Dotiamo Arduino di un modulo per comunicazioni long-range basato sulla tecnologia Semtech. Prima puntata.

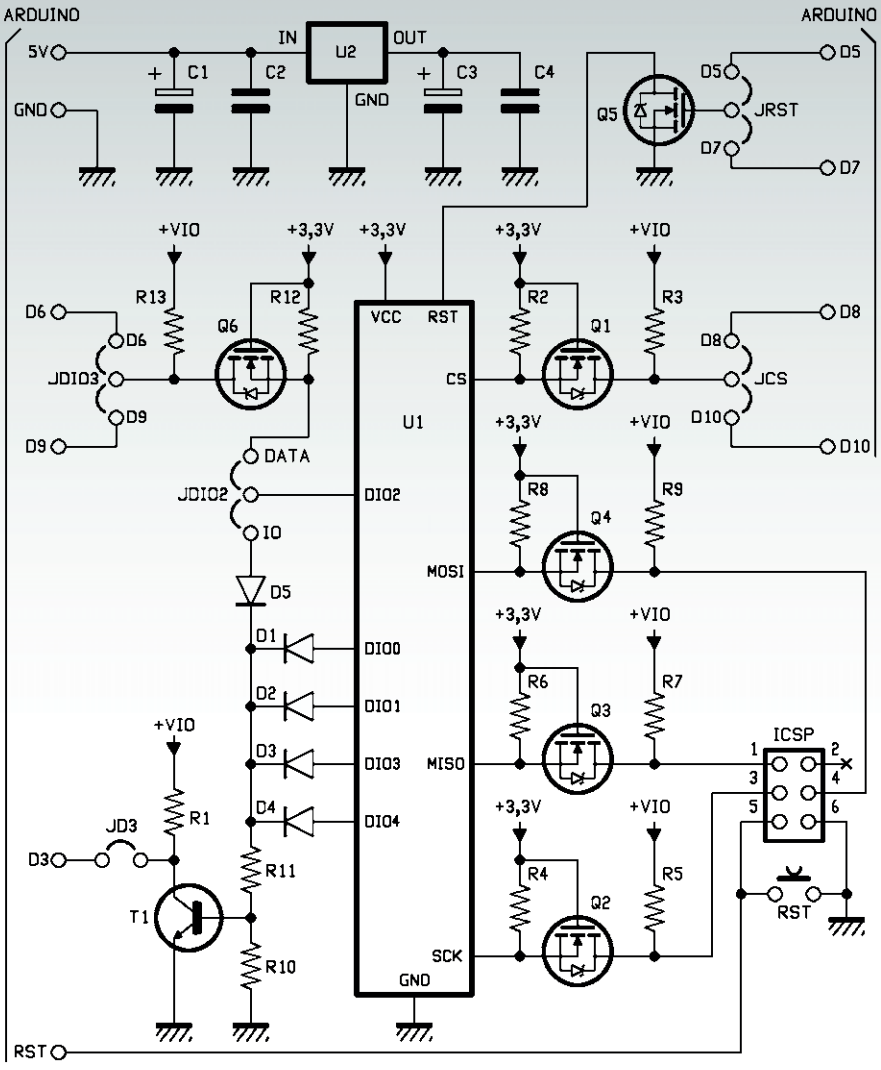
LoRa SHIELD

dell'ing DANIELE DENARO

Abbiamo avuto modo di parlarvi (nel numero 191) delle due tecnologie che si contendono il mercato della comunicazione wireless a lungo raggio (alcuni chilometri) e basso consumo: la tecnologia SigFox e la LoRa; su quest'ultima si basa la demoboard che abbiamo realizzato (e proposto nel fascicolo numero 194), che fa uso di un modulo LoRa realizzato dall'Aurel. Ora è venuto il momento di trasportare la tecnologia long-range wireless sulla piattaforma Arduino e per questo vi proponiamo uno shield che utilizza un modulo RTX LoRa ed è corredato dalla relativa libreria software per un utilizzo immediato. Prossimamente lavoreremo anche con la tecnologia SigFox, per la quale Atmel (casa produttrice dei microcontrollori di Arduino) ha preparato uno shield per Arduino, che presenteremo appena disponibile.
Prima di introdurre la scheda, è necessario riprendere brevemente i concetti base di queste due tecnologie. Innanzitutto chiariamo le motivazioni che hanno spinto alla ricerca di una comunicazione a lungo raggio ma a basso consumo:

- realizzare soluzioni IoT (Internet of Things), ovvero collegare via radio sensori ed attuatori distribuiti in quantità, ma senza ricorrere a Wi-Fi o WiMax, non adatte alla tipologia minimale dei sensori/attuatori e dispendiose in termini di energia, nonché soggette a saturazione;
- avere bassi costi per sensore/attuatore wireless.

Le due tecnologie sono differenti in termini di tecnica di trasmissione e di filosofia: mentre SigFox utilizza la tecnica della banda strettissima (UNB=Ultra Narrow Band), LoRa utilizza il concetto opposto dello spettro di emissione distribuito (Spread Spectrum); inoltre SigFox contempla una struttura a stella dove una stazione ricevente, più sofisticata dei singoli terminali, concentra la ricezione e la instrada su normale rete Internet.
In questa architettura i terminali sono, in genere, solo trasmittenti. Al contrario, LoRa ha una architettura orientata al punto-punto, e, con adeguato



software potrebbe realizzare una rete distribuita come Internet. Anche in questo caso si può pensare ad un instradamento su Internet realizzato da uno o più nodi di questa struttura, ma ciò non è determinante come in SigFox; insomma, LoRa è una struttura più flessibile di SigFox.
In ogni caso le implementazioni di ambedue i sistemi fanno ampio uso della tecnica SDR (Software Defined Radio), che consiste nella gestione software delle componenti dei ricetrasmettitori. I circuiti integrati sono perciò estremamente flessibili e configurabili (anche dinamicamente) sia in termini di frequenza fondamentale che di potenza e di molti altri parametri; possono realizzare, quindi, sistemi adattativi. In particolare l'implementazione LoRa della Semtech è molto flessibile, come vedremo più avanti. Dal punto di vista della comunicazione, ambedue i sistemi irradiano su frequenze minori di 1 GHz: SigFox, tipicamente sugli 868 MHz (in Europa) mentre LoRa sugli 868 MHz o sui 433 MHz (bande libere senza licenza). La potenza di emissione è decisamente bassa: dell'ordine di 10÷100 mW (corrispondenti a 10÷20 dBm).
Altra particolarità è che i dispositivi SigFox e LoRa, quando non trasmettono, praticamente non consumano elettricità; sono quindi l'ideale per le applicazioni alimentate con batterie, elementi fotovoltaici o con con soluzioni di "energy harvesting". La comunicazione può avvenire su distanze di chilometri o decine di chilometri. Ma come fanno, i dispositivi, a comunicare a queste distanze con una potenza così bassa? Ebbene, malgrado le diverse tipologie di modulazione adottate, LoRa e SigFox sono accomunati da un'altissima immunità ai disturbi ed alle interferenze, tali da ammettere sensibilità dell'ordine di -126 dBm (10÷13 mW). Ma, ovviamente nulla è gratis, e il prezzo da pagare è una bassissima velocità di comunicazione: alcune decine di bit al secondo (bps). Ma ciò non ha importanza nel tipo di utilizzo di questi sistemi, visto che è orientato a sensori/attuatori che devono scambiare brevissimi messaggi in maniera episodica (tipicamente qualche centinaio di volte al giorno).

MODULAZIONE

Lungi da pretendere una trattazione esaustiva di una problematica molto vasta e complessa come la trasmissione di segnali, facciamo il punto solo sui concetti base e sulla terminologia.
I segnali digitali possono modulare la frequenza portante sostanzialmente in modalità:

- AM (modulazione di ampiezza) detta anche ASK (Amplitude Shift Keying) di cui la più semplice implementazione è la OOK (On/Off Keying) nella quale allo zero corrisponde assenza di portante e ad uno la presenza di portante;
- FM (modulazione di frequenza) detta anche FSK (Frequency Shift Keying); spesso usata nella forma GFSK (Gaussian FSK) dove viene applicato un filtro gaussiano al segnale

SDR: la radio definita da software

La tecnologia "Software Defined Radio" introduce la programmabilità in ciò che finora era un vero tempio dell'hardware: l'elettronica degli apparati di radiocomunicazione. Complice il passaggio, ormai pressoché totale, dei segnali modulanti dall'analogico al digitale, e l'avanzare delle tecnologie digitali nella sintesi di frequenza, nei filtri e nella conversione del segnale, è inevitabile immaginare un apparato radio composto da moduli configurabili sia nella loro funzionalità che nel loro collegamento. Ecco allora i sistemi SDR, che possono dirsi sistemi radio generici, adattabili di volta in volta a frequenze, modulazioni, sensibilità e potenze varie mediante configurazione software.
La famiglia SX12xx della Semtech rappresenta un esempio di questo approccio.

per ammorbidire gli impulsi squadrati e ridurre così lo spettro; in questo modo la FSK modula quasi in modo analogico; la FM ha uno spettro maggiore rispetto alla AM ma è meno soggetta ai disturbi;
- PM (modulazione di fase) detta anche PSK (Phase Shift Keying); è la più efficiente ma anche la più complicata.

Esiste poi la modalità QAM che coniuga AM e PM (a diversi angoli di sfasamento) consentendo la trasmissione di più bit contemporaneamente e quindi aumentando il rateo dati.
Il segnale digitale modulante, poi, può avere diverse codifiche: NRZ (Non Return to Zero, il più semplice), RZ e ARZ (Alwais Return to Zero: in pratica il PWM), Manchester, codifiche bipolari, ecc.
La scelta della codifica influisce sullo spettro e su altre caratteristiche come la presenza o meno di una componente continua o la possibilità di perdere il sincronismo a causa di lunghe sequenze di zeri (come per NRZ).
Ma, in ogni caso, qualunque modulazione disperde la frequenza portante in una banda centrata sulla frequenza base. Ne consegue una dispersione di energia e una maggiore sensibilità al rumore o alle interferenze.

ULTRA NARROW BAND

Una possibile soluzione è restringere moltissimo lo spettro di modulazione ad alcune decine di hertz, sia riducendo il data-rate, che con accorgimenti di modulazione e filtri.
In SigFox la frequenza della portante in Europa è a 868 MHz, la modulazione è GFSK e il data-rate di 100 bps, con una larghezza dello slot di comunicazione di appena 100 Hz; quindi più terminali possono trasmettere contemporaneamente allocando lo slot disponibile. Poiché lo spettro è strettissimo, tutta l'energia è concentrata e di conseguenza ne basta poca perché il segnale sia d'ampiezza superiore al rumore.

SPREAD SPECTRUM

Un'alternativa è distribuire un segnale (ancora una volta con un basso rateo) su una banda molto

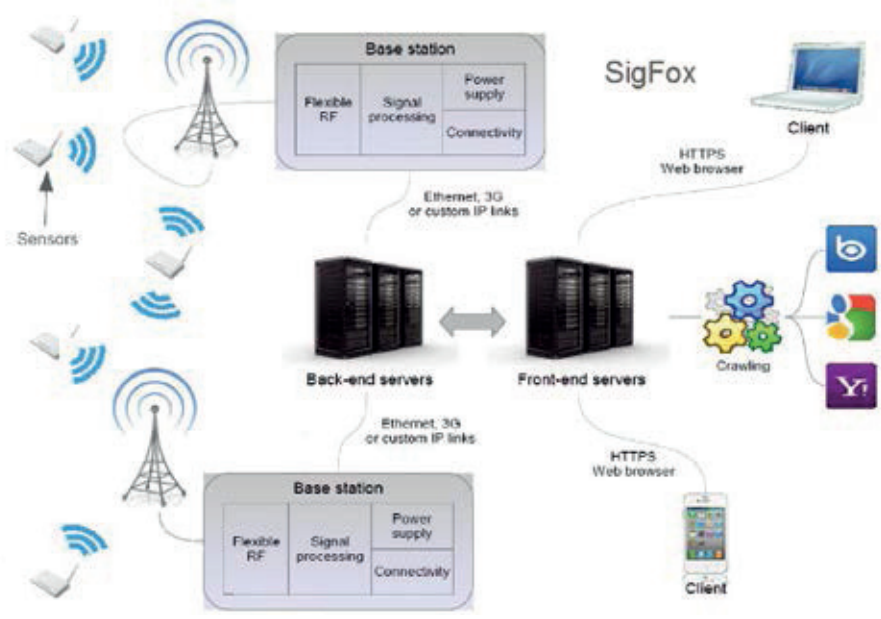


Fig. 1 *- Schema del sistema SigFox.*

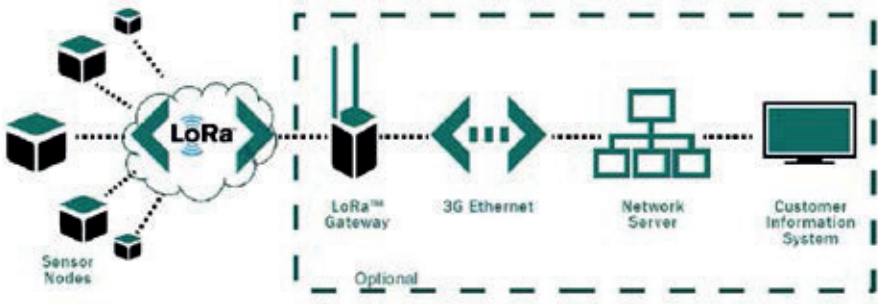


Fig. 2 *- Schema del sistema LoRa.*

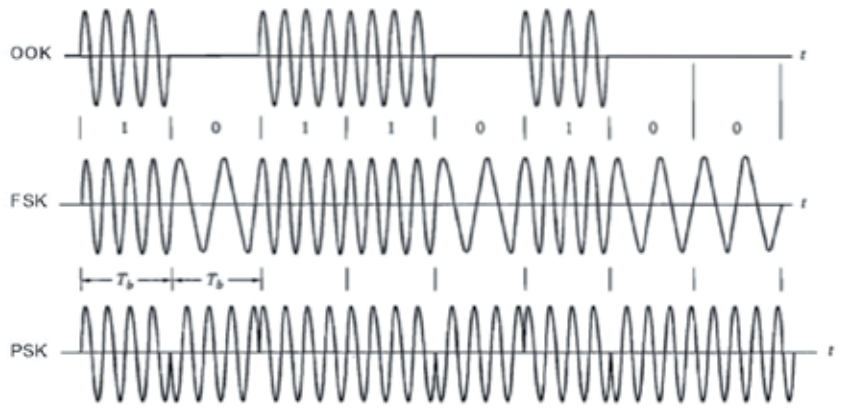


Fig. 3 *- Le modulazioni di ampiezza, frequenza e fase.*

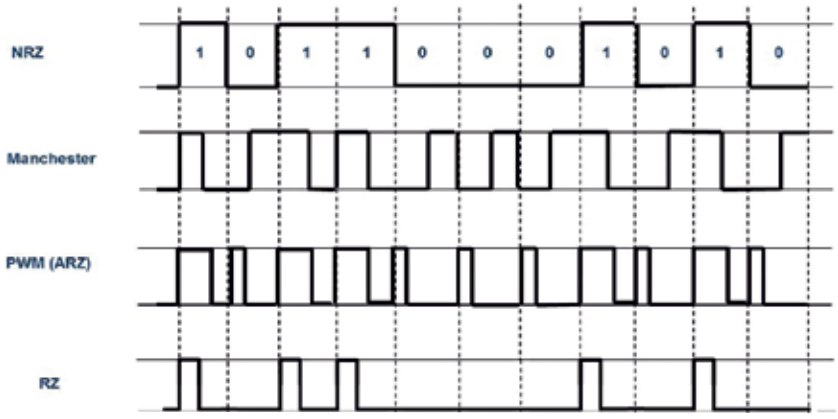
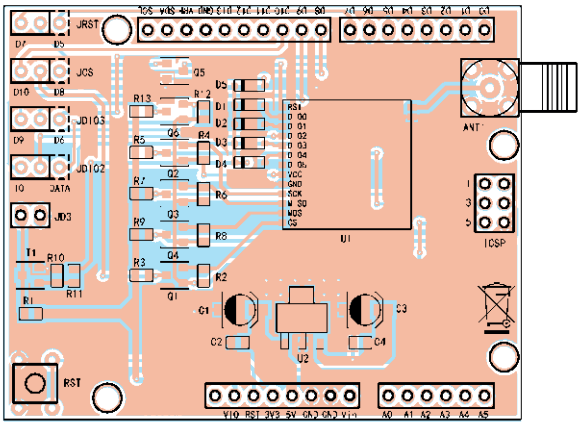


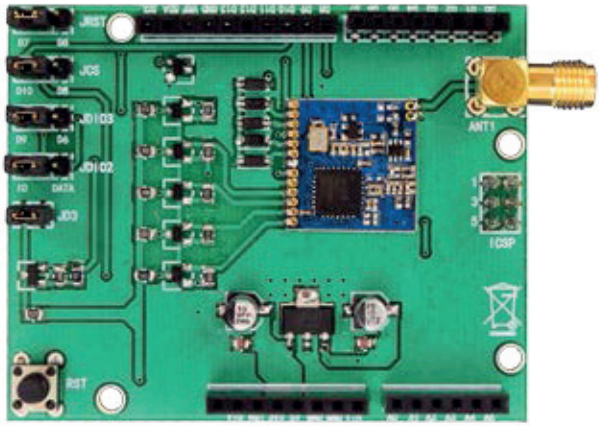
Fig. 4 *- Codifiche del segnale modulante.*

superiore a quella necessaria, utilizzando una qualche funzione di dispersione. La ricostruzione avviene tramite una funzione inversa.
Quella di dispersione è in genere una funzione pseudo-randomica, oppure una funzione che provoca un continuo salto di frequenza (Frequency Hopping) o, infine, una combinazione delle due tecniche. Il risultato è una dispersione del segnale a livello del rumore, il che rende difficile l'intercettazione e l'interferenza. Anche in questo caso l'energia è molto bassa, sebbene lo spettro sia molto ampio, perché non importa se il segnale si trova allo stesso livello del rumore, dato che il segnale può essere ricostruito dall'elaborazione. Questa tecnica è stata mutuata dal settore militare ed infatti realizza una sorta di "camouflage" via etere. Nel LoRa, la banda di dispersione può essere fissata da 7,8 kHz a 500 kHz.
Inoltre ogni byte del segnale è modulato su più frammenti radio (Spreading Factor). Questo numero può essere definito da 64 a 4.096.
Per cui il rateo effettivo può andare da circa 18 bps (minima banda, massimo spreading factor) a 78 kbps (massima banda, minimo spreading factor).
Ma anche la sensibilità ed il valore SNR (segnale/rumore) ne sono influenzati; in particolare, un aumento dello spreading factor aumenta il valore SNR e della sensibilità, mentre un aumento della banda diminuisce la sensibilità. Nello specifico, si va da -5 dB a -20 dB per l'SNR e da -124 dBm a -134 dBm per la sensibilità.

LO SHIELD PER ARDUINO

Lo shield che abbiamo preparato utilizza la tecnologia LoRa e si basa su un modulo DRF1278F della Dorji, il quale a sua volta monta il chip SX1278 della Semtech, che è il cuore del sistema. L'SX1278 è un componente molto sofisticato e versatile, che fa ampio uso dell'SDR, per cui è completamente configurabile.
La prima cosa da mettere in risalto è che lo SX1278 non è dedicato al solo sistema LoRa, ma può trasmettere e ricevere anche nelle classiche modalità FSK o OOK. In questi casi la comunicazione può avvenire con un protocollo a pacchetto con codifica NRZ, oppure perfino con una codifica qualunque, inserendo (o ricevendo) gli impulsi direttamente attraverso un pin di I/O. In quest'ultimo caso il chip funziona come semplice transceiver.
L'SX1278 fa parte della famiglia SX12xx, e si distingue dagli altri membri per il range di frequenza della portante, che può andare da 123 a 525 MHz; gli altri possono arrivare anche a 1.020 MHz.
In questa prima parte descriveremo la struttura dello shield e l'utilizzo di esso nella modalità OOK o FSK, mentre nella seconda parte affronteremo la comunicazione in modalità LoRa.
La struttura della scheda è abbastanza semplice perché, oltre a contenere la schedina DRF1278F, ha un alimentatore riduttore da 5 a 3,3V, alcuni adattatori di livello (perché SX1278 lavora a 3,3V) e una combinazione in logica OR per una eventuale gestione degli interrupt che l'SX1278 può produrre sui suoi piedini di I/O. Inoltre, alcuni ponticelli permettono la scelta e l'abilitazione dei segnali con i pin di Arduino. Per esempio il chip select (negato) può essere collegato al pin D8 o D10 di Arduino, mentre il reset





Elenco Componenti:

R1÷R10: 10 kohm (0805)
R11: 4,7 kohm (0805)
R12, R13: 10 kohm (0805)
C1, C3: 47 µF 6,3 VL elettrolitico (Ø4mm)
C2, C4: 100 nF ceramico (0805)
U1: DRF1278F
U2: TC1262-3.3VDB
RST: Microswitch
Q1÷Q6: BSS138 (SOT23)
T1: BC817 (SOT23)
D1÷D5: MMSD4148T1G

Varie:
- Connettore SMA da CS
- Strip maschio 2 vie (1 pz.)
- Strip maschio 3 vie (4 pz.)
- Jumper (5 pz.)
- Strip Maschio/Femmina 3 vie (2 pz.)
- Strip Maschio/Femmina 6 vie
- Strip Maschio/Femmina 8 vie (2 pz.)
- Strip Maschio/Femmina 10 vie (1 pz.)
- Antenna stilo cod. ANTSMAGSM
- Circuito stampato S1190

di SX1278 può essere collegato al pin D5 o D7. L'OR degli interrupt può essere collegato al pin D3, che è uno dei due possibili interrupt esterni di Arduino. Ma il DIO2 di SX1278 può anche essere usato come input/output di impulsi (come vedremo più avanti), e può essere collegato a D6 o D9. Infine, il bus SPI per comunicare con lo SX1278 è collegato al connettore ICSP di Arduino.
È stato necessario inserire un alimentatore riduttore da 5 a 3,2 V perché l'assorbimento in fase di trasmissione (alla massima potenza) può arrivare a 120 mA e il convertitore a 3,3V di Arduino non è sufficiente a garantire l'assorbimento.
È opportuno tener presente che, mentre l'SX1278 ha due uscite per l'antenna, la schedina DRF1278F ne riporta all'esterno solo una: quella dedicata al range maggiore di potenza (uscita "boost"). Questa risulta collegata al connettore d'antenna in formato SMA. Ne consegue che per avere segnale in antenna è necessario abilitare questa uscita (che per impostazione predefinita non è abilitata) tramite la configurazione di un registro dell'SX1278 (registro 0x09).
La configurazione dell'SX1278 comporta l'aggiornamento dei suoi registri interni tramite il bus SPI. I registri di configurazione sono ben 112, di cui la maggior parte gestisce più funzionalità mediante raggruppamenti di bit. Inoltre alcuni hanno un diverso comportamento, o, più precisamente hanno più copie, a seconda della modalità: LoRa o FSK/OOK. Nel caso di dati pacchettizzati, questi vengono inseriti in una coda FIFO (First In First Out) tramite il bus SPI e poi estratti e serializzati dal chip nel momento della trasmissione. Viceversa, nel caso della ricezione, vengono accumulati dal chip nella coda da cui possono essere estratti tramite il bus SPI.
La coda è diversamente configurata e gestita nel caso della

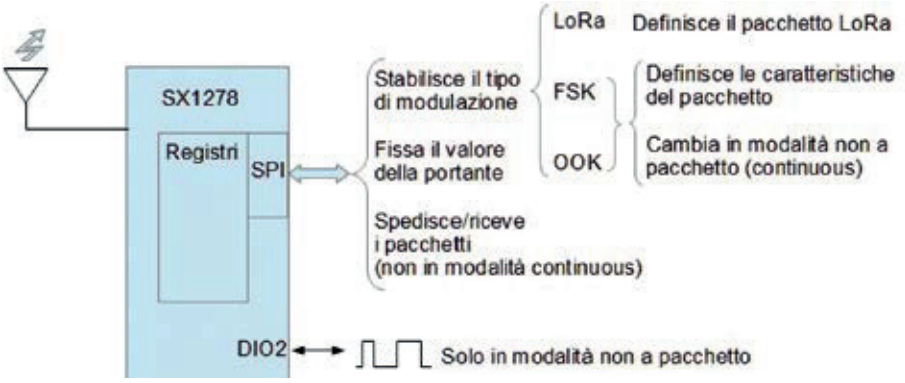


***Fig. 5** - Modalità di funzionamento dell'SX1278.*

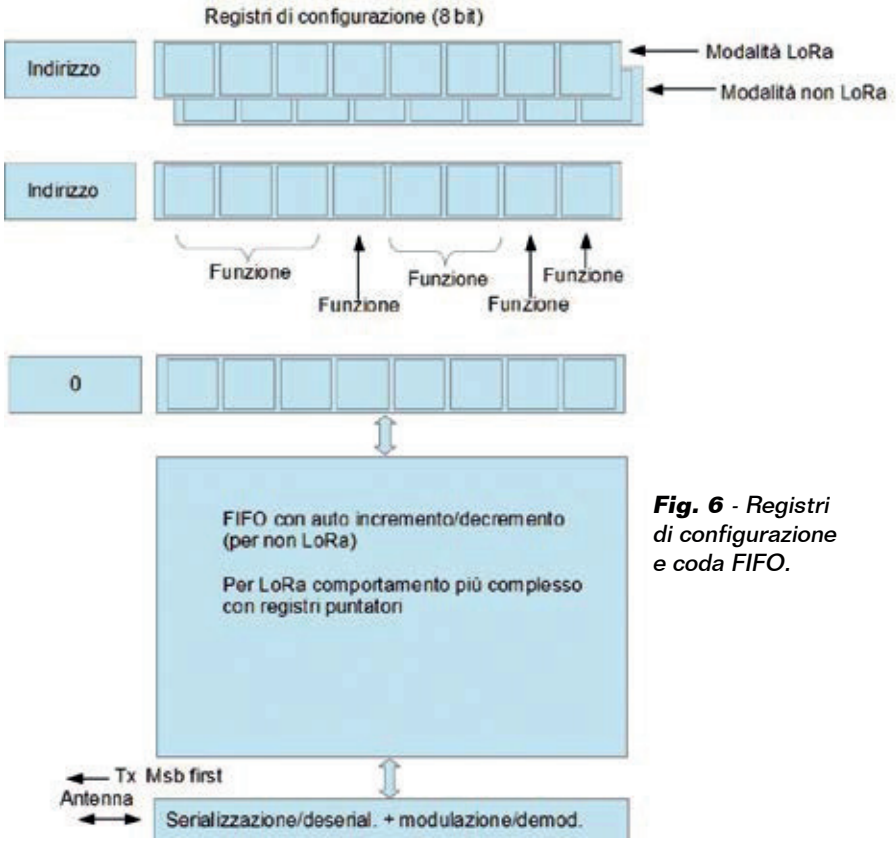


***Fig. 6** - Registri di configurazione e coda FIFO.*

modalità LoRa rispetto alla modalità non LoRa (FSK/OOK). La porta di comunicazione rimane, comunque, lo speciale registro di indirizzo 0.
Oltre alla porta per la coda FIFO (registro 0), uno degli altri registri comuni alle due modalità LoRa/NonLoRa è il registro 1, che definisce le seguenti modalità:

- bit 7; 0 = FSK/OOK 1=LoRa (predefinito 0=non LoRa);
- bit 6,5; 00=FSK 01=OOK (predefinito 00);
- bit 4,3 ; riservato o non applicabile al modello 78;
- bit 2,1,0; modalità operativa del chip:
  - 000 (0) Sleep mode;
  - 001 (1) Standby mode (predefinito);
  - 010 (2) FSTX mode (pronto a trasmettere);
  - 011 (3) TX (trasmette);
  - 100 (4) FSRX (pronto a ricevere);
  - 101 (5) RX (riceve pacchetto in FSK/OOK mode o riceve in modo continuo in LoRa mode);
  - 110 (6) RX SINGLE (solo in LoRa mode), riceve singolo pacchetto;
  - 111 (7) CAD (solo in LoRa mode), rilevazione attività nel canale.

Altri importanti registri sono 0x06,0x07,0x08, che definiscono insieme la frequenza della portante in Hz. In **Fig. 7** sono raggruppati i registri comuni alle due modalità.
Per un esame dettagliato si rimanda al Datasheet della serie SX12xx *http://www.semtech.com/images/datasheet/sx1276.pdf* .
I registri 0x40 e 0x41 sono destinati a configurare i pin di I/O che sono 6 (DIO0-DIO5); purtroppo queste configurazioni hanno un diverso significato a seconda della modalità. In ogni caso sono pin di output, eccetto DIO2, che può essere bidirezionale quando è configurato per la modalità *continuous*.
Gli altri pin sono soprattutto utili per veicolare all'esterno dei segnali di interrupt che comunque sono visualizzabili sui registri di flag 0x3E, 0x3F (per OOK/FSK mode) o 0x12 (per LoRa mode).
L'uso del interrupt permette una programmazione ad eventi, ma

Address	Register Name		Reset (POR)	Default (FSK)	Description	
	FSK/OOK Mode	LoRa™ Mode			FSK Mode	LoRa™ Mode
0x00	RegFifo		0x00		FIFO read/write access	
0x01	RegOpMode		0x01		Operating mode & LoRa™ / FSK selection	
0x06	RegFrfMsb		0x6C		RF Carrier Frequency, Most Significant Bits	
0x07	RegFrfMid		0x80		RF Carrier Frequency, Intermediate Bits	
0x08	RegFrfLsb		0x00		RF Carrier Frequency, Least Significant Bits	
0x09	RegPaConfig		0x4F		PA selection and Output Power control	
0x0A	RegPaRamp		0x09		Control of PA ramp time, low phase noise PLL	
0x0B	RegOcp		0x2B		Over Current Protection control	
0x0C	RegLna		0x20		LNA settings	
0x40	RegDioMapping1		0x00		Mapping of pins DIO0 to DIO3	
0x41	RegDioMapping2		0x00		Mapping of pins DIO4 and DIO5, ClkOut frequency	
0x42	RegVersion		0x12		Semtech ID relating the silicon revision	
0x4B	RegTcxo		0x09		TCXO or XTAL input setting	
0x4D	RegPaDac		0x84		Higher power settings of the PA	
0x5B	RegFormerTemp		-		Stored temperature during the former IQ Calibration	
0x61	RegAgcRef		0x13		Adjustment of the AGC thresholds	
0x62	RegAgcThresh1		0x0E			
0x63	RegAgcThresh2		0x5B			
0x64	RegAgcThresh3		0xDB			
0x70	RegPll		0xD0		Control of the PLL bandwidth	
others	RegTest		-		Internal test registers. Do not overwrite	

***Fig. 7** - Registri base e comuni.*

con l'interrogazione dei registri di flag si può, comunque, tenere sotto controllo la comunicazione, anche perché lo SX1278 gestisce molti passi in automatico.

COMUNICAZIONE IN MODALITÀ OOK O FSK

Come si è detto, la comunicazione in modalità OOK o FSK può avvenire sotto forma di pacchetti o in forma libera con l'intervento di Arduino che avrà il compito di serializzare/deserializzare i dati attraverso il pin D6/D9 che può essere collegato al DIO2 dell'SX1278.
Ma è molto più comodo ed efficiente far gestire la comunicazione completamente all'SX1278 utilizzando la modalità a pacchetto e caricando/scaricando i dati dalla coda tramite SPI. I pacchetti possono essere di lunghezza fissa o variabile.
Il pacchetto a lunghezza fissa è composto da:

- un preambolo (da 0 a 65.535 byte) composto da una sequenza di byte uguali (la lunghezza è definita dai registri 0x25,0x26); il tipo di byte può essere 0xAA (10101010) o 0x55 (01010101) (registro 0x27);
- campo di indirizzo (Sync), la cui lunghezza (da 0 a 8 byte) è definita nel registro 0x27 e i singoli byte sono definiti nei registri da 0x28 a 0x2F (ma non possono avere valore 0); il campo può definire un indirizzo (di rete o del terminale) perché quando è presente fa scartare automaticamente in ricezione il pacchetto se non corrisponde;
- Payload (lunghezza definita in 0x32); è l'unico che finisce nella coda FIFO e può avere lunghezza minima di 1 byte e massima di 2048 byte ed è composto da:
  - indirizzo utente (1 byte opzionale); è un ulteriore indirizzo.
  - messaggio;
  - CRC opzionale di 2 byte calcolato automaticamente dal chip.

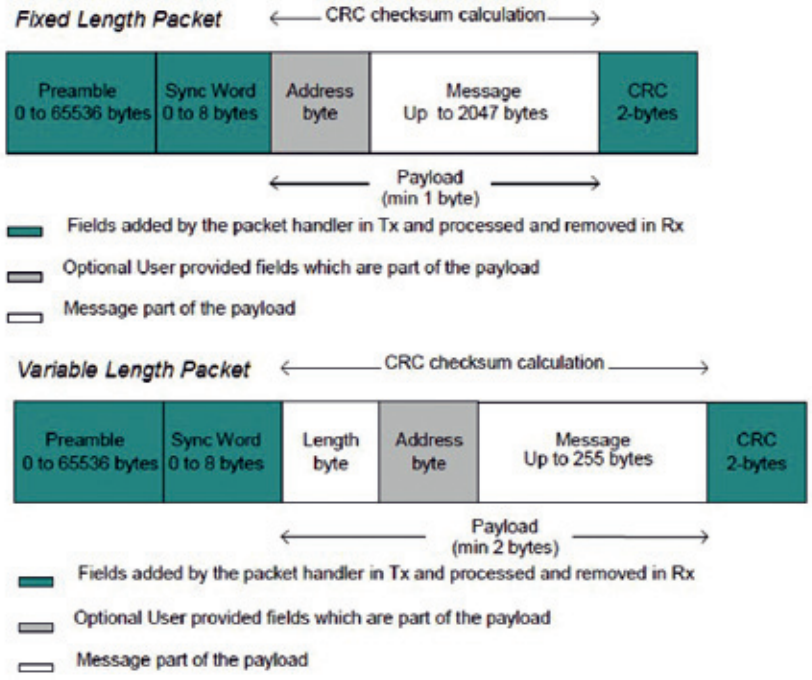


***Fig. 8** - Composizione dei pacchetti.*

Per quanto riguarda l'indirizzo utente, viene verificato se il registro 0x30 è configurato per "AddressFiltering"; in questo caso il pacchetto viene scartato se il byte di indirizzo non corrisponde a quello configurato nel registro 0x33. L'AddressFiltering può avvenire anche con un ulteriore controllo, prendendo in considerazione anche il registro 0x34 dove è stato inserito il "BroadcastAddress".
Il pacchetto di lunghezza variabile si differenzia per avere il primo byte del payload che indica la lunghezza e quindi consente un messaggio di 255 byte massimo.
Nel caso di comunicazione in modalità FSK/OOK, un settaggio fondamentale è quello della coppia di registri 0x02,0x03 i quali definiscono in maniera precisa il baud-rate della codifica NRZ. Infatti in modalità LoRa il protocollo proprietario ha altri parametri.
Come vedete, lo SX1278 è estremamente configurabile in termini di tipo di modulazione, frequenza portante, baud-rate e insieme ad altri moltissimi parametri consente un preciso "tuning" della comunicazione, anche se molti parametri sono definiti con valori predefiniti mediamente soddisfacenti. È quasi un modulo radio universale, nella logica della filosofia SDR.
Poiché la configurazione è notevolmente variabile, non possiamo esaurire l'argomento in questa sede; chi vorrà potrà consultare il data-sheet dell'SX1278.

UTILIZZIAMO LA SCHEDA PER COMANDARE LE PRESE RADIO-CONTROLLATE

Nel numero 196 di Elettronica In abbiamo descritto, in un apposito articolo, i protocolli di comunicazione delle prese di alimentazione a 220V radio-controllate (come i sistemi Avidsen o Velleman) e una realizzazione per controllare queste ultime tramite Arduino collegato a dei moduli radio a

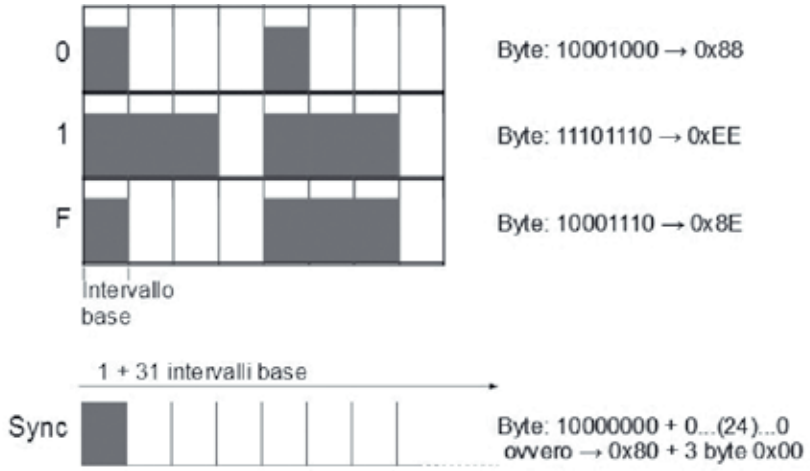


Fig. 9 *- Codifica degli impulsi dei sistemi radio-controllati.*

Fig. 10 *- Pacchetto per sistemi radio-controllati.*

basso costo. Nulla ci vieta di tentare la stessa strada con questa scheda, che contiene un così performante e flessibile sistema radio. La prima tentazione sarebbe quella di utilizzare la scheda nella modalità "*continuous*" e quindi come un normale modulo radio utilizzando la libreria software, descritta nell'articolo, per far costruire ad Arduino la sequenza degli impulsi previsti dal protocollo di comunicazione. Ma possiamo fare molto meglio: se vi ricordate le caratteristiche del protocollo utilizzato da questi sistemi, potete notare che, in effetti, la codifica dei valori three-state (0,1,Float) può essere immaginata come una sequenza di 8 bit, il cui baud-rate corrisponde alla frequenza dell'impulso base (vedi **Fig. 9**).

A questo punto teniamo presente che la codifica utilizzata nei pacchetti è di tipo NRZ e non essendo una trasmissione RS232, non ha i bit di start/stop, né la parità. Allora possiamo immaginare di trasformare il singolo frame del protocollo del radio-controllo in una sequenza di byte, ovvero un pacchetto. Basta porre a zero il preambolo e il campo Sync (indirizzo) e disabilitare il CRC. In realtà il pacchetto sarà composto da 4 ripetizioni di questo frame, come esige il proto-

Tabella 1 *- Libreria per Arduino: Classe SX1278 (funzioni generali e per la modalità FSK/OOK).*

Inizializza e reset	void begin(); void restart();
Attiva la modalità LoRa	void startModeLORA();
Attiva la modalità FSK/OOK	void startModeFSKOOK();
Decide se FSK (0) o OOK (1)	void setModulation(unsigned char mod); int readModulation();
Attiva il particolare modo operativo: SLEEP =0 STBY =1 FSTX =2 (pronto a trasmettere) TX =3 (trasmette) FSRX =4 (pronto a ricevere) RX =5 (se FSK/OOK) RXCONT (se LoRa) (riceve) RXSING =6 (solo LoRa) CAD =7 (solo LoRa)	void setStat(byte s); int readStat();
Frequenza portante (in Mhz)	void setFreq(float freq); float readFreq();
Bit Rate (se in modalità FSSK/OOK)	void setBPS(int bps); unsigned int readBPS();
Potenza di trasmissione; p può essere: 1= 7dBm(5mW) ; 2= 10dBm(10mW); 3= 13dBm(20mW) 4= 17dBm(50mW)(def); 5=20dBm(100mW)	void setPower(byte p);
Abilita(1)/disabilita(0) la modalità a pacchetto (def). Se disabilitata va in modalità "continuous"	void setPackNoPack(byte pk);
Definisce la funzionalità di ognuno dei 6 pin di I/O (per il codice val vedere tabella corrispondente sul datasheet)	void setIO(byte nio,byte val);
Lunghezza e tipo del preambolo del pacchetto	void setPreamble(unsigned int len, byte pol);
Indirizzo (rete): se presente, lunghezza, array di valori	void setSync(byte on,byte len, byte val[]);
Carica la coda FIFO con i dati da spedire	void dataToSend(byte data[],int len);
Legge dalla la coda FIFO i dati ricevuti	void dataReceived(byte data[],int len);
Legge il primo o il secondo registro di flag	byte getFlags(byte nreg);
Legge il flag n dal primo o dal secondo registro dei flag	byte getFlag(byte nflag, byte nreg);

***Tabella 2** - Libreria per Arduino: Classe REMOTEC.*

Inizializza (tutto)	void begin();
Switch on/off prese Avidsen	void avidsenSet(byte address[5],byte socket, byte onoff);
Switch on/off prese Valeman	void vellemanSet(byte address[7],byte socket, byte onoff);

collo del radio-controllo, per un totale di 64 byte. Il baud rate sarà definito in base all'inverso della lunghezza dell'impulso base. Vediamo di riassumere i passi necessari per realizzare questo "remote-controller".

1. Abilitare l'uscita boost collegata all'antenna, che per impostazione predefinita non è abilitata (reg. 0x09); con lo stesso registro si può regolare la potenza di trasmissione (quella predefinita è pari a 17 dBm, ossia 50 mW) che si può ridurre a 2 dBm(1,5 mW) in step di 1 dBm, oppure si può abilitare la extra potenza massima di 20 dBm (100 mW).
2. Porre il chip in modalità operativa "Sleep" (reg. 0x01) e attivare la modalità non LoRa (reg. 0x01); infatti questa fondamentale commutazione può essere effettuata solo quando il chip è in condizione "Sleep".
3. Settare la modalità OOK (reg. 0x01).
4. Definire la frequenza portante a 433,92 MHz (reg. 0x06,0x07,0x08).
5. Definire il baud-rate (per esempio 3.800 per Advidsen) (reg. 0x02,0x03).
6. Definire il pacchetto di lunghezza fissa di 64 byte senza preambolo, senza indirizzo e senza CRC (reg. 0x25, 0x26, 0x27, 0x30, 0x32).
7. Caricare i dati nel FIFO.
8. Portare il chip allo stato FSTX (pronto a trasmettere, reg. 0x01) ed attendere alcune decine di microsecondi.
9. Portare il chip allo stato TX (trasmette, reg. 0x01).
10. Portare eventualmente il chip allo stato "Sleep" (non necessario perché l'SX1278 si porta automaticamente allo stato "Standby" appena finita la trasmissione del pacchetto).

La qualità e l'efficienza dello SX1278 è decisamente maggiore dei moduli radio a basso costo presi in considerazione nel precedente articolo, anche se la scheda ha un costo abbastanza contenuto; ed inoltre scarica completamente Arduino dal compito gravoso di serializzare il treno di impulsi. Ne risulta un sistema affidabile e di buona portata radio. Tenete presente che quella proposta è solo una delle applicazioni possibili dell'SX1278.

LA LIBRERIA PER ARDUINO

Come avete potuto notare, la gestione del componente SX1278 non è affatto semplice a causa dei numerosi registri e delle moltissime funzionalità; abbiamo, perciò, scritto una libreria per semplificarne l'uso con Arduino. Sono state definite, innanzitutto, le funzioni base per leggere e scrivere nei registri appoggiandosi alla classica libreria SPI per Arduino. Successivamente sono state implementate le funzioni per i principali settaggi dell'SX1278. La libreria è comunque un cantiere aperto per l'implementazione delle funzionalità più

Listato 1

```
/*************************************************************************************************************
*  Commandi (seriale a 9600):
*     ? : risponde con il nome dello sketch (utile per RandA)
*     onx : switch on;  dove x=1,2,3,4,5  (1,2,3 per Velleman) (reply "ON")
*     ofx : switch off; dove x=1,3,3,4,5  (1,2,3 per Velleman) (reply "OF")
*     av  : predispone lo sketch per il sistema r Avidsen  (default) (reply "OK")
*     ve  : predispone lo sketch per il sistema  Velleman  (reply "OK")
*     adxxxxxxx : definisce l'indirizzo (three state es.: ad0200220) (solo 5 char per Avidsen)
*     pwx  : regola la potenza di trasmissione;
*               dove x=1(5mW), 2(10mW), 3(20mW), 4(50mW)(default), 5(100mW)
*************************************************************************************************************/

#include "SX1278.h"
#include "REMOTEC.h"
#include <SPI.h>

#define ln 10

REMOTEC RC;

/* valori di default (inizialmente su Advisen con indirizzo 2020222) */
byte addAv[5]={2,0,2,0,0};int alenAv=5;
byte addVl[7]={2,0,2,0,2,2,2};int alenVl=7;
byte* add=addAv;
int alen=alenAv;

int mode=1; //modo Advidsen
```

(continua)

Listato 1 (segue)

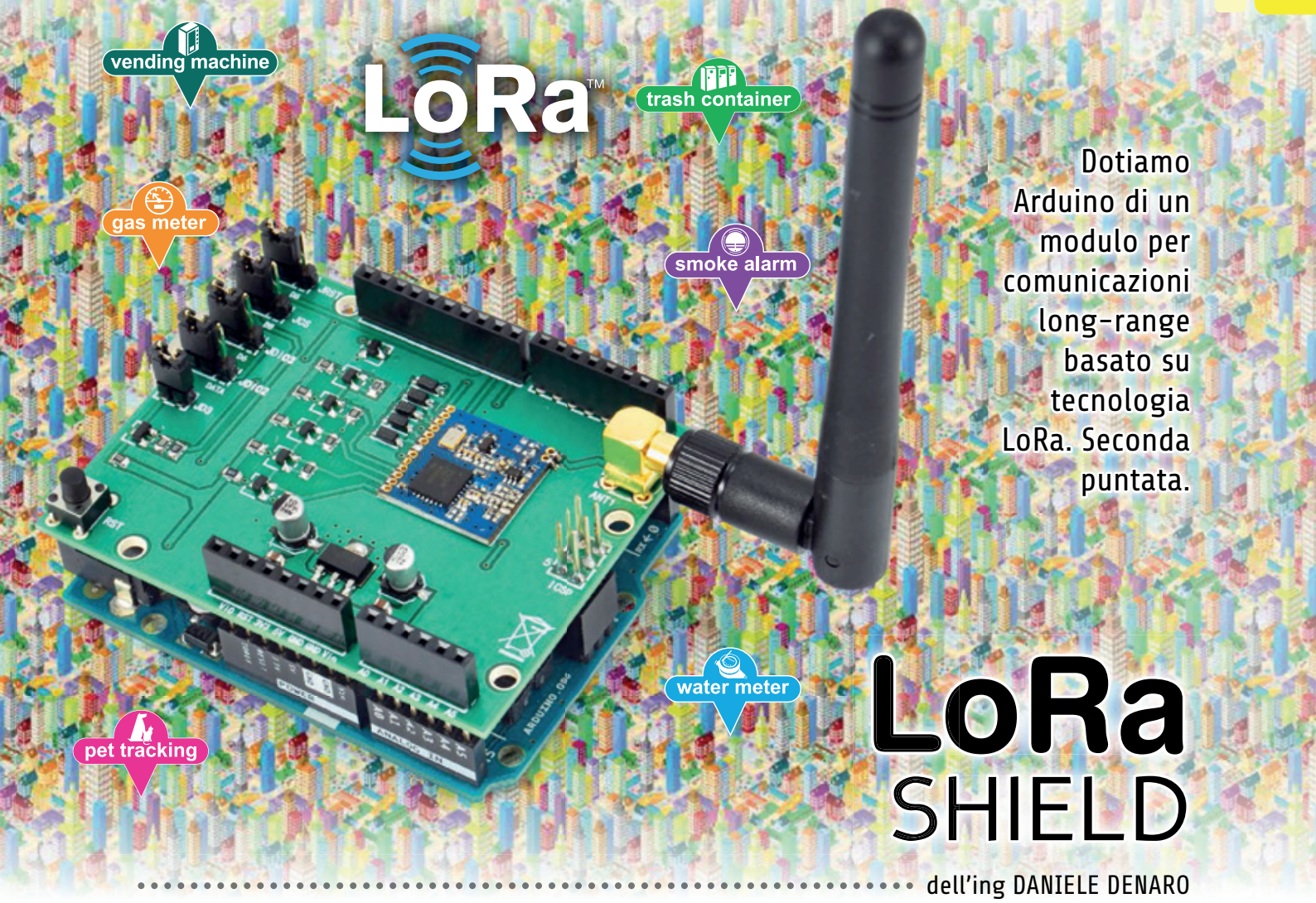
```
char buff[32];

byte sock;
byte onoff;

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

void loop()
{
  int c=Serial.available();
  if(c>0)
  {
   Serial.readBytesUntil(ln,buff,32);
   if (buff[0]=='?') itsMe();
   if (strncasecmp(buff,"on",2)==0)  commandSend(1);
   if (strncasecmp(buff,"of",2)==0)   commandSend(0);
   if (strncasecmp(buff,"av",2)==0)  setmodeavidsen();
   if (strncasecmp(buff,"vl",2)==0)   setmodevelleman();
   if (strncasecmp(buff,"ad",2)==0)  setaddress();
   if (strncasecmp(buff,"pw",2)==0)  setpower();
  }
}

/* Funzione di riconoscimento; permette di sapere quale sketch è caricato su Arduino*/
void itsMe()
{
  char name[]=__FILE__;
  char* c=strchr(name,'.');
  if (c != NULL) *c='\0';
  Serial.println(name);
}

void setmodeavidsen()
{
  mode=1;
  add=addAv;
  alen=alenAv;
  Serial.println("OK");
}

void setmodevelleman()
{
  mode=2;
  add=addVl;
  alen=alenVl;
  Serial.println("OK");
}

void commandSend(byte onoff)
{
  byte sock=atoi(&buff[2]);
  if (mode==1)  RC.avidsenSet(add,sock,onoff);
  if (mode==2)  RC.vellemanSet(add,sock,onoff);
  if (onoff==1)Serial.println(" ON"); else Serial.println(" OF");
}

void setaddress()
{
  int i;
  if (mode==1){for(i=2;i<7;i++) addAv[i-2]=buff[i]-'0';}
  if (mode==2){for(i=2;i<9;i++) addVl[i-2]=buff[i]-'0';}
  Serial.println("OK");
}

void setpower()
{
  byte power=atoi(&buff[2]);
  if ((power<1)|(power>5)){Serial.println("NK");return;}
  SX.setPower(power);
  Serial.println("OK");
}
```

sofisticate. Nella **Tabella 1** potete vedere elencate le principali funzioni. Sulla base di questa libreria (classe SX1278) è stata creata una ulteriore libreria (classe REMOTEC) finalizzata al controllo remoto delle prese 220V.
Nella **Tabella 2** sono descritte le funzioni di questa libreria.
Ne consegue che lo sketch del **Listato 1** è molto semplice. Lo sketch legge dalla seriale il comando e lo esegue rispondendo sulla seriale con una stringa "ON","OF" o "OK". Lo sketch è stato pensato (e verificato) anche per un utilizzo con RandA.
Potete immaginare la quantità di utilizzi che si possono facilmente implementare: dall'accensione/spegnimento programmato agli allarmi o alla gestione intelligente di carichi. Bene, con questa puntata abbiamo concluso; nella prossima analizzeremo le modalità di ricezione FSK/OOK e soprattutto affronteremo la modalità LoRa che permette a due schede di comunicare su lunghe distanze.

per il MATERIALE

La shield Long Range (cod. FT1190M) comprensiva di modulo radio e tutti i componenti necessari al suo funzionamento, è disponibile a 39,50 Euro. Il modulo DRF1278F è disponibile anche separatamene a 19,00 Euro. Arduino UNO (cod. ARDUINOUNOREV3) viene venduto a 24,50 Euro. Tutti i prezzi si intendono IVA compresa.

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



Dotiamo Arduino di un modulo per comunicazioni long-range basato su tecnologia LoRa. Seconda puntata.

LoRa SHIELD

dell'ing DANIELE DENARO

Avete avuto modo di conoscere, il mese scorso, un nuovo shield per Arduino basato sul chip SX1278 della Semtech, il quale, a fronte di un costo molto contenuto, permette di implementare una comunicazione radio in una vasta gamma di applicazioni: dalla trasmissione dei segnali digitali in modalità OOK o FSK, alla comunicazione su lunga distanza in modalità LoRa. Il chip, infatti, è stato concepito con un reale approccio alla filosofia SDR. Ricordiamo che la filosofia SDR consiste nel rendere l'apparato di trasmissione completamente configurabile e adattabile, facendo largo uso delle tecniche digitali in sostituzione di quelle analogiche; ciò rende l'SX1278 tanto flessibile e programmabile. La sua programmabilità è basata su un centinaio di registri di configurazione; per superare la complessità che questo comporta, abbiamo sviluppato una libreria che permette una veloce configurazione dei principali parametri. Per un approfondimento delle caratteristiche di SX1278 e dei suoi registri rimandiamo al datasheet scaricabile all'indirizzo: *www.semtech.com/images/datasheet/sx1276.pdf*.
Nella prima parte del progetto abbiamo illustrato gli aspetti principali dell'SX1278 e della libreria corrispondente nell'utilizzo dello shield in modalità trasmittente OOK; abbiamo anche visto una pratica e comoda implementazione di un controllore remoto dei sistemi Velleman o Avidsen per la gestione di prese di alimentazione a 220V. In questa puntata affronteremo la ricezione nei sistemi OOK o FSK e soprattutto la modalità di comunicazione LoRa.

LA RICEZIONE IN MODALITÀ OOK O FSK

Ricordiamo che l'SX1278 Semtech consente due modalità di trasmissione non a standard LoRa (lungo raggio), di seguito descritte.

1. Modalità elementare in cui l'SX1278 funziona come semplice modulo radio e i dati vengono

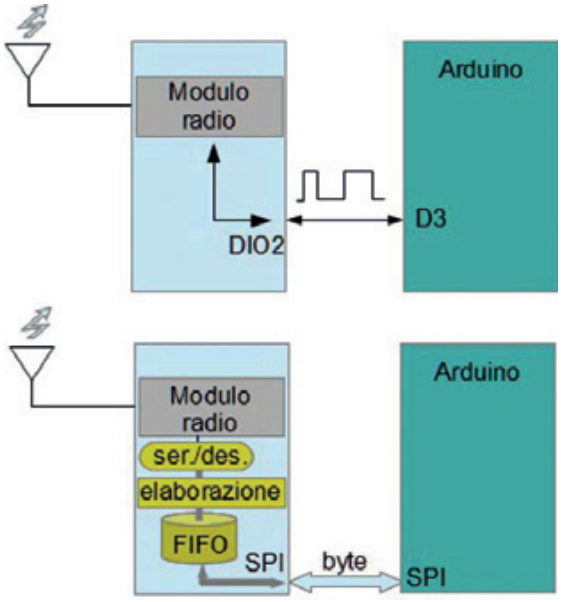


Fig. 1 *Modalità elementare ("continuous") ed a pacchetto.*

serializzati/deserializzati da Arduino tramite il pin I/O DIO2 dell'integrato Semtech collegato al pin D3 di Arduino.

2. Modalità a pacchetto, dove i dati vengono inseriti/recuperati byte per byte in/da una coda di trasmissione/ricezione e in cui l'SX1278 può agire con una notevole autonomia nella gestione del collegamento radio.

La modalità a pacchetto prevede una serie di dati aggiunti dall'SX1278 per sincronizzare la ricezione e un indirizzo di rete che può far scartare il pacchetto se non corrisponde a quello impostato sulla scheda ricevente. Per questi motivi non possiamo utilizzare la modalità a pacchetto per provare a ricevere e decodificare i radiocomandi dei sistemi di prese radiocontrollate. Se per la trasmissione, l'eliminazione del preambolo, dell'indirizzo (Sync) e del CRC non ha avuto alcun effetto, in ricezione tali elementi sono necessari per la sincronizzazione e la validazione e non possono essere rimossi. Per questo motivo, volendo realizzare una funzione di scanner per rilevare la codifica dei telecomandi dei sistemi Velleman o Avidsen, siamo stati costretti ad utilizzare la modalità elementare. Questa volta, quindi, Arduino riceve gli impulsi sul pin D3 e prova a decodificarli.

La ricezione degli impulsi è compito del modulo radio, quindi è comune alla modalità a pacchetto e a quella "continuous". Nel caso di modulazione OOK, si basa su tre possibili alternative:

1. gestione adattativa della soglia di rilevazione dell'impulso; è la modalità predefinita e consiste nell'adattare la soglia del comparatore al valore (continuamente campionato) del segnale radio (RSSI) diminuito di 6 dBm;
2. soglia fissata manualmente mediante il registro 0x15;
3. soglia continuamente aggiornata come media di RSSI (è la modalità più complessa).

Nel caso di soglia adattativa è possibile regolare il tempo di riassestamento della stessa dopo la fine dell'impulso, per ottimizzare il comportamento in casi particolari.

In verità anche la frequenza del campionamento del valore RSSI può essere cambiata tramite il registro 0x0E, anche se il valore predefinito è in genere adeguato.

Nel caso di modulazione FSK, può essere applicato un filtro Gaussiano con tre diversi valori, oppure può essere aggiustata la velocità di salita o discesa dell'impulso.

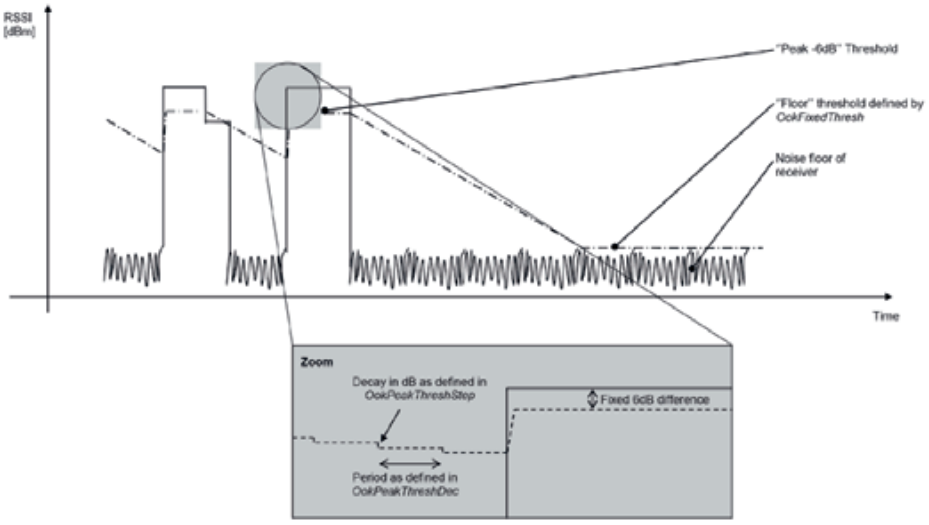


Fig. 2 *- Soglia adattativa nella demodulazione OOK.*

SCANNER PER LA DECODIFICA DEI TELECOMANDI

Ulteriori funzioni sono state inserite nella classe REMOTEC per realizzare questa ricezione semplificata; le trovate nella **Tabella 1**. Lo sketch che in questa puntata vi proponiamo per realizzare tale funzione è cortissimo, come si può vedere dal

Tabella 1 *- Classe REMOTEC funzioni di ricezione.*

boolean setScannerMode();	Per passare alla modalità di ricezione.
void setTransmitMode();	Per tornare alla modalità trasmissione.
int scanImpulses();	Riceve gli impulsi trasmessi dal radiocomando, determina l'impulso base e prova a decodificarli.

Listato 1, e permette la scansione dei radiocomandi.

TRASMISSIONE E RICEZIONE DEI PACCHETTI

La trasmissione e la ricezione di pacchetti normali richiede l'impostazione di alcune configurazioni; ma mentre per la trasmissione i passi sono limitati, per la ricezione le cose sono un po' più complicate.
Come abbiamo visto nella precedente puntata, l'SX1278 si preoccupa di aggiungere ai dati (payload) il preambolo, un indirizzo di rete ed un CRC, seguendo le indicazioni fornite dai registri di configurazione. È però opportuno sottolineare che in modalità non LoRa la coda FIFO è limitata a 64 byte, perciò se si vogliono spedire pacchetti più lunghi bisogna alimentare continuamente la coda. Lo stesso vale, al contrario, per la ricezione.
Per questo motivo sono stati predisposti degli interrupt (e flag) collegati alla coda FIFO: per esempio coda vuota, coda piena, coda piena oltre un certo livello (registro 0x35).
Per la ricezione, l'integrato SX1278 deve essere posto nello stato FSRX (stato 4) e, dopo qualche centinaio di microsecondi, in modalità RX (stato 5). A questo punto si tratta di verificare tramite interrupt o flag l'arrivo dei dati. Per esempio si può verificare il flag "PreambleDetect" o meglio ancora il flag "SyncAddressMatch" che segnala che il pacchetto è indirizzato proprio al ricevente.
Poi si può scaricare in una volta sola il pacchetto se questo è minore di 64 byte verificando il flag "PayloadReady"; oppure si può scaricare la FIFO fino a che il precedente flag non segnala il completamento.
A questo punto l'SX1278 può essere posto in standby oppure si può ricominciare la procedura in attesa di altri pacchetti.
In realtà la ricezione può essere automatizzata mediante l'utilizzo di un registro di condizioni di "AutoRestartRxMode".
Infine è possibile anche abilitare un "Sequencer" ovvero un vero programma di auto-gestione della trasmissione-ricezione. Questo sofisticato "automa a stati finiti" permette di automatizzare l'intero processo di comunicazione. Lasciamo a voi la lettura del data-sheet dell'SX1278 per l'approfondimento di questa complessa modalità.

MODALITÀ LORA

Con la modalità a pacchetto è possibile far comunicare le schede con SX1278 in modo sofisticato, affidabile e con pochissimo intervento di Arduino, sia con modulazione OOK che, meglio ancora, con modulazione FSK. Però, se non sono richieste buone velocità, ovvero alti bps, è molto più conveniente passare alla modalità principale dello SX1278, ovvero la modalità LoRa, che aumenta di molto la portata e l'affidabilità.
Come detto nella prima puntata, la modalità LoRa utilizza un protocollo di modulazione proprietario che è basato su una funzione di dispersione di ogni bit su più elementi di modulazione e su un ampio spettro centrato sulla frequenza della portante. In pratica ogni byte (o simbolo di informazione) è rappresentato da più elementi di modulazione (chip), e si parla allora di "Spreading Factor" che è il numero di chip utilizzati per ogni simbolo. Maggiore è lo "Spreading Factor" e migliore è la ricezione; ovvero migliora il rapporto segnale/rumore (SNR). Lo "Spreading Factor" può variare da 64 a 4.096. Il valore predefinito è 128 e può essere modificato usando il registro 0x1E (ricordiamo che i registri hanno copie differenti

Listato 1

```
/*************************************************************************/
/*
* Sketch per analizzare la trasmissione dei radiocomandi Avidsen o Velleman
* Determina la lunghezza dell'impuso base e prova a decodificare il comando.
*
*/

#include <SPI.h>
#include "REMOTEC.h"

REMOTEC RC;

void setup()
{
  Serial.begin(9600);
  RC.begin();
  if (RC.setScannerMode()) Serial.println("Receiving...");
  else Serial.println("Problem! No ready to receive");
}

void loop()
{
  RC.scanImpulses();
}
```

SpreadingFactor (RegModulationCfg)	Spreading Factor (Chips / symbol)	LoRa Demodulator SNR
6	64	-5 dB
7	128	-7.5 dB
8	256	-10 dB
9	512	-12.5 dB
10	1024	-15 dB
11	2048	-17.5 dB
12	4096	-20 dB

Tabella 2 *- Spreading factor e sensibilità.*

CodingRate (RegTxCfg1)	Cyclic Coding Rate	Overhead Ratio
1	4/5	1.25
2	4/6	1.5
3	4/7	1.75
4	4/8	2

Tabella 3 *- Bit di correzione ed overhead conseguente.*

per la modalità LoRa/OOK-FSK anche se condividono lo stesso indirizzo).
Lo "Spreading Factor" è codificato nel registro 0x1E con i numeri da 6 a 12 come si può vedere nella **Tabella 2**. Il valore predefinito è 128.
Inoltre per aumentare la robustezza del protocollo viene aggiunto un "Cyclic Error Correction", vale a dire alcuni bit aggiuntivi per correggere eventuali errori; ciò aumenta ulteriormente la lunghezza dell'informazione spedita. Il numero di questi bit di ridondanza può essere modificato nel registro 0x1D, che per default è regolato su 5 bit su 4 (codice 1); allo scopo riferitevi alla **Tabella 3** (il valore predefinito è 4/5).
A questo punto si può anche modificare lo spettro di emissione, che per impostazione predefinita è di 125kHz. Questo dato è definito anch'esso nel registro 0x1D e può avere i valori da 7,8kHz a 500kHz (codici da 0 a 9). Aumentando lo spettro aumenta anche il rate di emissione, ma diminuisce l'intensità del segnale e soprattutto bisogna tener conto delle normative di legge riguardo alle regole di emissioni radio.
In ogni caso con i valori predefiniti per lo "Spreading Factor", il "Cycling Coding Rate" e la banda di emissione si ha un rate di circa 8 kbps nominali, mentre con uno spread di 4.096 (codice 12) e una ridondanza 4/8 (codice 4) si hanno circa 150 bps. Per il calcolo basta considerare un chip di modulazione per ogni Hz di banda. Per il "Symbol Rate" basta dividere per 8: in pratica si va da un migliaio di byte al secondo a qualche decina di byte al secondo , senza voler contare casi estremi.

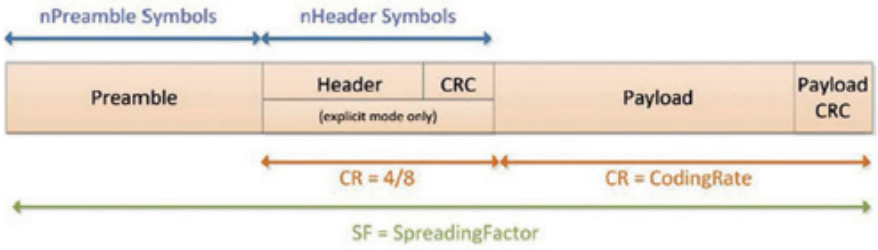


Fig. 3 *- Pacchetto in modalità LoRa.*

PACCHETTO LORA

Anche la modulazione LoRa comunica in modalità a pacchetto. Il pacchetto standard è detto "con header esplicito" ed ha lunghezza variabile. È formato dalle seguenti parti (**Fig. 3**).

1. Un preambolo di sincronizzazione di lunghezza modificabile e formato con un simbolo (byte) predefinito e proprietario; la lunghezza predefinita è 12 simboli.
2. Un header con il suo CRC; l'header contiene informazioni sul payload come la sua lunghezza, il "code rate" adottato nel payload e la presenza o meno del CRC alla fine del payload. L'header stesso, però, è sempre trasmesso con il massimo della ridondanza (code-rate=4/8) ed ha un suo CRC.
3. Payload di lunghezza variabile (255 byte massimo).
4. Eventuale CRC riferito al payload (abilitato tramite il bit 2 del registro 0x1E).

Esiste anche una modalità "implicita" in cui l'header è eliminato, per ridurre la lunghezza. In questo caso la lunghezza del payload deve essere fissata insieme alle sue altre caratteristiche e deve corrispondere tra il trasmettitore ed il ricevitore. Questa modalità è definita dal bit 0 del registro 0x10.
Nel caso di trasmissioni lunghe e lente è possibile abilitare un flag che costringe l'SX1278 ad una maggiore stabilità in frequenza. Questo flag "LowDataRateOptimize" corrisponde al

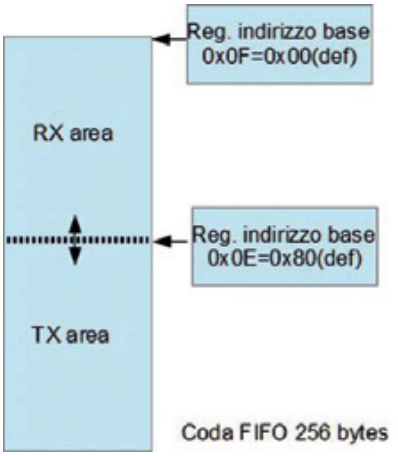


Fig. 4 *- Suddivisione del buffer FIFO.*

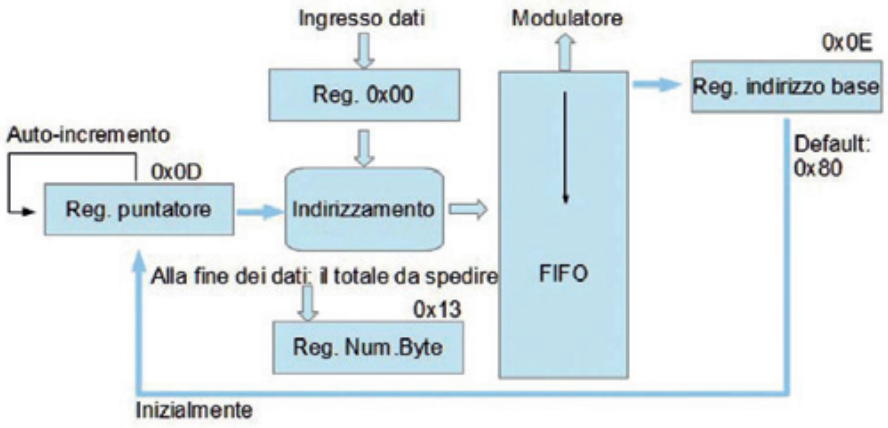


Fig. 5 *- Caricamento dei dati da trasmettere.*

bit 3 del registro 0x26, ma, nel caso, deve essere configurato sia per la funzione trasmittente che per la ricevente.
Come vedete, questa volta, non è presente un indirizzo di rete controllato automaticamente dall'SX1278; ne consegue che un eventuale indirizzamento può essere solo a carico del payload e verificato da Arduino.

OPZIONE SALTO DI FREQUENZA

Nel caso non si voglia impegnare per lungo tempo un canale della banda di frequenze portanti disponibili, è possibile ricorrere al salto di frequenza "Frequency Hopping Spread Spectrum" (FHSS). Questa modalità potrebbe essere necessaria nel caso di pacchetti lenti e lunghi in aree dove la legislazione in materia è più vincolante (per esempio in USA si tratta della banda 902÷928 MHz), quindi nel nostro caso è solo facoltativa.
Adottando questa modalità (configurando il registro 0x24 con un valore maggiore di zero), è richiesto un maggiore intervento di Arduino; infatti si tratta di stabilire una tabella delle frequenze comuni al trasmettitore ed al ricevitore. A questo punto, partendo dal canale di indice 0, dopo il tempo stabilito nel registro 0x24, viene generato un interrupt ed Arduino ha il compito di cambiare la frequenza sul canale 1 … e così via. La libreria non prende in considerazione questa modalità, ossia la frequency hopping.

LA CODA FIFO IN MODALITÀ LORA

Questa volta la coda FIFO è di 256 byte, ma, soprattutto ha una gestione completamente adattabile, nel senso che è composta da due settori separati ma variabili: uno per i pacchetti da trasmettere ed uno per quelli da ricevere. Per cui è possibile utilizzarla contemporaneamente per la trasmissione e la ricezione, contrariamente a quanto succedeva per la coda FIFO in modalità OOK/FSK.
Inoltre le grandezze dei due settori sono configurabili e si può aumentare l'area destinata alla ricezione a discapito di quella per la trasmissione o viceversa. Infatti la coda, questa volta, è gestita completamente con la tecnica dei puntatori. Un puntatore (registro 0x0F) punta alla base del settore dati ricevuti ed uno (registro 0x0E) alla base dei dati pronti per essere trasmessi. Un puntatore di accesso (registro 0x0D) serve ad indirizzare i dati da scrivere o da leggere, che sono, comunque acceduti tramite la porta della FIFO (registro 0x00).
Come configurazione originaria, la FIFO è suddivisa equamente tra area RX e area TX, per cui l'area RX parte all'indirizzo 0x00, mentre quella TX all'indirizzo 0x80. La **Fig. 4** chiarisce la com-

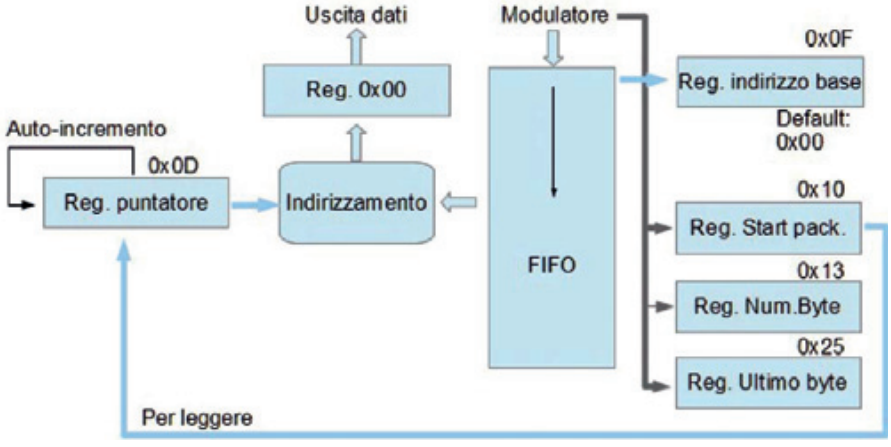


Fig. 6 *- Lettura dei dati ricevuti.*

Tabella 4 - *Flags (bit del registro 0x12).*

7	RxTimeout	Timeout nell'attesa di ricezione (vedi più avanti per l'impostazione del valore)
6	RxDone	È stata completata la ricezione di un pacchetto
5	PayloadCrcError	Errore sulla ricezione del payload o payload non affidabile
4	ValidHeader	L'header del pacchetto risulta valido (si può procedere)
3	TxDone	Il pacchetto è stato trasmesso
2	CadDone	La modalità CAD è stata attivata SX1278 è in ascolto di attività (preamboli)
1	FhssChangeChannel	È ora di cambiare canale (solo con il salto di frequenza abilitato)
0	CadDetected	È stata rilevato l'arrivo di un preambolo.

posizione della FIFO. I dati da trasmettere (il payload) vengono inseriti scrivendo sulla porta della FIFO (registro 0x00) dopo aver caricato sul registro puntatore il valore dell'indirizzo di partenza dell'area TX e inserendo alla fine il totale nel registro 0x13 (**Fig. 5**). I dati ricevuti, invece, vengono prelevati caricando sul registro puntatore il valore contenuto nel registro di partenza dell'ultimo pacchetto ricevuto. Poiché il pacchetto ha lunghezza variabile, un ulteriore registro (0x13) contiene il numero dei byte arrivati e quindi da scaricare.

INTERRUPT E TIMEOUT

Gli interrupt/flag si trovano in un registro (0x12) differente da quello usato nella modalità OOK/FSK e corrispondono alle segnalazioni riepilogate nella **Tabella 4**.
A ognuno di questi flag corrisponde un interrupt rilevabile su un determinato pin DIO secondo una configurazione impostabile tramite i registri di "mapping" 0x40 e 0x41 come riportato nella prima parte dell'articolo.
Inoltre gli interrupt possono essere disabilitati utilizzando il registro "maschera" 0x11. I flag/interrupt possono essere "resettati" manualmente, ma alcuni lo sono in modo automatico.
Esiste un timeout per l'attesa di ricezione impostabile mediante il registro 0x1F e i primi due bit del registro 0x1E per un valore massimo di 1023. Questo timeout, però, non è espresso in unità di tempo ma in unità di simboli (byte), quindi corrisponde al tempo necessario per trasmettere n simboli, il che rende il timeout variabile in funzione delle caratteristiche della comunicazione.
In pratica in termini di unità di tempo vale la formula di **Fig. 9**.

OPERATIVITÀ

L'attività dello SX1278 è scandita dall'aggiornamento del registro del Modo Operativo (registro 0x01). Né più né meno che in modalità OOK/FSK. C'è, però qualche modifica riguardante la ricezione ed è stato aggiunto un ulteriore stato. Per comodità riassumiamo tutti gli stati operativi nella **Tabella 5**.

$$Timeout = \frac{Valore\ registro}{SymbolRate} \quad \text{ovvero} \quad Valore\ registro = SymbolRate * Timeout\ desiderato$$

Fig. 9

Codice	Modo	Descrizione
0	Sleep	Modalità a bassissimo consumo (<<1uA). Registri accessibili ma non FIFO. Solo modo per definire la modalità di base LoRa o FSK/OOK (default)
1	Standby	Stato base durante l'operatività (consumo circa 1,5 uA)
2	FSTX	Pronto a trasmettere (circa 6mA)
3	FSRX	Pronto a ricevere (circa 6mA)
4	TX	Dà il via alla trasmissione ed alla fine si riporta automaticamente in standby (20-:-120mA)
5	RXcontinuous	Ricezione continua dei pacchetti (vedere più avanti; assorbimento di circa 12 mA)
6	RXsingle	Ricezione del singolo pacchetto ed alla fine si riporta automaticamente in standby (12mA)
7	CAD	In ascolto per rilevare l'arrivo di un preambolo.

Tabella 5 - *Modi Operativi in modalità LoRa (registro 0x01).*

TRASMISSIONE

Dopo aver definito le caratteristiche dei pacchetti e della comunicazione ("una tantum"), i passi necessari per trasmettere un pacchetto a partire dallo stato standby sono:

1. inizializzare il puntatore FIFO (0x0D) alla base dell'area TX (il valore predefinito è 0x80);
2. inserire i byte del payload tramite la porta FIFO (reg. 0x00); il puntatore si incrementerà da sè;
3. alla fine, inserire il numero dei byte da spedire (lunghezza del payload) nel registro 0x22;
4. passare allo stato FSTX ed attendere un centinaio di microsecondi;
5. dare il via alla trasmissione (stato TX);
6. dopo l'ultimo byte trasmesso l'SX1278 si porterà automaticamente allo stato di Standby;

Si può verificare l'avvenuta trasmissione mediante il flag/interrupt TxDone. Notate che il passaggio allo stato Sleep cancella il contenuto della memoria FIFO, mentre rimanendo in Standby, la FIFO può essere riutilizzata per rimandare lo stesso messaggio: basta non aggiornare il puntatore.

