offrono un minimo grado di protezione, che dovrebbe proteggere il controller per brevi periodi se per errore vi si inseriscono i 5 Volt, ma non bisogna farne conto ed utilizzarli stabilmente con quei valori di tensione!

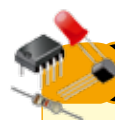
L'integrato dispone infatti di diodi di protezione al suo interno, che fanno le veci di quelli da noi inseriti esternamente sugli ingressi D11..D13; questi diodi sono però in grado di assorbire una corrente molto limitata e per brevi periodi di tempo, quindi non ci si deve fare assolutamente conto.

Nella sezione header trovate anche un connettore denominato REFSEL, che serve per selezionare la tensione che verrà connessa al pin VREF; è infatti possibile farvi giungere i 5V, nel caso si utilizzino shield "normali" funzionanti a quella tensione, oppure i 3,3 V destinati a shield a bassa tensione come quelli utilizzati nell'Arduino DUE.

CONCLUSIONI

Bene, con questo abbiamo concluso, almeno per questa prima puntata.

Termineremo il discorso nella prossima, affrontando la realizzazione della nostra Fishino 32 e la messa in esercizio, scoprendo e utilizzando i tool di Arduino e librerie specifiche importabili grazie alle funzioni di acquisizione di hardware di terze parti implementate nelle più recenti versioni dell'IDE.

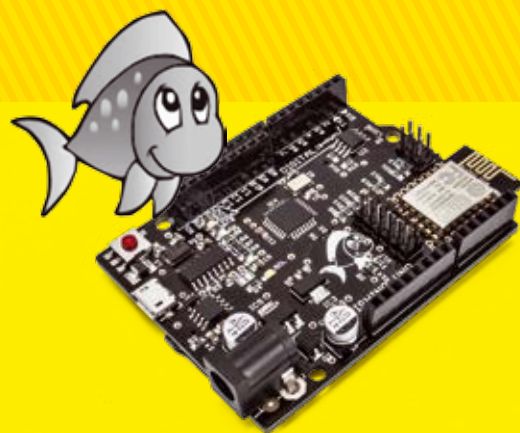


per il MATERIALE

La board Fishino32 (cod. FISHINO32) viene fornita montata e collaudata. Può essere acquistata presso Futura Elettronica al prezzo di Euro 59,00. Il prezzo si intende 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>

Fishino Contest: I PROGETTI VINCITORI



“FISHINO JOYPAD”

di GASPARE SANTAERA & MATTIA POGGIANI

Costruiamo un joypad wireless per RetroPie,
il nostro emulatore di videogiochi
arcade basato su Raspberry Pi.

Nel fascicolo n° 200 di Elettronica In abbiamo presentato il progetto RetroPie, un vero amarcord per chi ha vissuto l'era delle prime console giochi domestiche e da bar. Presentato a maker faire di Roma lo scorso anno, ha ispirato i due autori della soluzione premiata al secondo posto nel Fishino Contest, che hanno voluto aggiungere qualcosa al progetto originale, sfruttando le potenzialità di un microcontrollore unite all'espandibilità WiFi che caratterizza Fishino, realizzando così un joypad wireless.

In questo articolo, quindi, vedremo come realizzare un joypad per la Raspberry Pi; non si tratterà soltanto del classico joypad (modello super Nintendo) ma oltre ad offrire le connettivi-

tà wireless il device presentato potrebbe essere la base di futuri progetti più complessi capaci di espandere e migliorare l'esperienza di gioco, usando per esempio sensori di forza, accelerometri, sensori ad infrarossi ecc. Nel nostro caso ci occuperemo dell'interfaccia classica, pertanto Fishino sarà incaricato di gestire dieci tasti (freccie direzionali, start, select, A, B, X, Y), un piccolo display Nokia che riporta alcune informazioni di sistema e la ricarica della batteria LiPo con cui verrà alimentato l'insieme. Il joypad (tutti i componenti meccanici, s'intende...) è stato interamente stampato in 3D e alla pagina web del progetto (<http://blog.elettronica.in/2016/04/15/fishino-joypad-by-gaspere-santaera-mattia-poggiani/>) trovate i file

per la stampa. Nello specifico, ci siamo occupati di dimensionare il case esterno (composto da un coperchio e un fondo) e di creare i relativi tasti, che sono cinque pezzi diversi tra loro: uno per le frecce, uno per i tasti di start-select, due per i tasti funzione ed uno per il tasto di accensione. Lasciamo dunque la parola agli autori.

SCHEMA ELETTRICO

L'hardware del nostro progetto è centrato su Fishino e su una scheda compatibile con i pin di espansione di Arduino, che si occupa della connessione tra un display LCD Nokia e Fishino, di acquisire lo stato dei pulsanti presenti nel joypad e di gestire la ricarica della batteria. Per il visualizzatore abbiamo deciso di riutilizzare un display LCD Nokia in dotazione al modello 5110, in bianco e nero, caratterizzato da una risoluzione di 84x48 pixel. In questo caso lo shield si occupa solo di creare le connessioni fisiche tra Fishino e il display, che si interfaccia facilmente con un Arduino/Fishino tramite un connettore a otto poli (siglato SV1), mentre per la sua gestione basta usare l'apposita libreria, scaricabile dal web alla pagina <http://playground.arduino.cc/Code/PCD8544>, nella quale troverete tutte le informazioni riguardanti le connessioni, nonché la creazione e la gestione dei grafici sul display. Come accennato, la presenza di un display ci permette di avere un riscontro visivo sulle operazioni compiute (ad esempio indicare i tasti premuti) e di ottenere informazioni sul sistema (ad esempio livello di carica della batteria, indirizzo IP della Raspberry Pi, ecc.); vedremo nel dettaglio questi aspetti quando tratteremo la parte relativa al firmware. Per il momento torniamo sull'hardware, passando alla gestione dei tasti: nel nostro progetto abbiamo deciso di emulare il classico joypad del Super Nintendo che contiene dieci tasti; quattro frecce per le direzioni, un tasto di start, uno di select e poi quattro tasti di gioco, la cui funzione varia al variare del gioco. Per la realizzazione dei tasti, ovviamente abbiamo utilizzato dei classici pulsanti normalmente aperti, per la cui gestione ci siamo affidati a cinque ingressi analogici di Fishino e a due ingressi digitali. Per il controllo dello stato dei tasti, la soluzione ideale

sarebbe stata l'utilizzo di dieci ingressi digitali, in modo da sfruttare al massimo la velocità di commutazione di quest'ultimi e di evitare i tempi di conversione dell'ADC presente sugli ingressi analogici. Tuttavia la mancanza di dieci pin liberi ci ha spinto ad usare gli ingressi analogici ed una combinazione di partitori resistivi in modo da ridurre quanto più possibile l'utilizzo dei pin.

Infatti, come accennato, nel nostro progetto usiamo soltanto sette pin totali, a fronte dei dieci tasti, ovvero vengono usati quattro ingressi analogici (A1, A2, A3, A4) per la gestione dei tasti di gioco, due ingressi digitali (D8, D9) per i tasti Start e Select, ed un solo ingresso analogico (A0) per la gestione dei tasti freccia. Quest'ultimi vengono discriminati mediante sedici differenti partitori resistivi, formati da un'unica resistenza (R5) del valore di 10 kohm collegata tra Vcc e il pin A0 e quattro diverse resistenze, ognuna delle quali ha un valore di circa il doppio rispetto alla precedente (1 - 2 - 4,3 e 9,1 kohm) connesse in serie tra i tasti e massa

(GND). In questo modo ad ognuna delle possibili combinazioni di tasti (16 in tutto), corrisponde un partitore, ovvero un solo valore di tensione equivalente, il che permette di fare una mappatura di tutte le combinazioni e poi tramite uno *switch* all'interno del firmware riuscire a capire il tasto/i premuti. Ad esempio, il pulsante S1 (tasto freccia su) è connesso alla resistenza R4 (2 kohm), mentre il pulsante S2 (tasto freccia sinistra) è connesso alla resistenza R1 di 1 kohm, pertanto se si preme S1, il partitore fornisce una tensione pari a:

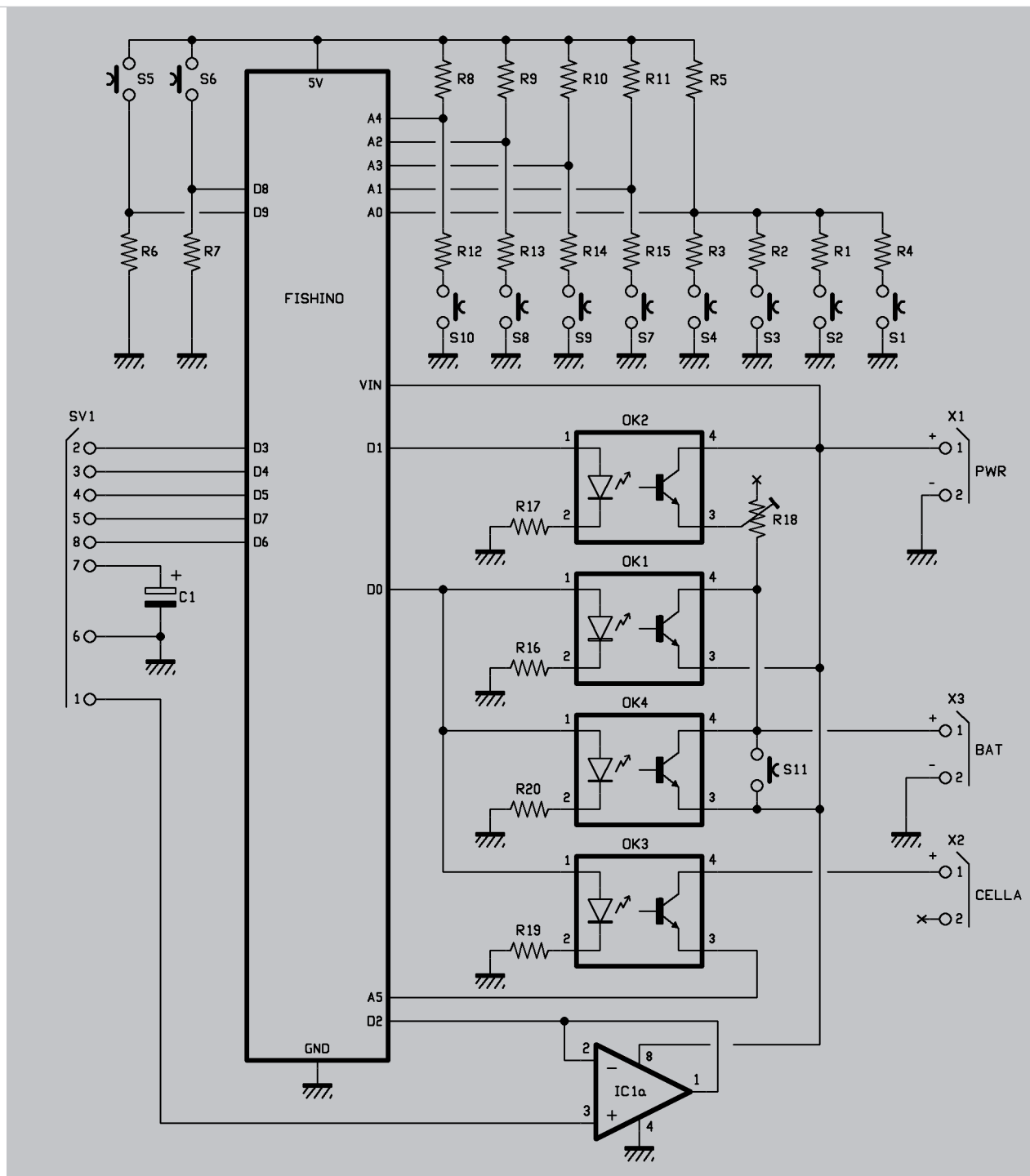
$$(2/12) \cdot 5V = 0,83 V;$$



Fig. 1
Il modello dei Joypad
riprodotto dal nostro progetto.



*Gli autori del progetto,
i Sig.ri Gaspere Santaera
e Mattia Poggiani.*



Invece se si premono contemporaneamente S1 ed S2, sul partitore ci sarà una tensione pari a:

$$((2//1)/(2//1+10))*5V = 0,31V$$

e così via nei restanti quattordici casi. L'ultimo compito della scheda è gestire l'alimentazione: il joypad viene alimentato tramite una batteria LiPo a due celle (7,4 V). Il joypad viene acceso e spento tramite due strategie diverse: per

l'accensione viene utilizzato un apposito tasto (S11), che quando premuto connette direttamente il polo positivo della batteria con il pin Vin di Fishino, dando così alimentazione alla scheda. A questo punto entra in gioco il firmware; infatti, appena riceve l'alimentazione dalla scheda, Fishino si avvia immediatamente abilitando tre distinti optoisolatori. OK1 e OK4 connettono il pin Vin di Fishino al polo positivo della batteria, mentre il terzo (OK3) connette il pin A5 di Fishino al pin che si trova tra

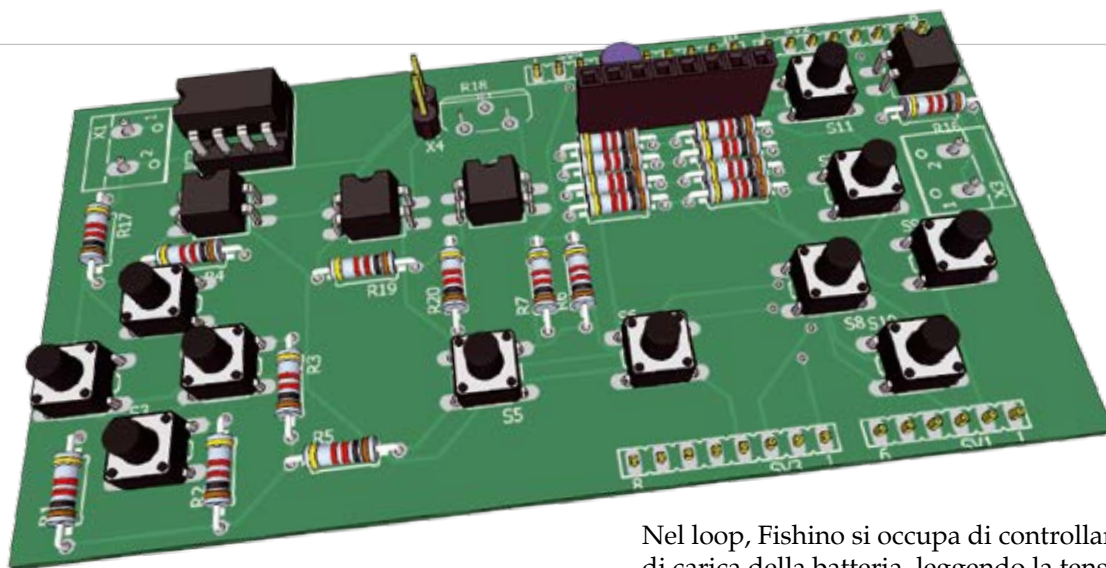


Fig. 2
Rendering
dello shield
per il joystick
ottenuto
da Eagle.

la serie delle due celle che compongono la batteria. Successivamente Fishino effettua una lettura su A5 (che va connesso tra le due celle LiPo utilizzando il connettore per il bilanciamento, che fornisce la tensione di una cella): se trova una tensione superiore ai 3,6V la batteria è carica e Fishino provvede a lasciare accesi i due optoisolatori connessi alla batteria, che fungono da interruttori digitali, attivandosi e fornendo a Fishino la tensione della batteria. La lettura di A5 rientra nella strategia per preservare la batteria, che prevede lo spegnimento automatico se essa si sta scaricando: quando viene acceso a batteria, lo stesso D0 che comanda OK1 e OK4 attiva anche OK3, che porta la tensione del contatto CEL (ridotta della caduta Vce del transistor interno) all'ADC di Fishino (assegnato ad A5); se la tensione letta su CELLA è inferiore a 3,6 V Fishino considera la batteria scarica e porta D0 a 0 logico, lasciando interdire i fotoaccoppiatori.

Per lo spegnimento manuale del joystick abbiamo previsto una procedura totalmente affidata al firmware: in caso di pressione contemporanea dei quattro tasti freccia, Fishino disattiva i tre optoisolatori suaccennati, scollegandosi dall'alimentazione. Diverse sono le considerazioni nel caso in cui si voglia ricaricare la batteria: partendo dall'alimentazione esterna, infatti, abbiamo escluso la possibilità di accendere o spegnere il joystick, che resta sempre acceso in modo da permettere a Fishino di tenere sempre sotto controllo la batteria. L'alimentazione esterna è collegata direttamente al pin Vin di Fishino. Tramite un amplificatore operazionale (IC2a) in configurazione buffer, la cui uscita è connessa al pin D2, Fishino capisce se viene alimentato dalla batteria o da un'alimentazione esterna e può, se necessario, spegnere i tre optoisolatori precedentemente descritti, al fine di bloccare la corrente tra la batteria e l'alimentazione esterna, rimanendo comunque acceso perché alimentato da una sorgente esterna.

Nel loop, Fishino si occupa di controllare lo stato di carica della batteria, leggendo la tensione sul pin A5 e nel caso in cui fosse necessario abilita l'optoisolatore OK2, collegando l'alimentazione esterna alla batteria al fine di ricaricarla; in tal caso quando su A5 legge una tensione superiore ad una certa soglia ritiene la batteria carica e torna ad isolarla dal resto del circuito spegnendo OK2.

Per quanto riguarda la carica della batteria, che è una LiPo, ci sembra doveroso dire che tale tipo di accumulatore se non viene "trattato bene" può diventare pericoloso e nel peggiore dei casi può esplodere. La strategia migliore per ricaricare una LiPo evitando di danneggiarla è quella di fornirle una corrente di ricarica che decresce man-mano che la batteria si ricarica e quando la corrente scende al di sotto del 5% della capacità nominale della batteria, concludere la ricarica. Normalmente a tale scopo viene utilizzata una resistenza posta in serie alla batteria, ai capi della quale viene misurata la caduta di tensione in modo da risalire all'attuale corrente di ricarica.

In questo progetto, viste le basse correnti in gioco abbiamo deciso di adottare una strategia di ricarica



Fig. 3 - Il joystick realizzato dagli autori.

leggermente differente, introducendo una resistenza tra l'optoisolatore e la batteria che funziona come una sorta di limitatore di corrente. In questo modo avremo una corrente di ricarica molto bassa e quindi un tempo di ricarica un po' più lungo, però potremo controllare lo stato della batteria direttamente sul pin A5, oltre a correre meno rischi durante la ricarica e ad aumentare la vita media della batteria. Nello schema elettrico tale resistenza (R18) è un trimmer del valore massimo di 100 ohm; abbiamo utilizzato un trimmer in modo da permettere una taratura più fine delle correnti desiderate oltre a compensare un po' la resistenza equivalente dell'optoisolatore. Nel nostro caso, ad esempio, abbiamo utilizzato una batteria di 240 mAh e volendo fissare una corrente massima di ricarica di 70 mA abbiamo utilizzato una resistenza di circa 50 ohm; considerando una resistenza interna dell'optoisolatore di circa 16 ohm, abbiamo pertanto impostato il trimmer a circa 30 ohm.

IL FIRMWARE

Come detto, Fishino si occupa della gestione della rete, dei tasti, del display e della batteria, pertanto nel firmware troveremo tutte le librerie e le variabili utili a questo scopo. Nel *setup*, una volta avviato,

Fishino si occupa di fare tutti controlli sulle alimentazioni e nel caso in cui la batteria è carica o è presente un'alimentazione esterna, connette Fishino alla rete, inizializzando, infine, il display. Successivamente Fishino entra nel loop, dove di volta in volta vengono dapprima effettuati tutti i controlli sulle alimentazioni, poi i controlli sui tasti ed infine inviati, ove necessario, i dati alla Raspberry Pi e stampate le informazioni sul display. Per quanto riguarda il controllo delle alimentazioni, Fishino controlla la presenza di alimentazioni esterne (*value_ext*), poi lo stato della batteria (*value_bat*) e di conseguenza decide se spegnere o meno il dispositivo, se è necessario ricaricare la batteria o interrompere l'eventuale ricarica in corso di quest'ultima, tramite opportuni flag (*flag_ext*, *flag_bat*). La procedura di check per la verifica dei tasti premuti è molto semplice: di volta in volta Fishino controlla i pin connessi ai tasti fisici e nel caso di pressione di quest'ultimi assegna ad una variabile (*j_key*) dei valori in funzione del tasto premuto, mentre con un'altra variabile (*key_num*) conta quanti tasti sono stati premuti. Successivamente invia alla Raspberry Pi dei pacchetti di dati, uno per ogni tasto, in accordo alla convenzione stabilita all'interno del *Fishino server* che gira su Raspberry Pi.

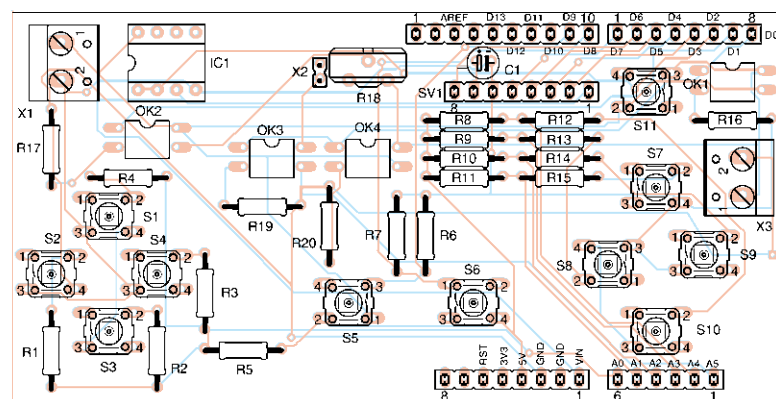
[piano di MONTAGGIO]

Elenco Componenti:

C1: 10 μ F 63 VL elettrolitico
 R1: 1 kohm
 R2: 9,1 kohm
 R3: 4,3 kohm
 R4: 2 kohm
 R5: 10 kohm
 R6÷R11: 1 kohm
 R12÷R15: 2 kohm
 R16: 50 ohm
 R17: 50 ohm
 R18: 100 ohm
 R19: 50 ohm
 R20: 50 ohm
 OK1÷OK4: SFH615
 S1÷S11: Microswitch
 IC1: AD620

Varie:

- Morsetto 2 poli (2 pz.)
- Strip Maschio/Femmina 6 vie



- Strip Maschio/Femmina 8 vie (2 pz.)
- Strip Maschio/Femmina 10 vie
- Strip Maschio 8 vie

- Strip Maschio 10 vie
- Strip Maschio 2 vie
- Zoccolo 4+4
- Circuito stampato (100 x 50 mm)

Listato 1

```
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons( 8888 );
int yes = 1;
setsockopt(socket_desc, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));
setsockopt(socket_desc, SOL_SOCKET, SO_REUSEPORT, &yes, sizeof(int));
bind(socket_desc, (struct sockaddr *)&server, sizeof(server)) < 0);
```

Infine Fishino gestisce il display, indicando su quest'ultimo il livello di carica della batteria e quali tasti sono premuti al momento dall'utente.

SOFTWARE

Oltre agli aspetti hardware, è stato necessario implementare un software scritto in C in Raspberry Pi, che va eseguito in background, parallelamente alla console Retropie, al fine di far comunicare Fishino con la Raspberry Pi. Il programma non è nient'altro che un mini-server che riceve gli eventi legati alla pressione di un tasto sul joypad e li reindirizza verso la console. Il concetto base del server è quello di stabilire un collegamento wireless (tramite la connessione ad una rete comune) con il joypad e di replicare sulla console gli stessi eventi software che vengono normalmente generati quando si preme un qualsiasi tasto su una tastiera connessa alla Raspberry Pi, come se si disponesse di una tastiera virtuale wireless. Riassumendo, i passaggi principali e le funzioni chiave del software sono quelli di seguito descritti.

1. Viene creato un socket, ovvero una porta per una connessione verso l'esterno, per una comunicazione dati di tipo UDP, meno affidabile di una connessione TCP dal punto di vista della perdita dei pacchetti di dati, ma più veloce e più indicata per applicazioni di questo genere, nelle quali è richiesta una prontezza di risposta elevata. *Server* e *Client* sono due strutture dati di tipo *sockaddr_in* definite nelle librerie standard di UNIX:

```
int socket_desc = socket(AF_INET, SOCK_DGRAM, 0);
struct sockaddr_in server, client;
```

2. In seguito alla creazione del socket, è necessario aprire una porta software (noi abbiamo scelto convenzionalmente la 8888), sulla quale arriveranno i pacchetti dati dal joypad, quindi fare il binding di questa sul socket; le relative istruzioni sono quelle visibili nel **Listato 1**.

3. A questo punto, il server si deve mettere in attesa di una connessione da parte del joypad e nel momento in cui gli arriva una richiesta, la deve accettare e creare un nuovo dispositivo di tastiera virtuale (**Listato 2**).

4. Fino a quando il joypad è connesso, il server resta in attesa di ricevere e gestire tutte le richieste di pressione dei tasti. La ricezione del pacchetto dati è svolta dalla funzione *recv()*, che memorizza il tasto ricevuto nella stringa *client_message*, mentre la funzione *send_key()* riconosce il tasto e genera gli eventi tastiera corrispondenti:

```
while (connected && (read_size = recv(client_sock,
client_message, 2, MSG_WAITALL) > 0)){ send_
key(&client_message);}
```

5. Funzione *setup_uinput_device()*: crea un dispositivo virtuale che, come ogni altro dispositivo sui sistemi UNIX, è identificato da un file di sistema. Anche il joypad sarà legato ad un file ('/dev/uinput' per la nostra applicazione) e diviene accessibile nel software tramite il descrittore di file *uinp_fd*. Dopo la creazione del file, che avviene tramite l'istruzione:

```
uinp_fd = open("/dev/uinput", O_WRONLY | O_NDELAY);
```

vengono associati al joypad un nome e le proprietà di gestione degli eventi di tipo tastiera, in modo che Raspberry Pi possa emulare correttamente un dispositivo di input (**Listato 3**). Tali proprietà vengono scritte sul descrittore e viene istanziato il dispositivo, tramite le istruzioni:

```
write(uinp_fd, &uinp, sizeof(uinp));
ioctl(uinp_fd, UI_DEV_CREATE);
```

6. Funzione *send_key()*: genera gli eventi di input per Retropie e i giochi. Ogni tasto premuto sul joypad Fishino viene mappato in un carattere diverso

Listato 2

```
listen(socket_desc, 1);
int client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t *)&c);
setup_uinput_device();
```

(come descritto sul firmware); a seconda del carattere riconosciuto, il server genera gli eventi di pressione, pressione prolungata e rilascio del tasto, scrivendo sul descrittore di file del dispositivo virtuale creato il tipo di evento, la codifica che identifica ciascun tasto e il valore che identifica l'azione associata. La funzione in questione è:

```
event.type = EV_KEY;
event.code = code;
event.value = 1;
write(uinp_fd, &event,
sizeof(event));
```

La struttura event è di tipo input_event, EV_KEY indica che il tipo di evento è legato ad un tasto della tastiera, il codice è proprio del tasto (es. KEY_A per il tasto 'A') e il campo value identifica l'azione associata (0 – rilascio del tasto, 1 – pressione del tasto, 2 – pressione prolungata).

CONFIGURAZIONE E UTILIZZO

Esaurito anche l'argomento software, riportiamo qui di seguito le istruzioni per compilare ed eseguire il server sul sistema Raspberry Pi ed i passi da seguire per utilizzare il joystick Fishino.

1. Avviare la Raspberry Pi con una tastiera USB e digitare F4 per uscire dalla console.
2. Caricare il file server_fishino.c su una cartella qualsiasi (es.in '/home/pi') e compilare su Raspberry Pi:
 - cd
 - gcc server_fishino.c -o server
3. Aggiungere il programma server all'elenco delle attività da eseguire allo startup del sistema:
 - sudo nano /etc/rc.local
 - aggiungere alla fine del file la seguente riga: sudo /home/pi/server &
 - uscire dall'editor di testo, digitando CTRL+X e Y per salvare le modifiche.
4. Digitare sul terminale 'emulationstation' per far ripartire l'emulatore, dopodiché collegarsi ad una rete wifi a disposizione e prendere nota dell'indirizzo IP assegnato alla Raspberry Pi.
5. Riavviare la Raspberry Pi (ancora F4, poi digitare 'sudo reboot'). Al riavvio il server del joystick partirà in automatico.
6. Configurare il joystick, programmando Fishino con le informazioni relative alla rete WiFi e l'ip acquisito da RetroPie, in modo che questo si connetta al server presente sulla Raspberry Pi.
7. Una volta avviata la console RetroPie (e solo allora) accendere il joystick Fishino, che si conatterà automaticamente alla rete WiFi e quindi al server di RetroPie.