RICEZIONE

La ricezione può avvenire per singolo pacchetto o in modo continuo pacchetto dopo pacchetto, nel senso che è possibile dare il via alla ricezione continua verificando l'arrivo di un pacchetto e scaricandolo senza che lo SX1278 cambi stato operativo. Per quanto riguarda la ricezione singola si compie con questa sequenza:

1. inizializzare il puntatore FIFO

(0x0D) alla base dell'area RX (il valore predefinito è 0x00);
2. passare allo stato FSRX e attendere un centinaio di microsecondi;
3. dare il via alla ricezione (stato RXsingle); la ricezione termina automaticamente riportando lo SX1278 allo stato Standby quando si è superato il Timeout configurato (flag/interrupt Timeout) oppure è arrivato un pacchetto (flag/interrupt RxDone);
4. nel caso sia arrivato un pacchetto, questo verrà estratto dalla FIFO leggendo la porta 0x00 per tante volte quanto è il valore del registro NumeroByte Ricevuti (0x13).

Quanto alla ricezione continua, la sequenza di esecuzione è:
1. inizializzare il puntatore FIFO (0x0D) alla base dell'area RX (il valore predefinito è 0x00);
2. passare allo stato FSRX e attendere un centinaio di microsecondi;
3. dare il via alla ricezione continua (stato Rxcontinuous); l'attività può essere bloccata solo manualmente cambiando stato;
4. se è arrivato un pacchetto il flag RxDone viene aggiornato e la ricezione continua;
5. se il flag RxDone è attivato si carica il registro puntatore con il valore del registro Start Pack (RxCurrentAddress 0x10) che punta all'inizio dell'ultimo pacchetto completo arrivato;
6. si legge la coda FIFO per un numero di volte corrispondente alla lunghezza dell'ultimo pacchetto arrivato (reg. 0x13);
7. si azzera il flag RxDone per segnalare un nuovo pacchetto;
8. se si vuole interrompere la ricezione continua basta portare l'SX1278 nello stato Standby, altrimenti si lascia attivo e la ricezione procede.

Mente geniale, fascino cinematografico...

Se pensate che la tecnica del salto di frequenza con algoritmo "randomico" sincronizzato sia opera di qualche camice bianco con molte penne nel taschino e l'aspetto trascurato, vi sbagliate di grosso. Il brevetto, ormai scaduto, appartiene alla bellissima attrice hollywoodiana Hedy Lamarr. Sì, Esatto! La tecnica usata anche dalla moderna telefonia mobile, nonché dai militari è stata inventata da un'attrice.
Hedy Lamarr (nome d'arte) di origine austriaca e di discendenza ebrea, si era iscritta alla facoltà di ingegneria negli anni trenta, ma fu notata da un registra che la convinse a intraprendere la carriera di attrice di teatro e cinema. Cosicché abbandonò l'università. Si sposò giovanissima con un imprenditore del settore armamenti. In questo ambiente recepì la problematica della sicurezza e dell'interferenza sulle trasmissioni radio militari soprattutto nell'uso con i radiocomandi. Quando emigrò negli USA, sia perché Hollywood la reclamò, sia per motivi politico-razziali, volle partecipare allo sforzo bellico come molte altre sue colleghe. Ma invece di limitarsi alle campagne per la ricerca di finanziamenti, presentò alla marina militare un progetto di trasmissione radio innovativo e non facilmente intercettabile. Purtroppo il progetto venne scartato, perché la tecnologia radio di allora (a valvole) non era facilmente adattabile al sistema, ma probabilmente anche per una certa sufficienza degli alti comandi. Il sistema, ed Hedy Lamarr, si ripresero la rivincita (solo morale) negli anni sessanta , quando la marina militare lo cominciò ad utilizzare estesamente. Senza contare che è anche alla base della telefonia mobile.
L'idea le venne insieme ad un suo amico compositore George Antheil; e , guarda caso, il brevetto riportava l'algoritmo per il salto di frequenza basato su 88 portanti: tante quanti sono i tasti di un pianoforte.

CHANNEL ACTIVITY DETECTION (CAD)

Poichè in modalità LoRa il segnale si confonde con il rumore, se si volesse monitorare la presenza di comunicazione, non sarebbe accettabile utilizzare il valore RSSI; per ovviare a ciò è stata predisposta una modalità per la quale l'SX1278 si pone in ascolto soltanto del preambolo. In questo modo si può verificare l'occupazione del canale o eventualmente passare allo stato di ricezione. In questo stato, la ricezione del preambolo viene

Listato 2 - Sketch trasmittente per test con eco

```
#include <LORA.h>
#include <SPI.h>

#define psound 9          // pin per  buzzer
#define pingo 8             // pin per push-button
#define pinf  7               // pin per il led di segnalazione

#define MESS "Simple echo test"   // messaggio da spedire e ricevere come eco

#define RXTIMEOUT 500    //decine di millisecondi (500=500*10=5 secondi)

LORA LR;                    // Istanza della classe LORA

#define inplen 64         // lunghezza del buffer di spedizione
#define reclen 64         // lunghezza del buffer di ricezione

char inpbuff[inplen];     //  buffer di spedizione
char recbuff[reclen];     // buffer di ricezione

char data[64];               // buffer per i valori RSSI and SNR
#define format "|Rssi: %d RssiPk: %d SnrPk %d|  "

int SF=9;  //Codice Spreading factor  (se si vuole cambiare per test)
int BW=6;  //Codice Bandwidth  (se si vuole cambiare per test)

int PWR=2;                       //potenza in trasmissione (codice)

boolean SHIELD=true;

void setup()
{
  pinMode(pingo,INPUT_PULLUP);   // Pull-up per il push-button
  digitalWrite(pinf,0);                            // inizializza led
  pinMode(pinf,OUTPUT);
  pinMode(psound,OUTPUT);            //inizializza cicalino
  Serial.begin(9600);
  if (!LR.begin())                                 //se la scheda non c'è esce
    {Serial.println("No LoRa shield detected! Stop!");SHIELD=false;return;}
  Serial.println("LoRa echo transmitter.");
  SX.setPower(PWR);
//  LR.setConfig(SF,BW,4);   // se si vuole cambiare per test( def: 9,6,4)
  showConfig();                                    //stampa la configurazione
  strlcpy(inpbuff,MESS,inplen);          //carica il messaggio nel buffer
  Serial.print("Close pin ");
  Serial.print(pingo);Serial.println(" to ground to send message");
}

void loop()
{
  delay(200);
  if (!SHIELD) return;
  if (getInput()) {sendBuff();getReplay();}
        //se push-button premuto spedisce e riceve eco
}

boolean getInput()
{
  if (digitalRead(pingo)>0) return false;     //test push-button
  return true;
}

void sendBuff()
{
  sound(300,1);  // avvisa che sta spedendo
  blinkpinf(100,10);
  digitalWrite(pinf,LOW);

  Serial.print("> ");Serial.println(inpbuff);
  int f=LR.sendMess(inpbuff);                 // spedisce il messaggio
  if (f<0) Serial.println("Error in transmission!");

  SX.setState(STDBY);
}
```

(continua)

evidenziata dal flag/interrupt CadDetected e lo SX1278 ritorna allo stato Standby. La rilevazione del preambolo è fatta molto velocemente (qualche byte) mediante algoritmi di correlazione e consuma, perciò, poco.

LA LIBRERIA IN MODALITÀ LORA

La libreria per la modalità LoRa è definita dalla classe LORA che è composta da poche istruzioni principali, che elenchiamo nella **Tabella 6**. Sono, poi, disponibili poche altre funzioni per modificare i parametri della trasmissione: spreading factor, larghezza di banda ecc.
La configurazione di base ha portante con frequenza 434 MHz, spreading factor codice 9, larghezza di banda codice 6 e ridondanza codice 4. Per cambiare altre caratteristiche si possono sempre utilizzare anche le funzioni della classe SX che permettono di agire su tutti i registri. La classe SX è importata automaticamente dalla classe LORA.Vediamo ora un esempio di comunicazione LoRa che è composto da due skhetch: uno che trasmette una breve frase ed uno che la riceve e la ritrasmette come eco. Questi sketch sono stati verificati in città ed hanno mostrato un range operativo di circa 200 m in presenza di ostacoli (mura e palazzi) mentre in campo aperto hanno consentito un range di circa 1 km.

SKETCH TRASMITTENTE

Lo sketch (**Listato 2**) verifica ed inizializza la scheda, quindi si pone in attesa di un pulsante che chiude a massa il pin 8. Quando il pulsante viene premuto, viene inviato il messaggio preordinato. Quindi lo sketch si pone in ricezione per un certo tempo, aspettando l'eco. Quando questo arriva, lo confronta con

il messaggio di partenza e se corrisponde avvisa della corretta ricezione e stampa sulla console una stima dei valori RSSI e SNR. Le segnalazioni sono fatte su console, ma anche con un LED sul pin 7 ed un cicalino sul pin 9.

SKETCH DI RICEZIONE ED ECO

Dopo la fase di inizializzazione, lo sketch si mette in modalità ricezione ed aspetta messaggi; appena ne riceve uno, esce dalla modalità ricezione, visualizza i dati di stima RSSI e SNR e lo rispedisce, quindi ritorna in modalità ricezione (**Listato 3**).

CONCLUSIONI

Come abbiamo potuto verificare, la scheda è molto flessibile ed adattabile a diversi tipi di trasmissioni. Anche se la modalità più innovativa ed efficiente è quella LoRa. Con un paio di schede si possono coprire distanze notevoli: 200m al chiuso e tra edifici e circa 1 km in campo aperto. Già con una potenza di 10mW si raggiungono una ottantina di metri tra gli edifici. Ma con un software di identificazione di indirizzo è possibile comunicare con molte schede e ricevere valori dei sensori o attivare azionatori. Se poi pensassimo di utilizzare la scheda con RandA (ovvero con l'accoppiata Raspberry Pi e Arduino), potremmo avere un instradamento dei dati su Internet. La libreria "LoRa.zip" è scaricabile dal sito e comprende anche la classe per il controllo remoto Avidsen o Velleman. La libreria fornisce le funzioni semplificate per utilizzare al meglio la scheda. Lasciamo a voi la possibilità di manipolare in modo più raffinato l'SX1278 mediante la gestione diretta dei suoi numerosi registri.
Le funzioni di libreria illustrate

Listato 2 (segue)

```
void getReplay()
{
  LR.receiveMessMode();   // si mette in ricezione
  boolean OK=false;
  int i;
  for (i=0;i<RXTIMEOUT;i++)
                          // verifica se arriva una risposta entro RXTIMEOUT
    {if (LR.dataRead(recbuff,reclen)>0) {OK=true;break;} delay(10);}
  if (!OK) {Serial.println("No replay!");blinkpinf(50,20);sound(300,3);return;}
        // se si stampa valori RSSI e verifica correttezza
  snprintf(data,63,format,SX.getLoraRssi(),SX.lastLoraPacketRssi(),
          SX.lastLoraPacketSnr());
  Serial.println(data);
  Serial.print("< ");Serial.println(recbuff);
  int inc=strlen(inpbuff);
 if (strncmp(recbuff,inpbuff,inc)==0) {digitalWrite(pinf,HIGH);sound(1000,1);}
  else {blinkpinf(500,4);sound(500,2);}
}

void blinkpinf(int time,int n)
{
  int i;
  byte p=1;
  for (i=0;i<n;i++) {digitalWrite(pinf,p);delay(time); p=p^1;}
  digitalWrite(pinf,0);
}

void sound(int time,int n)
{
//  return; //uncomment if you like sound
  int i;
  for (i=0;i<n;i++)
   {digitalWrite(psound,1);delay(time);digitalWrite(psound,0);delay(100);}
  digitalWrite(psound,0);
}

void showConfig()
{
  Serial.print("Replay timeout (millisec.): ");Serial.println(RXTIMEOUT*10);
  Serial.print("Frequence: ");Serial.println(SX.readFreq());
  Serial.print("Transmit power (mW): ");Serial.println(SX.getPower(3));
  Serial.print("Preamble bytes: ");
  Serial.print(SX.getLoraPreambleLen());Serial.println("+4");
  snprintf(data,63,"SpFactor: %d BandW: %d Cr: %d",
                        SX.getLoraSprFactor(),SX.getLoraBw(),SX.getLoraCr());
  Serial.println(data);
  Serial.print("Rate (byte/sec): ");Serial.println(SX.getSRate());
}
```

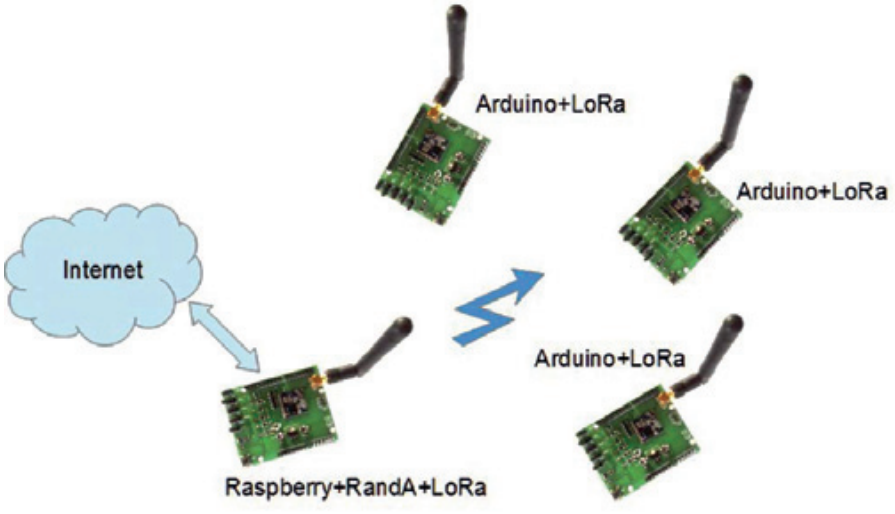


***Fig. 10** - Rete di sensori/attuatori.*

Nome	Formato	Descrizione
begin setModeLora	boolean begin() void setModeLora()	Inizializza la scheda direttamente in modalità LoRa. Se la scheda non è presente ritorna falso. La seconda funzione invece cambia in modalità LoRa se era stata già inizializzata in altra modalità.
sendMess	int sendMess(char mess[]) int sendMess(byte mess[],byte mlen)	Spedisce un messaggio (in formato caratteri o byte). mlen è la lunghezza del messaggio (massimo 255). Ritorna 0 o -1 in caso di errore.
receiveMessMode dataRead	void receiveMessMode() int dataRead(byte buff[],byte blen) int dataRead(char mess[],byte maxlen)	Si mette in ricezione continua. Se viene inserita in un ciclo la funzione dataRead(), questa restituisce il buffer riempito oppure 0 se non è arrivato alcun pacchetto.

***Tabella 6** - Funzioni della classe LORA.*

Listato 3 - Sketch ricevente per test con eco

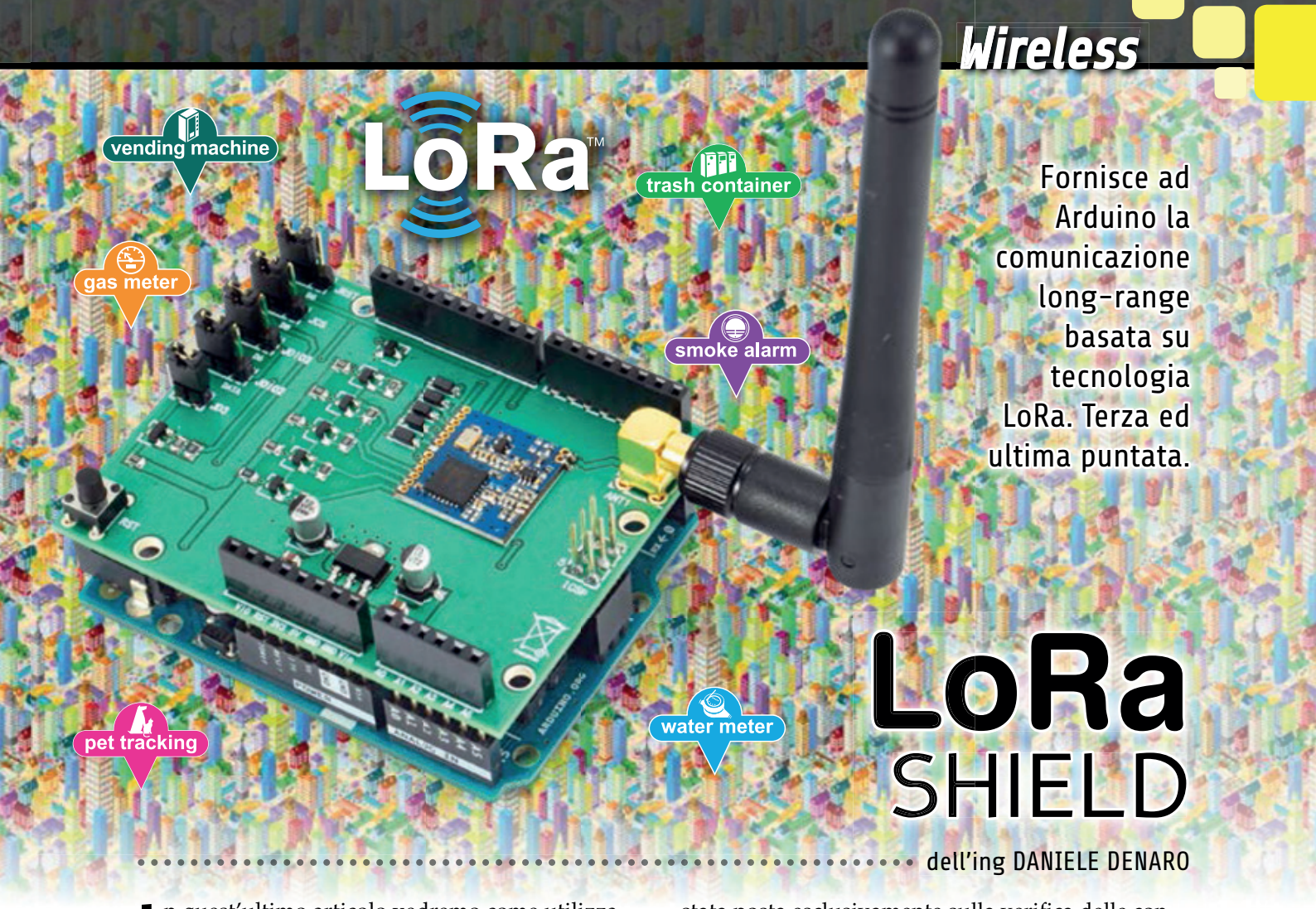
```
#include <LORA.h>
#include <SPI.h>
LORA LR;                    // Istanza della classe LORA
#define reclen 64         // lunghezza dei buffer
char recbuff[reclen];     //  buffer di spedizione
char sendbuff[reclen];    // buffer di ricezione
char data[64];              // buffer per i valori RSSI and SNR
#define format "|Rssi: %d RssiPk: %d SnrPk %d|  "
int SF=9;  //Codice Spreading factor  (se si vuole cambiare per test)
int BW=6;  //Codice Bandwidth  (se si vuole cambiare per test)
int PWR=2;                        //potenza in trasmissione (codice)
boolean SHIELD=true;
void setup()
{
  Serial.begin(9600);
  if (!LR.begin())
    {Serial.println("No LoRa shield detected! It can't perform echo!");SHIELD=false;return;}
  Serial.println("LoRa echo receiver.");
  SX.setPower(PWR);
//  LR.setConfig(SF,BW,4);     // se si vuole cambiare per test( def: 9,6,4)
  showConfig();
  Serial.println("Waiting for message...");
  LR.receiveMessMode();      // si mette in modalità ricezione
}

void loop()
{
  if (!SHIELD) {delay(200);return;}
  if (getMess()) {sendEcho();LR.receiveMessMode();} //se c'è mess. Spedisce eco
         // e si rimette in ricezione
}

boolean getMess()
{
  if (LR.dataRead(recbuff,reclen)<=0) {delay(10);return false;}
  Serial.print("< ");Serial.println(recbuff);
  snprintf(data,63,format,SX.getLoraRssi(),SX.lastLoraPacketRssi(),
         SX.lastLoraPacketSnr());
  Serial.println(data);
  return true;
}

void sendEcho()
{
  SX.setState(STDBY);                    // esce dalla modalità ricezione
  delay(300);
  strncpy(sendbuff,recbuff,reclen);
         //copia ciò che ha ricevuto nel buffer di spedizione
  int bufflen=strlen(sendbuff);
  Serial.print("> ");Serial.println(sendbuff);  //spedisce
  if (LR.sendMess(sendbuff,bufflen)<0)
       Serial.println("Sending error!");
}

void showConfig()
{
  Serial.print("Frequence: ");Serial.println(SX.readFreq());
  Serial.print("Transmit power (mW): ");Serial.println(SX.getPower(2));
  Serial.print("Preamble bytes: );Serial.print(SX.getLoraPreambleLen());Serial.println("+4");
  snprintf(data,63,"SpFactor: %d BandW: %d Cr: %d",SX.getLoraSprFactor(),
         SX.getLoraBw(),SX.getLoraCr());
  Serial.println(data);
  Serial.print("Rate (byte/sec): ");Serial.println(SX.getSRate());
}
```

in questo articolo, permettono una trasmissione punto-punto. Ma, se vogliamo realizzare una rete con diversi nodi (come in **Fig. 10**), dobbiamo inserire degli indirizzi e gestirli. Per questo, nel prossimo e conclusivo articolo, affronteremo la problematica dell'indirizzamento e della crittografia. Infatti, a causa del range operativo non limitato a pochi metri, è necessario proteggere le trasmissioni da intercettazioni e sovrapposizioni. Per questo motivo la libreria è stata modificata con l'introduzione di indirizzi e di funzioni di crittografia AES256 che però appesantiscono poco Arduino sia in termini di prestazioni che di occupazione di memoria. Vedremo, quindi, come, con semplici funzioni di libreria sarà possibile realizzare una rete efficiente e protetta.

per il MATERIALE

La shield Long Range (cod. FT1190M) comprensiva di modulo radio e tutti i componenti necessari al suo funzionamento, è disponibile a 39,50 Euro. Il modulo DRF1278F è disponibile anche separatamene a 19,00 Euro. Arduino UNO (cod. ARDUINOUNOREV3) viene venduto a 24,50 Euro. Tutti i prezzi si intendono IVA compresa.

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



Fornisce ad Arduino la comunicazione long-range basata su tecnologia LoRa. Terza ed ultima puntata.

LoRa SHIELD

dell'ing DANIELE DENARO

In quest'ultimo articolo vedremo come utilizzare la scheda proposta nel fascicolo n° 199 per realizzare una rete composta da nodi indirizzabili e sicuri. Come abbiamo visto nel precedente articolo, l'integrato SX1278 della Semtech, su cui è basato lo shield, permette una sofisticata ed efficiente comunicazione in tecnologia LoRa (Long Range transmission). Questa comunicazione è bidirezionale ed ha un consistente raggio di copertura: un centinaio di metri all'interno di edifici o circa un chilometro in campo aperto. La scheda opera nella banda dei 433 MHz ed è a basso costo; è quindi orientata ad applicazioni che vanno dalla "home automation" alla sicurezza, fino al monitoraggio e all'automazione in agricoltura.

Poiché la comunicazione è bidirezionale, la board si presta ad architetture di rete a "stella" o anche a "reticolo": nel primo caso c'è un nodo privilegiato cui gli altri fanno riferimento, comunicando esclusivamente con esso, mentre nel secondo ogni nodo può decidere di comunicare con qualunque altro.

Nella puntata precedente abbiamo visto come programmare la scheda per una comunicazione punto-punto; in tale applicazione l'attenzione è stata posta esclusivamente sulla verifica delle condizioni di comunicazione e della portata raggiungibile. Infatti l'applicazione era basata su un trasmettitore che lanciava un messaggio ed un ricevitore che ne faceva l'eco; non era presente alcuna forma di indirizzamento delle comunicazioni e volendo utilizzare più sistemi insieme l'unica discriminante delle trasmissioni delle singole unità poteva essere la frequenza portante.

È evidente che per realizzare qualunque struttura di rete è necessario un sistema di indirizzamento delle singole comunicazioni. Ma non basta: a causa della portata, che non è limitata a pochi metri, la trasmissione è facilmente intercettabile e quindi necessita di un sistema di protezione.

La Semtech ed altri soggetti della cosiddetta "LoRa Alliance" hanno progettato un protocollo di rete chiamato LoRaWAN; è abbastanza complesso e in esso le periferiche con sensori e attuatori ("End-device") sono collegate a dei Gateway che instradano il traffico sulla rete Internet fino al server dedicato alla loro gestione. Il protocollo è complesso a causa delle molte variabili che sono gestite; prima fra tutte la gestione dell'alimentazione delle periferiche,

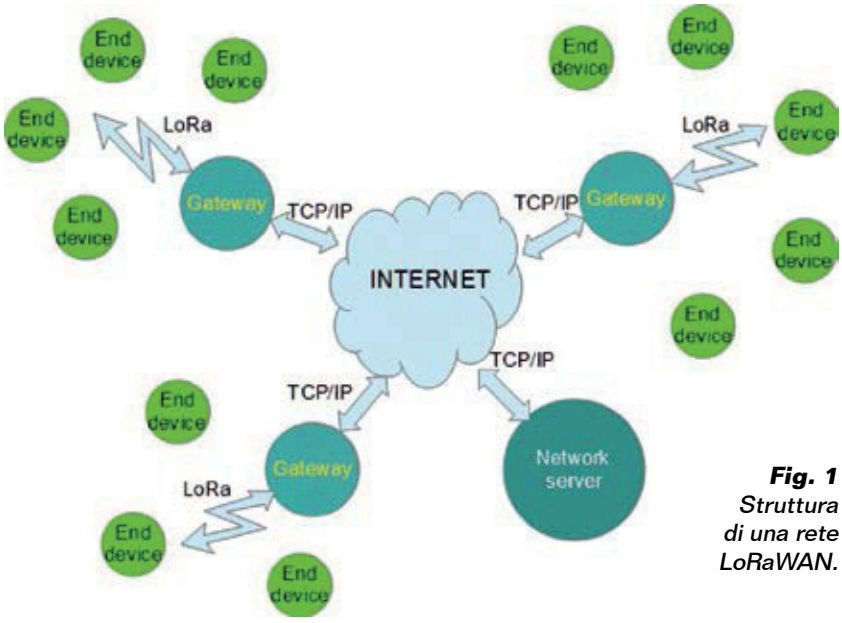


***Fig. 1** Struttura di una rete LoRaWAN.*

che si tende a ridurre al massimo. Inoltre sono gestiti in modo adattativo sia il salto di frequenza che il data-rate. Il protocollo prevede anche la crittografia AES con chiave di 128 bit e prevede tre diverse tipologie di "End-device": classe A, classe B e classe C. La classe A è quella di base e prevede l'iniziativa da parte della periferica che, dopo la trasmissione apre due piccole finestre temporali per la risposta da parte del server e poi si spegne. La classe B ha la possibilità di avere delle temporizzazioni di ascolto, mentre la C può stare sempre in ascolto, perché evidentemente non ha problemi di alimentazione. Le specifiche del protocollo LoRaWAN possono essere scaricate dal sito della Semtech.
Il protocollo LoRaWAN è troppo complesso per una struttura più limitata come può essere una piccola rete privata per utilizzo

Come **proteggere** le informazioni

La crittografia, ovvero l'attività di camuffare le informazioni verso occhi ed orecchie indiscrete, è antica praticamente come l'uomo. Senza contare i sistemi fisici di occultamento, come gli inchiostri simpatici di infantile memoria, già Giulio Cesare utilizzava regolarmente la crittografia a sostituzione di caratteri nei suo messaggi.
Si può dire che la crittografia nasce proprio con il metodo della sostituzione di ogni carattere con un altro secondo uno schema che prende il nome di "chiave". Non è il metodo ad essere segreto, ma la chiave, senza la quale non è possibile rendere intellegibile il testo. Ma la crittografia a sostituzione di carattere ha un fondamentale punto debole: lascia inalterata l'informazione elementare, cioè il carattere. Per cui, nonostante la notevole complicazione dello schema di spostamento, avvenuto nel corso dei secoli, l'analisi statistica della frequenza dei caratteri nel messaggio codificato, permette di ricostruire la chiave. Infatti, poiché ogni lingua ha una particolare distribuzione media dei singoli caratteri, se i messaggi sono molti o molto lunghi, è possibile associare il carattere vero al carattere di camuffamento.
Ma cosa succede se il metodo di sostituzione cambia continuamente secondo una meta-chiave? E' questo il metodo utilizzato dalla macchina a rotori Enigma, utilizzata dai tedeschi nella seconda guerra mondiale ed alla cui decriptazione ha lavorato il famoso matematico e precursore dell'informatica Alan Turing.
Questa volta la distribuzione dei caratteri nel testo cifrato è uniforme e non fornisce alcuna indicazione. È necessario ricorrere ad altri trucchi, come l'intuizione di parte del contenuto o la ricerca esaustiva ("forza bruta").
L'avvento dei computer, se da un lato ha aumentato a dismisura le esigenze di protezione -se non altro a causa dell'esplosione di comunicazione- d'altro canto ha fornito dei meccanismi formidabili di protezione: invece di trasformare i caratteri, vengono modificati i singoli bit. Già questo garantirebbe una notevole assenza di schema nel testo crittato, ma se poi ci aggiungiamo ripetute azioni non lineari con l'ausilio di chiavi particolarmente lunghe, il testo finale è praticamente non decodificabile in tempi umani ricorrendo al sistema della forza bruta. In pratica l'informazione di tutto il messaggio risulta sparpagliata come un rumore. Solo l'applicazione della chiave insieme al procedimento inverso può ricostruire il messaggio.
Questo tipo di crittografia è detta simmetrica, perché la stessa chiave è utilizzata per la codifica e la decodifica. Ne consegue che la chiave deve essere comunicata tra gli utenti in modo sicuro. Attualmente il migliore sistema a chiave simmetrica è il sistema AES con chiave di 256 bit. Si stima che con i computer attuali i tempi di decifrazione con la forza bruta siano di milioni di anni.
Ma come si fa se la chiave non può essere comunicata in modo sicuro? Questa, per esempio, è la comune realtà dell'e-commerce, dove non è pensabile che per acquistare un prodotto si debba chiedere "per lettera assicurata" una chiave da utilizzare. Per fortuna negli ultimi decenni è stata sviluppata la crittografia asimmetrica, anche detta a chiave pubblica. Questa è particolarmente impegnativa a livello computazionale e viene utilizzata soprattutto come canale sicuro per la comunicazione della chiave simmetrica utilizzata per l'effetti-

Fig. 2 *- Esempio di costruzione dell' indirizzo.*

Tabella 1 *- Indirizzamento.*

setNetAddress(767,64)	Definisce una rete di indirizzo 767 e con possibili 63 sub-indirizzi (0 non è contemplato) (ritornerebbe false se al posto di 767 ci fosse 3400)
sendNetMess(7,1,"Test",4)	Spedisce il messaggio "Test" al sub-indirizzo 7 (lui è 1) (l'indirizzo di rete è stato definito dalla precedente funzione)
receiveNetMess(1,7,buff,8)	Attende messaggio da 7 per lui (indirizzo 1). Se al posto di 7 ci fosse 0 , riceverebbe da tutti.

locale. Per questo motivo abbiamo pensato di completare la libreria LORA con l'implementazione di una struttura di indirizzi ed una protezione con crittografia AES.

LIBRERIA LORA: INDIRIZZAMENTO

Nel protocollo da noi definito, è stato inserito un indirizzo del destinatario ed uno del mittente. L'indirizzo è a 16 bit (word) che permettono 65.534 possibilità (lo 0 non è contemplato).
Per rendere più pratico il suo utilizzo, l'indirizzo è stato suddiviso in un indirizzo di rete ed un sub-indirizzo di "device"; però la suddivisione non è fissa e può essere decisa dall'utente, nel senso che la porzione dell'indirizzo riservata alla rete e di conseguenza quella delle periferiche, è variabile. In pratica si tratta di definire il massimo numero delle periferiche indirizzabili (che sia una potenza di 2) e un indirizzo

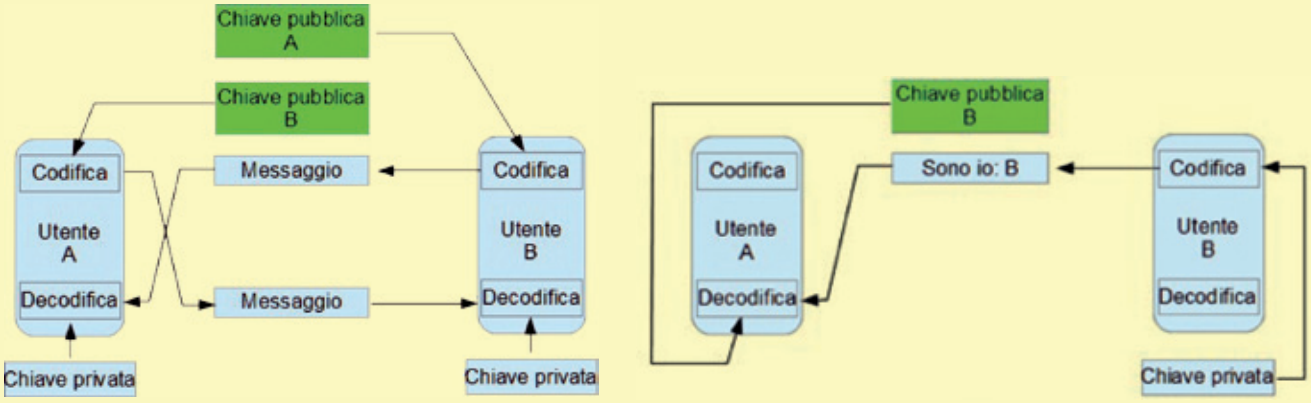


Fig. A *- Crittografia a chiave pubblica e firma elettronica*

vo trasferimento del messaggio. Oppure viene utilizzata per garantire l'originalità del mittente (firma digitale, **Fig. A**) o l'integrità del messaggio. Infatti un altro aspetto della sicurezza è sapere se il messaggio è stato spedito dal soggetto indicato o da qualcuno che si è sostituito a lui, oppure ha modificato il messaggio originale. Questo è il cosiddetto problema del "man in the middle".
La crittografia a chiave pubblica è piuttosto complessa ed utilizza in sostanza l'aritmetica dei numeri primi o delle funzioni "ellittiche" e soprattutto fa ampio uso dell'aritmetica "modulare" (resto della divisione o aritmetica dell'orologio); la sostanza può essere riassunta così:

- le chiavi sono due e unidirezionali (una codifica, l'altra decodifica);
- le due chiavi sono in qualche modo collegate, anche se non in modo decodificabile;
- le due chiavi sono intercambiabili (se la seconda codifica, la prima decodifica e viceversa).

C'è infine un ulteriore tipo di protezione: quello necessario per evitare che un soggetto esterno copi brutalmente il messaggio crittato e lo invii in secondo tempo senza esigenza di comprensione. Questo è un evento caratteristico dei radiocomandi unidirezionali che inviano sempre lo stesso messaggio per azionare un apparato: ad esempio per aprire le portiere di un auto. Poiché il messaggio è sempre uguale ed ottiene sempre lo stesso effetto, non c'è crittografia che tenga e basta ripeterlo così com'è. Per risolvere il problema è sufficiente rendere il messaggio sempre diverso sincronizzandolo con il ricevitore: è ciò che viene realizzato con il "rolling code". Un'alternativa più complessa è quella di rendere il radiocomando bidirezionale e fargli intraprendere un dialogo di "hand shaking".

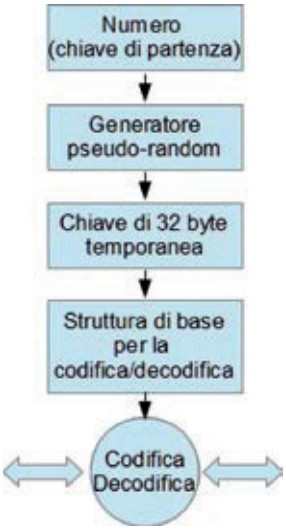


Fig. 3 *- Crittografia AES 256 per LORA.*

di rete che sia compreso nello spazio che rimane nella "word" di indirizzo effettivo. Se l'indirizzo di rete eccede, la funzione ritornerà "false". Ovviamente tutti i nodi faranno riferimento a questa struttura con lo stesso numero di rete e diverso indirizzo di nodo.
Nella **Tabella 1** è mostrato un esempio di utilizzo di funzioni di spedizione e ricevimento.

LIBRERIA LORA: CRITTOGRAFIA

Anche se l'applicazione che potreste avere in mente non ha veri vincoli di sicurezza, non è opportuno far viaggiare in chiaro pacchetti radio alle distanze raggiungibili dalle trasmissioni LoRa. Per questo motivo abbiamo pensato a una soluzione che fosse ragionevolmente sicura ma anche praticabile su un hardware limitato come Arduino.

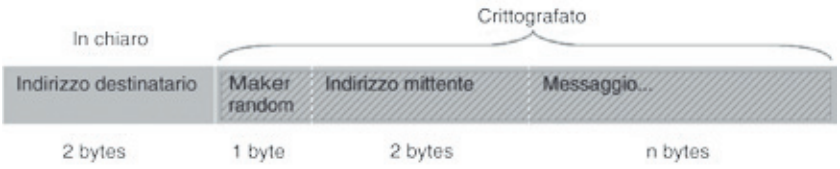


Fig. 4 *- Payload LoRa.*

Per fortuna in rete è possibile scaricare un software di crittografia AES adattato per Arduino.
Questo software, che utilizza la memoria flash di programma per alcune strutture di base, è stato da noi leggermente modificato nella gestione della chiave.
La crittografia AES utilizzata è su chiave di 256 bit, ma invece di fornire un array di 32 byte, la chiave è generata in maniera pseudo-random a partire da un valore-seme del generatore random di Arduino. Questa modalità riduce l'occupazione della RAM ad un solo numero (integer), ma ha la controindicazione di essere utilizzabile solo per Arduino. Infatti se si utilizzano librerie matematiche differenti, la produzione di chiavi pseudo-casuali a partire dallo stesso seme fornirebbe chiavi non coincidenti tra le varie schede della stessa rete.
La crittografia AES a 256 bit è considerata molto sicura, infatti l'algoritmo compie un certo numero di spostamenti e permutazioni con funzioni non lineari, producendo una sequenza di byte completamente mascherata e paragonabile a rumore puro. Solo l'utilizzo della chiave permette il procedimento inverso. Tuttavia la libreria utilizzata appesantisce pochissimo Arduino in termini sia di elaborazione che di occupazione di memoria.
Oltre alla crittografia AES, è stato aggiunto un byte al messaggio; tale byte ha un valore casuale ed ha lo scopo di rendere unici messaggi identici che si ripetono. Più che altro è stato pensato per consentire un'eventuale implementazione di una sicurezza aggiuntiva tipo rolling-code. Tenete presente che a causa del tipo di crittografia utilizzato, il valore di un byte si sparpaglia su tutto il messaggio codificato.
La chiave crittografica è definita direttamente nella funzione di attivazione della libreria LORA.
La definizione della chiave è:

	Chiave
begin(33333)	Inizializza la libreria con la chiave 33333. La funzione verifica la presenza della scheda e ritorna false se non è agibile.

PACCHETTO

Il pacchetto inviato e ricevuto in modalità LoRa (ovvero il payload) è quindi composto come si può vedere in **Fig. 4**.
La funzione di spedizione lo costruisce aggiungendo al messaggio i due indirizzi (previa ricostruzione in formato word) ed un byte di valore casuale. Quindi lo

Tabella 2 *- Funzioni LORA basilari.*

bool **begin**(unsigned int keyval);	Inizializza e definisce la chiave per la crittografia
bool **setNetAddress**(unsigned int netAdd, unsigned int devRange);	Definisce la struttura di rete
int **sendNetMess**(unsigned int devSubAdd, unsigned int sendSubAdd, byte *mess, int lmess);	Spedisce un messaggio
void **receiveMessMode**();	Si pone in continuous receiving mode
int **receiveNetMess**(unsigned int devSubAdd, unsigned int sendSubAdd, byte *buff, byte maxlen);	Verifica se è arrivato un messaggio a lui indirizzato. Ritorna 0 se non è arrivato nulla. Se sendSubAdd è 0 accetta messaggi da tutti altrimenti restituisce -1 se il mittente non corrisponde. Questa funzione va eseguita ripetutamente dopo aver utilizzato la precedente funzione.
unsigned int **getSubNetSender**(); byte **getMarker**();	Restituisce il mittente Restituisce il marker

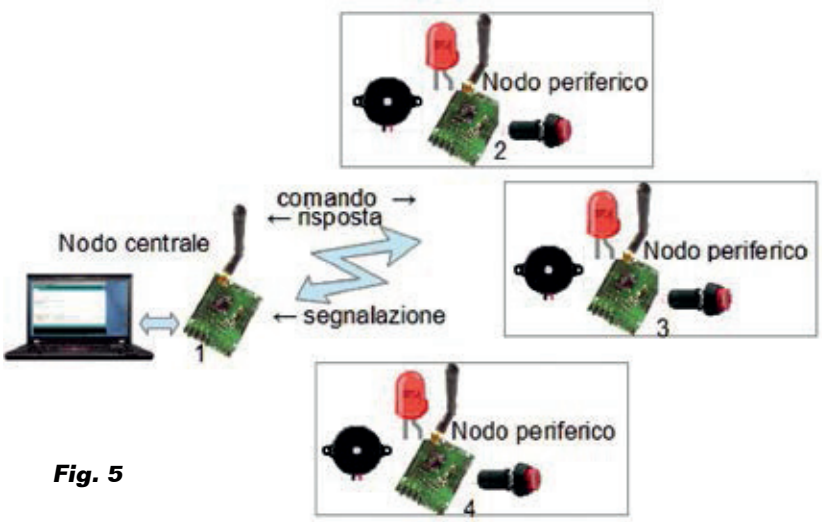


Fig. 5

codifica (con l'eccezione dell'indirizzo destinatario) e lo spedisce. Il destinatario non è codificato per velocizzare il riconoscimento all'arrivo. Appena un pacchetto è arrivato, la funzione di ricezione per prima cosa controlla il destinatario: se coincide con il suo indirizzo procede, altrimenti lo scarta. Se è di sua competenza lo decodifica e verifica se lo aspettava dal mittente indicato (può anche accettare messaggi da qualunque mittente). Alla fine restituisce il numero di byte del messaggio contenuto nel buffer predisposto. Restituisce 0 se non è in arrivo alcun messaggio, oppure se il messaggio non è di sua competenza. Nel caso di messaggio arrivato si può chiedere il mittente o il marker casuale.

FUNZIONI DI LIBRERIA LORA

Le funzioni basilari per utilizzare la scheda in modalità LoRa all'interno di una struttura di rete così definita sono veramente poche. La **Tabella 2** ne propone il riassunto. Ora vi mostriamo un semplice esempio che evidenzia la semplicità d'uso: è contenuto nella

Listato 1 - Sketch del nodo centrale

```
#include <LORA.h>
#include <SPI.h>

#define LORANET 333    // indirizzo di rete
#define DEVRANGE 16   // range di indirizzi di nodo
#define THISADD 1          // indirizzo di questo nodo
#define KEYVAL 111         // valore per la generazione della chiave AES

#define RXTIMEOUT 500    // timeout di replay

#define inplen 64        // buffer per la spedizione
#define reclen 64        // buffer per la ricezione

LORA LR;                  // Class LORA instance
boolean SHIELD=true;
:
:
void setup()
{
  Serial.begin(9600);
  if (!LR.begin(KEYVAL))
    {Serial.println("No LoRa shield detected! Stop!");SHIELD=false;return;}
  LR.setNetAddress(LORANET,DEVRANGE);
  Serial.print("Net address  : ");Serial.println(LORANET);
  Serial.print("Devices range: ");Serial.println(DEVRANGE);
  Serial.print("My address   : ");Serial.println(THISADD);
  Serial.println("LoRa transmitter ready...");
  SX.setPower(PWR);
  showConfig();
  LR.receiveMessMode();   // set shield in continuous receiving mode
}

void loop()
{
  delay(200);
  if (!SHIELD) return;    // se la scheda non è operativa non può fare nulla
  int nb;
  if ((nb=getInput())>0)                    //Se è stato inserito un comando
       {if (sendBuff(nb))getReplay();}
  if ((nb=getMess())>0) {}  //Se è arrivato un messaggio spontaneo lo stampa
}
```

Listato 2 - Sketch per i nodi periferici

```
#include "LORA.h"
#include <SPI.h>
/* Simulazione di sensori ed attuatori*/
#define psound 9          // pin for buzzer
#define pingo 8             // pin for push-button
#define pinf  7              // pin for signaling led
#define LORANET 333    //indirizzo di rete
#define DEVRANGE 16   //range di indirizzi di nodo
#define THISADD 2          //indirizzo di questo nodo (gli altri nodi lo avranno diverso)
#define KEYVAL 111         // valore per la generazione della chiave AES
#define MASTER 1           // indirizzo del nodo centrale
LORA LR;                  // Class LORA instance
boolean SHIELD=true;
char recbuff[reclen];   //receiving buffer
char sendbuff[reclen];  //transmitting buffer
:
:

void setup()
{
  pinMode(pingo,INPUT_PULLUP);
  digitalWrite(pinf,0);
  pinMode(pinf,OUTPUT);
  pinMode(psound,OUTPUT);
  Serial.begin(9600);
  if (!LR.begin(KEYVAL))
    {Serial.println("No LoRa shield detected! It can't perform echo!");SHIELD=false;return;}
  Serial.println("LoRa receiver.");
  LR.setNetAddress(LORANET,DEVRANGE);
  Serial.print("Net address   : ");Serial.println(LORANET);
  Serial.print("Devices range: ");Serial.println(DEVRANGE);
  Serial.print("My address    : ");Serial.println(THISADD);
  SX.setPower(PWR);
  showConfig();
  Serial.println("Waiting for message...");
  LR.receiveMessMode();   // set shield in continuous receiving mode
}
void loop()
{
  int nb;
  if (!SHIELD) {delay(200);return;}           // se la scheda non è operativa non può fare nulla
  if ((nb=getMess())>0)                          //Se riceve un comando lo esegue e risponde
       {decode(nb);LR.receiveMessMode();} // poi si rimette in ricezione
   if (getInput())                                  //Se il pulsante/interruttore è premuto manda un avviso
       {sendWarn();LR.receiveMessMode();} // poi si rimette in ricezione
}
```

libreria e in questa sede ne vedremo le caratteristiche principali, lasciando ai lettori l'analisi del listato completo. L'esempio è formato da due sketch e rappresenta una struttura a stella in cui un nodo interroga gli altri, ma può ricevere anche degli avvisi spontanei nel caso che sulla periferica si realizzi un certo evento. Nella nostra simulazione l'evento è l'azione su un pulsante (**Fig. 5**). Il primo sketch rappresenta il nodo centrale ed ha indirizzo 1 (indirizzo di nodo all'interno di una rete con 15 possibili nodi). Lo sketch inizializza la libreria e la scheda e poi si mette in attesa di un input da console o di un messaggio da una periferica. In caso di input da console (ovvero un comando) invia il corrispondente messaggio ed attende la risposta che visualizzerà sulla console. Nel caso di messaggio spontaneo, lo visualizzerà sulla console. Trovate il codice corrispondente nel **Listato 1**. I comandi inseribili sono relativi alla richiesta di un eco, ad accendere/spegnere un LED, a far suonare un cicalino o a verificare l'azione su un pulsante o un interruttore. Lo sketch del nodo periferico (**Listato 2**), invece, non fa altro che attendere l'arrivo di un messaggio e verificare l'azione su un pulsante o un interruttore. Nel caso arrivi un messaggio (co-

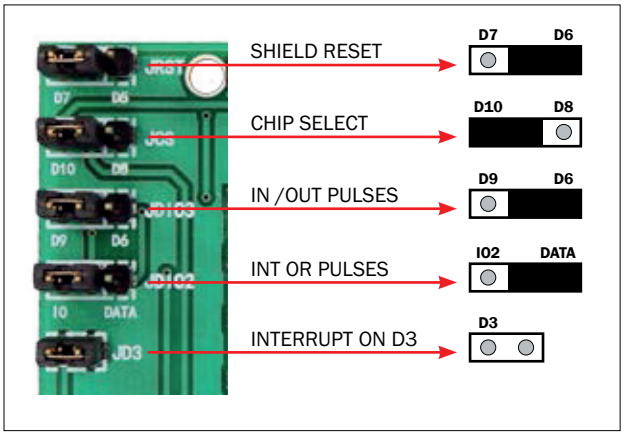


***Fig. 6** - Impostazione della libreria.*

mando) lo esegue e risponde. Nel caso verifichi il cambiamento di stato di un pulsante o interruttore, spedisce autonomamente un messaggio di attenzione.

CONCLUSIONI

In queste tre puntate abbiamo analizzato un componente molto sofisticato e flessibile come l'SX1278, sul quale abbiamo realizzato una shield di basso costo. Lungi dal pensare di aver esaurito le caratteristiche e le potenzialità di questa innovativa tecnologia, speriamo di aver realizzato una pratica ed efficiente base di lavoro per sviluppare interessanti applicazioni.
Sul sito di GitHub (github.com/open-electronics/LoRa) è stata aperta una pagina relativa a questo progetto, dove oltre alla libreria potrete trovare applicazioni che sia noi ma soprattutto voi, vorrete condividere. Le applicazioni, lo abbiamo detto, sono pressoché infinite: dal sistema di allarme casalingo alla gestione dei sensori/attuatori di una piccola impresa. Noi, per conto nostro, stiamo già pensando ad una miniaturizzazione del transceiver che appoggiandosi sempre su Arduino (in formato miniaturizzato), possa risolvere situazioni dove lo spazio e/o l'alimentazione sono critiche. Per esempio un radiocomando bidirezionale tascabile. Poiché pensiamo che questa tecnologia sia valida in termini di rapporto prestazioni/prezzo, vi invitiamo a scatenare la vostra fantasia e a suggerirci o a sottoporci progetti in tema. Vi ricordiamo che con l'utilizzo di RandA su Raspberry la scheda LoRa può diventare un completo gateway per controllare periferiche WorldWide.

per il MATERIALE

La shield Long Range (cod. FT1190M) comprensiva di modulo radio e tutti i componenti necessari al suo funzionamento, è disponibile a 39,50 Euro. Il modulo DRF1278F è disponibile anche separatamene a 19,00 Euro. Arduino UNO (cod. ARDUINOUNOREV3) viene venduto a 24,50 Euro. Tutti i prezzi si intendono IVA compresa.

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

PRESEPINO:
IL PRESEPE CON ARDUINO

di BORIS LANDONI

Centralina capace di simulare ciclicamente l'avvicendarsi del giorno e della notte, pilotando opportunamente fino a sei uscite (con cui alimentare LED o piccoli motori) corrispondenti alla luce del giorno, al bagliore delle stelle, al fuoco delle case, all'accensione della stella cometa e a due dispositivi mobili. Può, inoltre, controllare strip NeoPixel e, per rendere l'atmosfera più natalizia, riproduce musiche a vostra scelta.

Come ogni anno, all'approssimarsi delle Feste Natalizie pensiamo a un progetto a tema che possa renderle originali e più calorose. Dato che da tanti anni non proponevamo più quello che è poi uno dei classici dell'elettronica -il controllo delle luci del presepe- abbiamo deciso di optare per una centralina del genere, ma ci è sembrato giusto farlo in chiave moderna ed attuale; e per attuale intendiamo con un hardware conosciuto e apprezzato come quello che migliaia di sperimentatori hanno trovato in Arduino. Ecco nascere, così, Presepino, ossia una centralina di controllo luci e sonoro per presepe, il cui circuito è basato su un ATmega 328 (quindi l'hardware di Arduino UNO) e nasce per comandare direttamente dei LED, con tutti i vantaggi del caso, primo fra tutti il fatto che lavora a bassa tensione e può quindi essere collocata anche a portata dei bambini, senza che ne derivi pericolo per la loro salute. Poi oggi, con la grande disponibilità di strip a LED, realizzare un presepe automatizzato piccolo o grande che sia, risulta semplicissimo. Oltre alle luci, il nostro Presepino permette di gestire anche l'audio, grazie a un innovativo modulo riproduttore di MP3 montato sulla scheda, capace di farvi ascoltare in altoparlante (seppure in mono) i brani contenuti in una SD-Card che potrete scegliere secondo le vostre preferenze.

IL PROGETTO

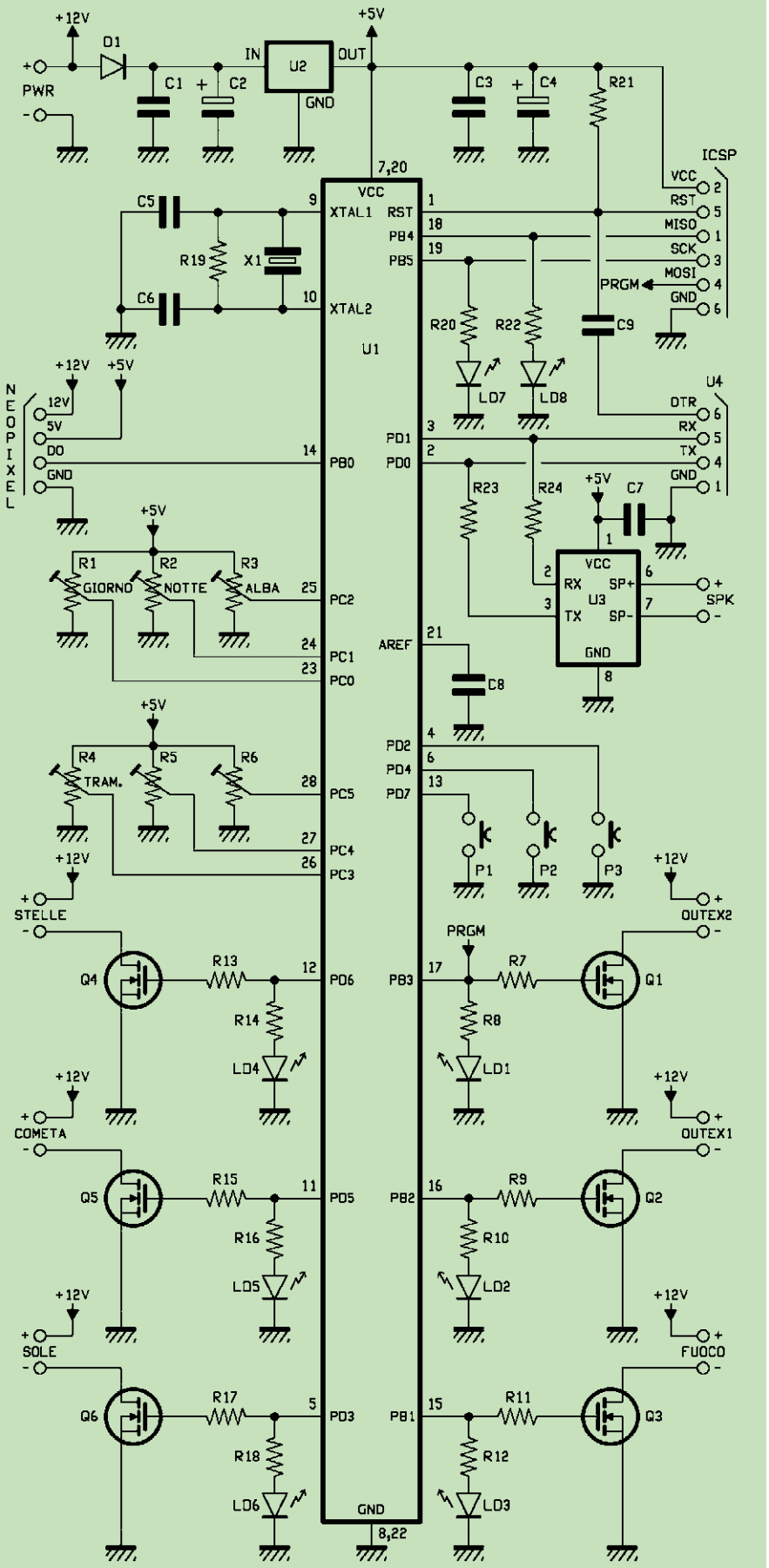
Ma andiamo con ordine e vediamo innanzitutto le funzioni disponibili, che differenziano la nostra centralina da quelle per il semplice controllo delle due luci che realizzano la simulazione di alba e tramonto: con Presepino è possibile realizzare un controllo che va oltre il piccolo o grande presepe di casa, per arrivare ai presepi allestiti nelle chiese, negli oratori, nei locali delle comunità e nei luoghi aperti. Per questo motivo abbiamo realizzato un circuito decisamente prestante, in grado di pilotare quattro carichi luminosi a LED funzionanti a 12 volt e che assorbano una corrente

individuale di 6 ampere; questa è anche la massima corrente che la scheda può gestire e può essere aumentata stagnando abbondantemente le piste che portano dalla morsettiera di alimentazione ai MOSFET e da questi alle relative morsettiere di uscita. Per corrente individuale si intende che ciascun canale può commutare 6 ampere, ma per come è costruito lo stampato non è possibile assorbire 6 ampere costantemente da tutte le uscite insieme. Questi valori, considerato che il circuito nasce per pilotare strip a LED o comunque composizioni di LED, sono più che accettabili e per un presepe da casa sono sovrabbondanti, pur bastando anche per una struttura all'aperto, grazie all'efficienza luminosa dei LED.
Le uscite della centralina sono sei, ma il firmware che forniamo contempla la gestione delle prime quattro per implementare la simulazione della luce del giorno, di quella delle stelle, dei fuochi della capanna/grotta (di Betlemme) e delle case dei pastori, ed infine di illuminare la stella cometa. Le luci si accendono e spengono gradatamente seguendo un ciclo che simula un'intera giornata.
A queste uscite di base potete aggiungere, intervenendo sullo sketch per l'ATmega 328, le altre due, cui potete assegnare il controllo di vari accessori del presepe come una ruota a pale, il movimento di personaggi animati o animali, ecc.
Per quanto riguarda le uscite di base, abbiamo suddiviso la sequenza completa in quattro fasi denominate giorno, tramonto, notte e alba. La durata delle fasi può essere regolata mediante trimmer, e l'escursione corrispondente si può reimpostare intervenendo sullo sketch (ma in maniera uguale per tutte); ciò vale dunque per il giorno e la notte, ma anche per le due fasi di transizione (tramonto e alba). Queste ultime due sono le fasi più suggestive: durante il tramonto, la luminosità del giorno (realizzata con i LED collegati all'uscita SOLE) diminuisce a poco a poco, mentre nel cielo iniziano a illuminarsi le stelle (si attiva l'uscita omonima); a un certo punto, prima che il ciclo si concluda, si accendono i fuochi delle case e della capanna (uscita FUOCO). Tra l'altro, il nostro circuito è in grado di simulare il tremolio della legna che arde, pilotando con un'opportuna sequenza di impulsi l'uscita FUOCO. Quando tutte le stelle in cielo sono completamente illuminate, appare anche la stella cometa (accesa dall'uscita COMETA). Ovviamente, salvo piccoli dettagli, durante l'alba si spengono gradatamente tutte le luci luminosi, mentre aumenta lentamente sino a raggiungere la massima luminosità la luce del giorno.

SCHEMA ELETTRICO

Scendiamo ora negli aspetti più tecnici, analizzando il circuito, che è basato sul microcontrollore Atmega 328 nel quale gira l'apposito sketch che implementa il controllo delle uscite e del modulo che realizza il sonoro del presepe; il micro, siglato U1, all'inizializzazione imposta PB0 come uscita per l'invio dei comandi seriali a un'eventuale strip Neopixel, e PB1, PB2, PB3, PD3, PD5, PD6 come output per il controllo delle uscite a MOSFET a canale N (i transistor sono dei robusti STP36N06, da 36 ampere di Id e 60V di Vdso) che materialmente commutano l'alimentazione sui LED o strip a LED corrispondenti. Le linee del micro utilizzate per gestire le uscite pilotano ciascuna il gate del proprio MOSFET attraverso un resistore da 1 kohm e ciascuna di esse e massa è applicato un LED (con rispettiva resistenza di limitazione della corrente) che accendendosi permette di monitorare lo stato del rispettivo transistor; ciascuna delle uscite fornisce impulsi rettangolari TTL-compatibili di frequenza costante e modulati in larghezza, secondo la tecnica PWM: ciò consente di variare il valore medio della tensione applicata dai MOSFET ai carichi loro collegati e di conseguenza la potenza e la luminosità, con un'elevatissima efficienza in quanto la resistenza in stato di ON (piena conduzione) dei MOSFET è bassissima (appena 40 milliohm) e quindi la potenza dissipata, a parità di potenza trasferita, è limitatissima se confrontata a quella che comporterebbe un pilotaggio serie a tensione continua, dove il transistor dovrebbe sobbarcarsi la differenza tra la tensione di alimentazione e quella applicata al carico.
I segnali PWM vengono generati dai moduli PWM interni al microcontrollore, opportunamente impostati dal firmware per ciascuna uscita.
L'attuale configurazione circuitale prevede di alimentare le strip di LED collegandone positivo e negativo rispettivamente a + e – della relativa uscita, tuttavia siccome i MOSFET possono "reggere" tensioni drain-source in interdizione pari a 60 volt, è anche pensabile partire da un'alimentazione esterna anche a 24÷36 volt c.c. in modo da ridurre l'assorbimento a parità di potenza elettrica gestita. Se l'alimentazione non viene presa dal circuito, la striscia di LED va connessa con il + al positivo dell'alimentatore scelto e il – all'uscita della nostra centralina, fermo restando che la

massa dell'alimentatore e della scheda vanno unite.
Abbiamo detto in precedenza che la sequenza completa generata dal nostro circuito è composta da quattro fasi (giorno, tramonto, notte e alba) la cui durata può essere regolata indipendentemente tramite gli appositi trimmer alimentati a 5 volt e la cui tensione del cursore viene letta da un I/O del microcontrollore configurato come input e acquisito tramite l'A/D converter interno all'ATmega; nel nostro progetto, l'ADC viene assegnato alle linee PC0÷PC5, con le quali si acquisiscono in scansione le tensioni dei sei trimmer presenti, vale a dire R1, R2, R3, R4 per le funzioni anzidette ed R5 e R6 disponibili per funzioni assegnabili a piacimento programmando opportunamente il microcontrollore.
Per come è strutturato il firmware, la durata dei rispettivi cicli è direttamente proporzionale alla tensione applicata dal relativo trimmer, quindi portando il cursore di R1 verso massa, per esempio, si accorcia la durata del giorno, la quale invece aumenta portandolo verso il positivo dei 5 V.
Attenzione ad un particolare: i trimmer R1, R2, R3, R4 non controllano direttamente il PWM delle uscite per le fasi, ma ne determinano i tempi di esecuzione; il duty-cycle dei segnali PWM alle uscite SOLE, STELLE, COMETA e FUOCO viene gestito secondo la periodicità prevista da apposite subroutine del firmware. Questo funzionamento è logico in quanto le fasi descritte coinvolgono più di un'uscita e, nel caso della transizione giorno/notte (TRAMONTO), per esempio, coinvolgono l'uscita SOLE e quella STELLE, che vanno in dissolvenza, come è, anche nel passaggio notte/giorno (ALBA),



Listato 1

```
  if(runtime>time_trim+200){
    for (int i=0;i<6;i++){
      trimvalue[i]=(ReadTrimmer(i));
      trimvalue[i]=trimvalue[i]*100;
      delay (1);
    }
    time_trim=millis();
  }


int ReadTrimmer (int id){
  return(analogRead(trimmer[id]));
}
```

seppure al contrario.
I trimmer da R1 a R4 servono dunque per gestire la durata del GIORNO (R1), della NOTTE (R2), dell'ALBA (R3) e del TRAMONTO (R4). Il led LD8, pilotato dalla linea PB4, si accende brevemente quando la centralina passa da una fase all'altra.
La durata di ogni singola fase viene impostata tramite l'apposito trimmer e la durata massima di ogni fase è di circa 100 secondi; per l'esattezza, la durata è ottenuta dalla lettura dell'ADC su 10 bit (tra 0 e 1.023) considerata in millisecondi, moltiplicata per 100 dal firmware. Se volete tempi differenti potete editare lo sketch nella porzione che si occupa di determinare la temporizzazione, che per praticità riportiamo nel **Listato 1**. Si tratta sostanzialmente della riga:

*trimvalue[i]=trimvalue[i]*100;*

Qui potete modificare il valore 100 sostituendolo con ciò che vi aggrada: ad esempio, inserendo 200 ottenete il raddoppio dei tempi, mentre con 50 li dimezzate.
Quanto agli altri due trimmer, sono personalizzabili da firmware, così come le uscite OUTEX1 e OUTEX2; con lo sketch da noi fornito, R5 è libero mentre R6 è stato utilizzato per gestire il volume di riproduzione dell'audio.
Alle uscite a MOSFET possono essere collegate sia delle strip a LED, sia lampadine a 12 volt di piccola potenza, ma anche singoli LED, motori elettrici per azionare ruote e altri meccanismi, come anche pompe per far muovere l'acqua in un laghetto o un fiume e tutto quello che la fantasia suggerisce per realizzare un presepe più "vivo".
Restando in tema uscite, è il caso di parlare di quella dedicata al controllo di LED NeoPixel, che permette di aggiungere ulteriori animazioni luminose, come ad esempio la coda pulsante di una cometa. Le strip NeoPixel vengono gestite in parallelo con una sola linea del microcontrollore (più in generale, di Arduino), che nel nostro caso è facilmente assegnabile a piacimento specificandola nello sketch; la comunicazione è unidirezionale e gestisce un gruppo di LED.
NeoPixel è una particolare soluzione a LED RGB "intelligenti" per il controllo, ad esempio, tramite Arduino; ogni LED NeoPixel dispone di controller a bordo, interfacciabile con una sola linea dati. La libreria proprietaria che l'Adafruit (*www.adafruit.com*) rende disponibile liberamente facilita l'integrazione nell'ambiente Arduino.
Ogni LED RGB può essere gestito individualmente tramite un apposito comando inserito nella stringa seriale e può produrre fino a 256 tonalità del proprio colore, determinando un totale di

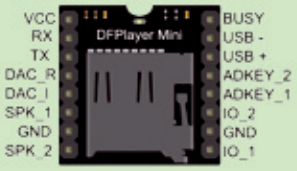
Listato 2

```
void gestAudio(int track){

  digitalWrite (LEDPLAY,LOW);
  if (playing==true){
      digitalWrite (LEDPLAY,HIGH);
      track++;
      if (track==5) track=1;
        if (stopVol==false){
          if(runtime>time_volume_running+200){
            if(startVol==false){
              if (volume>3) {
                volume=volume-3;
                Serial.print("dec ");
                Serial.println(volume);
              }
              else
              {
                playtrack(track);
                startVol=true;
              }
            }
            else
            {
              if (volume<(trimvalue[TRIMMER_VOL]/3500)) {
                volume=volume+3;
               Serial.print("inc ");
               Serial.println(volume);
              }
              else
              {
                startVol=false;
                stopVol=true;
                Serial.println("finito ciclo");
              }
            }
          time_volume_running=millis();
          setvolume(volume);
          }
        }
      }
}
```

16.777.216 combinazioni di colore. Una particolarità dei LED NeoPixel è che possono essere collegati in cascata in modo che la linea dati da uno passi al successivo, però il prezzo da pagare è che oltre un certo numero di LED, la velocità di gestione deve ridursi sensibilmente; a causa di ciò, se si devono realizzare matrici per mostrare della grafica veloce, occorre impiegare molte linee con pochi LED ciascuna. Ma questa limitazione non interessa il nostro progetto, nel quale peraltro non è stata prevista una funziona predefinita per l'uscita NeoPixel, che potrete invece aggiungere editando lo sketch.
Il canale dati usato per la comunicazione con i LED NeoPixel e quindi con le strip, è simile al tipo oneWire; la comunicazione avviene a un massimo di 800 kbps.
Per ogni strip di LED è possibile impostare a piacimento la frequenza di refresh, in modo da rendere impercettibili determinati giochi di luce, fermo restando il limite dettato dalla quantità di strip collegate.
Il protocollo di comando del sistema NeoPixel prevede l'invio di gruppi di tre byte in una stringa di 24 bit, ognuna delle quali contiene lo stato di illuminazione di ciascun colore base (prima gli otto bit del verde, poi quelli del rosso e infine quelli del blu).
L'alimentazione prevista per i LED NeoPixel è a 5 volt, prelevabili dal contatto 5V del connettore della scheda Presepino, dato che l'assorbimento di ciascuna strip normalmente non è elevato e che i LED tricolore NeoPixel vengono accesi alternativamente. La massa di riferimento di alimentazione e dati (è unica e fa capo al contatto G della strip) è sempre quella del microcontrollore e va quindi connessa a

Il modulo **DFR0299**

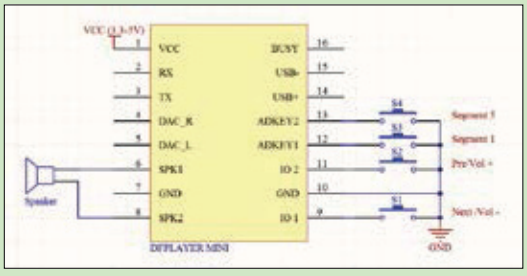
Il componente che utilizziamo per la riproduzione dell'audio permette di far riprodurre brani musicali MP3 senza dover utilizzare hardware costoso e impegnativo sul fronte dell'elaborazione: infatti il DFR0299 è un completo decoder che attinge a un supporto SD-Card, decifra l'audio e poi lo amplifica on-board pronto per inviarlo a un altoparlante (da 8 ohm di impedenza) con quasi 3 watt di potenza. È quindi l'ideale quando si desiderano riprodurre file audio standard come gli MP3 ma non si dispone delle risorse di calcolo (CODEC compreso) che la loro elaborazione diretta richiederebbe.
Il modulo nasce per essere gestito da Arduino, il quale di fatto si limita a inviargli comandi seriali per gestire la riproduzione; tutto il resto lo fa il DFR0299.
Questo dispositivo può essere utilizzato in stand-alone, alimentandolo a 5 volt (la corrente richiesta per erogare 3 watt su 8 ohm, che si ottengono a 5 volt, è 0,45 ampere), collegandogli un altoparlante e dei pulsanti. Oppure può essere usato in combinazione con un Arduino UNO o qualsiasi altro dispositivo dotato di porta seriale TTL. Dispone di controllo tramite pin di I/O, tramite seriale o tramite ingresso analogico. Supporta fino a 100 cartelle; ogni cartella può contenere fino a 255 brani MP3. La regolazione del volume, sia essa ottenuta per via seriale o localmente tramite pulsanti, prevede 30 livelli (± dalla posizione centrale). L'alimentazione prevista è da 3,2V a 5V in continua (la corrente assorbita in standby è pari a 20 mA). Le principali caratteristiche del modulo sono:



Piedinatura del modulo DFR0299.

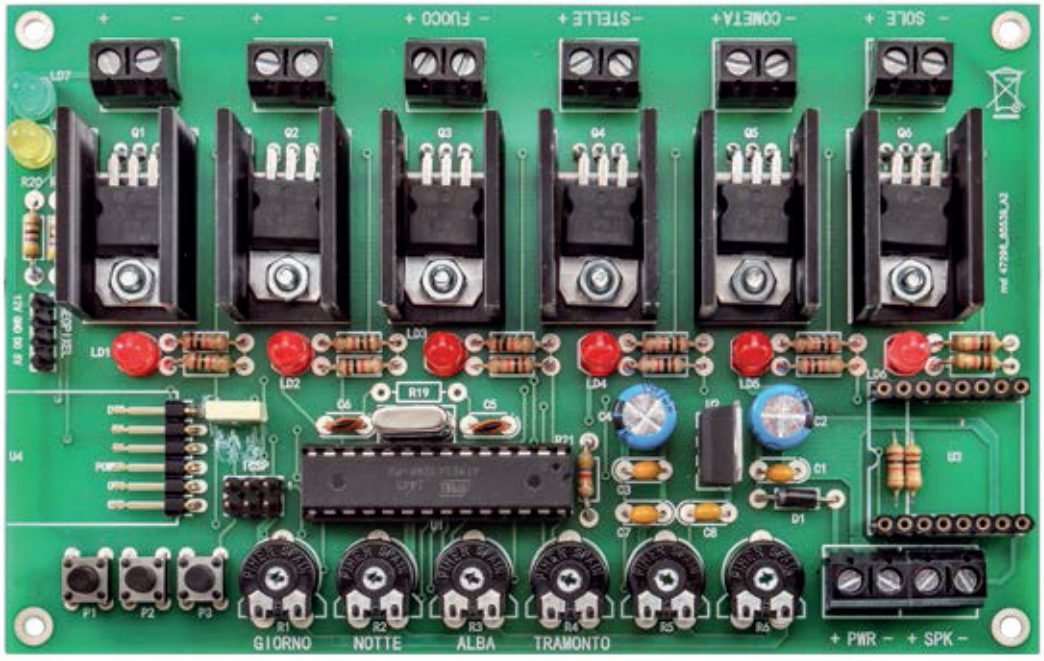
- decodifica MP3 11172-3 e ISO13813-3 layer3 audio decoding;
- possibilità di equalizzazione Pop, Rock, Jazz, Classica;
- frequenze di campionamento in riproduzione (kHz): 8/11,025/12/16/22,05/24/32/44,1/48
- DAC a 24 -bit, che supporta una gamma dinamica di 90 dB e un rapporto segnale/rumore di 85dB;
- supporto file system FAT16 e FAT32;
- massima capacità di memoria indirizzabile pari a 32 GB, 32G di USB disk, blocchi di 64 MB Flash NOR;
- dimensioni (mm): 20 x 20 x 13.

Per quanto nel nostro progetto lo utilizziamo in mono, il modulo dispone di un decoder MP3 stereo le cui uscite sono disponibili a basso livello e impedenza relativamente alta per pilotare amplificatori, ingressi di mixer, cuffie a 300 ohm ecc. L'amplificatore di potenza incorporato è mono e amplifica la miscelazione dei canali destro e sinistro. La tabella in questo riquadro riporta la piedinatura del modulo.



Il modulo può essere utilizzato autonomamente con soli quattro pulsanti: S1 e S2 hanno la duplice funzione di controllo volume (premuti a lungo) o di skip tra brani (pressione breve).

N°	Pin	Description	Note
1	VCC	Input Voltage	DC3.2~5.0V;Type: DC4.2V
2	RX	UART serial input	
3	TX	UART serial output	
4	DAC_R	Audio output right channel	Drive earphone and amplifier
5	DAC_L	Audio output left channel	Drive earphone and amplifier
6	SPK2	Speaker-	Drive speaker less than 3W
7	GND	Ground	Power GND
8	SPK1	Speaker+	Drive speaker less than 3W
9	IO1	Trigger port 1	Short press to play previous (long press to decrease volume)
10	GND	Ground	Power GND
11	IO2	Trigger port 2	Short press to play next (long press to increase volume)
12	ADKEY1	AD Port 1	Trigger play first segment
13	ADKEY2	AD Port 2	Trigger play fifth segment
14	USB+	USB+ DP	USB Port
15	USB-	USB- DM	USB Port



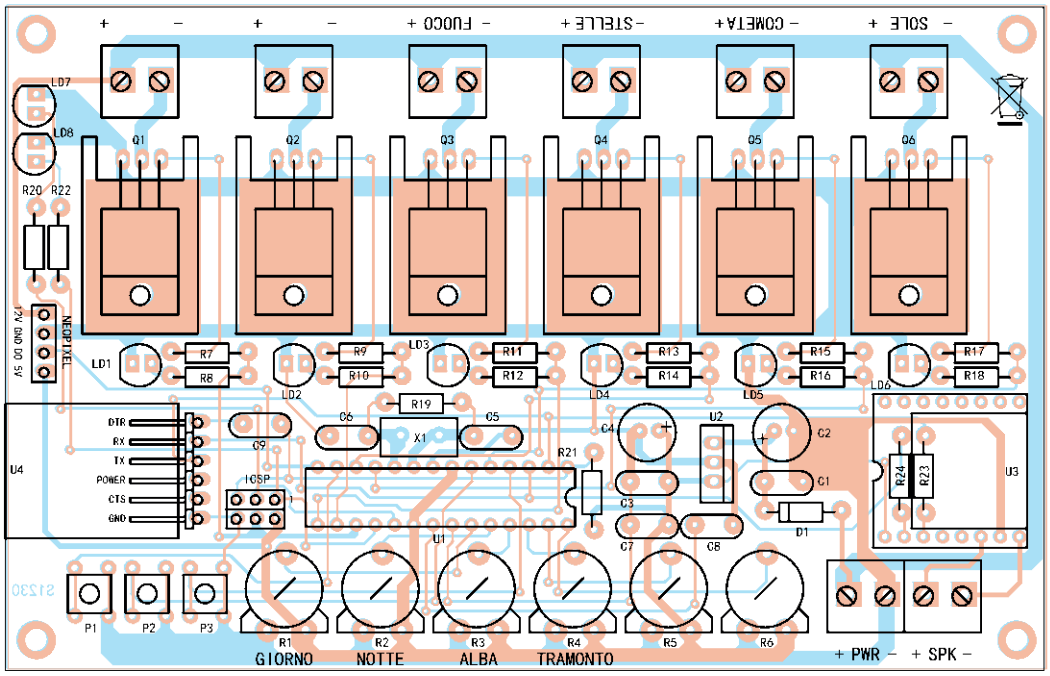
Elenco Componenti:

R1÷R6: Trimmer 10 kohm MO
R7÷R18: 1 kohm
R19: -
R20, R22: 470 ohm
R21: 4,7 kohm
R23, R24: 1 kohm
C1, C3: 100 nF ceramico
C2, C4: 220 μF 25 VL elettrolitico
C5: 15 pF ceramico
C6: 15 pF ceramico
C7÷C9: 100 nF ceramico
D1: 1N4007
U1: ATMEGA328P (MF1230)
U2: 7805
U3: Modulo DFPlayer Mini (DFR0299)
U4: Convertitore USB-TTL

GND. Per gestire eventualmente strip molto lunghe e quindi con un assorbimento più elevato, sul connettore siglato NEOPIXEL abbiamo portato anche i 12 volt, prelevati direttamente a monte del diodo D1. A questa uscita è possibile ad esempio collegare un regolatore switching in grado di fornire la corrente necessaria all'intera fila di led NEOPIXEL.
Passiamo adesso all'audio del presepe, che viene ottenuto grazie a un valido modulo siglato DFR0299 (lo trovate su *www.futurashop.it/mini-riproduttore-mp3-montato-9145-dfr0299*) controllabile tramite interfaccia seriale a livello TTL; nel nostro circuito utilizza la seriale hardware del microcontrollore, la stessa impiegata in fase di programmazione per caricare lo sketch sul microcontrollore, ma ciò non costituisce un problema in quanto le linee RX e TX dell'ATmega vengono usate per una funzione alla volta, ovvero una volta che è stato programmato il micro, lo sketch le utilizza per gestire il modulo. Il DFR0299 nello schema lo trovate siglato U3; è in grado di riprodurre direttamente i file MP3 e WAV memorizzati su una SD-Card della capacità massima di 32 GB purché formattata con FAT16 o FAT32, inserita nell'apposito slot.
La particolarità di questo modulo è che è stato progettato e realizzato per il mondo Arduino: anche nella scelta del modulo, vedete, è evidente il vantaggio di avere a bordo della nostra scheda un'architettura Arduino. Per la gestione del modulo è chiaramente stata sviluppata una libreria specifica (scaricabile anche dal nostro sito *www.elettronicain.it* insieme agli altri file del progetto) che il nostro sketch include, la quale permette di decidere quale file riprodurre, regolare il volume ecc.
Il modulo contiene un decoder in grado di decomprimere l'audio in formato MP3 e un controller in grado di accedere via SPI ai dati contenuti nella SD-Card: man mano che lo stream di dati viene letto, il decoder lo trasforma in audio non compresso, che poi viene amplificato da un piccolo finale integrato, mono con uscita a ponte, da 3 watt, che sono una potenza più che sufficiente a pilotare un altoparlante in grado di farsi sentire nell'ambiente. Se serve maggior potenza, utilizzate l'uscita audio (+ e – SPK) per pilotare un amplificatore di potenza cui applicare una cassa acustica; in alternativa potete prendere l'audio dai contatti 4 e 5, che sono le uscite audio a basso livello dei canali, rispettivamente, LEFT e RIGHT. In questo caso potete contare su un audio stereo.
Tutte le funzioni del modulo U3 si governano da seriale con specifiche istruzioni che il microcontrollore impartisce; quelle riguardanti il volume vengono generate leggendo l'ADC dell'ATmega quando campiona la tensione sul cursore del trim-

(FTDI5V)
LD1÷LD6: LED rosso 5 mm
LD7: LED verde 5 mm
LD8: LED giallo 5 mm
X1: Quarzo 16 MHz
Q1÷Q6: STP36NF06
P1÷P3: Microswitch

Varie:
- Morsetto 2 poli passo 5,08mm (8 pz)
- Strip femmina 8 vie (2 pz.)
- Strip maschio 3 vie (2 pz.)
- Strip maschio 4 vie
- Strip maschio 6 vie 90°
- Zoccolo 14+14
- Dissipatore (6 pz.)
- Vite 10 mm 3 MA (6 pz.)
- Dado 3 MA (6 pz.)
- Circuito stampato S1230

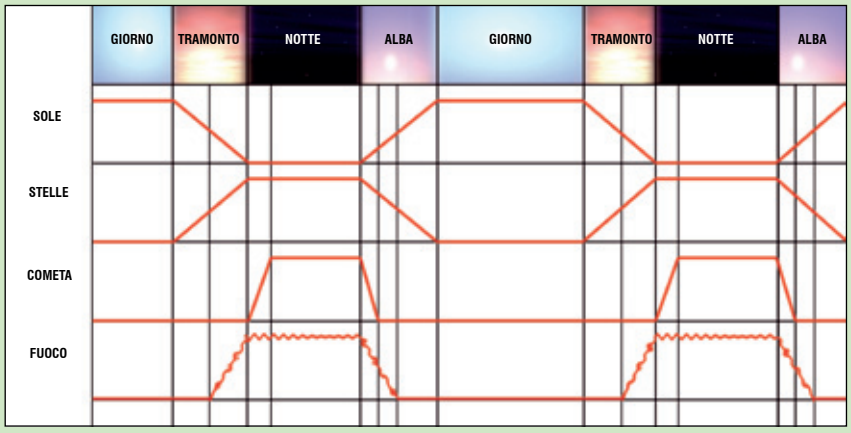


mer R6. Abbiamo anche previsto di utilizzare, tra i comandi di gestione della riproduzione (Play, Stop, Pause, FWD ecc.) solo quello di Play/Stop, che viene implementato con i tre pulsanti collegati al microcontrollore tramite le linee PD2, PD4 e PD7 (tutte inizializzate come input dotati di pull-up interno) che hanno tutti la stessa funzione, ovvero quella di abilitare/disabilitare l'audio (**Listato 2)**. Più esattamente, la pressione di uno dei pulsanti P1, P2, P3 quando è in riproduzione un brano causa l'arresto del medesimo e la successiva pressione fa ricominciare la riproduzione dall'inizio del brano arrestato in precedenza.
Quando l'audio è in riproduzione, il LED LD7 viene acceso dalla linea PB5 del microcontrollore, la quale nel normale funzionamento è utilizzata come output; le linee PD0 e PD1, facenti capo all'UART, vengono accoppiate al modulo U3 mediante resistenze per evitare che in programmazione, ovvero quando alla scheda si applica il convertitore Seriale/USB, vi siano interferenze tra le tensioni dei due moduli.
I file dell'audio che volete sia riprodotto devono essere scritti sulla SD-Card con nome nel formato 001.mp3, 002.mp3, 003.mp3 e 004.mp3, perché il primo file viene richiamato durante la fase GIORNO, il secondo durante il TRAMONTO, il terzo quando viene acceso il FUOCO (fuoco tremolante) e il file 004.mp3 viene richiamato quando arriva l'ALBA.
Durante il passaggio tra una fase e l'altra, il volume di riproduzione viene abbassato gradualmente fino al termine del brano e viene avviata, con volume in graduale aumento, la riproduzione del brano successivo (tipo ciò che avviene alla radio o negli album musicali); questo vale nel passaggio dal brano da una fase alla seguente.
Detto ciò completiamo la descrizione del circuito con le parti finora tralasciate, ossia l'alimentazione, l'oscillatore dell'ATmega 328P e la programmazione in-circuit; per quanto riguarda la sezione di alimentazione, il circuito va alimentato con 12÷14 Vcc (la corrente richiesta dipende da quanti utilizzatori collegate alle uscite) tramite la morsettiera +/- PWR, la cui tensione giunge direttamente alle uscite a MOSFET e al connettore per NeoPixel. La tensione di alimentazione principale giunge anche all'anodo del diodo D1, inserito per proteggere il resto del circuito dall'inversione di polarità, e viene applicata all'ingresso del regolatore di tensione U2, compito del quale è ricavare i 5 volt stabilizzati con cui funzionano il microcontrollore, i componenti di contorno e il modulo audio U3. L'eventuale converter Seriale/USB si alimenterà con la tensione dell'USB del computer cui verrà interfacciato. I condensatori posti sull'alimentazione d'ingresso servono a filtrare gli

Funzionamento delle **uscite**

La sequenza di accensione delle luci controllate dal nostro circuito. Di giorno l'unica lampada (o serie di lampade) accesa è quella che simula il sole e che illumina, appunto, "a giorno" il presepe. Questa lampada resta accesa per un tempo compreso tra 0 e 102 secondi a seconda di come viene settato il trimmer R1. Scaduto il tempo ha inizio una nuova fase: il tramonto, la cui durata dipende dal trimmer R4. Agendo su detto trimmer è possibile impostare un valore compreso tra 0 e 102 secondi circa. A poco a poco la luce del giorno si attenua mentre aumenta la luminosità delle lampade che simulano le stelle, sino al completo spegnimento del sole che coincide con la massima illuminazione delle stelle. Esattamente a metà di questa fase di transizione iniziano ad accendersi i fuochi delle case che raggiungono la massima luminosità al termine del ciclo. Da notare che questa uscita genera una luce tremolante che simula il bagliore del fuoco. Infine, al termine del ciclo, si illumina gradatamente anche la luce che simula la stella cometa. Il passaggio dalla minima alla massima luminosità avviene in tempo pari ad 1/4 di quello impostato per il TRAMONTO.

A questo punto ci troviamo in piena notte con le stelle in cielo che brillano, la cometa completamente illuminata ed i fuochi nelle case accesi col loro tipico tremolìo. La durata di questa fase (compresa, come per il giorno, tra 0 e 102 secondi circa), che è la NOTTE, viene impostata mediante il trimmer R2. Allo scadere del tempo impostato, ha inizio la quarta ed ultima fase: l'alba. Gradatamente l'intensità luminosa delle stelle si abbassa mentre aumenta la luminosità del giorno sino al completo spegnimento delle stelle ed alla completa accensione delle lampade che simulano la luce del giorno. All'inizio di questa fase anche la cometa si spegne gradatamente ma molto più velocemente tanto che, trascorso un periodo pari ad 1/4 del tempo impostato per l'alba, la cometa risulta completamente spenta. Sempre all'inizio di questa fase di transizione, anche la luminosità dei fuochi inizia a calare sino allo spegnimento. In questo caso il passaggio dalla massima luminosità allo spegnimento completo avviene in un tempo pari ad 1/2 di quello impostato per l'ALBA, mediante il trimmer R3. A questo punto abbiamo simulato un ciclo di 24 ore ed il sistema si appresta a ripetere all'infinito la sequenza programmata.



impulsi creatisi sulle piste per effetto dell'assorbimento dei LED e in generale dei carichi collegati alle uscite a MOSFET; ciò è necessario in quanto la pulsazione dell'alimentazione dei diodi è a frequenza elevata e altrimenti i disturbi (che poi sono abbassamenti di tensione pur lievi concomitanti con l'accensione dei singoli LED) potrebbero interferire con il corretto funzionamento del microcontrollore. Analoga è la funzione dei condensatori C3 e C4.

Veniamo ora all'interfaccia per la programmazione in-circuit, da utilizzare solo se volete caricare il bootloader in un microcontrollore vergine; a riguardo va detto che il microcontrollore fornito dalla Futura Elettronica (*www.futurashop.it*) con il kit è già programmato con bootloader e sketch, ma chi preferisce procurarsi da sè l'ATmega 328P dovrà prima caricare il bootloader per poter trasferire lo sketch dall'IDE di Arduino.

Il connettore preposto è quello siglato ICSP e ad esso fanno capo le linee di reset (RST), MISO (ingresso dati del programmatore e uscita del micro U1), MOSI (uscita dati del programmatore e ingresso dati del micro), SCK (clock della comunicazione seriale sull'SPI formato da MISO e MOSI), oltre che la massa e la Vcc, che durante la programmazione viene fornita dal programmatore per alimentare il circuito.

Ricordiamo che quando si programma il bootloader il circuito deve essere alimentato attraverso la morsettiera +/- PWR.

Concludiamo con l'oscillatore del microcontrollore, che è impostato come clock ricavato dal quarzo X1, in parallelo al quale troviamo la resistenza R19; i condensatori C5 e C6 completano l'oscillatore, che funziona a 16 MHz, frequenza più che sufficiente a gestire l'intero ciclo di lavoro della centralina.

REALIZZAZIONE PRATICA

Passiamo adesso alla costruzione della scheda, che richiede la preparazione (o l'acquisto, visto che il materiale è tutto disponibile presso Futura Elettronica) di un circuito stampato a doppia faccia le cui tracce sono scaricabili dal nostro sito web insieme agli altri file del progetto; incisa e forata la basetta, realizzate le vie (ossia le interconnessioni tra piazzole comuni alle due facce) si possono montare i componenti iniziando da quelli a più basso profilo (resistenze e diodo D1) e proseguendo con lo zoccolo del microcontrollore, il pin-strip a 6 poli con terminali a 90° per il convertitore Seriale/USB (facol-

Guarda Presepino in funzione su YouTube!

Vuoi vedere il nostro presepe 3D realizzato con la stampante 3Drag e illuminato da Presepino? Nulla di più semplice: collegati alla pagina *https://youtu.be/X0Ivpg3FLGg*. Dal nostro canale YouTube *https://www.youtube.com/user/ElettronicaIN/videos* potrai anche vedere i video di moltissimi altri nostri progetti proposti in passato.

tativo), i trimmer, i pulsanti da c.s. e i LED, da orientare come indicato nel piano di montaggio, rammentando che il catodo è l'elettrodo situato dal lato smussato del contenitore. Ora inserite e saldate i due pin-strip femmina a 8 poli ciascuno per montare il modulo audio, il pin-strip per l'ICSP (4 poli), le otto morsettiere e il regolatore 7805, che va tenuto in piedi con il lato delle scritte rivolto allo zoccolo del microcontrollore. Ora bisogna montare i sei MOSFET, ognuno dei quali va inserito nei rispettivi fori dello stampato dopo averne piegato ad angolo retto i terminali ed averlo appoggiato (con il lato scritte verso l'alto) a un dissipatore di calore ad "U" tipo ML26 (22° C/W di resistenza termica). Completate le saldature e verificato che ogni componente sia montato correttamente e nel verso previsto, inserite il modulo DFR0299 nei propri pin-strip (orientandolo in modo che il porta-SD sia rivolto all'esterno del circuito) e il microcontrollore, la cui tacca di riferimento deve essere rivolta al regolatore integrato U2. Nel DFR0299 bisogna inserire una microSD della capacità desiderata, con già caricati (fatelo su un computer) i file audio desiderati; per essi vale quanto detto qualche paragrafo indietro, rammentando che per ovvi motivi sarebbe meglio che la durata di ciascuno non ecceda quella impostata per la fase che deve accompagnare, quindi, ad esempio, il motivo suonato per il giorno deve durare non più di quanto impostato per il giorno. Se il file dura meno in ogni caso non è un problema, in quanto il brano viene riprodotto ciclicamente fino alla fine del tempo impostato per la relativa fase e quindi sfumato per tempo.
Ricordate che i brani vanno preferibilmente inseriti nella root della microSD, ma il modulo è comunque in grado di cercare nelle cartelle, fermo restando che l'ordine di riproduzione, se nella scheda esistono altri file audio, non sarebbe più garantito.
A questo punto la centralina può o meno richiedere la programmazione in base a come l'avete realizzata: se avete acquistato un microcontrollore vergine dovete, con il programmatore ATMEL o una scheda da Arduino impostata come programmatore (tecnica di programmazione stand-alone), caricare il bootloader utilizzando, allo scopo, la connessione ICSP. Fatto ciò, rimuovendo il programmatore e montando il modulo d'interfaccia Seriale/USB nel connettore U4 (si tratta dell'FTDI5V, un modulo commercializzato dalla Futura Elettronica, *www.futurashop.it*, basato sull'integrato FT232RL della FTDI) si collega la presa USB di quest'ultimo a quella di un PC nel quale sia installato l'IDE di Arduino e si carica lo sketch (l'IDE vede la nostra scheda come una Arduino UNO, perché il bootloader caricato nell'ATmega 328P è quello).
Tutto ciò non è richiesto se si acquista il kit di montaggio, dove il microcontrollore viene fornito programmato con le funzionalità base (4 uscite gestibili da 4 trimmer). Comunque per rendere il progetto facilmente adattabile alle vostre esigenze diamo lo sketch, che potete scaricare dal nostro sito, insieme ai file del progetto. Anche se avete acquistato il kit ma non vi accontentate della configurazione base e volete provare ad apportare qualche modifica, dovete procurarvi e montare il convertitore Seriale/USB.
Bene, a questo punto non ci resta che augurarvi buon divertimento e soprattutto Buone Feste!

per il MATERIALE

Tutto il materiale di questo progetto può essere acquistato presso Futura Elettronica. La scheda presepe con Arduino (cod. FT1230K) viene venduta già programmata a Euro 39,00; il convertitore USB/TTL (cod. FTDI5V) è disponibile a Euro 19,00, mentre il mini riproduttore MP3 (cod. DFR0299) costa Euro 12,00. Tutti i prezzi si intendono IVA compresa.

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

Sfoglia on-line

FUTURA ELETTRONICA®

il catalogo generale 2018

O DIRETTAMENTE
DAL TUO TABLET
O SMARTPHONE

APP SCARICABILE
GRATUITAMENTE DA:

Available on the App Store

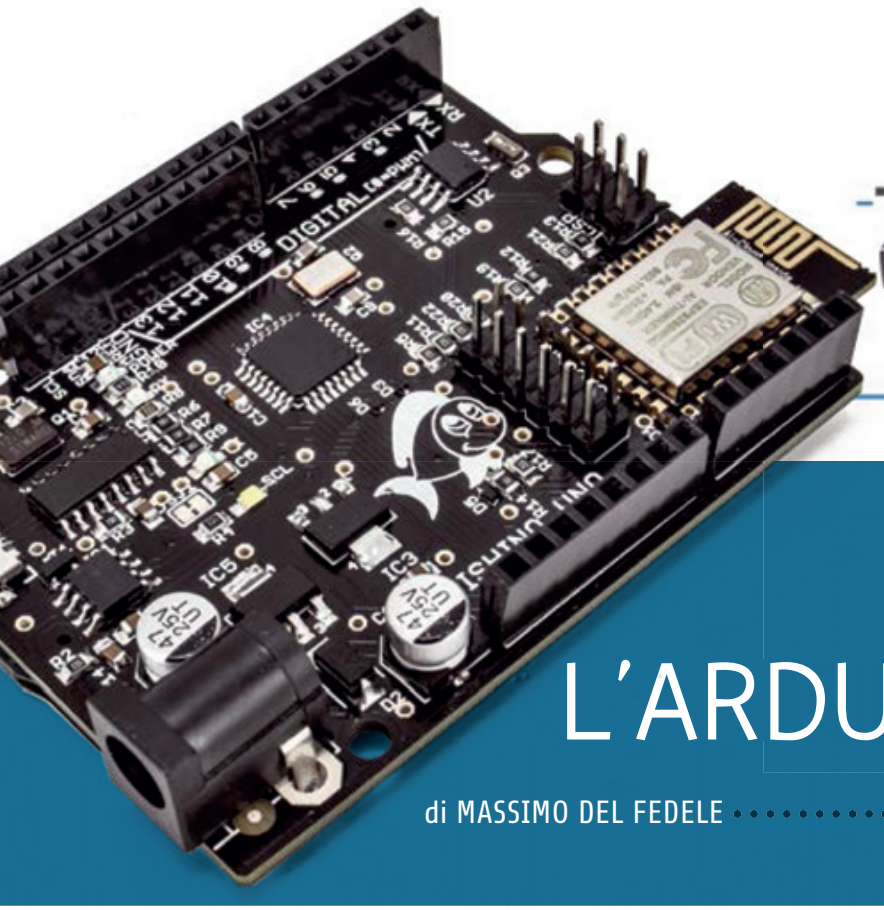
ANDROID APP ON Google play

FUTURA ELETTRONICA®

Futura Group srl
Via Adige, 11 • 21013 Gallarate (VA)
Tel. 0331/799775

Maggiori dettagli e acquisti on-line su
www.futurashop.it

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.**

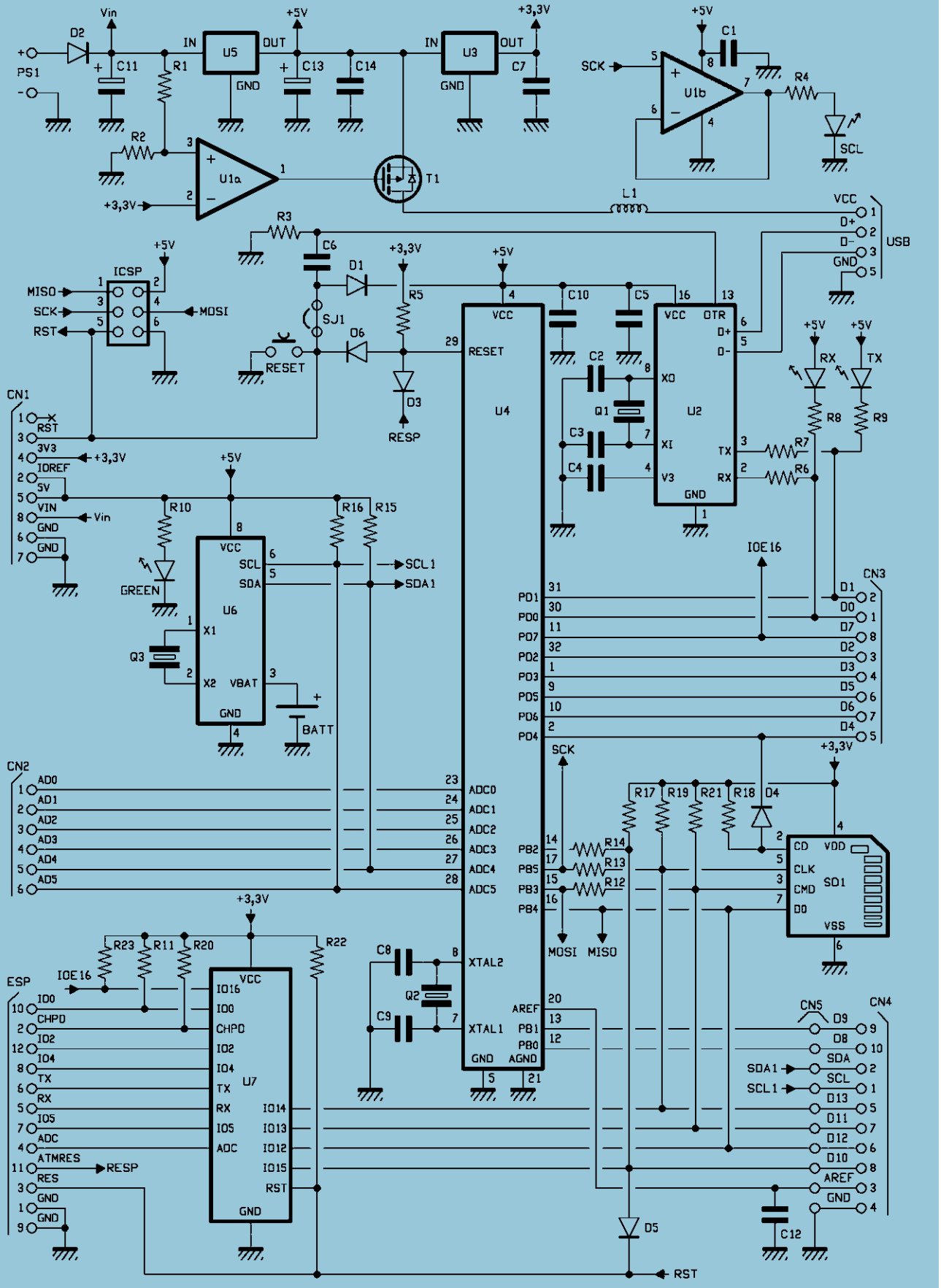
Tra le realizzazioni possibili ci sono, per esempio:

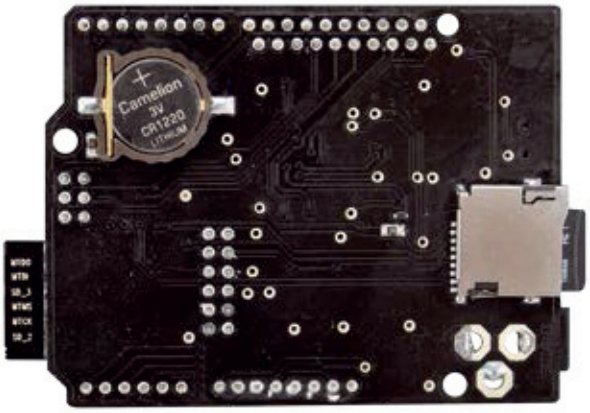
- sistemi di home automation gestibili via Internet tramite un Web Browser;
- data logger portatili in grado di connettersi e scaricare i dati sulla rete quando si entra nel campo di copertura di una rete WiFi;
- robot controllabili via rete e in grado di trasmettere tramite essa i dati rilevati da sensori.

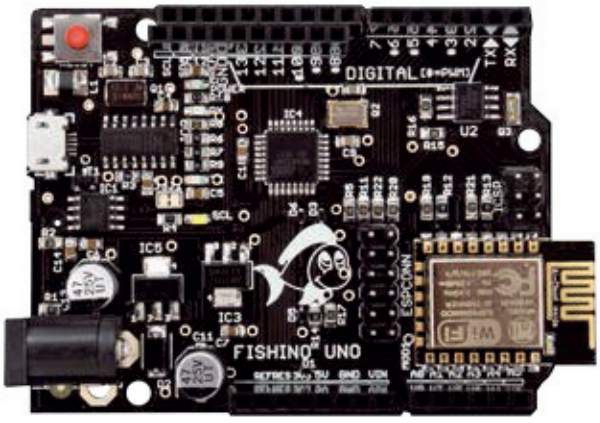
L'utilizzo di un modulo WiFi a basso costo ma con firmware da noi "hackerato" per ottenere prestazioni elevate e capace di funzionare anche come access point, ovvero senza la necessità di una struttura di rete WiFi esistente, permette il controllo via cellulare in qualsiasi momento, anche in assenza 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÷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 costituito 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÷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÷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.


PS1
D2
Vin
C11
R1
IN
U5
OUT
GND
+5V
C13
C14
IN
U3
OUT
GND
+3,3V
C7
R2
U1a
+3,3V
T1
L1
SCK
U1b
C1
R4
SCL
VCC
D+
D-
GND
USB
ICSP
MISO
SCK
RST
MOSI
R3
C6
D1
SJ1
RESET
D6
R5
D3
RESP
VCC
RESET
U4
C10
C5
VCC
OTR
XO
XI
V3
GND
U2
C2
Q1
C3
C4
RX
TX
R8
R9
R7
R6
CN1
RST
3V3
IOREF
5V
VIN
GND
GND
Vin
R10
GREEN
R16
R15
VCC
SCL
SDA
U6
X1
X2
VBAT
GND
Q3
BATT
SCL1
SDA1
IOE16
CN3
PD1
PD0
PD7
PD2
PD3
PD5
PD6
PD4
D1
D0
D7
D2
D3
D5
D6
D4
CN2
AD0
AD1
AD2
AD3
AD4
AD5
ADC0
ADC1
ADC2
ADC3
ADC4
ADC5
SCK
MOSI
MISO
R17
R19
R21
R18
D4
R14
R13
R12
PB2
PB5
PB3
PB4
CD
VDD
CLK
CMD
DO
VSS
SD1
ESP
IOE16
IO0
CHPD
IO2
IO4
TX
RX
IO5
ADC
ATMRES
RES
RESP
GND
GND
R23
R11
R20
R22
VCC
IO16
IO0
CHPD
IO2
IO4
TX
RX
IO5
ADC
RES
GND
U7
IO14
IO13
IO12
IO15
RST
C8
Q2
C9
XTAL2
XTAL1
GND
AREF
PB1
PB0
AGND
CN5
CN4
D9
D8
SDA
SCL
D13
D11
D12
D10
AREF
GND
SDA1
SCL1
D5
C12
RST





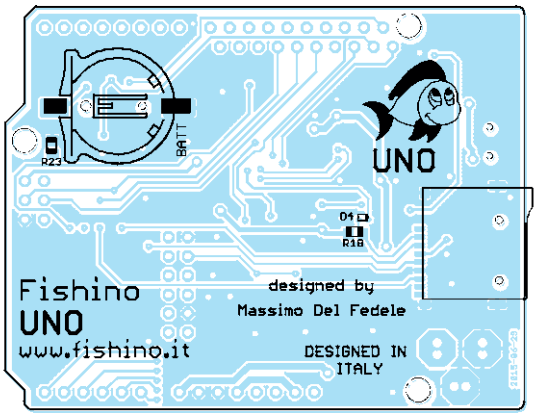
Elenco Componenti:

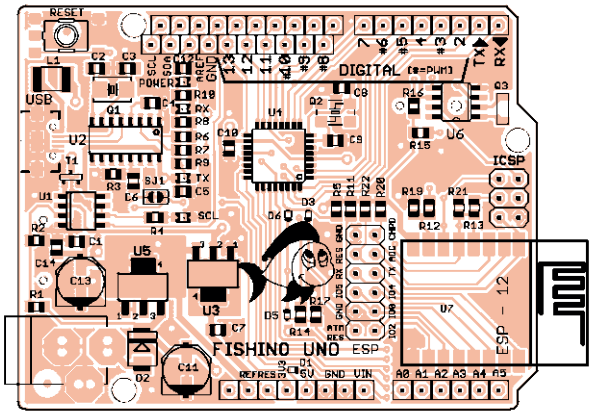
R1: 10 kohm (0805)
R2: 10 kohm (0805)
R3: 1 kohm (0805)
R4: 1 kohm (0805)
R5: 10 kohm (0805)
R6: 1 kohm (0805)
R7: 1 kohm (0805)
R8: 2,4 kohm (0805)
R9: 1 kohm (0805)
R10: 470 ohm (0805)
R11: 10 kohm (0805)
R12: 1 kohm (0805)
R13: 1 kohm (0805)
R14: 1 kohm (0805)
R15: 10 kohm (0805)
R16: 10 kohm (0805)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

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, correda-

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

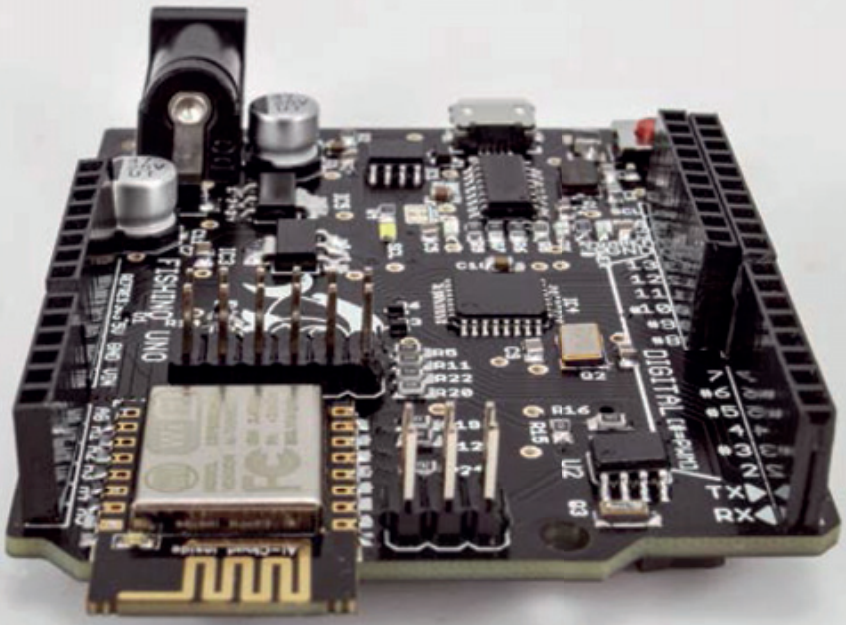




to dall'usuale quarzo a 16 MHz, dai due condensatori sul circuito oscillante e da un certo numero di condensatori di disaccoppiamento sulle linee di alimentazione. Una piccola parentesi su questi ultimi: spesso negli schemi si vedono alcuni (a volte molti) condensatori in parallelo all'alimentazione; 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:

Vo = 5 Volt • 3.3K / (1K + 3.3K) = 3,83 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 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 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 funzionamento, 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 precaricato 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 riusciti 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

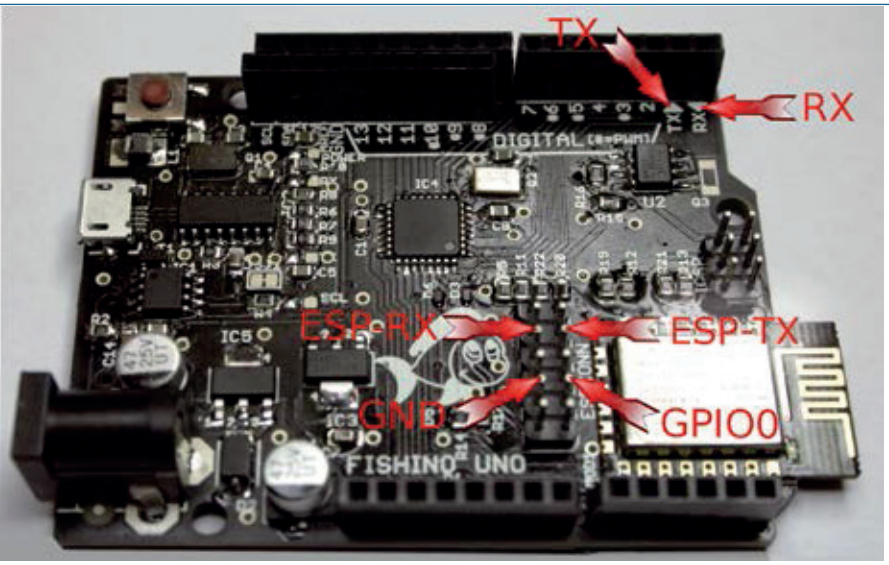


Fig. 1

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 ininfluente). 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 scartare 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 pur 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 µF 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) raggiunge 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

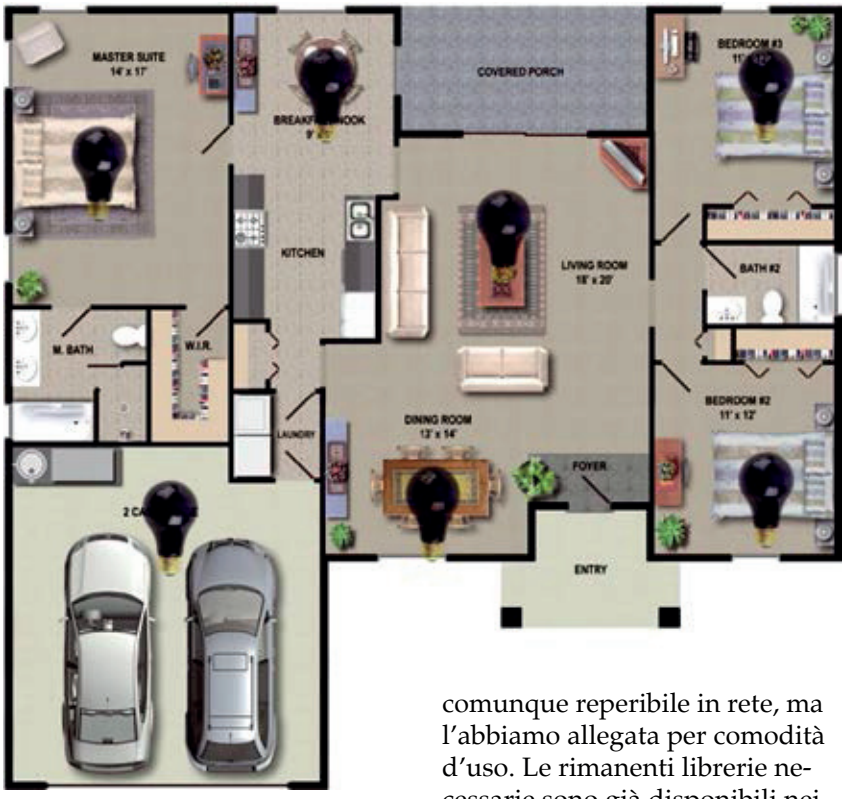
Per il converter USB/seriale Fishino richiede driver specifici, almeno per Windows fino alla versione 7 (si scaricano dal sito *www.fishino.it* o *www.fishino.com*, nella sezione Download); pare che dalla 8 in poi i driver siano inclusi. Per l'ambiente Linux, per contro, i driver non sono necessari, 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 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. 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

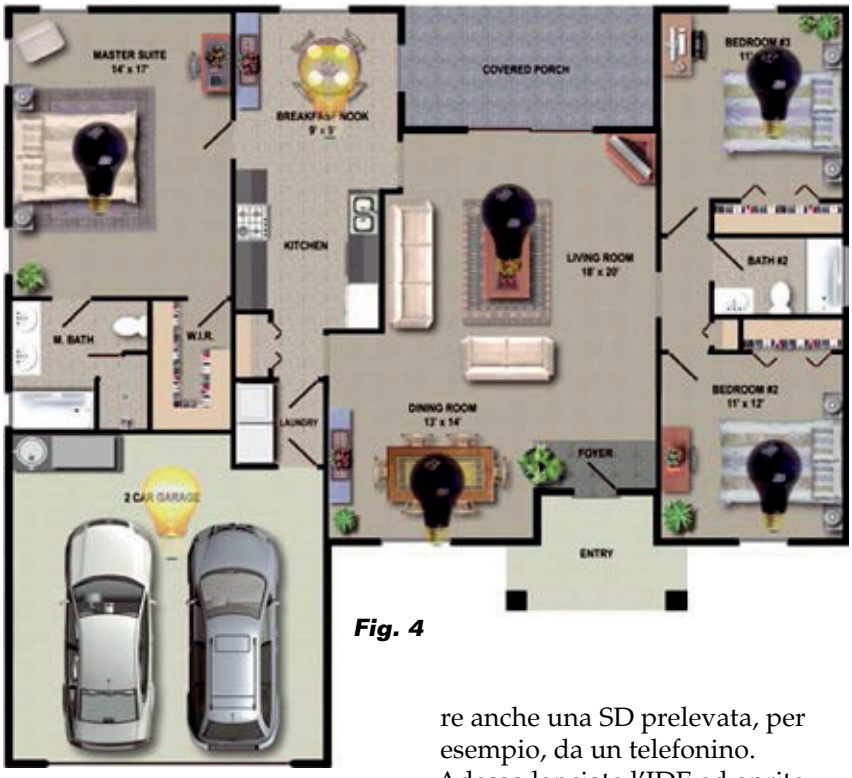


Fig. 4

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 utilizzare 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 indirizzi 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. Suggeriamo l'impostazione di un IP statico per le prime prove.

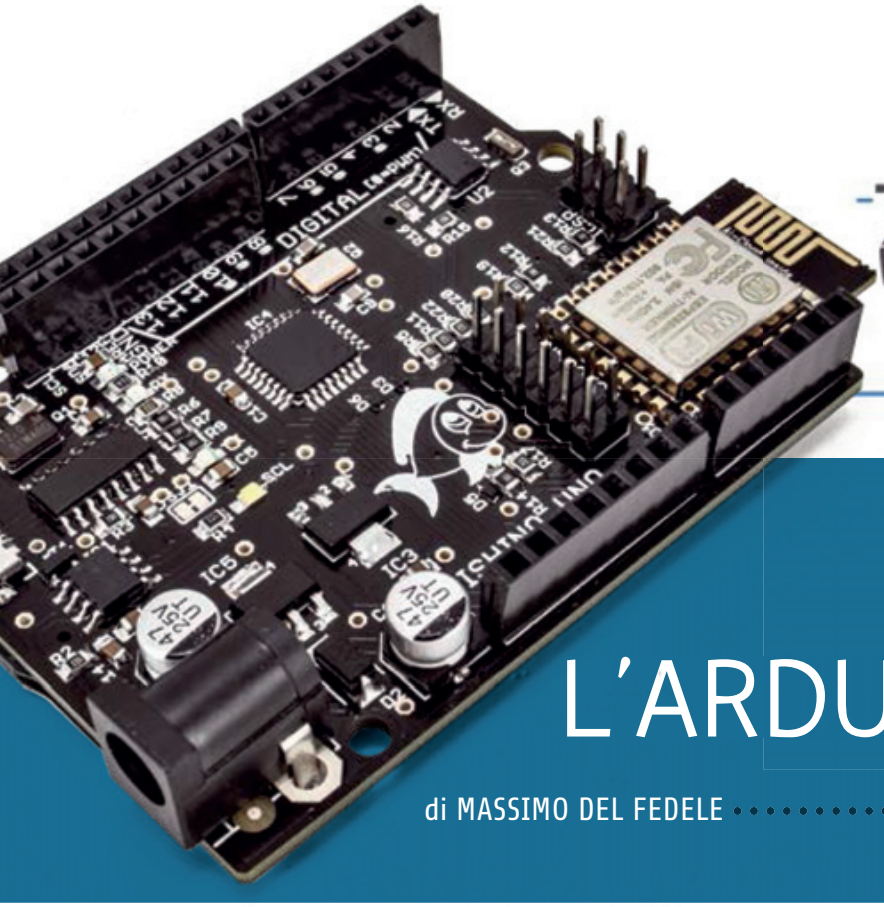
Se tutto è andato a buon fine, sul browser verrà visualizzata la pagina con la **Fig. 3**. La pagina è totalmente configurabile tramite i file presenti nella cartella STATIC e poi copiati sulla SD card. Cliccando su una delle lampadine spente (nere) l'immagine cambierà in una lampadina accesa (gialla) 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.

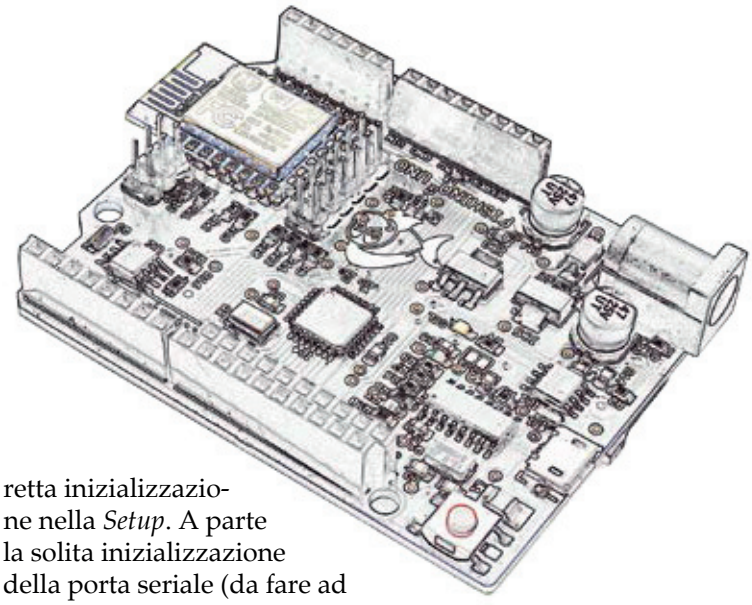
LIBRERIA 'FISHINO'

Partiamo con la descrizione della libreria (che potete scaricare dal sito della rivista *www.elettronicain.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 corretta 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.

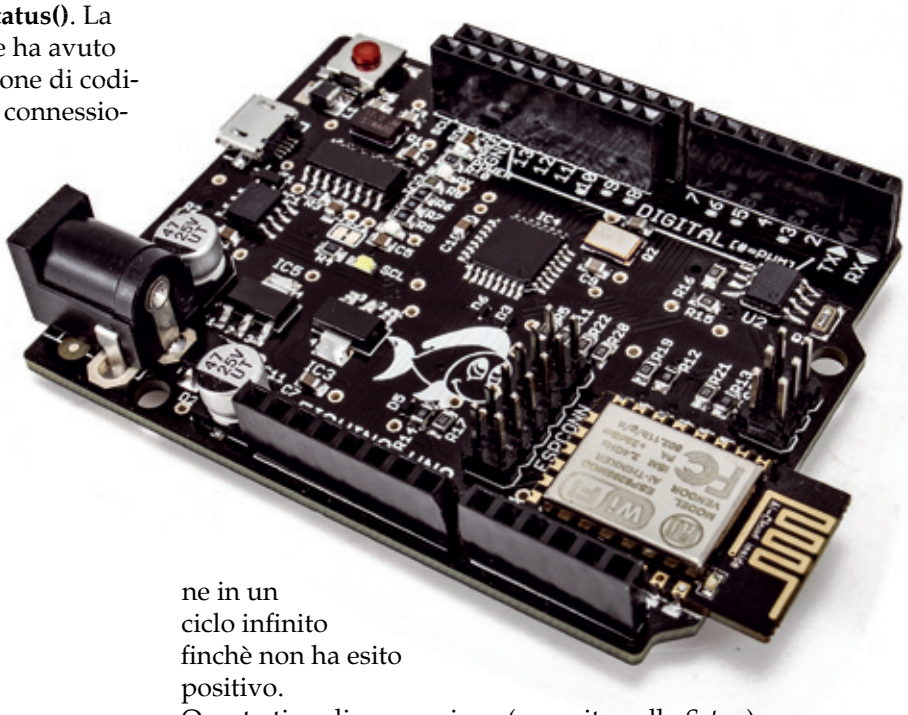
Per eseguire la connessione all'access point/router si utilizza **bool Fishino.begin(SSID, PASSWORD)**, dove al posto di SSID va inserito il punto di accesso e al posto di PASSWORD la chiave per accedervi (quest'ultima può essere una stringa vuota se non è richiesta).
Per controllare se la board Fishino è correttamente connessa il comando è **uin8_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 connessione in un ciclo infinito finchè 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 4

```
uint32_t connecTime;
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));
```



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 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, IPAddress 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, IPAddress 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");
```

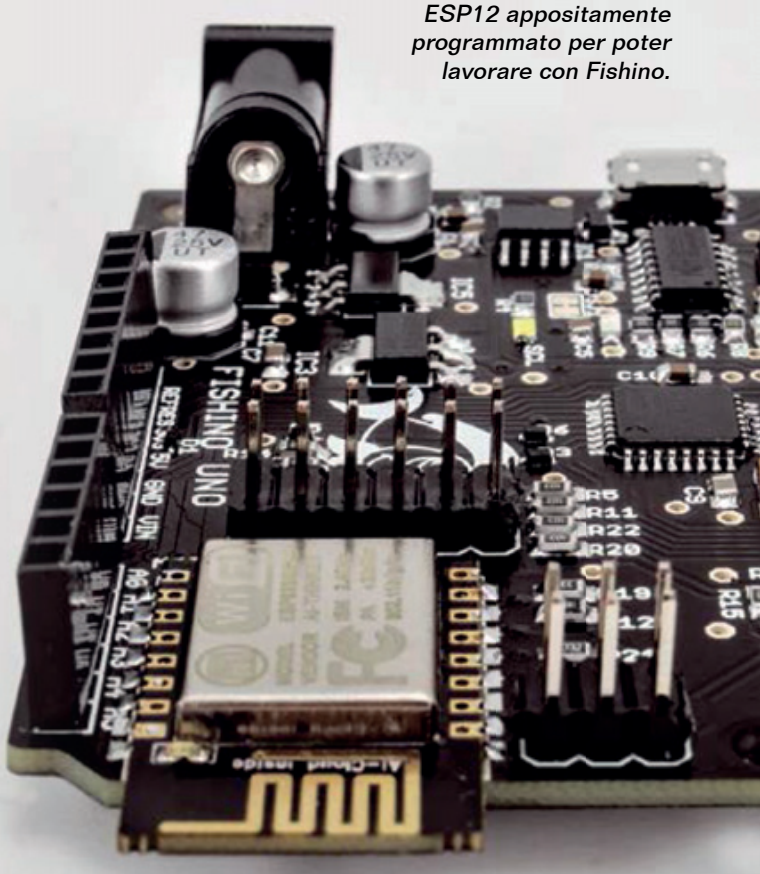
WiFi in un comando singolo (**Listato 7**).
Vedremo come usarle a fine articolo con un esempio completo. Per poter leggere i dati della connessione WiFi, quali lo SSID del router a cui ci si è connessi, il MAC del medesimo (BSSID), la potenza in dBm del segnale (RSSI) ed il tipo di protezione della rete, 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 networkItem)** 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.

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");
 }
}
```

CLASSI FISHINOCLIENT 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**.



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

Listato 11

```
#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 modalitè 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 possibiltà 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 richiesa http è terminata
                    // quindi è possibile inviera 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 disonnected");
  }
}
```

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 liberare 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

VUOI SVILUPPARE LE TUE APPLICAZIONI CON ARDUINO?

Da noi trovi tutto quello che ti serve!

l'ABC di ARDUINO

COD. ABCARDU

€ 9,90

Arduino e le tecniche di programmazione dei microcontrollori Atmel

COD. ATPROMA

€ 15,00

ARDUINO UNO Programmazione avanzata e librerie di sistema

COD. ARDUADVANCED

€ 22,00

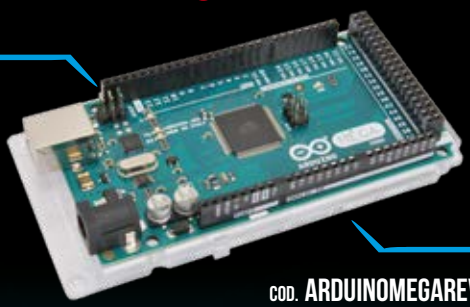
Prezzi IVA inclusa

Arduino Uno Rev3

COD. ARDUINOUNOREV3

€ 24,50

Arduino Mega



COD. ARDUINOMEGAREV3

€ 43,00

Arduino Due

ARDUINODUE

€ 44,00

kit V5 con Arduino Uno REV3

Set contenente tutti i componenti necessari per realizzare gli esperimenti descritti nel libro "L'ABC di Arduino".

COD. ARDUKITV5

€ 59,00

Starter kit per robot con Arduino Uno

Set contenente le parti necessarie per realizzare il Robot descritto nel libro incluso.

€ 155,00

COD. ARDUKITV4

FUTURA ELETTRONICA®

www.futurashop.it

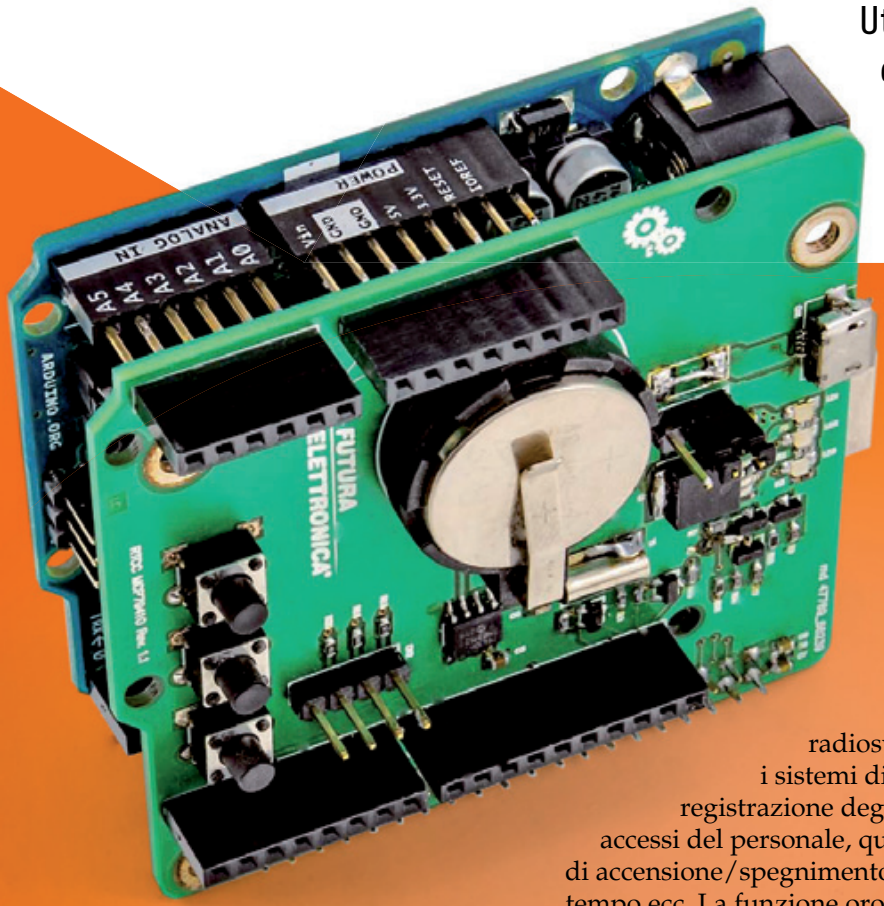
Futura Group srl • via Adige, 11 • 21013 Gallarate (VA)
Tel. 0331/799775 • Fax. 0331/792287

Caratteristiche tecniche di questi prodotti e acquisti on-line su *www.futurashop.it*

RTC SHIELD PER ARDUINO E RASPBERRY PI

di MATTEO DESTRO

Utilizziamo un nuovo Real Time Clock di Microchip rendendolo disponibile su uno shield per le due schede di prototipazione. Prima puntata.

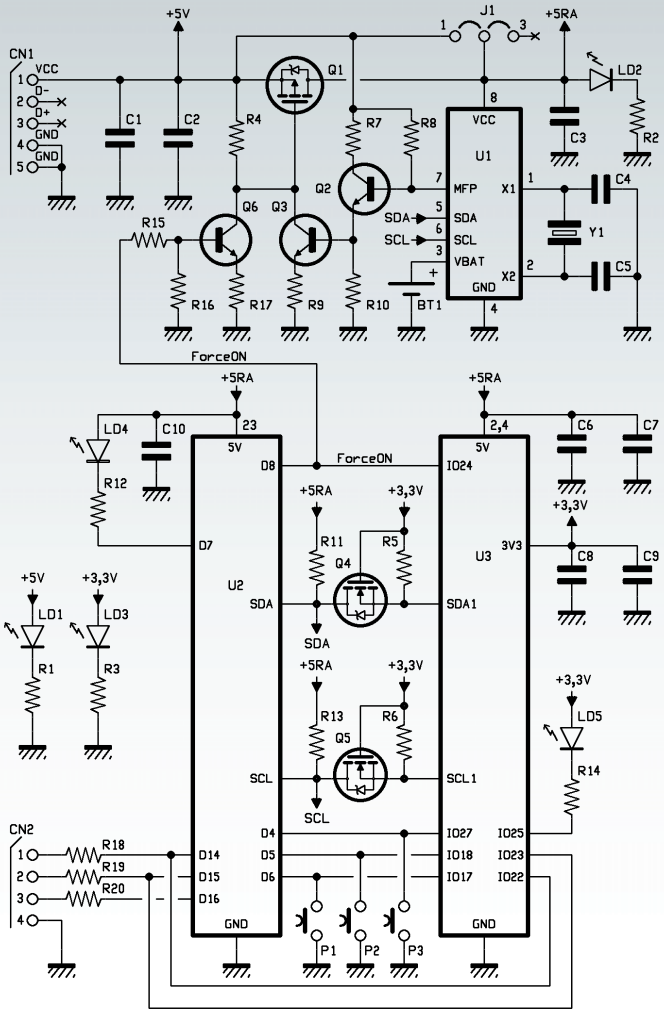


Molte applicazioni richiedono l'informazione oraria, che viene ottenuta localmente mediante circuiti chiamati RTC (Real Time Clock) o RTCC (Real Time Clock Calendar); tra esse le radiosveglie, i sistemi di registrazione degli accessi del personale, quelli di accensione/spegnimento a tempo ecc. La funzione orologio può essere implementata in maniera semplice grazie a circuiti integrati come il DS1307, da noi utilizzato più volte, o come il recente MCP79410 della Microchip; con questo abbiamo realizzato uno shield polivalente applicabile alle schede Arduino e Raspberry Pi. Cominciamo con il descrivere l'integrato Microchip e la libreria sviluppata per l'ecosistema Arduino. Nella seconda puntata vedremo l'impiego con la Raspberry Pi.

L'INTEGRATO MCP79410

L'RTCC appartiene a una famiglia di integrati che comprende anche MCP79411 e MCP79412; questi ultimi due hanno memorizzato un MAC address univoco per le applicazioni ethernet. Più esattamente, l'MCP79411 ha un MAC address in formato EUI-48 mentre l'MCP79412 ce l'ha in formato EUI-64. La dicitura EUI sta per *Extended Unique Identifier* e può essere utilizzata per assegnare un indirizzo hardware a 48 o 64 bit univoco nelle applicazioni di networking. Solitamente si utilizzano i 24 bit più significativi per identificare

[schema **ELETTRICO**]



- configurazione di ore, minuti e secondi sia nel formato 24h che nel 12h (AM/PM);
- configurazione di giorno, mese, anno e giorno della settimana;
- gestione automatica degli anni bisestili;
- oscillatore a 32.768 Hz;
- calibrazione/regolazione interna digitale con risoluzione di ±1ppm (range massimo ±129ppm);
- due allarmi programmabili;
- TimeStamp sia su power-up che su power-down;
- 64 Byte di memoria SRAM tamponata;
- 128 Byte di memoria EEPROM con possibilità di scrittura paginata a 8 byte alla volta;
- 8 Byte di memoria EEPROM protetta, per scrivere nella quale (si può un solo byte per volta) bisogna eseguire una sequenza di sblocco;
- interfaccia di comunicazione I²C fino a 400 kHz.

Inoltre l'integrato ha pin di uscita open-drain multifunzione che descriveremo più avanti.
La **Fig. 1** mostra lo schema a blocchi interno dell'integrato MCP79410 dove sono ben visibili i blocchi logici che vanno a formare il dispositivo.
L'interfaccia I²C prevede due indirizzi hardware, uno per la gestione del RTCC e l'altro per la gestione della memoria EEPROM. La **Fig. 2** mostra i due possibili control byte contenenti l'indirizzo hardware per la sezione RTCC o EEPROM. Il primo ha come valore ***0b1101111x*** dove al posto della "***x***" ci sarà "***0***" o "***1***" a seconda dell'operazione che si vuole eseguire ("***0***" - scrittura; "***1***" - lettura). Il secondo invece sarà ***0b1010111x***, dove "***x***" assume i valori descritti sopra.

in modo univoco un fornitore, produttore o organizzazione a livello globale: tale codice viene detto OUI (*Organizationally Unique Identifier*). L'acquirente di tale codice -di norma un costruttore di dispositivi elettronici- dispone dei rimanenti 24 Bit per completare il codice da assegnare ai propri prodotti (3 byte, per un totale di oltre 16 milioni di codici univoci possibili). Per fare un esempio, attualmente Microchip dispone di 3 codici OUI, che sono rispettivamente *0xD88039*, *0x001EC0* e *0x0004A3*. Con questi, Microchip può dare origine a ben 50 milioni di possibili indirizzi MAC univoci. Quindi acquistando un RTCC della serie MCP79411 o MCP79412 troverete un indirizzo MAC univoco utilizzabile senza dover pagare alcuna royalty. Non dovendo sviluppare un'applicazione di networking, abbiamo utilizzato l'MCP79410, che mette a disposizione:

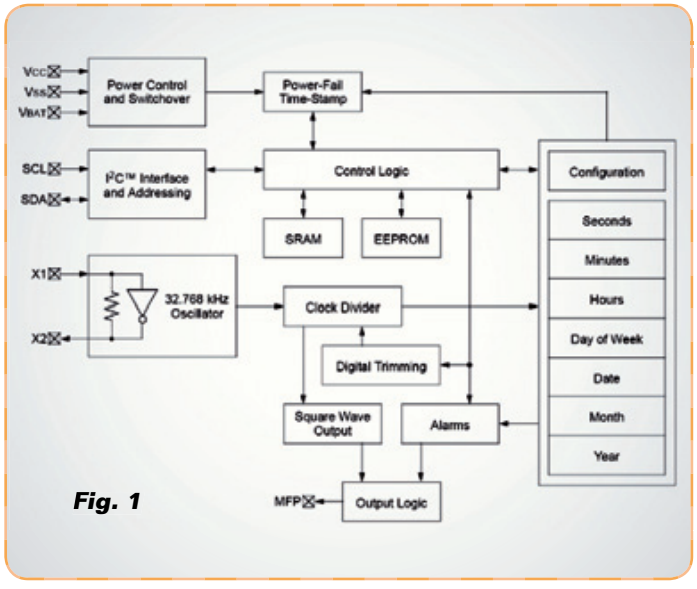
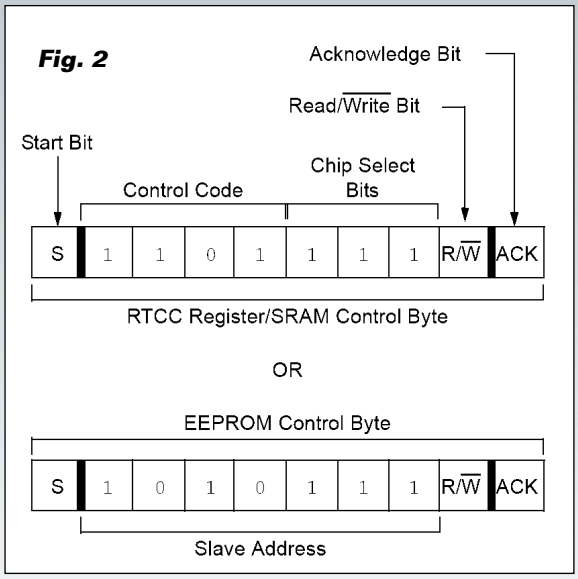


Fig. 1



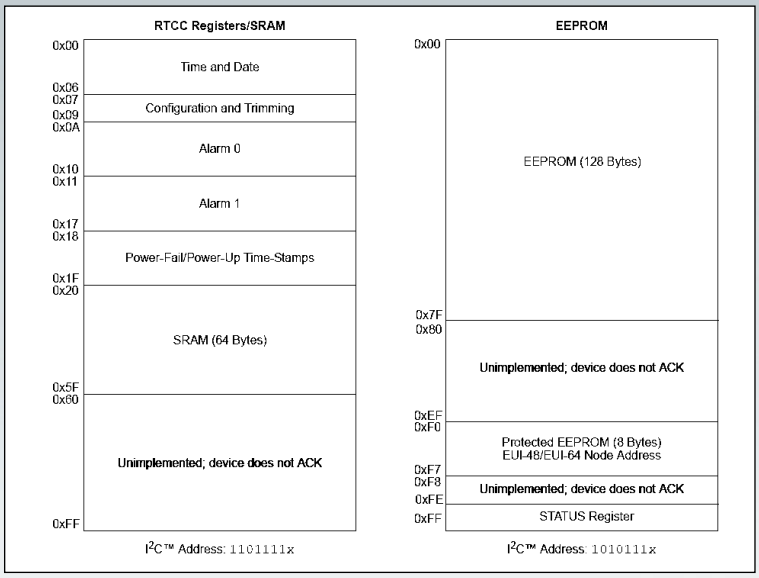
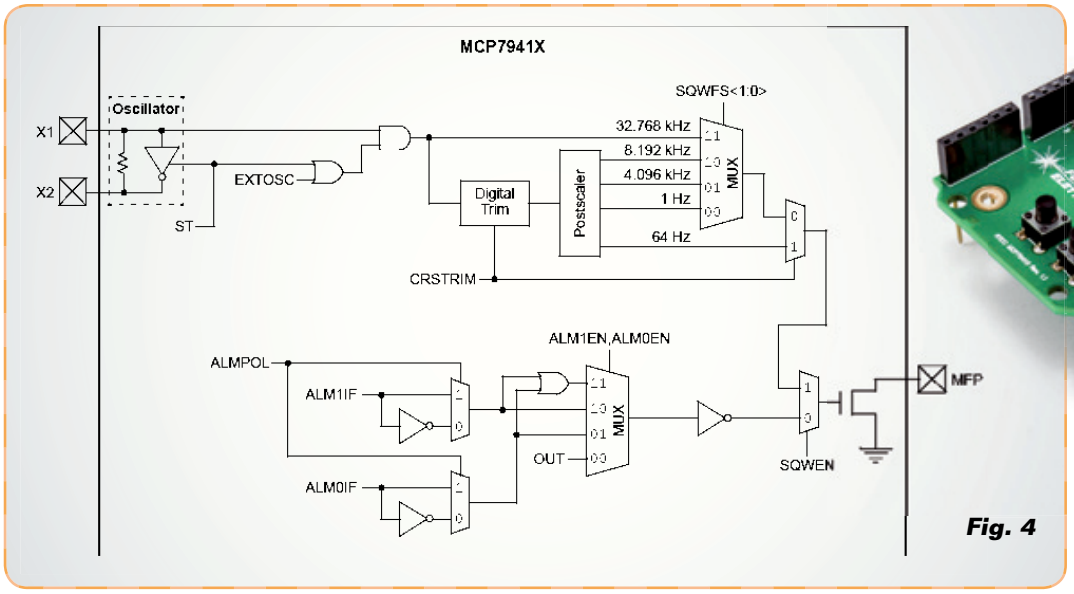


Fig. 3

La mappatura dei registri e delle memorie è riportata nella **Fig. 3**: gli indirizzi da ***0x00*** a ***0x1F*** contengono i registri di configurazione della data e dell'ora, nonché i registri per configurare gli allarmi 0 e 1. Concludono la sezione, i registri di gestione del TimeStamp al power-up e al power-down. Dall'indirizzo ***0x20*** all'indirizzo ***0x5F*** viene mappata la memoria SRAM tamponata. I restanti indirizzi fino a ***0xFF*** non sono utilizzati.
Quanto appena detto fa riferimento all'indirizzo hardware ***0b1101111x***. La EEPROM è invece mappata a partire dall'indirizzo ***0x00*** a ***0x7F***, ai quali seguono una serie di indirizzi non utilizzati (Da ***0x80*** a ***0xEF***); dall'indirizzo ***0xF0*** fino a ***0xF7*** troviamo la memoria EEPROM protetta, la quale può essere programmata solo un byte alla volta e solo dopo avere eseguito una sequenza di sblocco. Per ultimo, all'indirizzo ***0xFF*** trova posto il registro STATUS. Quanto appena detto fa riferimento all'indirizzo hardware ***0b1010111x.***

SCHEMA ELETTRICO

Con l'integrato MCP79410 abbiamo realizzato uno shield applicabile sia a una scheda Arduino Uno R3 che a una Raspberry Pi 2.0/3.0/B+. L'elettronica e un'opportuna programmazione dell'Arduino piuttosto che della Raspberry Pi, permettono di configurare in toto l'MCP79410, soprattutto nelle sue funzionalità di gestione allarmi necessarie all'accensione e allo spegnimento automatico della scheda collegata allo shield. Tutte le funzioni necessarie a gestire l'integrato sono state raccolte in librerie sia per Arduino che per Raspberry Pi.
Iniziamo lo studio dello schema elettrico partendo dal connettore CN1 (microUSB) il quale porta l'alimentazione stabilizzata +5Vcc, al MOSFET Q1 (a canale P) utilizzato come interruttore hardware. Il Q1 è in interdizione fintantoché il gate non viene cortocircuitato a massa, allorché la V_{GS} supera il valore di soglia aprendo il canale del MOSFET e quindi trasferendo l'alimentazione dal source al drain. La "*nuova*" alimentazione viene chiamata +5VRA. Per portare a massa il gate Q1 ci sono due possibilità: l'MCP79410, attraverso l'uscita open-collector "***MFP***" agisce sui due transistor Q2 e Q3; la linea di comando "***ForceOn***" polarizza il transistor Q6 che porta a massa il gate di Q1.
Torniamo sulla linea di uscita **MFP**, di cui la **Fig. 4** mostra la logica interna all'MCP79410: essa può essere configurata per lavorare come uscita generica, per fornire uno stato logico al verificarsi dei due allarmi configurabili, nonché come riporto della frequenza di clock. Se si vuole che l'uscita riporti la frequenza di clock dell'oscillatore o una delle frequenze derivate, si deve settare a "***1***" logico il bit "***SQWEN***" del registro "***CONTROL***". Per decidere quale frequenza debba essere riportata sull'uscita si devono configurare i bit "***SQWFS0***" e "***SQWFS1***" sempre del registro "***CONTROL***. Le frequenze possibili sono 1, 4.096, 8.192 e 32.768 Hz (consultate il data-sheet per i dettagli).
Volendo che MFP diventi un'uscita generica a livello logico si deve settare il bit "***SQWEN***" a zero e disabilitare entrambi gli allarmi. Il nostro caso, invece, prevede che MFP sia pilotata dagli allarmi 0 e 1, quindi "***SQWEN***" deve essere settato a zero e i bit "***ALM0EN***" e "***ALM1EN***" devono assumere entrambi, o almeno uno di essi, un valore logico alto. La MFP, essendo un'uscita open-collector, può essere intesa come linea di interrupt indicante che uno o entrambi gli allarmi sono scattati in seguito al raggiungimento delle condizioni impostate.



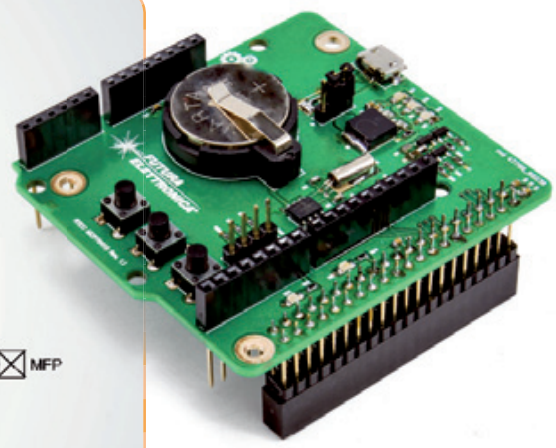


Fig. 4

Vedremo più avanti come configurare e sfruttare i due allarmi messi a disposizione. Per concludere, la MFP può essere portata a un microcontrollore per indicargli che si è attivato uno o entrambi gli allarmi configurati evitando di scrivere del codice di polling dello stato dei due allarmi. Della MFP si può inoltre definire se è attiva a livello 0 o 1; allo scopo si deve agire sul bit "***ALMPOL***" nei registri "***ALM0WKDAY***" e "***ALM1WKDAY***". Nella nostra applicazione l'uscita dev'essere attiva a livello alto. La **Tabella 1** mostra il comportamento della MFP in base alla configurazione del bit ALMPOL e dello stato del flag di interrupt dell'allarme *x*: quando ALMPOL è settato a "***1***" l'uscita MFP segue lo stato del flag di interrupt dell'allarme *n*, nel caso di un solo allarme attivo; se entrambi gli allarmi sono attivi e configurati, la linea MFP" segue l'andamento dell'OR logico dei due flag di interrupt degli allarmi 0 e 1. Se invece il bit ALMPOL è settato a 0 la MPF è il negato del flag di interrupt dell'allarme *n* impostato (un solo allarme impostato) oppure segue l'andamento del NAND logico dei due flag di interrupt degli allarmi 0 e 1. L'integrato MCP79410 viene alimentato tramite la tensione +5VAR, la quale viene anche portata alla scheda Arduino Uno R3 e alla scheda Raspberry Pi. L'MCP79410 ha anche un ingresso V_{BAT} al quale si connette una batteria tampone che lo tiene alimentato in caso di mancanza di alimentazione. Durante il funzionamento a batteria, solo una parte dell'integrato rimane attiva: in particolare, il blocco RTCC (gestione data e ora) e i 64 Byte della SRAM. Inoltre rimane attiva la linea MFP se configurata per lavorare in abbinamento agli allarmi 0 e 1, altrimenti sarà disabilitata.
Le funzioni di gestione della data e dell'ora richiedono una frequenza di clock di 32.768 Hz che viene ricavata collegando un quarzo ai pin X1 e X2. Infine l'interfaccia di comunicazione I^2C viene portata sia ad Arduino che a Raspberry Pi tramite le rispettive connessioni; notate che il bus I^2C richiede un traslatore di livello per l'interconnessione alla Raspberry Pi, in quanto le sue linee di I/O lavorano con un livello logico +3V3. Ciò viene ottenuto sfruttando i MOSFET Q4 e Q5 che funzionano come in **Fig. 5**: il traslatore di livello bidirezionale siffatto vale anche per SPI o qualsiasi linea bidirezionale e in esso ogni sezione è alimentata con un livello di tensione differente; nel nostro caso +3V3 per l'elettronica della scheda Raspberry Pi +5V per l'integrato MCP79410. La sezione a bassa tensione, a sinistra, prevede due resistenze di pull-up collegate al source del MOSFET, per le linee SDA e SCL, mentre il gate viene collegato direttamente alla tensione di alimentazione più bassa: nel nostro caso +3V3. La sezione ad alta tensione prevede anch'essa due resistenze di pull-up collegate al drain del MOSFET. Il MOSFET utilizzato per entrambe le linee è uno a riempimento a canale N con il substrato internamente connesso al source (deve avere integrato il diodo che connette il substrato

Tabella 1

ALMPOL	ALMXIF	MFP
0	0	1
0	1	0
1	0	0
1	1	1

ALMPOL	ALM0IF	ALM1IF	MFP
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

con il drain creando una giunzione NP); quelli da noi usati sono BSS123. Vediamo in dettaglio i tre stati di funzionamento, con la premessa che quanto detto vale per entrambe le linee:

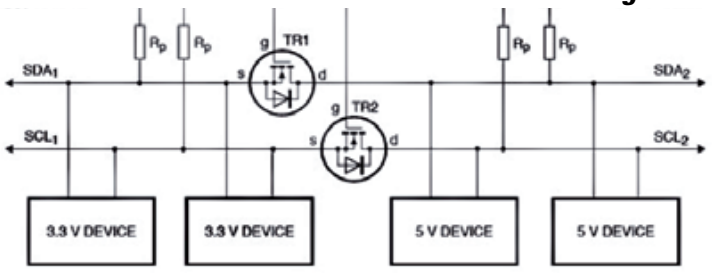
1) nessuna periferica impone un livello di tensione basso sulla linea, quindi sul lato a 3,3V la resistenza di pull-up impone l'1 logico e di conseguenza la V_{GS} non supera la soglia del MOSFET, in quanto V_G e Vs hanno la medesima tensione, quindi il MOSFET non entra in conduzione; sul lato a 5V la linea si ritrova a un livello logico alto imposto dalla rispettiva resistenza di pull-up;
2) la periferica a 3,3V impone sulla linea un livello di tensione basso, perciò il source del MOSFET va a livello logico basso; la V_{GS} supera la soglia e manda in conduzione il MOSFET; con il MOSFET in conduzione, il livello logico basso viene imposto anche nellla sezione a 5V;
3) la periferica a 5V alta tensione impone sulla linea un livello di tensione basso; sulla linea a bassa tensione abbiamo la resistenza di pull-up che impone un livello logico alto e quindi manda in conduzione il diodo presente nel MOSFET (la V_S scende e di conseguenza la V_{GS} supera la soglia mandando in conduzione il MOSFET, quindi a questo punto le due sezioni hanno il medesimo livello di tensione sulla linea).

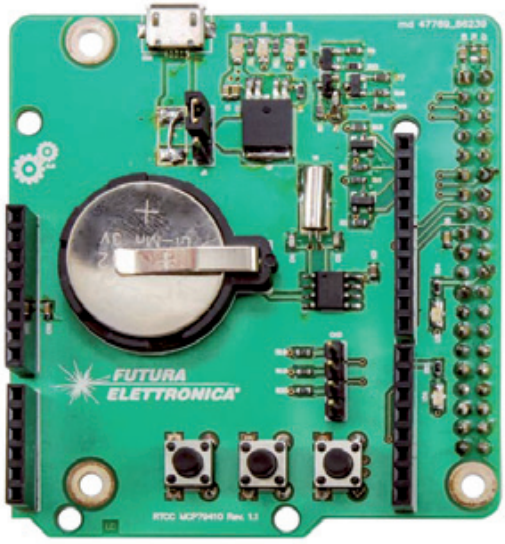
La massima frequenza ammessa sull'I²C è 400 kHz.
I pulsanti P1, P2 e P3 collegati sia ad Arduino che a Raspberry Pi sono utilizzati durante le fase di configurazione del RTCC e degli allarmi. Per agevolare le fasi di sviluppo sono stati predisposti tre segnali di trigger portati al connettore CN2 per Arduino e due per la Raspberry Pi, anch'essi portati al connettore CN2. I segnali di trigger sono in comune tra le due schede, in quanto non possono essere collegate contemporaneamente allo shield. Sempre dalle due schede, parte il segnale "***ForceOn***" che serve a mantenere in stato di ON l'interruttore elettronico Q1. Completano gli schemi elettrici i LED LD4 (collegato ad Arduino) e LD5 (collegato a Raspberry Pi) utilizzati durante la fase di configurazione dell'MCP79410. Infine LD1 indica la presenza dell'alimentazione principale in arrivo dal microUSB; LD2 indica invece che è presente l'alimentazione secondaria +5VAR. LD3 indica la presenza dell'alimentazione +3V3.
Un'ultima nota riguarda il jumper J1, che serve a bypassare Q1: serve durante la configurazione dell'MCP79410 soprattutto se si collega la Raspberry Pi. Arduino in realtà è già auto-alimentata dal connettore USB tipo B connesso al PC, per la programmazione tramite l'IDE Arduino.

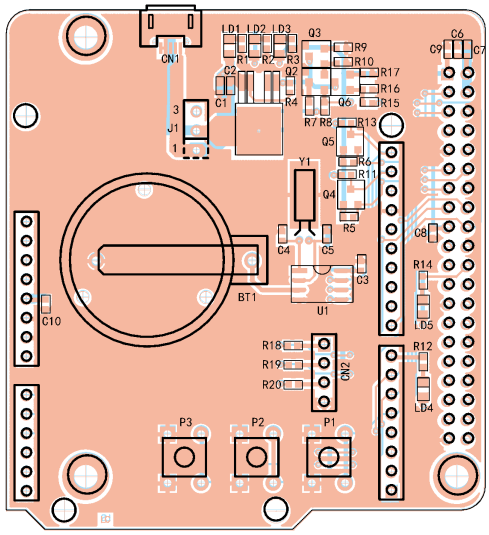
I REGISTRI DELL'MCP79410

Prima di passare alla libreria per Arduino Uno R3 conviene conoscere i registri mappati dell'MCP79410. La **Fig. 6** evidenzia la mappatura in memoria di ognuno e il significato di ogni singolo bit o gruppi di bit; è suddivisa in cinque sezioni, la prima delle quali contiene i registri per configurare il TimeKeeper, ovvero data e ora del RTCC. In questi registri sono presenti anche i bit di configurazione per l'attivazione dell'oscillatore e per il funzionamento a batteria dell'integrato. Trovano posto anche i registri "***CONTROL***" per attivare/disattivare gli allarmi, per configurare il pin MFP e il registro OSCTRIM per tarare la frequenza del clock, oltre al registro "***EEUNLOCK***" per sproteggere la memoria EEPROM per la scrittura di un singolo byte alla volta (per sproteggere la EEPROM occorre scrivere una serie di byte ben definiti nel registro).
La seconda sezione contiene i registri di configurazione dell'allarme zero, mentre la terza contiene i registri di configurazione dell'allarme uno. La quarta sezione contiene i registri per la lettura del power-down timestamp. La quinta contiene i registri per la lettura del power-up timestamp. Notate che le informazioni di data e ora sono gestite sempre in formato BCD, quindi, ad esempio, il registro "***RTCSEC***" contiene l'informazione dei secondi e sfrutta i bit da 0 a 3 per memorizzare le unità, mentre i bit da 4 a 6 vengono usati per le decine. Il bit più significativo serve per attivare l'oscillatore.
Quindi un ipotetico valore dei secondi di 39 viene suddiviso in 9 unità e 3 decine.
L'impostazione in formato BCD visibile nella **Tabella 2** viene usata in tutti i registri che contengono un'informazione di data e ora sia che siano del TimeKeeper sia degli allarmi che dei timestamp.