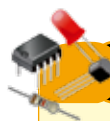
Listato 3

```
strncpy(uinp.name, "Fishino Joystick", UINPUT_MAX_NAME_SIZE);
uinp.id.version = 4;
uinp.id.bustype = BUS_USB;
// Setup the uinput device
ioctl(uinp_fd, UI_SET_EVBIT, EV_MSC);
ioctl(uinp_fd, UI_SET_EVBIT, EV_KEY);
ioctl(uinp_fd, UI_SET_EVBIT, EV_REL);
ioctl(uinp_fd, UI_SET_EVBIT, EV_REP);
for (i=0; i < 256; i++) { ioctl(uinp_fd, UI_SET_KEYBIT, i); }
ioctl(uinp_fd, UI_SET_MSCBIT, MSC_SCAN);
```

8. Al primo utilizzo potrebbe essere necessaria una configurazione dei tasti. È possibile, utilizzando una tastiera reale, avviare l'utilità preposta su RetroPie, navigando sul menu e scegliendo l'opzione 'Configure Input'. Da questo menù è possibile assegnare ogni tasto del joystick.
9. Una volta terminata la configurazione, il joystick è pronto per l'utilizzo e l'utente può quindi utilizzarlo per navigare nei menu e giocare.

CONCLUSIONI

Grazie a Fishino abbiamo aggiunto un prezioso device a un progetto che a nostro parere merita di essere provato e costruito. In questa sede ci siamo occupati di replicare un joystick, ma la potenza e la semplicità d'utilizzo di Fishino permettono di sbizzarrirsi creando joystick via-via più complicati o più performanti per determinati giochi. Ad esempio, si potrebbero inserire nel joystick degli accelerometri e associare i tasti ai movimenti del joystick, dei sensori di forza in modo da calibrare determinati comandi di gioco in funzione della forza che l'utente esercita sul tasto o un microfono, in modo da aggiungere comandi vocali ecc. Insomma l'hardware ha tutte le potenzialità e ora tocca a voi metterci la fantasia! ■



per il MATERIALE

Tutti i componenti necessari per realizzare questo progetto possono essere acquistati presso Futura Elettronica. Il kit RetroPie per videogiochi arcade (cod. RETROPIE_N) è in vendita a Euro 115. La board Fishino UNO (cod. FISHINOUNO) è disponibile a Euro 36,00 e display LCD Nokia 5110 (cod. LCD84X48) costa Euro 14,00. I prezzi si intendono IVA inclusa.

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

FISHINO SALE A QUOTA 32!



di MASSIMO DEL FEDELE

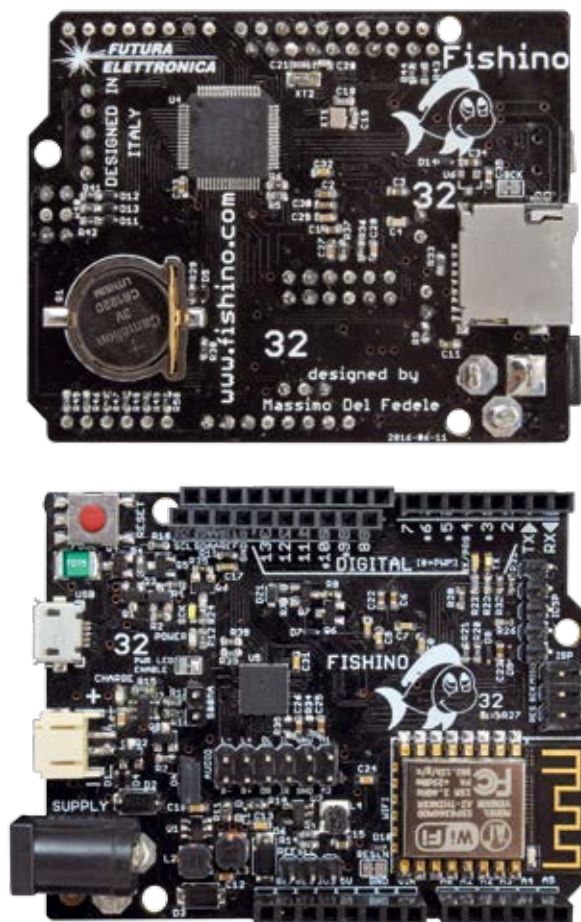
La nostra scheda di prototipazione guadagna il processore a 32 bit e ora punta in alto, verso applicazioni impegnative che possono sfruttare a pieno il supporto WiFi, l'RTC e l'SD-Card che distinguono Fishino dalla comune Arduino UNO. Seconda e ultima puntata.

Nel fascicolo di ottobre avete conosciuto la nostra nuova e potente scheda Arduino-like, che spicca per la disponibilità on-board di transceiver WiFi ed RTC e che ora si distingue ancor più da Arduino per il processore utilizzato; infatti è stata dotata di MCU a 32 bit della Microchip. Nella scorsa puntata vi abbiamo descritto lo schema elettrico rimar-

candone le caratteristiche e i vantaggi rispetto alle nostre Fishino basate su microcontrollore ATmega; ora è giunto il momento di passare alla costruzione e alla messa in esercizio.

REALIZZAZIONE PRATICA

Fishino32 è una scheda realizzata completamente con componenti SMD, alcuni di dimensioni particolarmente



Elenco Componenti:

C1: 100 μ F ceramico (1206)	(0603)
C2: 1 μ F ceramico (0603)	C40: 100 nF ceramico (0603)
C3: 1 μ F ceramico (0603)	D1: RB521S
C4÷C8: 100 nF ceramico (0603)	D2: SS34SMA
C9: 22 μ F 25 VL ceramico (0603)	D3: SS34SMA
C10: 100 nF ceramico (0603)	D4÷D7: RB521S
C11: 4,7 μ F 25 VL ceramico (0603)	D7: SS34SMA
C12: 22 μ F ceramico (0603)	D8÷D14: RB521S
C13: 360 pF ceramico (0603)	DZ1: MM3Z3V3
C14: 10 μ F ceramico (0603)	FUSE: MF-MSMF050-2 500mA
C15: 4,7 μ F ceramico (0603)	Q1: FDN340P
C16: 100 nF ceramico (0603)	Q2: NTR4171P
C17÷C20: 22 pF ceramico (0603)	Q3: 2N7002
C21, C22: 1 μ F ceramico (0603)	Q4÷Q5: FDN340P
C23: 4,7 μ F ceramico (0603)	Q6: NTR4171P
C24÷C28: 1 μ F ceramico (0603)	Q7÷Q9: 2N7002
C29: 22 μ F ceramico (0603)	R1, R2: 100 kohm (0603)
C30÷C31: 1 μ F ceramico (0603)	R3÷R10: 10 kohm (0603)
C32: 4,7 μ F ceramico (0603)	R6÷R8: 220 kohm (0603)
C33, C34: 1 μ F ceramico (0603)	R9: 97,6 kohm (0603)
C35, C36: 4,7 μ F ceramico (0603)	R10: 13,3 kohm (0603)
C37: 1 μ F ceramico (0603)	R11: 1 kohm (0603)
C38, C39: 2,2 μ F ceramico	R12: 475 kohm (0603)
	R13: 105 kohm (0603)
	R14: 470 ohm (0603)
	R15: 10 kohm (0603)
	R16: 2,7 kohm (0603)

ridotte, e popolata su entrambi i lati.

Pur utilizzando un PCB a due facce, quindi teoricamente realizzabile con metodi artigianali, ha molte piste particolarmente sottili e moltissime vie (connessioni tra le due facce) che ne rendono la realizzazione casalinga alquanto difficile.

Il montaggio non risulta particolarmente complesso, salvo per un componente, il codec audio ALC5631Q, che, essendo in package QFN ha i piedini sotto il corpo (ma anche lungo il bordo) ed è praticamente impossibile da saldare con un comune saldatore a stilo, ma richiede la stazione ad aria calda e del flussante co-

sparso sulle piazzole, dopo aver sciolto su queste ultime un velo di stagno. L'impresa è fattibile, a patto di disporre dell'attrezzatura adatta e di un PCB realizzato in modo professionale e non casalingo.

Suggeriamo quindi ai meno esperti di acquistare la scheda già pronta all'uso, visto che a nostro avviso il tempo ed il costo dei materiali per una realizzazione casalinga non vengono compensati dal risparmio che se ne potrebbe ottenere. Sul nostro sito web www.elettronica.in.it mettiamo comunque a disposizione tutti i file necessari per poterlo fare, come d'abitudine; poi decidete voi.

Per tutte le fasi del montaggio, seguite i disegni nella pagina accanto.

MESSA IN FUNZIONE

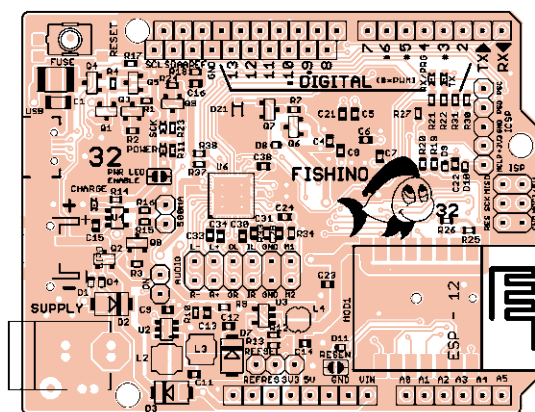
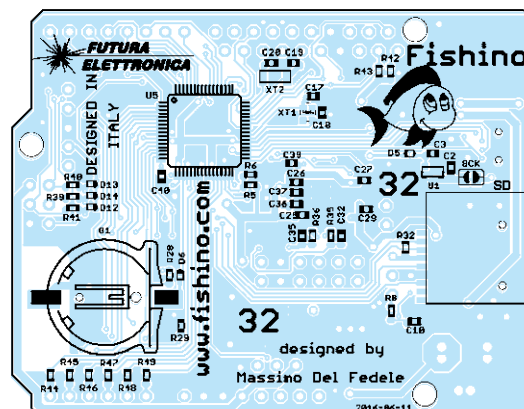
Una volta completata la vostra Fishino 32, dovete procedere alla sua programmazione, che, come abbiamo detto nella prima puntata, si può effettuare tranquillamente tramite l'IDE di Arduino, avendo l'accortezza di installare il package da noi fornito sul sito www.fishino.it contenente tutti i programmi aggiuntivi necessari (programmi di terze parti). L'installazione del package, grazie al lavoro del team di Arduino, è decisamente semplice: è sufficiente aprire l'IDE, selezio-

R17÷R18: 1 Mohm (0603)
 R19: 1 kohm (0603)
 R20: 10 kohm (0603)
 R21, R22: 330 ohm (0603)
 R23: 470 ohm (0603)
 R24: 220 kohm (0603)
 R25÷R26: 10 kohm (0603)
 R26, R28, R29: 10 kohm (0603)
 R27, R30, R31: 3,3 kohm (0603)
 R32: 10 kohm (0603)
 R33, R34: 680 ohm (0603)
 R35, R36: 2,2 kohm (0603)
 R37, R38: 10 kohm (0603)
 R39÷R41: 220 ohm (0603)
 R42, R43: 10 kohm (0603)
 R44÷R49: 220 ohm (0603)
 U1: XC6206P33
 U2: SX1308
 U3: LC3406
 U4: MCP73831
 U5: PIC32MX470F512HI/MR
 U6: ALC5631Q
 XT1: 16MHz
 XT2: 32.768 kHz
 TX: LED blu (0603)
 RX/PRG: LED giallo (0603)
 SCK: LED bianco (0603)

POWER: LED verde (0603)
 CHARGE: LED rosso (0603)
 L1: 2.2 µH
 L2: 6.8 µH
 L3: 6.8 µH
 USB: Connettore micro-USB
 SD: Connettore micro-SD
 MOD1: Modulo ESP12
 con ESP8266
 LIPO: Connettore JST
 2 vie 2.54 mm
 G1: Porta batterie Ø12 mm
 per CR1220
 SUPPLY: Plug alimentazione
 RESET: Microswitch

Varie:

- Strip femmina 3 vie (2 pz.)
- Strip femmina 6 vie
- Strip femmina 8 vie (2 pz.)
- Strip femmina 10 vie (2 pz.)
- Strip maschio 2 vie (2 pz.)
- Strip maschio 5 vie
- Strip maschio 6 vie (2 pz.)
- Jumper (2 pz.)
- Circuito stampato S1291



nare dal menu *File* il sottomenu *Impostazioni* e, nella riga intitolata *Url aggiuntive per il Gestore schede*, inserire il seguente percorso: www.fishino.it/arduinoide/package_fishino_index.json. Come si vede nella **Fig. 1** è possibile inserire più percorsi, facendo clic sul pulsante , tramite il quale si aprirà la finestra in cui confermare o cercare nuovi percorsi. Una volta inserito il percorso di Fishino è sufficiente entrare nel menu *Strumenti* e impartire il comando *Scheda>Gestore schede...*, tramite il quale si avrà accesso al pannello di gestione delle schede aggiuntive; verso la fine troverete il pacchetto *fishino_pic32 by Fishino* (**Fig. 2**).

Sulla destra potete notare il pulsante *Installa*, insieme a una casella per selezionare il numero di versione. Al momento della stesura di questo articolo il numero di versione è 1.0.1, ma il pacchetto è in continuo aggiornamento. Fate clic sul pulsante e verrà avviato lo scaricamento dei pacchetti, che potrà durare anche parecchi minuti; non preoccupatevi, perché è un'operazione da eseguire una sola volta all'inizio (o per aggiornare la scheda, nel qual caso verranno scaricati solo gli elementi modificati). Una volta terminata l'installazione, la schermata si presenterà come in **Fig. 3**. Come si nota, il

pulsante *Installa* è stato ora sostituito da *Rimuovi* ed è apparso l'elenco delle schede disponibili; non preoccupatevi se sarà diverso da quanto appare nell'immagine: stiamo studiando ulteriori schede!

Una volta completata questa operazione, nel menu di selezione della scheda troverete la nostra Fishino32.

Adesso la vostra Fishino 32 è quasi operativa; mancano solo i driver USB, che per alcuni sistemi operativi sono da installare (non è il caso di Linux, nel quale sono inclusi, e nemmeno di MacOS X).

Per Windows occorre procedere come spiegato di seguito: i driver

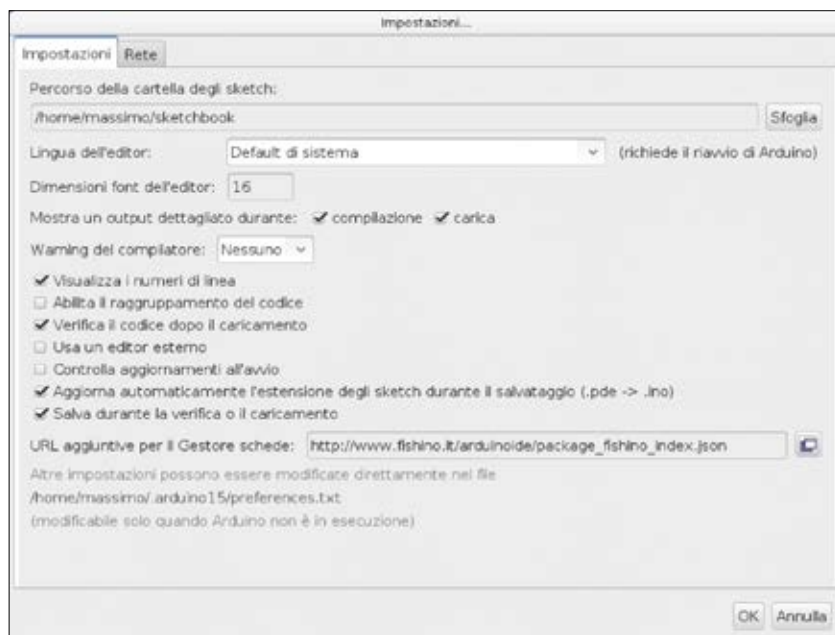


Fig. 1 - Definizione del percorso di ricerca della scheda.

vengono scaricati automaticamente insieme al pacchetto software contenente le descrizioni delle nuove schede che abbiamo appena installato; purtroppo non vengono installati automaticamente nel sistema operativo, per motivi tecnici che in questa sede sarebbe lungo esporre.

Per l'installazione è sufficiente, una volta collegato Fishino al PC, seguire le istruzioni che appariranno sullo schermo, cercando i driver manualmente nel percorso `C:\Users\nomeutente\AppData\Local\Arduino15\packages\fishino\tools\pic32-driver-windows\1.0.0\chipKIT Drivers`, dove

al posto di *nomeutente* va scritto il nome dell'utente da cui state installando i driver.

Una volta selezionato il file `Stk500v2.inf` il driver verrà installato ed apparirà la relativa porta COM virtuale nell'IDE. Una volta installati (se necessario) i driver, il sistema è pronto all'uso.

Adesso occorre selezionare la porta corretta nell'IDE, che sarà una COMxx su Windows o una TTYACMxx se lavorate con il sistema operativo Linux.

ESECUZIONE DEL PRIMO SKETCH

Eccoci pronti per il collaudo della nostra scheda! Qui troviamo subito una piccola differenza, che può sembrare una scomodità rispetto ad Arduino e alle schede Fishino ad 8 bit, ma che in realtà spesso si traduce in un vantaggio: la scheda attualmente non dispone di un autoreset che la mette automaticamente in modalità caricamento sketch; occorre quindi premere il pulsante di reset, tenendolo premuto per circa 2 secondi, finché non lampeggia il LED arancione di programmazione, ma non più di 4 secondi circa, dopodiché viene impostata la modalità di aggiornamento del firmware del modulo WiFi, condizione che sarebbe evidenziata dal lampeggio del LED blu.

Se ci è scappato il tempo, poco male: rilasciamo il tasto reset e ricominciamo.

Vi chiederete il perché di questo modo di funzionamento; ebbene, deriva dal fatto che, disponendo la scheda di un USB nativo, questo è legato al funzionamento del controller stesso, come nel caso delle schede Arduino Leonardo. Se il controller si blocca (basta uno sketch che si blocchi) la porta USB non risponde più ai comandi, nemmeno a quelli di reset. Abbiamo preferito quindi



Fig. 2 - Ricerca dei pacchetti d'installazione di terze parti.

Caricamento del bootloader

al momento un sistema manuale che eviti, ad esempio, la famigerata 'manovra di emergenza' necessaria in alcuni casi sulle schede Arduino, ovvero la pressione del tasto reset "un istante" dopo aver lanciato la programmazione dall'IDE.

Non escludiamo comunque di inserire, in una prossima versione del bootloader, un sistema di reset automatico, anche se i vincoli di cui abbiamo parlato restano. Torniamo quindi al nostro sketch; carichiamo l'esempio BLINK dall'IDE, quello che fa lampeggiare il led13 (che corrisponde al led bianco sulla scheda), teniamo premuto il tasto reset per circa 2 secondi e, quando inizia a lampeggiare il LED arancione, premiamo il pulsante di caricamento sketch dell'IDE.

Dopo pochi secondi (dipende dalla velocità del PC, dalla complessità dello sketch, dal numero di librerie utilizzate, eccetera...) il caricamento sarà completato, cosa evidenziata dallo spegnimento del led arancione e dal contemporaneo lampeggio del LED bianco.

Ecco fatto: scheda collaudata e funzionante! Ed ora?

AGGIORNIAMO IL MODULO WIFI

Se avete seguito gli articoli sulle schede Fishino ad 8 bit ricordere sicuramente che la procedura di aggiornamento firmware richiedeva alcune connessioni volanti tra gli I/O della scheda e la porta ESPCONN, il caricamento preventivo di uno sketch "neutro", ad esempio il blink, eccetera.

Nulla di tutto ciò è più necessario sul Fishino32: la scheda viene posta in modalità aggiornamento firmware utilizzando semplicemente il pulsante di reset!

La procedura è la seguente:

- premere e mantenere premuto

Se avete acquistato la scheda montata, questa è già dotata di bootloader sul controller, quindi potreste tranquillamente saltare questa parte di articolo. Diciamo "potreste" perché, come tutto il software, anche il bootloader può avere delle evoluzioni, anche se meno frequenti rispetto a librerie e firmware, quindi è sempre utile sapere come fare per riprogrammarlo.

La programmazione del bootloader diviene poi indispensabile se avete realizzato la scheda seguendo le nostre istruzioni.

Se qualcuno di voi ha già caricato il bootloader su un Arduino (o un Fishino) ad 8 bit saprà sicuramente che occorre un programmatore esterno; la cosa non si può fare attraverso la porta USB.

Per le schede ad 8 bit era sufficiente utilizzare un altro Arduino programmato per quello scopo e connesso opportunamente, seguendo facili istruzioni reperibili in rete, al connettore ISP, quello da 3+3 poli.

Per il Fishino32 le cose cambiano anche se, grazie soprattutto al lavoro della comunità, non di molto.

È sempre possibile utilizzare un Arduino (o un Fishino) ad 8 bit come programmatore, utilizzando lo schema di **Fig. A**.

Il programmatore va connesso non più alla porta ISP ma alla ICSP, ovvero al connettore con 5 pins in linea, denominati MCLR, +3V3, GND, PGD e PGC. Come si evince dallo schema, sulle porte PGD e PGC vanno inserite due resistenze di pullup connesse NON ai 5V del programmatore ma ai 3,3 Volt della scheda da programmare, che DEVE quindi essere alimentata durante la programmazione.

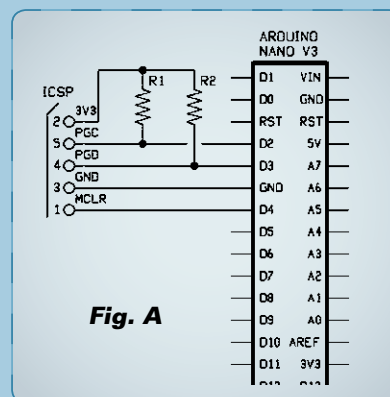


Fig. A

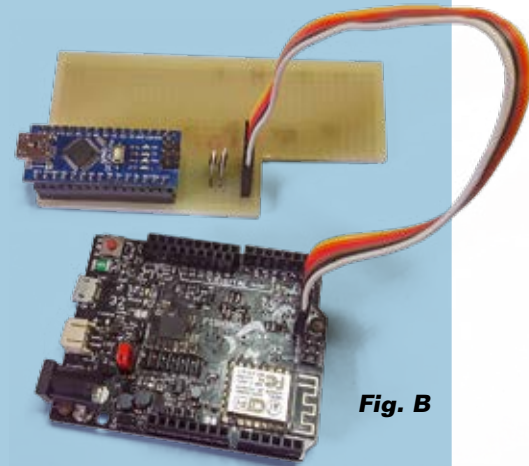


Fig. B

Il nostro programmatore sfrutta queste due resistenze come un traslatore di livello fittizio, in modo da non superare i 3.3 Volt ammissibili sulle due entrate del Fishino32; un livello basso si ottiene portando l'uscita a massa, un livello alto si ottiene lasciandola fluttuante, in modo che assuma i 3,3 Volt provenienti dal connettore stesso.

Il programmatore può essere tranquillamente realizzato su una breadboard o anche, data la semplicità, tramite fili volanti, visto che verrà usato pochissimo (**Fig. B**).

Una volta preparato e connesso il circuito, caricate l'apposito sketch sull'Arduino utilizzato come programmatore, scaricabile al seguente indirizzo:

https://github.com/sergev/pic32prog/blob/master/bitbang/ICSP_v1E.ino

Ora è sufficiente lanciare l'IDE (la quale ovviamente dev'essere dotata dei componenti aggiuntivi scaricati come da articolo), selezionare come programmatore "ARDUINO BITBANG" che appare in fondo alla lista relativa, selezionare come porta USB quella del programmatore, selezionare la scheda corretta (FISHINO32 in questo caso) e cliccare su "Scrivi il bootloader". Se è tutto a posto, l'IDE mostrerà alcuni messaggi di progresso e poi darà la conferma dell'avvenuto caricamento del bootloader.

Scollegando il programmatore vedrete quindi il led arancione lampeggiare, segno della modalità di caricamento sketch attiva, visto che qualsiasi eventuale sketch precedente sarà stato cancellato dalla procedura.

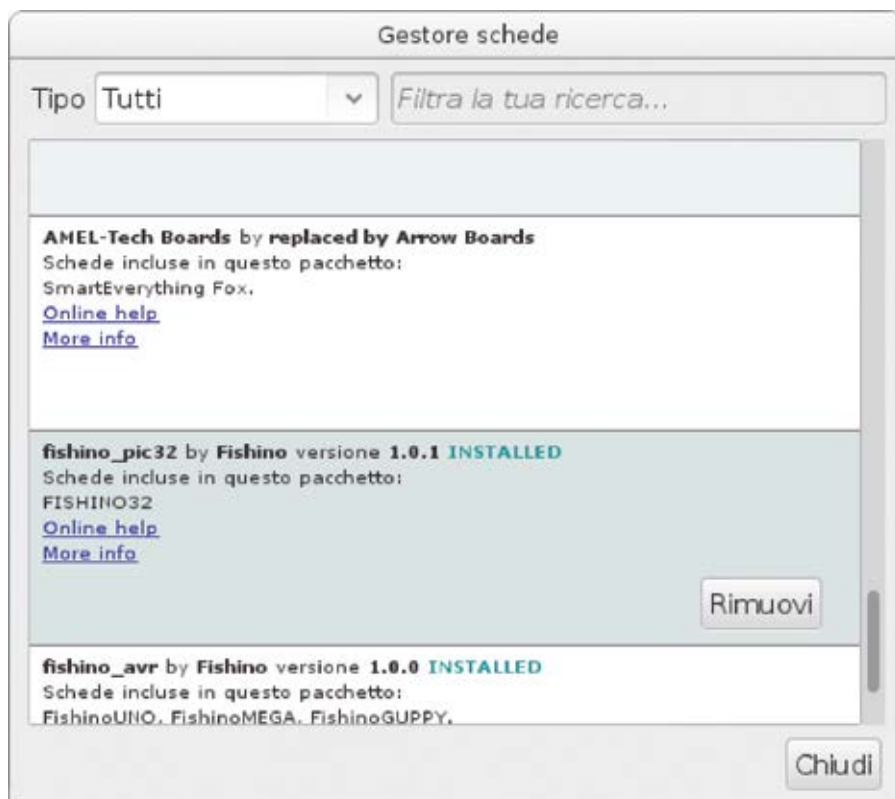


Fig. 3 - Installazione terminata.

il tasto reset finché non inizia a lampeggiare il LED blu; notate che occorrono 4 secondi, trascorsi i primi due dei quali

inizierà a lampeggiare il LED arancione, a indicare la modalità di caricamento sketch, poi (dopo altri 2 secondi) inizierà a

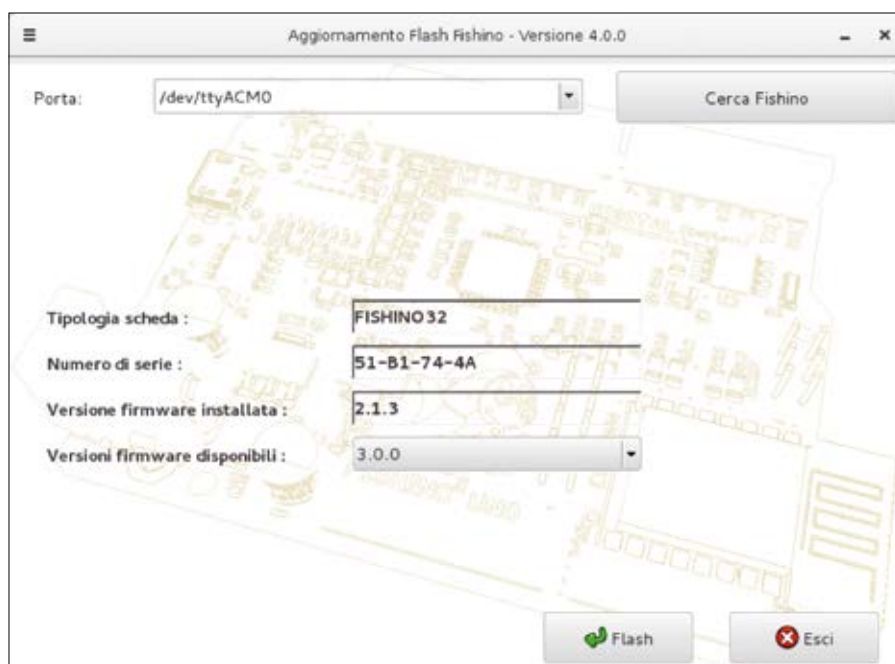


Fig. 4 - Schermata di FishinoFlasher.

lampeggiare il LED blu;

- lanciare il programma FishinoFlasher (scaricabile da <http://fishino.it/>), dopo aver verificato di avere una connessione Internet funzionante sul proprio computer, visto che il firmware verrà scaricato dalla rete wireless e quindi dal web.

Una volta avviato, il programma FishinoFlasher presenterà con la schermata di **Fig. 4**; nella schermata si possono notare alcuni elementi, tra i quali la tipologia della scheda rilevata (Fishino32), il numero di serie del modulo WiFi e la versione attualmente installata del firmware (in questo caso è la 2.1.3).

Nella casella sottostante è possibile scegliere una nuova versione da installare; viene proposta l'ultima versione disponibile (la 3.0.0, in questo caso), ma nulla vieta di installare una versione precedente, anche se normalmente non è consigliato.

Fate attenzione al fatto che ad un cambio di numero di versione "major" (la prima cifra a sinistra, che in questo caso è 3) corrisponde una non retrocompatibilità con le librerie precedenti; in pratica, una libreria installata e funzionante con la versione 2.5.5 funzionerà anche con un firmware di versione 2.5.7 e 2.9.9, per esempio, anche se non ne sfrutterà le migliorie aggiuntive, ma non è vero il contrario, ovvero una libreria fatta per la versione 2.5.7 richiederà un firmware di versione pari almeno a 2.5.7). Per contro, quella libreria non funzionerà con un firmware 3.0.0. In sintesi, una libreria creata per un firmware avente un numero di versione major X funzionerà con tutti i firmware successivi aventi lo stesso numero di versione major, ma non con i precedenti né con i successivi

aventi numero di versione major differente.

Detto ciò, scegliamo la nostra versione e facciamo clic sul pulsante Flash; dopo pochi istanti inizierà l'aggiornamento vero e proprio (Fig. 5). Al termine della procedura apparirà una finestra di notifica dell'avvenuto aggiornamento.

A questo punto è sufficiente fare clic su Ok, chiudere il programma FishinoFlasher e premere il tasto di reset sulla scheda per uscire dalla modalità aggiornamento.



LIBRERIE SOFTWARE DISPONIBILI

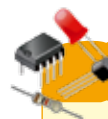
Possiamo dividere le librerie in due categorie logiche:

- librerie di sistema;
- librerie utente.

Si tratta di una divisione abbastanza fittizia, visto che bene o male tutte le librerie potrebbero essere considerate parzialmente in entrambi i settori; la vera differenza sta nel dove vengono collocate sull'hard disk e nelle modalità di aggiornamento. Le librerie di sistema sono collocate nella cartella arduino15 (che può trovarsi in posizioni differenti a seconda del sistema operativo) e vengono gestite ed aggiornate dal gestore di schede integrato nell'IDE. A questa categoria appartengono, ad esempio, la libreria SD, la AudioCodec,

la I2S, eccetera. Ad ogni aggiornamento disponibile l'IDE, ne segnalerà la presenza e darà la possibilità di scaricare automaticamente gli aggiornamenti. La libreria Fishino, per contro, insieme alla FishinoWebServer ed altre eventuali, appartiene alla seconda categoria per una nostra precisa scelta; questa libreria infatti dipende non dalla scheda in sé ma dalla versione di firmware del modulo WiFi, e va quindi aggiornata in base a questo. Inserirla tra le librerie di sistema avrebbe obbligato l'utente ad aggiornarla, e quindi ad aggiornare il firmware, ad ogni aggiornamento delle librerie di sistema, cosa non strettamente necessaria. Questa libreria va quindi scaricata direttamente dal sito www.fishino.it, sezione download, come per le precedenti schede; in quella sezione troverete anche i programmi di esempio, tra i quali il FishinoHomeAuto. Le librerie della sezione download sono fornite come files .zip da decomprimere nella cartella "libraries" presente dentro la cartella contenente gli sketch.

Una volta eseguita questa procedura, è possibile provare tutti gli esempi relativi al modulo wifi come per le precedenti schede. Una nota riguardo a questi ultimi: il loro funzionamento è identico, sulla Fishino32, a quello delle schede precedenti; noterete solo un notevole incremento delle prestazioni e, ovviamente, un'occupazione di memoria decisamente lontana dai limiti della scheda! Ora che avete tra le mani tutta la potenza di Fishino 32, siete pronti a sviluppare.



per il MATERIALE

La board Fishino32 (cod. FISHINO32) viene fornita montata e collaudata. Può essere acquistata presso Futura Elettronica al prezzo di Euro 59,00. Il prezzo si intende 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>

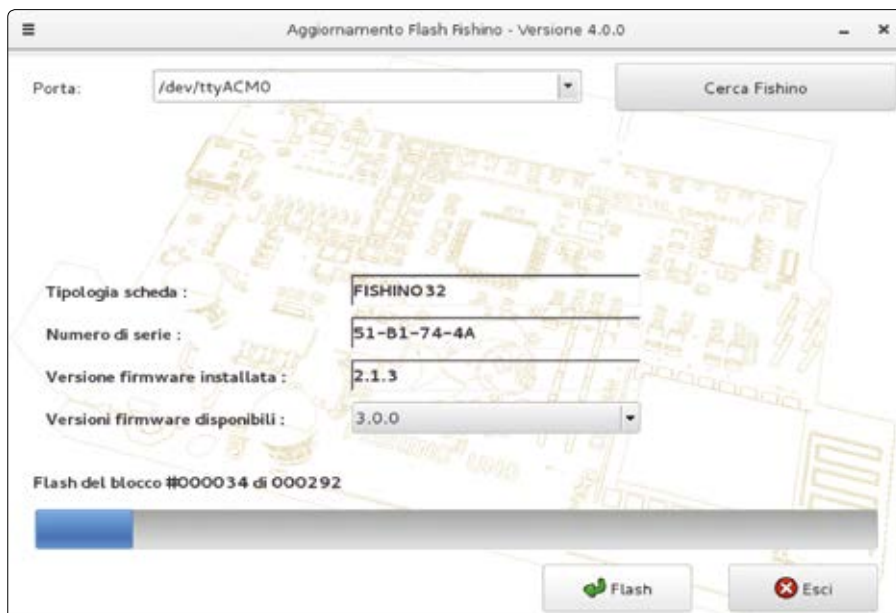
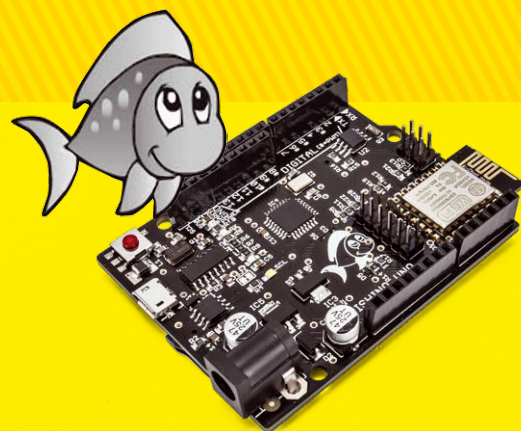


Fig. 5 - Aggiornamento del firmware.

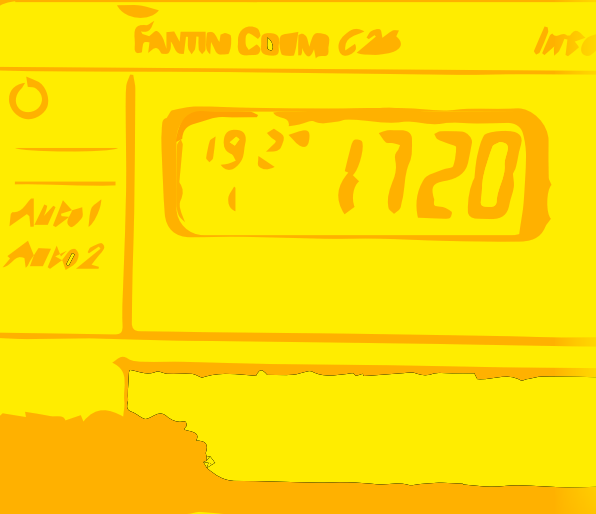
Fishino Contest: I PROGETTI VINCITORI



TERMOSTATO FISHINO

di CRISTIANO COLLIGIANI

Gestore dell'impianto di riscaldamento integrabile con il termostato esistente, che aggiunge funzioni come il comando remoto e il rilevamento dei consumi.



Quante volte ci chiediamo se l'impianto di riscaldamento di casa nostra è abbastanza efficiente? E quante vorremmo poterlo controllare da remoto in modo da trovare la casa calda al nostro ritorno dal lavoro o da un viaggio?

Ebbene, il progetto descritto in queste pagine, piazzatosi al terzo posto del nostro Fishino Contest, nasce dall'esigenza di supervisionare il sistema di riscaldamento di una casa senza sconvolgere l'impianto esistente, in modo da poter monitorare l'andamento della temperatura e dell'umidità durante la giornata. Inoltre dispone di un ingresso (da collegare al termostato tradizionale che governa l'impianto domestico) per vedere in quali momenti esso si attiva e quanto tempo sta

acceso nell'arco di una giornata o in un mese, così da valutarne l'efficienza e stimarne i consumi. Per concludere, nel circuito è presente anche un'uscita che ci permette di azionare l'impianto di riscaldamento da remoto. E per chi non volesse utilizzarlo per questo scopo ci sono svariati altri impieghi, come ad esempio la combinazione con una stazione meteorologica o una semplice scheda di controllo di un carico (ingresso/uscita) elettrico. Diamo dunque la parola all'autore del progetto, che ci spiegherà il lavoro presentatoci.

L'HARDWARE

Il cuore del sistema è la scheda Fishino, che a sua volta contiene il modulo ESP8266 che permette di collegarsi alla nostra rete WiFi,

il modulo RTC che ci consente di avere sempre a disposizione l'ora corrente anche quando viene interrotta l'alimentazione principale, uno slot per scheda SD che ci permette di ospitare le pagine web e di salvare i dati rilevati.

Come sensore ho utilizzato il DHT22 della Aosong, un componente molto diffuso e ben documentato in Internet (trovate tutto alla pagina web www.aosong.com/en). Questo sensore, che nello schema elettrico è siglato U1, consente di misurare contemporaneamente temperatura e umidità relativa con una risoluzione del decimo di unità.

Il DHT22 dialoga con il microcontrollore -nello specifico con la nostra scheda Fishino UNO- attraverso il protocollo "One Wire" che permette di comunicare utilizzando un solo filo (tramite il contatto DATA) ed una resistenza di pull-up (R7) da 10 kohm. La sezione di input (INGRESSO) è stata progettata per collegare un contatto deviatore come quello di un termostato che realizza la funzione estate/inverno e non ha il selettore, ovvero il contatto interruttore di un termostato con selettore estate/inverno. Prima di procedere è il caso di spiegare tale differenza: nei termostati elettromeccanici a lamina bimetallica, il contatto attacca sotto una certa temperatura e stacca superata un'altra temperatura, leggermente superiore per assicurare la necessaria isteresi; se il contatto viene utilizzato così com'è va bene per il controllo di un impianto di riscaldamento, mentre se deve comandare un condizionatore occorre invertire il funzionamento. Collegandolo al circuito, bisogna utilizzare i contatti 1 e 2 nel primo caso e nel secondo è necessario usare 2 e 3 applicando un resistore di pull-up tra 2 e Vcc.

Se il termostato ha un deviatore, invece, si collega il centrale al 2 e gli altri contatti a 2 e 3 a seconda di cosa dev'essere comandato. Se il termostato fornisce una tensione invece che il contatto pulito, bisogna usare 2 e 3.

Detto ciò, notate che per garantire un'ampia dinamica della tensione di ingresso ho inserito alcuni componenti che sono i diodi D1 e D3 e le resistenze R3 ed R4 (queste ultime formano un partitore di tensione) e che proteggono l'ingresso di Fishino UNO deputato a leggere la condizione del termostato dell'impianto (resistenza R3, che fa partitore con la R4) e lo zener D1 servono per ridurre la ten-

sione ad un valore accettabile dalla scheda Fishino UNO.. Grazie a questa rete, come livello logico alto si può utilizzare sia lo standard TTL (5V) sia tensioni più elevate, come 12 o 24 volt, spesso impiegate nei vari apparecchi elettronici. Naturalmente se si sceglie di utilizzare la logica 0/5V i componenti D1, D3 ed R3 possono non essere montati e l'ingresso può essere portato direttamente sul pin 2. Il diodo D3, infatti, serve per impedire che un'inversione di polarità danneggi il circuito, mentre la resistenza R4 provvede a portare a massa il pin di ingresso in modo da evitare che rimanga flottante quando non vi è applicato alcun segnale d'ingresso. Supponendo di avere (caso peggiore) una tensione di 24V in ingresso, la caduta su R3 sarà pari a

$$V_{r3} = 24V - 0,7V - V_z$$

dove 0,7 V è la caduta di tensione sul diodo D3 e V_z quella inversa sullo Zener, che ammonta a 4,7 V nelle condizioni tipiche. Sulla R3 cadono quindi 18,6 V; la corrente in essa ammonta a:

$$I_{r3} = 3,96 \text{ mA}$$

che vengono divisi tra R4 e lo Zener, i quali dissipano nel complesso:

$$P_d = 3,96 \text{ mA} \times 4,7 \text{ V} = 18,6 \text{ mW}$$

che rappresentano una potenza più che sopportabile dai componenti e in particolare dallo Zener, che ho scelto da 1/2 W. In questi calcoli non è stata considerata la resistenza R4, in quanto

la corrente che scorre al suo interno è trascurabile rispetto a quella assorbita dal diodo Zener. Se applichiamo all'ingresso una tensione di 5 volt, sottratta la caduta di 0,6V dovuta al diodo al silicio D3, la tensione sul pin D2 verrà letta come 1 logico. La differenza di potenziale ai capi della R3 è irrile-



*L'autore del progetto,
Cristiano Colligiani.*

Tabella 1 - Segnalazioni visive fornite dalla scheda.

Lampeggi	Stato
0	Errore Generico (probabile blocco della scheda)
1	OK (Nessun Errore)
2	Errore Scheda SD (o nel file "conf.txt")
3	Errore Connessione alla Rete
4	Access Point non trovato
5	Password di rete sbagliata
6	Errore Modulo RTC (o non configurato)
7	Errore Lettura del sensore DHT22

Listato 1

```
La libreria DHT

//
// FILE: dht.h
// AUTHOR: Rob Tillaart
// VERSION: 0.1.21
// PURPOSE: DHT Temperature & Humidity Sensor library for
// Arduino
// URL: http://arduino.cc/playground/Main/DHTLib
//
// HISTORY:
// see dht.cpp file
//

#ifndef dht_h
#define dht_h

#if ARDUINO < 100
#include <WProgram.h>
#include <pins_arduino.h> // fix for broken pre 1.0 version - TODO TEST
#else
#include <Arduino.h>
#endif

#define DHT_LIB_VERSION "0.1.21"

#define DHTLIB_OK 0
#define DHTLIB_ERROR_CHECKSUM -1
#define DHTLIB_ERROR_TIMEOUT -2
#define DHTLIB_ERROR_CONNECT -3
#define DHTLIB_ERROR_ACK_L -4
#define DHTLIB_ERROR_ACK_H -5

#define DHTLIB_DHT11_WAKEUP 18
#define DHTLIB_DHT_WAKEUP 1

#define DHTLIB_DHT11_LEADING_ZEROS 1
#define DHTLIB_DHT_LEADING_ZEROS 6

// max timeout is 100 usec.
// For a 16 Mhz proc 100 usec is 1600 clock cycles
// loops using DHTLIB_TIMEOUT use at least 4 clock cycles
// so 100 us takes max 400 loops
// so by dividing F_CPU by 40000 we "fail" as fast as possible
#ifndef F_CPU
#define DHTLIB_TIMEOUT 1000 // should be approx. clock/40000
#else
#define DHTLIB_TIMEOUT (F_CPU/40000)
#endif

class dht
{
public:
    dht() {}
    int8_t read(uint8_t pin);
    int humidity;
    int temperature;

private:
    uint8_t bits[5]; // buffer to receive data
    int8_t readSensor(uint8_t pin, uint8_t wakeupDelay,
    uint8_t leadingZeroBits);
};
#endif
//
// END OF FILE
//
```

Ho voluto allegare entrambe le tipologie perché i file "Offline" sono pesanti rispetto agli altri e per caricare una pagina occorrono circa 30 secondi, cosa che può risultare fastidiosa quando andiamo ad interagire con il sistema. Questa lentezza è do-

vuta a Fishino, che per quanto veloce è indubbiamente lento rispetto ai server che troviamo in rete. Tuttavia può capitare di inserire questo progetto in una rete che non è collegata direttamente ad Internet ed è questo il caso di utilizzare la versione "Offline".

Per collegare la scheda alla nostra rete domestica deve essere costruito il file "conf.txt":

```
Immettere SSID
Immettere Password
-570316608
```

Questo file può essere creato con un comune blocco note e deve essere così strutturato: la prima riga contiene il nome della nostra rete WiFi (SSID) mentre la seconda riga contiene la password. La terza riga contiene l'indirizzo IP in forma signed long e con i byte invertiti. Ad esempio se voglio assegnare l'indirizzo 192.168.1.253 devo considerare come byte più significativo 253 che di conseguenza porterà a un numero negativo, per la precisione -50222912. Ho deciso di rappresentare gli indirizzi IP in questo modo per semplificare la logica interna. La memoria (come vedremo tra poco) è molto limitata e dover manipolare le stringhe per convertirle in numeri risulta molto complesso per il microcontrollore e le funzioni richiedono molte risorse. Così facendo il numero viene acquisito direttamente grazie alla funzione **atoi()** integrata in Arduino e convertito istantaneamente nell'oggetto IPAddress. Siccome la procedura può sembrare complessa, soprattutto per chi è alle prime armi o non è esperto, ho preparato un file Microsoft Excel (scaricabile dal nostro sito web) che consente di calcolare velocemente il numero da mettere nella terza riga. Se volessimo utilizzare la scheda in modalità DHCP, facendo in modo che l'indirizzo venga assegnato automaticamente dal nostro modem, sarà sufficiente mettere 0 nella terza riga.

SOFTWARE

Il software è composto da due parti: una parte formata da pagine HTML, che consentono di interfacciarsi con la scheda e l'altra è costituita dal firmware del microcontrollore, creato con l'ambiente Arduino che registra i dati rilevati e li memorizza su file di testo. Vista la scarsità di risorse che la scheda Fishino può offrire in termine di potenza computazionale e di memoria rispetto a un computer, ho scelto di far elaborare i dati al PC attraverso il linguaggio javascript e lasciare a Fishino il semplice compito di "datalogger". In questo modo Fishino campiona i dati facendone una media e

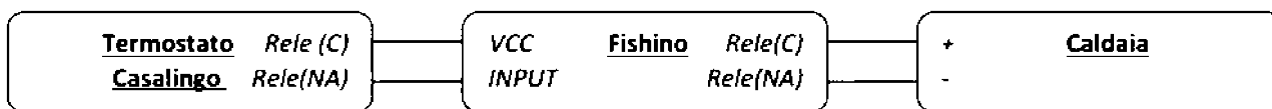


Fig. 2 - Connessioni tra l'hardware del progetto, la caldaia e il termostato tradizionale che la controlla.

ogni 5 minuti (tempo deciso da me) la salva su un file di testo. Il file è così strutturato: il suo nome indica la data a cui esso fa riferimento (ad esempio 20160320.txt fa riferimento al 20 marzo 2016) e all'interno del file ci sono diverse righe, ognuna relativa ad un intervallo di 5 minuti (ad esempio la prima riga contiene i dati dalle 00:00 alle 00:05, la seconda dalle 00:05 alle 00:10 e così via). I dati all'interno della riga sono espressi con numeri interi e separati da virgola: il primo valore che troviamo è la temperatura, poi vi è l'umidità e infine lo stato (ad esempio 123,456,1 sulla riga 3 indica che la temperatura media registrata tra le 00:10 e le 00:15 è di 12,3 °C, l'umidità è 45,6% e lo stato dell'ingresso è ON). I file vengono generati automaticamente dalla scheda con il passare del tempo scandito dall'RTC e non vengono sovrascritti se il tempo viene mandato indietro, cioè vengono preservati i dati raccolti in passato a meno che non si eliminino i file relativi a tale data dalla scheda SD. Se l'orologio RTC non viene impostato oppure perde la data a causa della perdita di alimentazione e dell'assenza della batteria tampone, il processo di salvataggio dei dati si interrompe e viene segnalato l'errore RTC (codice 6). Per questo software sono state utilizzate le seguenti librerie:

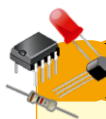
- Fishino (per interagire con il modulo ESP a bordo della scheda);
- FishinoWebServer (per fornire una connessione http al client);
- Flash (necessaria per le librerie sopra);
- DhtLib (per interfacciarsi con il sensore DHT22);
- RtcLib (per interfacciarsi con il modulo Rtc presente sulla scheda Fishino);
- SD (per leggere e scrivere i file contenuti nella scheda SD).

Il progetto di partenza è l'esempio WebServerSD che è il cuore del sistema. Esso si occupa di elaborare le richieste HTTP che arrivano e di rispondere automaticamente inviando i file richiesti. In questo modo possiamo inserire direttamente i file nella SD come se Fishino fosse un comune server web e creare pagine web articolate e ricche di contenuti. Questo esempio tuttavia occupa circa il 75% delle risorse della scheda e ciò ha reso molto difficile

la programmazione perché immettendo le altre librerie lo sketch diventava troppo grande. Per risolvere questo problema ho preso i sorgenti delle librerie di Fishino e li ho inseriti direttamente nello sketch in modo da poterli modificare. Le modifiche introdotte sono:

- da tutti i file sono stati rimossi i riferimenti alla porta seriale, che viene utilizzata da Fishino per comunicare eventuali errori; siccome nel nostro caso il pc non sarà collegato alla scheda questa funzione non ha interesse ed eliminandola si risparmia molta memoria;
- il buffer di Fishino è stato ridotto da 256 caratteri a 128, operazione che ridurrà sicuramente la velocità di Fishino nell'elaborare le richieste HTTP ma d'altra parte libererà 128 byte di memoria che sono indispensabili per la stabilità del sistema; senza tale riduzione, quando veniva richiesta una pagina non veniva inviata alcuna risposta perché la funzione **malloc()** presente nelle varie funzioni non riusciva ad allocare la memoria necessaria per memorizzare la stringa del filename;
- i file "FishinoDebug" sono stati eliminati perché non necessari.

Anche la libreria del DHT è stata integrata nel progetto e modificata; in particolare, è stato cambiato il tipo delle variabili "temperature" e "humidity" da double a int (**Listato 1**). Questo ha permesso di alleggerire notevolmente il programma perché le



per il MATERIALE

La board Fishino UNO (cod. FISHINOUNO) viene fornita montata e collaudata. Può essere acquistata presso Futura Elettronica al prezzo di Euro 36,00. Il prezzo si intende 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>

operazioni con gli interi sono meno complesse dei numeri in virgola mobile, come si specifica anche sul sito ufficiale di Arduino (www.arduino.cc/en/Guide/Troubleshooting#size).

La parte web è stata realizzata con il linguaggio HTML supportato da javascript e anche in questo caso ho utilizzato le librerie esistenti:

- JQueryUI, che ha permesso di realizzare Calendari e ProgressBar molto interattive e accattivanti da un punto di vista estetico;
- Highcharts, che permette in modo veloce di realizzare grafici anche molto complessi.

Le pagine html fondamentali per l'applicazione sono le sei descritte di seguito.

1. Home Page (index.htm): riassume lo stato corrente del sistema, in particolare la temperatura, l'umidità e lo stato dell'ingresso in tempo reale; contiene inoltre i pulsanti per attivare e disattivare l'uscita e lo stato della stessa.
2. Storico Giornaliero (giorno.htm): mostra l'andamento dei dati raccolti durante la giornata corrente; in questa pagina è presente anche un calendario che permetterà di scegliere un giorno diverso da quello corrente.
3. Storico Mensile (mese.htm): mostra l'andamento della temperatura e dell'umidità nei vari giorni del mese specificando anche il valore massimo e il minimo registrato. Riporta inoltre anche il tempo di accensione dell'ingresso. Quando questa pagina viene aperta sarà presente una barra di caricamento. Questo perché, come spiegavo prima, le pagine web vanno a cercare tutti i file del mese e li elaborano, calcolando ad esempio la temperatura massima o il tempo di accensione.
4. Storico Annuale (anno.htm): mostra l'andamento

della temperatura e dell'umidità durante l'anno, suddividendola per mesi. Questa operazione può richiedere molto tempo perché devono essere trasferiti 365 file da Fishino al PC.

5. Impostazioni (settings.htm): consente di impostare data e ora e di vedere lo stato del sistema. L'upload dei dati viene effettuato con il metodo PUT dell'HTTP. Per questa funzionalità ho esaminato l'esempio "FileUpload" nella libreria FishinoWebServer.
6. conf.txt: questo file contiene la configurazione della nostra rete ma quando viene richiesto via http per motivi di sicurezza non viene visualizzato. Al suo posto viene costruito un file sul momento, che viene utilizzato come via di scambio per i dati istantanei; le pagine infatti vengono aggiornate attraverso le funzioni XmlHttpRequest che consentono di aggiornare rapidamente la pagina senza provocare il fastidioso fenomeno di "refresh".

REALIZZAZIONE PRATICA

Il prototipo è stato realizzato su basetta millefori perché l'autore lo ha preferito, sia per rapidità che per mancanza dell'attrezzatura necessaria alla realizzazione di un PCB per incisione. Il montaggio è molto facile in quanto i componenti sono pochi; comunque Fishino ha uno strip aggiuntivo che consente di montarvi sopra una basetta 1.000 fori e cablare il circuito su di essa, quindi non scartate la soluzione millefori. Ho altresì deciso di lasciare sporgere la millefori utilizzata per il prototipo sul lato destro di Fishino, allo scopo di montare il relé, che è il componente più ingombrante. Anche le morsettiere sono state montate in questa sporgenza in modo da poterle raggiungere senza dover rimuovere lo "shield". Per il cablaggio delle varie parti costituenti il sistema vale la solita Fig. 2.

CONCLUSIONI

In realtà questo progetto era stato pensato per sostituire definitivamente un termostato casalingo e quindi curare anche la parte di controllo. Tuttavia per implementare delle regole di accensione, le soglie di temperatura e gli orari, occorre tanta memoria sia Flash che RAM di cui sfortunatamente il nostro microcontrollore non dispone. Proprio per questa ragione ho deciso di trasformare il progetto in un sistema di monitoraggio, che comunque rimane una soluzione molto interessante, con la speranza di poter estendere le sue funzionalità con l'uscita di Fishino Mega e poter costruire un termostato interamente controllato via web. ■

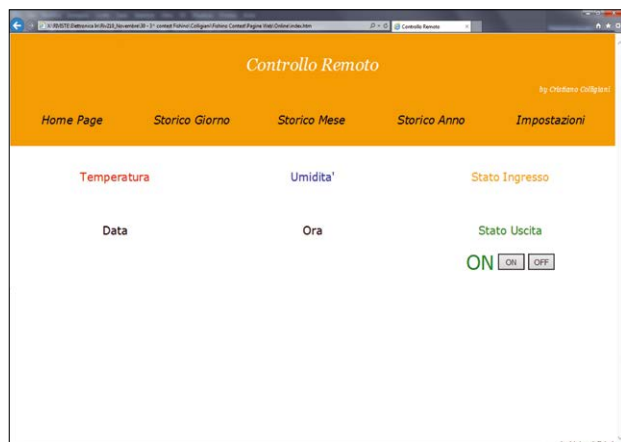


Fig. 3 - La home page dell'applicazione.

SMART HOME SYSTEM CON FISHINO

dell' Ing. MIRCO SEGATELLO

Supponiamo che vogliate realizzare un controllo per le vostre apparecchiature domestiche e che desideriate farlo con una soluzione semplice, modulare e caratterizzata da ampie possibilità di personalizzazione. Supponiamo anche che intendiate averne il controllo da remoto tramite il vostro smartphone, il tutto scrivendo solo poche

righe di codice nel linguaggio di programmazione usato da Arduino. Con le nuove tecnologie sull'IOT di certo non mancano i mezzi per farlo e la soluzione al problema è a portata di mano. In

queste pagine vedremo come trasformare il vostro intento in realtà, sfruttando la scheda Fishino UNO, progettata e realizzata in Italia, che integra tutte le funzionalità di una scheda Arduino UNO alle quali aggiunge un lettore di schede microSD, un



Realizziamo un sistema per il controllo degli elettrodomestici e apparati elettrici di casa attraverso WiFi gestito tramite smartphone.