Fig. 5







Elenco Componenti:

R1÷R3: 1,2 kohm 1% (0603)
R4: 22 kohm 1% (0603)
R5÷R7: 2,2 kohm 1% (0603)
R8, R16: 10 kohm 1% (0603)
R9: 4,7 kohm 1% (0603)
R10, R12, R14: 680 ohm 1% (0603)
R11: 2,2 kohm 1% (0603)
R13: 2,2 kohm 1% (0603)
R15: 3,3 kohm 1% (0603)
R17: 4,7 kohm 1% (0603)
R18÷R20: 1 kohm 1% (0603)
C1, C2: 10 µF ceramico (0603)
C3: 100 nF ceramico (0603)
C4, C5: 10 pF ceramico (0603)
C6: -
C7÷C10: 10 µF ceramico (0603)
LD1: LED rosso (0805)
LD2: LED giallo (0805)
LD3÷LD5: LED verde (0805)
P1÷P3: Microswitch
Q1: SPD50P03LG
Q2, Q3: BC817
Q4, Q5: BSS123
Q6: BC817
CN1: Connettore micro-USB
CN2: Strip maschio 4 vie
J1: Strip maschio 3 vie
BT1: Porta batterie CR2032
U1: MCP79410-I/SN
U2: Arduino UNO Rev3
U3: RaspberryPi 2/3/B+
Y1: Quarzo 32768 Hz

Varie:
- Batteria CR2032
- Strip M/F 6 vie
- Strip M/F 8 vie (2 pz.)
- Strip M/F 10 vie
- Circuito stampato S1254

LIBRERIA PER MCP79410

La libreria permette di configurare e gestire l'integrato MCP79410 ed è divisa in tre file; il primo contiene le funzioni sviluppate per l'occasione e ha estensione *.cpp* (lo abbiamo chiamato ***MCP79410.cpp***). Il secondo ha estensione *.h* e contiene le dichiarazioni di funzione del precedente più tutte le variabili e le strutture dati necessarie (***MCP79410.h***). Il terzo (***keywords.txt***) è un file di testo contenente le parole chiave delle funzioni pubbliche da utilizzare negli sketch Arduino. Tutti questi devono essere raggruppati sotto una cartella comune, all'interno della cartella di installazione dell'IDE Arduino, nominata ***MCP79410***. Così facendo, comparirà la libreria nello IDE Arduino sotto la voce di menu ***Sketch>Include Library***.
Quindi supponendo che lo IDE Arduino sia stato installato sotto ***C:\Program Files (x86)\Arduino*** la nostra libreria dovrà essere salvata nel seguente percorso: ***C:\Program Files (x86)\Arduino\libraries\MCP79410***. Oltre ai file di libreria è consuetudine aggiungere una cartella con degli sketch di esempio: nel nostro caso abbiamo creato la sotto-cartella ***Examples → MCP79410_SAdvancedSettings*** nella quale trovano posto i file del nostro sketch che ci permettono di configurare i registri di MCP79410.
Cominciamo a descrivere a grandi linee il file ***MCP79410.h*** che, come anzi detto, contiene le definizioni di tutte le funzioni utilizzate e la dichiarazione delle variabili e delle strutture dati pubbliche e private. In testa c'è una serie di dichiarazioni di costanti che identificano gli indirizzi hardware dell'integrato, per quanto riguarda sia la parte RTCC che la parte EEPROM,

Tabella 2

OSC START	SECTEN	SECONE
X	3	9

più tutti gli indirizzi dei registri presenti. Seguono una serie di costanti utili per settare o resettare i bit di configurazione presenti nei vari registri. Per ogni costante è stato pensato un commento che spiega come sfruttare la costante presentata. Ad esempio per attivare l'oscillatore del RTCC si deve agire sul registro "***RTCSEC***" e in particolare sul bit più significativo, eseguendo un'operazione booleana di tipo ***OR*** (*RTCSEC | OSCILLATOR_BIT_ON*). Invece per resettare tale bit si userà un'operazione booleana di tipo ***AND*** (*RTCSEC & OSCILLATOR_BIT_OFF*). Per eseguire una delle precedenti operazioni è necessario prima leggere il registro di interesse, eseguire l'operazione booleana desiderata e infine scrivere il nuovo valore nel registro.
Seguono una serie di costanti di configurazione con la descrizione di ogni singolo bit che li compone. Questa sezione, associata alla lettura attenta del data-sheet, aiuta a comprendere appieno come e cosa fare per configurare l'integrato. Le costanti possono essere modificate a piacere dall'utente a seconda delle proprie esigenze; nulla vieta di crearne di nuove. L'ultima sezione riguarda la gestione della EEPROM; ricordiamo che l'indirizzo hardware da utilizzare per accedere ad essa è diverso da quello del RTCC.
Dopo le dichiarazioni di costanti appena spiegate seguono le dichiarazioni di tutte le funzioni pubbliche disponibili all'utente finale per la realizzazione del proprio sketch, nonché la dichiarazione di tutte le variabili necessarie alla gestione della libreria. In particolare vale la pena soffermarci sulle strutture dati che ci danno la possibilità di agire a livello di bit per la configurazione dei registri. Vedremo tra poco che è possibile configurare i registri in due modi: uno prevede l'utilizzo di apposite funzioni dedicate e l'altro passa dalla configurazione delle strutture dati e successiva programmazione dei registri.
Ad esempio, la struttura dati nel **Listato 1** permette di configurare ogni singolo bit del registro "***CONTROL***". Supponendo di volere abilitare l'allarme 0 si dovrà settare il bit "***Alarm0_Enable***" con la seguente sintassi:

```
mcp79410.ControlReg.Bit.Alarm0_Enable = 1;
```

Così facendo abbiamo solo settato un valore in SRAM del microcontrollore ATmega 328P presente sulla scheda Arduino Uno R3. Il passo successivo sarà richiamare un'apposita funzione per andare a scrivere il dato nell'apposita locazione di memoria dell'integrato MCP79410, ad esempio utilizzando la funzione di libreria "***WriteSingleReg***" alla quale vanno passati una serie di parametri, tra cui l'indirizzo e il valore del registro di cui si vuole modificare il valore. La sintassi completa sarà:

```
mcp79410.WriteSingleReg(RTCC_HW_ADD, CONTROL_ADD,
mcp79410.ControlReg.ControlByte);
```

Oltre alla struttura dati appena illustrata, ne seguono altre, più articolate, che permettono la configurazione dei registri TimeKeeper, gli allarmi e la lettura dei TimeStamp al PowerUp e PowerDown. Come esempio vediamo -nel **Listato 2**- la struttura dati per la gestione dei registri TimeKeeper. Come si può notare si è definita, per ogni registro di configurazione del RTCC (TimeKeeper), una struttura di tipo "***union***" che permette di agire su ogni singolo bit del registro da configurare. Ogni "*union*" è stata raggruppata in una struttura dati "***struct***" nominata "***TimeKeeper***". Con questo approccio si ha un unico "*contenitore*" al quale è possibile accedere per la configurazione di ogni singolo bit o gruppi di bit di ogni registro. Facciamo un esempio chiarificatore: supponiamo di dover settare il bit "***StartOsc***" del registro "***RTCSEC***". Questo bit è presente nella "***union***" "***TimeKeeperSeconds***" quindi per settarlo si deve usare la seguente sintassi:

Fig. 6

Addr.	Register Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Section 5.3 "Timekeeping"									
00h	RTCSEC	ST	SECTEN2	SECTEN1	SECTEN0	SECONE3	SECONE2	SECONE1	SECONE0
01h	RTCMIN	—	MINTEN2	MINTEN1	MINTEN0	MINONE3	MINONE2	MINONE1	MINONE0
02h	RTCHOUR	—	12/24	AM/PM HRTEN1	HRTEN0	HRONE3	HRONE2	HRONE1	HRONE0
03h	RTCWKDAY	—	—	OSCRUN	PWRFAIL	VBATEN	WKDAY2	WKDAY1	WKDAY0
04h	RTCDATE	—	—	DATETEN1	DATETEN0	DATEONE3	DATEONE2	DATEONE1	DATEONE0
05h	RTCMTH	—	—	LPYR	MTHTEN0	MTHONE3	MTHONE2	MTHONE1	MTHONE0
06h	RTCYEAR	YRTEN3	YRTEN2	YRTEN1	YRTEN0	YRONE3	YRONE2	YRONE1	YRONE0
07h	CONTROL	OUT	SQWEN	ALM1EN	ALM0EN	EXTOSC	CRSTRIM	SQWFS1	SQWFS0
08h	OSCTRIM	SIGN	TRIMVAL6	TRIMVAL5	TRIMVAL4	TRIMVAL3	TRIMVAL2	TRIMVAL1	TRIMVAL0
09h	EEUNLOCK	Protected EEPROM Unlock Register (not a physical register)							
Section 5.4 "Alarms"									
0Ah	ALM0SEC	—	SECTEN2	SECTEN1	SECTEN0	SECONE3	SECONE2	SECONE1	SECONE0
0Bh	ALM0MIN	—	MINTEN2	MINTEN1	MINTEN0	MINONE3	MINONE2	MINONE1	MINONE0
0Ch	ALM0HOUR	—	12/24[1]	AM/PM HRTEN1	HRTEN0	HRONE3	HRONE2	HRONE1	HRONE0
0Dh	ALM0WKDAY	ALMPOL	ALM0MSK2	ALM0MSK1	ALM0MSK0	ALM0IF	WKDAY2	WKDAY1	WKDAY0
0Eh	ALM0DATE	—	—	DATETEN1	DATETEN0	DATEONE3	DATEONE2	DATEONE1	DATEONE0
0Fh	ALM0MTH	—	—	—	MTHTEN0	MTHONE3	MTHONE2	MTHONE1	MTHONE0
10h	Reserved	Reserved – Do not use							
Section 5.4 "Alarms"									
11h	ALM1SEC	—	SECTEN2	SECTEN1	SECTEN0	SECONE3	SECONE2	SECONE1	SECONE0
12h	ALM1MIN	—	MINTEN2	MINTEN1	MINTEN0	MINONE3	MINONE2	MINONE1	MINONE0
13h	ALM1HOUR	—	12/24[1]	AM/PM HRTEN1	HRTEN0	HRONE3	HRONE2	HRONE1	HRONE0
14h	ALM1WKDAY	ALMPOL[1]	ALM1MSK2	ALM1MSK1	ALM1MSK0	ALM1IF	WKDAY2	WKDAY1	WKDAY0
15h	ALM1DATE	—	—	DATETEN1	DATETEN0	DATEONE3	DATEONE2	DATEONE1	DATEONE0
16h	ALM1MTH	—	—	—	MTHTEN0	MTHONE3	MTHONE2	MTHONE1	MTHONE0
17h	Reserved	Reserved – Do not use							
Section 5.7.1 "Power-Fail Time-Stamp"									
Power-Down Time Stamp									
18h	PWRDNMIN	—	MINTEN2	MINTEN1	MINTEN0	MINONE3	MINONE2	MINONE1	MINONE0
19h	PWRDNHOUR	—	12/24	AM/PM HRTEN1	HRTEN0	HRONE3	HRONE2	HRONE1	HRONE0
1Ah	PWRDNDATE	—	—	DATETEN1	DATETEN0	DATEONE3	DATEONE2	DATEONE1	DATEONE0
1Bh	PWRDNMTH	WKDAY2	WKDAY1	WKDAY0	MTHTEN0	MTHONE3	MTHONE2	MTHONE1	MTHONE0
Power-Up Time Stamp									
1Ch	PWRUPMIN	—	MINTEN2	MINTEN1	MINTEN0	MINONE3	MINONE2	MINONE1	MINONE0
1Dh	PWRUPHOUR	—	12/24	AM/PM HRTEN1	HRTEN0	HRONE3	HRONE2	HRONE1	HRONE0
1Eh	PWRUPDATE	—	—	DATETEN1	DATETEN0	DATEONE3	DATEONE2	DATEONE1	DATEONE0
1Fh	PWRUPMTH	WKDAY2	WKDAY1	WKDAY0	MTHTEN0	MTHONE3	MTHONE2	MTHONE1	MTHONE0

Listato 1

```
union ControlReg {
    uint8_t  ControlByte;
    struct {
          uint8_t SquareWaveFreqOutput              :2;
          uint8_t CoarseTrimEnable                  :1;
          uint8_t ExtOscInput                       :1;
          uint8_t Alarm0_Enable                     :1;
          uint8_t Alarm1_Enable                     :1;
          uint8_t SquareWaveOutputEnable            :1;
          uint8_t LogicLevelOutput                  :1;
    } Bit;
} ControlReg;
```

```
mcp79410.TimeKeeper.Second.SecBit.StartOsc = 1;
```

Fatto ciò, si può richiamare la funzione di libreria "***WriteSingleReg***" per la scrittura di un singolo registro, oppure se si preferisce, configurare prima tutti i registri di cui sopra e poi chiamare la funzione di libreria "***WriteTimeKeeping***" alla quale dobbiamo passare un solo parametro ad indicare se il formato di gestione dell'ora debba essere nel formato 12h o 24h. Quindi la sintassi sarà:

```
mcp79410.WriteTimeKeeping(0);
```

Passiamo ora alle funzioni messe a disposizione dalla libreria. Sono state definite otto funzioni di uso generale, di cui tre servono a manipolare i singoli bit di un registro e le restanti lavorano direttamente sui singoli byte o gruppi di essi:

```
1) ToggleSingleBit(uint8_t ControlByte, uint8_t RegAdd, uint8_t Bit)
2) SetSingleBit(uint8_t ControlByte, uint8_t RegAdd, uint8_t Bit)
3) ResetSingleBit(uint8_t ControlByte, uint8_t RegAdd, uint8_t Bit)
4) WriteSingleReg(uint8_t ControlByte, uint8_t RegAdd, uint8_t RegData)
5) WriteArray(uint8_t ControlByte, uint8_t StartAdd, uint8_t Lenght)
6) ClearReg(uint8_t ControlByte, uint8_t RegAdd)
7) ReadSingleReg(uint8_t ControlByte, uint8_t RegAdd)
8) ReadArray(uint8_t ControlByte, uint8_t StartAdd, uint8_t Lenght)
```

La funzione (1) esegue la funzione "***Toggle***" sul bit desiderato del rispettivo registro. Quindi se il bit indicato è "***0***" diventa "***1***" e viceversa. I parametri da passare sono il "***ControlByte***" che identifica l'indirizzo hardware del nostro RTCC (consultare il file **MCP79410.h** per gli indirizzi hardware assegnati al device), il "***RegAdd***" che è l'indirizzo del registro su cui si vuole agire e infine il parametro "***Bit***" che identifica il bit che si vuole modificare (da 0 a 7). Il concetto appena esposto si può estendere alle successive due funzioni, (2) e (3), che eseguono rispettivamente un "***Set***" o un "***Reset***" del bit desiderato: i parametri da passare sono gli stessi. Le successive funzioni agiscono invece sui rispettivi byte sia in scrittura che in lettura. Per quanto riguarda le funzioni di scrittura (4) e lettura (7) di un singolo byte i parametri da passare alla funzione prevedono il solito "***ControlByte***", seguito dall'indirizzo su cui scrivere o da cui leggere "***RegAdd***" e, per quanto riguarda la sola scrittura il valore che si vuole scrivere ovvero "***RegData***". La funzione (6) "***ClearReg***" azzera un registro e necessita degli stessi parametri della funzione (7). Infine ci sono due funzioni di scrittura (5) e lettura (8) di n valori consecutivi le quali vogliono come parametri il "***ControlByte***", l'indirizzo di partenza "***StartAdd***" e infine il numero di byte "***Lenght***". Le funzioni (5) e (8) si appoggiano su un array di byte detto "***DataArray***" di lunghezza 16, più che sufficiente per le nostre esigenze. Le funzioni che fanno utilizzo dell'array di dati vengono utili per leggere o scrivere nella EEPROM o SRAM dell'integrato. Quando si deve scrivere nella EEPROM si può al massimo inviare otto byte alla volta, perché il buffer del device ha tale dimensione massima.
Passiamo ora a descrivere le funzioni più specializzate create apposta per gestire al meglio l'integrato MCP79410:

```
1) GeneralPurposeOutputBit(uint8_t SetReset)
2) SquareWaveOutputBit(uint8_t EnableDisable)
3) Alarm1Bit(uint8_t EnableDisable)
4) Alarm0Bit(uint8_t EnableDisable)
5) ExternalOscillatorBit(uint8_t EnableDisable)
6) CoarseTrimModeBit(uint8_t EnableDisable)
7) SetOutputFrequencyBit(uint8_t OutputFreq)
```

La funzione (1) serve per impostare a "***1***" logico o a "***0***" logico l'uscita MFP; ciò può avvenire nel solo caso che entrambi gli allarmi siano disabilitati e che l'uscita non sia impostata per riportare la frequenza di clock dell'oscillatore. Il parametro "***SetReset***", come suggerisce il nome, determina se l'uscita debba essere settata o resettata ("***1***" setta l'uscita, "***0***" viceversa). La funzione (2) abilita/disabilita la possibilità di riportare sulla linea MFP la frequenza di clock. Il parametro "***EnableDisable***" abilita o disabilita questa funzione ("***1***" abilita il riporto sull'uscita del clock, "***0***" viceversa). Le funzioni (3) e (4) servono per abilitare gli allarmi e, come per la precedente, l'unico parametro da passare è "***EnableDisable***". La funzione (5) serve per configurare l'integrato a ricevere un segnale di clock sul pin X1. Questo è utile se non si vuole usare il quarzo da 32.768Hz sui pin X1 e X2. Il solo parametro da passare è "***EnableDisable***" con il consueto significato.
La funzione (6) serve per attivare/disattivare il "***Coarse Trim mode***", il quale abbinato al registro "***OSCTRIM***" permette una regolazione grossolana della base dei tempi per la gestione del RTCC. La regolazione viene applicata con una cadenza di 128

Hz, apportando una notevole influenza sulla base dei tempi. È quindi preferibile lasciare disabilitata questa funzione e, se necessario, regolare la base dei tempi sfruttando il solo registro ***OSCTRIM***, ovviamente seguendo una apposita procedura (vedi riquadro *Digital Trimming*).
Le funzioni finora esposte modificano un solo bit per volta del registro "***CONTROL***" e per fare ciò sfruttano le funzioni generiche "***SetSingleBit***" e "***ResetSingleBit***" descritte precedentemente.
La funzione (7) serve per selezionare la frequenza da riportare sul pin MFP; la selezione perde di significato se l'uscita è configurata per la gestione degli allarmi o come uscita digitale generica.
Seguono ora un'altra serie di funzioni specializzate per la configurazione di alcune importanti caratteristiche di funzionamento messe a disposizione dell'integrato MCP79410. Queste impostazioni riguardano il TimeKeeper, gli allarmi e le funzioni di Timestamp:

```
1) StartOscillatorBit(uint8_t EnableDisable)
2) Hour12or24TimeFormatBit(uint8_t SetHourType)
3) AmPmBit(uint8_t SetAmPm)
4) VbatEnBit(uint8_t EnableDisable)
5) AlarmHour12or24TimeFormatBit(uint8_t SetHourType, uint8_t Alarm0_1)
6) AlarmAmPmBit(uint8_t SetAmPm, uint8_t Alarm0_1)
7) AlarmIntOutputPolarityBit(uint8_t SetReset, uint8_t Alarm0_1)
8) AlarmMaskBit(uint8_t Alarm0_1, uint8_t Mask)
9) ResetAlarmIntFlagBit(uint8_t Alarm0_1)
10) PowerHour12or24TimeFormatBit(uint8_t SetHourType, uint8_t PowerDownUp)
11) PowerAmPmBit(uint8_t SetAmPm, uint8_t PowerDownUp)
12) ResetPwFailBit(void)
```

La funzione (1) è fondamentale in quanto serve per attivare o disattivare l'oscillatore. Se l'oscillatore è spento non si ha nessuna attività da parte del RTCC e quindi nessuna gestione di data, ora e relativi allarmi. Il parametro "***EnableDisable***" abilita o disabilita l'oscillatore ("***1***" abilita l'oscillatore, "***0***" disabilita). Le funzioni (2), (5) e (10) servono per impostare il formato dell'ora rispettivamente per il timekeeper, gli allarmi e il timestamp al power-up/power-down. La rappresentazione dell'ora può essere del tipo 12h, accompagnato dalla dicitura AM/PM, oppure 24h; quindi a seconda del formato che si desidera utilizzare è conveniente allineare tutte e tre le possibili configurazioni in modo da non essere spaiati. Non ha molto senso avere il timekeeper impostato per lavorare in formato 24h e gli allarmi nel formato 12h, a meno di particolari applicazioni. La funzione (2) necessita di un solo parametro, "***SetHourType***", per selezionare uno dei due formati a disposizione: "***1***" imposta il formato 12h mentre "***0***" il formato 24h. Le funzioni (5) e (10) richiedono un parametro aggiuntivo per identificare rispettivamente a quale allarme/

Listato 2

```
typedef union TimeKeeperSecond {
    uint8_t  SecByte;
    struct {
        uint8_t SecOne          :4;
        uint8_t SecTen          :3;
        uint8_t StartOsc  :1;
    } SecBit;
} TimeKeeperSeconds;
typedef union TimeKeeperMinute {
    uint8_t  MinByte;
    struct {
        uint8_t MinOne          :4;
        uint8_t MinTen          :3;
        uint8_t Free            :1;
    } MinBit;
} TimeKeeperMinute;
typedef union TimeKeeperHour12 {
    uint8_t  Hour_12Byte;
    struct {
        uint8_t HrOne           :4;
        uint8_t HrTen           :1;
        uint8_t AmPm            :1;
        uint8_t _12_24          :1;
        uint8_t Free            :1;
    } Hour_12Bit;
} TimeKeeperHour12;
typedef union TimeKeeperHour24 {
    uint8_t  Hour_24Byte;
    struct {
        uint8_t HrOne           :4;
        uint8_t HrTen           :2;
        uint8_t _12_24          :1;
        uint8_t Free            :1;
    } Hour_24Bit;
} TimeKeeperHour24;
typedef union TimeKeeperWeekDay {
    uint8_t  WkDayByte;
    struct {
        uint8_t WkDay           :3;
        uint8_t VbatEn          :1;
        uint8_t PwrFail   :1;
        uint8_t OSCrun          :1;
        uint8_t Free            :2;
    } WkDayBit;
} TimeKeeperWeekDay;
typedef union TimeKeeperDate {
    uint8_t  DateByte;
    struct {
        uint8_t DateOne   :4;
        uint8_t DateTen   :2;
        uint8_t Free            :2;
    } DateBit;
} TimeKeeperDate;
typedef union TimeKeeperMonth {
    uint8_t  MonthByte;
    struct {
        uint8_t MonthOne  :4;
        uint8_t MonthTen  :1;
        uint8_t LeapYear  :1;
        uint8_t Free            :2;
    } MonthBit;
} TimeKeeperMonth;
typedef union TimeKeeperYear {
    uint8_t  YearByte;
    struct {
        uint8_t YearOne   :4;
        uint8_t YearTen   :4;
    } YearBit;
} TimeKeeperYear;

struct {
    TimeKeeperSeconds     Second;
    TimeKeeperMinute      Minute;
    TimeKeeperHour12      Hour12;
    TimeKeeperHour24      Hour24;
    TimeKeeperWeekDay     WeekDay;
    TimeKeeperDate        Date;
    TimeKeeperMonth       Month;
    TimeKeeperYear        Year;
} TimeKeeper;
```

timestamp applicare la modifica del parametro. Le funzioni (3), (6) e (11) servono per impostare, in merito al formato 12h dell'ora, se l'ora impostata fa riferimento al mattino o al pomeriggio (AM per la mattina e PM per il pomeriggio). Il parametro da passare è "***SetAmPm***", se "***1***" imposta PM invece se "***0***" imposta AM. Le funzioni (6) e (11) necessitano di un parametro aggiuntivo, come già spiegato per le funzioni (5) e (10). La funzione (4) serve per attivare la gestione della alimentazione a batteria in caso di mancanza di alimentazione primaria; quindi per alimentare a batteria l'RTCC si deve attivare il rispettivo bit e per fare ciò si utilizza la funzione appena introdotta: il parametro da passare è il consueto "***EnableDisable***".
La funzione (7) serve per impostare la polarità dell'uscita MFP (Tabella 1). I parametri da passare sono due: il primo ("***SetReset***") imposta la polarità, ovvero "***1***" polarità con logica ad alto livello e "***0***" polarità con logica a basso livello. Il secondo parametro ("***Alarm0_1***") serve per selezionare su quale allarme eseguire la modifica della polarità.
La funzione (8) serve per impostare la maschera di confronto per gli allarmi. Quando si configurano gli allarmi, oltre a impostare la data e l'ora desiderate per la generazione dell'interrupt, si deve anche impostare la maschera di confronto la quale ci permette di decidere su cosa eseguire il test per scatenare l'evento di allarme. Le opzioni disponibili sono:

- confronto solo sui secondi;
- confronto solo sui minuti;
- confronto solo sull'ora (tiene conto del formato impostato ovvero 12h o 24h);
- confronto solo sul giorno della settimana;
- confronto solo sulla data;
- confronto totale (vengono confrontati i secondi, i minuti, le ore, il giorno della settimana, la data e il mese).

Quando si genera l'interrupt dell'allarme n viene settato il rispettivo flag il quale deve essere azzerato manualmente da codice. Non va mai lasciato pendente. Per assolvere al compito si può sfruttare la funzione (9) a cui si deve passare un unico parametro che identifica l'allarme 0 o 1.
Per ultima la funzione (12) la quale serve per resettare il bit "***PwFail***" il quale indica che è mancata l'alimentazione principale e che quindi sono stati memorizzati i timestamp al power-down e al power-up. In altre parole viene salvata la data e l'ora del momento in cui è mancata/ritornata l'alimentazione. Quindi dopo avere letto i registri dei due timestamp si deve resettare obbligatoriamente questo flag se si vuole che alla prossima mancanza/ritorno alimentazione vengano nuovamente aggiornati i registri.
Rimangono da descrivere le ultime funzioni le quali, abbinate alle strutture dati precedentemente descritte, ci permettono di configurare agevolmente blocchi di registri associati alle varie funzioni messe a disposizione dal nostro RTCC:

```
1) WriteTimeKeeping(uint8_t Hour12or24Format)
2) ReadTimeKeeping(void)
3) WriteAlarmRegister(uint8_t Alarm0_1, uint8_t Hour12or24Format)
4) ReadAlarmRegister(uint8_t Alarm0_1)
5) WritePowerDownUpRegister(uint8_t PowerDownUp, uint8_t Hour12or24Format)
6) ReadPowerDownUpRegister(uint8_t PowerDownUp)
```

La funzione (1) serve per programmare la data e l'ora del RTCC, come già ampiamente detto sono compresi anche i bit di attivazione dell'oscillatore e della gestione dell'alimentazione a batteria. Quindi sfruttando la struttura dati apposita si configurano i registri con i valori corretti e poi si programma l'integrato usando la funzione (1). L'unico parametro che richiede è la selezione del formato dell'ora (12h o 24h). La funzione (2) serve per leggere la configurazione assegnata al TimeKeeper, la funzione legge i registri dell'integrato e li trasferisce nella struttura dati apposita che può essere poi utilizzata per le proprie applicazioni. Lo stesso discorso si applica alle funzioni (3) e (4) che servono rispettivamente per configurare i registri degli allarmi o leggerne il loro contenuto. Le due funzioni fanno riferimento alla loro apposita struttura dati. Essendo due i possibili allarmi la struttura dati è un array di strutture.
Chiudono la rassegna le funzioni (5) e (6) per la configurazione e lettura dei registri inerenti il timestamp al power-up e power-down. Anche in questo caso le funzioni sono associate ad una apposita struttura dati più precisamente un array di strutture. In realtà la libreria prevede altre due funzioni che riguardano la gestione della memoria EEPROM. Come già discusso l'integrato ha una memoria EEPROM interna da 128 Byte scrivibile a blocchi di otto Byte alla volta. Questa memoria è liberamente scrivibile dall'utente senza nessuna restrizione. Tuttavia si può decide di proteggere delle sezioni di EEPROM al fine di preservarne il contenuto dalla sovrascrittura. Questo si può fare configurando i bit "***BP1***" e "***BP0***" del registro "***STATUS***" mappato all'indirizzo 0xFF. Si può quindi decidere di proteggere tutta la memoria, metà memoria (Da 0x40 a 0x7F) o un quarto di memoria (Da 0x60 a 0x7F). La funzione che si occupa di configurare il registro "***STATUS***" è:

```
Set_EEPROM_WriteProtection(uint8_t Section)
```

alla quale si deve passare un solo parametro che identifica quale sezione proteggere. L'ultima funzione disponibile serve per sbloccare la memoria EEPROM protetta, per intenderci quella mappata dall'indirizzo 0xF0 a 0xF7, e scrivere un byte all'indirizzo desiderato. La funzione prevede due parametri ovvero l'indirizzo in cui scrivere e il dato da scrivere. La funzione è:

```
WriteProtected_EEPROM(uint8_t RegAdd, uint8_t RegData)
```

SKETCH COMPLETO

Per studiare le funzionalità messe a disposizione da MCP79410 abbiamo scritto uno sketch di esempio nel quale mostriamo come configurare la sezione TimeKeeper e gli allarmi 0 e 1. Lo sketch è suddiviso in sei file di cui il principale è "***MCP79410_AdvancedSettings.ino***", quindi con un doppio clic su questo si apre lo IDE Arduino e tutti i file ad esso associati. I file dello sketch sono:

- *MCP79410_AdvancedSettings* → File principale contenente le funzioni "***setup()***" e "***loop()***" tipiche di uno sketch Arduino, nonché le dichiarazioni di variabile, le costanti stringa, le costanti di tempo, le dichiarazioni delle macchine a stati ecc;
- *DigitalInput* → File di gestione degli ingressi digitali, ovvero i pulsanti P1, P2 e P3;
- *DigitalOutput* → File di gestione delle uscite digitali (il LED D4 e l'uscita "***ForceON***");
- *RTCC_Management* → File di gestione dell'integrato MCP79410 e relativa configurazione;
- *RTCC_Settings* → file per la programmazione dell'integrato MCP79410 TimeKeeper e allarmi;
- *TimersInt* → File di gestione dell'interrupt per la gestione dei timer.

Grazie al monitor seriale, attivabile dallo IDE Arduino, è possibile impartire una serie di comandi stringa per la configurazione del Timekeeper e degli allarmi 0 e 1. Inoltre durante il funzionamento "*normale*", ovvero quando non si stanno configurando i registri dell'integrato, il sistema è programmato per stampare sul monitor seriale alcune informazioni sullo stato del TimeKeeper e dei due allarmi. Lo schema di flusso del nostro sketch è illustrato nella **Fig. 7**: c'è una sequenza di inizializzazione che comprende sia gli ingressi che le uscite digitali, le macchine a stati, l'interrupt per le basi dei tempi e l'inizializzazione della libreria di gestione di MCP79410 (ciò identifica la funzione "***setup()***"). Seguono le funzioni di gestione, ovvero la lettura e il debouncing degli ingressi digitali, le macchine a stati di sistema e la funzione di lettura della configurazione assegnata ai registri di MCP79410 (ciò identifica la funzione "***loop()***").
Parliamo ora di come configurare i registri del nostro RTCC; prima di tutto è necessario collegare la scheda Arduino Uno R3 al PC tramite il connettore USB, così si assicura un'alimentazione stabile sia ad Arduino sia allo shield; in più si mette a disposizione la seriale e il relativo monitor per inviare i comandi da PC ad Arduino.
Attivato il monitor seriale si nota che il sistema esegue una lettura della configurazione dei registri ogni quindici secondi; la **Fig. 8** ne mostra il risultato. In testa, il sistema evidenzia che l'oscillatore del RTCC è attivo e mostra data e ora correnti. Seguono le configurazioni dei due allarmi:

1) stato dell'allarme = Abilitato/Disabilitato;
2) polarità uscita = livello logico alto/basso;
3) Flag interrupt = indica se è scattato l'allarme a seconda della sua configurazione;
4) ora impostata = mostra l'ora configurata per l'allarme;
5) data impostata = mostra la data impostata per l'allarme (l'anno perde di significato in quanto irrilevante per la gestione degli allarmi);
6) maschera = mostra la maschera di confronto per la generazione dell'interrupt a seconda della configurazione sopra esposta.

A questo punto possiamo decidere di fare due cose: configurare la data e l'ora dell'RTCC e quindi, se è la prima programmazione, attivare di conseguenza l'oscillatore e la gestione della alimentazione da batteria; la seconda è configurare gli allarmi 0 e 1.
Cominciamo col descrivere la configurazione di data e ora del RTCC: prima di tutto si deve mandare il sistema in configurazione di data e ora e per fare ciò si deve premere il pulsante P1 per più di due secondi; LD4 emetterà un lampeggio a indicare l'avvenuto riconoscimento della pressione del tasto. Entrati nella configurazione, LD4 lampeggia e sul monitor seriale viene stampata una stringa a indicare che il sistema è entrato in configurazione di data e ora. Per uscire da questa modalità, premere nuovamente il pulsante P1 per più di due secondi; anche in questo caso una stringa indicherà quanto appena fatto (**Fig. 9**).
Il primo passo è configurare la data e il giorno della settimana: allo scopo bisogna scrivere la

Digital **Trimming**

Per correggere l'imprecisione introdotta dall'oscillatore esterno si può sfruttare il registro ***"OSCTRIM"*** il quale può introdurre un fattore correttivo fino a ±129ppm, per sfruttare questa funzione è obbligatorio resettare il bit ***"CRSTRIM"*** del registro ***"CONTROL"***. Il registro ***"OSCTRIM"*** può anche essere sfruttato per correggere gli errori dell'oscillatore dovuti alle variazioni di temperatura. Il bit più significativo del registro indica il segno del valore correttivo mentre i restanti bit (***"TRIMVAL"***) contengono il valore da aggiungere o sottrarre. Ad ogni valore corrispondono due impulsi di clock che vengono aggiunti o sottratti ogni minuto. Se invece si ha il bit "OSCTRIM" settato si ha che il valore indicato viene aggiunto o sottratto 128 volte al secondo.
Il valore da caricare in ***"TRIMVAL"*** deve essere calcolato e per fare ciò ci sono due metodi. Il primo consiste nel misurare la frequenza di clock riportata sul pin di uscita ***"MFP"*** e calcolare di quanto è spostata rispetto al valore aspettato. Il secondo metodo consiste nel verificare l'ammontare di secondi guadagnati o persi durante un periodo di tempo. Con il primo metodo, l'equazione per calcolare il valore di ***"TRIMVAL"*** è:

$$TRIMVAL = \frac{\frac{(F_{IDEAL} - F_{MEAS}) * 32768}{F_{IDEAL}} * 60}{2}$$

Dove ***"F_{IDEAL}"*** è la frequenza che ci si aspetta di misurare mentre ***"F_{MEAS}"*** è la frequenza realmente misurata. Per quanto riguarda il secondo metodo si deve usare le seguenti formule:

$$PPM = \frac{SecDeviation}{ExpectedSec} * 1000000$$

$$TRIMVAL = \frac{PPM * 32768 * 60}{2000000}$$

Dove ***"ExpectedSec"*** sono i secondi che ci si aspetta di leggere in un dato periodo, per eseguire una calibrazione accurata lavorare su lunghi periodi. Mentre ***"SecDeviation"*** è il numero di secondi guadagnati o persi durante il periodo.

stringa di comando seguente, sfruttando l'apposita textbox del monitor seriale e cliccare sul pulsante "*Invia*":

```
DateSet: 12/12/2015 - SAT
```

dove "`DateSet:`" è il comando da attuare e "`12/12/2015 - SAT`" sono la data e il giorno della settimana. Il giorno della settimana è in inglese (`MON, TUE, WED, THU, FRI, SAT, SUN`).
Nel caso in cui il comando contenga degli errori il sistema ritornerà una stringa di avviso invitando l'utente ad editare nuovamente il comando.
Il secondo passo prevede di configurare l'ora, quindi digitare il seguente comando:

```
TimeSet: 14:30:10
```

Dove "`TimeSet:`" è il comando da attuare e "`14:30:10`" è l'orario (formato 24h) da impostare. Se invece si vuole impostare lo stesso orario ma nel formato 12h, si deve scrivere la seguente:

```
TimeSet: 02:30:10 - PM
```

Il sistema riconosce il formato desiderato e configura di conseguenza i registri del RTCC.
Passiamo ora alla configurazione degli allarmi 0 e 1: per attivare la configurazione degli allarmi, premere per più di due secondi il pulsante P2, anche in questo caso LD4 lampeggerà e a monitor verrà stampata una apposita stringa sia per l'attivazione che disattivazione dell'impostazione degli allarmi (**Fig. 10**). Come primo passo dovete configurare la data, la maschera e la polarità dell'allarme 0 o 1. Supponiamo di lavorare con l'allarme 0:

```
DateSetAlarm(0): 12/12/2015 - SAT - SecondsMatch - LHL
```

dove "`DateSetAlarm(0):`" è il comando di configurazione della data per l'allarme 0. Il terzo parametro è la maschera di confronto, che può essere impostata per lavorare solo sui secondi, solo sui minuti, solo sulle ore, solo sui giorni della settimana, solo sulla data, oppure imporre il confronto su tutti i parametri contemporaneamente. Nel nostro esempio scegliamo che il confronto debba avvenire solo sui secondi; gli altri a disposizione sono "`MinutesMatch`", "`HoursMatch`", "`DayOfWeekMatch`", "`DateMatch`" e "`AllMatch`". Il quarto parametro serve per selezionare la polarità dell'uscita dove "`LHL`" indica polarità positiva mentre "`LLL`" polarità negativa. Il secondo passo serve per configurare l'ora:

```
TimeSetAlarm(0): 14:30:10
```

dove "`TimeSetAlarm(0):`" è il comando di configurazione dell'ora per l'allarme 0. In questo caso, per congruenza con la configurazione dell'ora del RTCC il formato è 24h. Se si volesse impostare il formato 12h, la stringa diventerebbe:

```
TimeSetAlarm(0): 02:30:10 - PM
```

Come per la configurazione del RTCC il

sistema riconosce automaticamente il formato desiderato. Va da sè che se si volesse impostare l'allarme 1 i comandi sono identici ai precedenti ma con riferimento all'allarme 1; ad esempio `DateSetAlarm(1):.....` ecc. Volendo è anche possibile resettare gli allarmi impostati, ovvero cancellare completamente la loro programmazione, disabilitandoli. Per fare ciò basta impartire il comando:

`ResetAlarm(0):` oppure `ResetAlarm(1):`

Se invece si desidera semplicemente disabilitare un allarme usare:

`DisableAlarm(0):` oppure `DisableAlarm(1):`

Per abilitarlo, il comando è:

`EnableAlarm(0):` oppure `EnableAlarm(1):`

Tutte le stringhe, sia di comando che di avviso, sono memorizzate nella memoria Flash del microcontrollore Atmel al fine di risparmiare la memoria SRAM; infatti ricordate che tutte le volte che si utilizza la funzione "***println***" con del testo da stampare questo viene caricato in SRAM, occupando spazio prezioso.
I comandi sopra indicati servono per configurare le strutture dati di cui abbiamo parlato durante la descrizione della libreria e successiva programmazione dei registri; per questo le varie programmazioni sono divise in due step, di cui è sempre il secondo che congela i dati nei registri dell'RTCC. Ad esempio, durante la configurazione di data e ora dell'RTCC (TimeKeeper) una volta ricevuti i dati corretti il sistema richiama la funzione di programmazione seguente:

```
mcp79410.WriteTimeKeeping(0);
```

Il parametro "0" indica che si è selezionato il formato 24h. Invece durante la configurazione degli allarmi si useranno le seguenti:

```
mcp79410.WriteAlarmRegister(AlarmIndex, 0);
mcp79410.Alarm0Bit(1);
```

La prima funzione salva i dati nei registri per l'allarme n, dipende dal valore della variabile "***AlarmIndex***", 0 per l'allarme 0 e 1 per l'allarme 1, con formato 24h; la seconda attiva l'allarme 0. Configurando opportunamente i due allarmi

Fig. 7

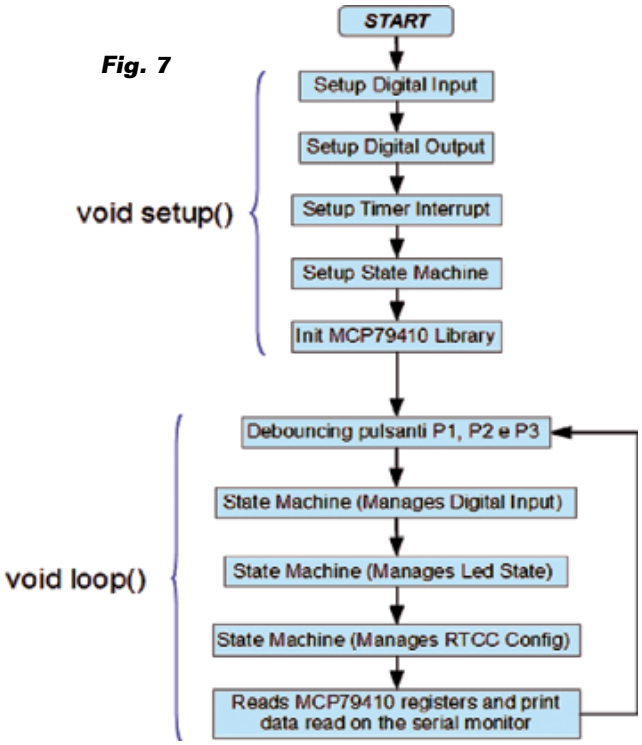
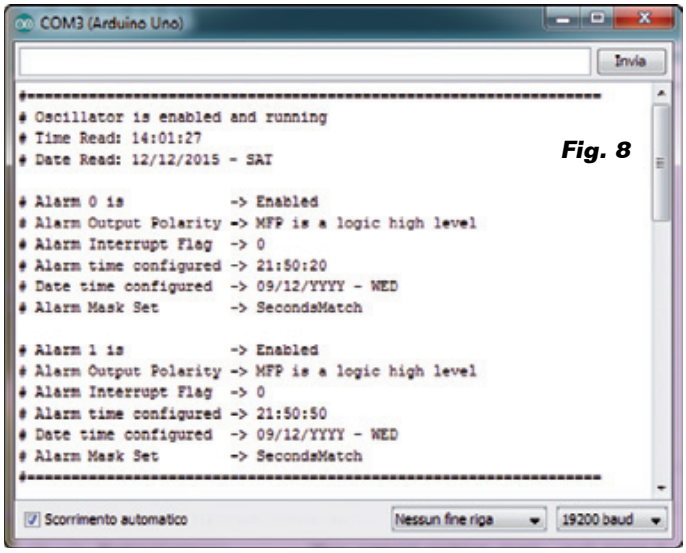


Fig. 8



si arriva ad avere una situazione di lavoro dove l'allarme 0 forza l'accensione e l'allarme 1 lo spegnimento. Chiariamo meglio quanto appena detto: una volta configurati i registri dei due allarmi si deve scollegare il cavo USB dal PC togliendo quindi alimentazione a tutta l'elettronica compreso lo shield. Poi si deve fornire alimentazione a quest'ultimo tramite il connettore microUSB assicurandosi che il jumper J1 sia in posizione 2-3. A questo punto l'elettronica a valle del MOSFET Q1 verrà alimentata solo se si attiverà l'interrupt dell'allarme 0 intervenendo sull'uscita MFP; in tal

```
#============================================================
# TimeKeeper settings in progress. System wait a command to set date and time
#============================================================

#============================================================
# TimeKeeper settings has been stopped
#============================================================
```

Fig. 9

caso lo sketch si accorge che il flag dell'interrupt dell'allarme 0 è andato a "*1*" logico e quindi forza alto il segnale "*ForceOn*" mantenendo sempre accesa l'alimentazione indifferentemente dallo stato dell'uscita MFP. Il sistema rimane quindi in attesa che venga intercettato l'evento per l'allarme 1, quando questo accade lo sketch se ne accorge e quindi forza basso il segnale "*ForceOn*" spegnendo automaticamente tutta l'elettronica. Tutte le volte che il sistema rileva che è scattato un allarme dovrà lui stesso provvedere al reset del flag corrispondente seguendo una apposita procedura. Per i dettagli vedere il codice dello sketch in particolare le due funzioni:

```
void ResetAlarmFlag_0(void)
void ResetAlarmFlag_1(void)
```

Per vedere questo andamento perpetuo di accensione e spegnimento automatici dell'alimentazione è sufficiente configurare i due allarmi a fare il confronto solo sui secondi, come evidenziato prima, e impostare i secondi dell'ora in modo appropriato ad esempio: 12:10:10 per l'allarme 0 (Accensione) e 12:10:40 per l'allarme 1 (Spegnimento). Così facendo l'elettronica rimane accesa per 30 secondi e spenta per altrettanti 30. Ovviamente questo è solo uno sketch di esempio e nei casi reali si vorrebbe che la scheda si accendesse a una determinata ora della giornata per poi spegnersi ad un'altra. Quindi si potrebbe pensare di configurare il confronto non sui secondi ma sulle ore e impostare ad esempio l'ora come segue: 08:00:00 per l'allarme 0 (Accensione) e 18:00:00 per l'allarme 1 (Spegnimento). Così per tutti i i giorni dell'anno si avrebbe una accensione e uno spegnimento alle ore desiderate. Le possibili combinazioni a disposizione sono veramente tante e modificando il codice dello sketch in modo opportuno si può rendere tutto più dinamico. In altre parole è il sistema a decidere quando avverrà la prossima accensione o spegnimento a seconda dei casi; dipende solo dalla fantasia dell'utente e dal codice implementato. La nostra libreria e la nostra elettronica servono solo per darvi il giusto mezzo per realizzare le vostre più disparate applicazioni.

Fig. 10

```
#============================================================
# Alarms settings in progress. System wait a command to set alarm 0/1
#============================================================

#============================================================
# Alarms settings has been stopped
#============================================================
```

REALIZZAZIONE PRATICA

Spendiamo ora qualche parola sulla costruzione dello shield, che richiede un circuito stampato a doppia ramatura le cui tracce sono disponibili sul nostro sito *www.elettronicain.it*. Lo shield è composto praticamente tutto da componenti SMD; fanno eccezione il portapile, gli strip maschio e femmina e i pulsanti. Per il montaggio dotatevi di un saldatore da 20W a punta molto fine, di pasta flussante, di una pinzetta e una lente d'ingrandimento per posizionare i componenti e verificare le saldature. I primi componenti da montare sono il MOSFET Q1 e l'U1, da posizionare con i pin al centro delle rispettive piazzole e stagnare un pin per lato; seguono tutti i passivi, il fusibile e poi il quarzo e il connettore micro USB. Fatto ciò si passa al montaggio dei pulsanti e del portapila, quindi si inseriscono e si saldano i pin-strip laterali per Arduino e quello per Raspberry Pi. Per l'orientamento dei componenti polarizzati fate riferimento al piano di montaggio che trovate nelle pagine precedenti. Completate le saldature e verificate con la lente che non siano cortocircuiti, inserite la pila CR2032.

CONCLUSIONI

In queste pagine abbiamo presentato lo shield basato sull'MCP79410 e lo sketch di gestione per Arduino. Nella prossima puntata descriveremo degli sketch specifici per singole funzioni e affronteremo l'applicazione e il software per l'utilizzo dello shield con la Raspberry Pi.

per il MATERIALE

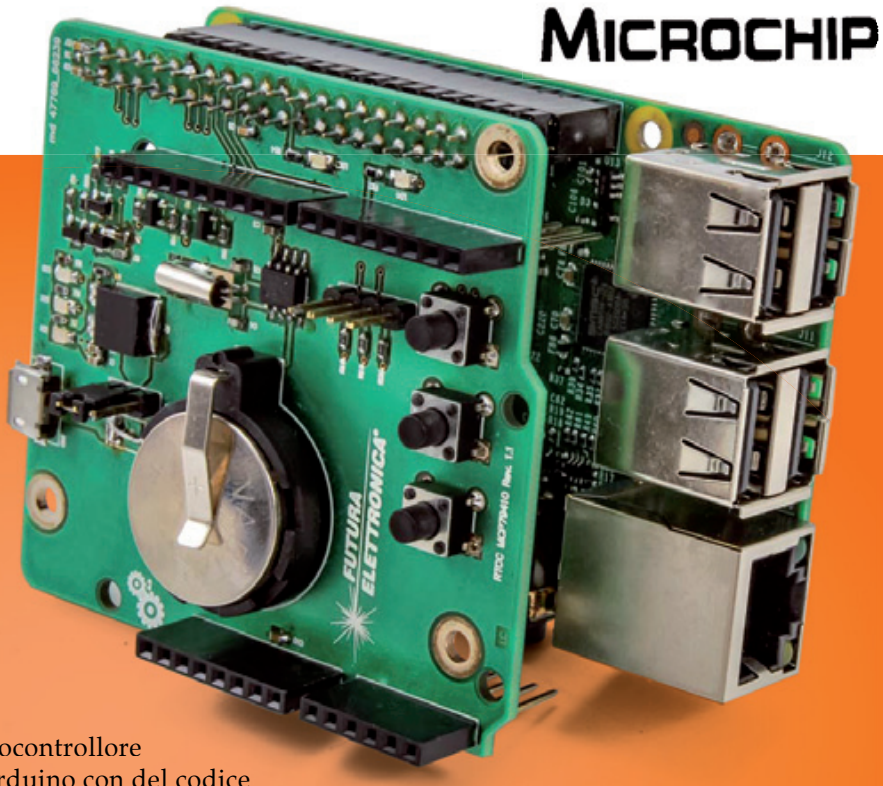
Lo shield RTC Raspberry-Arduino (cod. FT1254M) viene venduto montato e collaudato ed è disponibile presso Futura Elettronica al prezzo di Euro 20,00. Il prezzo si intende IVA compresa.

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

RTC SHIELD PER ARDUINO E RASPBERRY PI

di MATTEO DESTRO

Concludiamo il discorso su Arduino e vediamo come utilizzare lo shield con Raspberry Pi mediante un'apposita libreria. Seconda e ultima puntata.



Ci siamo lasciati dopo aver descritto un nuovo shield RTCC basato sull'integrato Microchip MCP79410 ed aver esaminato lo sketch per Arduino; ora, come anticipato nella prima puntata, completiamo il discorso su Arduino e passiamo "a bomba" all'applicazione con la Raspberry Pi.
Iniziamo presentandovi quattro mini sketch molto semplici e minimali per acquisire familiarità con la nostra libreria e imparare ad usarla. Questi ulteriori sketch vi aiuteranno a studiare e comprendere meglio lo sketch completo descritto il mese scorso e li abbiamo preparati come alternativa per chi desidera gestire solo alcune funzioni dell'integrato Microchip e non desidera riempire la memoria del microcontrollore di Arduino con del codice non utilizzato.

SKETCH PER TIMEKEEPR

Veniamo allo sketch per configurare solo il TimeKeeper del nostro RTCC. Anche in questo caso abbiamo suddiviso per semplicità lo sketch su tre file distinti in particolare abbiamo:

- *MCP79410_SetTimeKeeper* → file principale contenente le funzioni "***setup()***" e "***loop()***" nonché tutte le dichiarazioni di variabili, le costanti stringa, le costanti di tempo, le dichiarazioni delle macchine a stati ecc.;

- *RTCC_Settings* → file per la configurazione del TimeKeeper;
- *TimersInt* → file di gestione dell'interrupt per il controllo dei timer.

Concentriamoci sul file "***RTCC_Settings***" e in particolare sulla funzione "`void RTCC_TimeKeeperSettings(void)`" la quale ci permette di configurare la data e l'ora del nostro RTCC. Questa funzione viene richiamata solo durante l'esecuzione del "***setup()***" e mai in altra occasione. La funzione lavora sul formato 24h, ma volendo si può modificare per lavorare sul formato 12h; infatti ci sono delle parti di codice commentate che servono appunto per la configurazione dei registri nel secondo formato.
La funzione è impostata per configurare come data il Venerdì del 25/12/2015 e per fare ciò carica nella struttura dati del TimeKeeper i nuovi valori della data e il giorno della settimana, tenendo ben presente il formato, BCD dopodiché carica nella struttura dati i nuovi valori per l'ora nel formato 24h. L'ora scelta per l'esempio è 14:25:30. Infine, sempre della struttura dati, si attiva l'oscillatore agendo sull'apposito bit e si attiva la modalità batteria nel caso in cui venga a mancare l'alimentazione principale.
A questo punto per rendere effettive le modifiche si deve richiamare la funzione per programmare il RTCC e in particolare i registri del TimeKeeper:

```
mcp79410.WriteTimeKeeping(0);
```

Il parametro 0 indica che si sta lavorando in formato 24h. Durante l'esecuzione del "***loop()***" viene richiamata la sola funzione di lettura dei registri del TimeKeeper, una volta ogni 10 secondi, i quali vengono decodificati e stampati sul monitor seriale con la stessa filosofia adottata dal precedente sketch. Si può quindi vedere l'avanzare dei secondi e dei minuti. Tutte le volte che lo sketch verrà fatto ripartire, si avrà una riconfigurazione di data e ora con i valori detti prima.

SKETCH PER ALLARME 0

Vediamo ora uno sketch che riguarda semplicemente come configurare gli allarmi del nostro RTCC. Anche in questo caso abbiamo suddiviso per semplicità lo sketch su tre file distinti:

- *MCP79410_SetAlarm*→ file principale contenente le funzioni "***setup()***" e "***loop()***" nonché tutte le dichiarazioni di variabili, le costanti stringa, le costanti di tempo, le dichiarazioni delle macchine a stati ecc.;
- *RTCC_Settings* → file per la configurazione degli allarmi;
- *TimersInt* → file di gestione dell'interrupt per la gestione dei timer.

Concentriamoci sul file "***RTCC_Settings***" dove incontriamo nuovamente la funzione "`void RTCC_TimeKeeperSettings(void)`", vista nello sketch precedente, per la configurazione di data e ora e la funzione "`void RTCC_AlarmSettings(void)`" per la configurazione dell'allarme 0; per semplicità abbiamo omesso la configurazione dell'allarme 1.
La funzione di configurazione dell'allarme inizia disabilitando entrambi gli allarmi per poi passare a configurare la struttura dati apposita; questa è un array di strutture con dimensione 2, ossia allarme 0 e allarme 1. Per prima cosa si deve configurare la data omettendo l'anno che perde di significato nella configurazione degli allarmi. Al passo successivo si configura la maschera di confronto, noi scegliamo di eseguire il confronto solo sui secondi (le righe commentate mostrano che valori assegnare nel caso in cui si decidesse di utilizzare una diversa maschera di confronto).
Segue la configurazione della polarità dell'uscita MFP e per ultimo si imposta l'ora ovviamente nel formato 24h congruente con la data e l'ora impostata per il TimeKeeper.
Per rendere effettive le modifiche apportate alla struttura dati richiamare in sequenza le funzioni:

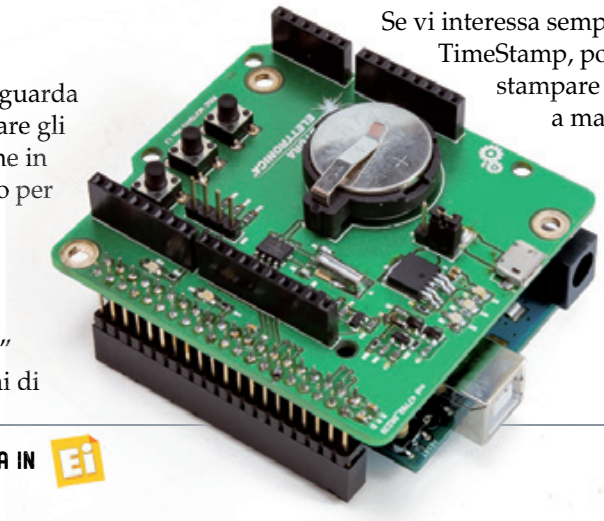
```
mcp79410.WriteAlarmRegister(0, 0);
mcp79410.Alarm0Bit(1);
```

dove la prima invia i dati appena inseriti ai registri di configurazione dell'allarme 0 mentre la seconda abilita l'allarme stesso.

SKETCH PER LEGGERE IL TIMESTAMP

Se vi interessa semplicemente leggere i registri TimeStamp, power-up e power-down, per stampare a monitor quando è venuta a mancare l'alimentazione e quando è ritornata, usate lo sketch descritto qui di seguito, che è suddiviso in due file:

- *MCP79410_timeStamp* → file principale contenente le funzioni "***setup()***" e "***loop()***"



nonché tutte le dichiarazioni di variabili, le costanti stringa, le costanti di tempo, le dichiarazioni delle macchine a stati e le funzioni di gestione dei TimeStamp;
- *TimersInt* → file di gestione dell'interrupt per la gestione dei timer.

In questo sketch non si eseguono configurazioni ma semplicemente si va a leggere il contenuto dei registri del timestamp. Questo è utile per stampare a video data e ora di quando è venuta a mancare l'alimentazione della scheda e data e ora di quando è ritornata. Durante il "***setup()***" non ci sono configurazioni particolari, carichiamo soltanto un timer per ritardare la lettura dei registri in modo da dare tempo di attivare il monitor seriale dopo il collegamento del cavo USB alla scheda Arduino UNO R3. La funzione che si occupa di leggere i registri viene eseguita una volta ogni dieci secondi, allo scadere del timer vengono letti sia i registri di data e ora correnti e, se il bit "***PwFail***" è a uno logico anche i registri del TimeStamp in quanto c'è stata sicuramente una mancanza di alimentazione. Tutte queste informazioni vengono poi stampate sul monitor seriale con la consueta formattazione. In questo sketch si sfruttano le funzioni di libreria:

```
ReadTimeKeeping(void)
ReadSingleReg(uint8_t ControlByte, uint8_t RegAdd)
ReadPowerDownUpRegister(uint8_t PowerDownUp)
ResetPwFailBit(void)
```

SKETCH PER GESTIRE LA EEPROM

Vediamo in ultimo lo sketch per utilizzare la EEPROM del nostro RTCC, sia quella protetta (8 Byte) che non protetta (128 Byte). Lo sketch è suddiviso in due file:
- *MCP79410_EEPROM* → file principale contenente le funzioni "***setup()***" e "***loop()***" nonché tutte le dichiarazioni di variabili, le costanti stringa, le costanti di tempo, le dichiarazioni delle macchine a stati e le funzioni di gestione della EEPROM;
- *TimersInt* → file di gestione dell'interrupt per la gestione dei timer.

Per testare la memoria EEPROM da 128 Byte proviamo a scrivere in memoria la scritta "***ElettronicaIn***" partendo dall'indirizzo 0x00, sfruttando la libreria e in particolare la funzione di scrittura:

```
WriteSingleReg(uint8_t ControlByte, uint8_t RegAdd, uint8_t RegData)
```

Sfruttando l'array presente nella nostra libreria e scrivendo un piccolo ciclo do-while, riusciamo a scrivere la nostra stringa nella memoria EEPROM. Quanto esposto è tutto raccolto in una funzione nominata "`void Set_EEPROM(void)`" la quale viene chiamata solo durante l'esecuzione del "***setup()***". Per verificare l'avvenuta scrittura abbiamo scritto una piccola funzione "`void Read_EEPROM(void)`" per rileggere, ogni dieci secondi, il contenuto delle prime locazioni di memoria EEPROM per poi stamparne il risultato sul monitor seriale. Anche in questo caso la funzione è composta da poche righe di codice compreso un ciclo do-while per la stampa. La funzione di libreria utilizzata per leggere la EEPROM è:

```
ReadArray(uint8_t ControlByte, uint8_t StartAdd, uint8_t Lenght)
```

Oltre alla memoria EEPROM classica abbiamo testato la scrittura e lettura della memoria EEPROM protetta (8 Byte) per memorizzare un ipotetico MAC address. Anche in questo caso abbiamo scritto due piccole funzioni, la prima per scrivere in EEPROM e la seconda per leggere la EEPROM. Quindi con "`void Set_Protected_EEPROM(void)`" scriviamo la memoria EEPROM solo durante il "***setup()***" e poi ogni dieci secondi andiamo a rileggerne il contenuto con la funzione "`void Read_Protected_EEPROM(void)`".
Per la scrittura si usa la funzione di libreria:

```
WriteProtected_EEPROM(uint8_t RegAdd, uint8_t RegData)
```

Si può scrivere un solo byte alla volta, anche in questo caso si sfrutta l'array della nostra libreria più un piccolo ciclo do-while.
Bene, detto ciò abbiamo concluso il discorso sull'applicazione con Arduino.

ED ORA...RASPBERRY PI

Lo shield RTCC dispone di una zoccolatura doppia che ne permette l'inserzione di volta in volta sulla scheda desiderata. Dopo avervi descritto il firmware per abbinarla ad Arduino, in quest'ultima puntata mostreremo come utilizzare il nostro shield RTCC con Raspberry Pi 2.0/B+ sfruttando la libreria in codice Python sviluppata per l'evenienza. Prima di procedere riepiloghiamo alcuni dettagli dello shield, che si basa sull'integrato Microchip MCP79410; nello specifico, diamo uno sguardo alle interconnessioni tra esso e la Raspberry Pi. A tale scheda vengono collegate diverse linee tra cui il bus di comunicazione I^2C per la gestione dell'integrato U1 (tra Raspberry Pi e l'integrato U1 è interposto un traslatore di livello meglio descritto nella prima puntata, in quanto Raspberry Pi lavora con livello di tensione +3,3V mentre U1 con livello

+5V), le linee dei pulsanti P1, P2 e P3, le linee di trigger, il LED e la linea di forzatura ON. Se necessario, il bus I²C, oltre a essere portato all'integrato U1, può essere prolungato verso altre schede connesse a Raspberry Pi tramite il solito connettore di espansione: l'importante è che non ci siano delle sovrapposizioni con gli indirizzi hardware già impegnati. Quindi, riassumendo, i seguenti pin della Raspberry Pi hanno questo utilizzo:

- GPIO 2 e 3 sono usati per collegare il bus I²C;
- GPIO 17, 18 e 27 sono usati per collegare i tre pulsanti utilizzati dallo sketch realizzato in codice Python;
- GPIO 22 e 23 sono usati per collegare le due linee di trigger utili durante le fasi di debug;
- GPIO 24 è usato per gestire la linea di forzatura ON dell'elettronica tramite il transistor NPN Q6 e il relativo MOSFET Q1, usato come interruttore;
- GPIO 25 serve per pilotare il LED LD5.

Concludiamo qui la descrizione dell'hardware messo a disposizione della nostra scheda, se ritenete opportuno approfondire l'argomento vi invitiamo a rileggere la precedenta puntata per i dettagli.

LIBRERIA PER RASPBERRY PI

La libreria sviluppata per l'occasione permette di configurare e gestire l'integrato MCP79410 in tutte le sue funzioni rendendo agevole configurarlo per le proprie esigenze, che possono essere diverse da quelle sviluppate nel nostro sketch di esempio di cui discuteremo più avanti.
Diversamente dal mondo Arduino la libreria è scritta in Python ed è composta da soli due file con estensione "***.py***".
Il primo si chiama "***MCP79410.py***" e contiene tutte le funzioni di libreria mentre il secondo è "***MCP79410_DefVar.py***" e contiene la dichiarazione di tutte le costanti e le strutture dati. I due file fanno quindi parte di un pacchetto che può essere installato sulla vostra Raspberry Pi e utilizzato nei vostri sketch. Vedremo in seguito come viene generato il pacchetto da distribuire e come si deve fare per installarlo nella propria distribuzione.
Oltre ai due file di libreria abbiamo scritto uno sketch per mostrare come utilizzare le funzioni implementate. Lo sketch è composto da tre file distinti: "***MCP79410_Main.py***" dove trova posto il main, "***MCP79410_PrintFunc.py***" dove troviamo tutte le funzioni di stampa a video dei dati letti dall'integrato MCP79410 e infine "***MCP79410_SetRegisters.py***", dove sono memorizzate le funzioni di configurazione dell'integrato MCP79410.

Cominciamo a descrivere a grandi linee il file "***MCP79410_DefVar.py***" il quale, come detto sopra, contiene le definizioni di tutte le funzioni utilizzate, nonché la dichiarazione delle variabili e delle struttura dati pubbliche e private. Da una prima occhiata si può vedere che è una copia quasi identica del relativo file realizzato per Arduino con alcune differenze dovute al differente linguaggio di programmazione utilizzato.
In testa al file ci sono le dichiarazioni delle costanti come ad esempio gli indirizzi hardware dell'integrato, ricordiamo che MCP79410 possiede due indirizzi hardware distinti, oppure gli indirizzi dei registri a cui puntare per la lettura/programmazione dei parametri operativi dell'integrato. Seguono una serie di costanti per la programmazione dei registri con dei valori standard, questa sezione, associata alla lettura attenta del data-sheet, aiuta a comprendere appieno come si deve fare e cosa si deve configurare per il corretto funzionamento dell'integrato. Tutte le costanti inserite possono essere modificate a piacere dall'utente a seconda delle proprie esigenze; per ogni costante è presente un commento utile a comprenderne il significato.
Infine trovano posto le costanti per la gestione della EEPROM; ricordiamo che per accedere ad essa l'indirizzo hardware da utilizzare è diverso da quello del RTCC. Oltre alle costanti appena descritte trovano parte le strutture dati fondamentali per una gestione nidificata dei dati per la programmazione/lettura dei vari registri presenti.
Le strutture dati sono suddivise in sezioni distinte, in particolare abbiamo la sezione per il TimeKeepr, la sezione per gli allarmi e la sezione per i TimeStamp. Come per il mondo Arduino, è possibile configurare i registri in due modi distinti: un metodo prevede l'utilizzo di apposite funzioni dedicate l'altro tramite la configurazione delle strutture dati e successiva programmazione dei registri. Ad esempio la struttura dati seguente permette di configurare ogni singolo bit del registro "***CONTROL***":

```
class CtrlBit(Structure):
      _fields_ = [("SquareWaveFreqOutput",    c_uint8,
2),
                  ("CoarseTrimEnable",        c_uint8, 1),
                  ("ExtOscInput",             c_uint8, 1),
                  ("Alarm0_Enable",           c_uint8, 1),
                  ("Alarm1_Enable",           c_uint8, 1),
                  ("SquareWaveOutputEnable", c_uint8, 1),
                  ("LogicLevelOutput",        c_uint8, 1)]

class CtrlByte(Structure):
      _fields_ = [("Byte0", c_uint8)]

class CtrlReg(Union):
      _fields_ = [("ControlBit",  CtrlBit),
                  ("ControlByte", CtrlByte)]
```

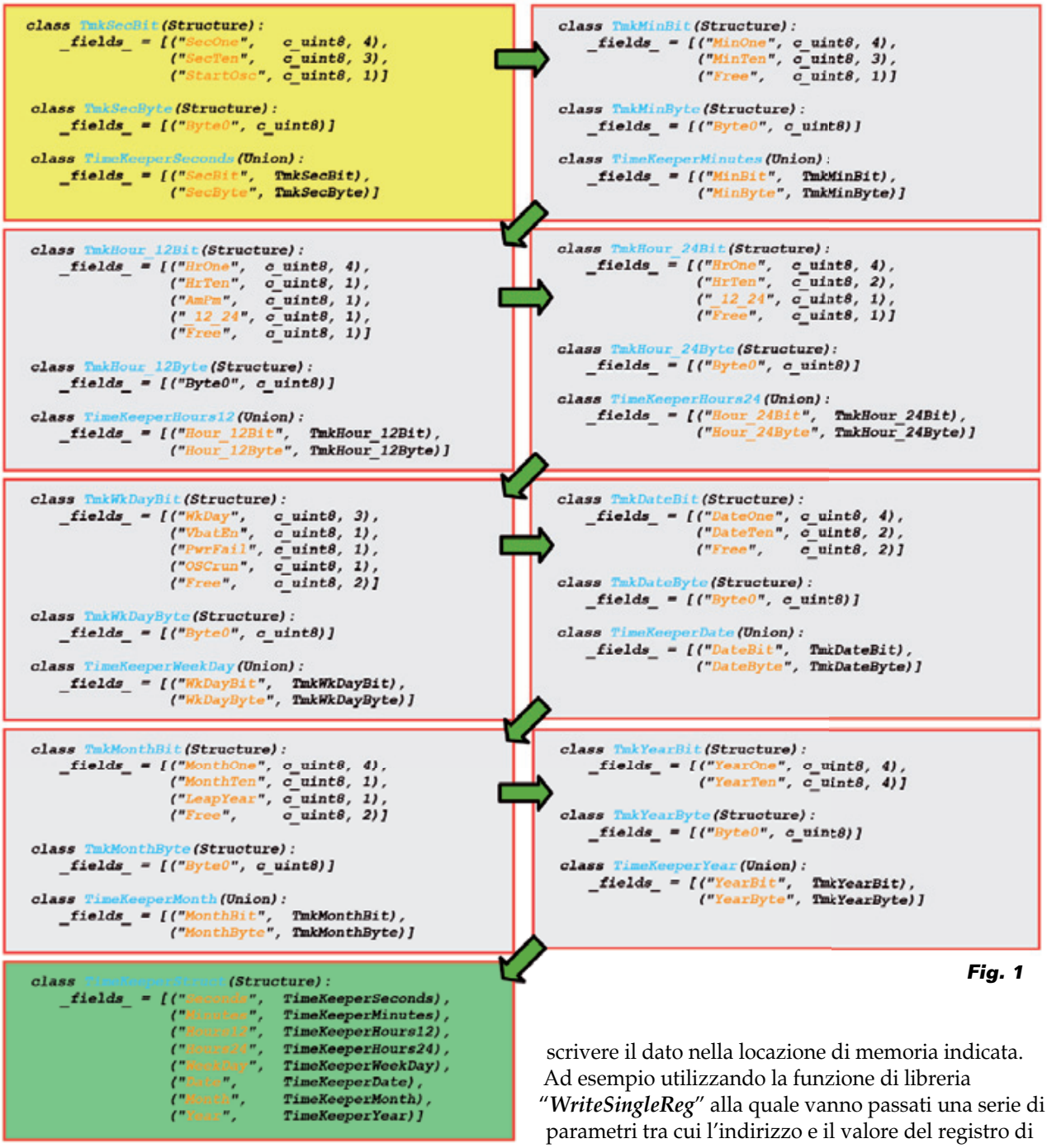


Fig. 1

Supponendo di volere abilitare l'allarme 0 si dovrà settare il bit "*Alarm0_Enable*" con la seguente sintassi:

```
MCP79410._CtrlReg.ControlBit.Alarm0_Enable = 1;
```

dove "*_CtrlReg*" è la dichiarazione globale della struttura dati in esame visibile dallo sketch. Ovviamente così facendo abbiamo solo settato un valore in RAM. Il passo successivo sarà quello di richiamare un'apposita funzione per andare a scrivere il dato nella locazione di memoria indicata. Ad esempio utilizzando la funzione di libreria "*WriteSingleReg*" alla quale vanno passati una serie di parametri tra cui l'indirizzo e il valore del registro di cui si vuole modificare il valore.
La sintassi completa sarà:

```
mcp79410.WriteSingleReg(RTCC_HW_ADD, CONTROL_ADD,
MCP79410._CtrlReg.ControlByte);
```

Oltre alla struttura dati appena illustrata ne seguono delle altre, più articolate, che permettono la configurazione dei registri TimeKeeper, gli allarmi e la lettura dei TimeStamp al PowerUp e PowerDown. A titolo d'esempio vediamo la struttura dati per la gestione dei registri TimeKeeper come già visto per

Listato 1

```
1)  ToggleSingleBit(ControlByte, RegAdd, Bit)
2)  SetSingleBit(ControlByte, RegAdd, Bit)
3)  ResetSingleBit(ControlByte, RegAdd, Bit)
4)  WriteSingleReg(ControlByte, RegAdd, RegData)
5)  WriteArray(ControlByte, StartAdd, Lenght)
6)  ClearReg(ControlByte, RegAdd)
7)  ReadSingleReg(ControlByte, RegAdd)
8)  ReadArray(ControlByte, StartAdd, Lenght)
9)  GeneralPurposeOutputBit(SetReset)
10) SquareWaveOutputBit(EnableDisable)
11) Alarm1Bit(EnableDisable)
12) Alarm0Bit(EnableDisable)
13) ExternalOscillatorBit(EnableDisable)
14) CoarseTrimModeBit(EnableDisable)
15) SetOutputFrequencyBit(OutputFreq)
16) StartOscillatorBit(EnableDisable)
17) Hour12or24TimeFormatBit(SetHourType)
18) AmPmBit(SetAmPm)
19) VbatEnBit(EnableDisable)
20) AlarmHour12or24TimeFormatBit(SetHourType, Alarm0_1)
21) AlarmAmPmBit(SetAmPm, Alarm0_1)
22) AlarmIntOutputPolarityBit(SetReset, Alarm0_1)
23) AlarmMaskBit(Alarm0_1, Mask)
24) ResetAlarmIntFlagBit(Alarm0_1)
25) PowerHour12or24TimeFormatBit(SetHourType, PowerDownUp)
26) PowerAmPmBit(SetAmPm, PowerDownUp)
27) ResetPwFailBit()
28) WriteTimeKeeping(SetHourType)
29) ReadTimeKeeping()
30) WriteAlarmRegister(Alarm0_1, SetHourType)
31) ReadAlarmRegister(Alarm0_1)
32) WritePowerDownUpRegister(PowerDownUp, SetHourType)
33) ReadPowerDownUpRegister(PowerDownUp)
34) Set_EEPROM_WriteProtection(Section)
35) WriteProtected_EEPROM(RegAdd, RegData)
```

Arduino (**Fig. 1**). Si noti la differenza di impostazione del codice Python (usato per Raspberry Pi) rispetto al codice C usato per Arduino: per ogni registro del TimeKeeper sono state definite due strutture dati seguite da una "***union***" per consentire l'accesso al singolo registro sia a livello di byte che di singolo bit; vedete, a riguardo, la prima struttura dati evidenziata in giallo e riguardante l'impostazione dei secondi più il bit di attivazione dell'oscillatore; seguono le altre strutture dati evidenziate in grigio.
Di queste sezioni ce ne sono otto solo per la gestione dei registri del TimeKeeper.
Tutte e otto le sezioni vengono poi raggruppate assieme in un'ulteriore struttura dati in modo da avere un unico accesso strutturato ai registri (sezione evidenziata in verde). Con questo approccio si ha un unico "*contenitore*" dal quale è possibile modificare un singolo bit o un intero byte dei registri relativi al TimeKeeper.
Per chiarire meglio il concetto facciamo qualche esempio, supponiamo di voler settare il bit "***StartOsc***" del registro "***RTCSEC***":

```
MCP79410._TimeKeeper.Seconds.SecBit.StartOsc = 1
```

dove "***_TimeKeeper***" è la dichiarazione globale della struttura dati in esame visibile dallo sketch.

Se invece si volesse settare il bit "***VbatEn***", per l'abilitazione della gestione a batteria dell'integrato, si dovrà procedere in questo modo:

```
MCP79410._TimeKeeper.WeekDay.WkDayBit.Vbaten = 1
```

Una volta configurati i registri della struttura dati sopra descritta si può procedere alla programmazione effettiva di ogni singolo registro dell'integrato, sfruttando la funzione di libreria "***WriteSingleReg***", cui bisogna passare tre parametri (indirizzo hardware, indirizzo del registro e valore da scrivere nel registro), oppure alla programmazione completa di tutti i registri del TimeKeeper sfruttando la funzione di libreria "***WriteTimeKeeping***" alla quale dobbiamo passare un solo parametro ad indicare se il formato di gestione dell'ora debba essere nel formato 12h o 24h. Quindi la sintassi per programmare i registri in un colpo solo con formato dell'ora in 24h sarà:

```
MCP79410.WriteTimeKeeping(0)
```

Le funzioni messe a disposizione dalla libreria sono le medesime di quelle della precedente per Arduino. Quindi eviteremo di descrivere ogni singola funzione ma ci limiteremo a farne un elenco per concentrarci sul codice e le sue funzionalità. L'elenco completo è illustrato nel **Listato 1**.
Ricordiamo che la variabile "***ControlByte***" identifica l'indirizzo hardware della periferica I²C in questo caso abbiamo due indirizzi uno per il RTCC e l'altro per la EEPROM, la variabile "***RegAdd***" identifica l'indirizzo del registro che si vuole leggere o scrivere, la variabile "***Bit***" indica quale bit del registro desiderato si vuole modificare mentre "***RegData***" indica il valore a 8 bit che si vuole scrivere nel registro.
Le variabili "***SetReset***" e "***EnableDisable***" servono rispettivamente per settare o resettare una funzione dell'integrato oppure per abilitare o disabilitare una funzione.
"***SetHourType***" serve per impostare il formato dell'ora (12H oppure 24H), "***SetAmPm***" serve per indicare se l'ora indicata è prima di mezzogiorno oppure dopo, "***Alarm0_1***" indica su quale allarme si vuole lavorare mentre "***Mask***" è la maschera di confronto per la generazione di una condizione di allarme.
Infine "***PowerDownUp***" indica se si vuole leggere/scrivere i registri con i riferimenti di data e ora rilevate durante le fasi di PowerUp o PowerDown.