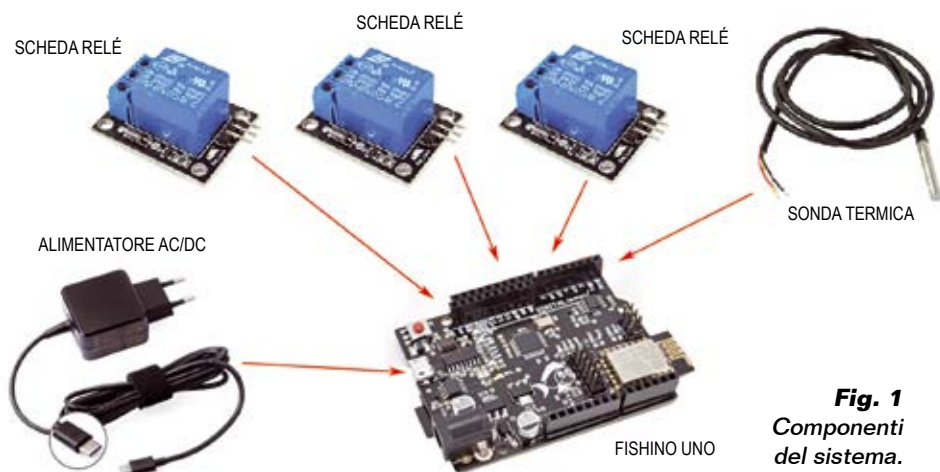


Fig. 1
Componenti
del sistema.

RTC ed un modulo WiFi basato sull'integrato ESP8266. Tale scheda è adatta a realizzare dispositivi domotici, grazie alla comunicazione WiFi configurabile di cui dispone, che permette una facile interazione con dispositivi remoti.

IL PROGETTO

Ipotizziamo di voler gestire una stufetta elettrica per il riscaldamento, un ventilatore ed un punto luce e di voler rilevare la temperatura di una stanza; vogliamo gestire gli utilizzatori accendendoli e spegnendoli secondo necessità tramite uno o più dispositivi "mobile" disponibili in casa. Affinché il tutto funzioni è necessario che nella nostra abitazione sia presente un router WiFi, che permetterà la comunicazione tra i vari dispositivi, mentre Fishino si occuperà della gestione diretta delle apparecchiature ad esso interfacciate. Per poter attivare un carico funzionante con la tensione di rete, la miglior soluzione è un modulo a relé, ossia una scheda conte-

nente un relé a bassa tensione e un minimo di logica per alimentarne la bobina quando viene ricevuto un certo livello logico; il relé rende disponibile lo scambio per commutare il circuito ad alta tensione. Il tutto garantisce l'isolamento galvanico da Fishino. L'occorrenza per realizzare il progetto si acquista tutto presso Futura Elettronica (www.futurashop.it) e i rispettivi codici prodotto sono riportati tra parentesi; la lista completa è:

- scheda Fishino UNO (7305-FISHINO UNO);
- sensore di temperatura DS18B20 (2846-SONDADS18B20);
- tre Modulo Relé (2846-RELAY-1CH);
- alimentatore AC/DC (8822-AL05-3USB).

L'insieme è rappresentato nella Fig. 1. Il Modulo relé è una scheda alimentata a 5 volt contenente un relé con bobina funzionante a tale tensione e singolo scambio capace di commutare 10 A di corrente in circuiti funzionanti a tensione alternata fino a 250 Veff. Per il collegamento delle varie parti del sistema potete fare riferimento allo schema di cablaggio di Fig. 2, che mostra come collegare un modulo relé ed un elettrodomestico che esso gestirà; gli altri moduli relé vanno collegati esattamente allo stesso modo, ma sui pin 4 e 5 di Fishino. I collegamenti completi sono comunque riassunti nella Tabella 1.

Il contatto normalmente aperto del relé farà parte del circuito di potenza connesso alla tensione di rete, pertanto dovete porre la massima attenzione nel cablare il circuito, vista la pericolosità di un circuito a 230V.

Il collegamento del sensore di temperatura DS18B20 è molto semplice e consiste nel connettere il filo rosso dell'alimentazione

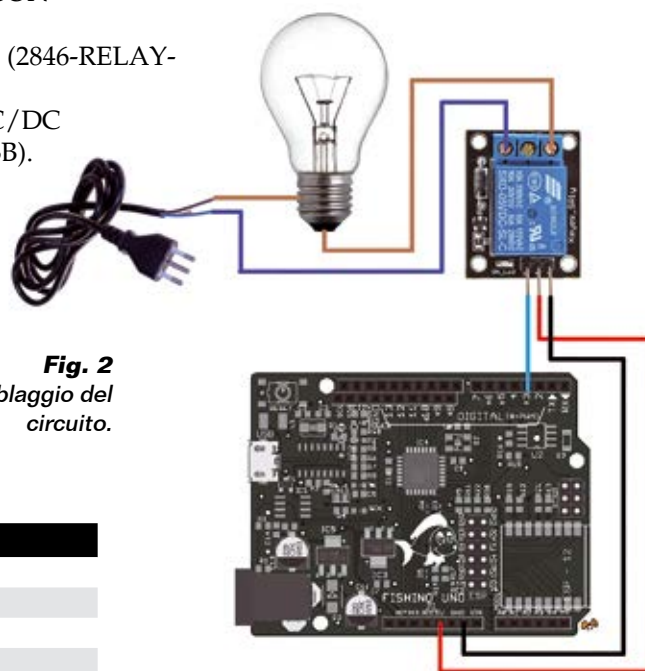


Fig. 2
Cablaggio del
circuito.

Tabella 1 - Collegamento delle periferiche.

Periferica	Pin Fishino	Funzione
sensore temperatura	2	misura temperatura
relé uscita 1	3	luci
relé uscita 2	4	ventilatore
relé uscita 3	5	stufetta

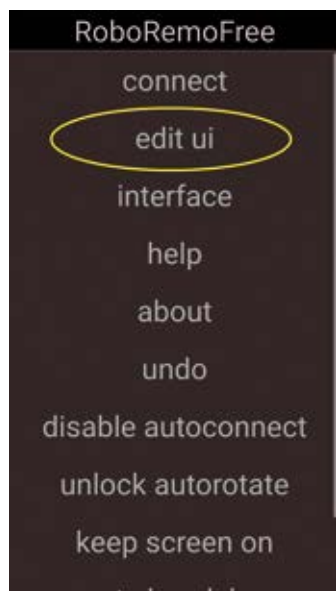


Fig. 3 - RoboRemo: abilitazione editing interfaccia.

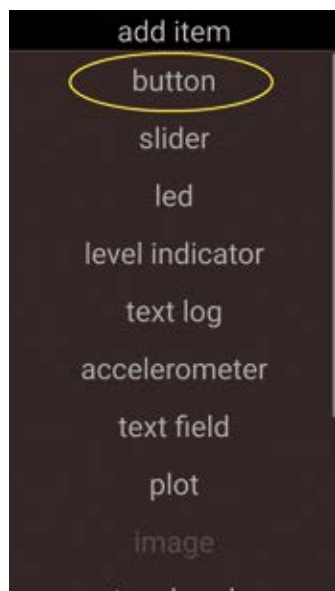


Fig. 4 - RoboRemo aggiunta di un pulsante.



Fig. 5 - RoboRemo opzioni del widget button.



Fig. 6 - Azione svolta dal pulsante quando viene tappato.

ai 5V, il filo nero a GND ed il filo giallo, del segnale, all'ingresso 2 di Fishino. Per il corretto funzionamento, il sensore richiede una resistenza di pull-up del valore di 10 kohm 1/4 di watt connessa tra il pin 2 ed il pin 5V di Fishino. Con la configurazione proposta potremo attivare tre utilizzatori ed anche misurare la temperatura di una stanza.

Vediamo ora come implementare il controllo con lo smartphone, prendendo in considerazione il fatto che vogliamo un'interfaccia personalizzata che sia di semplice e immediato utilizzo. Scarichiamo l'impiego di un modulo Bluetooth essenzialmente per la scarsa portata e per l'impossibilità di impartire un comando da più dispositivi contemporaneamente. Sfruttiamo invece la connessione WiFi nativa della scheda Fishino, considerando che oramai in ogni abitazione è presente una rete WiFi privata, che può far da ponte a tutte le apparecchiature connesse. Una prima soluzione sarebbe quella di implementare un Web Server su Fishino, cosa relativamente semplice da fare, ma dovremmo

accettare gli inevitabili tempi di latenza intrinseci della comunicazione con protocollo html. È quindi preferibile prevedere un'applicazione nativa in grado di garantire tempi di esecuzione pressoché immediati; possiamo trovarla nel Play Store Google ed è un'app che fa proprio al caso nostro perché, senza scrivere alcuna riga di codice, permette di creare la propria personale interfaccia, da utilizzare in tutti quei progetti in cui è richiesto un controllo remoto. L'app si chiama RoboRemo e integra un editor (in-app) a widget con il quale è possibile creare un'interfaccia custom ed assegnare ad ogni elemento un'azione rivolta ad un dispositivo remoto. RoboRemo supporta comunicazioni Bluetooth, TCP, UDP e persino USB, permettendo di sfruttare tutte le periferiche di comunicazioni disponibili in un dispositivo mobile. Sono disponibili i più importanti widget adatti ad interfacce di controllo, ma non manca la possibilità di fare grafici ed inviare comandi a intervalli regolari; il tutto è completamente configurabile senza scrivere una

sola riga di codice.

Il sito di riferimento dove trovare tutti i dettagli dell'applicazione è <http://www.roboremo.com>.

Scaricate pertanto RoboRemo-Free da Play Store (ovvero, se avete installata un'app che identifica ed esegue i QR-Code, leggete il codice in Fig. 12) e installatela; la versione free è completamente gratuita e non richiede alcun tipo di registrazione, però è limitata a cinque widget (senza contare il pulsante menu e i text field), più che sufficienti per la nostra applicazione. La versione a pagamento non ha questa limitazione.

Appena avviata l'applicazione, ci ritroveremo la pagina di lavoro vuota nella quale troviamo il solo pulsante **menu**; tappiamo pertanto su di esso e abilitiamo la creazione dell'interfaccia (Fig. 3) toccando **edit ui**; a questo punto potremmo premere su un punto qualsiasi dello schermo e si aprirà l'elenco delle widget dal quale sceglieremo **button** (Fig. 4). L'elemento scelto sarà inserito nella schermata; ponendo il dito sull'angolo alto sinistro e trascinandolo (funzione drag-

Tabella 2 - Widget e loro parametri.

Funzione	tipi di widget	set text	set press action	set id
richiesta temperatura	button	TEMP	TEMP	---
testo informativo	text	?	---	TEXT
accensione luci	button	LUCI	OUT1	---
accensione ventilatore	button	VENTILATORE	OUT2	---
accensione stufa	button	STUFETTA	OUT3	---
spia accensione	LED	---	---	LED

and-drop) potremo decidere la posizione, invece ponendo il dito sull'angolo in basso a destra e trascinandolo potremo ridimensionarlo. A questo punto premete sul pulsante appena inserito, allorché si apre un menu che permetterà di scegliere sia la grafica che le funzioni dell'oggetto; a noi interessa impostare il testo del pulsante tramite la proprietà **set text** e impostare l'azione svolta quando si clicca sopra, con la proprietà **set press action** (Fig. 5). Quest'ultima conterrà il testo inviato al dispositivo remoto quando si verifica l'evento impostato (Fig. 6).

Fate riferimento alla **Tabella 2** per inserire tutti gli oggetti necessari, con le relative impostazioni. Per impostazione predefinita, la stringa inviata sarà terminata con

il carattere di ritorno a capo “\n” eventualmente modificabile in caso di necessità. Anche se all'inizio può sembrare complicato, in pochi minuti si prende confidenza e si apprezza la semplicità con la quale si può realizzare l'interfaccia.

Terminato il lavoro ritornate sul menu principale e cliccate su **don't edit ui** per uscire dalla modalità di editing; se volete salvare il lavoro usate la funzionalità **interface-export**. Il testo che compare visibile in ogni pulsante (proprietà **set text**) può essere modificato a piacere; per tutti gli altri parametri fate attenzione a rispettare le indicazioni riportate, almeno per ora.

L'interfaccia utente completata della nostra app risulta graficamente come in Fig. 7.

Il passo successivo è scrivere lo sketch per Fishino, in modo da coordinare tutte le operazioni con l'applicazione RoboRemo. Per prima cosa scaricate ed installate i driver e la libreria per Fishino (<http://fishino.it/download>), per il progetto servono anche le librerie OneWire e DallasTemperature, queste ultime gestibili direttamente dall'IDE di Arduino (dalla versione 1.6.6) accedendo alla sezione *sketch>inclusion librerie>gestione librerie*.

Quando è tutto pronto, potete aprire lo sketch scritto appositamente per questo applicativo (il file si chiama *SmartHomeSystems.ino*) che è quasi pronto; l'unica aggiunta da fare è l'inserimento del nome (MY_SSID) e della password (MY_PASS) per l'accesso alla vostra rete WiFi. Fishino dovrà avere un indirizzo IP statico all'interno della vostra rete che non vada in conflitto con altri dispositivi; il valore da controllare ed eventualmente reimpostare è alla riga `#define IPADDR 192,168,1,242`. Questo indirizzo è importante perché sarà quel-



Fig. 7 - Interfaccia completa.



Fig. 8 - RoboRemo: configurazione comunicazione.



Fig. 9 - RoboRemo: impostazione indirizzo IP.



Fig. 10 - RoboRemo: comando luci

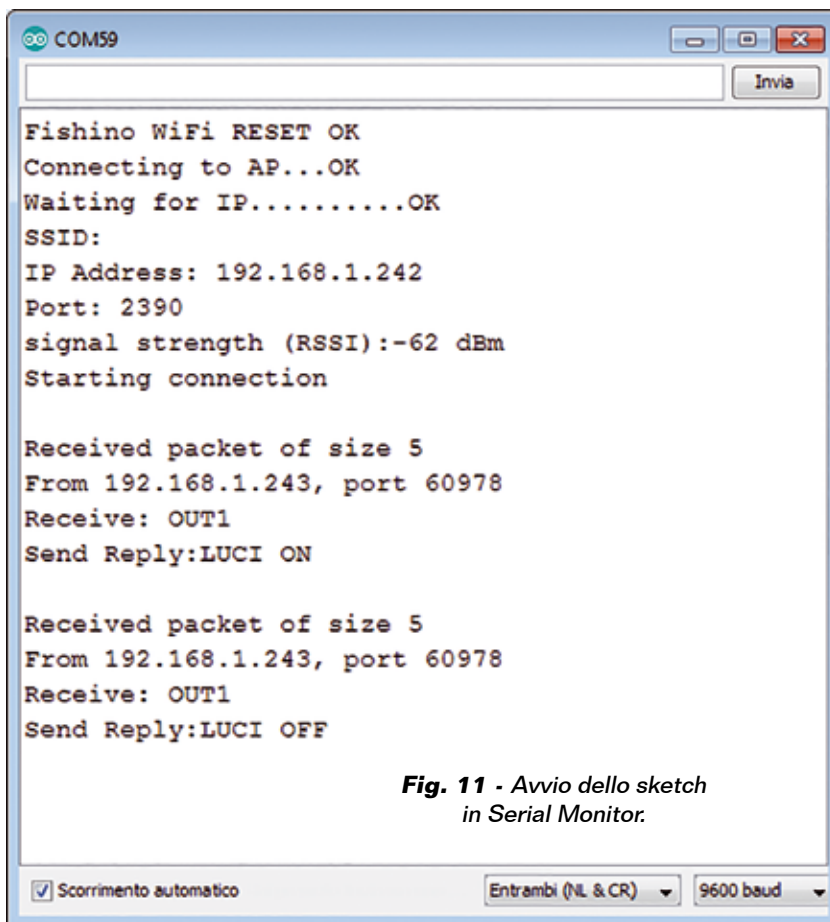


Fig. 11 - Avvio dello sketch in Serial Monitor.



Fig. 12 - QR-Code per scaricare RoboRemo dal Play Store Android.

lo usato da RoboRemo per la comunicazione e deve rimanere costante; ecco perché deve essere di tipo statico e non impostato dal DHCP del router, altrimenti potrebbe cambiare ogni qualvolta Fishino si connette alla rete impedendo la connessione e l'utilizzo dell'app.

Caricate lo sketch su Fishino ed aprite Serial Monitor per verificare l'avvenuta connessione alla rete, poi ritornate su RoboRemo e, nel menu principale, cliccate su **connect**, quindi (Fig. 8) selezionate **Internet (UDP)** e poi andate su **other** ed inserite l'indirizzo di Fishino (compresa la porta), che nel nostro esempio è 192.168.1.242:2390 (Fig. 9). L'interfaccia appena realizzata è valida indipendentemente dalla modalità di comunicazione che verrà impostata successivamente, secondo necessità; ricordatevi

che è possibile impostare anche la funzione di autoconnect in modo che RoboRemo si connetta automaticamente appena avviata, con i parametri dell'ultima connessione impostata.

Cliccate ad esempio sul pulsante LUCI (Fig. 10) e verificate su SerialMonitor di Arduino che i messaggi viaggino correttamente e che la relativa funzione venga svolta; nella schermata viene riportato l'indirizzo del mittente, cioè quello dello smartphone che state utilizzando, nonché il messaggio inviato (Fig. 11). Nella casella di testo di RoboRemo dovrà giungere il messaggio di risposta con la conferma dell'avvenuta esecuzione del comando; premendo su TEMP verrà mostrata la temperatura misurata.

Possiamo descrivere le parti essenziali dello sketch in modo da

permettere a tutti di eseguire delle modifiche, in quanto il progetto si presta ad essere personalizzato per le più svariate esigenze. La prima parte del programma ricalca l'esempio *FishinoUdpSendReceiveString.ino*; abbiamo solo aggiunto la gestione del sensore di temperatura e le uscite digitali. Il ciclo principale si occupa essenzialmente di attendere l'arrivo di messaggi UDP sulla porta 2390 specificata nello sketch; non appena questi arrivano, vengono codificati e resi disponibili al resto dello sketch. RoboRemo invia una stringa diversa a seconda dell'azione svolta; nel nostro caso il tap su uno dei quattro pulsanti invia le stringhe TEMP, OUT1, OUT2 e OUT3 (seguite dal carattere terminatore), per cui dovremo semplicemente identificare la stringa arrivata ed eseguire il comando corrispondente. In ogni caso sarà inviato un messaggio di ritorno a conferma della corretta ricezione del messaggio. Ad esempio, quando premiamo su LUCI viene inviata la stringa "OUT1" e di conseguenza attiviamo, in modalità toggle, l'uscita relativa alle luci (pin 3 di Fishino); a seconda che l'uscita venga attivata o disattivata, il messaggio di ritorno sarà "LUCI ON" oppure "LUCI OFF", che sarà poi visibile nel campo text di RoboRemo (vedete, a riguardo, il Listato 1). Nello stesso modo gestiremo le altre due uscite. Nel Listato 2

Listato 1

```
if(strcmp(cmd, "OUT1\n")==0) //roboremo LUCI button
{
    if (OUT1_status==0)
    {
        OUT1_status=1;
        digitalWrite(OUT1_pin, HIGH);
        sendcmd = "LUCI ON\n";
    }
    else
    {
        OUT1_status=0;
        digitalWrite(OUT1_pin, LOW);
        sendcmd = "LUCI OFF\n";
    }
}
```

Listato 2

```
Udp.beginPacket(Udp.remoteIP(), localPort);
Udp.print("TEXT ");
Udp.print(sendcmd);
Udp.print("\n");
Udp.endPacket();
```

Listato 3

```
if(strcmp(cmd, "TEMP\n")==0) //roboremo richiesta temperatura
{
    Serial.println("Request temperature...");
    sensors.requestTemperatures(); // Lettura sensore
    float Temp=sensors.getTempCByIndex(0);
    Serial.print("Temperature="); // eco su serial monitor
    Serial.print(Temp,1);
    Serial.write(176);
    Serial.println("C");
    char tmp[5];
    dtostrf(Temp,2,1,tmp); //Conversione da float a stringa
    sendcmd = sendcmd + tmp + "°C\n";
}
```

potete vedere la parte dello sketch che si occupa della risposta: si tratta essenzialmente di inviare una stringa che inizia con il valore *id* del campo testo di RoboRemo (*id*=TEXT) seguita dal testo da visualizzare e terminata con il carattere "\n". Allo stesso modo gestiremo l'accensione del LED inviando la stringa "LED 1\n" ed il suo spegnimento inviando la stringa "LED 0\n". La lettura della temperatura è altrettanto semplice e prevede di identificare la stringa TEMP inviata dal relativo pulsante, quindi leggere la temperatura dal sensore DS18B20, successivamente

te comporre la stringa di risposta contenente il valore di temperatura (riferitevi al Listato 3).

La stringa di ritorno viene inviata all'indirizzo IP del dispositivo che ha trasmesso per ultimo sfruttando la funzione **Udp.remoteIP()**, che riporta proprio l'indirizzo IP relativo all'ultima ricezione, la funzione **Udp.remotePort()** sembra invece non funzionare correttamente, però RoboRemo è in attesa nella stessa porta indicata per la trasmissione e quindi LocalPort e RemotePort coincidono. Nel nostro esempio la porta di comunicazione è la 2390, quindi sia Fishino che RoboRemo sono

in attesa sulla porta suaccennata; l'unico parametro che cambia è l'indirizzo IP del mittente (così più dispositivi possono comunicare con Fishino). Tale funzionalità è permessa dall'utilizzo del protocollo di comunicazione UDP, che non prevede una connessione diretta tra un server ed un client, ma nel quale tutti i dispositivi sono paritari. Il meccanismo che sta alla base del funzionamento dei messaggi è molto semplice e ben implementato e risulta così flessibile da potersi adattare praticamente a qualsiasi controllo; inoltre, a livello di programmazione non vi sono particolari difficoltà. Considerate che è possibile visualizzare grafici e immagini e che la versione free è -tutto sommato- sufficiente per piccoli impianti, inoltre la flessibilità di poter utilizzare interfacce di comunicazione sia Bluetooth che USB permette una notevole flessibilità. ■



per il MATERIALE

Tutti i componenti necessari per realizzare questo progetto possono essere acquistati presso Futura Elettronica. La board Fishino UNO (cod. FISHINOUNO) è in vendita al prezzo di Euro 36,00; la sonda di temperatura waterproof con DS18B20 (cod. SONDADS18B20) è disponibile a Euro 7,00; il modulo a 1 relè 5Vdc 10A (cod. RELAY1CH) costa Euro 5,00, mentre l'alimentatore switching 5Vdc/3A con uscita micro USB (cod. AL05-3USB) si può acquistare a Euro 10,50.

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

Ricordate il progetto **Fish'n Tweets**, che permetteva di controllare gli I/O di Fishino tramite dei tweet? Ebbene, si è trattato del nostro primo esperimento di interazione tra Fishino e il mondo dei Social. L'applicativo è tutt'ora valido ma, a causa della "lentezza" di Twitter, a volte il tempo intercorso tra un comando e la sua esecuzione può superare i 20 secondi.

Twitter inoltre è piuttosto complesso da gestire, soprattutto nella fase iniziale di creazione delle chiavi di accesso, dell'utente "robot", eccetera.

Visto che volevamo tornare sull'argomento e che San Valentino incombe, abbiamo pensato di prendere i classici "due piccioni con una fava" realizzando un'applicazione, -stavolta basata sul servizio di instant messaging Telegram- che può essere un modo suggestivo per mandare messaggi personalizzati alla propria amata, stampati da una micro-stampante. I messaggi possono essere sia personalizzati che scelti da un database predisposto su un server. L'applicazione

qui proposta non è limitata al giorno degli innamorati ma trova impiego nella realtà quotidiana, perché implementa anche una funzione di gestione della "lista della spesa", che permette di memorizzare una sequenza di prodotti, inviandone il nome tramite un messaggio Telegram in qualsiasi momento e permettendo di consultarla e/o stamparla, sempre da remoto. Ciò trova applicazione ad esempio in un ristorante, dove l'avventore può fare il proprio ordine, che verrà stampato alla cassa in automatico. L'applicazione ci permette di introdurre una libreria di interfaccia con Telegram appositamente sviluppata, che ci consentirà non solo di controllare Fishino da qualsiasi parte del mondo, ma anche di leggerne gli ingressi e, se vogliamo, di farci avvisare in caso si verifichino eventi particolari, sempre tramite un messaggio Telegram. La libreria è dotata dei necessari strumenti di "sicurezza": è possibile lasciare l'accesso a tutti, vincolarlo solo ad utenti specifici oppure, se necessario, lasciar-



lo aperto a tutti salvo alcuni utenti "disturbatori". Come vedrete la libreria si presenta in modo piuttosto modulare e semplice da usare, nascondendo all'utente i dettagli tecnici dell'implementazione e della comunicazione con Fishino. Questi applicativi possono funzionare anche con una scheda Arduino dotata di shield WiFi, tuttavia dipende dallo shield. Quello più comunemente disponi-

bile (ed economico) non supporta la connessione sicura (SSL/HTTPS), quindi non è utilizzabile per l'interfacciamento con i social che la richiedono, come Twitter, Telegram ed altri. Disponendo, invece, di uno shield più recente, è possibile adattare la libreria ad esso.

TELEGRAM E I SUOI "BOT"

Cos'è un "bot"? In sintesi, è un account "virtuale" di Telegram,

NOTES MACHINE CON FISHINO

di MASSIMO DEL FEDELE

Appoggiandoci a un bot, stampiamo con una stampantina termica remota messaggi composti su Telegram. Una soluzione dalle mille applicazioni, utilizzabile per stampare liste e ordini in un locale e, perché no, per mandare messaggi stampati alla propria "bella" il giorno di San Valentino e non solo.

per creare il quale non servono numeri di telefono, dati personali, eccetera. Questo tipo di account deve essere creato da un account "reale"; per il resto, può funzionare come un utente simulato, ricevere ed inviare messaggi, ed altro. A noi interessa qualcosa che possa interagire sia con il nostro Fishino che con uno o più utenti; creeremo quindi un bot che verrà controllato

da Fishino e che sarà in grado di rispondere alle nostre richieste e/o inviarci messaggi. Il bot è quindi un robot, programmabile per eseguire determinate azioni. I bot:

- non mostrano il loro stato (online/offline); sono sempre attivi e "vedono" all'istante (o quasi) i nostri messaggi;
- hanno una memoria limitata; i vecchi messaggi possono

- sparire dal server poco dopo essere stati processati;
- non possono iniziare una conversazione ma rispondono se interessati; per comunicare con un bot occorre aggiungerlo ad un gruppo oppure inviargli un messaggio iniziale;
- hanno un "nome utente" che finisce sempre in "bot"; per esempio, **TriviaBot**, **Fishino_bot**;
- anche se aggiunti a un gruppo, non ricevono tutti i messaggi se non impostati allo scopo.

CREIAMO IL NOSTRO BOT

Iniziamo subito con la creazione del bot che ci servirà per interagire con Fishino.

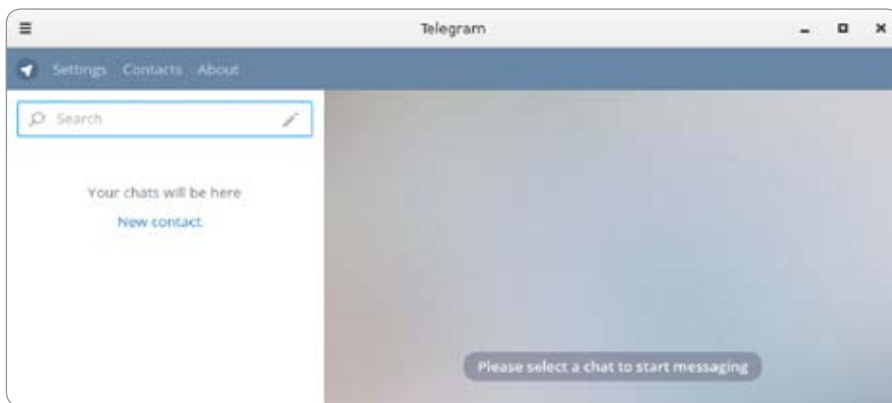
Per prima cosa, se non l'avete ancora fatto, occorre scaricare ed installare Telegram; nelle immagini in queste pagine vedete l'installazione su un desktop Linux (l'abbiamo fatto per catturare le schermate), ma normalmente l'applicazione è installata su uno smartphone. Nell'installazione vi verrà chiesto di confermare il vostro numero di telefono e/o un codice di sicurezza, a

seconda della piattaforma. Una volta installato ed eseguito, vi troverete di fronte alla schermata di **Fig. 1**. Per creare il bot bisogna ricorrere al "padre di tutti i bot", ovvero all'utente BotFather: è sufficiente digitare il nome nella casella di ricerca (**Fig. 2**). Vi apparirà la schermata introduttiva di BotFather; cliccate su **Start** (in basso) per avviare il bot; BotFather risponderà con un messaggio esplicativo contenente una lista dei principali comandi disponibili (**Fig. 3**).

Siccome vogliamo creare un nostro bot, clicchiamo sul comando **/newbot** (**Fig. 4**).

BotFather ci chiede di dare un nome al nostro bot; scegliamo un nome a caso, per esempio **Pippo** (**Fig. 5**).

Ci viene quindi chiesto uno "user name" per il nostro bot; questo deve per forza essere unico e terminare in "bot"; nel caso scegliessimo uno username già occupato BotFather ci avviserà e ci imporrà di cambiarlo. In questo caso PippoBot è già in uso (ovviamente!), quindi sceglieremo **PippoRobot** che casualmente è libero; BotFather ci avviserà



dell'avvenuta creazione del nostro bot e ci darà le istruzioni per utilizzarlo da remoto (**Fig. 6**). Come potete notare, il nostro bot è accessibile tramite l'indirizzo Telegram:

telegram.me/PippoRobot

e per controllarlo è necessario possedere il token di sicurezza fornito da **BotFather**:

Use this token to access the HTTP API:
nnnnnnnnnn:XXXXXXX-
YYYYYY_
ZZZZZZZZZZZZZZZZZZZZZZZZ

Il token (qui oscurato) è indispensabile per garantire la sicurezza che nessuno possa “manomettere” il nostro bot. Questo token andrà inserito nel codice del nostro sketch, come vedremo in seguito, quindi prendetene nota e, soprattutto, mantenetela ben segreta visto che chiunque ne viene in possesso è in grado di fare qualsiasi cosa con il nostro bot, compreso cancellarlo. Clicchiamo ora sul link *telegram.me/PippoRobot* (**Fig. 7**) e successivamente sul comando **Start**, che avvierà il nostro bot inviandogli un messaggio iniziale (**Fig. 8**). Ecco fatto! Abbiamo creato il nostro primo bot di Telegram!

LA LIBRERIA FISHGRAM

Una volta creato il nostro bel-

lissimo bot occorre, perché sia utile, potergli inviare e ricevere messaggi, oltre che dall'app Telegram, anche attraverso il nostro Fishino. Per interagire con il bot Telegram utilizza il protocollo HTTPS con un formato ben specifico; in particolare, i comandi possono essere impartiti tramite direttive HTTP GET e POST, e negli headers http inviati deve sempre essere presente il nostro token. Per esempio, una richiesta di tipo `'getUpdates'`, che ci

permette di richiedere i messaggi ricevuti dal bot ha il formato seguente:

```
GET /botTOKEN/getUpdates?offset=
nnn&timeout=4&limit=1&allowed_
updates=messages HTTP/1.1
User-Agent: FishGram 1.0.0
Host: api.telegram.org
```

In questo formato di richiesta:

- **TOKEN** è il nostro token di accesso, visto al paragrafo precedente;
- **offset=nnnn** rappresenta il primo ID di messaggio cui siamo interessati;
- **limit=1** indica a Telegram di fornirci un solo messaggio per richiesta (Telegram può fornire più messaggi per ogni richiesta di aggiornamento);
- **allowed_updates=messages** indica che siamo interessati solo ai messaggi e non, per esempio, ai file audio e/o immagini.

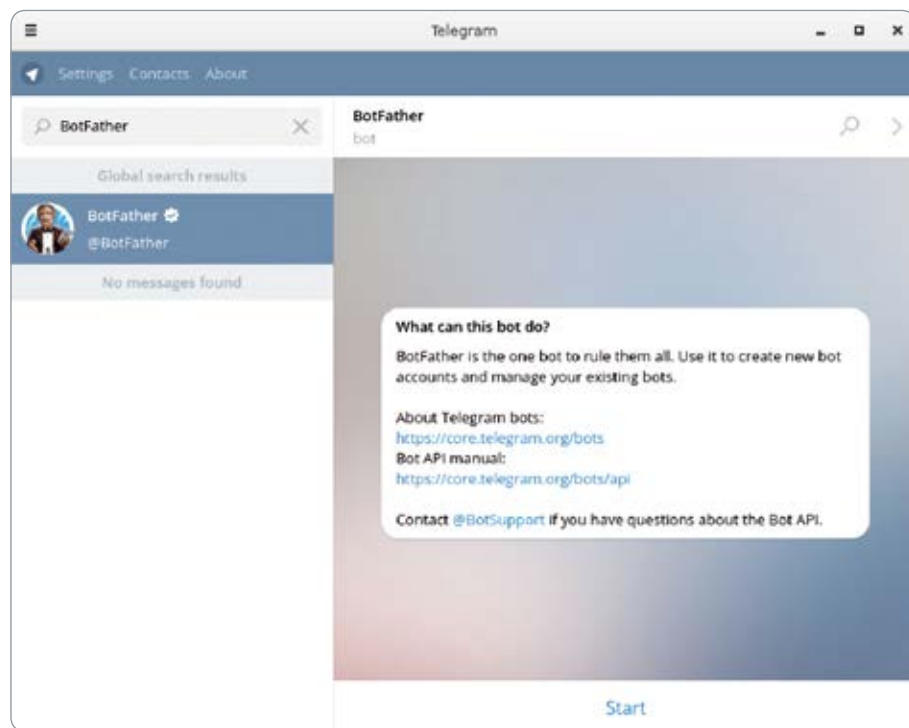


Fig. 2 - Ricerca di BotFather.

Possiamo provare il comando direttamente sul browser, immettendo nella casella dell'indirizzo il testo seguente (sostituire il TOKEN con il vostro token di accesso):

https://api.telegram.org/botTOKEN/getUpdates?offset=0&timeout=4&limit=1&allowed_updates=messages

Scrivete, ovviamente, tutto su una sola riga. Telegram risponderà con un testo in formato JSON come il seguente:

```
{ "ok": true, "result": [ { "update_id": 904783368,
  "message": { "message_id": 2, "from": { "id": nnnnnnnnn, "first_name": "Massimo", "last_name": "Del Fedele", "chat": { "id": nnnnnnnnn, "first_name": "Massimo", "last_name": "Del Fedele", "type": "private", "date": 1483380231, "text": "ciao Pippo" } } } ] }
```

Come si può notare, il testo

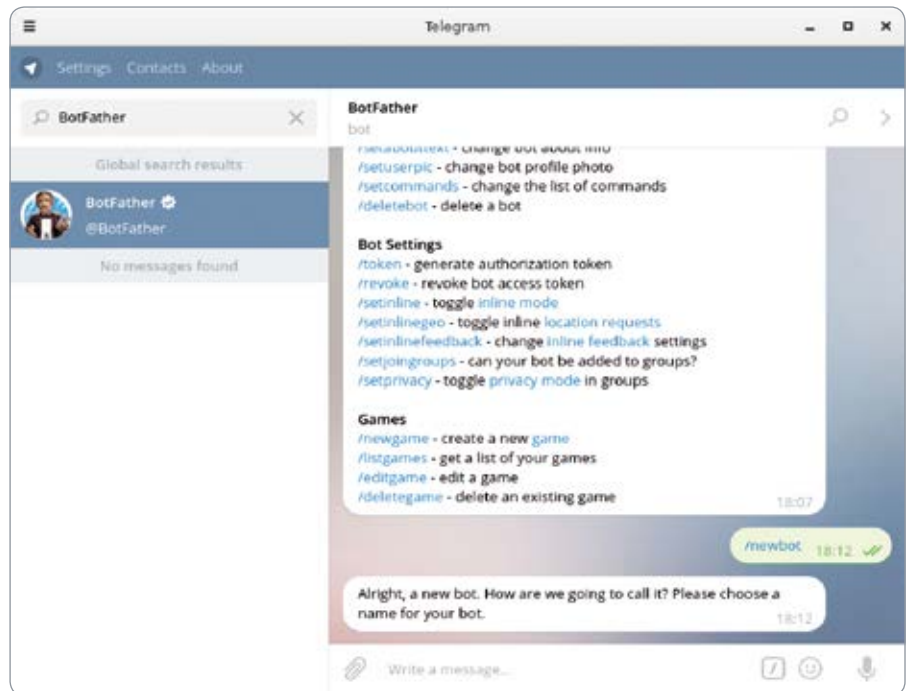


Fig. 4 - Creazione del nostro Bot.

di risposta contiene parecchie informazioni, tra cui il mittente del messaggio, l'id del messaggio (tramite il quale, per esempio, in una chiamata successiva invian-

do come **offset=id+1** potremo richiedere i messaggi a partire dal successivo), eccetera. È facile capire che, senza un'apposita libreria software, il funzionamento della cosa risulta piuttosto complesso. Se l'invio della richiesta è relativamente semplice (basta utilizzare un oggetto FishinoClient, collegarlo all'indirizzo richiesto, ed inviare una serie di stringhe di caratteri), l'analisi del JSON restituito non è altrettanto banale, soprattutto a causa delle risorse limitate del nostro Fishino, specialmente per quanto riguarda la memoria RAM.

Per questo ci viene in aiuto la libreria **JSONStreamingParser**, scritta per il progetto Fish'n Tweets, che è in grado di "digerire" l'output di Telegram, carattere per carattere, senza bisogno di memorizzare nulla, e di spezzettarlo in tanti elementi del tipo **nome: valore** e, ad ogni elemento ricevuto, chiamare una funzione da noi definita.

Siccome spiegare il funzionamento della libreria richiederebbe

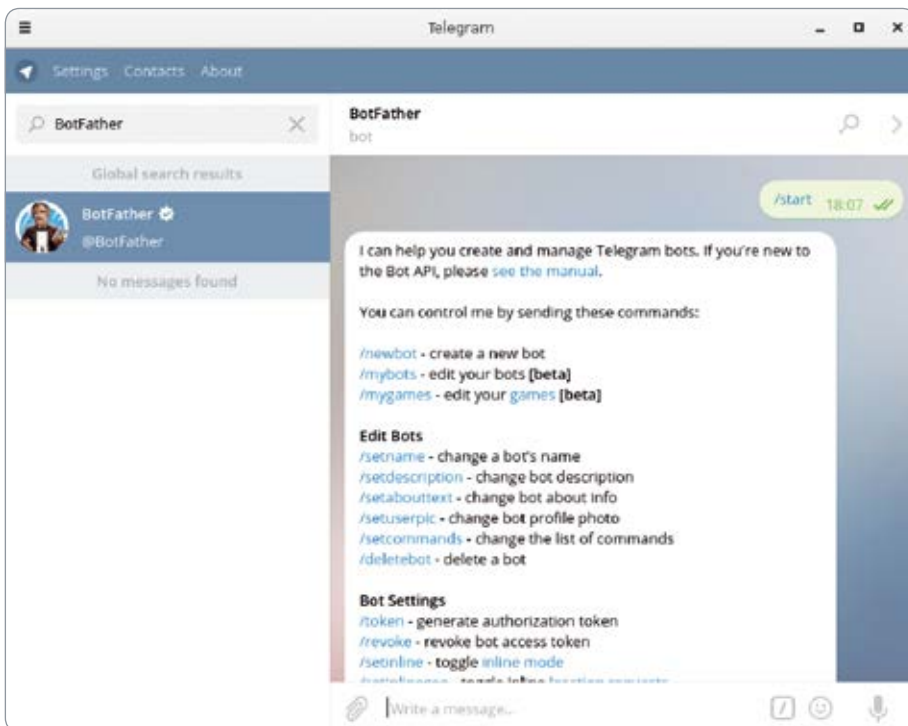


Fig. 3 - Risposta di BotFather.

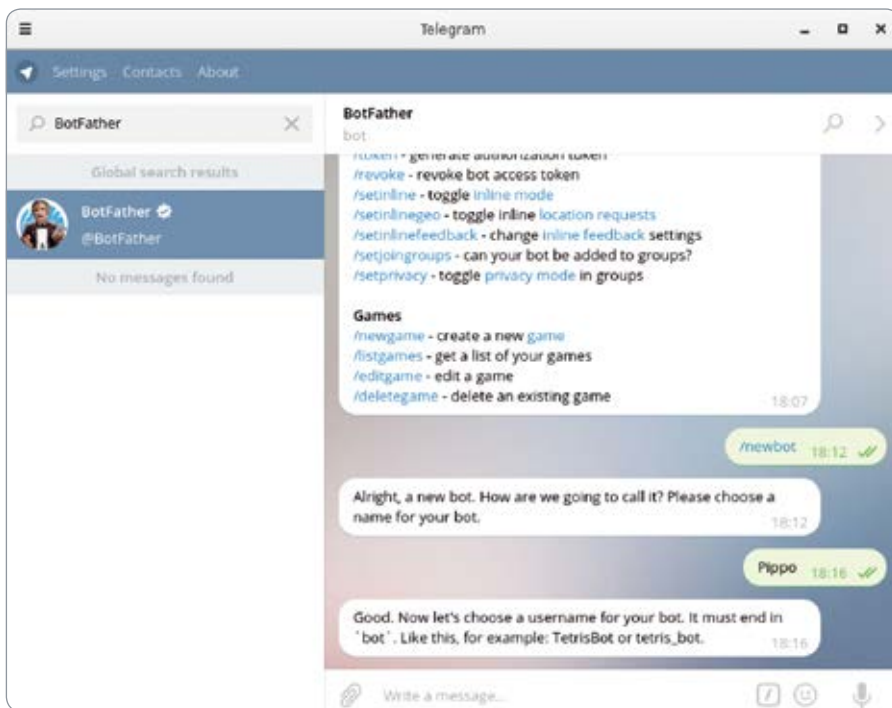


Fig. 5 - Assegnazione del nome al Bot.

metà delle pagine di questo fascicolo, se non oltre, partiremo da un piccolo sketch di esempio per mostrarne l'utilizzo; chi volesse approfondire troverà il codice della libreria stessa molto commentato e quindi relativamente facile da capire. Affronteremo comunque, durante la spiegazione dell'utilizzo, di alcune peculiarità e scelte apparentemente strane che abbiamo effettuato nella scrittura della stessa.

UN PRIMO SKETCH, SEMPLICE SEMPLICE

Iniziamo... dall'inizio! Proviamo a visualizzare i messaggi che vengono inviati al nostro bot sul monitor seriale; per far questo lo sketch è davvero semplicissimo. Per questo (primo) esempio, inseriremo il codice completo (compresa l'inizializzazione di Fishino, la connessione al router WiFi, eccetera, in modo da permettere a chiunque di collaudarlo "al volo" semplicemente copiando il codice nell'IDE; per i prossimi esempi eviteremo tutto questo ed inseriremo solo il codice specifico

di **FishGram** (Listato 1).

Come potete vedere, il grosso dello sketch è l'inizializzazione di Fishino, il collegamento al router, eccetera. Per quanto riguarda FishGram in pratica troviamo

rilevanti tre punti, il primo dei quali è costituito dalla funzione di gestione degli eventi (di cui parleremo tra poco):

```
bool FishGramHandler(uint32_t id,
const char *firstName, const char *lastName, const char *message)
```

Il secondo è "inizializzazione di FishGram":

```
// start FishGram
FishGram.
event(FishGramHandler);
FishGram.begin(F(MY_TELEGRAM_TOKEN));
```

E il terzo è la chiamata nel loop:

```
FishGram.loop();
```

Semplice, vero? Ma come funziona? Ebbene, a differenza della maggior parte degli sketch cui siamo abituati, la libreria FishGram funziona per eventi; una volta inizializzata, resta quasi

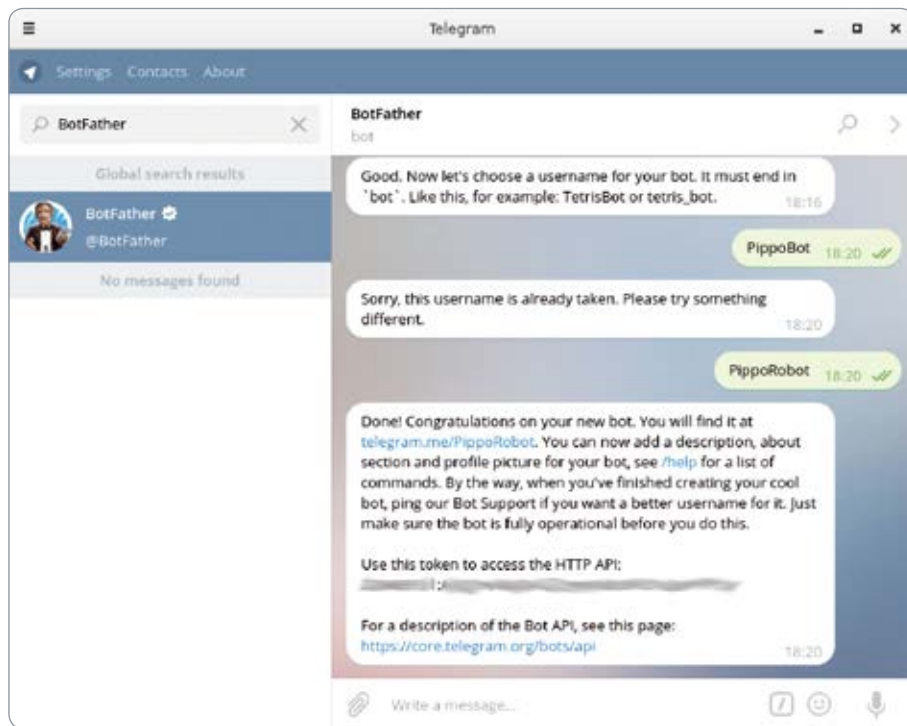


Fig. 6 - Completamento del bot.

invisibile e, soprattutto, occupa pochissime risorse di calcolo e di memoria. L'importante è che la chiamata `FishGram.loop()`; dentro al loop venga eseguita spesso.

Questa chiamata, che normalmente dura poche decine di microsecondi, consente alla libreria di gestire un timer interno e di richiamare, ad intervalli prefissati, il server di Telegram e richiedere eventuali aggiornamenti.

Se questi aggiornamenti non sono presenti, torna senza fare nulla; se invece c'è qualche novità la raccoglie, organizza e chiama la funzione evento `FishGramHandler()` passandole come parametri l'id, il nome ed il cognome del mittente ed il testo del messaggio.

Con questa funzione possiamo quindi stampare sulla porta seriale quello che riceviamo.

Le due chiamate nel setup, ovvero la `FishGram.event()` e la `FishGram.begin()`, servono

rispettivamente per collegare la nostra funzione di gestione degli eventi e per dire a FishGram chi è il nostro bot tramite il token. Non saremo quindi noi, dentro allo sketch, a doverci ricordare di

chiedere a FishGram gli aggiornamenti "ogni tanto", ma con questo meccanismo sarà la stessa libreria ad avvisarci se e solo se arriva qualcosa di interessante dal nostro bot.

Quali sono gli svantaggi di un simile metodo? In pratica ce n'è solo uno: se la `FishGram.loop()` non viene eseguita abbastanza spesso il programma risponde male. Per esempio, se infilassimo una `delay(1000)` dentro alla `loop()`, la libreria riuscirebbe ad elaborare solo un carattere ricevuto ogni secondo, quindi passerebbero giornate prima di avere una risposta, o, più probabilmente, il server di Telegram chiuderebbe la connessione prima.

Quindi è ben possibile (anzi, lo scopo di questo sistema è proprio quello!) far eseguire al nostro Fishino qualcos'altro, mentre è in attesa di messaggi, ma questo qualcos'altro non deve né bloccare l'esecuzione del `loop()` né impiegare troppo tempo. Quindi, niente `delay()`, niente `while()` con durate lunghe e/o attesa di pul-

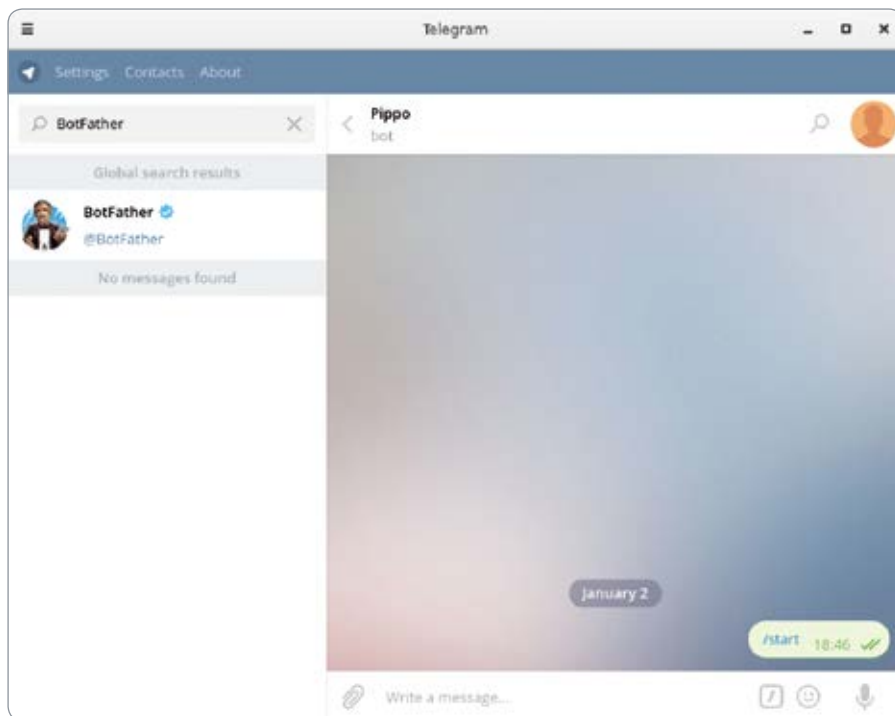


Fig. 7 - Avvio del bot.

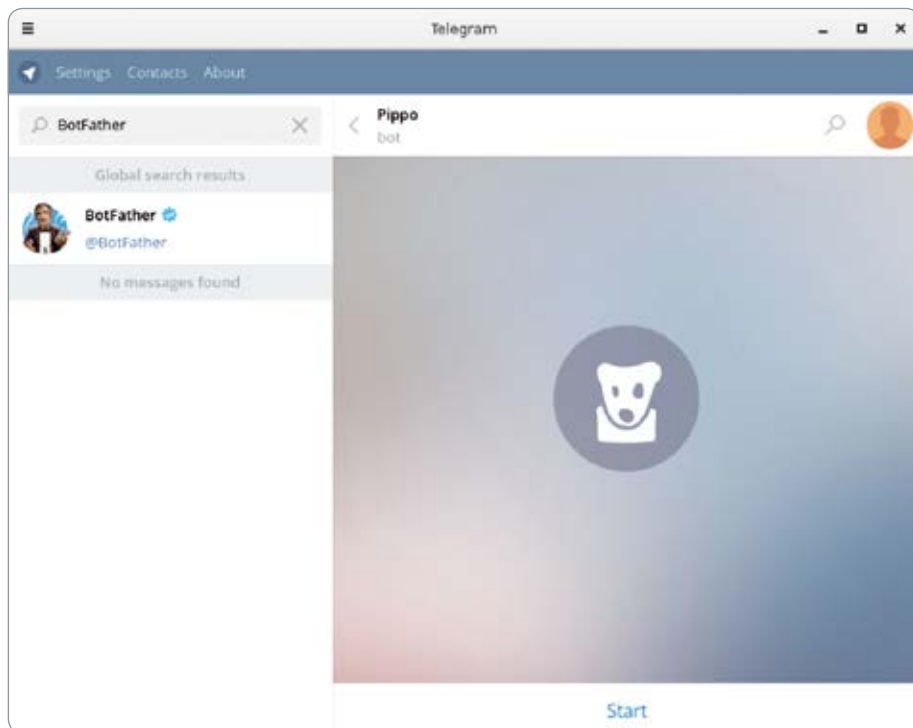


Fig. 8 - Messaggio iniziale del bot.

Listato 1

```
#include <Fishino.h>
#include <JSONStreamingParser.h>
#include <FishGram.h>
#include <SPI.h>

#define MY_SSID    "IL_MIO_SSID"
#define MY_PASS    "LA_MIA_PASSWORD"

// se volete un IP dinamico basta commentare/eliminare le 3 righe seguenti
#define IPADDR     192, 168, 1, 251
#define GATEWAY    192, 168, 1, 1
#define NETMASK    255, 255, 255, 0

#define MY_TELEGRAM_TOKEN "LA_TOKEN_DEL_MIO_BOT_TELEGRAM"

#ifdef IPADDR
    IPAddress ip(IPADDR);
    #ifdef GATEWAY
        IPAddress gw(GATEWAY);
    #else
        IPAddress gw(ip[0], ip[1], ip[2], 1);
    #endif
    #ifdef NETMASK
        IPAddress nm(NETMASK);
    #else
        IPAddress nm(255, 255, 255, 0);
    #endif
#endif

// fishgram event handler -- just display message on serial port
bool FishGramHandler(uint32_t tid, const char *firstName, const char *lastName, const char *message)
{
    Serial << F("Ho ricevuto un messaggio da ") << firstName << "\n";
    Serial << F("Il messaggio dice: ") << message << "\n";
    return true;
}

void setup(void)
{
    Serial.begin(115200);
    while (!Serial)
        ;

    while(!Fishino.reset())
        Serial << F("Fishino RESET FAILED, RETRYING...\n");
    Serial << F("Fishino WiFi RESET OK\n");

    Fishino.setMode(STATION_MODE);
    Fishino.setPhyMode(PHY_MODE_11G);

    Serial << F("Connecting to AP...");
    while(!Fishino.begin(MY_SSID, MY_PASS))
    {
        Serial << ".";
        delay(2000);
    }
    Serial << "OK\n";

#ifdef IPADDR
    Fishino.config(ip, gw, nm);
#else
    Fishino.staStartDHCP();
#endif

    Serial << F("Waiting for IP...");
    while(Fishino.status() != STATION_GOT_IP)
    {
        Serial << ".";
        delay(500);
    }
    Serial << F("OK\n");

    // start FishGram
    FishGram.event(FishGramHandler);
    FishGram.begin(F(MY_TELEGRAM_TOKEN));
}

void loop(void)
{
    FishGram.loop();
}
```

santi da premere, eccetera. Tutto dev'essere eseguito senza bloccare il loop, quindi con `millis()` per i ritardi e controlli "volanti" per eventuali pulsanti da rilevare. Esistono alternative ma vanificano in parte i vantaggi dell'approccio; ad esempio, se vogliamo attendere che un I/O vada a livello HIGH, invece di fare:

```
while(!digitalRead(5))
    ;
```

possiamo scrivere:

```
while(!digitalRead(5))
    FishGram.loop();
```

In modo che FishGram continui ad essere richiamata nell'attesa. Allo stesso modo, invece di utilizzare:

```
delay(1000);
```

dovremo procedere con l'utilizzo della `millis()` in questo modo:

```
uint32_t tim = millis() + 1000;
while(millis() < tim)
    FishGram.loop();
```

Ma è possibile aggiungere alla libreria una funzione del tipo seguente?

```
FishGram.aspettaMessaggio()
```

Certamente sì... ma se il messaggio non arriva, lo sketch rimane bloccato all'infinito oppure per un tempo prefissato senza comunque poter fare altro, quindi abbiamo appositamente evitato di introdurla.

Sul monitor seriale verrà visualizzato qualcosa di simile:

```
Fishino WiFi RESET OK
Connecting to AP...OK
Waiting for IP.....OK
```

```
Ho ricevuto un messaggio da 'Massimo'
```

Il messaggio dice: 'Ciao Pippo, come stai?'

INVIARE UN MESSAGGIO DA FISHINO A UN UTENTE

Se ricordate, nel paragrafo sulle differenze tra un bot ed un umano era specificato un punto apparentemente poco importante ma che invece è fondamentale: un bot non può iniziare una conversazione. Non possiamo quindi dire al nostro Fishino "cerca Giuseppe e spediscigli un messaggio". Questa è stata una scelta ben precisa di chi ha sviluppato le API di Telegram, per impedire un uso improprio dei bot. Come potete ben immaginare, se ci fosse questa possibilità sarebbe semplicissimo realizzare uno spam-bot in grado di inviare migliaia di messaggi indesiderati a chiunque! Come possiamo fare, quindi? Semplicemente, una volta inviato almeno un messaggio al nostro bot, questo avrà a disposizione l'id della chat (che poi è anche l'id di chi ha inviato il messaggio) e potrà quindi rispondere a questo utente e ad altri eventuali che l'hanno contattato. Vediamo quindi una

Il sistema in versione San Valentino: la stampante è montata in un quadro in tema.