LA LIBRERIA DIVENTA UN PACCHETTO DATI

Per agevolare l'utilizzo della libreria abbiamo creato un pacchetto di distribuzione detto "***package***". Con

questo accorgimento l'utente installa la libreria Python creata sul proprio Raspberry Pi, questa viene posizionata in un apposito percorso sul disco, per poi essere richiamata nei propri sketch con la funzione "***import***".
Per creare un pacchetto si deve organizzare il codice nel seguente modo:

1) creare una cartella base con un nome significativo. Nel nostro caso "***MCP79410***";
2) all'interno della cartella di cui sopra creiamone un'altra, sempre con nome "***MCP79410***" e all'interno di essa copiamo i nostri due file di libreria "***MCP79410.py***" e "***MCP79410_DefVar.py***".
3) creiamo un ulteriore file Python vuoto con il seguente nome "***__init__.py***"; in questo modo l'interprete Python vedrà questa cartella come un pacchetto di distribuzione (tecnicamente viene chiamato "***package***");
4) nella cartella base (vedere punto 1), creare il seguente file "***setup.py***", il quale deve contenere il seguente codice Python:

```
from distutils.core import setup

setup(name             = "MCP79410",
      version          = "1.0",
      description      = "MCP79410 Library",
      long_description = "This package is usefull
                          to manage the Real Time clock
                          Calendar MCP79410 developed
                          by Microchip",
      author           = "Matteo Destro",
      author_email     = "info@open-electronics.org",
      url              = "www.open-electronics.org",
      license          = "GPL",
      platform         = "RaspberryPi",
      packages         = ["MCP79410"])
```

Siccome quando si crea un file "***setup.py***" si deve sempre importare, dalla libreria "***distutils***", la voce "***setup***" e procedere successivamente alla compilazione di alcuni campi, nel nostro caso abbiamo dato un nome alla nostra distribuzione con la voce "***name***", ne abbiamo evidenziato la versione con la voce "***version***", ne abbiamo dato una breve descrizione con la voce "***description***" seguita da una descrizione più esaustiva con la voce "***long_description***". Segue la voce "***author***" e relativo indirizzo e-mail di supporto "***author_email***". Segue la voce "***url***" con il link dello sviluppatore e relativa licenza software "***license***". Infine la voce "***platform***" per cui è stato sviluppato il codice e la voce "***package***" ad identificare che quello in essere è un pacchetto di distribuzione. Per creare il pacchetto di distribuzione si deve eseguire il seguente comando:

```
sudo python setup.py sdist
```

che crea un file compresso contenente il pacchetto di distribuzione; il file creato è "***MCP79410-1,0.tar.gz***". Quindi l'utente che vorrà installare la libreria dovrà prima scompattare il file in un direttorio a piacere e poi eseguire il seguente comando per l'installazione della libreria:

```
sudo python setup.py install
```

Per importare e usare la libreria nei vostri sketch dovrete inserire in testa al file le seguenti righe di codice:

```
import sys
sys.path.insert(0, "/usr/local/lib/python2.7/dist-packages/MCP79410")
import MCP79410
```

UN ESEMPIO DI CODICE

Il codice che abbiamo realizzato ha due scopi distinti: il primo è mostrare come utilizzare la libreria messa a disposizione, nonché le procedure per la configurazione dei registri di interesse; il secondo è spiegarvi come interagire con l'hardware della nostra scheda demo, in quanto l'obiettivo è l'accensione e lo spegnimento automatizzato del Raspberry Pi in determinate condizioni dettate da come si sono configurati i registri degli allarmi.
Iniziamo la trattazione dal file principale ovvero quello che contiene la routine di "***main()***".
Facciamo notare che in testa al file ci sono le famose righe di codice con le istruzioni di "***import***" per inglobare la libreria appena trattata più altre librerie di sistema necessarie alla realizzazione dello sketch.
Subito dopo seguono le dichiarazioni delle variabili globali, la dichiarazione delle costanti che identificano i pin GPIO utilizzati e la loro configurazione come ingressi o uscite. Come ingressi abbiamo i soli pulsanti P1, P2 e P3 a cui viene imposto un pull-up interno più un evento sul fronte di discesa, con tempo di debouncing di 100 millisecondi, per intercettare la pressione del pulsante.
Per andare a leggere con cadenza regolare i registri di interesse del nostro MCP79410 abbiamo predisposto due timer, uno con periodo di 5 secondi (`def ReadsAndPrintsRegisters():`) e l'altro con periodo pari a 1 secondo (`def CheckAlarmsFlag():`).
Con cadenza pari a 5 secondi si vanno a leggere i registri del TimeKeeper, allo scopo di mostrare a video la data e l'ora correnti, i registri di allarme per visualizzarne la configurazione e, se necessario, i registri di TimeStamp. Questi ultimi vengono letti una sola volta all'accensione della nostra scheda Raspberry Pi.

Configurare **Raspberry Pi**

Se è la prima volta che utilizzate Raspberry Pi o non avete mai provato a scrivere del codice Python conviene eseguire una serie di passi per configurare Raspberry Pi in modo da utilizzare le librerie di gestione dei GPIO e del bus I²C. Vediamo passo-passo come procedere:

1) Per prima cosa installare la libreria Python per la gestione degli I/O di Raspberry Pi e se già installata provvedere ad aggiornarla all'ultima release disponibile che attualmente è la 0.5.11.
   - Per verificare l'attuale versione della libreria GPIO eseguire il seguente comando:

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

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

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

   oppure impartendo i seguenti:

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

2) Passiamo ora alla configurazione della periferica I²C messa a disposizione da Raspberry Pi:
   - procediamo con l'installazione della libreria Python "***smbus***" per la gestione del bus I²C:

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

   - seguita da quella dei tools I²C:

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

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

```
sudo raspi-config
```

   - come in **Fig. 3**, selezioniamo la voce "***Advanced Options***";
   - nella nuova schermata selezioniamo la voce ***"A7 I2C Enable/Disable automatic loading"***, come in **Fig. 4**;
   - nelle seguenti schermate rispondere in sequenza ***"Yes"***, ***"Ok"***, ***"Yes"*** e infine ancora ***"Ok"***;
   - usciamo quindi dalla finestra di configurazione selezionando la voce ***"Finish"***;

4) Procediamo con un reboot del sistema eseguendo il comando:

```
sudo reboot
```

   - dopo avere riavviato Raspberry Pi aprire con un editor di testo generico il file ***"modules"***;

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

   - Aggiungere, se mancanti, le seguenti due righe come evidenziato nella **Fig. 5.**

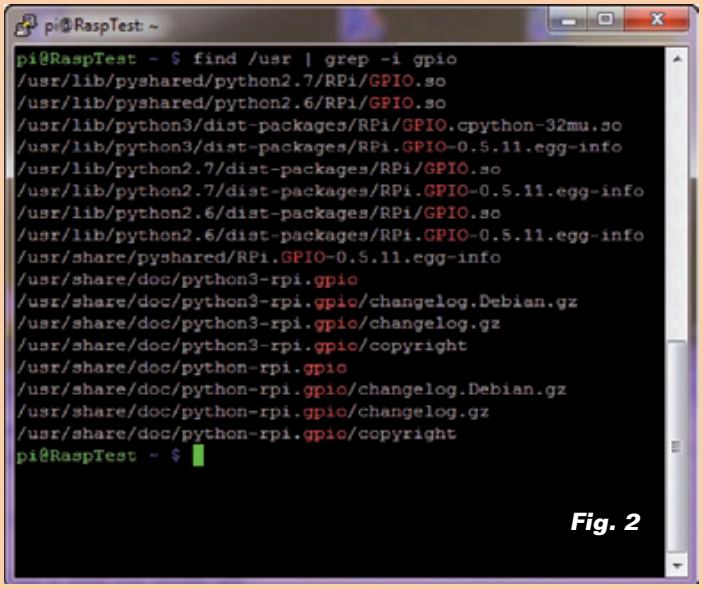


Fig. 2

Il secondo timer, impostato con cadenza pari a 1 secondo, serve per monitorare gli allarmi; nel caso in cui uno dei due allarmi sia attivo, si provvederà ad eseguire una delle due azioni di seguito elencate:

1) Azione accensione programmata scheda. L'allarme 0 è impostato per accendere a un dato orario l'elettronica della nostra scheda e di conseguenza anche la Raspberry Pi.
Quindi in questo caso lo sketch rileva che il flag dell'allarme 0 è a 1, forza l'alimentazione ad ON tramite la linea "*FORCE_ON*" e resetta il flag di allarme.
Prima di concludere il processo, disabilita l'allarme 0 e attiva l'allarme 1.

2) Azione spegnimento programmato scheda. L'allarme 1 è stato impostato per forzare lo spegnimento della scheda Raspberry Pi e quindi anche della nostra elettronica, a un determinato orario. In questo caso lo sketch rileva che il flag dell'allarme 1 è a 1 logico e di conseguenza abilita l'allarme 0 e disabilita l'allarme 1, resetta il flag pendente dell'allarme 1 e invia il comando di shutdown alla Raspberry Pi il quale inizierà la propria procedura di spegnimento. Concluso

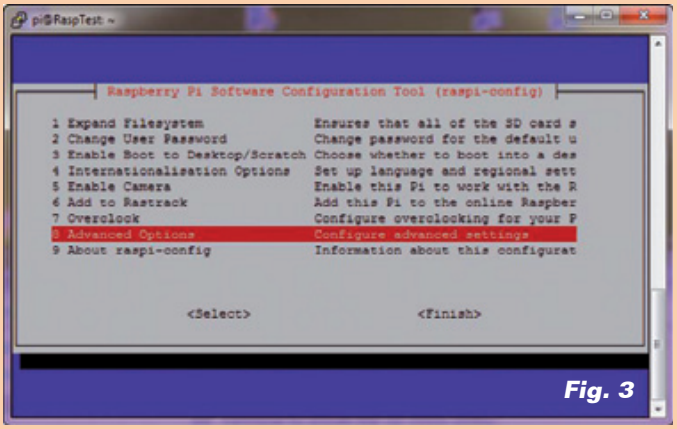


Fig. 3

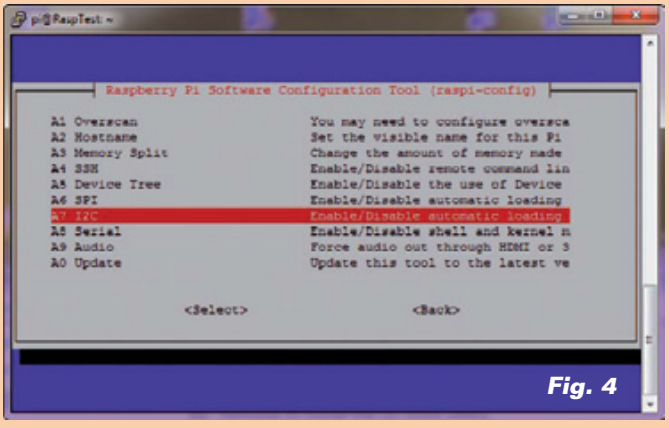


Fig. 4

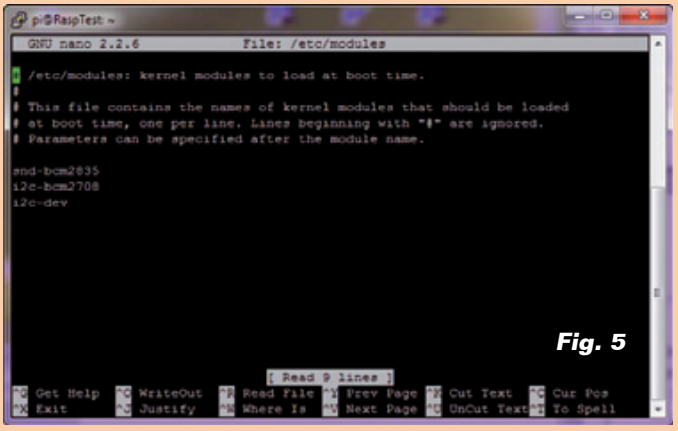


Fig. 5

Fig. 6

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

5) Per ultimo dovete verificare se nella vostra distribuzione è presente il file:

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

- Se dovesse essere presente apritelo con un editor di testo:

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

- Se presenti commentate, utilizzando il carattere **"#"**, le seguenti righe:

```
blacklist spi-bcm2708
blacklist i2c-bcm2708
```

6) A questo punto non rimane che riavviare nuovamente la Raspberry Pi e verificare che la comunicazione I²C sia attiva. Per fare ciò eseguite il comando:

```
sudo i2cdetect -y 1
```

Se tutto è andato a buon fine, il comando inviato ritorna una tabella come quella mostrata in **Fig. 6**. In pratica il comando analizza il bus I²C in cerca di periferiche e nel nostro caso ha individuato l'integrato MCP79410 con i suoi due indirizzi 0x57 e 0x6F.

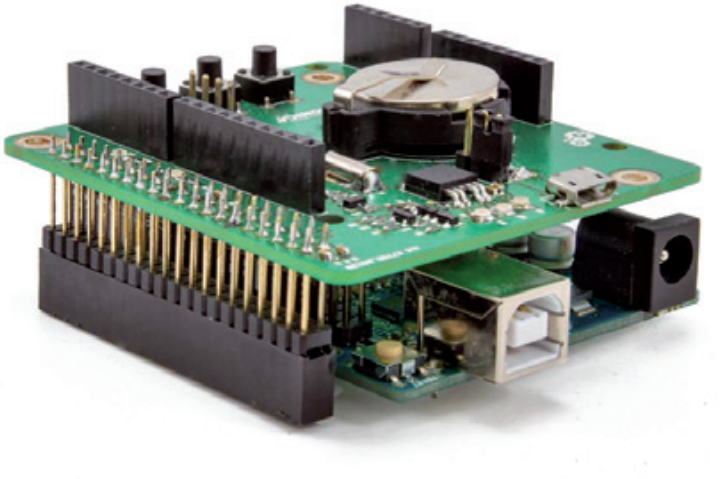
lo shutdown della Raspberry Pi, la linea "***FORCE_ON***" verrà rilasciata, con conseguente spegnimento della nostra elettronica.

I due time esaminati sono dei "***thread***" e per avviarli si deve usare la seguente linea di codice:

```
threading.Timer(1.0, CheckAlarmsFlag).start()
```

dove il primo parametro indica la cadenza, in questo caso 1 secondo, mentre il secondo parametro indica a quale funzione è associato. Il primo avvio avviene semplicemente richiamando la funzione "`CheckAlarmsFlag()`" dal main prima di entrare nel loop infinito.
Osserviamo ora, più nel dettaglio, il codice presente nei due timer: ad esempio quello in "`CheckAlarmsFlag()`". Si noteranno più richiami alla nostra libreria MCP79410, sia a livello di lettura della sola struttura dati sia a livello di chiamata di funzione. Ad esempio, si testano i flag dei due allarmi sfruttando la struttura dati:

```
if (MCP79410._Alarm.Alm[0].WeekDay.WkDayBit.AlarmIF == 1):
```



Questa struttura dati è un array di strutture perché gli allarmi sono due. Nel codice esposto, si punta all'allarme 0, in particolare al registro WeekDay (***ALM0WKDAY***) e al relativo flag di allarme (***ALM0IF***). Per i dettagli sui registri consultate il data-sheet. Oltre al test sopra esposto ci sono delle chiamate a funzione, come ad esempio "`MCP79410.Alarm0Bit(0)`" per disabilitare l'allarme 0 oppure "`MCP79410.ResetAlarmIntFlagBit(0)`" per resettare il flag dell'allarme 0.
Analizziamo ora la routine "***main()***": in testa ad essa si esegue una lettura della EEPROM interna all'MCP79410, esclusivamente per scopo didattico, e per fare ciò si sfrutta la funzione di libreria per leggere un array di byte a partire da un dato indirizzo. La funzione prevede tre parametri:

```
MCP79410.ReadArray(MCP79410.EEPROM_HW_ADD, 0, 13)
```

Il primo è l'indirizzo hardware che come già ampiamente detto è diverso da quello del RTCC, segue l'indirizzo di partenza da cui iniziare la lettura dei dati e infine il numero di byte che si vuole leggere. Il risultato della lettura viene messo in un array a cui si punta con la seguente:

```
MCP79410.DataArray[i]
```

dove "***i***" è l'indirizzo dell'array. La dimensione massima è impostata a 16 byte. Il contenuto viene poi stampato a video facendo comparire la scritta "***ElettronicaIn***". La scrittura in EEPROM è stata fatta con l'omonima funzione:

```
MCP79410.WriteArray(MCP79410.EEPROM_HW_ADD, n, m)
```

alla quale si passano gli stessi parametri della precedente. Segue l'attivazione dei due Timer, di cui abbiamo parlato prima, e infine il ciclo "***while***" infinito.
All'interno del ciclo "***while***" trovano posto la gestione dei tre pulsanti P1, P2 e P3 ai quali sono state assegnate tre funzioni differenti.
Il pulsante P1 serve per programmare la data e l'ora di MCP79410 e per fare ciò richiama una funzione apposita che legge data e ora di sistema e la trasferisce all'integrato. La funzione che si occupa di programmare data e ora di MCP79410 si chiama "`SetTimeKeeperByLocalDateTime("24H")`" a cui va passato un solo parametro, ovvero se il formato dell'ora debba essere 12h o 24h. Il parametro è di tipo stringa (la funzione è memorizzata nel file "***MCP79410_SetRegister.py***"). Se invece si vuole impostare la data e l'ora manualmente, va richiamata la seguente funzione:

```
ManualSetTimeKeeper(Hours, Minutes, Seconds,
Set_12H_24H, Set_AM_PM, Date, Month, Year, WkDay)
```

la quale richiede una serie di parametri. Un esempio di utilizzo potrebbe essere il seguente:

```
ManualSetTimeKeeper(15, 30, 00, "24H", "AM", 25, 12, 15, 5)
```

Così si imposta l'ora nel formato "***24H***" alle 15:30:00, la data è Venerdì 25/12/2015 e la dicitura "***AM***" rispetto a "***PM***" perde di significato in quanto il formato dell'ora è "***24H***".
Se fosse stata in "***12H***" avremmo dovuto passare i parametri nel seguente modo:

```
ManualSetTimeKeeper(03, 30, 00, "12H", "PM", 25, 12, 15, 5)
```

In entrambi i casi le funzioni riempono la struttura dati inerente i registri del TimeKeeper per poi richiamare la funzione di libreria che esegue la scrittura fisica sul dispositivo.
Il pulsante P2 serve per programmare gli allarmi 0 e 1; allo scopo richiama due funzioni distinte, una per l'allarme 0 e l'altra per l'allarme 1.
La configurazione è solo di tipo manuale e viene eseguita solo da codice; ciò significa che non abbiamo predisposto un sistema di inserimento dati da tastiera. La funzione da chiamare per impostare gli allarmi è la seguente:

```
ManualSetAlarm0(Index, Hours, Minutes, Seconds,
Set_12H_24H, Set_AM_PM, Date, Month, WkDay, AlarmMask, AlarmPol)
```

Come si può notare, ci sono parecchi parametri da

passarle; un esempio potrebbe essere:

```
ManualSetAlarm0(0, 12, 40, 00, "24H", "AM", 25, 12, 1, 1, "LHL")
```

Così facendo stiamo dicendo alla funzione di lavorare sull'allarme 0, l'ora è nel formato "***24H***" impostata alle 12:40:00. La data è impostata a 25/12/****. L'anno non è specificato perché privo di significato per gli allarmi. Il giorno della settimana è lunedì, la maschera per gli allarmi è fissata sui minuti e la polarità è "***Logic High Level***".
Anche in questo caso la funzione riempe una struttura dati per poi richiamare una funzione di libreria che esegue la scrittura fisica sul dispositivo. Suggeriamo di consultare il data-sheet per i dettagli sui registri.
Il pulsante P3 viene usato per disabilitare e resettare gli allarmi 0 e 1. Dopo il reset degli allarmi e la loro disabilitazione lo sketch si limita a mostrare la data e l'ora letta dal RTCC con cadenza di 5 secondi.
Per la stampa a video di tali informazioni si richiama una funzione di stampa contenuta nel file "***MCP79410_PrintFunc.py***" alla quale si deve passare una serie di parametri per decidere cosa stampare e cosa no.
La funzione in esame si chiama:

```
PrintDataMCP79410(PrintTimeKeeper, PrintAlarm0,
PrintAlarm1, PrintPowerDown, PrintPowerUp)
```

I parametri da passargli non sono altro che valori booleani -vero o falso- e dicono alla funzione cosa si vuole stampare a video.
Come suggeriscono i nomi, in sequenza abbiamo:

- stampa a video di data e ora;
- stampa a video impostazioni Allarme 0;
- stampa a video impostazioni Allarme 1;
- stampa a video valori di TimeStamp PowerDown;
- stampa a video valori di timeStamp PowerUp.

Se tutti i valori booleani sono posti a "***False***" niente verrà stampato a video.

AVVIO AUTOMATICO DELLO SKETCH DOPO IL BOOT

Il passo successivo consiste nel rendere auto-avviante il nostro codice dopo il boot del sistema operativo. Allo scopo, prima di tutto si deve decidere se si intende mantenere il login attivo oppure no; nel caso in cui si decida di saltare la fase di login si deve procedere così:

1) utilizzando il terminale, eseguire il seguente comando:

```
sudo nano /etc/inittab
```

2) cercare nel file la seguente riga:

```
1:2345:respawn:/sbin/getty -noclear 38400 tty1
```

3) commentarla utilizzando il carattere "**#**" in testa alla riga; sotto ad essa aggiungere la seguente ("***pi***" è lo user-name):

```
1:2345:respawn:/bin/login -f pi tty1</dev/tty1>/
dev/tty1 2>&1
```

4) uscire e salvare il file così modificato premendo "***ctrl+x***" e successivamente "***y***";
5) salvato il file, eseguire il seguente comando:

```
sudo nano /etc/profile
```

6) andare in fondo al file e aggiungere la seguente:

```
sudo python /home/pi/PythonProject/MCP79410_
Lib/WK/MCP79410_SetAndReadRegisters.py
```

Ovviamente come percorso dovete mettere quello del vostro sistema; nel nostro caso il path in cui si trovano i tre file dello sketch di esempio è "***/home/pi/PythonProject/MCP79410_Lib/WK/***". Alla fine riavviate il sistema.

CONCLUSIONI

Con questo abbiamo concluso la trattazione della gestione della nostra scheda con RTCC tramite Raspberry Pi. Vi abbiamo dato diversi spunti di riflessione per realizzare le vostre più disparate applicazioni sfruttando la nostra libreria Python appositamente scritta per l'occasione.
Non ci rimane che augurarvi buon lavoro!

per il MATERIALE

Lo shield RTC Raspberry-Arduino (cod. FT1254M) viene venduto montato e collaudato ed è disponibile presso Futura Elettronica al prezzo di Euro 20,00 IVA compresa.

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

NOVITÀ ASSOLUTA!

STAMPANTE 3D 40x40x40 cm

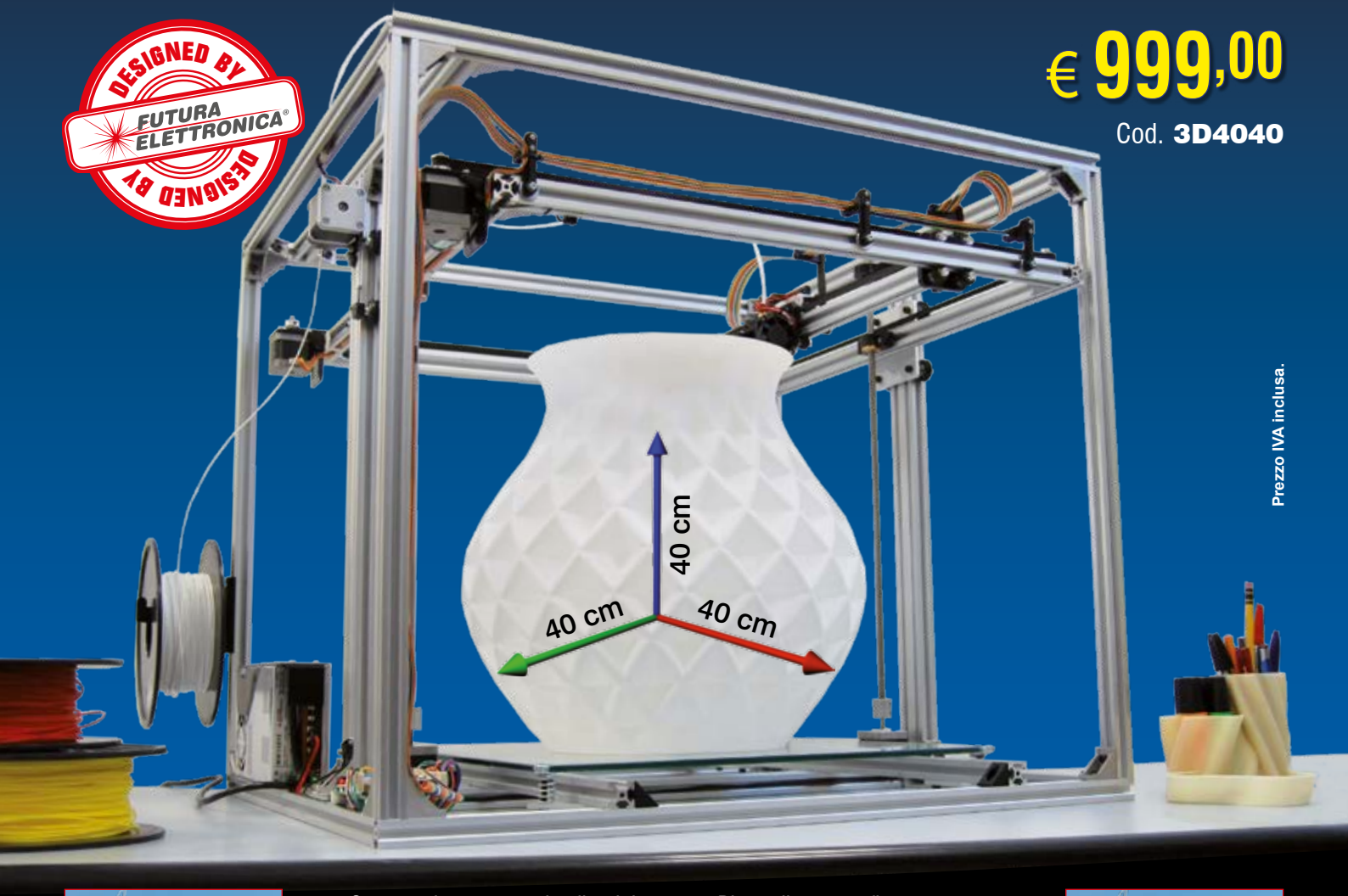
Nuovissima stampante 3D FDM (Fused deposition modeling) in KIT capace di produrre stampe di 40x40x40 cm (64.000 cm³)

DESIGNED BY FUTURA ELETTRONICA DESIGNED BY

€ 999,00

Cod. **3D4040**

Prezzo IVA inclusa.



- ✔ Struttura interamente in alluminio
- ✔ Area di stampa: X 40 cm, Y 40 cm, Z 40 cm
- ✔ Diametro filamento: 1,75 mm
- ✔ Tipo di filamento: ABS, PLA, NYLON ed altri ancora
- ✔ Diametro ugello fornito: 0,4 mm
- ✔ Diametro ugelli opzionali: da 0,3 mm a 0,8 mm
- ✔ Velocità di stampa massima: 300 mm/s (in funzione dell'oggetto da stampare)
- ✔ Piatto di stampa fisso: vetro temperato da 6 mm
- ✔ Riscaldatore per piatto di stampa: 40 x 40 cm – 12V/240 W con adesivo 3M (opzionale)
- ✔ Controllabile da PC o da modulo LCD (opzionale)
- ✔ Alimentazione tramite modulo switching 220 VAC/12 VDC 350 W
- ✔ Istruzioni di montaggio in italiano con illustrazioni

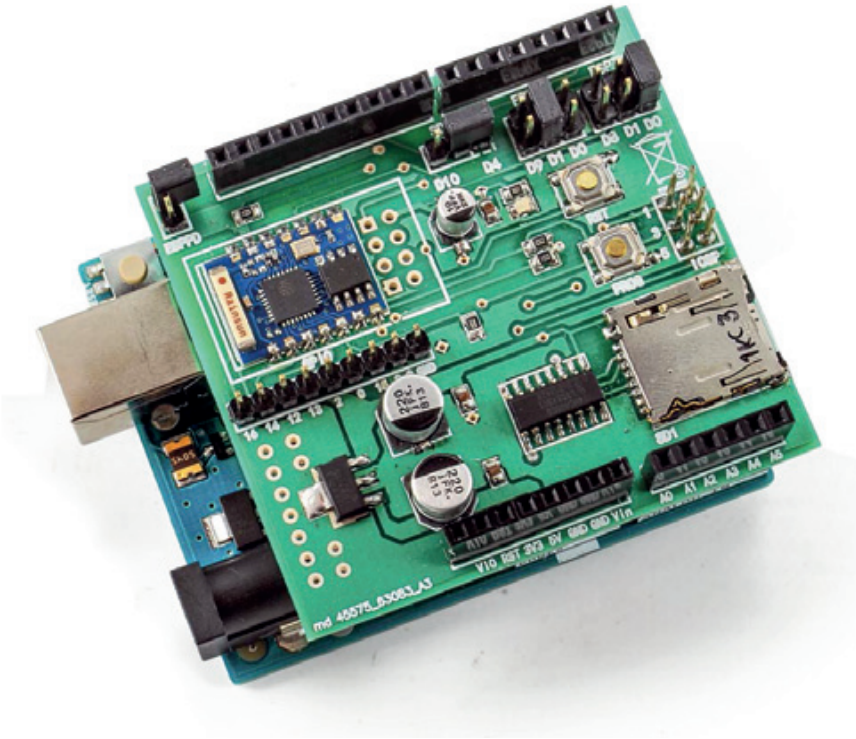
FUTURA ELETTRONICA®

www.futurashop.it

Futura Group srl
Via Adige, 11 • 21013 Gallarate (VA)
Tel. 0331/799775

Caratteristiche tecniche di questi prodotti e acquisti on-line su *www.futurashop.it*

Fornisce ad Arduino la connettività WiFi e il supporto di memoria su SD-Card; basato su un nuovo modulo interamente programmabile, offre il miglior rapporto qualità/prezzo con consumi di elettricità ridottissimi che lo rendono ideale per applicazioni a batteria.



WiFi SHIELD
CON ESP8266

dell'Ing. MIRCO SEGATELLO

Non è la prima volta che proponiamo uno shield dedicato alle connessioni in rete wireless e la ragione di ciò è duplice: da un lato c'è l'interesse del mondo Arduino per applicazioni che si connettano a smartphone e tablet o che possano accedere a Internet tramite WiFi e dall'altro c'è la fervente attività dei produttori che, con una buona frequenza, immettono sul mercato prodotti sempre più aggiornati e concorrenziali.
Lo shield che vi proponiamo in queste pagine si basa sul modu-

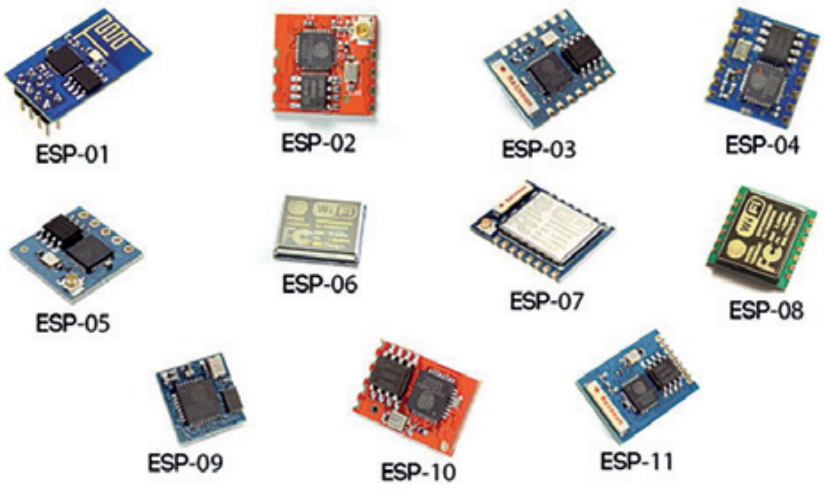


***Fig. 1** - La famiglia di moduli ESP.*

lo WiFi che abbiamo utilizzato nella demoboard ESP pubblicata sullo scorso numero della rivista. Rispetto a quelli utilizzati in passato (per esempio della Microchip), questo modulo, presenta un prezzo molto contenuto pur mantenendo caratteristiche di rilievo. Il modulo si chiama ESP03, viene commercializzato dalla Futura Elettronica (*www.futurashop.it*) ed è basato sull'integrato ESP8266, prodotto dalla Espressif Systems (*http://espressif.com*) per funzionare in modo autonomo all'interno di una rete Wi-Fi. Infatti l'ESP8266 contiene non solo lo stadio RF, ma anche un microcontrollore pienamente programmabile ed un registro cui fanno capo degli I/O, oltre a bus di comunicazione come l'SPI e l'I²C. Questo significa che il componente può da solo implementare la comunicazione in WiFi; inoltre, la possibilità di far eseguire al chip un programma, permette la gestione di alcuni pin (GPIO, bus vari) al fine di realizzare un sistema che possa interfacciarsi con periferiche esterne di vario genere o gestire ingressi e uscite.

L'INTEGRATO ESP8266

All'interno del chip ESP8266 troviamo un microcontrollore a 32 bit completo di ROM, RAM e SRAM per programma e dati; il chip dispone di linee di I/O digitali ed ingressi analogici, oltre che di porte di comunicazione come I²C, SPI e UART. Le porte di comunicazione possono funzionare da slave in modo da interfacciare memorie esterne al modulo ed ampliarne il funzionamento. Le linee digitali disponibili sono sedici, configurabili come ingressi o uscite, tutte con resistenza di pull-up e con la possibilità di generare un *interrupt*. Sono disponibili anche alcune linee di ingresso e di uscita analogiche basate su un convertitore sigma-delta a PWM.
Il chip è stato progettato per funzionare ad una tensione di 3,3V con un massimo ammissibile di 3,6V, gli ingressi digitali dispongono di un apposito circuito (overvoltage protection) per proteggere l'integrato da tensioni in ingresso superiori a 6V.
Per ultimo menzioniamo il supporto a tutti i protocolli di cifratura più moderni:WEP (RC4), CCMP (CBC-MAC, counting mode), TKIP (MIC, RC4) o WAPI (SMS4), WEP (RC4), CRC, WPA, WPA2 e WPS.
Per completare la descrizione del chip ESP8266 riportiamo il funzionamento dei singoli pin nella

Tabella 1. Le applicazioni cui il componente è destinato sono home automation, reti mesh, wireless industriale, baby monitor, telecamere di rete wireless, reti di sensori, elettronica indossabile, dispositivi di localizzazione wireless location-aware devices, security ID tag ecc.
Le caratteristiche e le funzioni implementate sono:

- compatibilità 802.11 b/g/n;
- WiFi Direct (P2P), soft-AP;
- supporto stack TCP/IP;
- TR switch, balun, LNA, amplificatore RF e accoppiamento alla rete integrati;
- PLL e gestore dell'alimentazione integrati;
- potenza in trasmissione in 802.11b di +19,5 dBm;
- sensore termico integrato;
- supporto per antenna diversity;
- CPU a 32 bit;
- SDIO 2.0, SPI, UART;
- STBC, 1x1 MIMO, 2x1 MIMO;
- A-MPDU, A-MSDU;
- solo 2 ms per connessione e trasferimento di pacchetti dati;
- corrente assorbita in stato di off minore di 10 µA;
- consumo in standby minore di 1 mW (DTIM3).

***Tabella 1** - Funzione dei pin del chip ESP8266.*

GPIO	INPUT	OUTPUT	PWM
GPIO0	sì	no	no
GPIO2	sì	no	no
GPIO12	sì	sì	sì
GPIO13	sì	sì	sì
GPIO14	sì	sì	no
GPIO15	sì	sì	sì

Siccome l'ESP8266 nasce per fornire la connettività WiFi in dispositivi mobili o indossabili alimentati a batteria, implementa modalità a basso consumo di energia che sono: *active mode*, *sle-*

ep mode e *deep sleep mode*, ognuna delle quali prevede l'attivazione di un certo numero di periferiche. Nella sleep mode il modulo radio è spento mentre rimangono attivi il timer e il watchdog e l'assorbimento è di soli 12 µA; in questa modalità è possibile programmare il timer interno affinché attivi il modulo radio a intervalli regolari per inviare o ricevere dati. All'occorrenza è possibile attivare il modulo radio e abilitare le trasmissioni, tramite segnali dall'esterno. Da software è anche possibile ridurre la potenza di emissione e ridurre i consumi in applicazioni in cui sono richieste comunicazioni a brevi distanze come in dispositivi indossabili. La linea CHIP_PD disponibile sul connettore esterno permette di portare il modulo nello stato di deep sleep mode con un assorbimento di corrente quasi zero. L'assorbimento di corrente durante l'active mode raggiunge un massimo di 215 mA alla massima potenza di trasmissione con protocollo 802.11b a 11 Mbps e scende a 145 mA con protocollo 802.11g a 54 Mbps (diviene soli 60 mA per l'invio di un pacchetto di 1.024 byte con potenza in antenna di -65dBm). Prima di procedere facciamo notare che esistono vari moduli WiFi basati sull'ESP8266, che si distinguono per form-factor; noi nel nostro shield prevediamo l'ESP03, ma è possibile montarvi anche l'ESP01. In alcuni moduli della serie ESP sono presenti dei LED: il LED rosso indica presenza della tensione di alimentazione (POWER STATUS) mentre il LED blu lampeggia quando il modulo avvia il boot (MODULE STATUS) o operazioni di comunicazione.

SCHEMA ELETTRICO

Andiamo ora a descrivere lo shield appositamente progettato per questo modulo. Oltre all'ESP03 (siglato U3 nel circuito), nello shield si trovano alcuni jumper per le impostazioni e un lettore di SD-Card multiuso (SD1), interfacciato tramite un adattatore di livelli 74HC4050 (U2) ad Arduino; l'adattatore serve a trasformare i segnali TTL delle linee di Arduino usate per la gestione della SD in 0/3,3V come previsto dalle card di memoria. Lo slot per la SD permette di memorizzare vari dati: ad esempio una pagina web, oppure le registrazioni di un data-logger sul web ed altro ancora.
Tutta la logica è alimentata con i 3,3V ricavati dal regolatore LDO U1, che partendo dai 5V forniti dalla omonima linea di Arduino

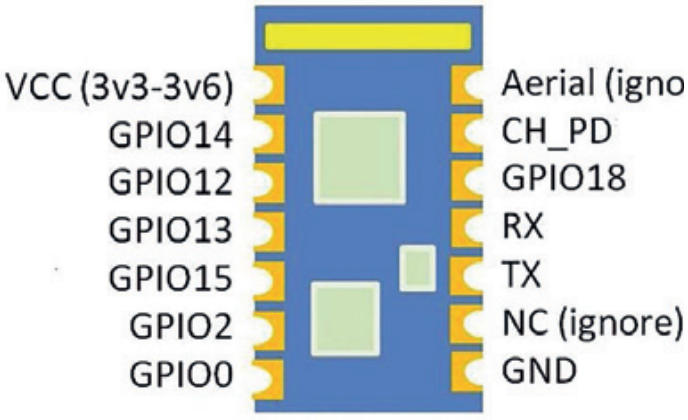


***Fig. 2** - Piedinatura del modulo ESP03.*

Un **integrato**, tanti moduli **WiFi**

In commercio si trovano varie versioni di modulo WiFi basato sul chip ESP8266 che si differenziano per le dimensioni ed i pin resi disponili sul connettore di comunicazione oltre a diverse versioni del firmware interno. Vediamo i principali.

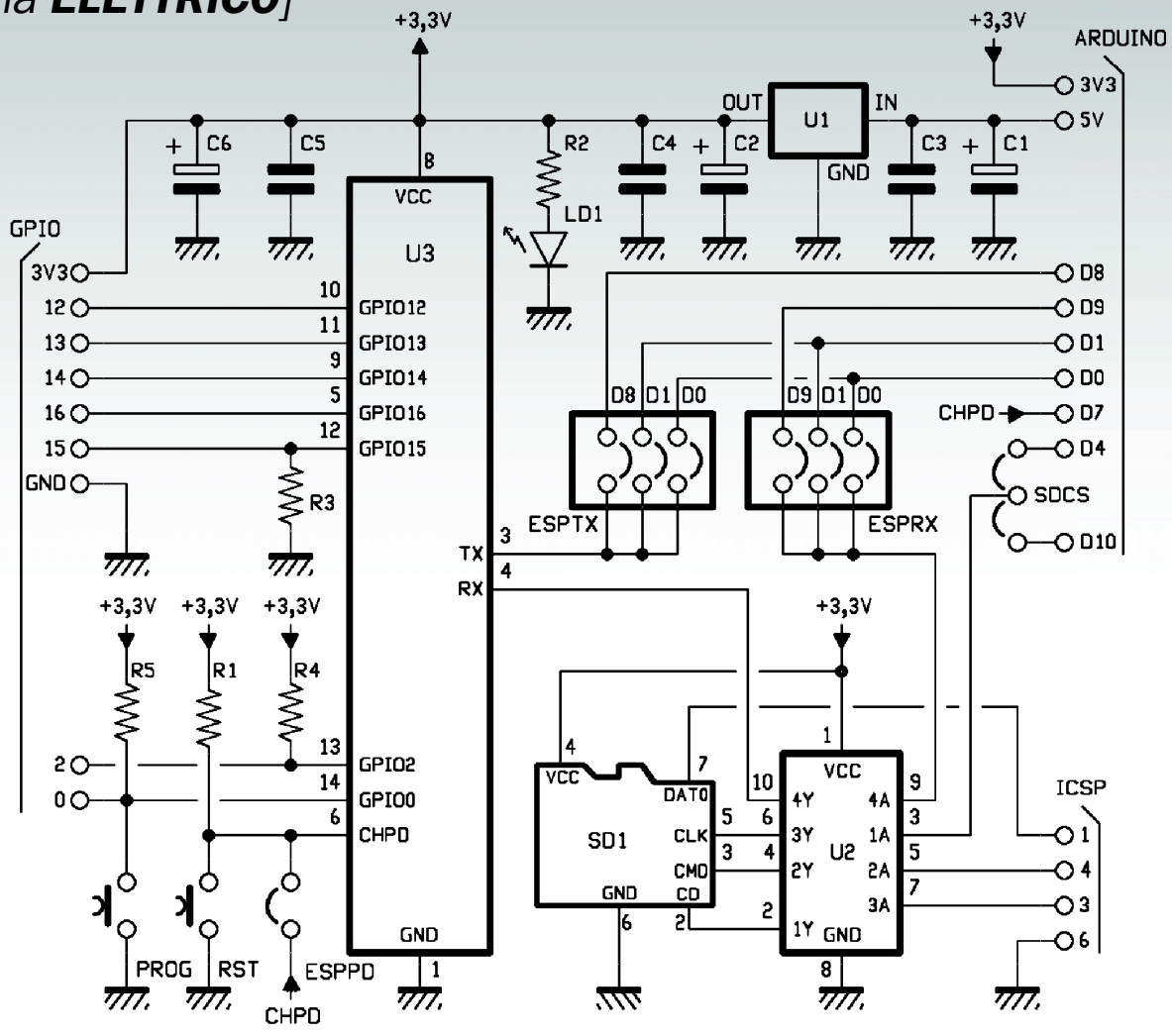
ESP-01: il più comune modulo con antenna integrata nel PCB
- piedinatura a 2x4 contatti a passo 2,54;
- antenna su PCB;
- 2 GPIO;
- UART URXD/UTXD;
- reset e funzione CH_PD (Power Down);
- dimensioni di 14,2 x 14,2 mm.

ESP-02: modulo con connettore per antenna esterna, compatibile con breadboard
- piedinatura 2x4 contatti a passo 2,54 DIP;
- 2 GPIO;
- UART URXD/UTXD;
- reset e funzione CH_PD (Power Down);
- dimensioni di 14,7 mm x 14,2 mm.

ESP-03: Versione SMT (Sufrace Mount Technology) con tutti i pin disponibili e antenna ceramica integrata
- 2 file di 7 contatti a passo 2,54 mm;
- antenna ceramica integrata;
- 7 GPIO;
- UART URXD/UTXD;
- reset e funzione CH_PD (Power Down);
- dimensioni 12,2 mm x 17,4 mm.

ESP-04: Versione SMT (Sufrace Mount Technology) con tutti i pin disponibili ma senza antenna
- 2 file di 7 contatti a passo 2,54 mm;
- antenna esterna;
- 7 GPIO;
- UART URXD/UTXD;
- reset e funzione CH_PD (Power Down);
- dimensioni di 12 mm x 15 mm.

ESP-05: Versione con sola comunicazione UART e connettore per antenna
- piedinatura a 2x4 contatti a passo 2,54 mm;
- connettore U-FL per antenna esterna;
- nessun GPIO;
- UART URXD/UTXD;
- reset e funzione CH_PD (Power Down);
- dimensioni di 14,2 x 14,2 mm.



ricava detta tensione, opportunamente filtrata dai condensatori C2, C4, C5, C6; il LED LD1, con in serie R2, indica la presenza dei 3,3 volt.

Il jumper SDCS (SD chip select) permette di selezionare il pin da usarsi per la gestione della SD; per impostazione predefinita, la libreria di Arduino utilizza il pin 4. Il jumper ESPPD (ESP Power Down), se cortocircuitato permette di selezionare il pin D7 per il controllo della modalità di funzionamento del modulo; in questo caso, tramite la linea D7 di Arduino è possibile accendere e spegnere il modulo secondo esigenza, con un considerevole risparmio energetico (diversamente, il modulo rimarrà sempre attivo). Nello shield sono disponibili, in un connettore strip maschio, tutti i pin del modulo ESP8266-03, utili per chi volesse gestire direttamente il modulo. Entrambi i moduli previsti

***Tabella 2** - Disposizione dei jumper per la comunicazione.*

COMUNICAZIONE DI ESP8266	JUMPER
seriale software pin 8 e 9 di Arduino	ESPTX=D8 e ESPRX=D9
UART di Arduino	ESPTX=D0 e ESPRX=D1
Comunicazione con PC tramite convertitore USB/seriale di Arduino	ESPTX=D1 e ESPRX=D0

***Tabella 3** - Collegamento hardware dello shield.*

PIN	FUNZIONE	PIN ARDUINO
SD_CS	SD chip select	D4 o D10
SD_MOSI	SD MOSI	D11
SD_MISO	SD MISO	D12
SD_CLK	SD clock	D13
ESPTX	ESP-8266 linea TX	D0 (RX0) o D1 (TX1) o D8
ESPRX	ESP-8266 linea RX	D0 (RX0) o D1 (TX1) o D9
ESPPD	ESP-8266 chip select (power down control)	3V3 o D7

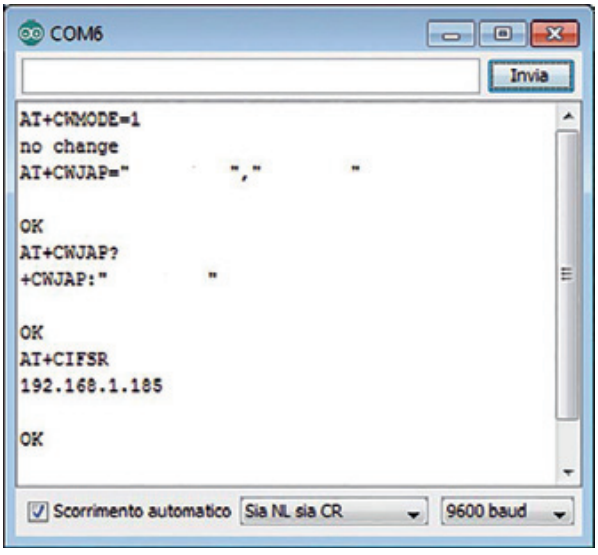
Comandi AT disponibili per il modulo ESP-8266

General functions
AT – Test AT start up Test
AT+RST – Restart module
AT+GMR T+GMR – View version info
AT+GSLP Enter deep-sleep mode
ATE – AT commands echo

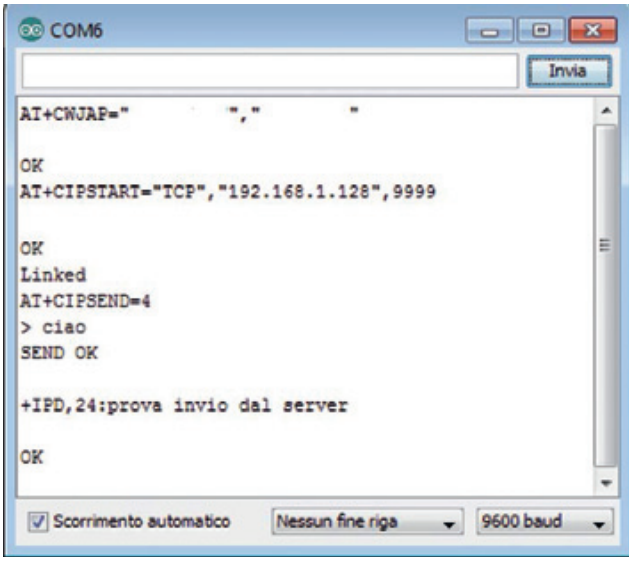
WiFi functions
AT+CWMODE – WIFI mode
AT+CWJAP – Connect to AP
AT+CWLAP – List available APs
AT+CWQAP – Disconnect from AP
AT+CWSAP – Configuration of softAP mode
AT+CWLIF – IP of stations
AT+CWDHCP – Enable/Disable DHCP
AT+CIPSTAMAC – Set mac address of station
AT+CIPAPMAC – Set mac address of softAP
AT+ CIPSTA – Set ip address of station
AT+ CIPAP – Set ip address of softAP

TCP/IP functions
AT+ CIPSTATUS – Information about connection
AT+CIPSTART – Start connection
AT+CIPSEND – Send data
AT+CIPCLOSE – Close TCP or UDP connection
AT+CIFSR – Get local IP address
AT+ CIPMUX – Enable multiple connections
AT+ CIPSERVER – Configure as TCP server
AT+ CIPMODE – Set transfer mode
AT+ CIPSTO – Set server timeout
AT+ CIUPDATE – Update through network
+IPD – Receive network data

dispongono di un'interfaccia di comunicazione seriale standard compatibile con Arduino, consistente nelle due linee dati TX e RX, le quali possono essere convogliate verso la seriale hardware, oppure sui pin D8 e D9 (gestibili con una seriale software). La selezione della comunicazione con U3 avviene tramite i jumper ESPRX (commutano la linea RX) ed ESPTX (commutano la linea RX) secondo la **Tabella 2**. Occorre, in ogni caso, considerare le limitazioni intrinseche di Arduino: utilizzando la seriale software di Arduino per comunicare con il modulo ESP03, la seriale hardware rimane disponibile per la programmazione e per le comunicazioni di debug, mentre utilizzando la seriale hardware di Arduino (modulo UART) la comunicazione con il PC creerà conflitti, quindi non sarà possibile programmare Arduino né inviare a Serial Monitor messaggi di debug. È anche possibile mettere in comunicazione il chip ESP8266 con il PC tramite il convertitore USB/seriale a bordo di Arduino; il microcontrollore Atmel328 non viene interessato dalla comunicazione ed il modulo UART deve essere disabilitato caricando uno sketch che non usa la seriale hardware (il modulo UART è disabilitato per impostazione predefinita).
Vediamo ora l'utilizzo pratico di questo shield partendo da alcune semplici applicazioni che ci permettono di prendere confidenza con le funzioni implementate.
La gestione del chip ESP8266 avviene con semplici comandi AT da inviare serialmente; anche l'invio e la ricezione dei dati avviene serialmente. Occorre però porre attenzione alla versione del modulo in possesso, in quanto i primi moduli commercializzati con firmware versione 080 comunicavano alla velocità di 115 kbps, mentre quelli più recenti (con firmware versione 09xx) funzionano a 9.600 bps. La possibilità di comunicare a una velocità di 9.600bps permette di gestire il modulo con la seriale software, facilmente implementabile con l'apposita libreria disponibile



Serial Monitor per connessione alla rete Wi-Fi di casa.



Serial Monitor per il test come client.

Classificazione delle **SD-Card**

In commercio esistono vari tipi di SDCard che si distinguono per la tecnologia impiegata, evolutasi negli anni a partire dalla SD base. Non tutte sono compatibili con le altre anche se tendenzialmente i costruttori tendono a mantenere la retrocompatibilità.
Di seguito sono descritte le tipologie una ad una.

SD
Capacità che vanno da 128MB a 2GB.
Formato predefinito: FAT16.
Le schede SD funzionano in tutti i dispositivi host che supportano SD, SDHC o SDXC.

SDHC
SD High Capacity (SDHC ™) è una scheda SD ™ basata sulla specifica SD 2.0.
Capacità che vanno da 4GB a 32GB.
Formato predefinito: FAT32.
Poiché SDHC funziona in modo diverso rispetto alle schede SD standard, questo nuovo formato non è compatibile con i dispositivi host che supportano solo SD (128MB – 2GB). La maggior parte dei lettori e dispositivi host costruiti dopo il 2008 dovrebbe essere compatibile con SDHC.
Per garantire la compatibilità, cercate il logo SDHC sia sulle schede che sui dispositivi host (fotocamere, videocamere, ecc).

SDXC
SD Extended (SDXC ™) è una scheda SD™ basata sulla specifica SDA 3.0.
Le capacità SDXC variano da 64 GB a 2 TB.
Formato predefinito: exFAT.
Poiché SDXC utilizza un file system diverso, chiamato exFAT e poichè il suo funzionamento è diverso rispetto alle schede SD standard, questo nuovo formato non è compatibile con i dispositivi host che supportano solo SD (128MB a 2GB). La maggior parte dei dispositivi host costruiti dopo il 2010 dovrebbe essere compatibile SDXC. Per essere sicuri della compatibilità, cercate il logo SDXC sia sulle schede sia sui dispositivi host (fotocamere, videocamere, ecc.).

NOTA: lettori di schede interne dei computer portatili a partire dal 2008 e prima potrebbero NON supportare le schede SDXC. Le schede SDXC funzionano nei lettori compatibili con SDHC (non i lettori SD) e se il sistema operativo del computer supporta exFAT.

nell'IDE di Arduino, lasciando libera la seriale hardware di comunicare con il PC, permettendo così una facile funzione di Debug.
Per questi primi esempi, in cui vogliamo prendere confidenza con il chip, faremo in modo da poter comunicare con il chip ESP direttamente dall'applicativo Serial Monitor di Arduino, lasciando inutilizzato il microcontrollore ATMEL328 della scheda.
Caricate su Arduino uno scketch che non usi la seriale, *Blink.ino* va benissimo, e impostate i jumper in modo che il modulo possa comunicare con il PC (riga 3 della **Tabella 2)**.
Aprite Serial Monitor di Arduino ed impostate una comunicazione a 9.600bps attivando sia il fine linea che il ritorno carrello (sia NL che CR) quindi inviate la stringa AT ed attendete risposta; se tutto è a posto il modulo risponde con "OK". Questo primo invio non fa assolutamente nulla ma permette di capire se il modulo è attivo e funzionante; all'accensione il modulo invia comunque la stringa "ready...".
Inviando la stringa AT+RST si esegue un reset software del dispositivo, il quale risponderà con la stringa "OK" seguita dal nome del fabbricante ed alcuni dati interni al chip. La stringa AT+GMR permette di conoscere la versione del firmware installato. Con questa configurazione il modulo ESP8266 potrà comunicare con qualsiasi applicativo del PC come Processing o qualsiasi monitor seriale.
Possiamo sempre più prendere confidenza con il modulo provando a connetterci alla rete Wi-Fi di casa; il comando AT+CWMODE=1 permette di impostare l'accesso ad un router, mentre il comando AT+CWJAP ="nomerete","password" ne permette l'accesso. Per sapere se la connessione è riuscita possiamo chiedere al modulo lo stato della connessione con il comando AT+CWJAP?, mentre per conoscere l'indirizzo IP assegnato dal router il comando è il seguente: AT+CIFSR.
Non appena connessi alla rete, possiamo testare alcune funzionalità del modulo facendolo ad esempio funzionare da Server nei confronti del PC; per fare questo è necessario avviare un semplicissimo client sul PC utilizzando ad esempio un software portable di nome SocketTest, una specie di monitor seriale ma per reti ethernet. Invieremo in sequenza i comandi:

```
AT+CWJAP="nomerete" , "password"
AT+CIPSTART="TCP" , "192.168.1.128", 9999
AT+CIPSEND=4
```

inviate una stringa di 4 caratteri come ad esempio "ciao" disabilitando l'invio del fine linea ed il ritorno carrello.
La ricezione di una stringa dal server sarà segnalata dalla stringa "+IPD" seguita dal numero di caratteri ricevuti e quindi il testo ricevuto.
Abbiamo provato diverse configurazioni, utilizzando il modulo sia come *client* che come *server* ed avviato anche una comunicazione UDP; in ogni caso è sempre stato semplice e veloce impostare il modulo con pochissimi comandi AT.
Nei vari listati disponibili, troverete sempre una prima parte relativa alla modalità di comunicazione con il modulo ESP ed alla modalità di *debug*, infatti in que-

sti primi esempi è sempre molto comodo prevedere una modalità per visionare i dati in transito dal modulo. Collegando il jumper ESPTX alla linea D8 e ESPRX alla linea D9 sarà possibile gestire il modulo utilizzando una seriale software, lasciando libero il modulo UART dell'ATmega328 di comunicare con il PC per scopi di debug e programmazione; questa modalità funziona solo con i moduli ESP che comunicano a 9.600bps, modalità che comunque presenta i suoi limiti qualora si comincino a gestire flussi dati importanti contenenti oltre un centinaio di caratteri, come ad esempio avviene con la gestione di pagine web.
Ricordiamo che tutti gli esempi proposti devono essere personalizzati introducendo il nome della vostra rete Wi-Fi e la password nei rispettivi campi SSID e PASS.
Abbiamo previsto, oltre alla classica applicazione come Web Server, anche un'applicazione come Web Client, per dimostrare la facilità con la quale possono essere richieste informazioni

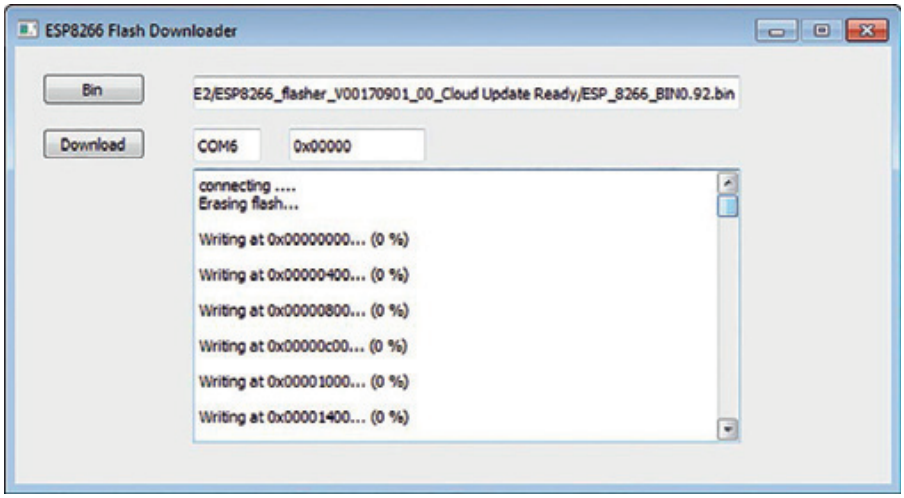


***Fig. 3** - Installazione nuovo firmware.*

ai siti internet. Gli unici dati da fornire sono l'indirizzo del server ed il nome della pagina alla quale accedere; nel listato scaricabile dal nostro sito trovate (come commenti) diversi esempi di impostazioni che abbiamo verificato personalmente, tra i quali l'interrogazione del server di Google oppure la lettura dei dati meteo dal server openweathermap. Quest'ultimo esempio è particolarmente interessante per la possibilità di conoscere le condizioni meteo interrogando la più vicina stazione meteo senza

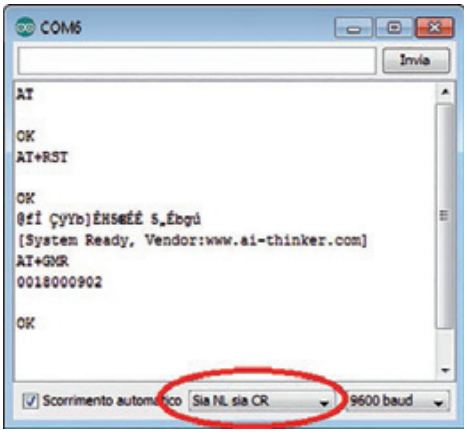
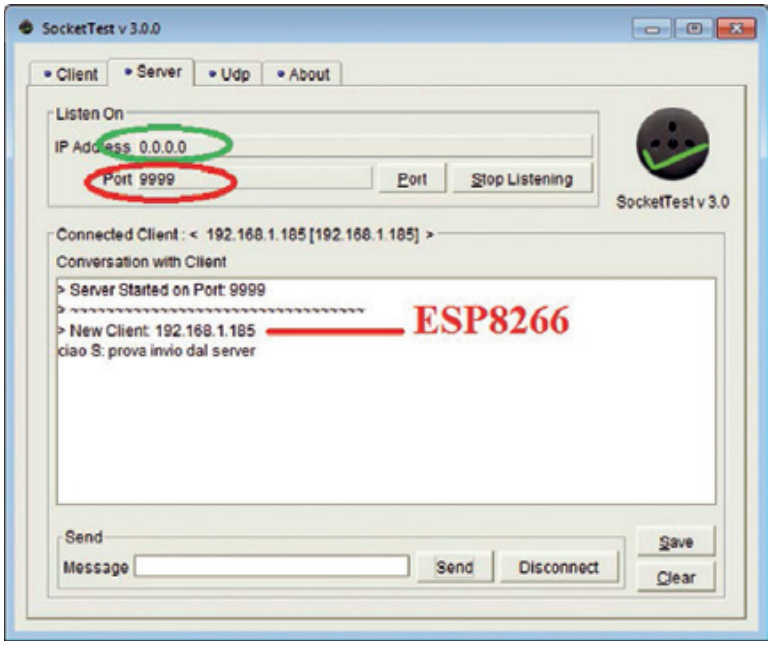


***Fig. 4** - Serial Monitor delle comunicazioni con ESP.*

Server con ScckeTest.



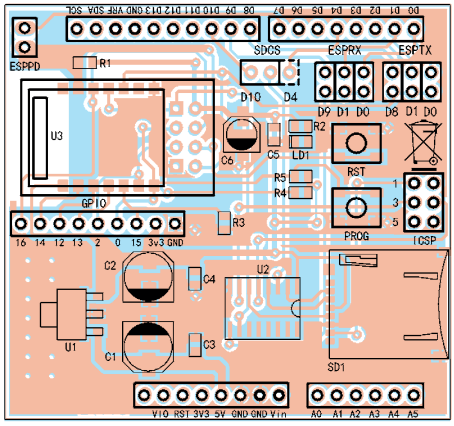
la necessità di utilizzare alcun sensore. Esiste anche la possibilità di richiedere le previsioni meteo per i prossimi giorni; l'unico dato da inserire è il codice della città di cui si vogliono i dati ed è reperibile a questo indirizzo: *http://openweathermap.org/help/city_list.txt*. Similmente, è possibile interrogare il server di Yahoo per conoscere la data e l'ora attuale.
In questi ultimi esempi abbiamo usato la sintassi classica html che è composta da tre sezioni: request (riga di richiesta), body (corpo del messaggio) e header (informazioni aggiuntive). La sezione request a sua volta si compone di tre campi: il primo è

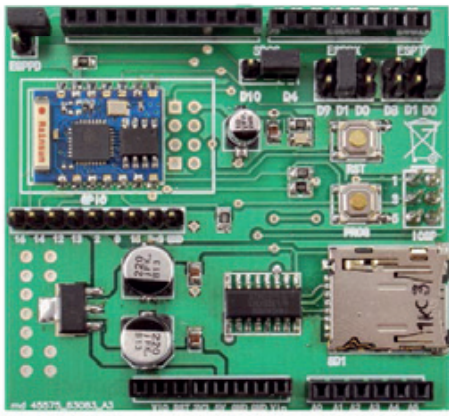
[piano di **MONTAGGIO**]

Elenco Componenti:

R1: 10 kohm (0805)
R2: 330 ohm (0805)
R3÷R5: 10 kohm (0805)
C1, C2: 220 µF 6,3 VL elettrolitico (Ø6mm)
C3÷C5:100 nF ceramico (0805)
C4, C5: 100 nF ceramico (0805)
C6: C6: 22 µF 6,3 VL elettrolitico (Ø4mm)
U1: TC1262-3.3VDB
U2: 74HC4050D
U3: ESP03
RST: Microswitch
PROG: Microswitch
SDA: Connettore micro SD-Card
LD1: LED verde (0805)

Varie:
- Strip maschio 2 vie
- Strip maschio 3 vie (5 pz.)
- Jumper (4 pz.)
- Strip maschio 9 vie
- Strip Maschio/Femmina 3 vie (2 pz.)
- Strip Maschio/Femmina 6 vie
- Strip Maschio/Femmina 8 vie (2 pz.)
- Strip Maschio/Femmina 10 vie (1 pz.)
- Circuito stampato S1192





il metodo di accesso e può essere di tipo GET (recupero informazioni), POST (invio informazioni), HEADER (come GET, ma riceve solo l'intestazione della pagina); il secondo è l'URI (Uniform Resource Identifier) cioè l'indirizzo della pagina cui si vuole accedere e il terzo è la versione del protocollo (nel nostro caso HTTP/1.1). La sezione body contiene semplicemente la richiesta vera e propria, mentre la sezione header contiene nel nostro caso l'header "HOST:". Ecco che ad esempio la richiesta delle condizioni meteo può essere fatta inviando la richiesta: "GET /data/2.5/weather?id=3164699 HTTP/1.1 HOST: api.openweathermap.org" La richiesta deve essere inviata dopo che, ovviamente, ci saremo connessi al server con l'istruzione: "AT+CIPSTART = \"TCP\",\""DEST_HOST"\" , "DEST_PORT"\r\n".

Tutti gli esempi sono stati testati con moduli aventi firmware versione 09.02, che è la più stabile ed esente da *bug*; attualmente non esiste ancora una versione definitiva del firmware ed in rete si possono trovare versioni anche più recenti ma ancora tutte da testare. È comunque possibile installare una versione differente del firmware con una semplice procedura che ora descriveremo. Tra i file di questo articolo troverete una cartella di nome *Firmware*, che contiene un eseguibile di nome *ESP8266_flasher.exe*, utilizzato per caricare un nuovo firmware sul modulo; abbiamo messo a disposizione anche il firmware di versione 09.02 che normalmente trovate già installato nei moduli ESP8266.

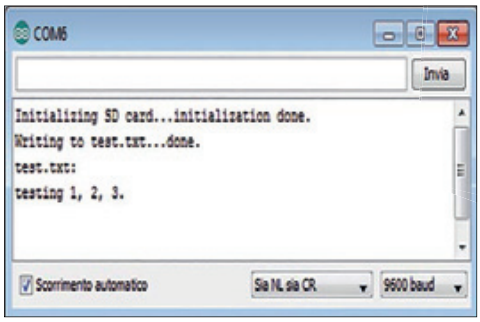
Una volta avviato il software (non richiede installazione) è sufficiente specificare la porta COM utilizzata per la comunicazione con il modulo ed aprire il file .bin, per sapere quale seriale utilizzare potete usare le informazioni fornite dall'IDE di Arduino quando è connessa la scheda Arduino.

Ponete il modulo nella modalità di comunicazione con il PC ed abilitate la modalità di boot mode del modulo ponendo la linea GPIO0 a massa, premendo e mantenendo premuto il pulsante PROG, dopodiché riavviare il modulo ponendo per un attimo la linea CHPD a livello basso tramite il pulsante RST. Dopo il riavvio, il modulo si pone in modalità di boot mode, pronto a ricevere il nuovo firmware, cliccate quindi sul pulsante Download per attivare il caricamento.

Appena avviato il download del firmware, potete rilasciare il pulsante PROG e attendere la conclusione dell'operazione, che richiede in tutto alcuni minuti.

Utilizzate il comando AT+GMR per verificare se il modulo funziona correttamente e quale versione del firmware risulta installata.

La disponibilità di una SD per salvare dei dati ci permette di realizzare un interessante data



Web server con ESP8266.

logger che dispone della funzionalità di accesso alla rete Internet per reperire informazioni utili. Nel caso di un data logger un dato sicuramente importante è la data e l'ora in cui avviene il salvataggio dei dati, funzione di solito svolta da un RTC (Real Time Clock) interfacciato con Arduino. Disponendo di un accesso ad Internet, è semplice interrogare un server NTP per conoscere data e ora precisi senza bisogno di eseguire alcuna impostazione iniziale. Una prima semplice prova per verificare se tutto funziona correttamente consiste nell'inserire nello slot una SD (micro SD per la precisione) ed eseguire lo sketch *cardInfo.ino* disponibile negli esempi di sistema; assicuratevi che il jumper SDCS sia nella posizione D4. Potete vedere nelle immagini pubblicate la risposta visualizzata su Serial Monitor relativa ad una SD HC cat4 da 8GB formattata FAT32 utilizzata per le nostre prove.

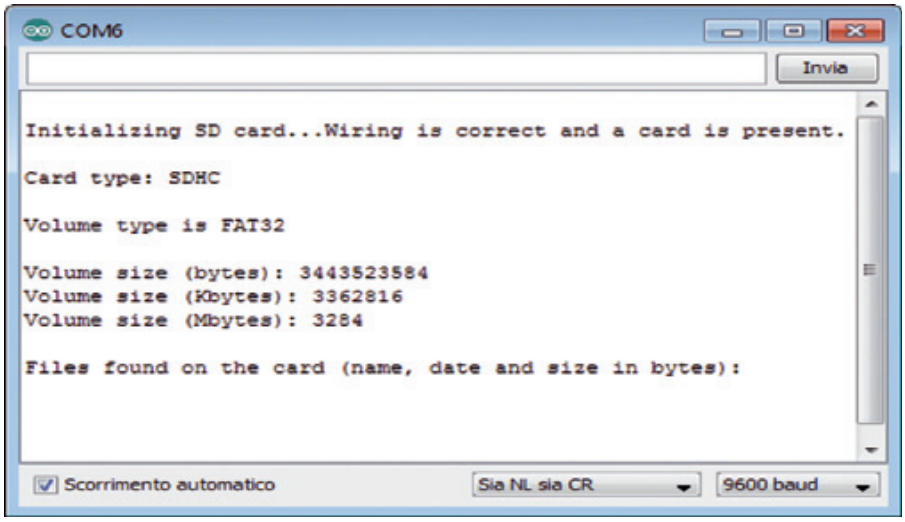


***Fig. 5** - Test SD*

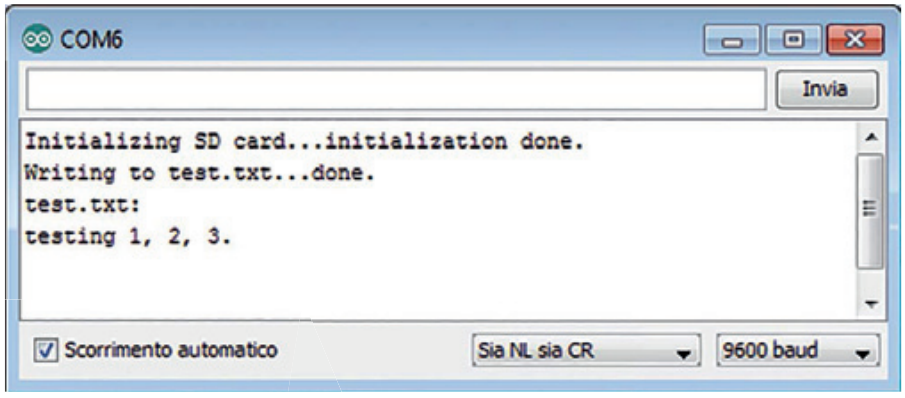
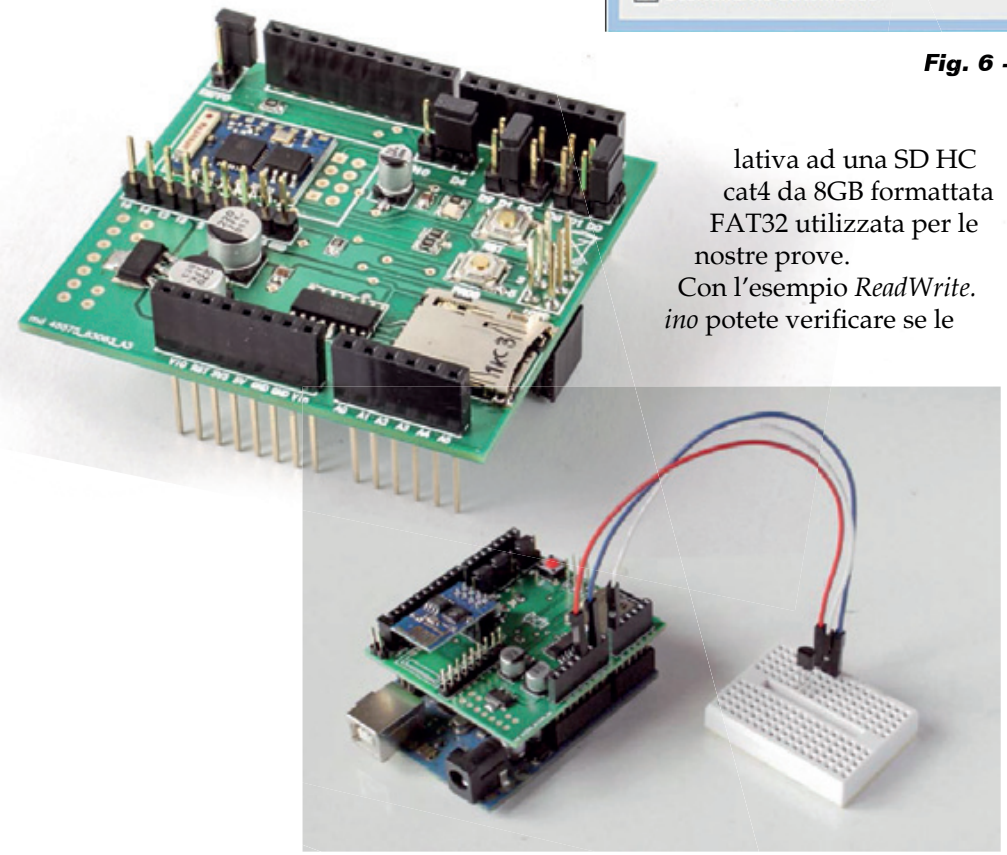
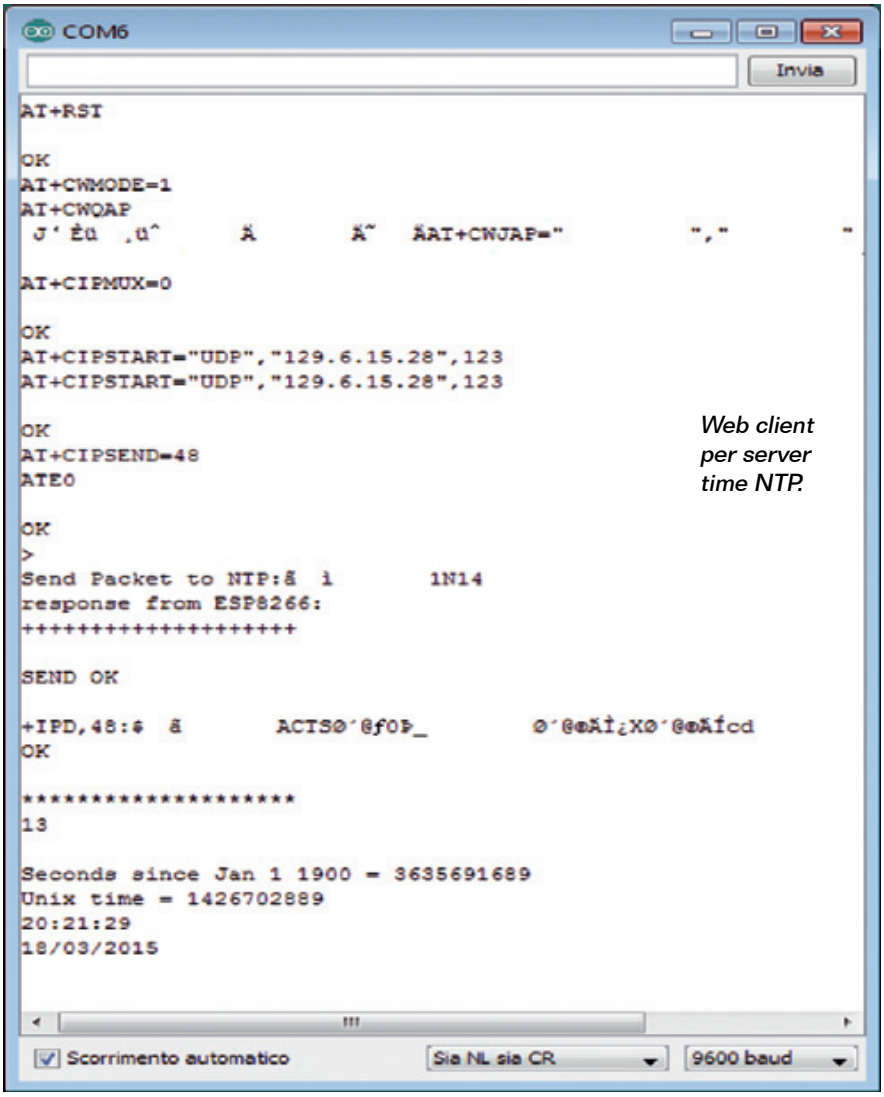


***Fig. 6** - Test scrittura/lettura su SD.*



Con l'esempio *ReadWrite.ino* potete verificare se le funzioni di lettura funzionano correttamente.

A questo punto siamo pronti per implementare il nostro data logger; come semplice esempio andiamo a salvare il valore della temperatura ambiente misurata tramite un semplicissimo integrato come l'LM35. Lo sketch per la funzione di data logger si chiama *ESP8266_DataLogNTP.ino* e può essere scaricato assieme ai file allegati di questo progetto. Per testare questo programma dovete per prima cosa inserire il nome e la password della vostra rete Wi-Fi nei campi SSID e PASS. Il sensore di temperatura andrà alimentato da Arduino tramite i pin GND e +5V mentre la sua uscita va connessa all'ingresso analogico AN0. Questo



Web client per server time NTP.

sensore fornisce una tensione di 10 mV ogni grado centigrado rilevato pertanto, per migliorare la risoluzione, si rende necessario impostare il fondo scala del convertitore A/D al valore di 1,1 volt tramite l'istruzione analogReference(INTERNAL). La temperatura sarà ricavata convertendo il valore letto dal convertitore A/D tramite l'equazione TEMP = analogRead(A0)/9.31. Per questo sketch abbiamo previsto l'utilizzo della linea ESPPD (ESP8266 Power Down) che consente di spegnere il modulo nei momenti di inutilizzo riducendo drasticamente i consumi.
Nella parte iniziale dello sketch è possibile definire la modalità di comunicazione tra il modulo ESP8266 ed Arduino. La modalità di default che abbiamo previsto consiste nel far comunicare il microcontrollore ATmega328 con il modulo Wi-Fi tramite una seriale software e di lasciare la seriale hardware per la funzione di debug. La variabile DEBUG può esse impostata su true o false a seconda se volete un report più o meno completo nei confronti delle comunicazioni con il modulo Wi-Fi. Nel caso vogliate gestire il modulo Wi-Fi con la seriale hardware è sufficiente porre i jumper in modo che i segnali ESPTX e ESPRX siano connessi rispettivamente a D0 e D1, ma ricordatevi che ogni volta vorrete programmare Arduino dovrete sconnettere il modulo rimuovendo i jumper, altrimenti le due comunicazioni andranno in conflitto. Nel caso siano disponibili due seriali hardware come nella board Arduino Leornardo, la seriale principale sarà utilizzata per il debug mentre la seriale secondaria (Serial1) sarà usata per la comunicazione con il modulo. Nella sezione relativa al setup viene inizializzato il modulo e la SD verificando che siano operativi. Nella sezione loop è contenuto il codice relativo alla funzione di data log, la prima parte si occupa di richiamare la funzione per l'accesso alla rete Wi-Fi; sono previsti cinque tentativi massimi. Segue la richiesta della data e dell'ora richiamando la funzione requestTime() che si occupa di interrogare il server NTP, le stringhe relative alla data e all'ora sono disponibili tramite le funzioni getDateString() e getTimeString(). Queste funzioni permettono la formattazione di dati e possono essere modificate a seconda di quale formato si vuole utilizzare; questo è importante nel caso di una successiva elaborazione tramite foglio di calcolo dei dati acquisiti.
Per estrapolare la data e l'ora dalla stringa di ritorno utilizziamo la libreria time appositamente studiata per questa funzione; come al solito è sufficiente copiare l'intera cartella (di nome time) all'interno della cartella library di Arduino e quindi riavviare l'IDE. Occorre ricordare che il tempo ricavato dal server NTP fa riferimento al meridiano di Greenwich (ora GMT Greenwich Mean Time) occorre quindi tenere conto del fuso orario Italiano (+1) e dell'eventuale ora legale (+1 nel periodo estivo), per questo abbiamo previsto la variabile di correzione UTCcorrection che

viene impostata a 7200 per tenere conto dei 3600 secondi relativi al fuso orario e dei 3.600 secondi relativi all'ora legale. Segue la sconnessione dalla rete Wi-Fi e lo spegnimento del modulo per risparmiare corrente. Le seguenti righe di codice vengono utilizzate nello sketch per costruire la stringa che verrà scritta nella SD card:

```
String dataString = getDateString();
 dataString += " ";
 dataString += getTimeString();
 dataString += "   AN0=";
 dataString += String(analogRead(A0));
```

Questa stringa potrà essere modificata secondo necessità aggiungendo o meno altri valori e altri separatori. Segue la scrittura sulla SD ed il ciclo a vuoto per l'attesa di una nuova acquisizione. L'intervallo di tempo tra le acquisizioni è impostabile tramite la variabile timeToLog che, per impostazione predefinita, vale 60.000 ms (1 minuto). Abbiamo previsto che il modulo rimanga acceso solo per il breve lasso di tempo necessario a connettersi alla rete e ricevere i dati dal server NTP, per tutto il resto del tempo tra un log ed il successivo il modulo è portato nello stato di power-down, grazie al segnale del pin D7 che comanda il pin CH_PD dell'integrato EPS8266. Questo permette di avere un considerevole risparmio energetico anche se sono necessarie continue connessioni alla rete Wi-Fi. Nella SD vi ritroverete un file di nome DATALOG.TXT contenente i valori acquisiti, come visibile nella **Fig. 8**. Questo sketch è compatibile con moduli in cui è installato il firmware versione 9.02 e 9.03 come quello montato nel nostro shield.

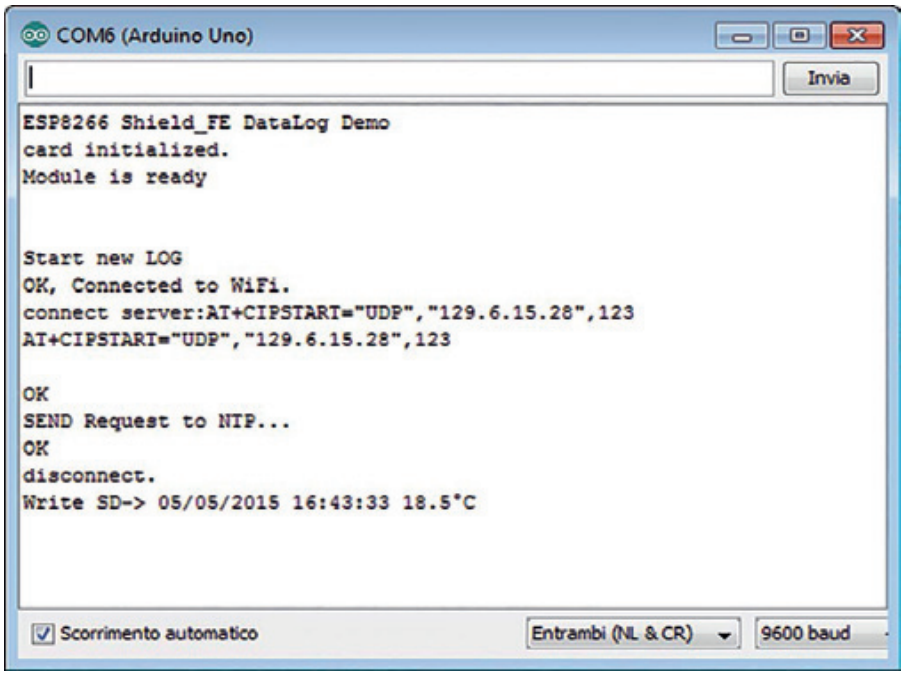


***Fig. 7** - Report relativo allo sketch.*

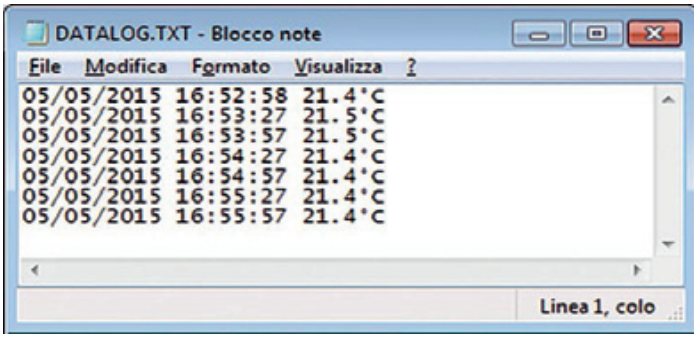


***Fig. 8** Contenuto del file di log dello sketch.*

CONCLUSIONI

Possiamo definire l'ESP un piccolo grande modulo vista la versatilità ed il package particolarmente ridotto, il cui punto di forza è sicuramente il prezzo bassissimo. Anche acquistato già installato in un comodo shield il costo finale è decisamente più contenuto dei classici shield ai quali eravamo abituati. Grazie al rilascio dell'SDK da parte dell'azienda produttrice la comunità si è mobilitata per trovare nuovi ed avanzati modi di utilizzare questo modulo, primo fra tutti la possibilità di utilizzarlo in modalità stand-alone, senza bisogno di interfacciarlo con logiche programmabili esterne.

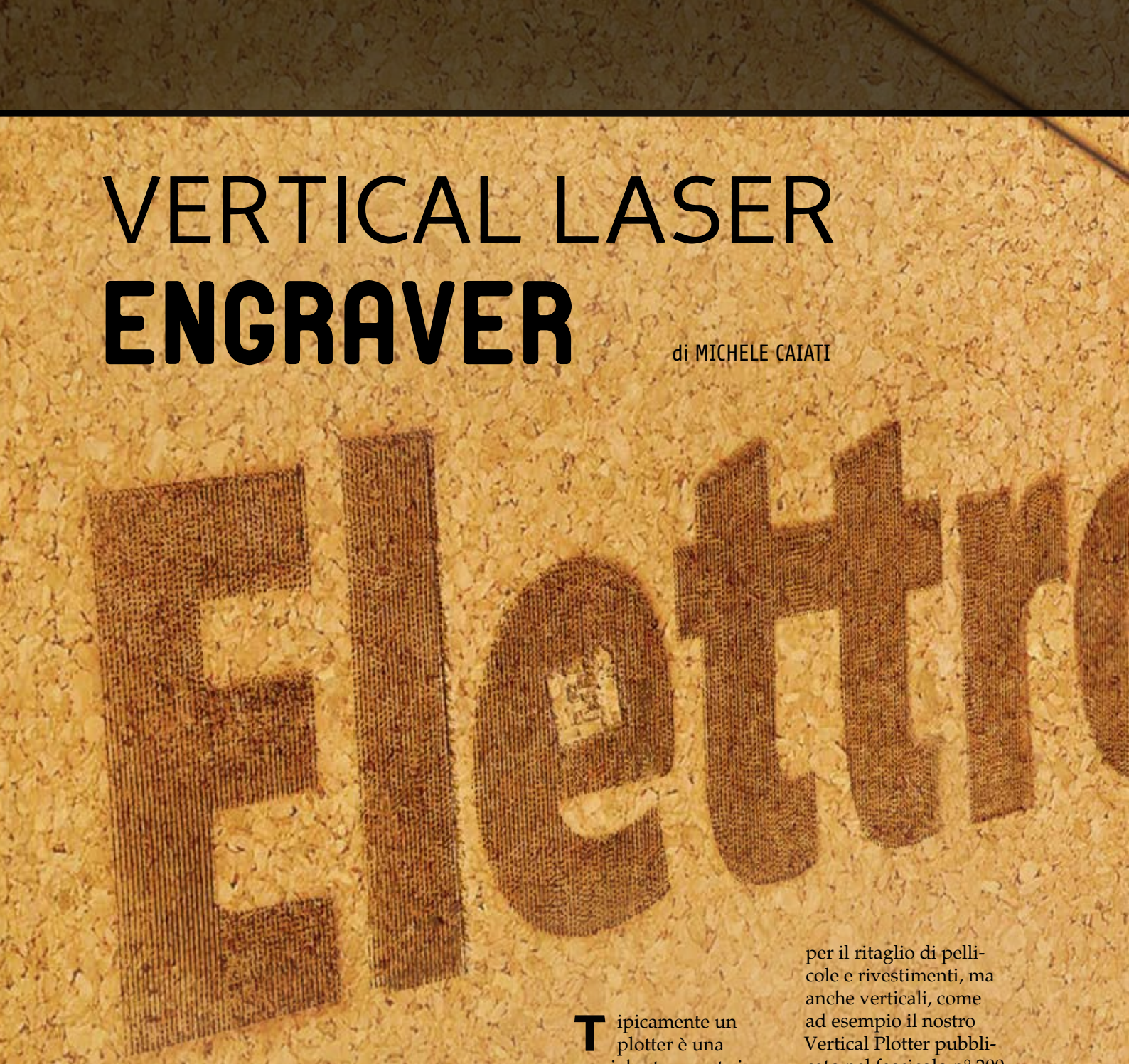
per il MATERIALE

Tutto il materiale di questo progetto può essere acquistato presso Futura Elettronica. La scheda Wi-Fi per Arduino con modulo ESP03 (cod. FT1192M) viene venduta al prezzo di Euro 27,00.
I moduli WiFi, ESP03 e ESP8266, sono disponibili anche separatamente rispettivamente al prezzo di Euro 8,00 e Euro 9,00. Tutti i prezzi si intendono IVA compresa.

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

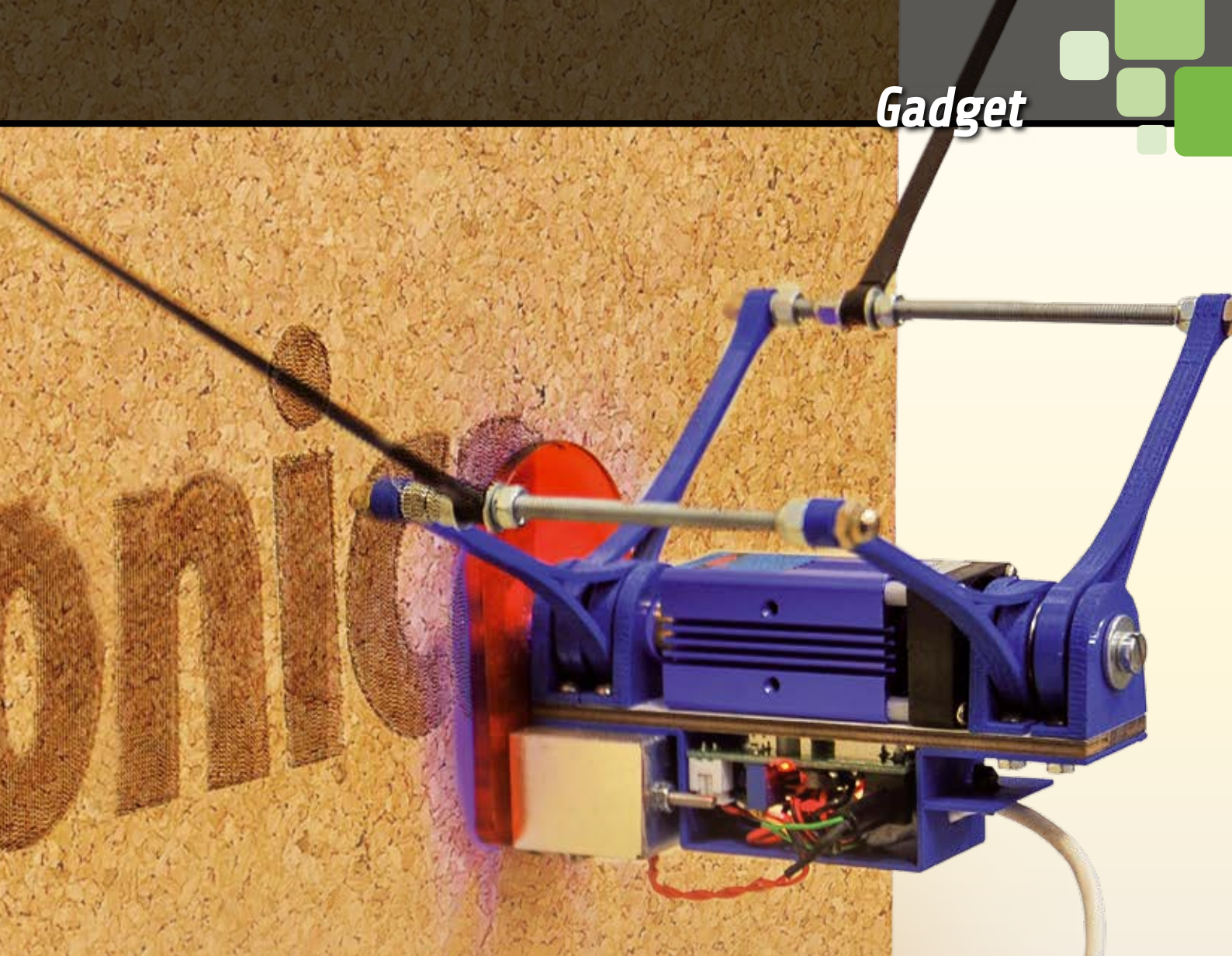
VERTICAL LASER **ENGRAVER**

di MICHELE CAIATI



Plotter verticale a incisione termica con il quale stampare immagini tipo BMP in bianco e nero su diversi materiali piani come legno, cartone e laminati, letteralmente bruciandone la superficie pixel dopo pixel.

Tipicamente un plotter è una speciale stampante in grado di disegnare con una penna a inchiostro come fosse a mano libera, con linee continue, opportunamente guidata da una meccanica di precisione; il foglio è disposto in orizzontale e può essere anche di materiale diverso dalla carte. Del plotter sono state create nel tempo varie versioni, anche per il ritaglio di pellicole e rivestimenti, ma anche verticali, come ad esempio il nostro Vertical Plotter pubblicato nel fascicolo n° 200. Ora abbiamo voluto spingerci più avanti, riprogettando tale plotter verticale ma in versione a laser, quindi con al posto del pennarello un laser allo stato solido in grado di tracciare col calore.
Così è nato l'incisore a laser su supporto verticale descritto in questo articolo, che è un appa-



rato dove, su un robusto pannello, si muove una testa di plottatura la quale, tramite un laser a semiconduttore sorretto e posizionato da due cinghie dentate, è in grado di riprodurre immagini tipo BMP in bianco e nero su diversi materiali piani, letteralmente bruciandone la superfice pixel dopo pixel. Il modulo laser utilizzato è a luce visibile, blu a 445 nm di lunghezza d'onda e la sua potenza ottica (ben 2 watt) è sufficientemente elevata da produrre rapidamente il calore che serve a incidere legno, cartone e laminati. Se è ideale per effettuare stampe abbastanza rapide, una potenza del genere richiede una certa cautela: sarà quindi indispensabile, durante l'utilizzo del plotter e nelle fasi di collaudo, munirsi di occhiali protettivi e avere molta prudenza.
Ma di questo ed altri dettagli spiegheremo più avanti; ora è il momento di gettare uno sguardo alla composizione della macchina.

LA STRUTTURA

Il plotter verticale che vi proponiamo, meglio descritto nella **Fig. 1**, ha una struttura portante (base) che è costituita da una lastra di legno truciolare laminato bianco da 18 mm di spessore, quadrata, avente il lato di 1 metro.
La base del Vertical Laser Engraver oltre a sostenere la meccanica dell'attrezzatura ospita i vari materiali da incidere, che possono essere tenuti in posizione da nastro adesivo o puntine. È possibile utilizzare una base più grande o addirittura fissare la meccanica su una parete con dei tasselli.
La meccanica è costi-

Materiale occorrente

In questo riquadro riepiloghiamo le parti occorrenti alla realizzazione del Vertical Plotter a laser, distinte per categoria.

COMPONENTI ELETTRONICI

- Modulo LASER blu 2 W - 445 nm
- Scheda Arduino Mega 2560 R.3
- Modulo lettore/scrittore SD Card Reader compatibile Arduino PIC;
- Scheda di memoria SD (capacità 4 - 8GB)
- 2 Stepper driver DRV8825 + aletta di raffreddamento
- 2 Scheda Stepper Motor Driver Control Panel
- Alimentatore stabilizzato 220V-12V 10A
- Alimentatore stabilizzato 220V- 5V 1A
- 2 stepper motor Nema 17 59Ncm bipolare
- 2 finecorsa ottici
- pulsante
- LED
- Cavi e connettori

COMPONENTI MECCANICI E FERRAMENTA

- Pannello in truciolare laminato 1 x 1 spessore 18 mm
- 2 Cinghie GT2 circa 4 m
- 2 Cinghie chiuse GT2 110 denti
- 2 Pulegge GT2 20 denti asse ø8mm
- 2 Pulegge GT2 20 denti asse ø5mm
- 2 Pulegge GT2 36 denti asse ø8mm
- 6 Cuscinetti a sfere tipo 608ZZ
- 4 Cuscinetti a sfere 12 x 21 x 5 mm
- 2 Assi ø8x51 mm alluminio
- 2 Tubi ø12x22 mm alluminio
- 2 Molle elicoidale 4x20 mm
- Barra filettata M5 x 110mm
- Barra filettata M5 x 124mm
- 2 Viti M8 x 20 mm testa esagonale
- 4 Viti M5 x 50 mm testa cilindrica
- 10 Viti M3 x 50 mm testa bombata
- 2 Viti M3 x 25 mm testa bombata
- 8 Viti M3 x 12 mm testa bombata
- 2 Viti M2 x 30 mm testa bombata
- 2 Dadi M8 cieco
- 12 Dadi M5 autobloccanti
- 4 Dadi M5 ciechi
- 13 Dadi M3 autobloccanti
- 2 Dadi M2 autobloccanti
- 4 Rondelle spessore 1,8 mm foro ø8mm
- 12 Rondelle spessore 1 mm foro ø5 mm
- 6 Rondelle spessore 0,5 mm foro ø3 mm
- 4 Rondelle isolanti spessore 2 mm foro ø3 mm
- 2 Rondelle foro ø2 mm
- 4 Viti autofilettanti ø3mm x 9mm
- 2 Viti autofilettanti ø3mm x 20mm
- 12 OR ø10/12mm
- Contrappeso da 300g

COMPONENTI REALIZZATI IN STAMPA 3D

TESTINA DI STAMPA:

- 4 pezzi T1
- 4 pezzi T2
- 4 pezzi T3

TRASCINATORE CINGHIA:

- 1 pezzo V1-SX
- 1 pezzi V1-DX
- 2 pezzi V2-V3
- 1 pezzo V4-SX
- 1 pezzo V4-DX
- 2 pezzi V5
- 1 pezzo V6-SX
- 1 pezzo V6-DX

tuita da due trascinatori e una testina collegati tramite cinghie dentate tipo GT2. I componenti principali del Vertical Laser Engraver sono stati realizzati con la stampa 3D e completati da ferramenta facilmente reperibile sul mercato.
Per assemblare la macchina, la prima cosa da fare sarà quindi stampare le varie parti e procurarci il necessario; allo scopo ricordiamo che delle parti da stampare in 3D (se avete la nostra stampante 3Drag tanto meglio...) rendiamo disponibili sul nostro sito *www.elettronican.it* tutti i file .STL per la stampa, accessibili e scaricabili insieme ai file del progetto del plotter. Sono previste anche due piastre in PMMA (polimetilmetacrilato) brunito che andranno sagomate e forate come in **Fig. 2** e rispettivamente, P1 serve a sostenere i componenti della testina di stampa, mentre P2, è una protezione per schermare eventuali raggi riflessi durante il funzionamento del laser.
Un altro componente da realizzare è il contrappeso, che è composto da un blocchetto in metallo che serve a bilanciare la testina e a tenerla orizzontale durante gli spostamenti. Questo componente deve avere un peso approssimativo di 300 g e gli vanno praticati dei fori filettati, come mostrato nella **Fig. 3**, per il fissaggio.
Tutti gli elementi occorrenti alla realizzazione del progetto sono riportati per chiarezza nel riquadro "Materiale occorrente", distinti per categoria.
Adesso analizziamo uno ad uno i sottosistemi componenti il plotter, partendo dagli azionamenti della Testina Laser.

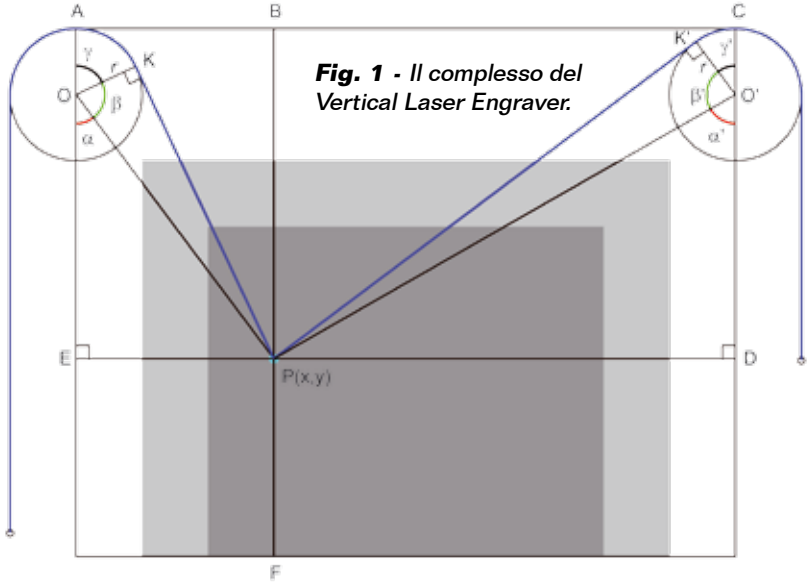


***Fig. 1** - Il complesso del Vertical Laser Engraver.*

GLI AZIONAMENTI

Queste parti del Vertical Laser

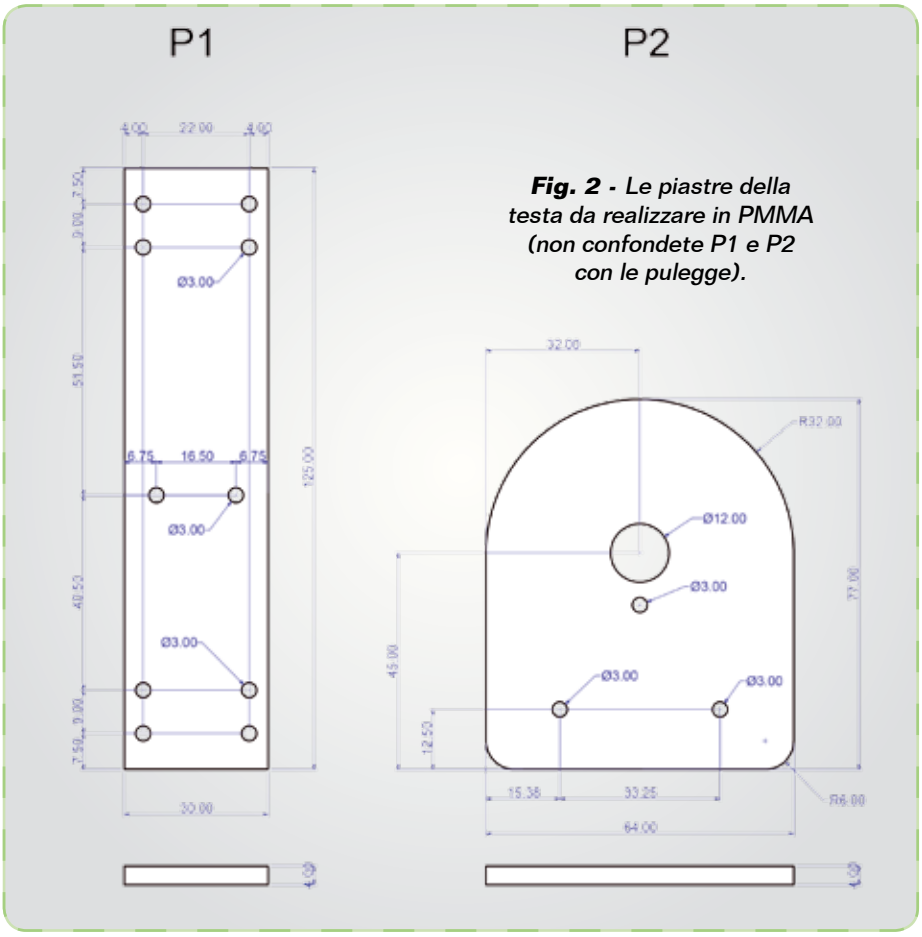


***Fig. 2** - Le piastre della testa da realizzare in PMMA (non confondete P1 e P2 con le pulegge).*

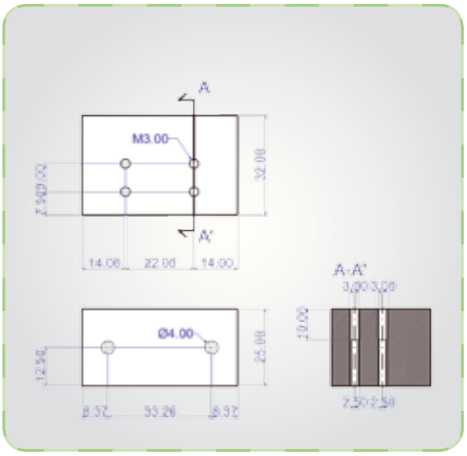


***Fig. 3** - Dettagli del contrappeso.*

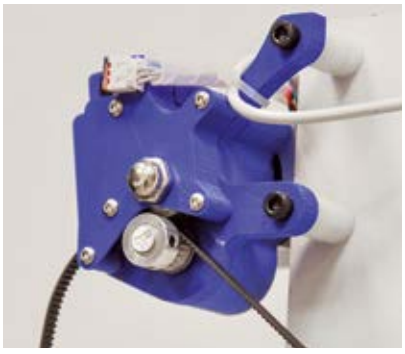


***Fig. 4** - Un azionamento.*

Engraver sono speculari e si occupano di muovere le cinghie che sorreggono la testina, ricevendo il moto dai motori; esse vanno fissate saldamente nella parte superiore della base in truciolato, secondo quanto mostrato nella **Fig. 4**. Il montaggio va effettuato utilizzando le viti M5 x 50 mm con relativi dadi e rondelle.
Per aiutarvi nell'assemblaggio e nella composizione vi proponiamo, nella **Fig. 5**, l'esploso della sezione azionamenti del plotter.

LA TESTINA LASER

Componente vitale del plotter è la testa di stampa, che porta il laser da 2 W. L'insieme è proposto nella **Fig. 6**, dove la testina laser è la parte della macchina che andrà a effettuare l'incisione. Il raggio emesso dal diodo laser e concentrato in un punto dovrà carbonizzare quasi istantaneamente il materiale da incidere, quindi è vitale regolare la lente del laser affinchè il fuoco coincida con la superfice del piano. La testina deve rimanere verticale e aderente al materiale da incidere durante gli spostamenti e ciò è possibile regolando il punto di attacco delle cinghie in avanti o indietro cercando di tenere ortogonali il piano di lavoro e l'asse del diodo laser. Abbiamo previsto, a bordo, un microswitch la cui levetta verrà premuta quando la testina sarà appoggiata al piano; un accidentale allontanamento o inclinazione anomala della testina farà scattare il microinterruttore, spegnendo il laser per garantire la sicurezza dell'operatore. La **Fig. 7** propone la vista in esploso del complesso della testina.

AZIONAMENTO DELLE CINGHIE

Per il movimento della cinghie dentate principali abbiamo previsto due gruppi (**Fig. 4**) basati ognuno su motori stepper da 59 Ncm con una risoluzione di 200 passi per giro, montando sul loro albero delle pulegge GT2 da 20 denti (P1). Le pulegge dei motori sono collegate a pulegge più grandi da 36 denti (P2) tramite cinghie chiuse, creando una prima demoltiplica dei giri. Le pulegge da 36 denti sono solidali con altre da 20 denti (P3) che provvedono al trascinamento delle cinghie che sorreggono e posizionano la sorgente laser. I motori sono pilotati dai rispettivi driver in 1/8 di passo cosicché per far fare all'asse del motore un giro completo saranno necessari 1.600 micropassi da 1/8 di passo. Il rapporto di demoltiplica tra la puleggia P2 e P1 è di 1,8 e quindi per far compiere a P2 e P3 un giro completo, il motore dovrà

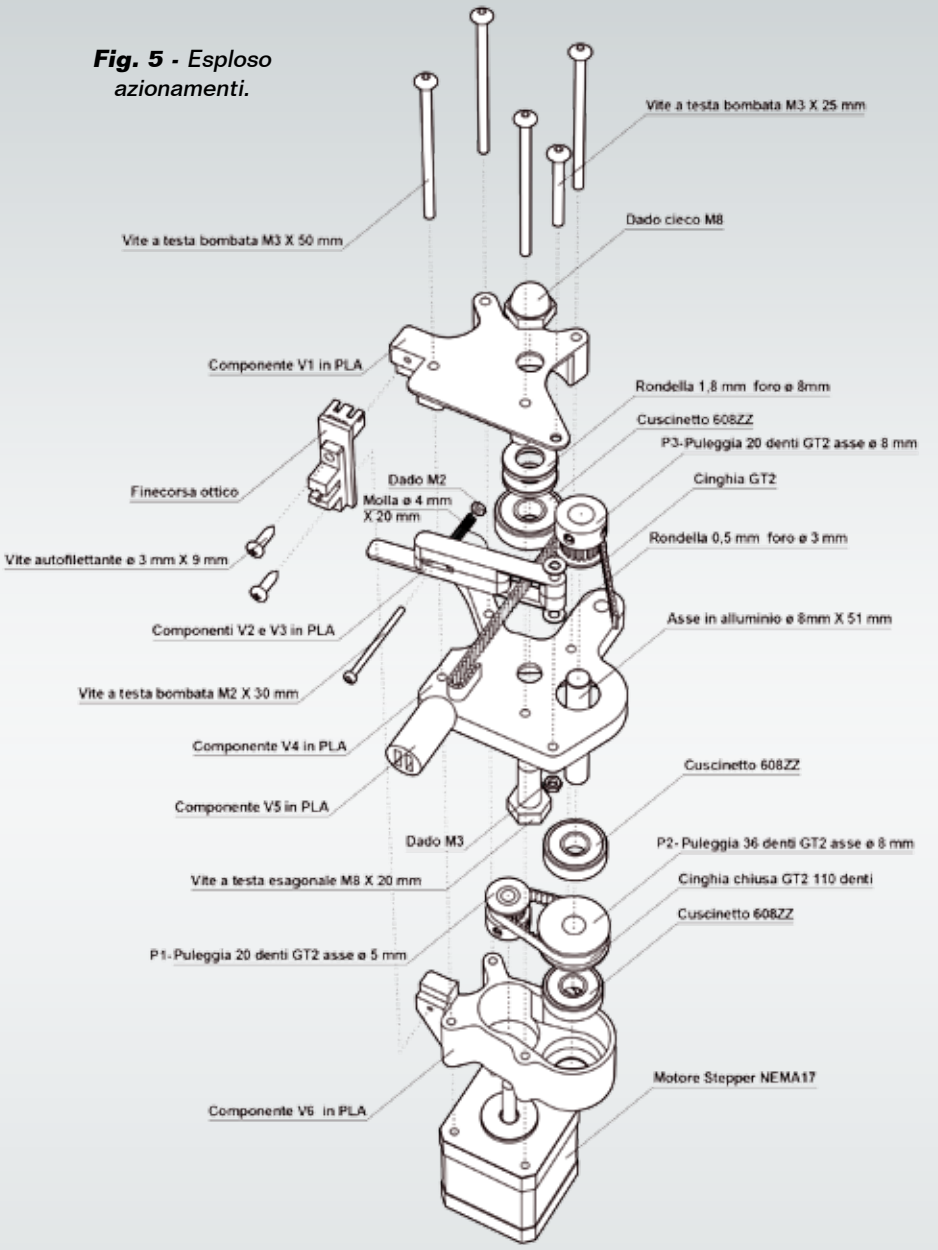


***Fig. 5** - Esploso azionamenti.*

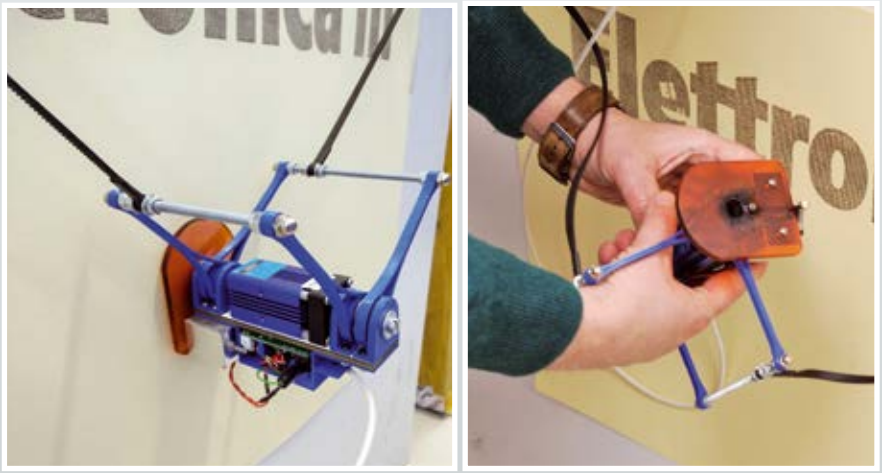


***Fig. 6** - Testina laser completa: in evidenza il microswitch di sicurezza.*

fare 2.880 micropassi. Per calcolare quanti passi dovrà eseguire il motore per spostare la cinghia di 1 mm, dobbiamo considerare che la puleggia di trascinamento ha 20 denti e utilizza una cinghia tipo GT2, cioè con un dente ogni 2 mm.
Moltiplicando 20 denti per 2 mm otteniamo 40 mm, ovvero lo spostamento della cinghia ad ogni giro completo di P3.
Sapendo che per ogni giro di P3 il motore compie 2.880 micropassi, possiamo calcolare i microstep per mm, che sono 2880/40, vale a dire 72.
Il valore trovato ci servirà per poter controllare con precisione il nostro Vertical Laser Engraver, che è teoricamente in grado di compiere spostamenti minimi di 0,013888888888888 mm. Ricapitolando, abbiamo:

- motore da 200 passi giro passo 1,8°;
- driver 1/8 passo microstep;
- P1 = puleggia motore 20 denti;
- P2 = puleggia di raccordo 36 denti;
- P3 = puleggia di trascinamento 20 denti;
- cinghia GT2 passo 2 mm;
- 200 x 8 = 1600 micropassi giro;
- rapporto di demoltiplica P2/P1 = 1,8.

Quindi otteniamo 1600 x 1,8 = 2.880 micro passi per giro su P2 e P3. Avendo 20 denti x 2 mm, ci sono 40 mm di cinghia per ogni giro di P3.
Dividendo il numero di microstep per la lunghezza della cinghia corrispondente otteniamo:

2880 / 40 = 72 micropassi per mm

vale a dire che il minimo movimento ottenibile della cinghia ammonta a:

1/72 = 0,013888888888888 mm

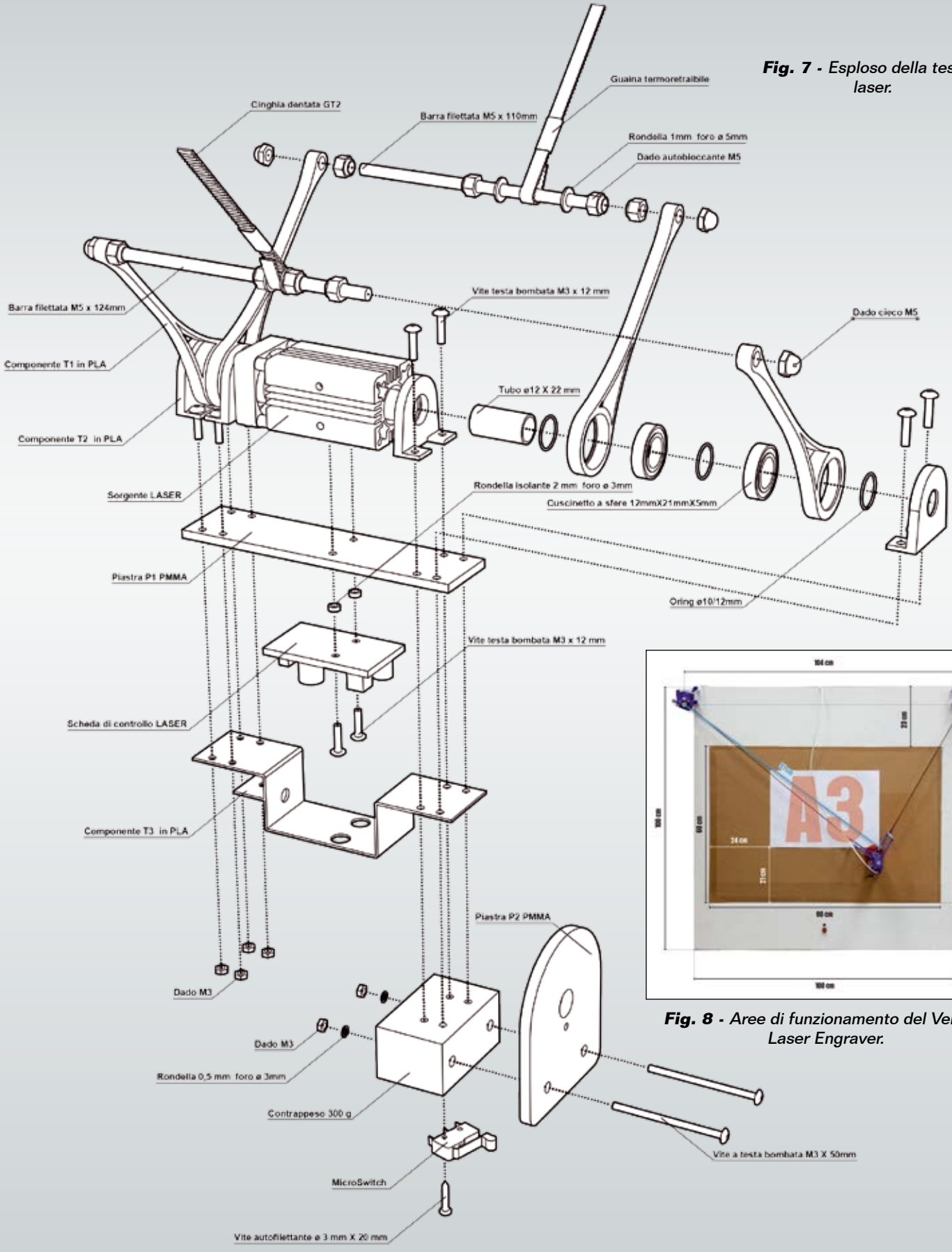
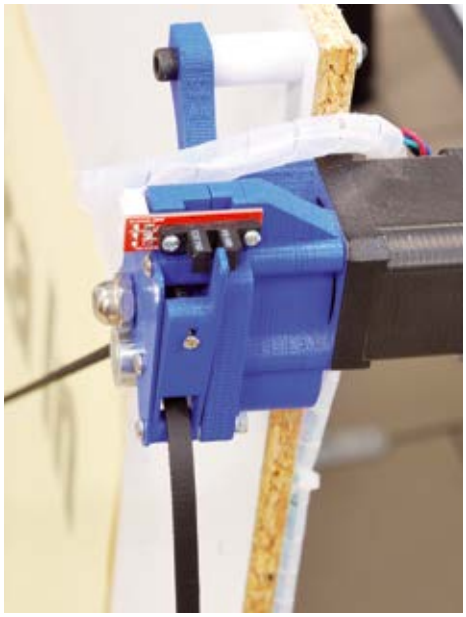


Fig. 7 - *Esploso della testina laser.*

Fig. 8 - *Aree di funzionamento del Vertical Laser Engraver.*



Azionamento della cinghia visto dall'interno e dall'esterno: il sistema prevede un finecorsa, basato su una levetta che penetra un fotoaccoppiatore aperto.

Ciò detto passiamo alla generazione dei file di stampa.

GENERARE IL FILE

Per l'incisione creeremo un'immagine bitmap in bianco e nero con dimensioni in pixel doppie rispetto a quelle in mm che realmente otterremo con il laser. Dalla **Fig. 1**, notiamo che se l'interasse tra le pulegge (lo chiamiamo AC) è di 1.040 mm e la distanza (BF) altri 1.040 mm, la nostra immagine avrà una larghezza di 2.080 e un'altezza di 2.080 pixel. Questo espediente ci consentirà di incidere dei punti distanti tra loro 0,5 mm ottenendo un miglioramento della risoluzione.
Nella **Fig. 1** notate delle aree in toni di grigio: quella più chiara rappresenta l'area limite che possiamo utilizzare mentre la più scura rappresenta l'area consigliata per la stampa tenendo presente le limitazioni meccaniche dell'apparato. Tornando all'esempio, se le nostre pulegge hanno interasse 1.040 mm sarà necessario, nel creare l'immagine da incidere, lasciare delle fasce di pixel bianchi ai lati e sopra il soggetto effettivo; saranno sufficienti 10 cm circa ai lati (200 pixel) e 20 cm sopra (400 pixel) così da poter limitare lo spostamento della testina. Tornando alla **Fig. 1**, avremo le due pulegge di raggio r, il piano di larghezza AC e altezza BF e, rappresentate in blu, le due cinghie che si congiungono nel punto P che indica il diodo laser che puntato verso il supporto andrà ad effettuare l'incisione. Per poter controllare la posizione di P dovremo quindi modificare la lunghezza delle due cinghie attraverso la rotazione coordinata delle pulegge e quindi conoscerne effettivamente le dimensioni. Considerando il triangolo rettangolo EOP, possiamo dire che EP è uguale alla metà del valore della coordinata x di P, perché l'immagine che consideriamo ha dimensioni doppie in pixel rispetto a ciò che dobbiamo realmente stampare, quindi:

$$EP = x / 2$$

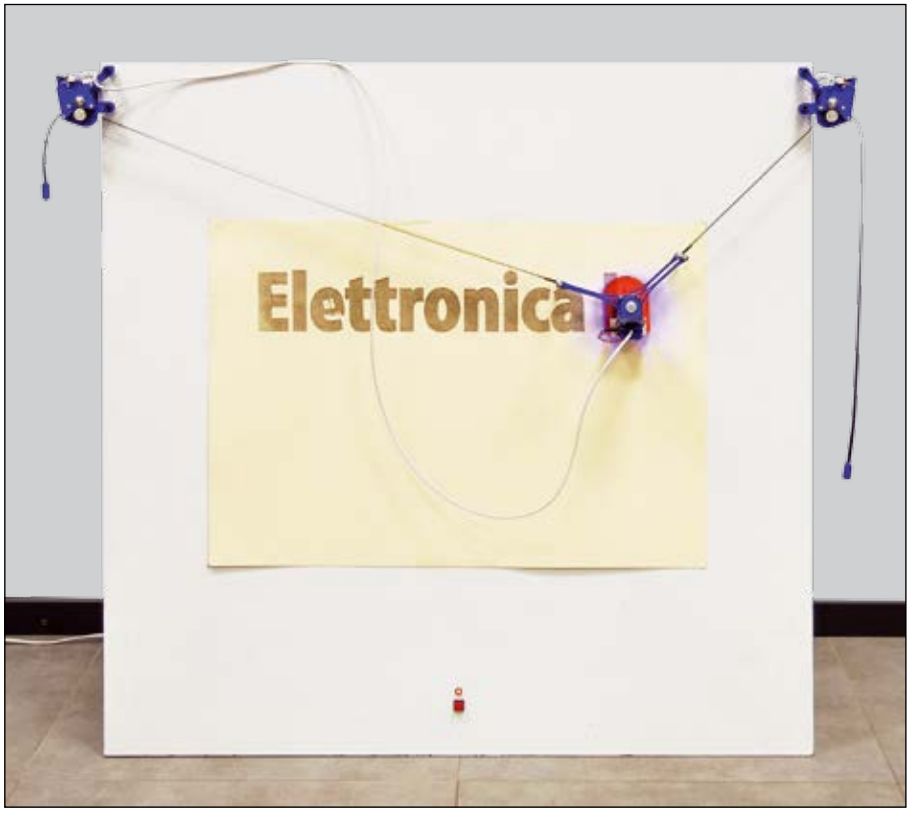
Considerando EO possiamo dire che:

$$EO + OA = BP$$

Siccome OA = r e PF = BF - (y / 2), poiché nelle immagini tipo bitmap le coordinate di y crescono spostandosi dall'alto verso il basso, otteniamo che:

$$EO = BF - PF - r$$

Adesso, con il teorema di Pitagora calcoliamo OP:



Complesso del Vertical Laser Engraver visto dal davanti, montato e operativo su un foglio di cartoncino.

Listato 1 – Il codice in Python

```
import PIL.Image
import math
AC = 1040.000 #interasse pulegge in mm
r = 6.11  #raggio pulegge in mm
im = PIL.Image.open("incisione.bmp") #apriamo l'immagine da incidere
print (im.format, im.size, im.mode) #visualizziamo le informazioni sull'immagine
pix = im.load()

dimx = im.size[0] #otteniamo la larghezza dell'immagine
dimy = im.size[1] #otteniamo l'altezza dell'immagine
print (dimx) #visualizziamo la larghezza dell'immagine
print (dimy) #visualizziamo l'altezza dell'immagine

Velocita = 5000 #impostiamo la velocitÃ massima dei motori
Delytime = 50  #impostiamo il tempo di accensione del laser
SIntensita = 255 #impostiamo l'intensitÃ  del laser
Liniziale0 = 47520 #impostiamo la lunghezza di partenza delle cinghia 1 espressa in passi del motore che equivale a 1500 mm
Liniziale1 = 47520 #impostiamo la lunghezza di partenza delle cinghia 2 espressa in passi del motore che equivale a 1500 mm

documento = open("Laser.txt", "w")#apriamo ed eventualmente creiamo il file di stampa

direzione = 0
for x in range(dimx): #scorriamo l'immagine da sinistra verso destra individuando i pixel non bianchi
    if direzione == 0:
        for y in range(dimy): #scorriamo l'immagine dall'alto verso il basso individuando i pixel non bianchi
            if pix[x,y] == 0:
                print (x,',',y,',',255 - pix[x,y])
                EP = x / 2.0000
                BP = y / 2.0000
                EO = BP - r

                OP = math.sqrt(math.pow(EO,2) + math.pow(EP,2))
                KP = math.sqrt(math.pow(OP,2) - math.pow(r,2))

                alfa = math.asin(EP/OP)
                alfag = math.degrees(alfa)
                beta = math.asin(KP/OP)
                betag = math.degrees(beta)
                gammag = 180-(alfag + betag)

                KA = 2*math.pi*r*gammag/360
                Lp = KA + KP

                PD = AC - EP

                O1P = math.sqrt(math.pow(EO,2) + math.pow(PD,2))
                K1P = math.sqrt(math.pow(O1P,2) - math.pow(r,2))

                alfa1 = math.asin(PD/O1P)
                alfa1g = math.degrees(alfa1)
                beta1 = math.asin(K1P/O1P)
                beta1g = math.degrees(beta1)
                gamma1g = 180-(alfa1g + beta1g)
                K1C = 2*math.pi*r*gamma1g/360
                Lp1 = K1C + K1P

                StepLp = Lp * 72 #trasformiamo la lunghezza della cinghia 1 in passi
                StepLpr = int(round(StepLp))#poichè il numero di passi deve essere un intero arrotondiamo il valore
                StepLp1 = Lp1 * 72
                StepLp1r = int(round(StepLp1))

                deltapassi0 = abs(StepLp - Liniziale0)#calcoliamo la variazione di passi e quindi della lunghezza della cinghia 1
                deltapassi1 = abs(StepLp1 - Liniziale1)#calcoliamo la variazione di passi e quindi della lunghezza della cinghia 2

#calcoliamo la velocità  dei motori in base al numero di passi che devono compiere e alla velocità massima

                if deltapassi0 > deltapassi1:
                   rapporto = deltapassi1 / deltapassi0
                   V1 = int(Velocita)
                   V2 = int(round(Velocita * rapporto))
                if deltapassi0 < deltapassi1:
                   rapporto = deltapassi0 / deltapassi1
                   V1 = int(round(Velocita * rapporto))
                   V2 = int(Velocita)
                if deltapassi0 == deltapassi1:
                   V1 = int(Velocita)
                   V2 = int(Velocita)

#componiamo la riga di valori da scrivere sul file relativa ad un pixel

                Lc = str(StepLpr)
                Lc1 = str(StepLp1r)
```

(Continua)

Listato 1 – segue

```
                SInt = str(SIntensita)
                V1s = str(V1)
                V2s = str(V2)
                timd = str(Delytime)
                RIGA =  Lc + ",” + Lc1 + ",” + V1s + ",”  + V2s + ",” + SInt + ",” + timd  + "\n”
                documento.write(RIGA)
#aggiorniamo la posizione del laser
                Liniziale0 = StepLpr
                Liniziale1 = StepLp1r
    if direzione == 1:
        dimyrev = dimy - 1
        for y in range(dimyrev,-1,-1):#scorriamo l’immagine dal basso verso l’alto individuando i pixel non bianchi
            if pix[x,y] == 0:
                print (x,’,’,y,’,’,255 - pix[x,y])
                EP = x / 2.0000
                BP = y / 2.0000
                EO = BP - r

                OP = math.sqrt(math.pow(EO,2) + math.pow(EP,2))
                KP = math.sqrt(math.pow(OP,2) - math.pow(r,2))

                alfa = math.asin(EP/OP)
                alfag = math.degrees(alfa)
                beta = math.asin(KP/OP)
                betag = math.degrees(beta)
                gammag = 180-(alfag + betag)

                KA = 2*math.pi*r*gammag/360
                Lp = KA + KP

                PD = AC - EP

                O1P = math.sqrt(math.pow(EO,2) + math.pow(PD,2))
                K1P = math.sqrt(math.pow(O1P,2) - math.pow(r,2))

                alfa1 = math.asin(PD/O1P)
                alfa1g = math.degrees(alfa1)
                beta1 = math.asin(K1P/O1P)
                beta1g = math.degrees(beta1)
                gamma1g = 180-(alfa1g + beta1g)
                K1C = 2*math.pi*r*gamma1g/360
                Lp1 = K1C + K1P

                StepLp = Lp * 72 #trasformiamo la lunghezza della cinghia 1 in passi
                StepLpr = int(round(StepLp))#poichè il numero di passi deve essere un intero arrotondiamo il valore
                StepLp1 = Lp1 * 72
                StepLp1r = int(round(StepLp1))

                deltapassi0 = abs(StepLp - Liniziale0)
                deltapassi1 = abs(StepLp1 - Liniziale1)
                if deltapassi0 > deltapassi1:
                    rapporto = deltapassi1 / deltapassi0
                    V1 = int(Velocita)
                    V2 = int(round(Velocita * rapporto))

                if deltapassi0 < deltapassi1:
                    rapporto = deltapassi0 / deltapassi1
                    V1 = int(round(Velocita * rapporto))
                    V2 = int(Velocita)

                if deltapassi0 == deltapassi1:
                    V1 = int(Velocita)
                    V2 = int(Velocita)

                Lc = str(StepLpr)
                Lc1 = str(StepLp1r)
                SInt = str(SIntensita)
                V1s = str(V1)
                V2s = str(V2)
                timd = str(Delytime)
                RIGA =  Lc + ",” + Lc1 + ",” + V1s + ",”  + V2s + ",” + SInt + ",” + timd  + "\n”
                documento.write(RIGA)

                Liniziale0 = StepLpr
                Liniziale1 = StepLp1r
    if direzione == 0:
        direzione = 1
    else:
        direzione = 0


documento.close()
```

OP = √ EO 2 + EP 2
Utilizzando il teorema di Pitagora calcoliamo anche KP:

KP = √ OP 2 - r 2

Per conoscere la lunghezza esatta della cinghia da A a P dobbiamo ora calcolare la lunghezza dell'arco AK e quindi individuare l'angolo γ:

α = Arcosen (EP / OP)

β = Arcosen (KP / OP)

γ = 180 – (α + β)
KA = 2 ∏ r γ / 360

Ora possiamo dire che la lunghezza L della prima cinghia quando il laser si trova nel punto P sarà:

L = KA + KP

Per calcolare la lunghezza della seconda cinghia procederemo in modo analogo:

PD = AC – EP

DO′ = EO

PO′ = √ EO 2 + PD 2

PK′ = √ PO′ 2 - r 2
α′ = Arcosen (PD / PO′)

β′ = Arcosen (PK′ / PO′)

γ′ = 180 – (α′ + β′)

K′C = 2 ∏ r γ ′ / 360

Da cui deriva che la seconda cinghia L′ avrà lunghezza :

L′ = PK′ + K′C

Definite le lunghezze variabili delle due cinghie possiamo realizzare un piccolo codice che

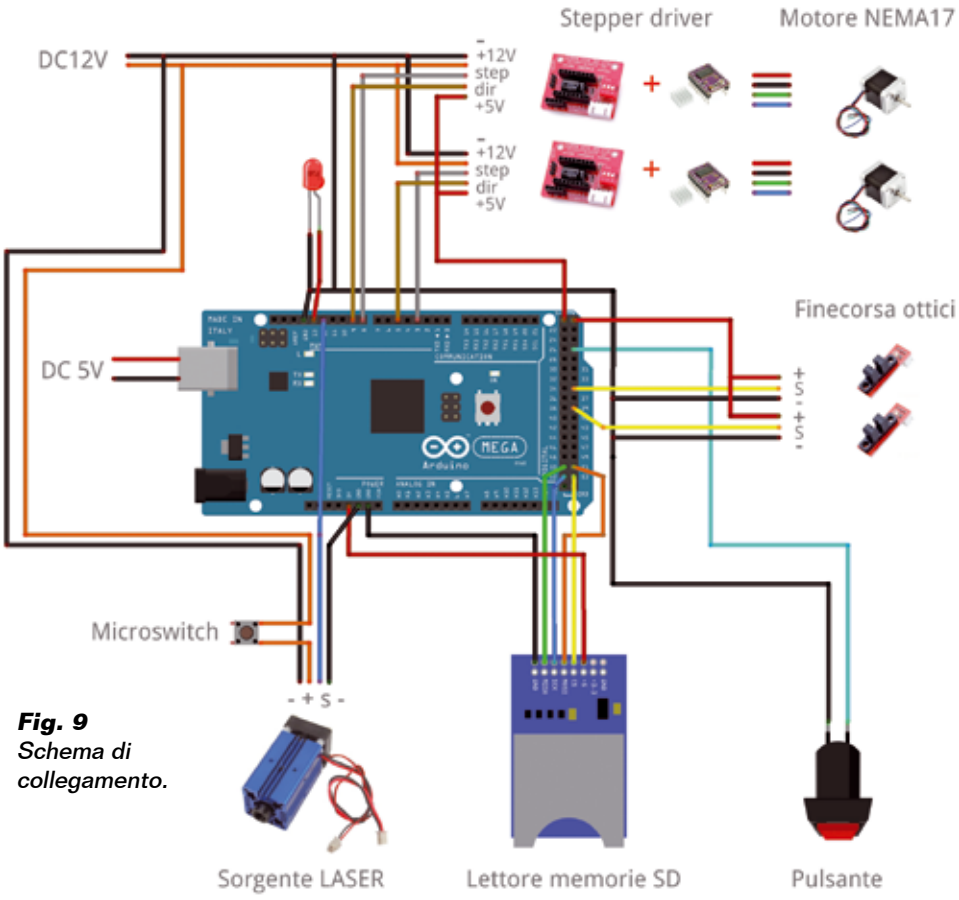
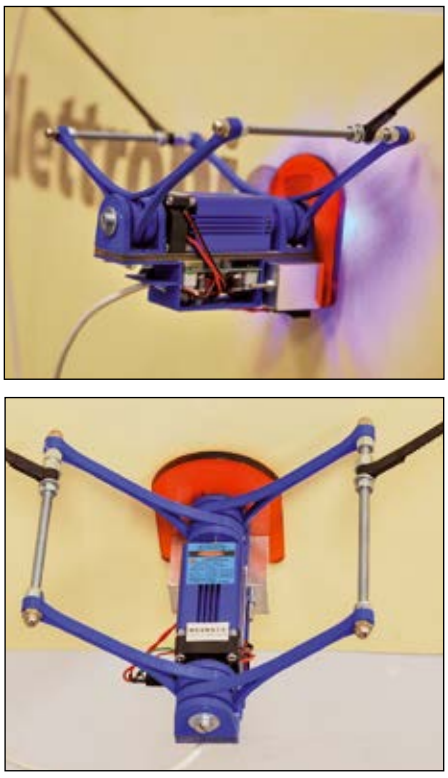


***Fig. 9** Schema di collegamento.*

si occuperà di:

- aprire l'immagine che vogliamo incidere;
- analizzarla pixel per pixel per calcolare le lughezze L e L′ relative a tutti i punti di colore nero;
- convertire tali lunghezze in passi da far eseguire ai motori ad una determinata velocità;
- associare ad ogni coppia di lunghezze i valori di intensità e durata di incisione del laser;
- salvare i dati in un file.txt opportunamente formattato.

Per la realizzazione del software utilizzeremo Python in versione 2.7.11 in associazione a due librerie: PIL 1.1.7 e Math



Dettagli della testa di stampa accesa (si vede la luce blu parzialmente riflessa).

Guarda Vertical Plotter in azione su YouTube!

Puoi vedere il Vertical Laser Engraver mentre incide la scritta "Elettronica In" su un cartoncino alla pagina https://youtu.be/nquqyK4Mrmg. Dal nostro canale YouTube *www.youtube.com/user/ElettronicaIN* potrai anche vedere i video di moltissimi altri nostri progetti proposti in passato.

(quest'ultima è già presente nell'istallazione di Python). La libreria Pil (Python Imaging Library) ci consentirà di leggere le dimensioni dell'immagine, le coordinate e il colore dei vari pixel, mentre la libreria Math ci darà accesso alle funzioni trigonometriche che serviranno per il calcolo della lunghezza delle cinghie.
Per scaricare Python andate su Internet alla pagina *http://www.python.it/*, mentre la Libreria PIL è disponibile per il download su *http://www.pythonware.com/products/pil/*.
Il codice (**Listato 1**) apre l'immagine in bianco e nero **incisione.bmp** che deve essere collocata nella stessa cartella che lo contiene quindi per prima cosa dobbiamo convertire, ridimensionare e rinominare il soggetto che vogliamo incidere. Verrà generato o sovrascritto nella stessa cartella un file **Laser.txt** che sarà da copiare sulla memoria SD del Vertical Laser Engraver e che contiene i dati per l'incisione.
Sulla memoria SD deve essere presente un solo file alla volta.
Se osserviamo il codice,vedremo, nella prima parte la definizione di alcune variabili che dipendono dalla parte meccanica della nostra attrezzatura.
Queste impostazioni devono coincidere con le grandezze reali della nostra macchina. In particolare, la lunghezza iniziale delle cinghie e cioè la lunghezza che assumono dopo che il Vertical-Laser Engraver viene acceso e si resetta utilizzando i finecorsa, deve coincidere con quella del firmware caricato su Arduino MEGA.Questa lunghezza é espressa in passi e nel codice di esempio ha un valore di 47.520, che diviso 72 restituisce 660 mm. Ogni volta che accenderemo la macchina le due cinghie dentate andranno una alla volta a scorrere fino ad eccitare il finecorsa, per poi raggiungere la lunghezza di 660mm misurata dalla puleggia al centro dell'emettitore laser.Un'altra grandezza fondamentale per un corretto funzionamento della macchina è la lunghezza massima delle cinghie dentate, che va misurata al momento dell'innesco dei fine corsa e misurata sempre dalla puleggia al centro dell'emettitore laser. Nelle impostazioni iniziali viene anche definita la potenza e l'intervallo di tempo con cui viene pilotato il diodo laser e queste impostazioni dipendono dal materiale che vogliamo incidere e dalla potenza del laser.
Il file *Laser.txt* che viene generato è un elenco di righe formate da 6 valori separati da una virgola e apparirà in questo modo:

```
41166,59235,5000,3501,255,50
41137,59215,5000,3397,255,50
41109,59196,5000,3413,255,50
41080,59176,5000,3482,255,50
41052,59156,5000,3497,255,50
41024,59137,5000,3387,255,50
40995,59117,5000,3453,255,50
.....
```

i primi due valori racchiudono la lunghezza delle cinghie espresse in passi, i secondi le velocità alle quali devono muoversi i motori, il quinto valore indica la potenza del laser che va da 0 a 255 e il sesto valore indica il tempo di accensione espresso in millisecondi.
Ogni riga definisce un punto, quindi proporzionalmente alla grandezza dell'immagine e alla densità di punti che contiene, questo file può raggiungere dimensioni importanti.

I COLLEGAMENTI

Bene, a questo punto possiamo passare a descrivere il cablaggio della macchina, che va eseguito con riferimento alla **Fig. 9**.
Il cervello dell'elettronica é una scheda Arduino MEGA 2560 R.3 alla quale sono collegati i due driver, basati sul chip A4988 dell'Allegro, che è uno dei più utilizzati per il genere di motore passo-passo utilizzato nel progetto. Per comodità i driver sono montati su schede "A4988/DRV8825 stepper motor driver control panel/expansion board" (**Fig. 10**) che consentono

***Fig. 10** - 4988/DRV8825 stepper motor driver control panel/expansion board.*

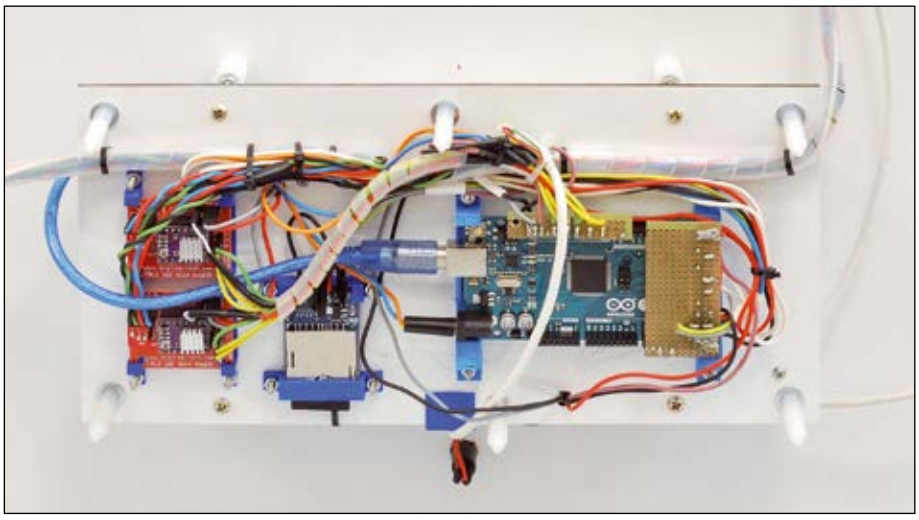
Listato 2 – Il codice per Arduino

```
#include <SPI.h>
#include <SD.h> // utilizziamo la libreria SD per leggere il file Laser dalla scheda SD
#include <AccelStepper.h> // utilizziamo la libreria AccelStepper per controllare i motori
AccelStepper stepper1(1, 3, 5); //assegniamo i pin 3 e 5 al motore 2 con tipologia di controllo step,dir
AccelStepper stepper2(1, 8, 9); //assegniamo i pin 8 e 9 al motore 2 con tipologia di controllo step,dir

File myFile;
long Cinlong1 = 65520; //lunghezza totale della cinghia 1 espressa in numero di passi
long Cinlong2 = 65520; //lunghezza totale della cinghia 2 espressa in numero di passi
long Stzero1 = 47520;  //lunghezza della cinghia 1 nella posizione iniziale espressa in numero di passi
long Stzero2 = 47520;  //lunghezza della cinghia 2 nella posizione iniziale espressa in numero di passi

const int BUTTON = 27; //creiamo la costante BUTTON per utilizzare un pulsante sul pin 27
int val = 0;
int vecchio_val = 0;
int stato = 0;

int intenslaser = 0; //creiamo la variabile per controllare la potenza del diodo laser
int delytime = 0; //creiamo la variabile per controllare i tempi diaccensione del diodo laser

void setup() {

  pinMode(BUTTON, INPUT); //definiamo il pin 27 come input

  pinMode(12, OUTPUT); //definiamo il pin 12 come output
  analogWrite(12, 0); //portiamo a 0 l'uscita del pin

  pinMode(13, OUTPUT); //definiamo il pin 13 come output
  digitalWrite(13, LOW); //portiamo a 0 l'uscita del pin

 const int Endstop1 = 39; //creiamo la costante Endstop1 per poter collegare il finecorsa ottico della cinghia1 sul pin 39
 const int Endstop2 = 35; //creiamo la costante Endstop1 per poter collegare il finecorsa ottico della cinghia1 sul pin 35
 pinMode(Endstop1, INPUT);//definiamo il pin 39 come input
 pinMode(Endstop2, INPUT);//definiamo il pin 35 come input

 // definiamo velocità  massima e accelerazione e velocità di entrambi i motori
 stepper1.setMaxSpeed(10000);
 stepper1.setAcceleration(500);
 stepper2.setMaxSpeed(10000);
 stepper2.setAcceleration(500);
 stepper1.setSpeed(1440);
 stepper2.setSpeed(1440);
 while (digitalRead(Endstop1) == LOW){
 // muoviamo il motore della cinghia 1 per raggiungere il finecorsa
 stepper1.setSpeed(10000);
 stepper1.runSpeed();
}
// muoviamo il motore della cinghia 1 di 216 passi in senso opposto per disimpegnare il finecorsa
 stepper1.move(-216);
 while (stepper1.distanceToGo() != 0){
 stepper1.run();
 }
 // muoviamo il motore della cinghia 1 di 9 passi alla volta per raggiungere un allineamento preciso con il finecorsa
  while (digitalRead(Endstop1) == LOW){
 stepper1.move(9);
 stepper1.run();
}
 // muoviamo il motore della cinghia 2 per raggiungere il finecorsa
while (digitalRead(Endstop2)== LOW){
 stepper2.setSpeed(10000);
 stepper2.runSpeed();
}
// muoviamo il motore della cinghia 2 di 216 passi in senso opposto per disimpegnare il finecorsa
stepper2.move(-216);
 while (stepper2.distanceToGo() != 0){
 stepper2.run();
 }
 // muoviamo il motore della cinghia 2 di 9 passi alla volta per raggiungere un allineamento preciso con il finecorsa
while (digitalRead(Endstop2)== LOW){
 stepper2.move(9);
 stepper2.run();
}
//assegniamo alle cinghie il loro valore massimo di lunghezza poiché si trovano i posizione di finecorsa
stepper1.setCurrentPosition(Cinlong1);
delay(15);
stepper2.setCurrentPosition(Cinlong2);
delay(15);
```

(Continua)

Listato 2 – segue

```
  // portiamo la testina del laser nella posizione di partenza
  stepper1.move(Stzero1 - Cinlong1);
  stepper2.move(Stzero1 - Cinlong2);
  while ((stepper1.distanceToGo() != 0) or (stepper2.distanceToGo() != 0)) {
  stepper1.setSpeed(5000);
  stepper1.runSpeedToPosition();
  stepper2.setSpeed(5000);
  stepper2.runSpeedToPosition();
  stepper2.run();
  }
  // inizializziamo la scheda SD e apriamo il file Laser.txt

  pinMode(53, OUTPUT);

  if (!SD.begin(53)) {

    return;
  }

    myFile = SD.open("Laser.txt");
}
void loop() {
// utilizziamo un pulsante per far partire la stampa
  val = digitalRead(BUTTON);
   if ((val == HIGH) && (vecchio_val == LOW)){
 stato = 1 - stato;
 delay(15);
  }
   vecchio_val = val;
//leggiamo il file Laser.txt e assegniamo i valori alle variabili
  if (stato == 1) {
     if (myFile.available() > 0) {
    long Lung1= myFile.parseInt();
    long Lung2= myFile.parseInt();
    int V1= myFile.parseInt();
    int V2= myFile.parseInt();
    int Intensita= myFile.parseInt();
    int delytime= myFile.parseInt();
    // quando Lung1 e Lung2 sono uguali a zero significa che il file è stato letto completamente e il Vertical laser si ferma
    if ((Lung1 == 0) && (Lung2 == 0)){

    Stzero1 = 0;
    Stzero2 = 0;
    }
    // definiamo il numero di passi e la direzione da far eseguire ai due motori
    stepper1.move(Lung1 - Stzero1);
    stepper2.move(Lung2 - Stzero2);

    //aspettiamo finché il laser non abbia raggiunto la posizione assegnatagli
     while ((stepper1.distanceToGo() != 0) or (stepper2.distanceToGo() != 0)){
  stepper1.setSpeed(V1);
  stepper1.runSpeedToPosition();
  stepper2.setSpeed(V2);
  stepper2.runSpeedToPosition();
  stepper2.run();

   }
   //attiviamo il laser sul pin 12 e una spia sul pin 13 con intensità e durata definiti dal file Laser.txt in associazione
al punto raggiunto.
    if ((stepper1.distanceToGo() == 0)  && (stepper2.distanceToGo() == 0)){
    analogWrite(12, Intensita);
    digitalWrite(13, HIGH);
    delay(delytime);
    analogWrite(12, 0);
    digitalWrite(13, LOW);
    //aggiorniamo la posizione della testina laser
    Stzero1 = Lung1;
    Stzero2 = Lung2;
    //se premiamo nuovamente il pulsante il processo di incisione si annulla e il Vertical laser si ferma
    if (digitalRead(BUTTON) == 1){
    stato = 0;
    myFile.close();
    }
   }
  }
 }
}
```



L'insieme dell'elettronica schematizzato nel disegno di cablaggio, montato sul retro del piano del plotter e chiuso con del plexiglass. Gli alimentatori sono sotto.

la definizione dei micropassi e semplificano i cablaggi. I piedini utilizzati per il comando dei due driver sono:

- pin 3 -> STEP
- pin 5 -> DIR

per quello del motore sinistro e:

- pin 8 -> STEP
- pin 9 -> DIR

per quello dello stepper destro.
La macchina è provvista di finecorsa ottici alimentati a 5V e collegati rispettivamente al pin 35 e al pin 39, di un pulsante di azionamento collegato al pin 27, di una spia a LED (sostanzialmente un LED con tanto di resistore di limitazione incorporato nel contenitore) collegata al pin 13 e di un lettore di SD-Card per il funzionamento in modo autonomo collegato secondo lo schema:

- pin50 -> MISO
- pin51 -> MOSI
- pin52 -> SCK
- pin53 ->Z CS

Due alimentatori stabilizzati forniscono energia al sistema: uno da 12V per il movimento dei motori e l'accensione del laser e l'altro da 5V per il funzionamento della scheda di controllo. Il modulo laser é collegato al pin 12 di Arduino MEGA e necessita di un'alimentazione esterna, visto l'assorbimento. In serie al positivo dell'alimentazione del laser va posto il microswitch di sicurezza che la interrompe se si stacca la testina dal piano.
Per la SD-Card l'autore ha previsto un modulo esterno, tuttavia vi ricordiamo che esiste un apposito shield compatibile con la Uno e la MEGA, più pratico perché montabile direttamente a catasta sulla Arduino; l'unico accorgimento da utilizzare è, in questo caso, modificare nello sketch i pin assegnati alla lettura della SD-Card, che utilizzando lo shield diventano:

- D4 -> SD_CS
- D11 -> SD_DI
- D12 -> SD_DO
- D13 -> SD_CLK

Bene, ciò detto vediamo gli aspetti firmware del progetto.

LO SKETCH ARDUINO

Il firmware che andrà caricato nella Arduino MEGA si occupa di azzerare la meccanica della macchina al momento dell'accensione (per azzerare intendiamo portare la testa di stampa nella posizione iniziale) e successivamente attende che venga premuto il pulsante sul pin 27 per iniziare a leggere il file sulla SD-Card e iniziare ad incidere i dati che riceve; il codice corrispondente al firmware per Arduino è riportato nel **Listato 2**.

CONCLUSIONI

Bene, crediamo di avervi detto tutto; il progetto è completamente aperto, quindi si presta alle vostre personalizzazioni ed eventuali espansioni, che vi saranno facili, alla luce della spiegazione fornita in queste pagine. Non possiamo fare altro che augurarvi buon divertimento con un'unica raccomandazione: durante l'utilizzo del plotter state lontani con gli occhi dalla luce del laser e non guardate di lato la macchina, se non altro per evitare la luce rifratta.

per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono di facile reperibilità. Il materiale elencato di seguito può essere acquistato presso Futura Elettronica: il modulo laser blu 2 W - 445 nm (cod. LASER2W) è in vendita a Euro 339,00, la board Arduino MEGA2560 REV3 (cod. ARDUINOMEGAREV3) è disponibile a Euro 43,00, lo shield per Arduino con slot per SD Card (cod. SDCARDSHIELD) costa Euro 12,00, il driver per la scheda controller della stampante 3DRAG (cod. DDRIVER) costa Euro 7,90, l'alimentatore switching 220 VAC/12 VDC 150 W - (cod. FR548) è in vendita a Euro 32,00.

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

Elettronica In - la più prestigiosa rivista di progettazione elettronica

TUTTI i MESI in EDICOLA!

- Progetti pratici
- Articoli didattici
- Corsi hardware e software
- News dal mondo dell'elettronica

...e molto altro ancora.

TI SEI PERSO DEGLI ARTICOLI INTERESSANTI? CREA LA TUA RACCOLTA ARRETRATI SU SUPPORTO USB

Crea con pochi clic una raccolta digitale di arretrati personalizzata. Riceverai, direttamente a casa tua, una card contenente una pen drive USB da 2GB nella quale troverai i pdf dei numeri di Elettronica In che hai scelto.

Le raccolte sono disponibili in tre differenti formati: da 10, da 25 e da 50 arretrati.

ABBONATI! SOLO PER TE ALTRI VANTAGGI:

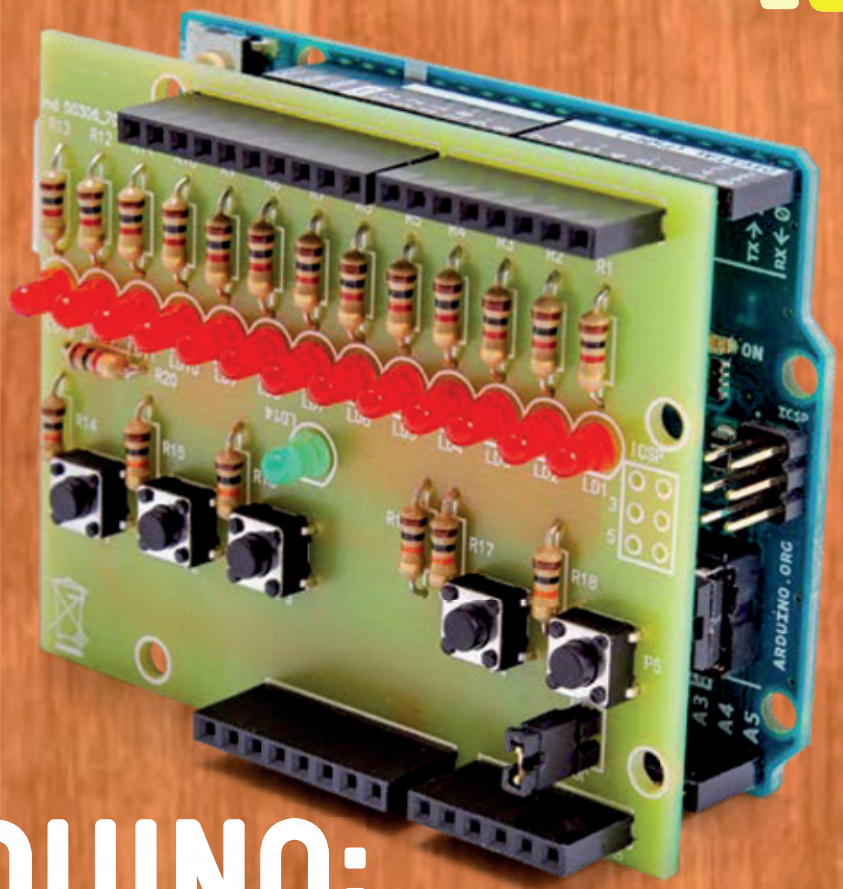
+ La DISCOUNT CARD che ti consentirà di usufruire di uno sconto del 10% su tutti i prodotti acquistati direttamente e on-line da Futura Elettronica.
+ Un volume a scelta della collana "L'Elettronica per tutti".
+ La possibilità di leggere la versione digitale della rivista direttamente da PC.