La stampante termica

Per stampare i nostri messaggi inviati da Telegram ci siamo avvalsi di una piccola stampante a carta termica facilmente reperibile in commercio negli store on-line per maker, come www.futurashop.it. La stampantina è del tipo a 20 colonne e stampa su rotolo di carta termica da 57,5 ± 0,5 mm e si interfaccia tramite una connessione seriale a livello TTL. Di seguito ne elenchiamo le caratteristiche.

- tensione di alimentazione: 5÷9 Vcc;
- assorbimento max: 1,5 A;
- velocità di stampa: 50÷80 mm/s
- risoluzione: 8 pixel/mm, 384 pixel/linea
- larghezza effettiva di stampa: 48mm
- set caratteri: ASCII, GB2312-80;
- font stampa: ANK: 5 × 7, cinese: 12 × 24, 24 × 24;
- protocollo: TTL Serial, 19.200

baud;

- dimensioni (LxPxH): 111 x 65 x 57 mm;
- temperatura di esercizio: 5°÷ 50°C.

La stampante è la classica termica simile a molti modelli esistenti da anni, però nasce -guardacaso- per il mondo Arduino; allo scopo viene fornita con a corredo l'apposita libreria firmware.

Nello specifico, nel nostro sketch per Fishino abbiamo integrato e utilizzato la stessa libreria fornita da Adafruit.



piccola modifica allo sketch, che ci permette di ottenere un feedback da parte di FishGram sul nostro telefonino; per far questo è sufficiente modificare la funzione di gestione degli eventi come nel **Listato 2**. E, al prossimo messaggio inviato, il nostro bot, oltre a mostrarci l'output sul monitor seriale, ci risponderà come mostrato in **Fig. 9**.

La funzione **FishGram.sendMessage()**, non dovendo attendere alcunché, si comporta come siamo abituati: viene richiamata dove vogliamo, esegue il suo compito e poi ci lascia proseguire. Il primo parametro, id, è l'id dell'utente a cui si vuole inviare il messaggio; il secondo è il messaggio vero e proprio.

LA LIBRERIA IN DETTAGLIO

Come avete visto nei due esempi precedenti, la libreria è piuttosto semplice; contiene comunque altre utili funzioni che vediamo

qui di seguito. La prima è:

```
// end - termina la libreria
//FishGram
bool end(void);
```

e in pratica non serve mai, a meno di non utilizzare FishGram sporadicamente e voler liberare tutta la memoria che occupa.

```
// cancella i dati locali
FishGramClass &clear(void);
```

in pratica non è indispensabile; libera la memoria utilizzata da FishGram fino al prossimo evento.

```
// abilita/disabilita la lista di
//utenti ammessi
FishGramClass &restrict(bool b =
true);
FishGramClass &noRestrict(void) {
return restrict(false); }
```

queste ultime due funzioni

sono utilissime per eseguire un controllo su chi può mandare messaggi al nostro bot; se attivata con **restrict()** la funzione scarta automaticamente tutti i messaggi provenienti da id non contenuti nella lista al punto seguente.

```
// aggiunge un ID utente alla lista
//di utenti ammessi
FishGramClass &allow(uint32_t id);
```

questa funzione permette di aggiungere un id utente alla lista degli utenti ammessi; se la funzione viene abilitata con **restrict()**, solo gli utenti presenti in questa lista vedranno accolti i loro messaggi dal bot.

```
// aggiunge un ID utente alla lista
//di utenti bloccati
FishGramClass &block(uint32_t id);
```

Nel **Listato 3** invece vediamo la lista degli utenti bloccati; qualsiasi id presente in questa lista viene considerato come spam e quindi ignorato.

All'avvio, di default FishGram legge l'ultimo messaggio disponibile, lo scarta ed attende nuovi messaggi; a volte è desiderabile leggere anche messaggi precedenti all'avvio, cosa ottenibile con questa funzione. Se introduciamo **recoverOldMessages(10)**, ad esempio, all'avvio, FishGram

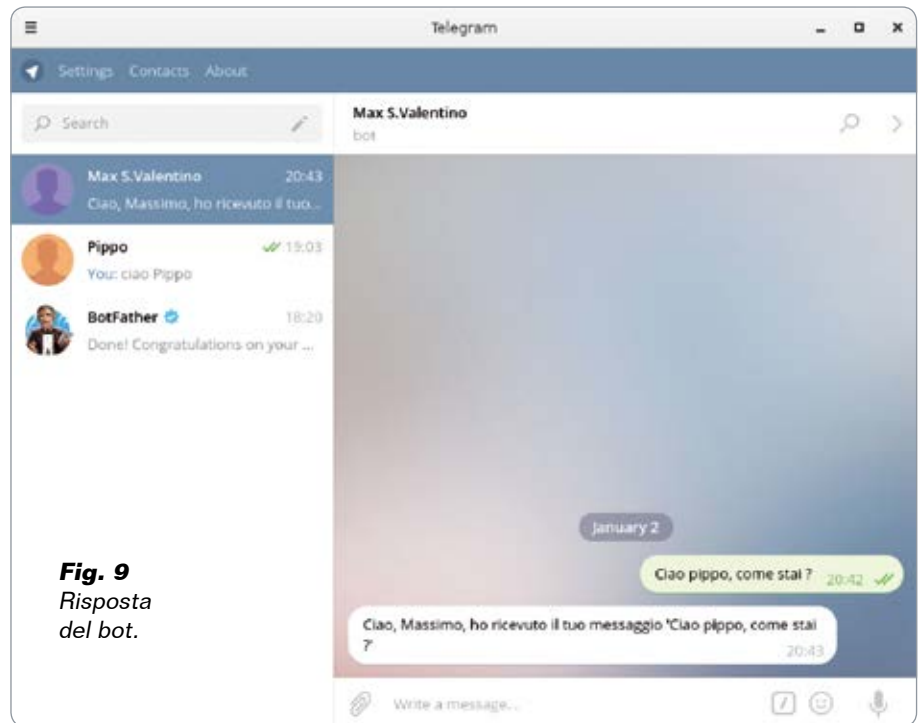


Fig. 9
Risposta
del bot.

recupererà i 10 messaggi precedenti:

```
// imposta la funzione di gestione
//degli eventi
FishGramClass &event(FishGramEvent e);
```

Questa funzione installa il gestore di eventi, come visto in precedenza:

```
// invia un messaggio ad un ID
//specifico
bool sendMessage(uint32_t const &id,
const char *msg);
bool sendMessage(uint32_t const &id,
const __FlashStringHelper *msg);
```

Queste due funzioni permettono di inviare un messaggio tramite **FishGram** a un ID specifico:

```
// invia un messaggio ad un ID specifico pezzo per pezzo
bool startMessage(uint32_t const &id,
uint16_t len);
bool contMessage(const char *msg);
bool contMessage(const __FlashStringHelper *msg);
bool contMessage(char c);
bool endMessage(void);
```

queste cinque funzioni permettono di inviare un messaggio **pezzo per pezzo**, quindi comode se la memoria RAM è scarsa. L'unico "inghippo" è che è necessario conoscere a priori la lunghezza complessiva del messaggio, da passare come parametro alla **startMessage()**. Il messaggio può poi essere inviato tramite varie chiamate alla **contMessage()**, e terminato con la **endMessage()**.

LA NOTE MACHINE

Dopo aver esaminato la libreria **FishGram** in dettaglio possiamo vederne un'applicazione un po' più complessa. In principio le

Listato 2

```
// fishgram event handler -- display message on serial port and give feedback to sender
bool FishGramHandler(uint32_t id, const char *firstName, const char *lastName, const char *message)
{
    Serial << F("Ho ricevuto un messaggio da ") << firstName << "\n";
    Serial << F("Il messaggio dice: ") << message << "\n";

    Strings;
    s = "Ciao, ";
    s += firstName;
    s += ", ho ricevuto il tuo messaggio ";
    s += message;
    s += "\n";
    FishGram.sendMessage(id, s.c_str());
    return true;
}
```

Listato 3

```
// imposta il numero di messaggi precedenti da recuperare
// -1 per TUTTI (usare con cautela!), 0 per nessuno.
// dev'essere chiamata prima della begin()
FishGramClass &recoverOldMessages(uint32_t n = (uint32_t)-1);
```

cose sono abbastanza semplici: basta “trasformare” i messaggi in comandi per il nostro Fishino, eseguire questi comandi ed inviare un’eventuale risposta/conferma al mittente. Come dicevamo nell’introduzione, abbiamo pensato di implementare un sistema che permetta di:

- stampare messaggi inviati da Telegram;
- rispondere ad un messaggio particolare con una citazione o un messaggio romantico;
- stampare una citazione o un messaggio romantico;
- gestire una lista di oggetti che potrebbe corrispondere ad una lista della spesa;
- consultare e stampare la stessa.

Scendendo in dettaglio, i comandi sono implementati analizzando l’inizio del messaggio inviato, ricercandovi le seguenti parole:

- **stampa** invia alla stampante il resto del testo del messaggio;
- **romantico** preleva una frase romantica da un database e la invia come risposta;
- **citazione** preleva una citazione da un database e la invia come risposta;
- **stampa romantico** preleva una frase romantica da un database e la stampa;
- **stampa citazione** preleva una citazione da un database e la stampa;
- **aggiungi** aggiunge un oggetto alla lista della spesa;
- **rimuovi** rimuove un oggetto dalla lista della spesa;
- **mostra lista** invia come risposta il contenuto della lista della spesa;
- **stampa lista** stampa la lista della spesa;
- **svuota lista** svuota la lista della spesa;
- **help** mostra l’elenco dei comandi disponibili (questa lista).

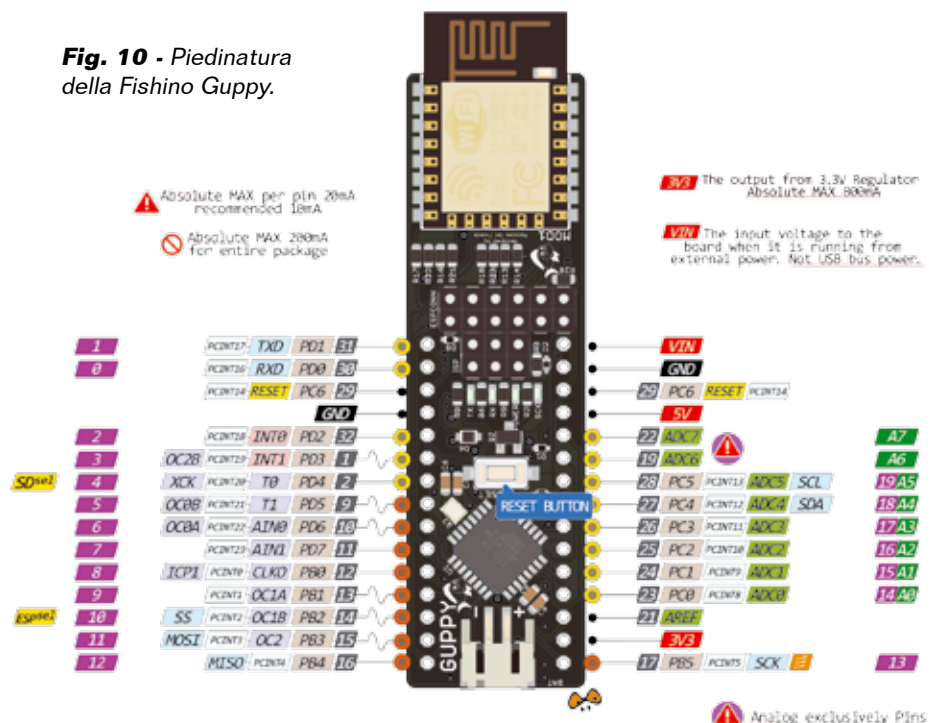
Ad esempio, inviando il messaggio ‘**stampa Ci vediamo stasera?**’ la frase ‘**Ci vediamo stasera?**’ verrà stampata immediatamente e, nello stesso tempo, riceveremo su Telegram un messaggio di conferma.

Vediamo quindi le parti salienti dell’applicazione. Innanzitutto, abbiamo bisogno di una funzione che gestisca l’evento proveniente da Telegram (quella che nei due piccoli esempi di prima si limitava a mostrare il messaggio sulla seriale e/o reinviarlo al mittente come conferma). In questo caso le cose si complicano leggermente, dovendo analizzare il testo del messaggio per estrarne i comandi. Abbiamo poi bisogno della **lista di comandi**, memorizzata in qualche modo “comodo” e di una serie di funzioni che gestiscano i comandi stessi.

Il modo più semplice per implementare la cosa è quello di una “linked list”, ovvero una **lista di elementi collegati** alla quale è possibile aggiungere o togliere (quest’ultima funzione qui non è utilizzata) degli elementi. Gli elementi stessi, poi, sono costituiti dal **testo del comando** abbinato alla **funzione che lo gestisce**.

Iniziamo con questi ultimi, dichiarati come nel **Listato 4**. La **typedef** iniziale serve a fornire una **sintassi abbreviata** per gestire le funzioni che eseguiranno i nostri comandi. In poche parole, queste funzioni hanno come parametri l’**id**, **nome** e **cognome** del mittente ed il **corpo del messaggio** senza il nome del comando; ritornano un valore booleano in base all’esito del comando stesso (valore qui non usato, ma potrebbe servire per

Fig. 10 - Piedinatura della Fishino Guppy.



Listato 4

```
typedef bool (*CommandHandler)(uint32_t, const char *, const char *, const char *);
struct CommandElement: public ListElement<CommandElement>
{
    const FLASH_HELPER *_name;
    CommandHandler handler;
    CommandElement(const FLASH_HELPER *_name, CommandHandler _handler):
    name(_name), handler(_handler) {}
};
```

Listato 5

```
commandList.add(new CommandElement(F("aggiungi")      , Cmd_AggiungiLista));
commandList.add(new CommandElement(F("rimuovi")        , Cmd_RimuoviLista));
commandList.add(new CommandElement(F("mostra lista")   , Cmd_MostraLista));
commandList.add(new CommandElement(F("stampa lista")   , Cmd_StampaLista));
commandList.add(new CommandElement(F("svuota lista")   , Cmd_SvuotaLista));
commandList.add(new CommandElement(F("stampa romantico"), Cmd_StampaRomantico));
commandList.add(new CommandElement(F("romantico")     , Cmd_Romantico));
commandList.add(new CommandElement(F("stampa citazione"), Cmd_StampaCitazione));
commandList.add(new CommandElement(F("citazione")    , Cmd_Citazione));
commandList.add(new CommandElement(F("stampa")        , Cmd_Stampa));
commandList.add(new CommandElement(F("help")         , Cmd_Help));
```

comandi differenti). La **struct CommandElement** rappresenta invece la descrizione del comando vero e proprio, ovvero il testo (**name**) ed il puntatore alla funzione che lo gestisce (**handler**). Nella struct (che è praticamente la stessa cosa di una class, salvo qualche lieve differenza) abbiamo anche inserito un **costruttore** che permette di inizializzarla. La **FLASH_HELPER** è una macro (#define) che dipende dal controller usato; per gli 8 bit viene tradotta in **__FlashStringHelper**, per poter utilizzare le stringhe nella memoria flash (e quindi risparmiare preziosa RAM), mentre nei 32 bit viene tradotta in una semplice **char**, visto che per queste tipologie di controller non c'è differenza tra stringhe in RAM o nella Flash. La lista dei comandi viene quindi rappresentata in questo modo:

```
List<CommandElement> commandList;
```

Qui utilizziamo una **template** scritta da noi per gestire una **lista di oggetti generici**. Perché una **template** (e soprattutto... cos'è una **template**???) , e perché non

utilizzarne una già fatta da altri? Alla prima domanda e mezza si può rispondere che una **template** è quello che si chiama una **classe generica**, ovvero può essere utilizzata per gestire differenti tipi di oggetti. Quindi, se volessimo realizzare una lista di patate, basterebbe scrivere:

```
List<Patate> listaPatate;
```

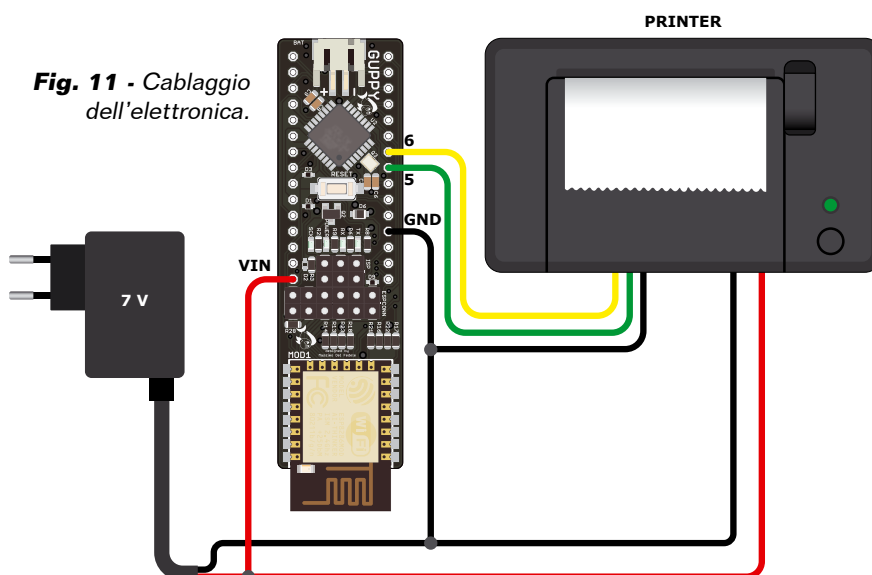
ed otterremmo quanto desidera-

to, senza dover scrivere una sola riga di codice aggiuntivo. Allo stesso modo, se desiderassimo una lista di Rose, potremmo scrivere:

```
List<Rose> listaRose;
```

Una bella differenza dal dover riscrivere ogni volta tutto il codice per gestire tipi di dati differenti! La **template List** si trova nel file **List.h** dentro alla cartella del progetto; è probabile che in una futura versione delle librerie raccoglieremo un po' di codice riutilizzabile come questo in una libreria apposita. Analizzando il codice corrispondente, si nota che il template fornisce funzioni per aggiungere, eliminare e percorrere tutti gli elementi della lista; non si tratta di un codice particolarmente complesso né performante, ma svolge egregiamente il suo compito.

Alla seconda domanda... non c'è una risposta: è vero che esistono in rete decine di implementazioni di Liste, Array, eccetera; semplicemente abbiamo preferito scriverne una limitata (e facilmente comprensibile) adatta al nostro



scopo, anche per motivi didattici. Una volta dichiarata la lista di comandi occorre riempirla, e questo avviene nelle linee del **Listato 5**. La funzione **add** aggiunge un nuovo (**new**) elemento (**CommandElement**) inizializzandolo con il **nome** del comando e la funzione di gestione (**handler**) corrispondente. Ad esempio, la prima riga implementa il comando **"aggiungi"** gestito dalla funzione **Cmd_AggiungiLista()**. Torniamo ora al gestore degli eventi di FishGram, che utilizza il codice di **Listato 6**.

Questo non fa altro che percorrere tutta la lista di comandi, confrontandoli con l'inizio del messaggio ricevuto; quando trova una corrispondenza termina la ricerca ed esegue il comando corrispondente.

Se non trova una corrispondenza invia al mittente un messaggio di errore (la penultima riga).

La funzione **starts()** richiamata dall'**handler** è un piccolo **helper** (funzione di utilità) che controlla se un testo inizia con una stringa prefissata, ed è definita poche righe sopra (**Listato 7**).

Non ci resta che vedere una delle funzioni di gestione dei comandi; per semplicità vedremo quella che implementa il comando 'stampa' (**Listato 8**).

Questa funzione non fa altro che stampare, come richiesto, il messaggio (**sendToPrint(str)**) e risponderci con un messaggio di conferma, costruito per risparmiare ripetizioni con la **helloMsg** (che crea una stringa costituita da "Ciao <nome>, ", aggiungendovi il testo "ho stampato il tuo messaggio '", il testo del messaggio e la virgoletta di chiusura finale, reinviandolo poi al mittente tramite la **FishGram.sendMessage()**.

CITAZIONI E LISTA DELLA SPESA
Per queste due funzionalità da-

remo solo una breve descrizione; non si tratta di codice particolarmente complesso ma appesantirebbe comunque troppo l'articolo. Il codice è comunque ben commentato e scaricabile dal nostro sito www.elettronica.in e, in seguito, verrà incluso tra gli esempi nelle librerie di Fishino. Le citazioni (e le frasi romanti-

che) vengono implementate per forza di cose appoggiandoci ad un server esterno, tramite un programmino in php contenente una serie di frasi preimpostate ed un algoritmo per poterne generare una casuale ad ogni accesso, a meno di non richiederne una specifica. Ma perché un php esterno? Non si potevano implementare

Listato 6

```
// fishgram event handler
bool FishGramHandler(uint32_t id, const char *firstName, const char *lastName, const char *message)
{
    CommandElement *elem = commandList.head();
    while(elem)
    {
        if(starts(message, elem->name))
        {
            message += strlen_P((const char *)elem->name);
            while(*message && isspace(*message))
                message++;
            return elem->handler(id, firstName, lastName, message);
        }
        elem = elem->next();
    }

    // command not found
    FishGram.sendMessage(id, F("Comando sconosciuto - inviare 'help' per lista comandi"));
    return false;
}
```

Listato 7

```
bool starts(const char *msg, const FLASH_HELPER *cmd)
{
    char c;
    uint16_t i = 0;
    while((c = charAt(cmd, i++)) != 0)
        if(toupper(*msg++) != toupper(c))
            return false;
    return true;
}
```

Listato 8

```
bool Cmd_Stampa(uint32_t id, const char *firstName, const char *lastName, const char *str)
{
    // print the message
    sendToPrint(str);

    // send a confirmation back to bot
    String ans = helloMsg(firstName, lastName);
    ans += F("ho stampato il tuo messaggio ");
    ans += str;
    ans += "'";
    FishGram.sendMessage(id, ans.c_str());
    return false;
}
```

Listato 9

```
// la lista della spesa
struct ShoppingListElement: public ListElement<ShoppingListElement>
{
    char *item;
    ShoppingListElement(const char *s) { item = strdup(s); }
    virtual ~ShoppingListElement() { if(item) free(item); }
};
List<ShoppingListElement> shoppingList;
```

direttamente sul Fishino, magari con l'utilizzo di una scheda SD? Sì, ma non con le versioni **UNO** e **GUPPY** che abbiamo scelto per l'applicazione, soprattutto per motivi di compattezza. Queste infatti non hanno lo spazio di memoria sufficiente per gestire sia **FishGram** che la **scheda SD**. Utilizzando lo (scarso) spazio lasciato libero in Flash dall'applicazione avremmo potuto mettere insieme poche frasi, cosa che avrebbe portato a ripetizioni nel breve termine. Utilizzando una Fishino Mega o, ancora meglio, la nuova Fishino32 questi problemi spariscono completamente ed è possibile implementare in locale anche un grosso database di citazioni su scheda SD o anche direttamente nella memoria Flash del controller.

Tornando alle citazioni, la richiesta avviene, tramite protocollo HTTP (per i limiti di Fishino di non poter aprire più di una connessione HTTPS alla volta) tramite una semplice richiesta **GET** ad una path situata sul sito *www.fishino.it*. Il modulo PHP risponde con una stringa di testo semplice con questo formato:

ID, LEN, citazione

dove **ID** è un numero identificativo della citazione, per poterla "ripescare", ad esempio, per mandare al mittente la conferma di quanto si è stampato; **LEN** è la lunghezza del testo della citazione stessa, necessaria per poter utilizzare le funzioni **startMessage()** e annesse della libreria **FishGram** (ricordate? Permettono

di inviare un messaggio anche carattere per carattere, senza quindi la necessità di doverlo scaricare per intero sul Fishino). Il codice di richiesta della citazione al server è contenuto nei moduli **Cit.h** e **Cit.cpp**, anch'essi reperibili nella cartella dell'applicazione, ed è di facile comprensione e ben commentato. Per quanto riguarda la lista della spesa, invece... abbiamo sfruttato ancora una volta la nostra **template List**, essendo questa perfetta per gestire questo tipo di dati; la dichiarazione è riportata sul **Listato 9**.

Come si può vedere, abbiamo prima definito una nuova struct contenente il dato che ci interessa (un semplice puntatore a carattere, che conterrà una stringa di testo allocata dinamicamente tramite **strdup()**); la novità qui è il distruttore (**~ShoppingListElement()**) che si occupa di liberare la memoria dinamica quando eliminiamo l'elemento.

La dichiarazione della lista è quindi immediata, come si vede dalla linea successiva alla struct, ed il suo utilizzo identico a quello relativo alla lista di comandi. In questo caso utilizziamo anche la funzione **remove()** per eliminare, a richiesta, elementi dalla lista. Per esempio, se vogliamo aggiungere "Pasta" alla lista tramite codice, possiamo scrivere:

```
shoppingList.add(new
ShoppingListElement("Pasta"));
```

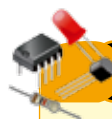
Come accennato, la nostra implementazione di **List** è ben lungi dall'essere completa; manca, per esempio, una funzione di ricerca,

che sarebbe utile ma che avrebbe complicato il codice oppure limitato la generalità dello stesso. Per cercare un oggetto nella lista dovremo quindi eseguirne una scansione completa "a mano", come fa il gestore di eventi di **FishGram** visto sopra.

L'ELETTRONICA

Il collegamento della stampante è davvero semplicissimo (**Fig. 11**); bastano tre cavetti: uno da collegare alla massa del Fishino, uno alla linea **D6** (il **TX**, ovvero l'**RX** della stampante) ed uno alla linea **D5** (l'**RX**, ovvero il **TX** della stampante). La stampantina comunica tramite un semplice protocollo seriale, per il quale sfruttiamo una **SoftwareSerial** tramite l'apposita libreria, ed è gestita dalla libreria **Adafruit_Thermal**.

Bene, si conclude qui la descrizione della nostra Note Machine, utilizzabile anche come spunto per controlli molto diversi, quali per esempio la gestione di luci, allarmi ed altro. Prossimamente presenteremo altre interessanti applicazioni di Telegram. ■



per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono disponibili presso Futura Elettronica. La mini stampante termica (cod. COM10438) è in vendita a Euro 79,00, la board Fishino GUPPY (cod. GUPPY) è disponibile a Euro 33,90 e l'alimentatore da rete switching multitensione (cod. PSSMV1) costa Euro 15,50. I prezzi si intendono IVA compresa.

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

OCTOPUS: LA PIOVRA CHE SUONA IL PIANO

..... di GABRIELE DAGHETTA

Con Fishino e il driver Octopus facciamo suonare una pianola giocattolo che riprodurrà motivi musicali nati con le suonerie Nokia.

Da parecchi anni le tastiere elettroniche, in special modo i sintetizzatori musicali, hanno la possibilità di suonare "da soli" brani, opportunamente pilotati tramite l'interfaccia MIDI o riproducendo file in formato standard

caricati da supporti esterni di memorizzazione di massa. Ma quello che ci siamo proposti di fare è far suonare motivi a nostra scelta a strumenti musicali più abbordabili, come piccole tastiere musicali e pianole giocattolo

acquistabili nei centri commerciali per poche decine di euro, che per evidenti ragioni non dispongono di interfacce dati o slot per supporti di memorizzazione di massa e che quindi non possono essere pilotate elettronicamente. In questi casi l'unica soluzione è farle suonare in maniera tradizionale, agendo sui tasti con qualcosa che trasformi le note dei brani musicali in comandi meccanici. Insomma, un'interfaccia da elettronica a meccanica. Questo lo abbiamo fatto e ha





Fig. 1
Il sistema
carrozzato con
le tre piovre e
applicato alla
pianola.

dato origine al progetto descritto in queste pagine, ossia un gadget che abbiamo sviluppato affinché possa essere utilizzato in varie occasioni, come ad esempio da demo in una vetrina o nel reparto giocattoli di un negozio, come pure da intrattenimento per grandi e piccoli in eventi e manifestazioni, per esempio, di robotica applicata.

Il progetto è nato dal desiderio di poter riprodurre con una tastiera musicale low cost (una tastiera giocattolo) delle musiche standard, però suonandole premendo effettivamente i tasti, pur senza metterci un dito. Abbiamo quindi pensato a una soluzione automatizzata che consiste in un piccolo robot che suona per noi i tasti della pianola. Le musiche che abbiamo scelto non sono file audio qualsiasi ma, per praticità, abbiamo pensato ai ring tone

(vale a dire le suonerie) sviluppati dalla Nokia e definiti RTTTL (acronimo di Ring Tone Text Trasfer Language), “suonati” dai cellulari del costruttore.

La nostra piattaforma robotica, che va applicata alla pianola adattandosi a forma e distanza dei tasti di quest’ultima, è composta da un “cervello” basato sulla ormai popolare scheda Fishino Uno e da un sistema di attuazione realizzato col driver Octopus, già presentato nel fascicolo 203 di Elettronica In. Il tutto è stato “carrozzato” in stile fondale marino, con il risultato visibile nella Fig. 1.

La gestione dei toni è affidata ad uno sketch appositamente creato per la nostra board Fishino Uno, mentre il controllo fisico dei tasti è ottenuto mediante un sistema elettromeccanico composto da 24 servomotori (cod. SERVO206 del catalogo Futura Elettronica, reperibili su www.futurashop.it) gestiti da due shield Octopus (codice OCTOPUS Futura Elettronica) a cui sono collegati gli otto tentacoli di tre amene piovre color rosso stampate in PLA con la stampante 3D, azionati dai servo mediante un semplice ma preciso sistema di leveraggio (Fig. 2). Tramite i 24 servo è possibile gestire due ottave (sulla pianola da

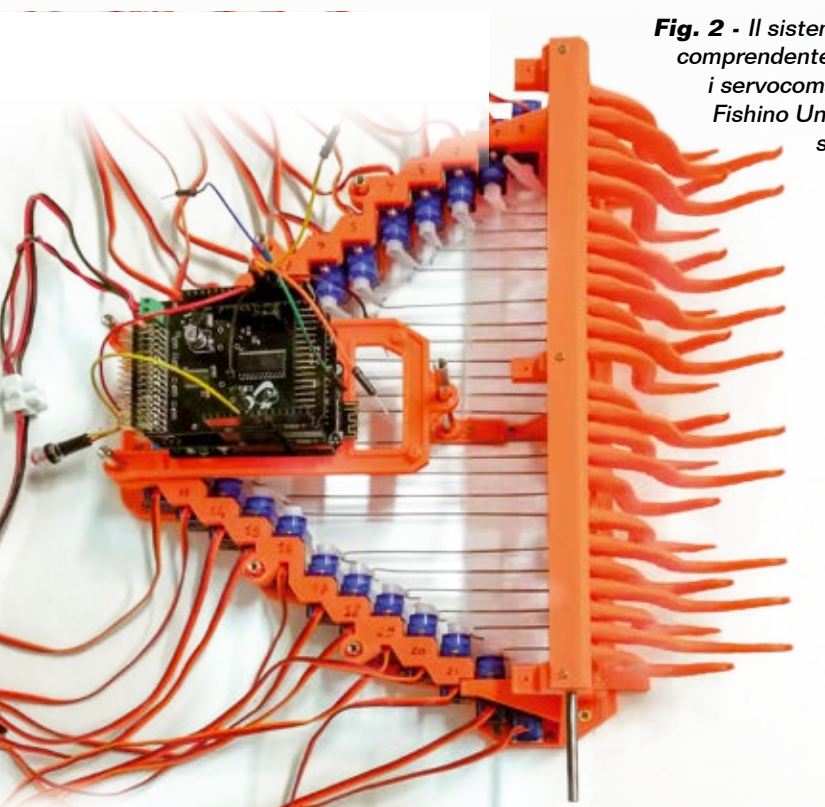
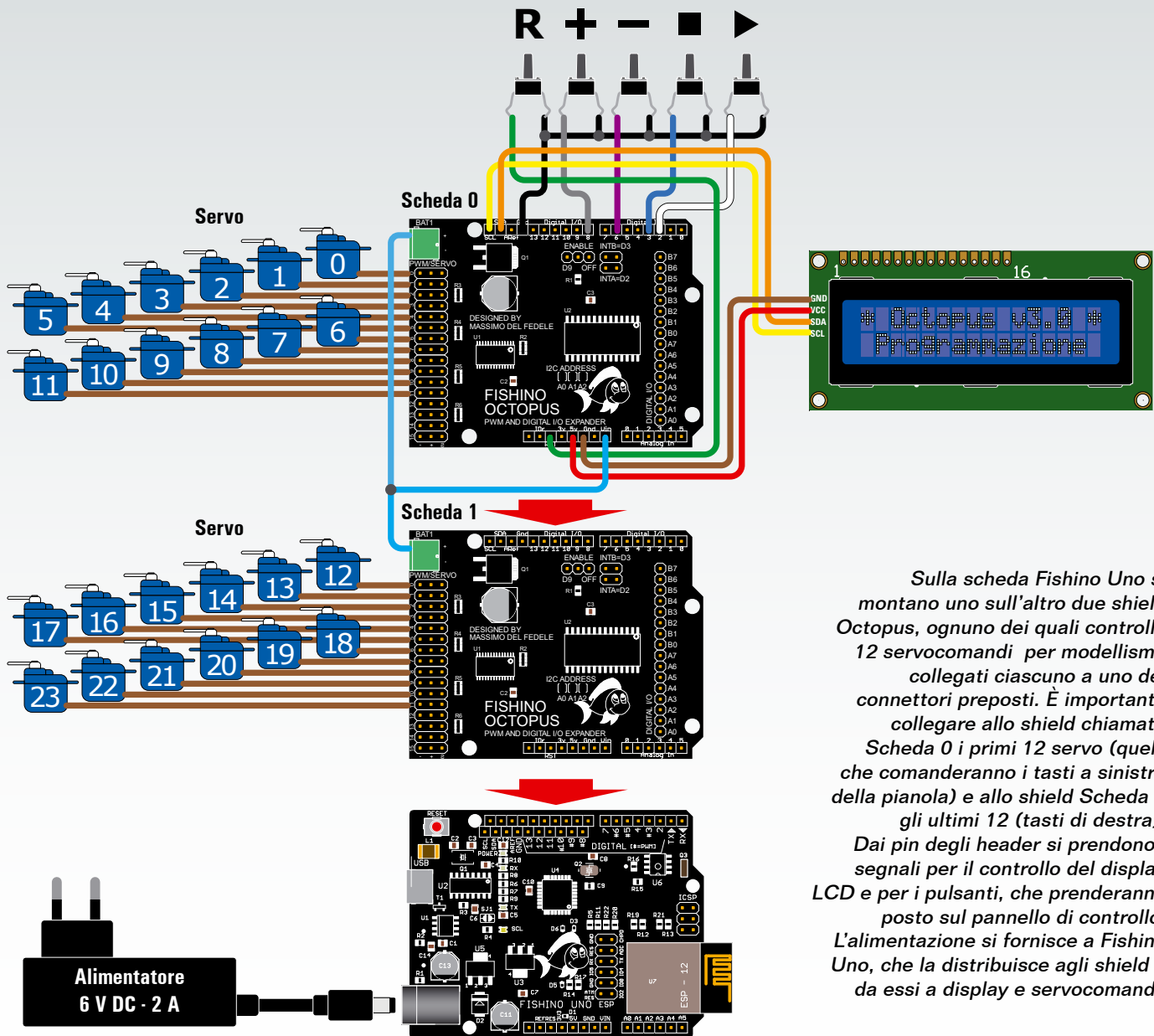


Fig. 2 - Il sistema di attuatori, comprendente la meccanica, i servocomandi e la board Fishino Uno con sopra gli shield Octopus.



Fig. 3 - Il pannello di controllo contenuto in un contenitore stampato in 3D con del PLA.



Sulla scheda Fishino Uno si montano uno sull'altro due shield Octopus, ognuno dei quali controlla 12 servocomandi per modellismo collegati ciascuno a uno dei connettori preposti. È importante collegare allo shield chiamato Scheda 0 i primi 12 servo (quelli che comanderanno i tasti a sinistra della pianola) e allo shield Scheda 1 gli ultimi 12 (tasti di destra). Dai pin degli header si prendono i segnali per il controllo del display LCD e per i pulsanti, che prenderanno posto sul pannello di controllo. L'alimentazione si fornisce a Fishino Uno, che la distribuisce agli shield e da essi a display e servocomandi.

noi utilizzata corrispondono alla 4^a e alla 5^a del pianoforte), quindi 12 tasti per ottava corrispondenti alle 7 note intere (Do, Re ecc..) più i semitoni (ad esempio Do Diesis o RE Bemolle). Per chi volesse approfondire l'argomento, i tasti di ciascuna ottava sono convenzionalmente i seguenti:

C = nota DO
C# = DO Diesis o RE Bemolle
D = nota RE

D# = RE Diesis o Mi Bemolle
E = nota MI
F = nota FA
F# = FA Diesis o Sol Bemolle
G = nota SOL
G# = SOL Diesis o LA Bemolle
A = nota LA
A# = LA Diesis o SI Bemolle
B = nota SI

Le note intere sono i tasti bianchi e quelli neri rappresentano i semitoni standard della tastiera del pianoforte.

IL SISTEMA

Bene, detto ciò possiamo entrare nel vivo del progetto, che consta di un'elettronica (formata da Fishino Uno e due shield Octopus) e una meccanica (l'insieme è visibile, smontato, nella Fig. 2); la prima è stata già meglio descritta nell'articolo che le abbiamo dedicato nel fascicolo di marzo 2016 (dove trovate l'hardware e il firmware necessari a gestirlo) quindi ci limitiamo a dire che gli shield sono due e vengono impi-



Fig. 4 - Tentacolo e relativa asta di rinvio.



Fig. 5
Dettaglio dell'azionamento dei tentacoli da parte dei servocomandi, tramite la relativa asta di rinvio piegata per entrare nel braccetto dei servo.

Per il controllo dei servo ci siamo avvalsi di uno shield, compatibile con le schede Arduino e con la nostra Fishino Uno, che, senza praticamente impegnare risorse hardware, permette di avere a disposizione ben 16 uscite in PWM e 16 ingressi/uscite digitali aggiuntivi. Non solo, le schede sono impilabili fino ad un massimo di 8, consentendo di gestire con Arduino e Fishino fino a 128 I/O digitali e 128 uscite PWM aggiuntive; il tutto reso completamente trasparente all'utilizzatore tramite una libreria realizzata ad-hoc. La libreria software, denominata Octopus, è dotata di alcune particolarità che ne rendono semplicissimo l'utilizzo. Lo shield nasce perché le schede

lati uno sull'altro su Fishino Uno. Gli shield sono gestiti attraverso l'I²C-Bus di quest'ultimo. Sebbene gli shield Octopus dispongano di 16 uscite per controllo di servo, nella nostra applicazione ognuna ne pilota 12; le schede prelevano l'alimentazione per i servocomandi dal pin Vin di Fishino (qui arrivano circa 5,25V) e quella per il proprio funzionamento dal pin 5V. L'intero sistema viene alimentato da un alimentatore switching da 6V -

2A che fornisce tutta la corrente necessaria alla Fishino Uno. L'interfaccia utente è rappresentata da un display LCD 2 righe e 16 caratteri alfanumerici (cod. LCD1602ABLU Futura Elettronica) a cui è stata applicata un'interfaccia I²C (cod. LCDI2CINTERFACE Futura Elettronica) e da un tastierino a 5 pulsanti che permettono di selezionare il tono che si desidera riprodurre ed anche di effettuare la taratura dei servo ed eseguire un test di verifica della pressione dei tasti della pianola. Agli header sono collegati i cinque pulsanti, come segue:

Mem/PLAY -> pin 2
Più -> pin 8
Meno -> pin 6
Esci/STOP -> pin 3

R -> pin di reset/riavvio di Fishino

Invece il display è collegato così:
SDA display -> SDA
SCL display -> SCL
Vcc display -> 5V
GND display -> GND.

I pin sono, anche in questo caso, quelli degli header. L'utilizzo di un display con interfaccia I²C permette di limitare il numero minimo di pin della board Fishino Uno utilizzati.

Nella Fig. 3 potete vedere come abbiamo realizzato il pannello di controllo del nostro sistema, che poi è stato collocato nella parte posteriore dell'insieme, appena sotto all'elettronica di controllo formata da Fishino Uno e dagli shield Octopus.

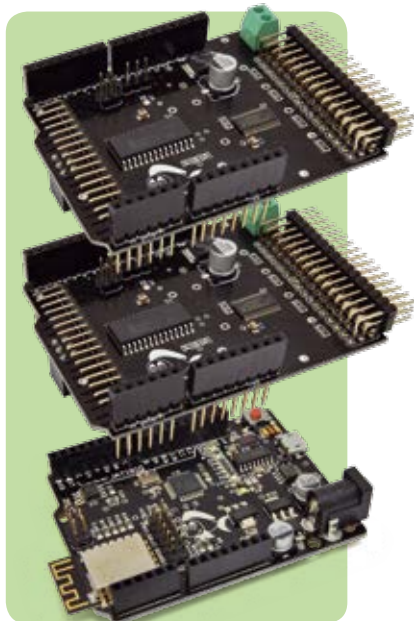


Fig. 6 - Sistema di attuazione dei tentacoli.



Fig. 7 - Tentacoli e aste di rinvio visti dal retro.

Octopus: lo shield tentacolare

Arduino e compatibili hanno una ridotta quantità di uscite disponibili, specie di I/O cui si può assegnare il PWM. Per esempio, una Arduino/Fishino UNO dispone di sole sei uscite PWM e, a meno di non generare i relativi segnali via software (con notevole impegno del processore), permette il pilotaggio di soli sei servocomandi. Anche gli I/O digitali sono limitati: per Arduino abbiamo un totale di 13 I/O digitali e 6 ingressi analogici, utilizzabili anch'essi in digitale. Tale dotazione sembrerebbe abbondante, se non fosse che molti I/O vengono utilizzati per le periferiche a bordo o dagli shield di espansione. Octopus estende gli I/O sfruttando la sola connessione I²C-Bus; come tutte le

periferiche del genere va indirizzata per assicurare l'univocità dei comandi diretti a ciascuno shield ed allo scopo prevede

tre jumper da impostare diversamente da uno shield all'altro. I jumper consentono 8 combinazioni logiche.



PARTE MECCANICA

Dopo aver analizzato l'elettronica del progetto, occupiamoci della meccanica, che è ugualmente importante perché è la sezione attuatrice che interviene fisicamente sui tasti della pianola. Tutta la struttura necessaria a sostenere i 24 tentacoli e a ospitare gli altrettanti servomotori (provvisi ciascuno di un braccetto avvitato all'alberino con le viti in dotazione), nonché le schede elettroniche (il telaio, quindi...) è stata progettata con Google SketchUp e realizzata mediante stampa 3D con del PLA; il telaio vero e proprio si stampa utilizzando il file STL da noi fornito insieme agli altri file del progetto (sketch ecc.) nel nostro sito Internet www.elettronica.in.it. Al telaio devono poi essere applicati i particolari metallici richiesti per l'assemblaggio delle parti, come il fulcro dei tentacoli; più esattamente, il telaio consta di più parti (lati, elemento di congiunzione posteriore, traversa anteriore...) da unire mediante viti. Ciascun servocomando aziona un tentacolo (anche questo realizzato per stampa 3D il cui file STL è reperibile sul nostro sito insieme agli altri file del progetto)

to) mediante un'astina di rinvio (opportunamente sagomata) ottenuta stirando un filo in acciaio inox da 0,9 mm di diametro (Fig. 4). Il filo va poi ripiegato per vincolarlo sia dalla parte dell'albero del servocomando, sia dal lato del foro previsto nel tentacolo corrispondente (Fig. 5). L'insieme dei tentacoli è libero di oscillare per assecondare il movimento imposto dai servocomandi; allo scopo ogni tentacolo, disposto affiancato agli altri, è

imperniato su un'asta in metallo da 4 mm di diametro. La lunghezza delle aste di rinvio non è uguale ma va scelta di volta in volta sulla base della distanza tra il braccetto del servocomando e la base della leva di attuazione del tentacolo corrispondente, come si vede nella Fig. 6, che mostra l'insieme della meccanica di un lato (manca la parte destra dei servocomandi e il relativo sostegno. La Fig. 7 dettaglia, invece, l'aggancio delle



Fig. 8 - La meccanica a ridosso della tastiera della pianola.



Fig. 9 - Entrata in programmazione.



Fig. 10 - Programmazione dei servo.



Fig. 11 - Fine programmazione.



Fig. 12 - Esecuzione del test iniziale.



Fig. 13 - Schermata di scelta traccia.

aste alla base dei tentacoli, Le fotografie fatte al prototipo chiariscono meglio d'ogni parola come si assembla la struttura meccanica, la quale, una volta terminata, va montata a ridosso della pianola, frontalmente, in modo che ogni tentacolo prema uno dei tasti.

Siccome è probabile che la pianola si sposti durante il funzionamento a causa della pressione esercitata dai tentacoli, è opportuno vincolare il telaio alla stessa: ad esempio con delle strisce metalliche ripiegate a squadra all'altezza del fondo dello strumento musicale, oppure appoggiando e avvitando la meccanica a una piastra di base in legno o plastica (forex, per esempio...) e fissando alla stessa, una volta determinata la posizione ottimale, la pianola mediante squadrette in plastica opportunamente stampate in 3D e avvitate alla piastra di base). La Fig. 8 propone un possibile fissaggio dell'insieme.

FUNZIONI DEI TASTI

Bene, descritta anche la parte meccanica possiamo vedere come si utilizza e quindi in che modo possiamo far suonare il nostro speciale strumento. Descriviamo di seguito il funzionamento dei pulsanti.

- **Mem/PLAY:** permette di avviare l'esecuzione del tono occupante la posizione visualizzata sul display. Se tenuto premuto all'accensione permette di entrare in "Modalità impostazione posizione servo".
- **Più:** permette di passare al tono successivo. In "Modalità impostazione posizione servo" permette di modificare la posizione del tentacolo al fine di premere + o - il tasto della pianola. Se viene tenuto premuto all'accensione permette di entrare in modalità TEST.

**Guarda Octopus
in azione su YouTube!**



Puoi vedere Octopus, realizzato con la stampante 3Drag, mentre suona il piano in un video demo



che abbiamo pubblicato alla pagina <https://youtu.be/ayyAver7gMg>. Dal nostro canale YouTube www.youtube.com/user/ElettronicaIN potrai anche vedere i video di moltissimi altri nostri progetti proposti in passato.



- **Meno:** permette di passare al tono precedente. In "Modalità impostazione posizione servo" permette di modificare la posizione del tentacolo al fine di premere + o - il tasto della pianola. Se tenuto premuto all'accensione permette di entrare in modalità TEST.
- **Esci/STOP:** permette di interrompere l'esecuzione di un tono. Se viene premuto quando ci si trova in "Modalità impostazione posizione servo" permette di uscire da tale modalità e di tornare alla schermata principale.
- **R:** permette di resettare e/o riavviare il sistema.

Reset della posizione di tutti i servo

Prima di applicare la leva di

comando, facente capo al tentacolo, sul perno di ciascun servo, occorre posizionare tutti i servo a metà corsa (posizione centrale, -90°); tale operazione è necessaria per essere certi che una volta portati a zero i servocomandi, i braccetti siano tutti nella posizione che lascia a riposo i tentacoli. Per effettuare il reset della posizione di tutti i servocomandi bisogna premere e tenere premuto il tasto **Esci/STOP** quindi fornire alimentazione; il display indicherà la scritta "Reset

in corso" dopo-diché tutti i perni dei servo si collocano in posizione centrale e sul display apparirà la scritta "Reset eseguito". Ora rilasciate il tasto. Fatto ciò applicate su ciascun perno la propria levetta in modo che il relativo tentacolo sfiori (senza premere) il corrispondente tasto della pianola.

MODALITÀ PROGRAMMAZIONE

Vediamo adesso come si programma la posizione di pressione del tasto corrispondente al singolo tentacolo. Ad ogni servo deve essere assegnata una posizione che permette la pressione del tasto assegnato. Per effettuare l'impostazione del caso bisogna seguire la procedura seguente.

- 1) Accendere la pianola, premere e tenere premuto il tasto **Mem/PLAY** quindi fornire alimentazione; il display visualizza la scritta "Programmazione" (Fig. 9).
- 2) Rilasciare il tasto. Il servo gestito sarà il primo a sinistra (la prima nota della prima ottava); il display visualizzerà la scritta "Programmazione Servo no.1" (Fig. 10).
- 3) Premere il tasto + o - in modo che il relativo tentacolo prema sul corrispondente tasto quanto basta per emettere la



nota. Raggiunto l'obiettivo, premere nuovamente il tasto **Mem/PLAY** per passare alla programmazione del servo successivo.

- 4) Giunti all'ultimo servo premere il tasto **Mem/PLAY** per concludere la programmazione (sul display apparirà la scritta "Programmazione Completata" (Fig. 11).

TEST DEL SISTEMA

Ora verificate se tutti i tentacoli premono correttamente il tasto assegnato: per fare ciò bisogna:

- accendere la pianola, premere e tenere premuto il tasto Più (o Meno) quindi fornire alimentazione; il display visualizza la scritta "Test" (Fig. 12).
- Rilasciare il tasto.

Ora i tentacoli verranno azionati in successione, da sinistra verso destra, così da verificare la corretta pressione di tutti i tasti ad essi assegnati. Al termine del test il sistema tornerà alla schermata principale "Scelta traccia".

SCELTA TRACCIA / PLAY / STOP

Quando il sistema è in standby, sul display appare la scritta "Scelta traccia" seguita dal numero della traccia (della suoneria) attuale (Fig. 13). Per avviare l'esecuzione della traccia indicata premete il tasto PLAY (il display mostra ad esempio "Play Traccia no.1"); per arrestare l'esecuzione premete STOP.

Per scegliere un'altra traccia premete il tasto Più o il Meno (il comando non ha effetto se il pulsante viene premuto durante l'esecuzione della traccia); prima ricordate di premere STOP.

IL FIRMWARE

Concludiamo questo articolo con un accenno al firmware, nel quale sono contenuti sia i Ringtone da

riprodurre, sia le istruzioni per suonarli. Partiamo proprio dagli RTTTL, dicendo che nel sistema è previsto ne vengano suonati 30: ciò viene definito nell'istruzione `int Tracce_Max = 29;` nell'inizializzazione dello sketch. Notate che 30 corrisponde al valore 29 perché si considera di partire da 0. Modificando tale valore si può modificare il numero massimo di suonerie richiamabile.

Ciascun motivo viene costruito secondo lo standard RTTTL, che prevede la definizione di ogni nota con la relativa durata, separata da una virgola dalla seguente e dalla precedente. Il formato tipico di una stringa è:

`const char PROGMEM Music-SongPlay1[] = "wewish:d=4,o=5,b=140:d,g,8g,8a,8g,8f#,e,c,e,a,8a,8b,8a,8g,f#,d,f#,b,8b,8c6,8b,8a,g,e,8d,8d,e,a,f#,2g.";` che riguarda l'esecuzione del Ringtone *wewish*, composto dalle note definite dalla lettera con, seguita da uguale, la durata, quindi `d=4`, `o=5`, `b=140` ecc. (`d` è la durata, o l'ottava e `b` le battute). Notate che tali istruzioni possono essere composte scrivendole nell'IDE Arduino, ovvero copiate direttamente dai siti sul web che riportano i Ringtone, dato che li espongono con detta sintassi.

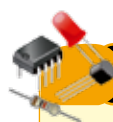
In genere tali siti forniscono dei file che si possono aprire con un editor di testo (sono dei `.txt`); per scrivere la musica vi basta copiare la stringa di testo interessata e incollarla nell'editore dell'IDE di Arduino.

Nello sketch, ciascuna nota viene definita dall'istruzione del tipo: `#define NOTE_C4 262`

che specifica la posizione del servo da far attivare per premere il tasto corrispondente sulla tastiera della pianola.

Quanto all'indirizzamento degli shield, la libreria *Octopus.h* è

preimpostata in modo che le uscite 0÷15 competano allo shield indirizzato come 0, le 16÷32 a quello indirizzato come 1, ecc. Quindi prima di applicare gli shield ad Arduino o Fishino dovete impostare i jumper dell'indirizzo del primo shield come 000 e quelli del secondo come 001, altrimenti non verrà rispettata la corrispondenza tra note e servo-comandi stabilita dal firmware. Lo sketch per Arduino/Fishino è piuttosto voluminoso e, per ragioni di spazio, non possiamo riportarlo in queste pagine; lo potete comunque trovare e scaricare nella sezione Download del nostro sito www.elettronica.in.it, insieme agli altri file del progetto. Bene, con questo diciamo che abbiamo concluso; lasciamo a voi e alla vostra fantasia eventuali sviluppi e personalizzazioni. ■



per il MATERIALE

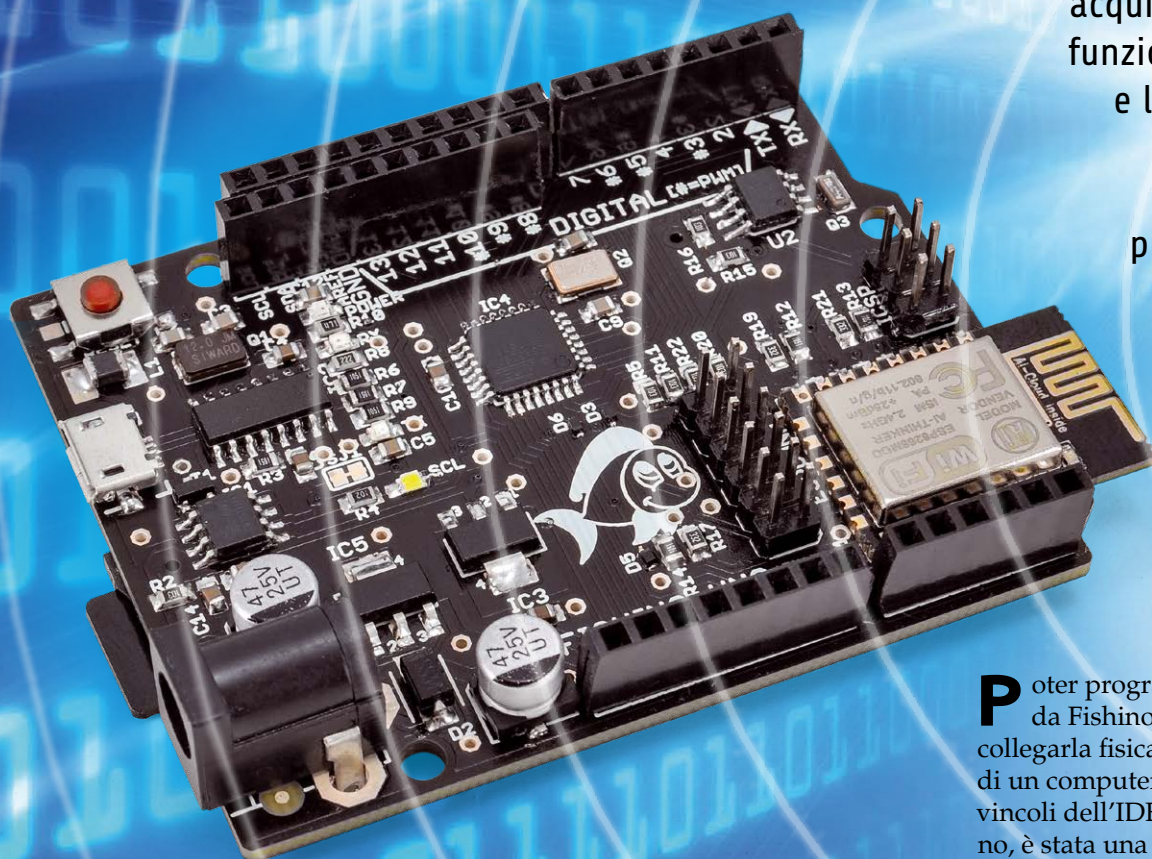
Tutti i componenti utilizzati in questo progetto sono di facile reperibilità, mentre il materiale elencato di seguito è disponibile presso Futura Elettronica. La board Fishino UNO (cod. FISHINOUNO) è in vendita a Euro 36,00, l'OCTOPUS shield 16 I/O per Fishino/Arduino (cod. OCTOPUS) costa Euro 24,00 (cadauno). Il SUB MICRO SERVO (cod. SERVO206) è disponibile a Euro 7,50 (cadauno), il display LCD 16x2 con retroilluminazione blu (cod. LCD1602ABLU) è in vendita a Euro 8,70 e l'interfaccia I²C per display LCD 16x2 (cod. LCDI2CINTERFACE) costa Euro 6,00. Tutti i prezzi si intendono IVA compresa.

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

WiFish, PROGRAMMIAMO FISHINO DAL WiFi

di MASSIMO DEL FEDELE

La nostra board Arduino-compatible acquisisce una nuova funzionalità: l'upload e l'aggiornamento degli sketch in wireless, senza più bisogno di un collegamento via cavo.