PER ABBONARTI COLLEGATI AL SITO

www.elettronicain.it

Futura Group Edizioni

FUTURA GROUP srl
Via Adige 11
21013 Gallarate (VA)
TEL.: 0331 752668
Fax: 0331 792287



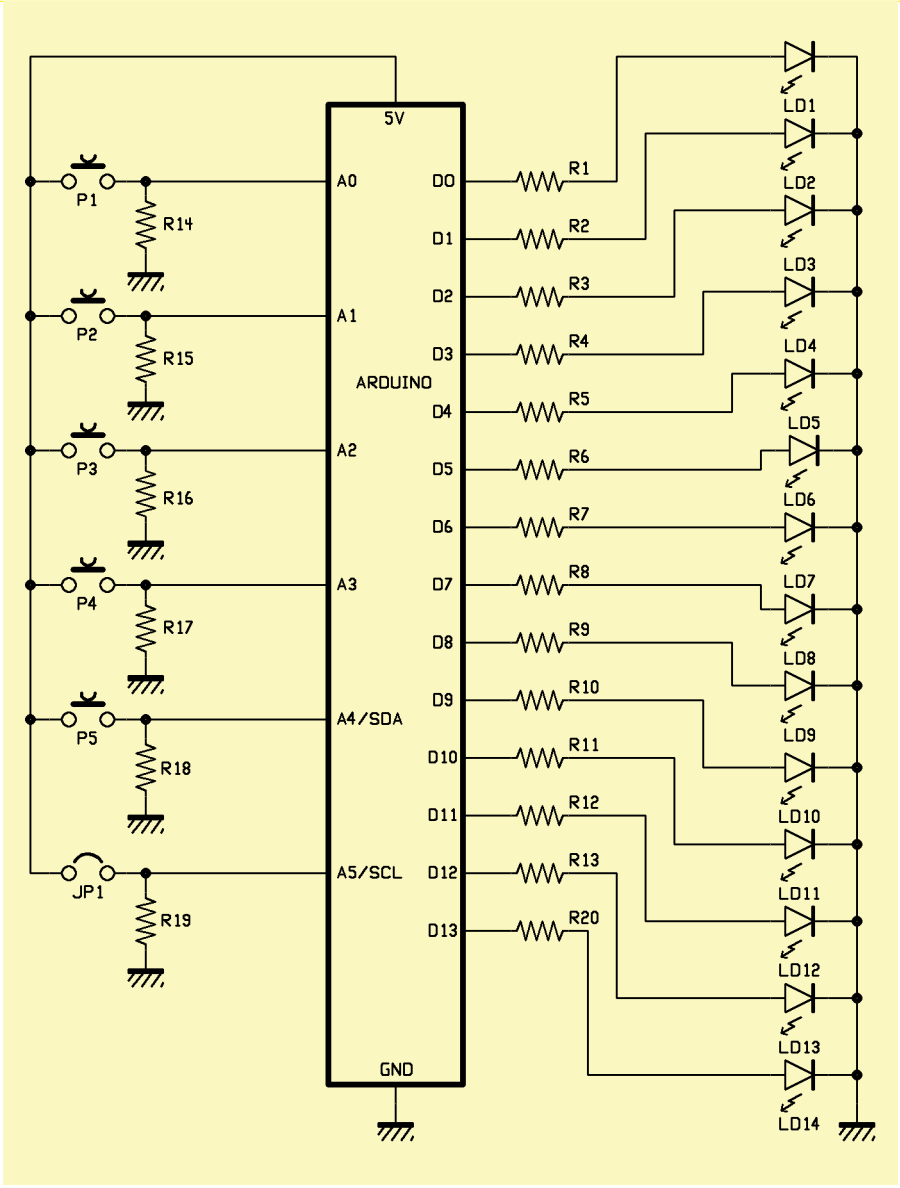
Realizziamo un semplice circuito per giocare in tre modalità differenti utilizzando la scheda Arduino Uno e pochi altri componenti elettronici.

GiocArduino: ARDUINO TI SFIDA IN GIOCHI DI LOGICA

di BRUNO BERTUCCI

In questi anni abbiamo visto Arduino in molte vesti e in tutti gli ambiti possibili, che spaziano dalle applicazioni didattiche e quelle semi-professionali, da quelle più serie alle ludiche: per intendersi, ai giochi e ai gadget. Nei paragrafi che seguono continuiamo sul filone dei giochi, per proporvi un circuito -sempre basato sulla popolare scheda di prototipazione- che vi consentirà di misurarvi in giochi di logica e, per l'esattezza, con tre giochi; questi, pur appartenendo alla tradizione popolare, sono sconosciuti ai più, forse anche a causa del proliferare di giochi elettronici e videogiochi sempre più tecnologici e improntati sull'azione. Abbiamo pensato che, insieme a voi e grazie all'utilizzo della più innovativa delle schede elettroniche, potremmo dare nuova linfa vitale a questi avvincenti giochi classici.

Il progetto originariamente nasce in ambito scolastico, da un'idea sviluppata insieme a un gruppo di studenti di un istituto tecnico (Ludovica Siclari, Vincenzo Buccafurri, Paolo Grimal-



di, Luca Lepera, Rosario Mussari, Carmine Scarpino e Vitaliano Vizzani che ringraziamo per la collaborazione [ndr]). Sempre più spesso si parla della necessità di introdurre nelle scuole italiane l'innovazione didattica e tecnologica come elemento indispensabile per il rilancio della formazione tecnica.
Da sempre Elettronica In è impegnata in primo piano in questo campo, al punto da rappresentare per i lettori -siano essi hobbisti, studenti, docenti, professionisti- una fonte inesauribile di idee da sperimentare in forme sempre nuove.

Il circuito che vi presentiamo rappresenta un nostro ulteriore contributo in tal senso. La nostra "mission", infatti, è divulgare e diffondere l'elettronica a tutti i livelli, come dimostra, oltre un ventennio di rivista e la creazione dell'iniziativa Futura Academy (*http://futura.academy/*), nata con lo scopo di trasmettere le nostre competenze anche agli insegnanti delle scuole a indirizzo tecnico, affinchè loro stessi possano poi metterle a disposizione dei propri studenti per arricchirne la formazione e il bagaglio culturale.
L'applicazione qui descritta, per esempio, oltre che essere un gioco o un gadget da regalare a un figlio, a un conoscente o parente, potrebbe diventare un'occasione per avvicinare tali persone allo studio dell'elettronica attraverso il divertimento.
In ogni caso, se anche siete lontani dal mondo della scuola, potrete proporre ai vostri amici e parenti di accettare la sfida lanciata da Arduino. Il clima di entusiasmo e partecipazione che i giochi realizzabili con esso sono in grado di suscitare nei giocatori coinvolti vi sorprenderà.

SCHEMA ELETTRICO

Per realizzare il nostro gioco trivalente facciamo uso, oltre che di una Arduino (la Uno va benissimo...) di un apposito shield progettato per lo scopo.
Lo schema elettrico di tale shield (che vedete qui accanto) è davvero semplice: i pin digitali di Arduino Uno sono tutti utilizzati come uscite digitali, ognuna delle quali alimenta uno dei tredici LED rossi da 3 mm collegati ai pin 0÷12; un LED di colore verde è invece collegato al pin 13.
I 13 LED rossi servono a simulare i fiammiferi e a dare le segnalazioni durante la partita, come sarà meglio spiegato più avanti; il LED verde è solo per le segnalazioni di gioco.
Le resistenze da 1 kΩ collegate in serie a ciascun LED limitano la corrente circolante in ognuno di essi a 3 mA. Con questo valore di corrente si ottiene una buona luminosità dei LED senza sovraccaricare gli I/O della scheda Arduino, i quali, lo ricordiamo, erogano al massimo 40 mA, ma in tali condizioni la tensione corrispondente allo stato logico cala fortemente (scende a circa 2 V, insufficienti ad accendere LED che non siano

rossi); per avere livelli logici accettabili dobbiamo restare entro i 10 milliampere e nel nostro caso tale condizione è verificata, in quanto con 5V su ciascun I/O in corrispondenza dell'1 logico e una caduta sui LED di poco inferiore ai 2 volt, in ogni resistenza (e quindi in ciascun LED acceso) scorrono 3 milliampere.
Avendo già utilizzato tutti i pin digitali di Arduino come uscite destinate all'accensione del LED, abbiamo fatto ricorso a un semplice artificio che ci ha consentito di utilizzare gli ingressi analogici alla stregua di quelli digitali. Come sappiamo, gli ingressi analogici di Arduino leggono dei valori compresi tra un minimo di 0 e un massimo di 5 volt e restituiscono un numero intero variabile tra 0 e 1.023, essendo il convertitore analogico digitale (ADC) utilizzato dalla scheda a 10 bit.
Tramite delle resistenze da 10 kΩ gli ingressi analogici del microcontrollore sono forzati ad assumere la tensione di 0 volt, per scongiurare la lettura di valori casuali. Solo a seguito della pressione dei pulsanti P1÷P5, che fanno capo agli ingressi A0÷A4, o alla chiusura del ponticello JP1, che fa capo all'ingresso A5, sul pin analogico corrispondente sarà presente una tensione di 5 volt; questo perché un capo dei pulsanti e del jumper è collegato alla linea dei 5V di Arduino.
Il codice scritto per Arduino è in grado di distinguere queste due condizioni di funzionamento e di associare a delle variabili booleane due stati logici: *false* (corrispondente a valori restituiti dal convertitore ADC inferiori a 800) e *true* (corrispondente a valori restituiti dal convertitore ADC maggiori di 800).
Dunque, Arduino leggerà lo stato dei pulsanti e del jumper J1 attraverso il proprio ADC sfruttando la tensione presente ai capi delle rispettive resistenze di pull-down.

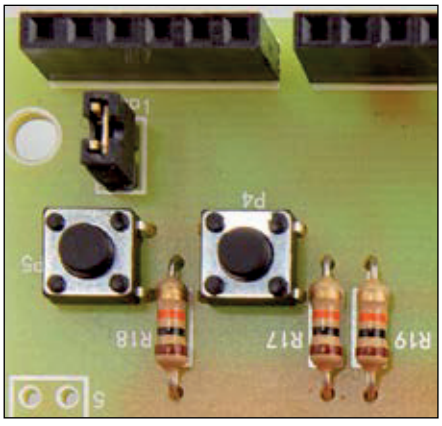
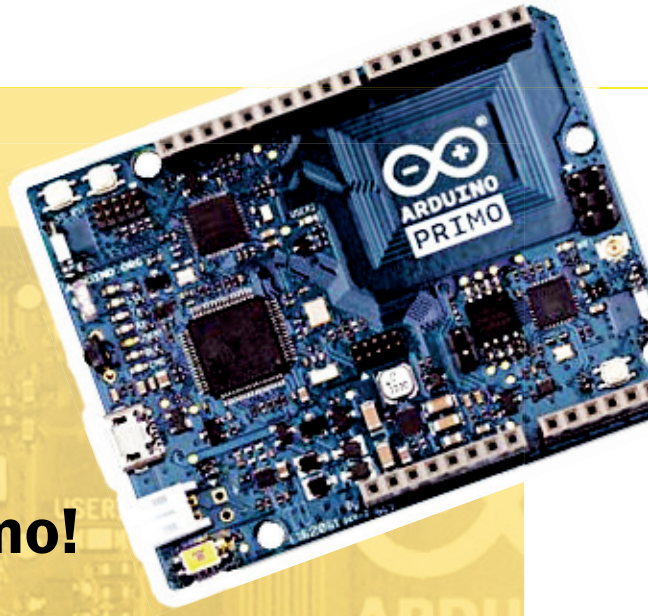


***Fig. 1** - Posizionamento del ponticello JP1 per impostare la modalità di gioco casuale.*



Arduino ora è **primo!**

La popolare scheda di prototipazione fa il suo ingresso nel mondo dell'Internet of Things con due novità all'insegna della connettività. L'universo Arduino vede infatti brillare due nuove stelle, chiamate Arduino Primo e Arduino Primo Core, presentate poco dopo il Maker Faire di San Francisco.
Non solo IoT, comunque, ma una proposta destinata anche a maker, sviluppatori e a chiunque abbia il desiderio di applicarsi nello sviluppo di progetti innovativi.
Progettate per lavorare insieme, le due board dispongono nativamente di connettività con sensori a infrarossi e Bluetooth grazie a un chip fornito dal neo-partner Nordic Semiconductors; è implementata anche la connettività Wi-Fi, anche se presente al momento nella sola Arduino Primo. Le due schede sono quindi connesse con più di un link, integrando funzionalità che in genere richiedevano l'aggiunta di moduli supplementari; in un certo senso vanno nella direzione in cui già ci siamo mossi un anno fa proponendo le nostre Fishino.
La Primo è espandibile con gli shield e una porta MicroUSB e si potrà programmare tramite l'IDE, ma anche con strumenti più potenti. La Core è una micro scheda circolare che si attiva con una pila "a pastiglia"; si immette in una "carrier board" (chiamata AlicePad) che fornisce la connettività USB e l'elettricità per la ricarica delle batterie della Core stessa.

IL PRIMO GIOCO PROPOSTO

Il meccanismo del primo gioco è molto semplice. Disposti 13 fiammiferi su un piano, a turno,

Listato 1

```
//*************************************************************************************************************************
//*** gioca ***************************************************************************************************************
//*************************************************************************************************************************

void gioca() {

......

      if ((mossaAllaMacchina==true) && (abilitaMossa==true) && (vinceGiocatore==false) && (vinceMacchina==false)) {
          fiammiferiDaPrendere = random(1,4);                 // Arduino inizia prendendo un numero a caso compreso tra
          if (fiammiferiDaPrendere+fiammPresi >= 13) {        // 1 e 3 per non far scoprire il gioco all'avversario
              fiammiferiDaPrendere = 1;                       // Arduino evita di perdere finché è possibile
          }
          if (principiante==false) {                          // Arduino gioca secondo la strategia ottimale per vincere:
              if ((fiammPresi+1) % 4 == 0) {                  // Dopo la giocata del secondo giocatore il numero totale di
                  fiammiferiDaPrendere = 1;                   // fiammiferi raccolti deve essere pari a 4 o un suo multiplo
              }
              if ((fiammPresi+2) % 4 == 0) {
                  fiammiferiDaPrendere = 2;
              }
              if ((fiammPresi+3) % 4 == 0) {
                  fiammiferiDaPrendere = 3;
              }
          }
 ......
```

due giocatori raccolgono un numero variabile tra uno e tre di essi per volta. Perde il giocatore che raccoglie l'ultimo fiammifero.
Nel nostro progetto i fiammiferi sono sostituiti da 13 LED rossi e la raccolta di questi da parte del giocatore è effettuata tramite tre pulsanti. Altri due pulsanti sono utilizzati per stabilire chi inizia a giocare per primo.
Avrete intuito certamente che c'è il trucco: perde sempre chi gioca per primo. Pertanto se cominciate voi a giocare non avete alcuna possibilità di vittoria. Allo stesso modo, se Arduino inizia a giocare per primo è destinato a perdere. Se però sbagliate anche una sola mossa, Arduino recupera lo svantaggio e vince.
Abbiamo pensato di svelarvi la strategia ottimale, che dovrete usare per vincere, "nascondendola" tra i commenti del codice, per non togliervi il gusto di riuscire da soli e senza aiuti a battere Arduino.
Inoltre, per rendere più equilibrata la partita, il codice prevede che il microcontrollore possa giocare in modo casuale, così come farebbe un principiante. Basta chiudere il ponticello JP1 sulla scheda, come mostrato in **Fig. 1**, in modo da portare l'ingresso analogico A5 a 5 volt. Questa opzione risulta interessante anche nel caso in cui vorrete far capire gradualmente il trucco all'ignaro giocatore che avrete invitato a sfidare Arduino.
Le istruzioni per giocare sono riportate di seguito. Fate riferimento alla **Fig. 2** per individuare i pulsanti e i LED che vengono richiamati nella procedura.

1. se tutti i LED rossi lampeggiano, vuol dire che non è stata effettuata la scelta iniziale su chi deve cominciare; ciò avviene sia al primo avvio di Arduino, sia dopo che sono trascorsi tre secondi dalla premiazione del vincitore;
2. per iniziare a giocare bisogna premere il pulsante P5 (se si desidera che Arduino faccia la prima mossa) o il pulsante

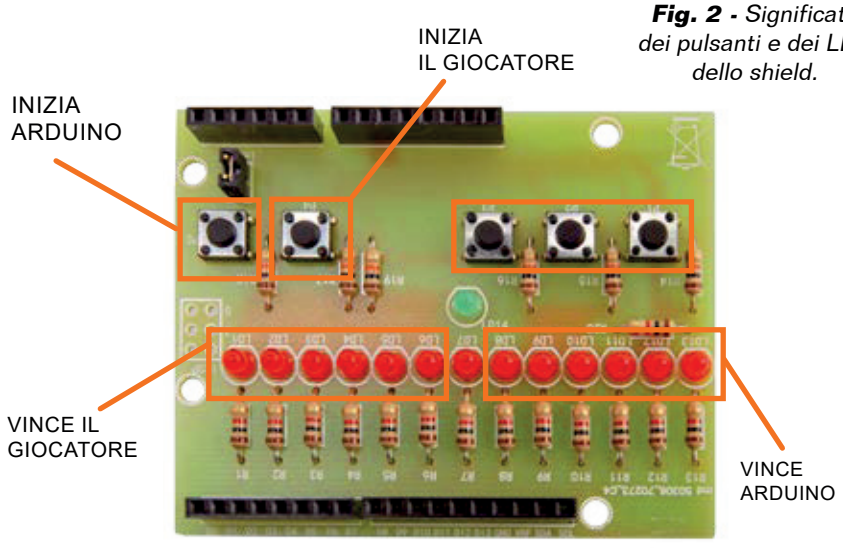


***Fig. 2** - Significato dei pulsanti e dei LED dello shield.*

P4 (se si desidera che la prima mossa spetti al giocatore);
3. inizialmente vengono accesi tutti i 13 LED rossi;
4. il turno del giocatore è segnalato dal lampeggio del LED verde LD14; il lampeggio termina appena il giocatore compie la sua scelta, premendo il pulsante P1 per prendere tre fiammiferi, il pulsante P2 per prenderne due e il pulsante P3 per prendere un fiammifero;
5. quando è il turno di gioco di Arduino, la sua mossa è ritardata di 2 secondi;
6. se vince il giocatore, i sei LED rossi LD1÷LD6 lampeggiano per tre secondi;
7. se vince Arduino, i sei LED rossi LD1÷LD6 lampeggiano per tre secondi;
8. passati 3 secondi dalla premiazione si ritorna al punto 1; comunque la scelta di iniziare una nuova partita può essere effettuata in qualsiasi istante.

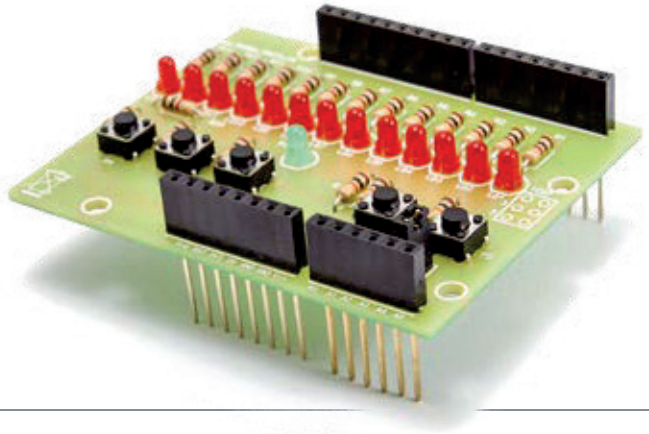
***Tabella 1** - Strategia vincente per il secondo gioco: si possono raccogliere uno o due fiammeferi e vince chi raccoglie l'ultimo fiammifero. Il gioco è attivo quando il ponticello JP1 sulla scheda è aperto.*

Numero di fiammiferi	Vincitore	Note sulla strategia vincente
1	Chi gioca per primo	Il giocatore che gioca per primo vince sicuramente perché raccoglie l'unico fiammifero
2	Chi gioca per primo	Il giocatore che gioca per primo vince sicuramente perché raccoglie due fiammiferi.
3	Chi gioca per secondo	Per vincere il secondo giocatore raccoglierà un fiammifero, se il primo giocatore ne raccoglie due, o due fiammiferi, se il primo giocatore ne raccoglie uno.
4	Chi gioca per primo	Il giocatore che gioca per primo vince sicuramente se raccoglie un fiammifero, poiché riconduce il gioco alla situazione con 3 fiammiferi.
5	Chi gioca per primo	Il giocatore che gioca per primo vince sicuramente se raccoglie 2 fiammiferi, poiché riconduce il gioco alla situazione con 3 fiammiferi.
6	Chi gioca per secondo	Il giocatore che gioca per secondo vince sicuramente se raccoglie 2 fiammiferi e riconduce il gioco alla situazione con 3 fiammiferi.
7	Chi gioca per primo	La stessa strategia del caso di 4 fiammiferi.
8	Chi gioca per primo	La stessa strategia del caso di 5 fiammiferi.
9	Chi gioca per secondo	La stessa strategia del caso di un numero di fiammiferi multiplo di 3.
10	Chi gioca per primo	La stessa strategia del caso di 4 fiammiferi.
11	Chi gioca per primo	La stessa strategia del caso di 5 fiammiferi.
12	Chi gioca per secondo	La stessa strategia del caso di un numero di fiammiferi multiplo di 3.
13	Chi gioca per primo	La stessa strategia del caso di 4 fiammiferi.

LO SKETCH PER IL PRIMO GIOCO

Lo sketch da installare su Arduino per il primo gioco è riportato nel **Listato 1**. Esso è contenuto nel file *GiocoDeiFiammiferi-1.ino*, scaricabile dal nostro sito Internet *www.elettronicain.it*. Abbiamo pensato di strutturare il codice in modo da assicurarne un'elevata leggibilità. Esso è stato anche documentato, a tutto vantaggio di chi è alle prime armi. Pertanto nel loop sono presenti i richiami alle varie sezioni di codice che gestiscono il gioco. Come potete notare, il tutto è organizzato per far ragionare la scheda un po' come faremmo noi al suo posto:

```
void loop() {
     leggiIngressi();
     segnalaEseguiSceltaIniziale();
     segnalaTurnoDelGiocatore();
     stabilisciChiGiocaPerPrimo();
     gioca();
     premiaVincitore();
}
```



Infatti, il microcontrollore Atmega contenuto in Arduino interroga gli ingressi per vedere quali sono le mosse dell'avversario, stabilisce se bisogna iniziare una nuova partita o segnalare che ancora non è stata fatta la scelta iniziale, giocare e, infine, premiare il vincitore con l'accensione dei LED di uno o dell'altro lato (vedere la **Fig. 2**).

***Tabella 2** - Strategia vincente per il terzo gioco: si possono raccogliere uno o due fiammiferi e perde chi raccoglie per ultimo il fiammifero. Il gioco è attivo quando il ponticello JP1 sulla scheda è chiuso.*

Numero di fiammiferi	Vincitore	Note sulla strategia vincente
1	Chi gioca per secondo	Il giocatore che gioca per primo perde sicuramente perché raccoglie l'unico fiammifero.
2	Chi gioca per primo	Il giocatore che gioca per primo vince sicuramente perché raccoglie un fiammifero lasciando l'ultimo all'avversario.
3	Chi gioca per primo	Il giocatore che gioca per primo vince sicuramente perché raccoglie due fiammiferi lasciando l'ultimo all'avversario.
4	Chi gioca per secondo	Per vincere il secondo giocatore raccoglierà un fiammifero, se il primo giocatore ne raccoglie due, o due fiammiferi, se il primo giocatore ne raccoglie uno.
5	Chi gioca per primo	Il giocatore che gioca per primo vince sicuramente se raccoglie un fiammifero, poiché riconduce il gioco alla situazione con 4 fiammiferi.
6	Chi gioca per primo	Il giocatore che gioca per primo vince sicuramente se raccoglie due fiammiferi, poiché riconduce il gioco alla situazione con 4 fiammiferi.
7	Chi gioca per secondo	Il giocatore che gioca per secondo vince sicuramente, poiché riconduce il gioco alla situazione con 4 fiammiferi (raccoglierà un fiammifero, se il primo giocatore ne raccoglie due, o due fiammiferi, se il primo giocatore ne raccoglie uno).
8	Chi gioca per primo	La stessa strategia del caso di 5 fiammiferi.
9	Chi gioca per primo	La stessa strategia del caso di 6 fiammiferi.
10	Chi gioca per secondo	La stessa strategia del caso di un numero di fiammiferi multiplo di 3 e aumentato di 1.
11	Chi gioca per primo	La stessa strategia del caso di 5 fiammiferi.
12	Chi gioca per primo	La stessa strategia del caso di 6 fiammiferi.
13	Chi gioca per secondo	La stessa strategia del caso di un numero di fiammiferi multiplo di 3 e aumentato di 1.

GLI ALTRI DUE GIOCHI

Il secondo gioco che vi proponiamo è una variante del primo: il numero di fiammiferi da utilizzare può essere compreso fra 4 e 13; disposti i fiammiferi su un piano, a turno, i due giocatori raccolgono uno o due di essi per volta. Vince la partita il giocatore che raccoglie l'ultimo fiammifero. Per selezionare questo gioco è necessario che il ponticello JP1 sullo shield, mostrato in **Fig. 1**, sia aperto.

Infine vediamo il terzo gioco, che è simile al secondo; questa volta, però, perde la partita il giocatore che raccoglie l'ultimo fiammifero. Per selezionare questo gioco è necessario che il ponticello JP1 sullo shield, mostrato in **Fig. 1**, sia chiuso.

Anche in questo caso c'è il trucco: vince sempre un determinato giocatore. Ovviamente ciò avviene solo se questi gioca secondo una strategia ottimale.

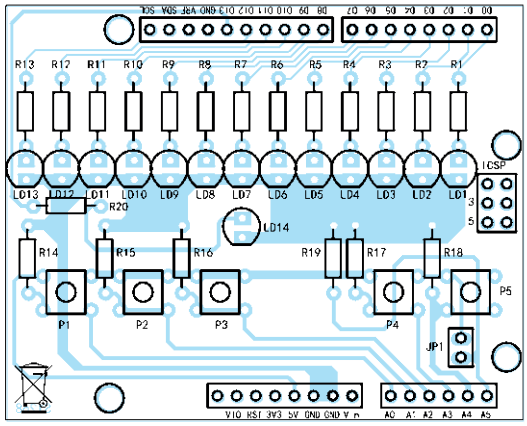
Questa strategia, che cambia al mutare delle condizioni di gioco, dipende dal gioco selezionato, dal numero di fiammiferi totali usati e da quale giocatore inizia

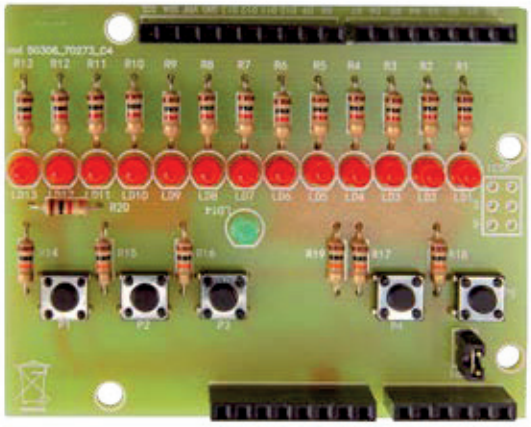
[piano di **MONTAGGIO**]

Elenco Componenti:

P1÷P5: Microswitch
R1÷R13, R20: 1 kohm
R14÷R19: 10 kohm
LD1÷LD14: LED 3mm rosso

Varie:
- Jumper
- Strip M/F 10 vie
- Strip M/F 6 vie
- Strip M/F 8 vie (2 pz.)
- Strip M/F 2x3 vie
- Strip maschio 2 vie
- Circuito stampato S1268 (68 x 54 mm)





Listato 2

```
//*************************************************************************************************************
//*** giocoUno() *************** VINCE CHI RACCOGLIE L'ULTIMO FIAMMIFERO *********************************
//*************************************************************************************************************

void giocoUno() {

......

     if ((mossaAllaMacchina==true) && (abilitaMossa==true) && (vinceGiocatore==false) && (vinceMacchina==false)) {
          switch (fiammiferiGioco-fiammPresi){  // La strategia vincente è indicata di seguito (vedi tabella 1)
            case  1:
                fiammiferiDaPrendere = 1;
                vinceMacchina=true;
                break;
            case  2:

             ......

           case  13:
......



//*************************************************************************************************************
//*** giocoDue() *************** PERDE CHI RACCOGLIE L'ULTIMO FIAMMIFERO *********************************
//*************************************************************************************************************

void giocoDue() {

......

      if ((mossaAllaMacchina==true) && (abilitaMossa==true) && (vinceGiocatore==false) && (vinceMacchina==false)) {
          switch (fiammiferiGioco-fiammPresi){  // La strategia vincente è indicata di seguito (vedi tabella 2)
            case  1:
                fiammiferiDaPrendere = 1;
                vinceGiocatore=true;
                break;
            case  2:

             ......

           case  13:
......
```

per primo. In verità, più che di un trucco si tratta di una formula numerica. La strategia ottimale di gioco, infatti, ha molto a che fare con la matematica e con la logica. Avvalendovi della **Tabella 1**, della **Tabella 2** e dei commenti del codice potrete approfondire, se ne avete voglia, questo aspetto.

Le istruzioni per giocare ai due nuovi giochi sono riportate di seguito.

A differenza del gioco precedente, per consentire al giocatore di scegliere il numero totale dei fiammiferi con cui giocare, è stata predisposta una semplice procedura di programmazione; fate riferimento alla **Fig. 2** per individuare facilmente i pulsanti e i LED che vengono richiamati nella lista.

La procedura è di seguito descritta passo per passo.

1. Se tutti i LED rossi lampeggiano, vuol dire che non è stata effettuata la scelta iniziale su chi deve cominciare. Ciò avviene sia al primo avvio di Arduino, sia dopo che sono trascorsi tre secondi dalla premiazione del vincitore.
2. Per entrare in modalità programmazione bisogna premere il pulsante P5 (se si desidera che Arduino faccia la prima mossa) o il pulsante P4 (se si desidera che la prima mossa spetti al giocatore).
3. La modalità di programmazione è segnalata dall'accensione del LED verde LD14.
4. Inizialmente risulteranno accesi i quattro LED rossi LD1÷LD4. A ogni pressione del pulsante P1 il numero di LED rossi accesi aumenterà di uno. Dopo l'accensione del tredicesimo LED rosso si riparte dai quattro LED accesi.
5. Per uscire dalla programmazione occorre premere insieme i pulsanti P2 e P3. Il LED verde LD14 si spegne e subito dopo il gioco comincia.
6. Il turno del giocatore è segna-

lato dal lampeggio del LED verde LD14. Il lampeggio termina appena il giocatore compie la sua scelta, premendo il pulsante P3 per prendere due fiammiferi e il pulsante P2 per prenderne uno.

7. Quando è il turno di gioco di Arduino, la sua mossa è ritardata di due secondi.
8. Se vince il giocatore, i sei LED rossi LD1÷LD6 lampeggiano per tre secondi.
9. Se vince Arduino, i sei LED rossi LD1÷LD6 lampeggiano per tre secondi.
9. Passati 3 secondi dalla premiazione del giocatore evidenziato dai LED, si ritorna al punto 1. Comunque la scelta di iniziare una nuova partita può essere effettuata in qualsiasi istante, quindi si può anche ricominciare senza attendere il termine della partita.

LO SKETCH PER IL SECONDO E IL TERZO GIOCO

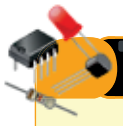
Passiamo adesso a esaminare l'ultimo sketch da installare in Arduino, che è quello necessario per implementare il secondo e terzo gioco; tale firmware è contenuto nel file *GiocoDeiFiammiferi-2e3.ino* scaricabile dal nostro sito Internet *www.elettronicain.it* insieme agli altri file del progetto.
Anche in questo caso il codice è strutturato in modo da assicurarne un'elevata leggibilità ed è ben documentato.
La parte principale è il loop, come visibile qui di seguito:

```
void loop() {
     leggiIngressi();
     segnalaEseguiSceltaIniziale();
     segnalaTurnoDelGiocatore();
     impostaChiGiocaPerPrimo_Fiam-
  miferiGioco();
     switch (gioco2){
        case  false:
            giocoUno();
            break;
        case  true:
            giocoDue();
            break;
     }
     premiaVincitore();
}
```

In base allo stato del ponticello JP1 il programma principale richiama due diverse sezioni di codice per gestire il gioco. Le strategie ottimali di gioco riportate nella **Tabella 1** e **Tabella 2**, vengono implementate, utilizzando la struttura switch ... case, in due differenti sottoprogrammi, la cui parte iniziale è quella che vedete nel **Listato 2** .

REALIZZAZIONE PRATICA

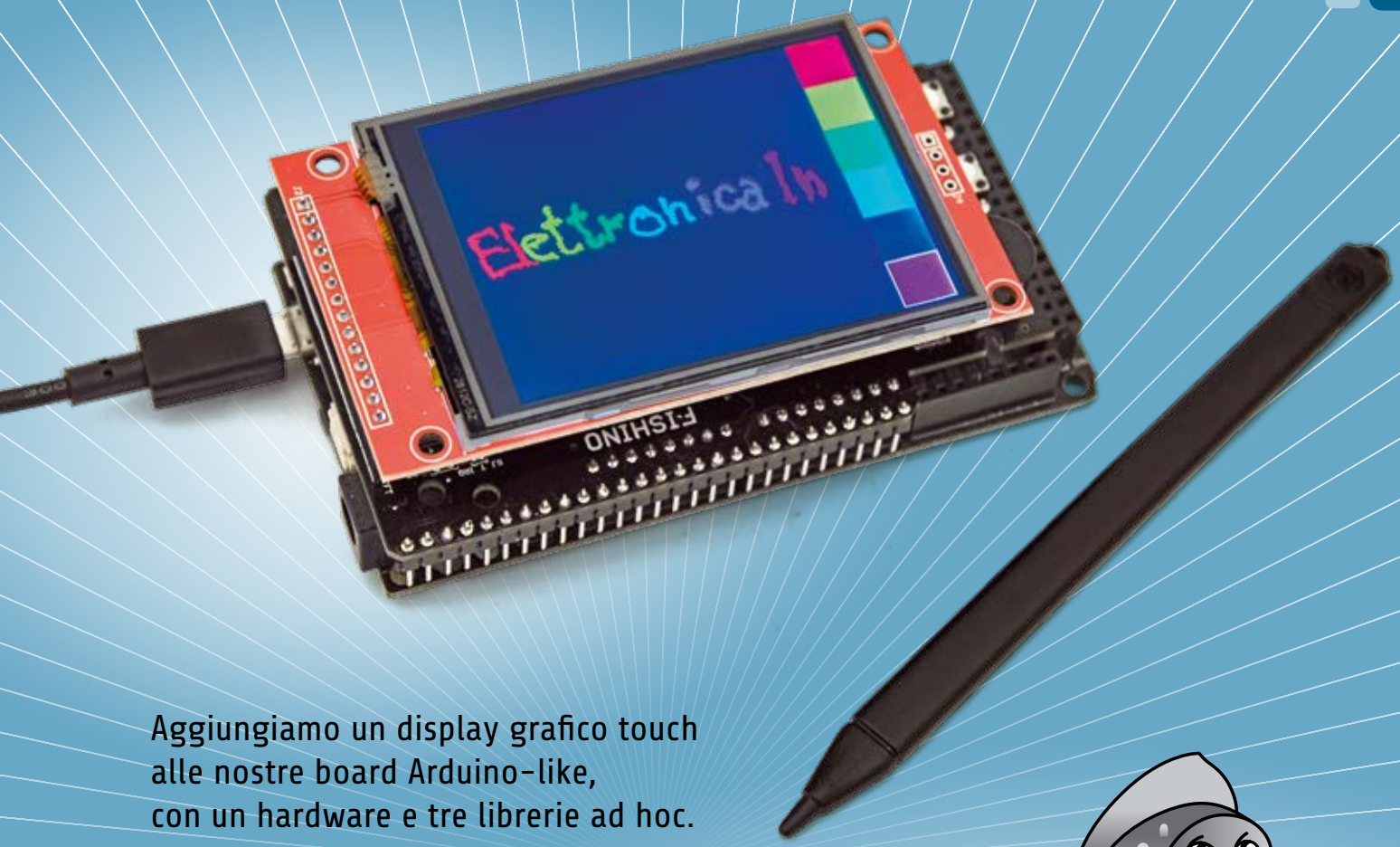
Descritto l'hardware, ossia lo shield, e analizzato il firmware per i tre giochi, possiamo passare alla descrizione della costruzione del circuito; data l'estrema semplicità dello schema, potreste anche montare il circuito direttamente su una basetta millefori o su una scheda breadboard, comunque abbiamo previsto un circuito stampato di cui sul nostro sito Internet *www.elettronicain.it* pubblichiamo la traccia lato rame della basetta.
Il c.s. è del tipo a semplice ramatura, quindi realizzarlo è facile e alla portata di tutti, dato che non è richiesto di allineare le due tracce né il doppio passaggio di esposizione nel bromografo con il relativo sviluppo, né occorre creare le vie tra le due facce.
Anche la scelta dei componenti utilizzati, che sono tutti a montaggio tradizionale (THT) va nella direzione di rendere la realizzazione dello shield a portata di chiunque sappia utilizzare un saldatore.
Dunque, incisa e forata la basetta potete iniziare il montaggio dei componenti: partite dalle resistenze e dai pulsanti miniatura, che sono i componenti a più basso profilo, quindi sistemate i LED (riferitevi, per l'orientamento, al piano di montaggio visibile nelle pagine precedenti e ricordate che nei LED il catodo è l'elettrodo che sta dalla parte dello smusso del contenitore) e i connettori strip femmina a passo 2,54 mm con piedini lunghi 20 mm. Proseguite montando il pin-strip a due poli per il jumper J1.
Fatto ciò il montaggio è completato: verificato che tutti i componenti siano al loro posto e orientati correttamente potete inserire lo shield sulla vostra Arduino, quindi caricare il firmware e iniziare a giocare.
Se vi è più comodo potete inserire l'insieme in un contenitore in plastica di dimensioni adeguate, prolungando l'attuatore dei pulsanti con dei tondini di plastica (PVC) piena del diametro di 5 o 6 mm. È anche pensabile cambiare tipo di pulsanti, utilizzandone di più grandi e magari colorati, da applicare a un pannello della scatola collegandoli poi alle rispettive piazzole dello shield tramite spezzoni di filo in guaina.
Detto ciò non c'è altro da aggiungere; ci resta solo da augurarvi buon divertimento!



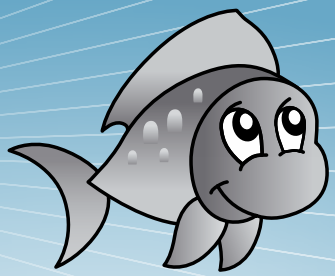
per il MATERIALE

Tutto il materiale utilizzato in questo progetto è di facile reperibilità. Il master del circuito stampato e i file degli sketch possono essere scaricati dal sito della rivista.

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



Aggiungiamo un display grafico touch alle nostre board Arduino-like, con un hardware e tre librerie ad hoc.



TFT SHIELD PER FISHINO

di MASSIMO DEL FEDELE

Dopo avervi presentato ben quattro schede di prototipazione della serie Fishino (la UNO, la Fishino 32, la MEGA, la Guppy...ed una quinta è in arrivo!), abbiamo iniziato a sentire il bisogno di a proporre un po' di elettronica di contorno. In particolare, abbiamo sentito la mancanza di un display da abbinare loro, per corredare le applicazioni con output visivi che non fossero semplicemente LED accesi o spenti. Ma non ci bastava un semplice display per testo (come ad esempio i comunissimi alfanumerici a 2 o 4 righe basato su controller HD44780), perché volevamo un bello schermo grafico caratterizzato da dimensioni ridotte e una discreta risoluzione, sul quale mostrare icone, simboli e fare un po' di grafica. Ci siamo quindi messi alla ricerca di qualcosa di esistente, riscontrando, tuttavia, che i pur molti display (e shield che li supportano) hanno parecchie limitazioni.
Per questo abbiamo voluto realizzarci qualcosa su misura, ossia uno shield con a bordo un prestante display, che vi descriviamo in questo articolo.
Partiamo dall'inizio, ossia dalla fase di progetto,

Fig. 1
L'assemblaggio del display sullo shield.

dicendo che volevamo qualcosa in grado di soddisfare i seguenti requisiti:

- dimensioni sufficientemente ridotte (2,4 o al massimo 3,2 pollici);
- risoluzione minima di 320x240 pixel, con profondità di colore elevata;
- interfaccia touch che ci permetta di interagire con il nostro Fishino;
- impegno di poche risorse hardware della scheda, in modo da poterlo utilizzare anche con i modelli di Fishino dotati di pochi I/O (la Guppy, per esempio);
- uno shield che si adattasse a tutti i prodotti della serie Fishino ed anche a molti della serie Arduino, senza bisogno di connessioni volanti
- la possibilità di collegare altri componenti insieme allo shield, quindi l'esposizione di tutti gli I/O delle varie schede anche con il display montato.

Crediamo di essere riusciti nei nostri propositi quasi al 100% (vedremo in seguito alcuni piccoli "limiti", facilmente superabili), con un prodotto interessante e di utilizzo semplicissimo, che vi presenteremo in queste pagine.

Per renderne ancora più agevole l'utilizzo abbiamo provveduto a preparare tre apposite librerie, due delle quali di derivazione Adafruit, in grado di gestire il display stesso, ed una totalmente sviluppata da noi per la gestione del touch-panel che esso incorpora.

UNO SHIELD "UNIVERSALE"

Contrariamente al solito, inizieremo la descrizione dello shield...dalla fine! Infatti, la cosa più complessa nello sviluppo del progetto è stata proprio la creazione del layout della scheda, dovendo abbinare ad essa board anche parecchio diverse le une dalle altre. Abbiamo quindi iniziato con lo sviluppo "grafico" della scheda, per poi andare a ritroso fino ad ottenere lo schema elettrico.

Iniziamo dall'ingombro: il primo requisito era quello di ottenere una scheda con dimensioni paragonabili alla nostra Fishino UNO (ma anche alla Fishino32 e ad Arduino UNO); quindi il display va sovrapposto senza o quasi aumentarne l'ingombro laterale: si sviluppa tutto nello spessore. Abbiamo quindi optato per un "sandwich" consistente, nell'ordine, dal display TFT, dallo shield e, infine, dalla scheda a microcontroller; la **Fig. 1** chiarisce meglio il concetto.

In questa immagine potete vedere il "mockup" (modello grafico) dello shield con display ed un Fishino MEGA montati, che è stata la nostra base di partenza.

L'esigenza di accedere a tutti gli I/O della scheda ha imposto l'aggiunta, lateralmente, di due file di connettori maschio paralleli a quelli destinati ad infilarsi nei connettori femmina della scheda, ai quali è possibile connettere i consueti cavetti Dupont.

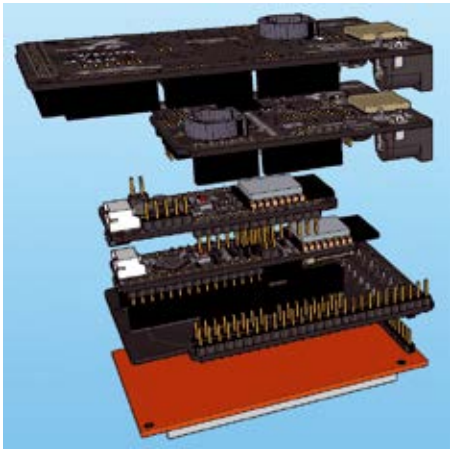


Fig. 2
Esploso dello shield e delle board Fishino.

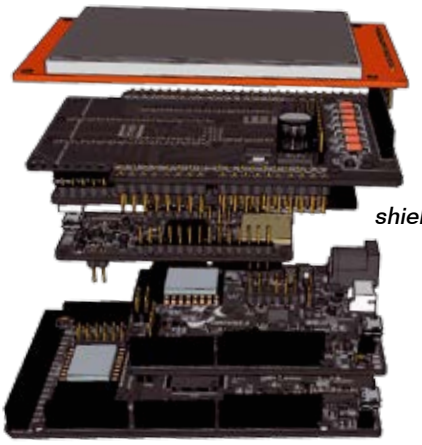


Fig. 3
Esploso dello shield e delle board Fishino, visto da sopra.

Una volta stabilito a grandi linee il layout della scheda, la dimensione del display è stata fissata in 2,8 pollici come massimo (può accettare anche il display più piccolo da 2,4 pollici), in modo da non superare l'ingombro previsto.
Un altro parametro che ha vincolato la dimensione è stato l'esigenza di utilizzare il minor numero di I/O possibili; abbiamo quindi scartato tutti i display con interfaccia parallela (8 ma anche 16 bit) che avrebbero occupato praticamente tutte le risorse delle schede più "piccole" e ci siamo orientati verso un display con comunicazione SPI, reperibile facilmente nel formato, appunto, di 2,4 pollici o 2,8 pollici. Questo display utilizza le linee SPI (MISO, MOSI e SCK), in comune con il modulo WiFi e le schede SD (ed altro eventuale!), una linea di selezione ed una di controllo per il display ed altre 2 linee per lo schermo touch-sensitive, quindi quattro I/O utilizzati in "esclusiva" contro 10÷12 minimo per i modelli ad interfaccia parallela.
Come vedremo in seguito, abbiamo poi aggiunto altre funzionalità opzionali, ottenibili con l'utilizzo di pochi altri I/O.
Fissati quindi layout e display, occorreva trovare un modo per poter "infilare" tutti i modelli di board previsti sullo stesso shield. Visto che la tecnologia attuale lo permette, abbiamo realizzato i modelli 3D di tutte le schede, il modello 3D del display ed il modello 3D iniziale dello shield, in modo da poterli sovrapporre graficamente e visualizzare eventuali interferenze; nella **Fig. 2** e nella **Fig. 3** trovate i disegni in esploso visti dal basso e dall'alto del nostro shield con sovrapposte un buon numero di schede Fishino. Resta inteso che di Fishino se ne monta una sola per volta.
Abbiamo quindi cercato la posizione migliore per le schede "piccole" (Fishino Guppy/Arduino Nano e la futura Fishino Piranha, pin-out compatibile con l'Arduino MKR1000) in modo da non avere interferenze tra i connettori (cosa impossibile al 100%, come vedremo a breve) e da poter restare negli ingombri previsti.
Ecco quindi nascere il layout definitivo della scheda, mostrato nella **Fig. 4** dal lato di inserimento dei controller (vale a dire delle schede Fishino/Arduino).
Tutto perfetto, quindi? Purtroppo no, perché le schede Fishino UNO e 32 hanno due connettori ausiliari, per la precisione il connettore ESPCONN sulla UNO ed il connettore ICSP sulla 32 che vanno ad interferire con i connettori delle schede "piccole" (Nano/Guppy e Piranha/MKR1000); l'inserimento è fisicamente possibile, ma i segnali non sono compatibili con i circuiti della scheda.

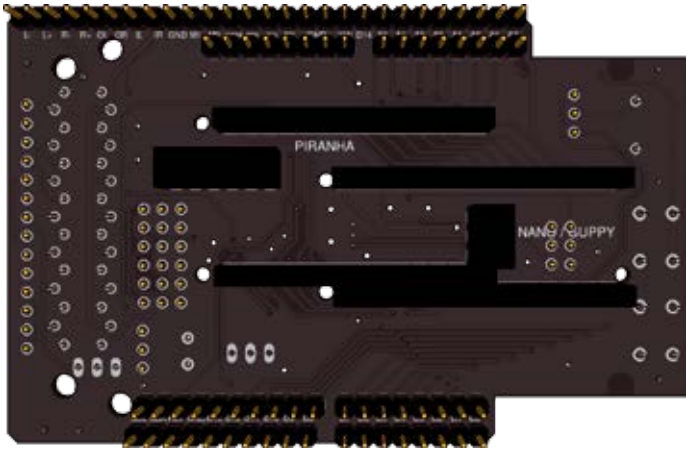


***Fig. 4** - Layout dei connettori dello shield.*

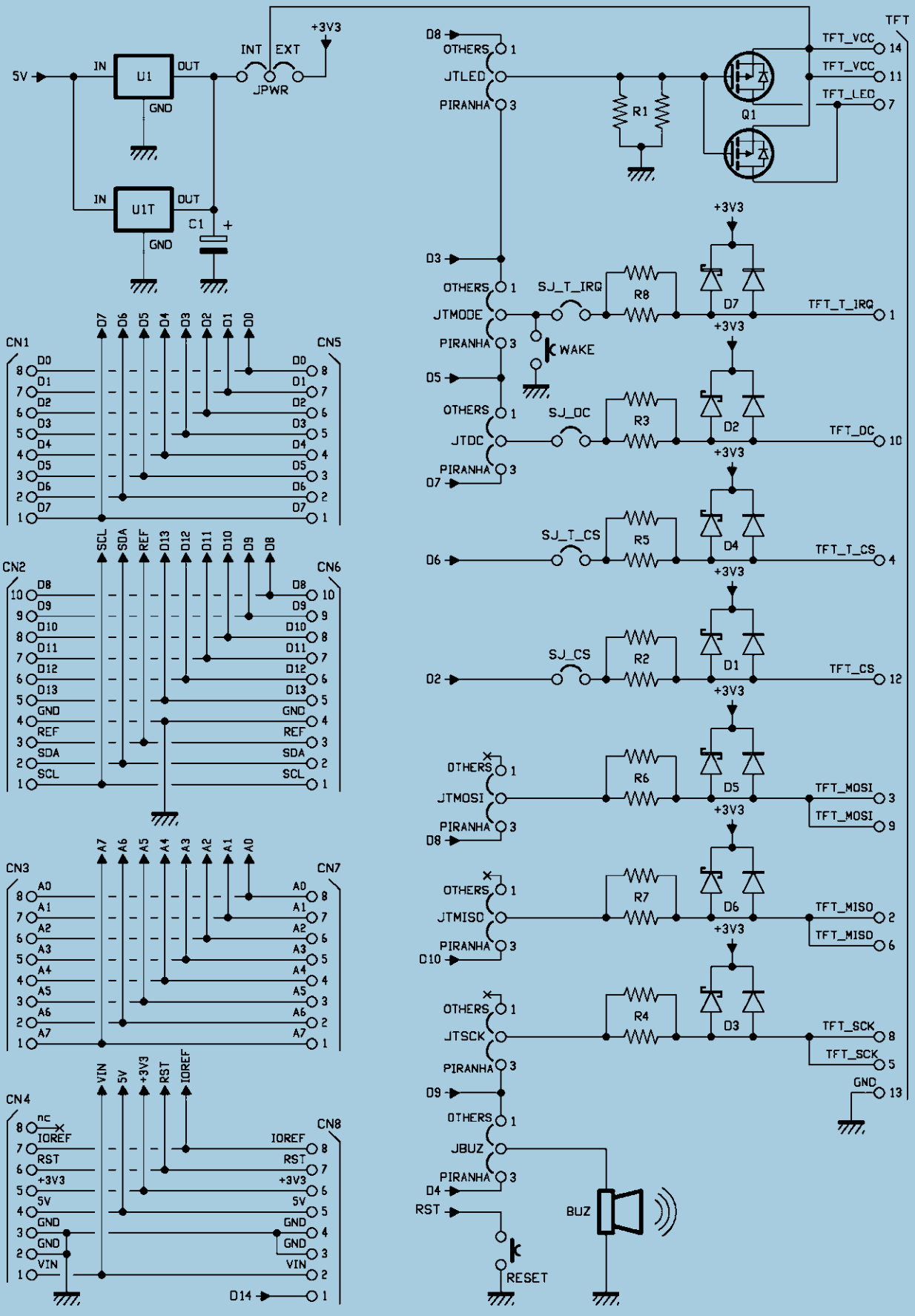
Il "problema" non è risolvibile, visto che alcuni connettori devono avere una posizione ben precisa gli uni con gli altri: per esempio il connettore ISP che porta i segnali SPI sia dell'UNO che del Mega.
Per ovviare all'inconveniente ci sono più possibilità:

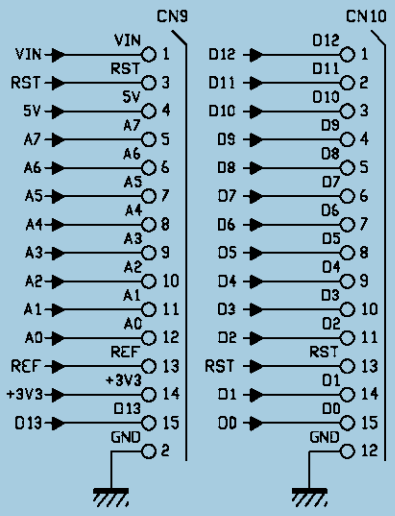
- montare solo i connettori strettamente necessari alla Fishino da innestare, soluzione, questa, da noi scelta nel progetto, visto che è presumibile che lo shield, pur essendo "universale", venga destinato di volta in volta ad un solo controller; lo svantaggio è che lo shield, una volta montato per poter supportare UNO/MEGA/32 non è in grado di montare Guppy/Nano/Piranha, e viceversa, senza sostituire i connettori;
- montare tutti i connettori eliminando i soli pin che interferiscono, soluzione, questa, che sarebbe quasi ottimale, salvo che i segnali corrispondenti ai pin eliminati non verrebbero portati ai connettori aggiuntivi laterali (qualcuno potrà preferire questa soluzione, quindi la esporremo dettagliatamente in seguito);
- "girare" i connettori ESPCONN ed ICSP sulle schede UNO e 32 dal lato opposto della board, soluzione che risolve tutto, ma purtroppo richiede una modifica delle schede suddette, consistente nel dissaldare i connettori presenti e risaldarli sul lato opposto; anche questa soluzione è praticabile, ma richiede una buona manualità per non danneggiare le schede.

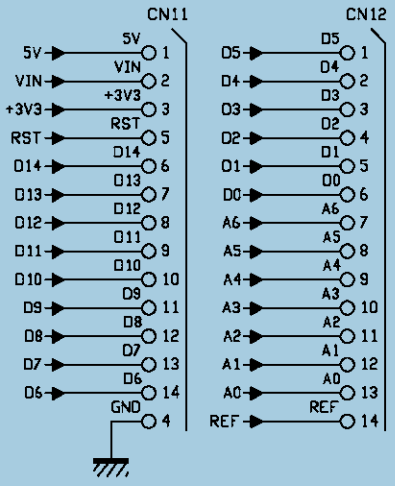
Come detto, noi abbiamo optato per la prima soluzione, soprattutto tenendo conto che lo shield verrà proposto in kit, quindi con i connettori da montare; nulla vieta comunque di sceglierne un'altra.

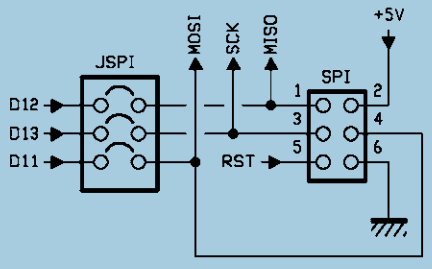
SCHEMA ELETTRICO

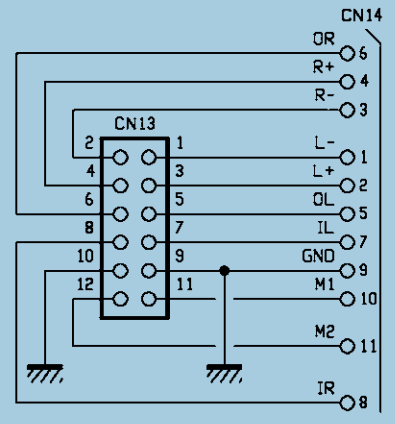
Dopo aver studiato il layout della scheda, eccoci pronti per lo schema elettrico! Il tutto è piuttosto


5V
IN
U1
OUT
GND
INT
EXT
JPWR
+3V3
U1T
C1
CN1
CN5
CN2
CN6
CN3
CN7
CN4
CN8
D14
TFT
TFT_VCC
TFT_LED
R1
Q1
OTHERS
JTLED
PIRANHA
D8
D3
D5
D2
D7
JTMODE
JTDC
JTMOSI
JTMISO
JTSCK
JBUZ
SJ_T_IRQ
SJ_DC
SJ_T_CS
SJ_CS
WAKE
R8
R3
R5
R2
R6
R7
R4
D7
D2
D4
D1
D5
D6
D3
TFT_T_IRQ
TFT_DC
TFT_T_CS
TFT_CS
TFT_MOSI
TFT_MISO
TFT_SCK
GND
BUZ
RESET
RST

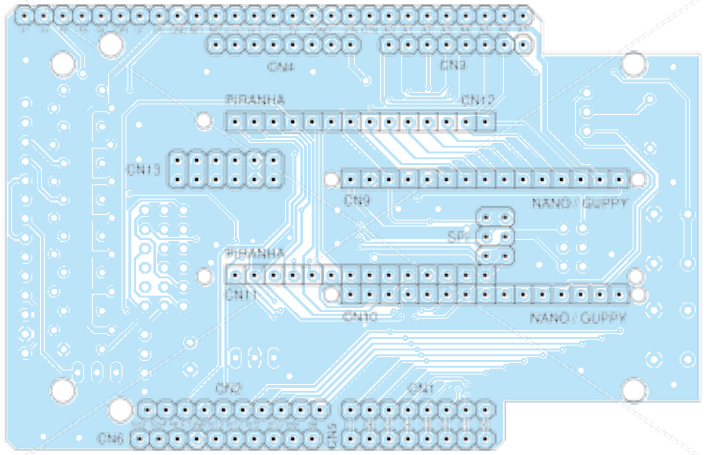


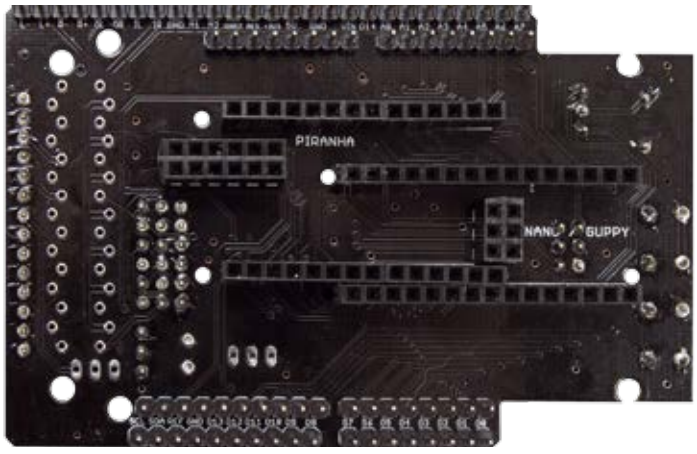






semplice, si tratta soltanto di collegare i vari segnali e prevedere i necessari ponticelli per poterli adattare alle varie schede. In particolare, l'esigenza si fa sentire per le schede Piranha/MKR1000, che hanno un pin-out completamente diverso rispetto alle altre, ed anche i segnali di interrupt/SPI relativi.
La prima cosa strana che si può notare nello schema elettrico è il raddoppio di tutti i componenti attivi e passivi. Questo è stato fatto per poter realizzare una scheda in grado di montare sia componenti in formato SMD, cosa vantaggiosa per una produzione industriale, sia in formato THT (con i reofori passanti), in modo da poter offrire la scheda in forma di kit assemblabile anche senza esperienza nei montaggi superficiali. Quindi, nello schema tutti i componenti "sdoppiati" sono da intendersi come alternativi tra il formato SMD e THT.
Iniziamo col descrivere l'alimentazione, che potrebbe sembrare superflua (tutte le schede utilizzate hanno un'uscita di alimentazione a 3,3 volt), ma che, vista la scarsa disponibilità di corrente su alcune board, in particolar modo le Arduino originali e molti cloni economici, risulta indispensabile.
Si tratta di un semplicissimo regolatore lineare low drop-out da 3,3 V, in grado di erogare poco meno di un ampere in uscita, corrente fin troppo abbondante per lo scopo.
Il ponticello denominato PWR permette di selezionare la tensione proveniente dal regolatore interno o direttamente dalla linea a 3,3 volt della board connessa. Può sembrare superfluo ma, avendo una scheda con sufficiente disponibilità di corrente sui 3,3 volt, come la serie Fishino, ci permette di risparmiare il regolatore U1 ed i condensatori a corredo volendo limitare i costi al massimo.
Analizziamo ora il connettore del display TFT e la componentistica annessa: il visualizzatore utilizzato lavora con una tensione di 3,3 V e richiede livelli logici corrispondenti; fornendo livelli a 5 volt si può facilmente danneggiare. Per ovviare al problema abbiamo inserito dei limitatori di livello costituiti ciascuno da una resistenza (R1÷R8) ed un diodo connesso verso il positivo dell'alimentazione a 3,3 volt. Questo permette, in presenza di un segnale di valore superiore, di "scaricarlo" sui 3,3 volt, limitandolo automaticamente a quel valore.
Le resistenze scelte hanno due esigenze contrapposte: devono essere sufficientemente alte di valore per non assorbire una corrente eccessiva e nel contempo sufficientemente basse da non causare un ritardo nei segnali veloci. Come potete notare, nelle linee di abilitazione e di controllo (IRQ, DC ed i due CS), dove viaggiano segnali "lenti", le resi-





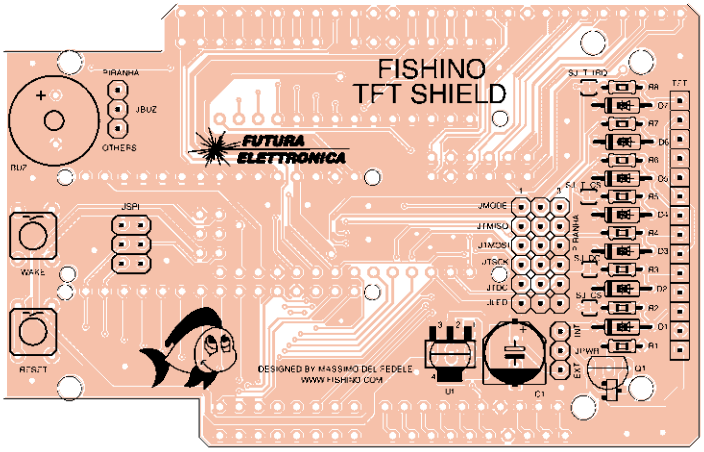
Elenco Componenti:

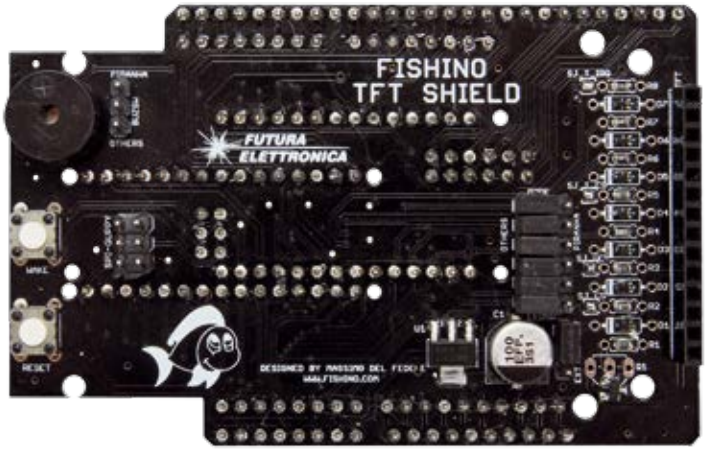
R1: 10 kohm
R2: 1 kohm
R3: 1 kohm
R4: 330 ohm
R5: 1 kohm
R6: 330 ohm
R7: 330 ohm
R8: 1 kohm
C1: 100 μF 16 VL elettrolitico
Q1: BS170
U1:NCP1117ST33
D1: 1N4148
D2: 1N4148
D3: 1N4148
D4: 1N4148
D5: 1N4148
D6: 1N4148
D7: 1N4148
RESET: Microswitch
WAKE: Microswitch
BUZ: Buzzer senza elettronica
CN1: Strip Maschio 8 vie
CN2: Strip Maschio 10 vie
CN3: Strip Maschio 8 vie
CN4: Strip Maschio 8 vie
CN5: Strip Maschio 8 vie

stenze hanno un valore di 1 kohm, dando quindi la precedenza alla bassa corrente rispetto alla velocità di risposta; per contro, nelle tre linee dove viaggiano i segnali SPI, che possono assumere valori di frequenza sopra ai 10 MHz, abbiamo optato per un valore decisamente più basso, ovvero 330 ohm. Anche con questo valore, superare i 12÷16 MHz diventa difficile, infatti in libreria abbiamo limitato la frequenza dei segnali SPI a 12 MHz: un valore più che sufficiente per lavorare a una velocità discreta.
I connettori MODE (montati vicino al condensatore C1) servono per selezionare l'utilizzo con schede "standard" (UNO, MEGA, 32 e simili) oppure di tipo Piranha/MKR1000 che hanno segnali differenti. Anche i connettori JTMOSI, JTMISO, JTSCK vengono utilizzati per la selezione del controller, e più precisamente per gestire la differente posizione dei segnali SPI sulle schede.
Come possibilità aggiuntiva abbiamo inserito inoltre dei ponticelli sul PCB per poter sconnettere le 4 linee di controllo del display: SJ_T_IRQ, SJ_T_CS, SJ_DC e SJ_CS. A cosa servono? Semplicemente, nel tempo ci siamo accorti che spesso gli shield, avendo collegamenti prestabiliti ai pin digitali, sono molto vincolanti. Se si ha la necessità di connettere qualcos'altro che utilizza gli stessi I/O (un altro shield in cascata, per esempio) le cose si complicano e si ha la scelta tra il non usare gli shield insieme oppure il tagliare alcune piste del PCB. Tagliando questi ponticelli, invece, è possibile liberare gli I/O corrispondenti senza dover manomettere il circuito stampato; ovviamente è poi necessario realizzare dei collegamenti volanti con altri I/O per utilizzare il display, ma questo è molto semplice.
Il MOSFET Q1, insieme alla resistenza R1 ed al connettore JTLED viene utilizzato, opzionalmente, per il controllo della retroilluminazione del display. Lasciandolo aperto, il MOSFET risulta polarizzato da R1 e quindi l'illuminazione è accesa; inserendo un jumper è possibile controllarla tramite un I/O

CN6: Strip Maschio 10 vie
CN7: Strip Maschio 8 vie
CN8: Strip Maschio 8 vie
CN9: Strip Femmina 15 vie
CN10: Strip Femmina 15 vie
CN11: Strip Femmina 15 vie
CN12: Strip Femmina 15 vie
CN13: Strip Femmina 2x6 vie
CN14: Strip Maschio 10 vie
TFT: Strip Femmina 14 vie
JTLED: Strip Maschio 3 vie
JTMODE: Strip Maschio 3 vie
JTDC: Strip Maschio 3 vie
JTMOSI: Strip Maschio 3 vie
JTMISO: Strip Maschio 3 vie
JTSCK: Strip Maschio 3 vie
JBUZ: Strip Maschio 3 vie
JPWR: Strip Maschio 3 vie
JSPI: Strip Maschio 2x3 vie
SPI: Strip Femmina 2x3 vie

Varie:
- Jumper (8 pz.)
- Display Touch 2,4" o 2,8"
- Circuito stampato S1328 (88x56mm)





digitale del controller, e più precisamente il D8 sulle schede UNO/MEGA/32 e il D3 sulla Piranha/MKR1000.
Tramite questa caratteristica è quindi possibile risparmiare corrente di alimentazione quando non serve che il display sia visibile.
Successivamente possiamo notare i due pulsanti, RESET e WAKE, che vengono utilizzati rispettivamente per resettare il controller (una comodità rispetto al dover "cercare" il pulsante sulla scheda che, specialmente sul MEGA, risulta difficilmente raggiungibile) e per "risvegliarlo", oppure per altri usi a scelta. Il pulsante WAKE è infatti connesso alla stessa linea di interrupt del touch controller e può quindi venire usato per uscire dalla modalità di stand-by anche col display completamente spento.
Abbiamo poi voluto inserire anche un cicalino, per poter avere un feedback acustico, sia del tocco sul display che di eventuali condizioni di errore. Il cicalino è connesso (opzionalmente) ad un altro I/O digitale, e più precisamente al D9 sulle schede UNO/MEGA/32 o al D4 su Piranha/MKR1000, selezionabili anche qui tramite un connettore, il JBUZ. Lasciando il connettore aperto risparmiamo una linea di I/O ma non potremo sfruttare il cicalino. Quest'ultimo è di tipo passivo, va quindi pilotato con un segnale PWM di frequenza opportuna, gestito via software.
Vediamo, infine, il percorso dei segnali SPI e l'ultimo connettore di selezione schede, lo JSPI. Come detto in precedenza, i segnali SPI viaggiano su diversi I/O a seconda della scheda. Sulla UNO e sulla Mega (e la Fishino 32) la cosa è semplificata dal connettore ISP, che li riporta indipendentemente da dove vengono connessi ; questo viene sfruttato tramite il connettore SPI che raccoglie tali segnali. Sulle schede Piranha/MKR1000 abbiamo già visto i relativi jumper. Restano fuori Guppy ed Arduino Nano che, pur avendo il connettore ISP, l'hanno montato dal lato "sbagliato" della scheda, e quindi

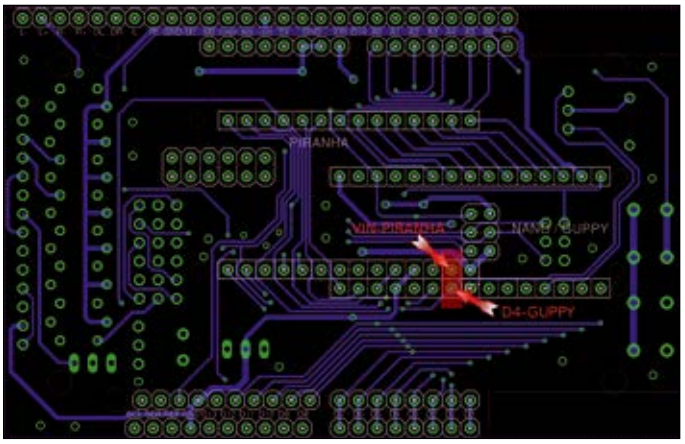


***Fig. 5** - I punti di interferenza su Fishino32.*

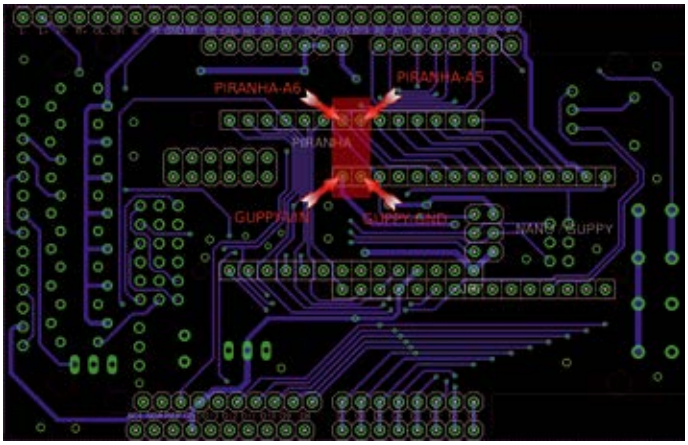


***Fig. 6** - Punti di interferenza su Fishino UNO.*

non è possibile sfruttarlo nel nostro shield. Montando Guppy o Nano è quindi necessario inserire i tre jumper nel connettore JSPI che realizzano il collegamento richiesto alle linee SPI.

REALIZZAZIONE PRATICA

La costruzione dello shield è alla portata anche di chi ha poca esperienza, grazie alla possibilità di montare componenti passanti tradizionali. Nulla vieta, nel caso preferiate, di utilizzare i componenti SMD.
Consigliamo come sempre di montare per primi i componenti a basso profilo e poco ingombranti, quindi resistenze e diodi, seguiti dai condensatori elettrolitici, il regolatore di tensione ed il MOSFET. Successivamente vanno disposti i pulsanti ed il cicalino e, per ultimi, essendo piuttosto ingombranti, i connettori.
Qui di seguito potete vedere due immagini dello shield, visto sia dal lato superiore (componenti passivi/attivi, jumper e connettore per il display TFT) sia dal lato inferiore (connettori per le board).
L'unica cosa a cui occorre fare particolare attenzione è il lato di montaggio dei vari header; montandoli dal lato sbagliato ovviamente non riuscirete ad infilare i controller, i jumper o il display!
Come anticipato in precedenza, a causa di alcune interferenze tra i vari connettori delle schede utilizzabili con il nostro shield, non è possibile montare tutti i connettori nello stesso tempo, a meno di non utilizzare particolari accorgimenti. Se ricordate, abbiamo prospettato 3 possibilità, la terza delle quali comporta una modifica alla scheda controller originale che però lasciamo alla scelta di chi è dotato di buona manualità.
Le altre due alternative sono quella di montare solo i connettori necessari alla propria scheda, cosa peraltro che non ha bisogno di ulteriori spiegazioni, oppure quella di eliminare da alcuni connettori i pin che interferiscono, rendendo lo shield sempre utilizzabile con tutte le schede, al prezzo di non portare all'esterno alcuni del segnali disponibili. Vediamo qui di seguito le interferenze e i pin da eliminare.

UTILIZZO CON FISHINO32

La board Fishino32 interferisce con i connettori delle schede Guppy/Piranha solo su due punti, visibili nella **Fig. 5**.
Evitando di montare il connettore nei due punti segnalati si risolvono completamente le interferenze, al costo di non portare all'esterno i segnali VIN della scheda PIRANHA ed il D4 della scheda Guppy. Il VIN è necessario solo se si alimenta la scheda con una tensione esterna, mentre il D4 del Guppy, essendo utilizzato anche dalla scheda SD interna, è probabilmente inutile per un utilizzo esterno.
Eliminando questi due "spezzoni" di connettore è quindi possibile montare senza ulteriori modifiche le seguenti schede: Arduino UNO, MEGA e Nano, Fishino MEGA, Fishino32, Fishino Guppy, Fishino Piranha ed Arduino MKR1000. La scheda Fishino UNO risente di altre interferenze, che descriveremo nei prossimi paragrafi.

UTILIZZO CON FISHINO UNO

La scheda FishinoUNO interferisce con i connettori delle board Guppy/Piranha in quattro punti, evidenziati nella **Fig. 6**.
Anche in questo caso è possibile rendere il nostro shield TFT "universale" eliminando i pin dei connettori indicati dalle frecce nella predetta figura. Tuttavia in questo caso occorre sacrificare le connessioni esterne sui due canali analogici A5 ed A6 del PIRANHA e la VIN della Guppy. La linea GND è presente anche su un altro pin quindi non dà problemi.
Eliminando questi quattro pin è possibile montare

le seguenti schede: Arduino UNO/MEGA/NANO, Fishino UNO, Fishino MEGA, Fishino GUPPY, Fishino PIRANHA, ma non la Fishino32 che richiede l'eliminazione dei pin al paragrafo precedente. Prendendo entrambi gli accorgimenti è ovviamente possibile montare tutte le schede disponibili.

ED ORA... SOFTWARE!

Come accennato nell'introduzione, per la gestione dello shield abbiamo approntato tre librerie che ne consentono il controllo completo. Queste sono:

- **FishinoGFX**, versione praticamente identica all'analoga di Adafruit, che gestisce le funzioni grafiche "ad alto livello";
- **FishinoILI9341**, che gestisce le funzioni di interfaccia con il display a livello hardware; anche questa libreria è stata realizzata partendo dall'analoga di Adafruit, ma con modifiche abbastanza sostanziali;
- **FishinoXPT2046**; che gestisce il touch screen, scritta da zero di nostro pugno.