Poter programmare la scheda Fishino in campo, senza collegarla fisicamente all'USB di un computer né sottostare ai vincoli dell'IDE Arduino/Fishino, è stata una delle ambizioni "nascoste" del creatore della popolare board di prototipazione con WiFi ed RTC integrato, che avete visto nascere dalla Uno per poi trasformarsi nella Mega, nella Guppy e nella 32. Parliamo

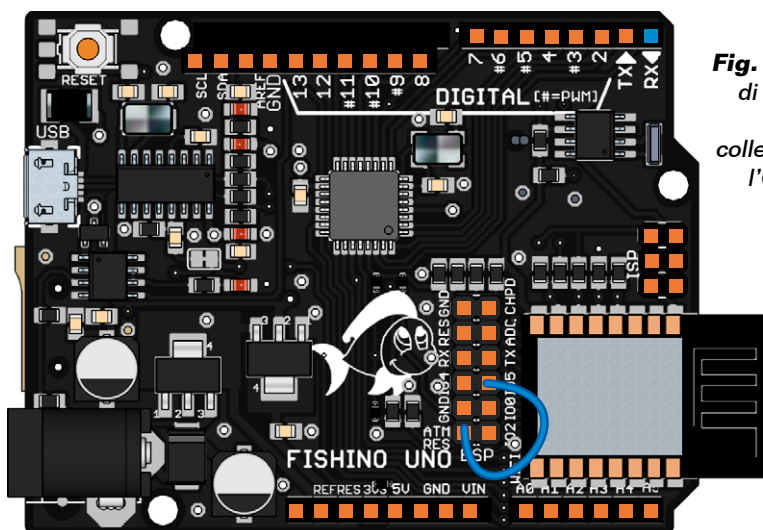


Fig. 1 - I contatti di Fishino UNO interessati al collegamento per l'OTA da WiFi.

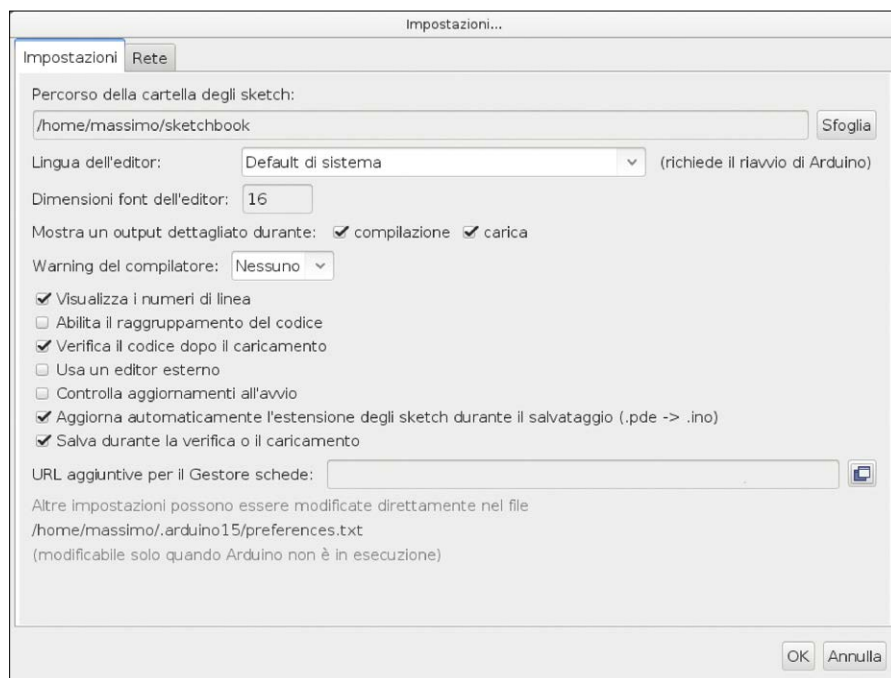


Fig. 2 - Finestra di dialogo Impostazioni dell'IDE Arduino.

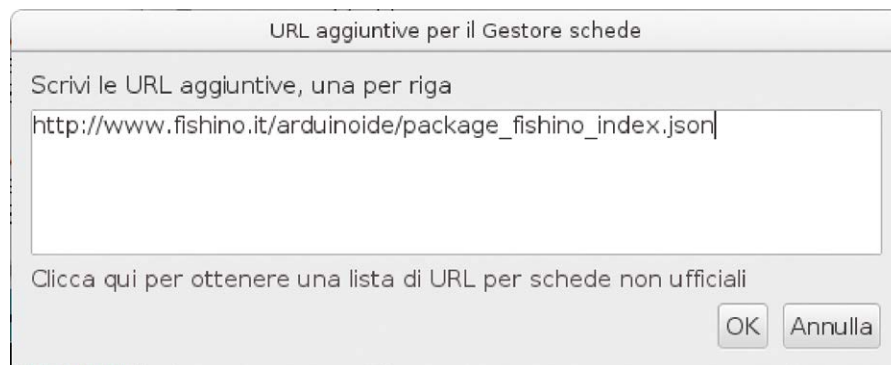


Fig. 3 - Finestra di dialogo dove inserire gli URL.

della possibilità di aggiornare e sostituire il firmware (ossia lo sketch) attraverso la connessione WiFi nativa della scheda. Questo consente enormi vantaggi sia per la semplicità d'uso, sia quando la board è posizionata in luoghi dove non è facile accedere, ma anche, e soprattutto, quando è necessario operare on-line da remoto per risparmiare l'intervento diretto di personale tecnico. Fino a poco tempo fa l'autore di questo articolo l'aveva solamente teorizzato, oggi è diventato realtà. Certi che la cosa possa interessare un vasto pubblico, vogliamo descriverla nei dettagli, partendo dalle origini, ossia dall'idea che vi sta alla base.

LA PROGRAMMAZIONE REMOTA DI FISHINO

Nonostante fosse stata prevista nell'hardware da parecchio tempo, l'attivazione della programmazione da remoto ha richiesto un notevole impegno sul piano software, un po' a causa di limiti hardware del modulo WiFi (uno dei pin previsti è risultato inadatto allo scopo perchè durante l'avvio genera impulsi che disturbano) un po' per l'IDE di Arduino, piuttosto problematico quando si tratta di utilizzare IP dinamici nella connessione. Con la versione 5.0.0 del firmware e la contestuale 5.0.0 del programma di aggiornamento FishinoFlasher siamo finalmente riusciti nell'impresa, e questo per tutte le versioni di Fishino, sia quelle ad 8 bit (Uno, Mega e Guppy), sia quelle a 32 bit (Fishino32 e la Fishino Piranha di prossima pubblicazione). Fishino diventa quindi programmabile, sempre utilizzando l'IDE di Arduino, anche attraverso la rete WiFi, senza più bisogno di connessione USB.

Questo apre moltissime possibi-

lità: basti pensare ad un sensore remoto, magari situato in mansarda o in cantina, alimentato da un pannello solare, che diventa riprogrammabile senza necessità di doverlo smontare, né di doverlo raggiungere con il nostro laptop, magari piazzandoci in posizioni tutt'altro che comode.

ATTIVAZIONE DELLA PROGRAMMABILITÀ REMOTA

Prima di entrare nel vivo della programmazione OTA (Over The Air, ossia via radio) occorre distinguere tra le versioni ad 8 bit e quelle a 32 bit della nostra scheda Fishino; se per queste ultime non è necessaria alcuna connessione, grazie allo sfruttamento di I/O interni alla PIC utilizzata, per le versioni ad 8 bit occorre "sacrificare" un I/O sul modulo WiFi, e per la precisione il GPIO5, che va connesso con un ingresso reset previsto in origine proprio per questo scopo, il pin ATM-RES.

Il collegamento per la scheda Fishino Uno è illustrato nella Fig. 1. Lo stesso collegamento va effettuato sui modelli Mega e Guppy, prendendo come riferimento per l'orientamento il modulo WiFi.

Come già detto, per le schede a 32 bit (nelle quali tra l'altro non è presente il connettore ESPCONN) non va effettuato alcun collegamento aggiuntivo.

INSTALLAZIONE DEI PACKAGES NELL' IDE DI ARDUINO

Il passo successivo, che chi ha utilizzato il modello a 32 bit già conosce, è l'installazione nell'IDE di Arduino di un package contenente la definizione delle schede Fishino e i relativi programmi necessari per l'upload senza fili. L'installazione del package è semplicissima; innanzitutto occorre aprire, nell'IDE, la finestra

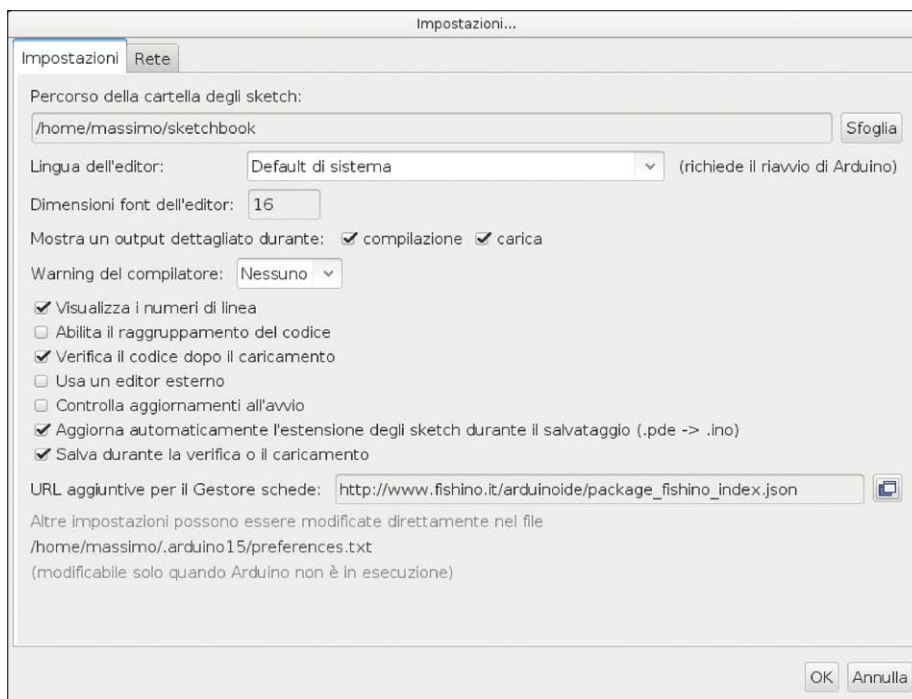


Fig. 4 - URL aggiuntivo inserito nella finestra di dialogo.

di dialogo Impostazioni (aprire il menu *File* e impartire il comando *Impostazioni*) che si presenta come mostrato nella Fig. 2.

Notate il punto "URL aggiuntive per il gestore schede", con la casella di testo alla sua destra e il pulsante dopo di questa; facendovi clic, appare la finestra di dialogo mostrata nella Fig. 3.

Come si vede dall'immagine, in questa finestra è possibile inserire più righe contenenti gli URL

(indirizzi web) dove si trovano i package necessari alle schede non originariamente previste dall'IDE. In questo caso abbiamo inserito l'URL relativa al package per Fishino, che è:

http://www.fishino.it/arduinoide/package_fishino_index.json

Dopo averlo inserito, è sufficiente fare clic sul pulsante OK per tornare alla schermata precedente.

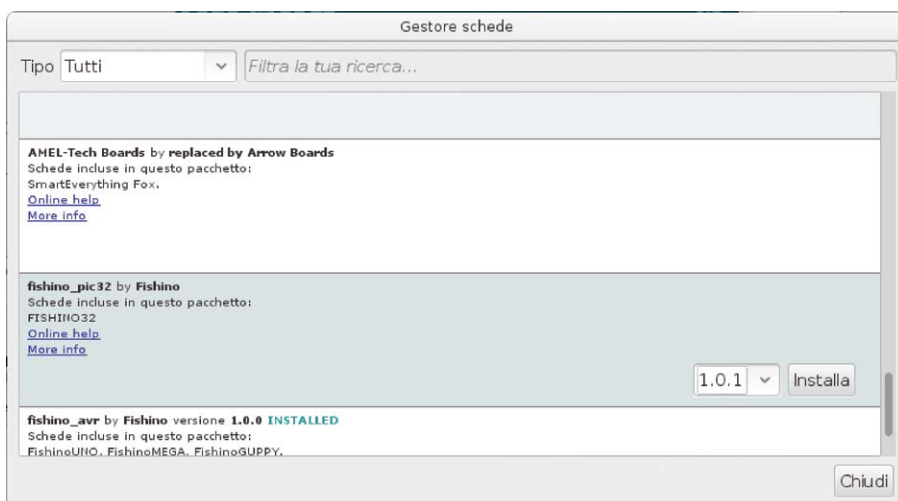


Fig. 5 - Finestra del gestore di schede.

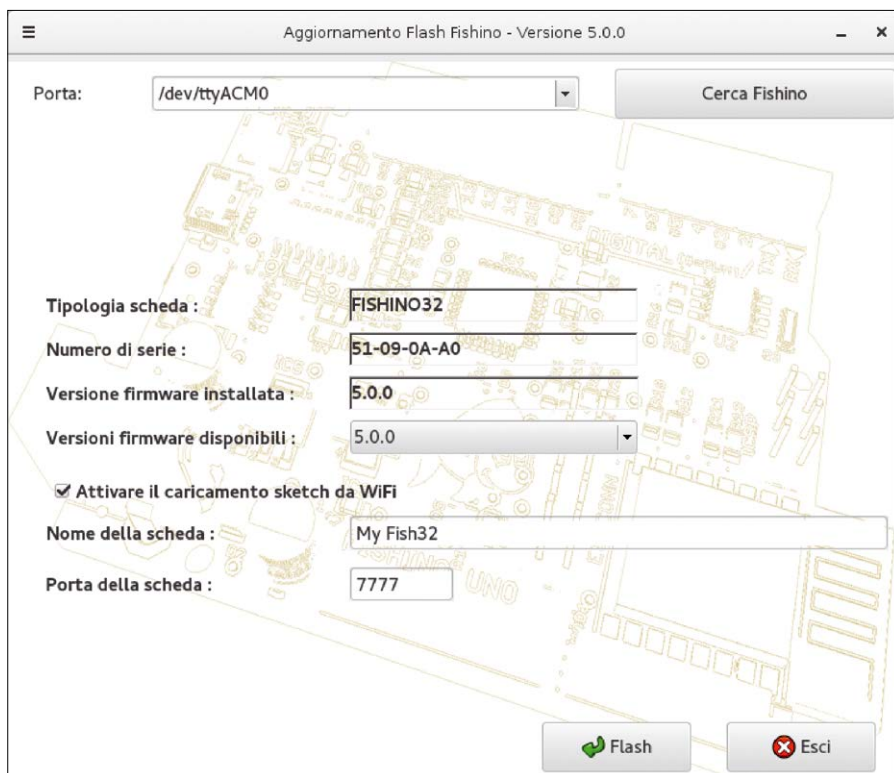


Fig. 6 - La finestra di Fishino Flasher aggiornata.

te, nella quale si può ora vedere la riga inserita contenente l'URL (Fig. 4).

Se avete già utilizzato questa funzione, magari per installare hardware tipo schede basate sul chip WiFi ESP8266, nella finestra precedente troverete già una o più righe inserite, alle quali dovrete aggiungere il nostro percorso di rete. Facendo clic sul pulsante OK accettiamo le modifiche e torniamo alla schermata principale dell'IDE.

Una volta inserita l'URL, occorre aprire il gestore di schede, scegliendo il comando di menu *Strumenti>Scheda>Gestore schede...*; se avete installato molte schede è possibile che il predetto comando *Gestore schede* si trovi fuori dalla zona visibile, nel qual caso è sufficiente spostare il puntatore del mouse verso la cima del menu fino a rintracciarlo. La schermata del gestore di schede è quella mostrata nella Fig. 5. Come potete vedere, alla fine sono presenti due voci: *Fishi-*

no_PIC32 e *Fishino_AVR*; nella Fig. 5 la *Fishino_AVR* risulta già installata, mentre la *Fishino_PIC32* non ancora; nel vostro caso appariranno entrambe non installate, quindi con a fianco attivato il pulsante INSTALLA.

Quindi selezionate la voce desiderata, ovvero *Fishino_AVR* per i modelli ad 8 bit e *Fishino_PIC32* per quelli a 32 bit e fate clic sul pulsante INSTALLA corrispondente. Ovviamente è possibile installare entrambi i package, nel caso possediate le due tipologie di scheda.

Per quanto riguarda la voce AVR l'installazione è rapidissima, mentre per la voce PIC32 occorre più tempo, essendo parecchi i package occorrenti da scaricare; questo avviene comunque solo la prima volta o quando ci sono aggiornamenti notevoli nel package. Come versione scegliete sempre l'ultima disponibile, che poi dovrebbe essere quella proposta nella casella, anche se un controllo in più non guasta.

Ed ora, l'unica nota dolente di tutta l'operazione: l'IDE ha un bug che impedisce (almeno, fino alla versione corrente, che è la 1.8.2) l'aggiornamento indolore dei package: se lo aggiornate ad una versione successiva, spesso e volentieri in fase di compilazione appare un errore del tipo "Scheda Fishino sconosciuta".

Quindi, nel caso si presenti il problema, suggeriamo di eseguire la procedura seguente.

1. Disinstallare i due package, facendo clic sul pulsante *Remove* che avrà sostituito quello *Installa*; nel caso ci sia la possibilità, selezionare tutte le versioni disponibili in casella e, se appare il pulsante *Remove* disinstallare anche quelle.
2. Chiudere l'IDE (questo passaggio è necessario).
3. Aprire la cartella **arduino15**, che è situata in percorsi differenti a seconda del proprio sistema operativo:
 - `/home/nomeutente/arduino15` per i sistemi linux;
 - `C:\Users\nomeutente\AppData\Local\Arduino15` per windows7 in inglese;
 - `/Users/nomeutente/Library/Arduino15` per Mac OS/X
 Nel caso abbiate una versione differente del sistema operativo occorre cercare la cartella, che si chiamerà comunque sempre **arduino15**, con o senza '.' davanti al nome, e con la prima lettera maiuscola o minuscola.
4. Entrare nella sottocartella **staging/packages**, ed eliminare tutti i file .zip il cui nome contenga il termine **Fishino**;
5. Tornare in **arduino15**, entrare nella sottocartella **packages** ed eliminare la cartella '**fishino**' contenuta.
6. Tornare in arduino15 ed eliminare i files .json contenenti **Fishino** nel nome.

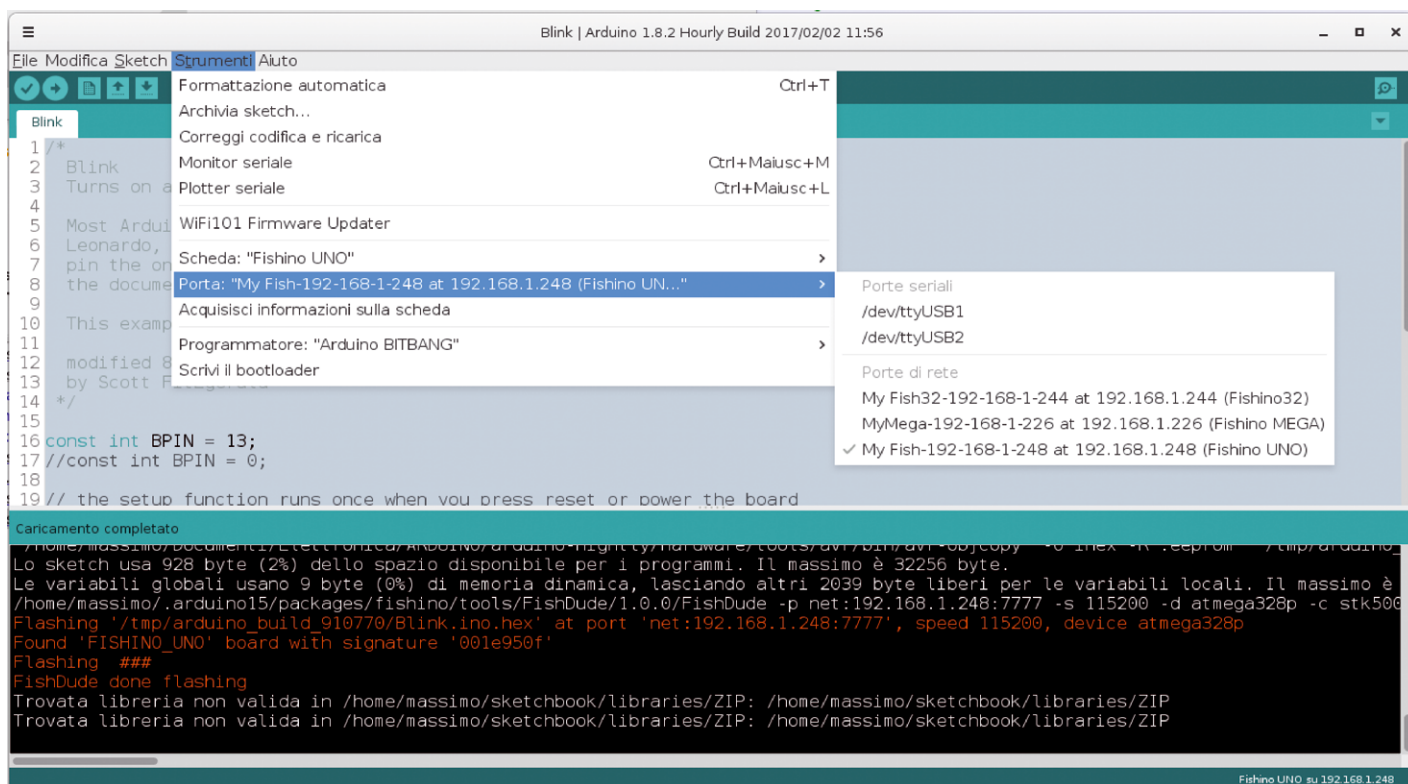


Fig. 7 - Il menu contenente le schede disponibili nell'IDE.

Completati questi passaggi, avrete ripulito completamente l'IDE dai package (vecchi e nuovi!) e dovrete ripetere l'installazione dall'inizio.

Volendo, è possibile eliminare una quantità minore di file e risparmiare il tempo occorrente alla re-installazione, ma consigliamo questa procedura per evitare intoppi.

DOWNLOAD DEL NUOVO FISHINOFLASHER

Anche il programma di aggiornamento firmware di Fishino è stato modificato per poter gestire le impostazioni del caricamento wireless degli sketch. In particolare, come vedremo tra breve, permette di impostare il nome della scheda, (come sarà visibile dall'IDE), e la porta TCP utilizzata per il caricamento.

L'applicazione può essere scaricata come sempre dalla pagina *Download* del sito www.fishino.it, oppure direttamente dai seguenti

indirizzi web:

- **Windows:** <http://fishino.it/wp-content/uploads/2015/09/FishinoFlasherWin.zip>;
- **Linux 32 bit:** <http://fishino.it/wp-content/uploads/2017/02/FishinoFlasherLinux32.zip>
- **Linux 64 bit:** <http://fishino.it/wp-content/uploads/2015/08/FishinoFlasherLinux64.zip>.

Esiste (novità in anteprima) una versione "beta" che richiede il pacchetto **XQUARTZ**, per **MAC OS/X**; potete scaricarla da: <http://fishino.it/wp-content/uploads/2017/02/FishinoFlasherMac.zip>. L'applicazione non ha bisogno di installazione: è sufficiente decomprimere l'archivio, metterla in una cartella a piacere ed eseguirla con un doppio clic del mouse.

FishinoFlasher si presenta con la schermata proposta nella **Fig. 6**. Rispetto alla versione precedente, potete notare le tre voci relative alla programmazione remota

che appariranno solamente se sceglierete un firmware dalla versione 5.0.0 in poi.

La prima voce permette di abilitare o meno la funzione, mentre le altre due contengono rispettivamente il nome della scheda come apparirà nell'IDE di Arduino e la porta TCP utilizzata. Per la prima suggeriamo l'utilizzo di un nome univoco per ogni scheda disponibile, con il limite che potranno essere utilizzati solo caratteri alfanumerici, spazi e caratteri '-' (il primo carattere dovrà essere alfabetico). Il limite massimo di lunghezza è di 45 caratteri.

Per la porta TCP abbiamo scelto come impostazione predefinita la numero 7777; vi è comunque possibile utilizzare una qualsiasi porta libera disponibile. In ogni caso vi suggeriamo di rimanere sui numeri alti, essendo quelli bassi riservati spesso a funzioni del sistema operativo.

Per il funzionamento di Fishino-

Flasher rimandiamo agli articoli che abbiamo pubblicato nei fascicoli precedenti (per esempio nel fascicolo n° 210 dove abbiamo descritto l'utilizzo della scheda Fishino 32) o al sito web di riferimento (www.fishino.it), dove troverete tutte le istruzioni necessarie.

Una volta che avrete abilitato il caricamento remoto, inserito il nome della scheda e il numero della porta da utilizzare per l'operazione, potrete aggiornare il firmware e iniziare ad usare la scheda con le nuove funzionalità.

UTILIZZO DELLA PROGRAMMAZIONE REMOTA

Lanciando l'IDE noterete subito che nel menu di scelta della porta seriale appariranno una o più porte di rete, a seconda di quante schede Fishino saranno state rilevate dal sistema.

Prestate attenzione a un dettaglio: il modulo WiFi si connette all'ultimo access-point utilizzato o comunque ad un numero limitato di access point "conosciuti"; questo vuol dire che al primo utilizzo (scheda mai utilizzata e/o dopo aggiornamenti firmware) il modulo non riuscirà a connettersi. Per ovviare a tale inconveniente è necessario caricare uno sketch qualsiasi che utilizzi il WiFi connettendosi all'access-point che vorrete venga poi utilizzato per l'accesso; questo occorre farlo nel modo tradizionale, via USB.

Le schede sono elencate nella finestra dell'IDE come viene illustrato nella Fig. 7.

Come possiamo vedere nella schermata proposta, vengono mostrati nell'ordine:

- il nome della scheda, così come l'abbiamo inserito nel FishinoFlasher;
- un trattino;
- l'IP attuale della scheda;

- la parola at e di nuovo l'IP della scheda tra parentesi;
- per concludere viene mostrato il modello della scheda (Fishino Uno eccetera).

Il "doppio IP" è stato necessario per evitare un altro bug dell'IDE Arduino, il quale, una volta che vede una scheda di rete, non è in grado di sostituirla con un'altra avente lo stesso nome; abbiamo quindi optato per "appiccicare" l'IP al nome della scheda in modo da renderlo univoco e permettere l'utilizzo di IP dinamici (e che quindi possono cambiare tra uno sketch e l'altro) senza dover per questo chiudere e riaprire l'IDE.

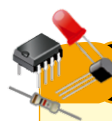
Vedrete, infatti, che spegnendo la scheda la porta sparirà (entro pochi secondi) e riapparirà quando ricollegherete la scheda, con lo stesso indirizzo IP oppure con un IP nuovo a seconda dello sketch caricato.

Per caricare lo sketch via WiFi sarà quindi sufficiente selezionare la porta corretta; a riguardo dovete prestare attenzione al nome della scheda: se come tipo scheda avete scelto Fishino Uno, dovete scegliere una porta WiFi il cui nome riporti tra parentesi Fishino Uno. Idem per le eventuali altre tipologie.

CONCLUSIONI

La programmazione OTA di Fishino è resa possibile dal fatto che il modulo WiFi, con il suo firmware, esegue la programmazione del controller in via hardware diretta, utilizzando quindi i pin del connettore ISP previsti per questo scopo, come se si trattasse di un programmatore esterno collegato alla scheda; viene quindi saltata la procedura tramite il boot-loader contenuto nel controller stesso, il quale, anzi, viene ricaricato ad

ogni riprogrammazione eseguita tramite WiFi. Questo significa che anche se il vostro bootloader si danneggia (cosa che ogni tanto capita) potrete comunque caricare gli sketch via WiFi e, al primo sketch ricaricato verrà ripristinato anche il bootloader originale. Un'ultimissima nota: il monitor seriale NON funziona quando la connessione alla Fishino avviene tramite WiFi; questo non è dovuto all'impossibilità di realizzare la "cosa" lato Fishino, ma per un limite dell'IDE, che considera tutte le connessioni remote del monitor seriale come connessioni via SSH ad un Arduino YÙN. Nel caso l'IDE Arduino venisse un giorno migliorato in tal senso, provvederemo ad aggiungere anche quella funzionalità. Tenetevi quindi informati sul nostro sito www.elettronica.in, nelle pagine della nostra rivista o sul sito di riferimento del progetto Fishino, www.fishino.it. ■



per il MATERIALE

Tutta la gamma delle board Fishino è disponibile presso Futura Elettronica. Fishino UNO (cod. FISHINO UNO) basata sull'ATmega328, è in vendita a Euro 36,00; Fishino MEGA (cod. FISHINOMEGA) basata sull'ATmega2560, è disponibile a Euro 49,90; Fishino GUPPY (cod. GUPPY) compatibile con Arduino NANO, costa Euro 33,90 mentre Fishino 32 (cod. FISHINO32) basata sul PIC32MX470F512, è in vendita a Euro 59,00. I prezzi si intendono IVA compresa.

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