Iniziamo dalla sezione display, con uno sketch semplicissimo che inizializza lo schermo e disegna una croce in due colori; lo trovate nel **Listato 1**.
Come potete notare lo sketch è semplicissimo. La funzione **loop()** risulta vuota (non c'è nulla da ripetere!), mentre tutto si svolge nella **setup()**.
La prima linea di questa:

```
tft.begin();
```

inizializza lo schermo. Utilizzando lo shield con le schede previste non occorre specificare i pin di connessione; la libreria si occupa di tutto. Volendo utilizzare connessioni differenti, occorre specificare quali I/O utilizzare, nella funzione **begin()**:

```
tft.begin(cs_pin, dc_pin);
```

Dove in cs_pin e dc_pin vanno indicati i pin cui sono connesse le linee CS e DC del display.
Andiamo alla linea:

```
tft.fillScreen(ILI9341_BLACK);
```

la quale riempie semplicemente lo schermo di nero, cancellandolo. Successivamente troviamo le due righe di codice:

```
tft.drawLine(0, 0, tft.width(), tft.height(), ILI9341_RED);
tft.drawLine(0, tft.height(), tft.width(), 0, ILI9341_GREEN);
```

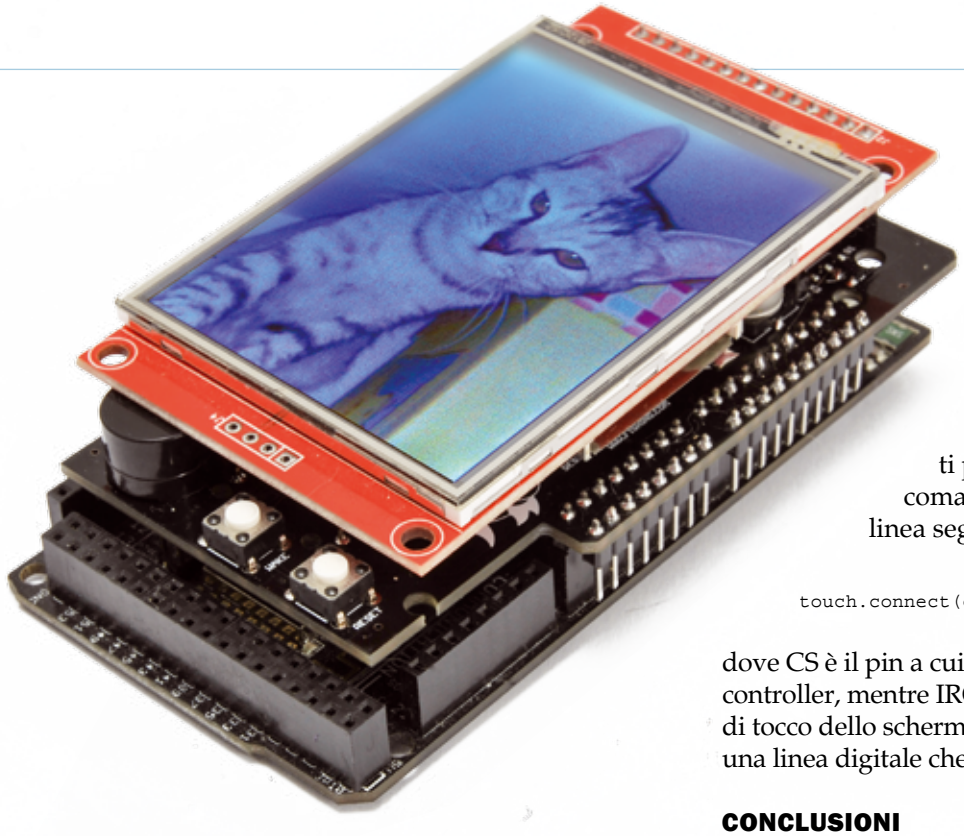
che si occupano di disegnare due linee di colori differenti (rosso e verde) che attraversano lo schermo. Le funzioni **tft.width()** e **tft.height()** forniscono rispettivamente la larghezza e l'altezza del display in pixel.
La libreria è molto estesa e tra le funzioni che implementa permette di tracciare punti, linee, cerchi, rettangoli, immagini, testi, eccetera. In essa è anche possibile "ruotare" lo schermo, in

Listato 1

```
#include <SPI.h>
#include <FishinoGFX.h>
#include <FishinoILI9341.h>

// questa linea è per comodità, in modo da non dover scrivere
// FishinoILI9341 ad ogni comando inviato al display
#define tft                FishinoILI9341

void setup()
{
    // inizializza lo schermo
    tft.begin();

    // cancella lo sfondo riempiendolo di nero
    tft.fillScreen(ILI9341_BLACK);

    // disegna una croce che percorre tutto lo schermo
    // in due colori
    tft.drawLine(0, 0, tft.width(), tft.height(), ILI9341_RED);
    tft.drawLine(0, tft.height(), tft.width(), 0, ILI9341_GREEN);

}

void loop()
{
}
```

Listato 2

```
#include <Flash.h>
#include <FishinoXPT2046.h>
#include <SPI.h>

// questa linea è per comodità, in modo da non dover scrivere
// FishinoXPT2046 ad ogni comando
#define touch              FishinoXPT2046

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

void loop()
{
        if(touch.touching())
        {
                uint16_t x, y, z;
                touch.read(x, y, z);
                Serial << "X : " << x << "\n";
                Serial << "Y : " << y << "\n";
                Serial << "Z : " << z << "\n";
                Serial << "------------------\n";
        }
        delay(200);
}
```



Shield e display montati sulla Fishino MEGA.

occorre specificare le connessioni hardware SE si utilizza lo shield con le schede previste; altrimenti prima di iniziare ad usare i comandi occorre indicarle tramite la linea seguente nella **setup()**:

```
touch.connect(cs, irq);
```

dove CS è il pin a cui è collegata la linea CS del touch controller, mentre IRQ è la linea relativa al segnale di tocco dello schermo, che deve essere connessa ad una linea digitale che supporta gli interrupt.

CONCLUSIONI

In questo articolo abbiamo colmato quella che ritenevamo una lacuna, ossia la mancanza di una periferica display per le nostre Fishino; ecco quindi uno shield con LCD grafico e dotato addirittura di interfaccia tattile, abbinabile a tutte le board Fishino sinora proposte e alla nascente Piranha.
Con i brevissimi esempi di codice appena proposti e descritti si conclude la presentazione del nostro shield display per le board Fishino.
Prossimamente pubblicheremo un esempio di utilizzo più complesso, sfruttando le caratteristiche di Fishino32. ■

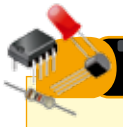
modo da disegnarci sopra in modalità orizzontale o addirittura di capovolgere l'immagine visualizzata.
Nella libreria sono contenuti alcuni esempi, tra cui due "painter" che permettono di disegnare sullo schermo sfruttando il touch-screen incorporato, oltre a un test grafico che mostra le varie primitive disponibili.
Vediamo ora la sezione touch-screen, della quale si occupa la libreria **FishinoXPT2046**.
Anche qui mostriamo un esempio semplice-semplice, rimandando quelli allegati alle librerie per le caratteristiche più complesse.
Lo sketch mostrato nel **Listato 2** attende semplicemente che si tocchi lo schermo e stampa sul monitor seriale dell'IDE Arduino la posizione in cui avviene il tocco.
Come potete notare, anche qui la semplicità è disarmante; la **setup()** si limita ad avviare la porta seriale, mentre nella loop si attende il tocco dello schermo tramite l'istruzione:

```
if(touch.touching())
```

e, quando questo avviene, si leggono le coordinate ed il valore di pressione tramite le istruzioni:

```
uint16_t x, y, z;
touch.read(x, y, z);
```

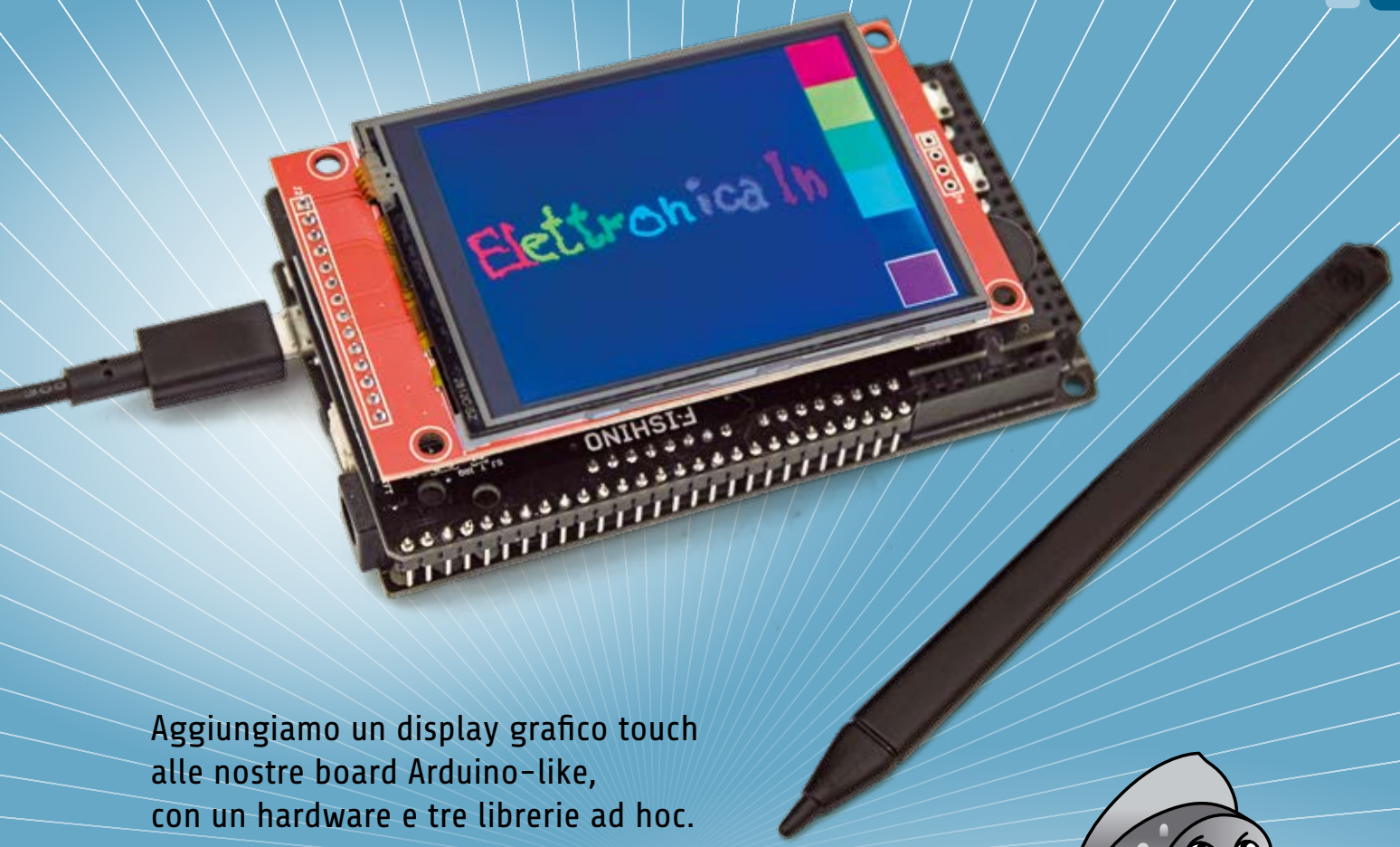
che poi vengono visualizzati sul monitor seriale.
Anche qui, come per la libreria del display, non



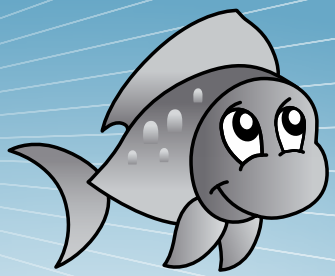
per il MATERIALE

Lo shield (cod. FT1328K) viene venduto in Kit al prezzo di Euro 19,00. Lo shield non comprende il display che però è possibile acquistare separatamente scegliendo fra il display LCD touch da 2,4" SPI (cod. LCDTOUCH24SER) che costa Euro 16,90, o quello da 2,8" SPI (cod. LCDTOUCH28SER) che è disponibile a Euro 19,50. Tutti questi prodotti si possono acquistare presso Futura Elettronica. 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



Aggiungiamo un display grafico touch alle nostre board Arduino-like, con un hardware e tre librerie ad hoc.



TFT SHIELD PER FISHINO

di MASSIMO DEL FEDELE

Dopo avervi presentato ben quattro schede di prototipazione della serie Fishino (la UNO, la Fishino 32, la MEGA, la Guppy...ed una quinta è in arrivo!), abbiamo iniziato a sentire il bisogno di a proporre un po' di elettronica di contorno. In particolare, abbiamo sentito la mancanza di un display da abbinare loro, per corredare le applicazioni con output visivi che non fossero semplicemente LED accesi o spenti. Ma non ci bastava un semplice display per testo (come ad esempio i comunissimi alfanumerici a 2 o 4 righe basato su controller HD44780), perché volevamo un bello schermo grafico caratterizzato da dimensioni ridotte e una discreta risoluzione, sul quale mostrare icone, simboli e fare un po' di grafica. Ci siamo quindi messi alla ricerca di qualcosa di esistente, riscontrando, tuttavia, che i pur molti display (e shield che li supportano) hanno parecchie limitazioni.
Per questo abbiamo voluto realizzarci qualcosa su misura, ossia uno shield con a bordo un prestante display, che vi descriviamo in questo articolo.
Partiamo dall'inizio, ossia dalla fase di progetto,

Fig. 1
L'assemblaggio del display sullo shield.

dicendo che volevamo qualcosa in grado di soddisfare i seguenti requisiti:

- dimensioni sufficientemente ridotte (2,4 o al massimo 3,2 pollici);
- risoluzione minima di 320x240 pixel, con profondità di colore elevata;
- interfaccia touch che ci permetta di interagire con il nostro Fishino;
- impegno di poche risorse hardware della scheda, in modo da poterlo utilizzare anche con i modelli di Fishino dotati di pochi I/O (la Guppy, per esempio);
- uno shield che si adattasse a tutti i prodotti della serie Fishino ed anche a molti della serie Arduino, senza bisogno di connessioni volanti
- la possibilità di collegare altri componenti insieme allo shield, quindi l'esposizione di tutti gli I/O delle varie schede anche con il display montato.

Crediamo di essere riusciti nei nostri propositi quasi al 100% (vedremo in seguito alcuni piccoli "limiti", facilmente superabili), con un prodotto interessante e di utilizzo semplicissimo, che vi presenteremo in queste pagine.

Per renderne ancora più agevole l'utilizzo abbiamo provveduto a preparare tre apposite librerie, due delle quali di derivazione Adafruit, in grado di gestire il display stesso, ed una totalmente sviluppata da noi per la gestione del touch-panel che esso incorpora.

UNO SHIELD "UNIVERSALE"

Contrariamente al solito, inizieremo la descrizione dello shield...dalla fine! Infatti, la cosa più complessa nello sviluppo del progetto è stata proprio la creazione del layout della scheda, dovendo abbinare ad essa board anche parecchio diverse le une dalle altre. Abbiamo quindi iniziato con lo sviluppo "grafico" della scheda, per poi andare a ritroso fino ad ottenere lo schema elettrico.

Iniziamo dall'ingombro: il primo requisito era quello di ottenere una scheda con dimensioni paragonabili alla nostra Fishino UNO (ma anche alla Fishino32 e ad Arduino UNO); quindi il display va sovrapposto senza o quasi aumentarne l'ingombro laterale: si sviluppa tutto nello spessore. Abbiamo quindi optato per un "sandwich" consistente, nell'ordine, dal display TFT, dallo shield e, infine, dalla scheda a microcontroller; la **Fig. 1** chiarisce meglio il concetto.

In questa immagine potete vedere il "mockup" (modello grafico) dello shield con display ed un Fishino MEGA montati, che è stata la nostra base di partenza.

L'esigenza di accedere a tutti gli I/O della scheda ha imposto l'aggiunta, lateralmente, di due file di connettori maschio paralleli a quelli destinati ad infilarsi nei connettori femmina della scheda, ai quali è possibile connettere i consueti cavetti Dupont.

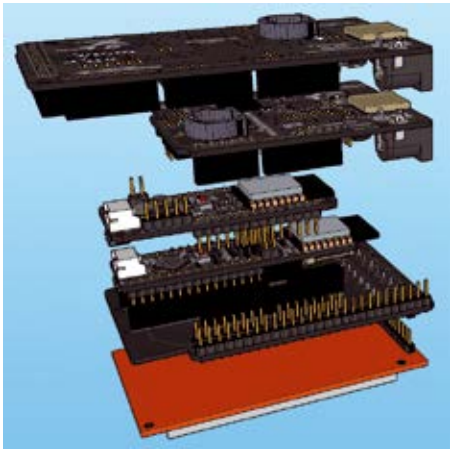


Fig. 2
Esploso dello shield e delle board Fishino.

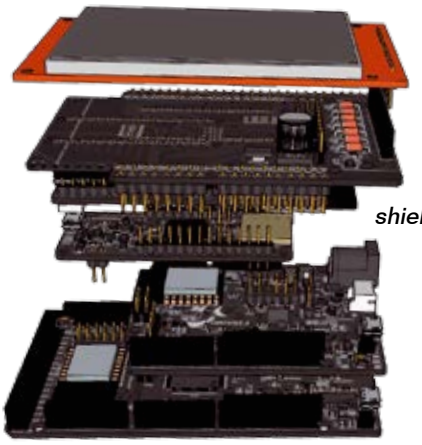


Fig. 3
Esploso dello shield e delle board Fishino, visto da sopra.

Una volta stabilito a grandi linee il layout della scheda, la dimensione del display è stata fissata in 2,8 pollici come massimo (può accettare anche il display più piccolo da 2,4 pollici), in modo da non superare l'ingombro previsto.
Un altro parametro che ha vincolato la dimensione è stato l'esigenza di utilizzare il minor numero di I/O possibili; abbiamo quindi scartato tutti i display con interfaccia parallela (8 ma anche 16 bit) che avrebbero occupato praticamente tutte le risorse delle schede più "piccole" e ci siamo orientati verso un display con comunicazione SPI, reperibile facilmente nel formato, appunto, di 2,4 pollici o 2,8 pollici. Questo display utilizza le linee SPI (MISO, MOSI e SCK), in comune con il modulo WiFi e le schede SD (ed altro eventuale!), una linea di selezione ed una di controllo per il display ed altre 2 linee per lo schermo touch-sensitive, quindi quattro I/O utilizzati in "esclusiva" contro 10÷12 minimo per i modelli ad interfaccia parallela.
Come vedremo in seguito, abbiamo poi aggiunto altre funzionalità opzionali, ottenibili con l'utilizzo di pochi altri I/O.
Fissati quindi layout e display, occorreva trovare un modo per poter "infilare" tutti i modelli di board previsti sullo stesso shield. Visto che la tecnologia attuale lo permette, abbiamo realizzato i modelli 3D di tutte le schede, il modello 3D del display ed il modello 3D iniziale dello shield, in modo da poterli sovrapporre graficamente e visualizzare eventuali interferenze; nella **Fig. 2** e nella **Fig. 3** trovate i disegni in esploso visti dal basso e dall'alto del nostro shield con sovrapposte un buon numero di schede Fishino. Resta inteso che di Fishino se ne monta una sola per volta.
Abbiamo quindi cercato la posizione migliore per le schede "piccole" (Fishino Guppy/Arduino Nano e la futura Fishino Piranha, pin-out compatibile con l'Arduino MKR1000) in modo da non avere interferenze tra i connettori (cosa impossibile al 100%, come vedremo a breve) e da poter restare negli ingombri previsti.
Ecco quindi nascere il layout definitivo della scheda, mostrato nella **Fig. 4** dal lato di inserimento dei controller (vale a dire delle schede Fishino/Arduino).
Tutto perfetto, quindi? Purtroppo no, perché le schede Fishino UNO e 32 hanno due connettori ausiliari, per la precisione il connettore ESPCONN sulla UNO ed il connettore ICSP sulla 32 che vanno ad interferire con i connettori delle schede "piccole" (Nano/Guppy e Piranha/MKR1000); l'inserimento è fisicamente possibile, ma i segnali non sono compatibili con i circuiti della scheda.

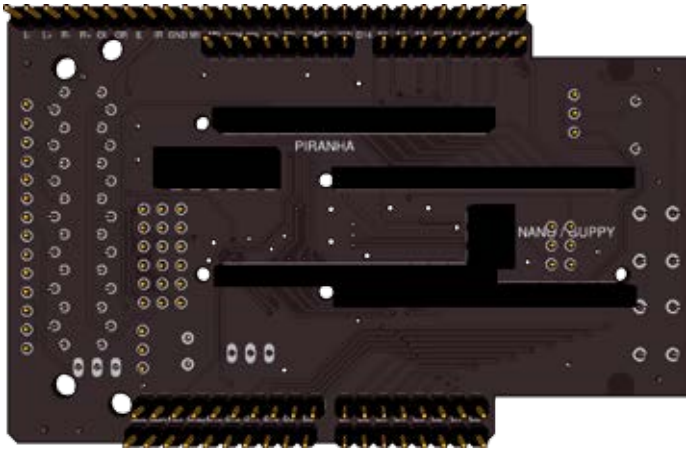


***Fig. 4** - Layout dei connettori dello shield.*

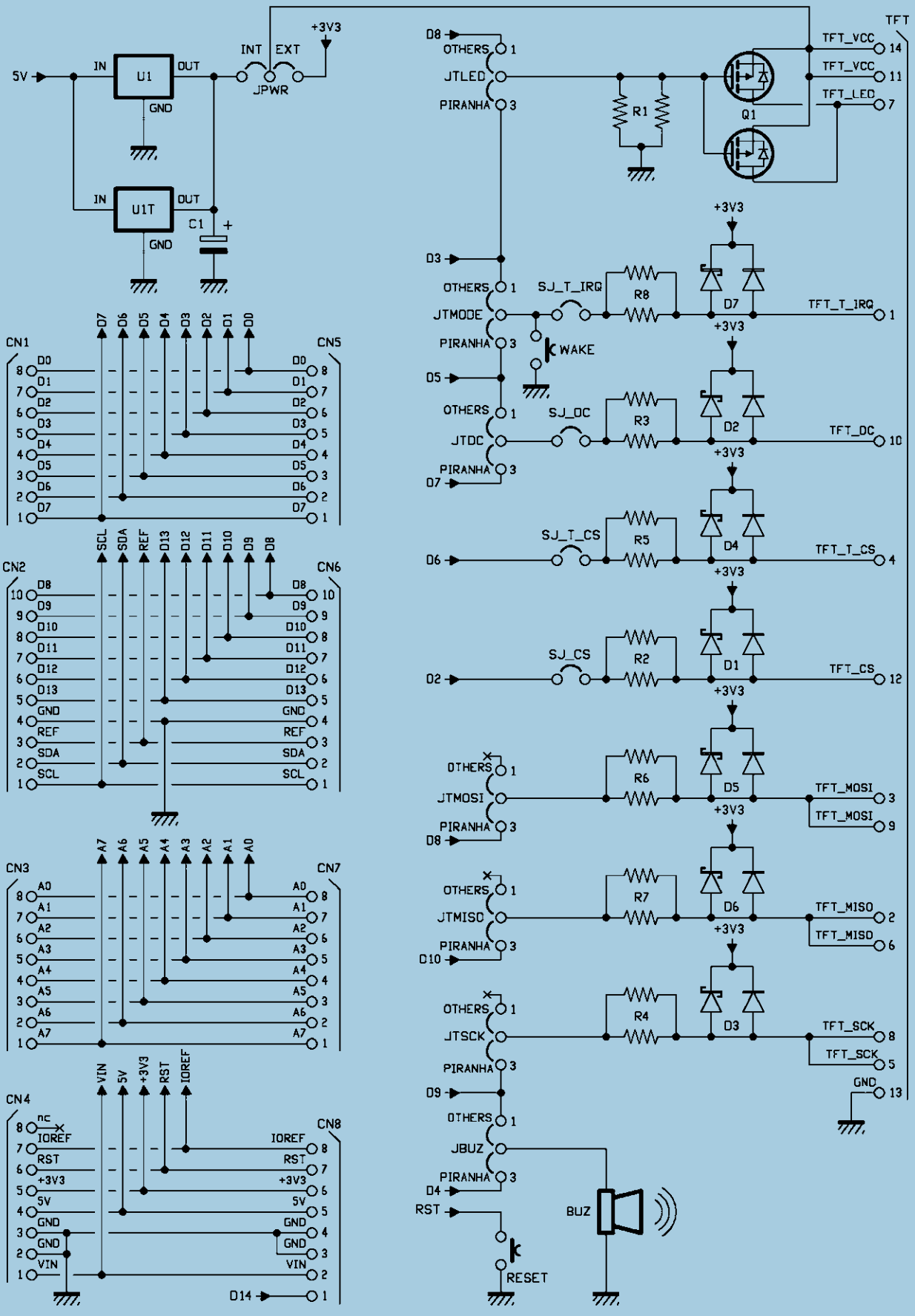
Il "problema" non è risolvibile, visto che alcuni connettori devono avere una posizione ben precisa gli uni con gli altri: per esempio il connettore ISP che porta i segnali SPI sia dell'UNO che del Mega.
Per ovviare all'inconveniente ci sono più possibilità:

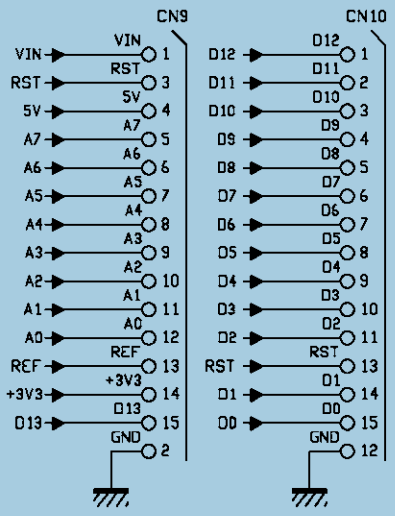
- montare solo i connettori strettamente necessari alla Fishino da innestare, soluzione, questa, da noi scelta nel progetto, visto che è presumibile che lo shield, pur essendo "universale", venga destinato di volta in volta ad un solo controller; lo svantaggio è che lo shield, una volta montato per poter supportare UNO/MEGA/32 non è in grado di montare Guppy/Nano/Piranha, e viceversa, senza sostituire i connettori;
- montare tutti i connettori eliminando i soli pin che interferiscono, soluzione, questa, che sarebbe quasi ottimale, salvo che i segnali corrispondenti ai pin eliminati non verrebbero portati ai connettori aggiuntivi laterali (qualcuno potrà preferire questa soluzione, quindi la esporremo dettagliatamente in seguito);
- "girare" i connettori ESPCONN ed ICSP sulle schede UNO e 32 dal lato opposto della board, soluzione che risolve tutto, ma purtroppo richiede una modifica delle schede suddette, consistente nel dissaldare i connettori presenti e risaldarli sul lato opposto; anche questa soluzione è praticabile, ma richiede una buona manualità per non danneggiare le schede.

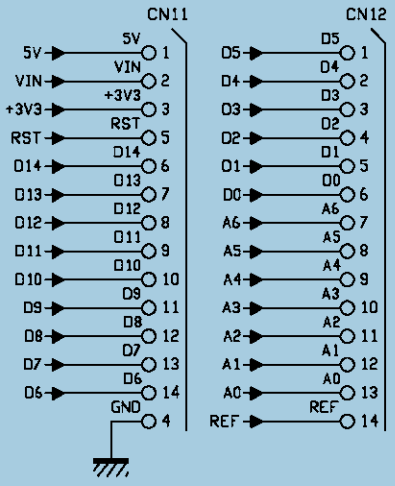
Come detto, noi abbiamo optato per la prima soluzione, soprattutto tenendo conto che lo shield verrà proposto in kit, quindi con i connettori da montare; nulla vieta comunque di sceglierne un'altra.

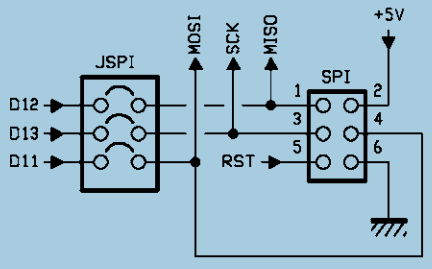
SCHEMA ELETTRICO

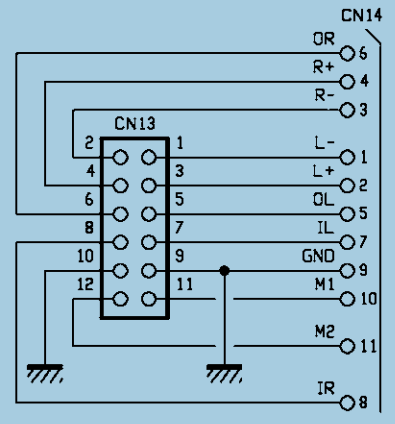
Dopo aver studiato il layout della scheda, eccoci pronti per lo schema elettrico! Il tutto è piuttosto


5V
IN
U1
OUT
GND
INT
EXT
JPWR
+3V3
U1T
C1
CN1
CN5
CN2
CN6
CN3
CN7
CN4
CN8
D14
TFT
TFT_VCC
TFT_LED
R1
Q1
OTHERS
JTLED
PIRANHA
D8
D3
D5
D2
D7
JTMODE
JTDC
JTMOSI
JTMISO
JTSCK
JBUZ
SJ_T_IRQ
SJ_DC
SJ_T_CS
SJ_CS
WAKE
R8
R3
R5
R2
R6
R7
R4
D7
D2
D4
D1
D5
D6
D3
TFT_T_IRQ
TFT_DC
TFT_T_CS
TFT_CS
TFT_MOSI
TFT_MISO
TFT_SCK
GND
BUZ
RESET
RST

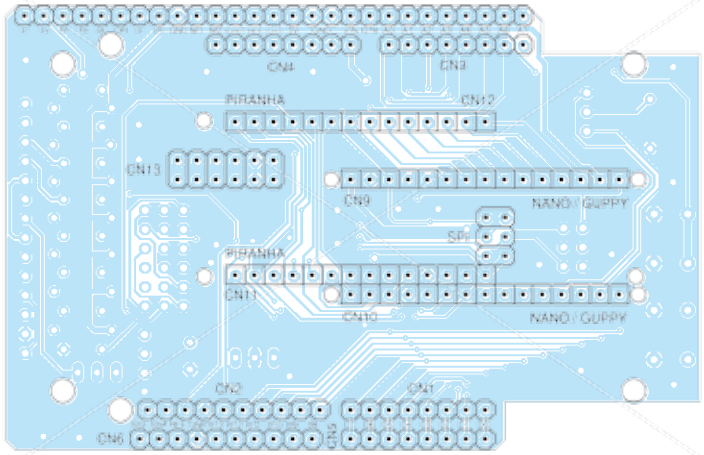


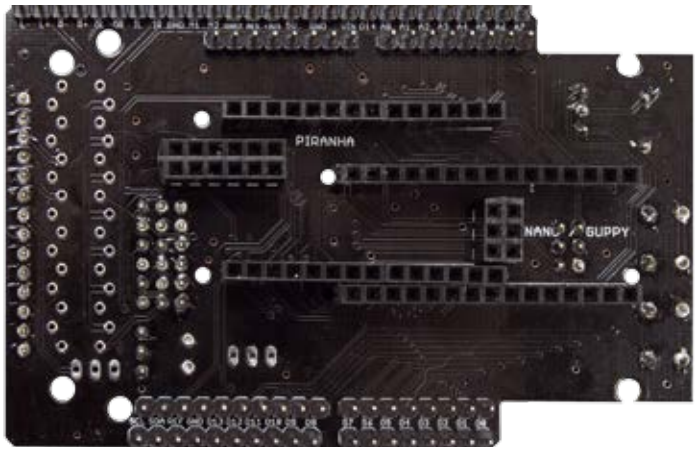






semplice, si tratta soltanto di collegare i vari segnali e prevedere i necessari ponticelli per poterli adattare alle varie schede. In particolare, l'esigenza si fa sentire per le schede Piranha/MKR1000, che hanno un pin-out completamente diverso rispetto alle altre, ed anche i segnali di interrupt/SPI relativi.
La prima cosa strana che si può notare nello schema elettrico è il raddoppio di tutti i componenti attivi e passivi. Questo è stato fatto per poter realizzare una scheda in grado di montare sia componenti in formato SMD, cosa vantaggiosa per una produzione industriale, sia in formato THT (con i reofori passanti), in modo da poter offrire la scheda in forma di kit assemblabile anche senza esperienza nei montaggi superficiali. Quindi, nello schema tutti i componenti "sdoppiati" sono da intendersi come alternativi tra il formato SMD e THT.
Iniziamo col descrivere l'alimentazione, che potrebbe sembrare superflua (tutte le schede utilizzate hanno un'uscita di alimentazione a 3,3 volt), ma che, vista la scarsa disponibilità di corrente su alcune board, in particolar modo le Arduino originali e molti cloni economici, risulta indispensabile.
Si tratta di un semplicissimo regolatore lineare low drop-out da 3,3 V, in grado di erogare poco meno di un ampere in uscita, corrente fin troppo abbondante per lo scopo.
Il ponticello denominato PWR permette di selezionare la tensione proveniente dal regolatore interno o direttamente dalla linea a 3,3 volt della board connessa. Può sembrare superfluo ma, avendo una scheda con sufficiente disponibilità di corrente sui 3,3 volt, come la serie Fishino, ci permette di risparmiare il regolatore U1 ed i condensatori a corredo volendo limitare i costi al massimo.
Analizziamo ora il connettore del display TFT e la componentistica annessa: il visualizzatore utilizzato lavora con una tensione di 3,3 V e richiede livelli logici corrispondenti; fornendo livelli a 5 volt si può facilmente danneggiare. Per ovviare al problema abbiamo inserito dei limitatori di livello costituiti ciascuno da una resistenza (R1÷R8) ed un diodo connesso verso il positivo dell'alimentazione a 3,3 volt. Questo permette, in presenza di un segnale di valore superiore, di "scaricarlo" sui 3,3 volt, limitandolo automaticamente a quel valore.
Le resistenze scelte hanno due esigenze contrapposte: devono essere sufficientemente alte di valore per non assorbire una corrente eccessiva e nel contempo sufficientemente basse da non causare un ritardo nei segnali veloci. Come potete notare, nelle linee di abilitazione e di controllo (IRQ, DC ed i due CS), dove viaggiano segnali "lenti", le resi-





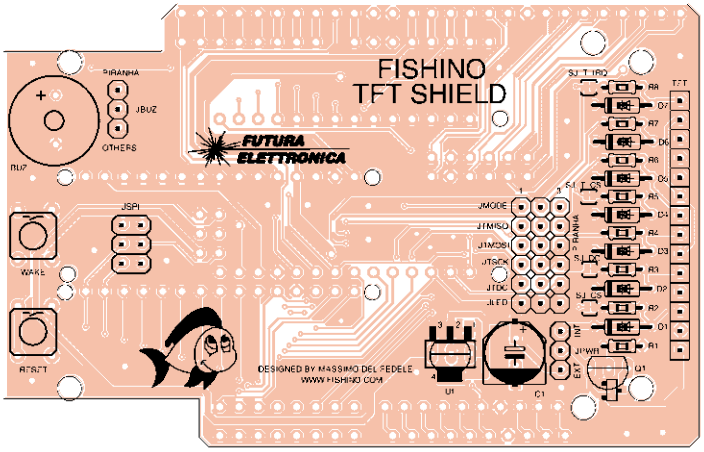
Elenco Componenti:

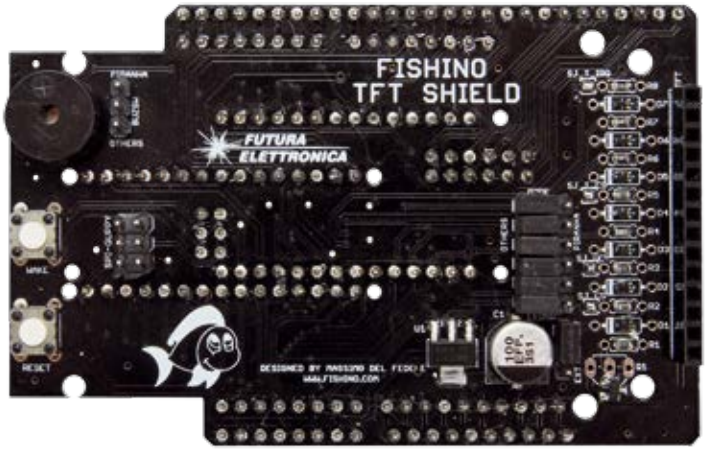
R1: 10 kohm
R2: 1 kohm
R3: 1 kohm
R4: 330 ohm
R5: 1 kohm
R6: 330 ohm
R7: 330 ohm
R8: 1 kohm
C1: 100 μF 16 VL elettrolitico
Q1: BS170
U1:NCP1117ST33
D1: 1N4148
D2: 1N4148
D3: 1N4148
D4: 1N4148
D5: 1N4148
D6: 1N4148
D7: 1N4148
RESET: Microswitch
WAKE: Microswitch
BUZ: Buzzer senza elettronica
CN1: Strip Maschio 8 vie
CN2: Strip Maschio 10 vie
CN3: Strip Maschio 8 vie
CN4: Strip Maschio 8 vie
CN5: Strip Maschio 8 vie

stenze hanno un valore di 1 kohm, dando quindi la precedenza alla bassa corrente rispetto alla velocità di risposta; per contro, nelle tre linee dove viaggiano i segnali SPI, che possono assumere valori di frequenza sopra ai 10 MHz, abbiamo optato per un valore decisamente più basso, ovvero 330 ohm. Anche con questo valore, superare i 12÷16 MHz diventa difficile, infatti in libreria abbiamo limitato la frequenza dei segnali SPI a 12 MHz: un valore più che sufficiente per lavorare a una velocità discreta.
I connettori MODE (montati vicino al condensatore C1) servono per selezionare l'utilizzo con schede "standard" (UNO, MEGA, 32 e simili) oppure di tipo Piranha/MKR1000 che hanno segnali differenti. Anche i connettori JTMOSI, JTMISO, JTSCK vengono utilizzati per la selezione del controller, e più precisamente per gestire la differente posizione dei segnali SPI sulle schede.
Come possibilità aggiuntiva abbiamo inserito inoltre dei ponticelli sul PCB per poter sconnettere le 4 linee di controllo del display: SJ_T_IRQ, SJ_T_CS, SJ_DC e SJ_CS. A cosa servono? Semplicemente, nel tempo ci siamo accorti che spesso gli shield, avendo collegamenti prestabiliti ai pin digitali, sono molto vincolanti. Se si ha la necessità di connettere qualcos'altro che utilizza gli stessi I/O (un altro shield in cascata, per esempio) le cose si complicano e si ha la scelta tra il non usare gli shield insieme oppure il tagliare alcune piste del PCB. Tagliando questi ponticelli, invece, è possibile liberare gli I/O corrispondenti senza dover manomettere il circuito stampato; ovviamente è poi necessario realizzare dei collegamenti volanti con altri I/O per utilizzare il display, ma questo è molto semplice.
Il MOSFET Q1, insieme alla resistenza R1 ed al connettore JTLED viene utilizzato, opzionalmente, per il controllo della retroilluminazione del display. Lasciandolo aperto, il MOSFET risulta polarizzato da R1 e quindi l'illuminazione è accesa; inserendo un jumper è possibile controllarla tramite un I/O

CN6: Strip Maschio 10 vie
CN7: Strip Maschio 8 vie
CN8: Strip Maschio 8 vie
CN9: Strip Femmina 15 vie
CN10: Strip Femmina 15 vie
CN11: Strip Femmina 15 vie
CN12: Strip Femmina 15 vie
CN13: Strip Femmina 2x6 vie
CN14: Strip Maschio 10 vie
TFT: Strip Femmina 14 vie
JTLED: Strip Maschio 3 vie
JTMODE: Strip Maschio 3 vie
JTDC: Strip Maschio 3 vie
JTMOSI: Strip Maschio 3 vie
JTMISO: Strip Maschio 3 vie
JTSCK: Strip Maschio 3 vie
JBUZ: Strip Maschio 3 vie
JPWR: Strip Maschio 3 vie
JSPI: Strip Maschio 2x3 vie
SPI: Strip Femmina 2x3 vie

Varie:
- Jumper (8 pz.)
- Display Touch 2,4" o 2,8"
- Circuito stampato S1328 (88x56mm)





digitale del controller, e più precisamente il D8 sulle schede UNO/MEGA/32 e il D3 sulla Piranha/MKR1000.
Tramite questa caratteristica è quindi possibile risparmiare corrente di alimentazione quando non serve che il display sia visibile.
Successivamente possiamo notare i due pulsanti, RESET e WAKE, che vengono utilizzati rispettivamente per resettare il controller (una comodità rispetto al dover "cercare" il pulsante sulla scheda che, specialmente sul MEGA, risulta difficilmente raggiungibile) e per "risvegliarlo", oppure per altri usi a scelta. Il pulsante WAKE è infatti connesso alla stessa linea di interrupt del touch controller e può quindi venire usato per uscire dalla modalità di stand-by anche col display completamente spento.
Abbiamo poi voluto inserire anche un cicalino, per poter avere un feedback acustico, sia del tocco sul display che di eventuali condizioni di errore. Il cicalino è connesso (opzionalmente) ad un altro I/O digitale, e più precisamente al D9 sulle schede UNO/MEGA/32 o al D4 su Piranha/MKR1000, selezionabili anche qui tramite un connettore, il JBUZ. Lasciando il connettore aperto risparmiamo una linea di I/O ma non potremo sfruttare il cicalino. Quest'ultimo è di tipo passivo, va quindi pilotato con un segnale PWM di frequenza opportuna, gestito via software.
Vediamo, infine, il percorso dei segnali SPI e l'ultimo connettore di selezione schede, lo JSPI. Come detto in precedenza, i segnali SPI viaggiano su diversi I/O a seconda della scheda. Sulla UNO e sulla Mega (e la Fishino 32) la cosa è semplificata dal connettore ISP, che li riporta indipendentemente da dove vengono connessi ; questo viene sfruttato tramite il connettore SPI che raccoglie tali segnali. Sulle schede Piranha/MKR1000 abbiamo già visto i relativi jumper. Restano fuori Guppy ed Arduino Nano che, pur avendo il connettore ISP, l'hanno montato dal lato "sbagliato" della scheda, e quindi

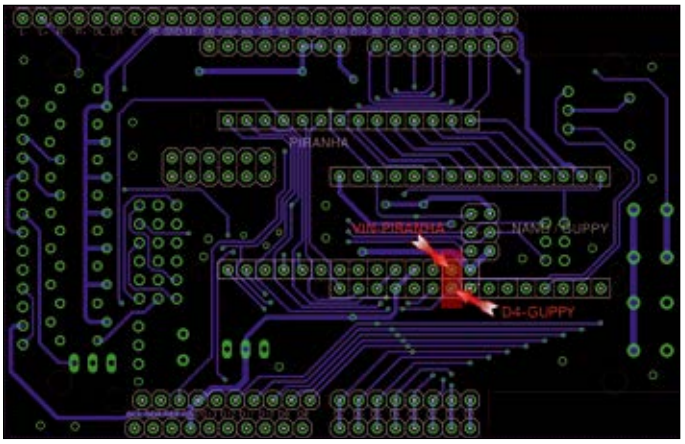


***Fig. 5** - I punti di interferenza su Fishino32.*

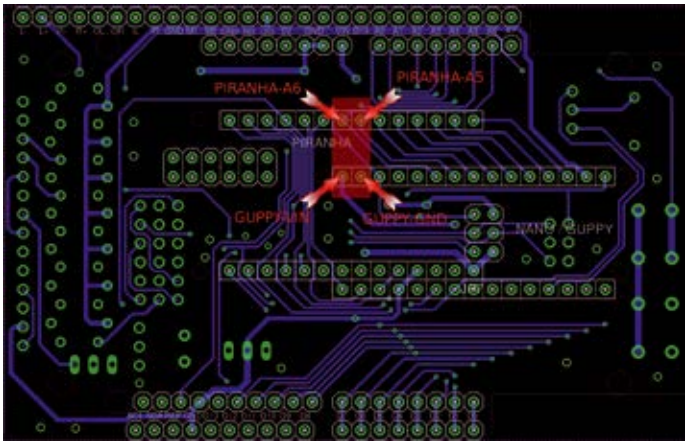


***Fig. 6** - Punti di interferenza su Fishino UNO.*

non è possibile sfruttarlo nel nostro shield. Montando Guppy o Nano è quindi necessario inserire i tre jumper nel connettore JSPI che realizzano il collegamento richiesto alle linee SPI.

REALIZZAZIONE PRATICA

La costruzione dello shield è alla portata anche di chi ha poca esperienza, grazie alla possibilità di montare componenti passanti tradizionali. Nulla vieta, nel caso preferiate, di utilizzare i componenti SMD.
Consigliamo come sempre di montare per primi i componenti a basso profilo e poco ingombranti, quindi resistenze e diodi, seguiti dai condensatori elettrolitici, il regolatore di tensione ed il MOSFET. Successivamente vanno disposti i pulsanti ed il cicalino e, per ultimi, essendo piuttosto ingombranti, i connettori.
Qui di seguito potete vedere due immagini dello shield, visto sia dal lato superiore (componenti passivi/attivi, jumper e connettore per il display TFT) sia dal lato inferiore (connettori per le board).
L'unica cosa a cui occorre fare particolare attenzione è il lato di montaggio dei vari header; montandoli dal lato sbagliato ovviamente non riuscirete ad infilare i controller, i jumper o il display!
Come anticipato in precedenza, a causa di alcune interferenze tra i vari connettori delle schede utilizzabili con il nostro shield, non è possibile montare tutti i connettori nello stesso tempo, a meno di non utilizzare particolari accorgimenti. Se ricordate, abbiamo prospettato 3 possibilità, la terza delle quali comporta una modifica alla scheda controller originale che però lasciamo alla scelta di chi è dotato di buona manualità.
Le altre due alternative sono quella di montare solo i connettori necessari alla propria scheda, cosa peraltro che non ha bisogno di ulteriori spiegazioni, oppure quella di eliminare da alcuni connettori i pin che interferiscono, rendendo lo shield sempre utilizzabile con tutte le schede, al prezzo di non portare all'esterno alcuni del segnali disponibili. Vediamo qui di seguito le interferenze e i pin da eliminare.

UTILIZZO CON FISHINO32

La board Fishino32 interferisce con i connettori delle schede Guppy/Piranha solo su due punti, visibili nella **Fig. 5**.
Evitando di montare il connettore nei due punti segnalati si risolvono completamente le interferenze, al costo di non portare all'esterno i segnali VIN della scheda PIRANHA ed il D4 della scheda Guppy. Il VIN è necessario solo se si alimenta la scheda con una tensione esterna, mentre il D4 del Guppy, essendo utilizzato anche dalla scheda SD interna, è probabilmente inutile per un utilizzo esterno.
Eliminando questi due "spezzoni" di connettore è quindi possibile montare senza ulteriori modifiche le seguenti schede: Arduino UNO, MEGA e Nano, Fishino MEGA, Fishino32, Fishino Guppy, Fishino Piranha ed Arduino MKR1000. La scheda Fishino UNO risente di altre interferenze, che descriveremo nei prossimi paragrafi.

UTILIZZO CON FISHINO UNO

La scheda FishinoUNO interferisce con i connettori delle board Guppy/Piranha in quattro punti, evidenziati nella **Fig. 6**.
Anche in questo caso è possibile rendere il nostro shield TFT "universale" eliminando i pin dei connettori indicati dalle frecce nella predetta figura. Tuttavia in questo caso occorre sacrificare le connessioni esterne sui due canali analogici A5 ed A6 del PIRANHA e la VIN della Guppy. La linea GND è presente anche su un altro pin quindi non dà problemi.
Eliminando questi quattro pin è possibile montare

le seguenti schede: Arduino UNO/MEGA/NANO, Fishino UNO, Fishino MEGA, Fishino GUPPY, Fishino PIRANHA, ma non la Fishino32 che richiede l'eliminazione dei pin al paragrafo precedente. Prendendo entrambi gli accorgimenti è ovviamente possibile montare tutte le schede disponibili.

ED ORA... SOFTWARE!

Come accennato nell'introduzione, per la gestione dello shield abbiamo approntato tre librerie che ne consentono il controllo completo. Queste sono:

- **FishinoGFX**, versione praticamente identica all'analoga di Adafruit, che gestisce le funzioni grafiche "ad alto livello";
- **FishinoILI9341**, che gestisce le funzioni di interfaccia con il display a livello hardware; anche questa libreria è stata realizzata partendo dall'analoga di Adafruit, ma con modifiche abbastanza sostanziali;
- **FishinoXPT2046**; che gestisce il touch screen, scritta da zero di nostro pugno.

Iniziamo dalla sezione display, con uno sketch semplicissimo che inizializza lo schermo e disegna una croce in due colori; lo trovate nel **Listato 1**.
Come potete notare lo sketch è semplicissimo. La funzione **loop()** risulta vuota (non c'è nulla da ripetere!), mentre tutto si svolge nella **setup()**.
La prima linea di questa:

```
tft.begin();
```

inizializza lo schermo. Utilizzando lo shield con le schede previste non occorre specificare i pin di connessione; la libreria si occupa di tutto. Volendo utilizzare connessioni differenti, occorre specificare quali I/O utilizzare, nella funzione **begin()**:

```
tft.begin(cs_pin, dc_pin);
```

Dove in cs_pin e dc_pin vanno indicati i pin cui sono connesse le linee CS e DC del display.
Andiamo alla linea:

```
tft.fillScreen(ILI9341_BLACK);
```

la quale riempie semplicemente lo schermo di nero, cancellandolo. Successivamente troviamo le due righe di codice:

```
tft.drawLine(0, 0, tft.width(), tft.height(), ILI9341_RED);
tft.drawLine(0, tft.height(), tft.width(), 0, ILI9341_GREEN);
```

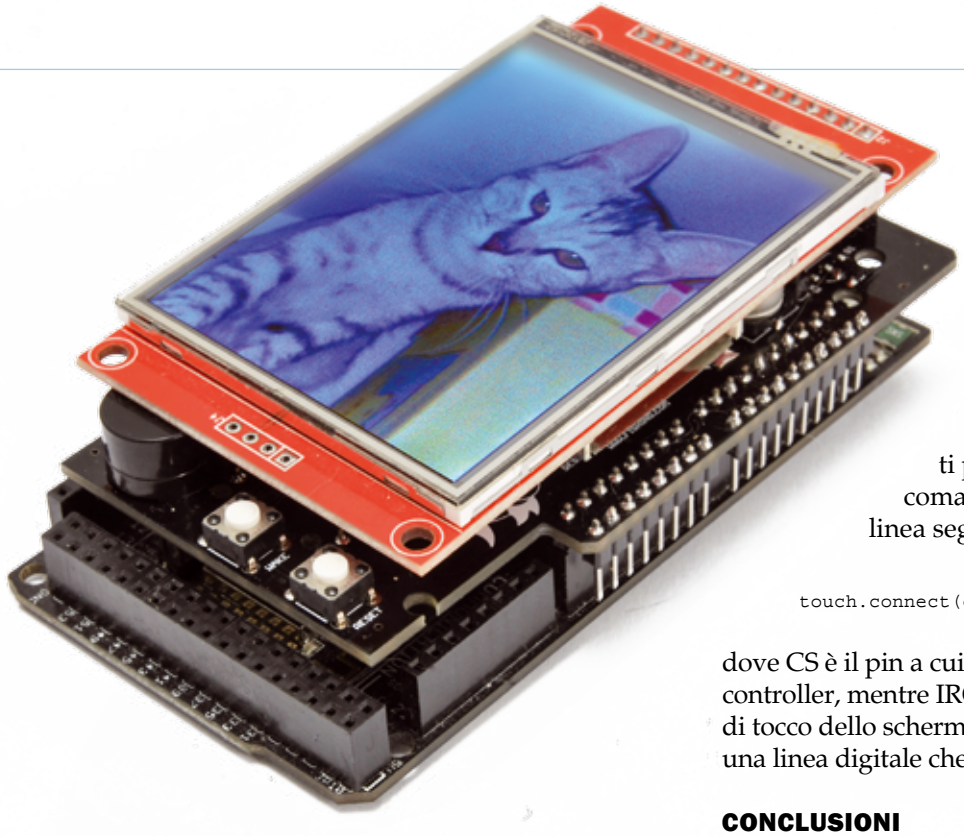
che si occupano di disegnare due linee di colori differenti (rosso e verde) che attraversano lo schermo. Le funzioni **tft.width()** e **tft.height()** forniscono rispettivamente la larghezza e l'altezza del display in pixel.
La libreria è molto estesa e tra le funzioni che implementa permette di tracciare punti, linee, cerchi, rettangoli, immagini, testi, eccetera. In essa è anche possibile "ruotare" lo schermo, in

Listato 1

```
#include <SPI.h>
#include <FishinoGFX.h>
#include <FishinoILI9341.h>

// questa linea è per comodità, in modo da non dover scrivere
// FishinoILI9341 ad ogni comando inviato al display
#define tft                FishinoILI9341

void setup()
{
    // inizializza lo schermo
    tft.begin();

    // cancella lo sfondo riempiendolo di nero
    tft.fillScreen(ILI9341_BLACK);

    // disegna una croce che percorre tutto lo schermo
    // in due colori
    tft.drawLine(0, 0, tft.width(), tft.height(), ILI9341_RED);
    tft.drawLine(0, tft.height(), tft.width(), 0, ILI9341_GREEN);

}

void loop()
{
}
```

Listato 2

```
#include <Flash.h>
#include <FishinoXPT2046.h>
#include <SPI.h>

// questa linea è per comodità, in modo da non dover scrivere
// FishinoXPT2046 ad ogni comando
#define touch              FishinoXPT2046

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

void loop()
{
        if(touch.touching())
        {
                uint16_t x, y, z;
                touch.read(x, y, z);
                Serial << "X : " << x << "\n";
                Serial << "Y : " << y << "\n";
                Serial << "Z : " << z << "\n";
                Serial << "------------------\n";
        }
        delay(200);
}
```



Shield e display montati sulla Fishino MEGA.

occorre specificare le connessioni hardware SE si utilizza lo shield con le schede previste; altrimenti prima di iniziare ad usare i comandi occorre indicarle tramite la linea seguente nella **setup()**:

```
touch.connect(cs, irq);
```

dove CS è il pin a cui è collegata la linea CS del touch controller, mentre IRQ è la linea relativa al segnale di tocco dello schermo, che deve essere connessa ad una linea digitale che supporta gli interrupt.

CONCLUSIONI

In questo articolo abbiamo colmato quella che ritenevamo una lacuna, ossia la mancanza di una periferica display per le nostre Fishino; ecco quindi uno shield con LCD grafico e dotato addirittura di interfaccia tattile, abbinabile a tutte le board Fishino sinora proposte e alla nascente Piranha.
Con i brevissimi esempi di codice appena proposti e descritti si conclude la presentazione del nostro shield display per le board Fishino.
Prossimamente pubblicheremo un esempio di utilizzo più complesso, sfruttando le caratteristiche di Fishino32. ■

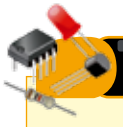
modo da disegnarci sopra in modalità orizzontale o addirittura di capovolgere l'immagine visualizzata.
Nella libreria sono contenuti alcuni esempi, tra cui due "painter" che permettono di disegnare sullo schermo sfruttando il touch-screen incorporato, oltre a un test grafico che mostra le varie primitive disponibili.
Vediamo ora la sezione touch-screen, della quale si occupa la libreria **FishinoXPT2046**.
Anche qui mostriamo un esempio semplice-semplice, rimandando quelli allegati alle librerie per le caratteristiche più complesse.
Lo sketch mostrato nel **Listato 2** attende semplicemente che si tocchi lo schermo e stampa sul monitor seriale dell'IDE Arduino la posizione in cui avviene il tocco.
Come potete notare, anche qui la semplicità è disarmante; la **setup()** si limita ad avviare la porta seriale, mentre nella loop si attende il tocco dello schermo tramite l'istruzione:

```
if(touch.touching())
```

e, quando questo avviene, si leggono le coordinate ed il valore di pressione tramite le istruzioni:

```
uint16_t x, y, z;
touch.read(x, y, z);
```

che poi vengono visualizzati sul monitor seriale.
Anche qui, come per la libreria del display, non



per il MATERIALE

Lo shield (cod. FT1328K) viene venduto in Kit al prezzo di Euro 19,00. Lo shield non comprende il display che però è possibile acquistare separatamente scegliendo fra il display LCD touch da 2,4" SPI (cod. LCDTOUCH24SER) che costa Euro 16,90, o quello da 2,8" SPI (cod. LCDTOUCH28SER) che è disponibile a Euro 19,50. Tutti questi prodotti si possono acquistare presso Futura Elettronica. I prezzi si intendono IVA compresa.

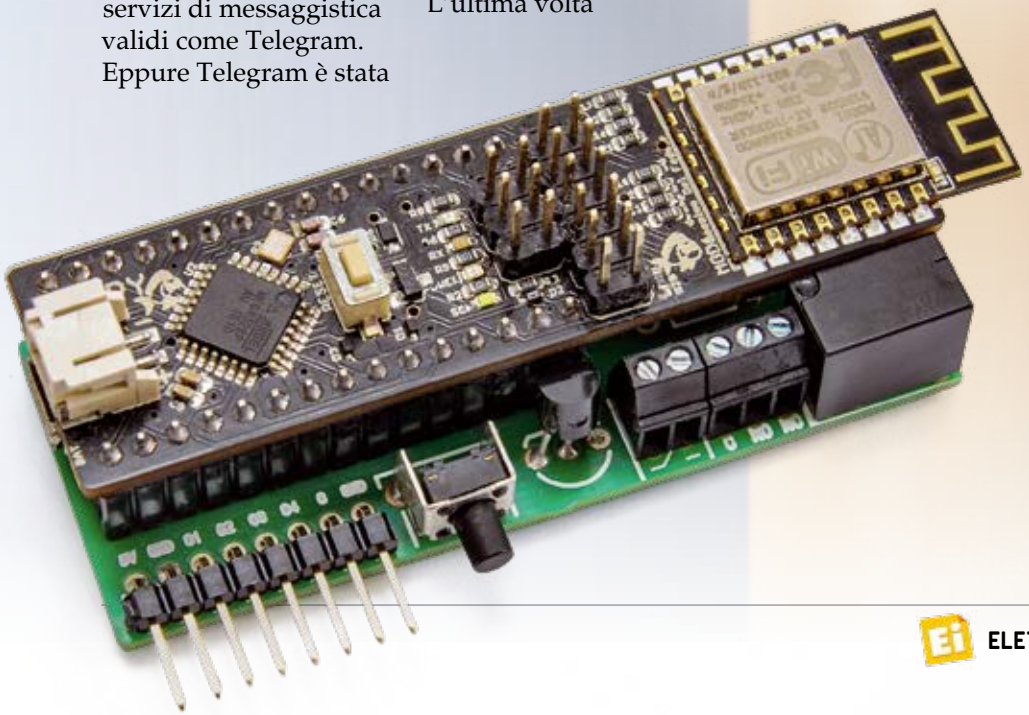
Il materiale va richiesto a:
Futura Elettronica, Via Adige 11, 21013 Gallarate (VA)
Tel: 0331-799775 - http://www.futurashop.it

TERMOSTATO CON FISHINO

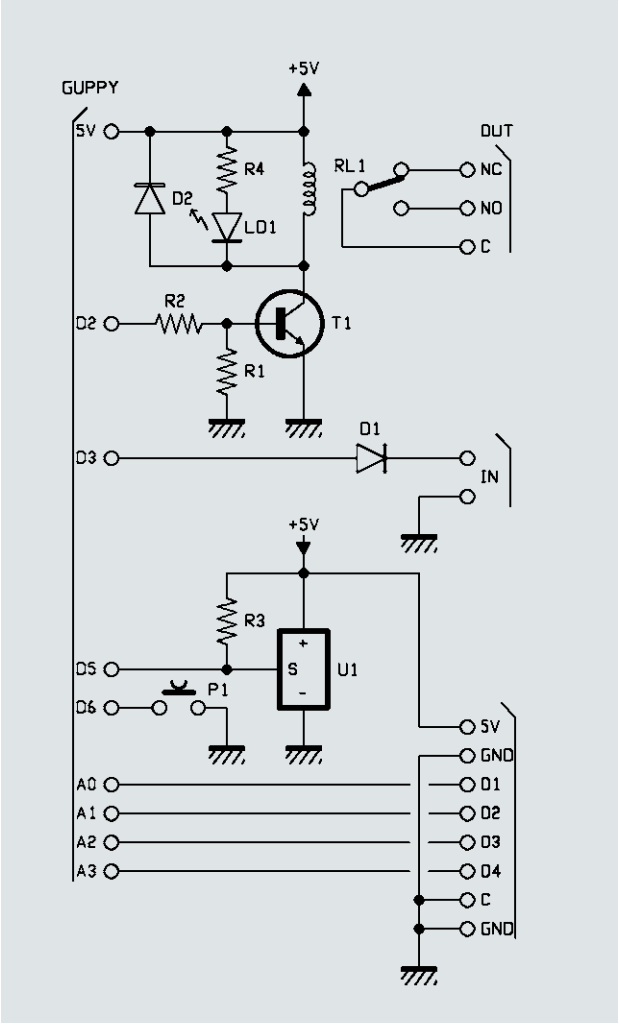
di ANDREA SIMONE COSTA

Malgrado in questi anni si siano affermati, dilagando, i social network e i servizi di instant messaging, molte persone non vanno oltre Whatsapp, ignorando l'esistenza di servizi di messaggistica validi come Telegram. Eppure Telegram è stata la prima app di messaggistica ad adottare la crittografia dei messaggi, implementata solo di recente in Whatsapp. La particolarità che rende "superiore" Telegram è il fatto che supporta i bot, che sono utenti virtuali o, se preferite, dei robot software, in grado di interagire come fossero veri utenti e di realizzare varie funzioni in maniera automatica; utenti particolari anche perché non devono essere associati a un numero di telefono.
L'ultima volta che abbiamo parlato di Telegram e utilizzato i suoi bot è stato nel progetto Notes Machine comparso nel numero di Febbraio 2017. In esso abbiamo già analizzato le potenzialità della libreria Fishgram, creata per sfruttare appieno il nostro bot, perciò senza dilungarci troppo sulla sua implementazione, in questo articolo ne vedremo un'altra interessante applicazione.
Accenneremo anche a una piattaforma IoT, ovvero ThingSpeak, la quale ci permette di inviare gratuitamente dei dati dal nostro Fishino per disporli su un grafico, anch'esso in buona parte personalizzabile.
Per questo progetto è stato utilizzato un Fishino Guppy, molto utile date le sue ridotte dimensioni, in simbiosi con una semplice PCB creata su misura. Su di essa possiamo trovare:

- un pulsante, con la possibilità di colle-



Sfruttando la versatilità dei bot messi a disposizione da Telegram, sviluppiamo un termostato controllabile da smartphone con semplici messaggi istantanei.



garne un altro tramite un pratico morsetto;
- un connettore S.I.L. maschio per utilizzare quattro I/O, Vcc e massa all'esterno (utili per eventuali espansioni);
- un sensore di temperatura digitale DS18B20;
- un relé con relativo morsetto, il cui funzionamento è segnalato da un LED.

Per praticità, con il nostro codice gestiremo solamente il sensore di temperatura e il relé, ma nulla vieta di connettere più relé trasformando gli ingressi analogici in uscite digitali via software.

IL TERMOSTATO

Innanzitutto spendiamo due parole per spiegare cos'è e come funziona il nostro termostato, la cui logica di funzionamento verrà replicata nello sketch. Si tratta di un termostato composto da un elemento sensibile alle variazioni di temperatura, che aziona un interruttore o deviatore al superamento o alla discesa sotto una soglia preimpostata. Il controllo delle temperature avviene direttamente da smartphone, grazie all'app Telegram. Il nostro sistema, inoltre, pubblicherà on-line la variazione di temperatura su una piattaforma consultabile da qualsiasi dispositivo e in caso di superamento di una temperatura di allarme, ci avviserà immediatamente tramite un messaggio personalizzabile da sketch.
Di norma i termostati sono caratterizzati da un'isteresi, vale a dire che l'apertura o chiusura dell'interruttore avviene a due soglie di temperatura: una superiore e una inferiore e anche nel nostro sistema questa possibilità è stata prevista.
Nel programma che andremo a creare sarà possibile impostare un'isteresi a minimo 1 °C, ciò significa che la differenza tra la soglia superiore e quella inferiore della temperatura che vogliamo mantenere, sarà di 2 °C (ritoccando il software potrete restringere l'isteresi).
Per chiarire questo concetto immaginiamo di essere in estate e di voler mantenere una temperatura stabile di 20°C; per impostazione predefinita, la soglia superiore verrà posizionata a 21°C e quella inferiore a 19°C. Appena la temperatura dell'ambiente supererà i 21°C il termostato attiverà il condizionatore per far scendere la temperatura. Quando verrà interrotta l'azione del condizionatore? Qui entra in gioco la soglia inferiore: appena la temperatura scenderà sotto i 19°C il condizionatore verrà arrestato. Esso rientrerà in funzione solo nel caso in cui la temperatura superasse di nuovo i 21 °C.
Anche queste soglie (oltre quelle di allarme, necessarie per avvisarci in caso di malfunzionamento) e la temperatura da mantenere saranno impostate grazie a Telegram.

IL SENSORE DI TEMPERATURA DS18B20

Il DS18B20 è un sensore di temperatura digitale. Ciò significa che esso comunica i valori registrati alla Fishino non tramite una tensione da leggere su un pin analogico, come ad esempio il famoso LM35, ma attraverso un segnale digitale. Per interpretare correttamente questo segnale è perciò necessaria una libreria, anzi due.
La prima è la OneWire, la quale permette di comu-

CARATTERISTICHE TECNICHE

- **Connessione WiFi**
- **Gestione tramite smartphone**
- **Sensore di temperatura DS18B20**
- **Temperatura e isteresi programmabili**
- **Soglie di allarme programmabili**
- **Uscita a relé**

nicare con qualsiasi sensore che, appunto, invia i dati attraverso un solo pin digitale.
La seconda è la DallasTemperature, scritta per questa famiglia di sensori, che si appoggia alla OneWire per comunicare con il nostro DS18B20, come mostrano le seguenti righe di codice:

```
#define TEMP 5
OneWire oneWire(TEMP);
DallasTemperature sensors(&oneWire);
```

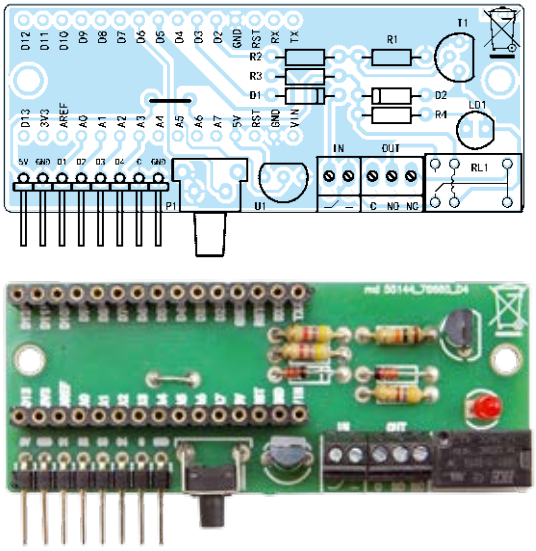
La PCB è stata strutturata in modo che il pin di comunicazione del sensore sia connesso al pin 5 della Guppy.
Perciò "diamo in pasto" questo pin alla libreria OneWire creandone un'istanza, dopodiché attiviamo anche la DallasTemperature inviando ad essa il riferimento all'istanza precedentemente creata.
Queste istruzioni sono da inserire prima della **void setup()**. In essa invece è sufficiente avviare la comunicazione tra il sensore e il microcontrollore con:

```
sensors.begin()
```

Qui termina la configurazione necessaria per il sensore. Nella **void loop()**, la quale, ricordiamo, deve durare il meno possibile per evitare chiamate a `FishGram.loop()` troppo distanti tra loro in termini di tempo, sfruttiamo la nota millis() per leggere dal sensore il valore della temperatura ogni tot secondi.
Il sensore non è tra i più veloci, nel senso che si adegua lentamente alle variazioni di temperatura; ma questo non è un problema poiché la temperatura in un ambiente come una stanza difficilmente presenta variazioni repentine.
Infatti abbiamo previsto circa 30 secondi tra un'interrogazione e l'altra del sensore.
Le istruzioni principali che troviamo nel blocco di codice sono le seguenti:

```
sensors.requestTemperatures();
temp = sensors.getTempCByIndex(0);
relay();
```

Le prime due sono necessarie per richiedere e ricevere la temperatura dall'unico sensore presente, nonché primo e per questo legato all'indice di valore 0. La temperatura viene poi memorizzata nella variabile `temp`.
La **void relay()** si occupa, come è facile intuire, della gestione del relé presente sulla PCB e verrà analizzata più avanti.



Elenco Componenti:

R1: 10 kohm
R2, R3: 4,7 kohm
R4: 470 ohm
T1: BC547
LD1: LED 3 mm rosso
P1: Microswitch 90°
D1, D2: 1N4148
U1: DS18B20
RL1: Relé 5V singolo scambio

Varie:
- Strip maschio 90° 8 vie
- Morsetto 2 poli passo 2,54mm
- Morsetto 3 poli passo 2,54mm
- Strip femmina tornito 15 vie (2 pz.)
- Circuito stampato S1314 (69 x 28mm)

THINGSPEAK

Sappiamo perfettamente che il mondo dell'IoT non è limitato ad apparecchiature sofisticate e costose ed oggi ne vedremo un esempio: creeremo un basilare sistema IoT costituito dalla Fishino, la quale si preoccuperà di raccogliere i dati relativi alla temperatura per poi inviarli sulla rete, e da una piattaforma IoT sul web, la quale avrà come compito l'analisi e la visualizzazione di questi dati. Parliamo di Thingspeak, che rappresenta un ottimo connubio tra semplicità di utilizzo e affidabilità, oltre ad avere il supporto nativo al mondo di Arduino e una formula base gratuita che è più che sufficiente per il nostro scopo.
Una volta creato un account su questa piattaforma web (accessibile da *https://thingspeak.com)*, dopo aver confermato la nostra identità tramite un'e-mail di conferma che riceveremo nella nostra casella di posta, ci ritroveremo davanti alla schermata rappresentata nella **Fig. 1**.
Dovremo quindi creare un nuovo canale, il quale si può considerare come il "luogo" virtuale sul quale i nostri dati verranno immagazzinati e visualizza-

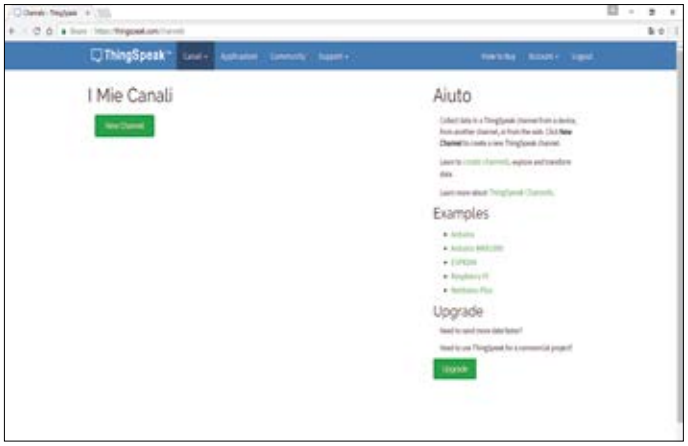


Fig. 1 *- L'account in Thingspeak.*

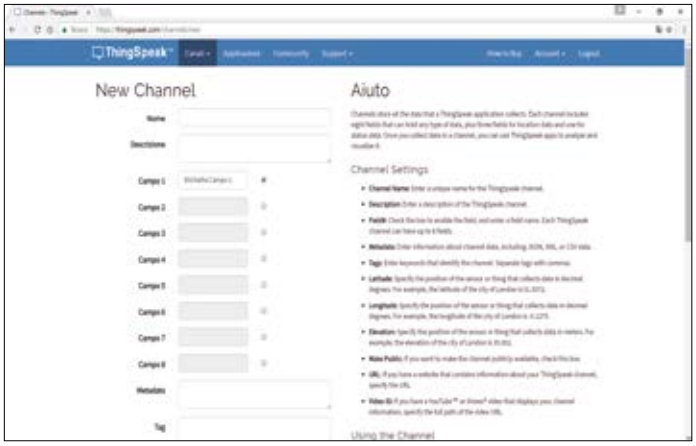


Fig. 2 *- La finestra di creazione del canale.*

ti, facendo clic sul pulsante *New Channel*. Questo ci aprirà la pagina visualizzata nella **Fig. 2**, dove troviamo le impostazioni da effettuare per il nostro canale: possiamo scegliere un nome, una descrizione, decidere se rendere o meno il canale pubblico e scegliere quanti diversi campi dovrà avere il nostro canale (fino a un massimo di otto), potendoli rinominare a piacimento. In altre parole, nel caso in cui vogliamo visualizzare una temperatura, l'umidità e la luminosità di una stanza, non c'è bisogno di creare tre canali diversi. È sufficiente creare un canale e assegnare ad esso tre campi: uno per ciascuna grandezza fisica misurata.
Nella **Fig. 3** possiamo vedere una delle possibili configurazioni del canale utilizzato per questo progetto.
Creato e configurato il canale, clicchiamo sul pulsante *Save Channel*, che si trova ben evidente nella parte finale della pagina web visibile nella **Fig. 4**, in modo da salvare le impostazioni del nostro canale.
Fatto ciò, non ci rimane che inviare i nostri dati; per fare questo abbiamo bisogno della chiave di lettura (la quale non sarà utilizzata per il nostro progetto) e scrittura del canale, oltre al suo ID. Nella schermata che si apre, dopo aver fatto clic sul pulsante *Save Channel*, clicchiamo su Chiavi API e annotiamo le due chiavi, oltre all'ID del canale che compare poco più sopra **Fig. 5**.
Adesso possiamo finalmente analizzare il codice presente nel nostro sketch.
Come prima cosa includiamo la libreria ThingSpeak, senza la quale dovremmo occuparci manualmente delle varie richieste http. Prima della **void setup()** troviamo anche:

```
FishinoClient client;
unsigned long myChannelNumber =  241871;
const char * myReadAPIKey = "****************";
const char * myWriteAPIKey = "****************";
```

La comunicazione, come già accennato, avviene tramite il protocollo http, per questo è necessario creare un'istanza della classe FishinoClient. Dopodiché inseriamo l'ID del canale e le due chiavi precedentemente ottenute nelle rispettive variabili.
Nella **void setup()** troviamo invece un'unica istruzione, la quale avvia la libreria grazie all'oggetto client precedentemente creato:

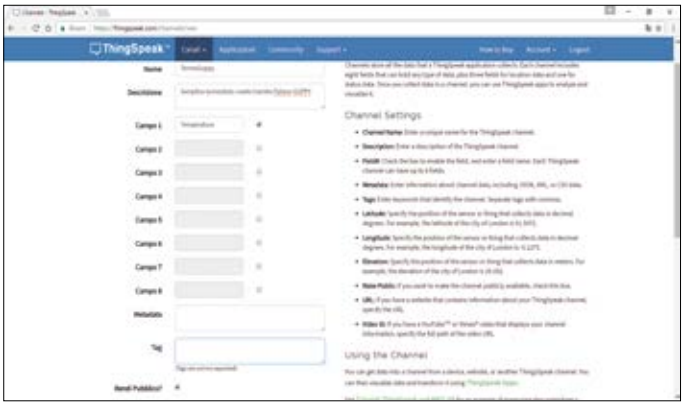


Fig. 3 *- Una possibile configurazione del canale.*

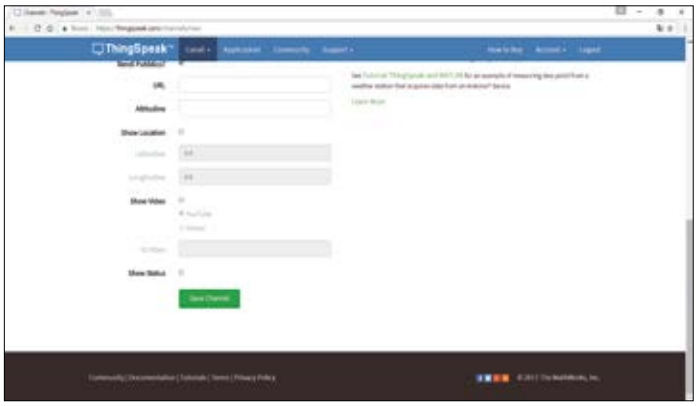


Fig. 4 *- Il pulsante per il salvataggio del canale creato.*

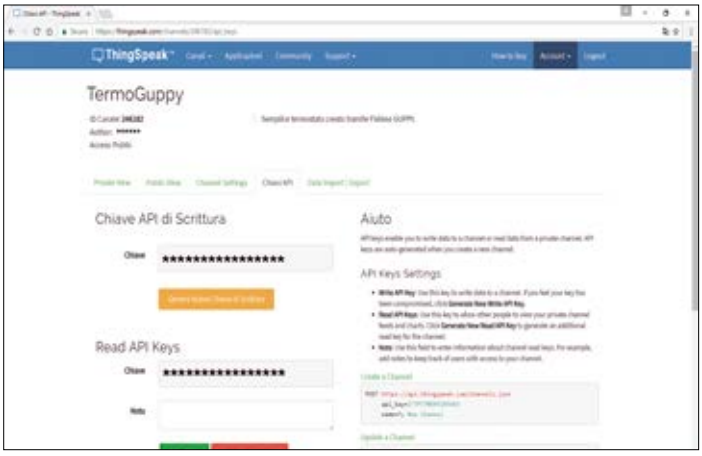


Fig. 5 - *Chiavi e ID canale.*

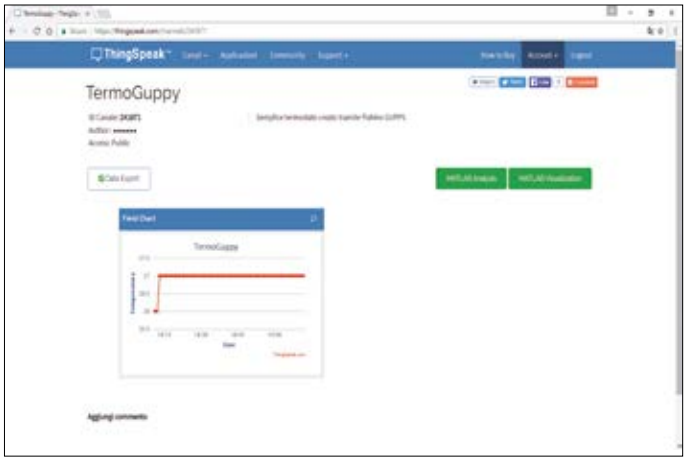


Fig. 6 - *Visualizzazione di un canale di tipo pubblico*

```
ThingSpeak.begin(client);
```

Infine nella void **loop()**, sempre sfruttando la funzione **millis()**, inviamo al nostro canale su ThingSpeak il valore della temperatura tramite le seguenti due istruzioni:

```
ThingSpeak.setField(1, temp);
ThingSpeak.writeFields(myChannelNumber,
myWriteAPIKey);
```

Nella prima scegliamo in quale degli otto campi visualizzare la temperatura; nel nostro caso abbiamo attivato solo il primo per questo scopo, perciò inseriamo il numero 1, e scegliamo quale valore inviare. Dato che la temperatura viene memorizzata nella variabile temp è questa che dobbiamo inserire come secondo parametro.
Nella seconda permettiamo alla Fishino Guppy di scrivere i dati sul canale tramite il suo ID e la chiave di scrittura. Ecco fatto!
Nel caso in cui abbiamo impostato il nostro canale come pubblico, è sufficiente digitare il seguente URL per visualizzare da un qualunque dispositivo il risultato (**Fig. 6**), naturalmente sostituendo alla voce 'idcanale' l'ID del nostro canale:

https://thingspeak.com/channels/idcanale

È doveroso fare una precisazione: la versione base di ThingSpeak prevede 15 secondi di intervallo minimo tra due aggiornamenti del valore della grandezza sotto osservazione, oltre a 20 secondi di analisi del dato ricevuto.
Inviare il valore della temperatura ogni 60 secondi è perciò un buon compromesso.

FISHGRAM

L'articolo comparso sul numero di Febbraio 2017 ha esaustivamente presentato e analizzato la versatilità di questa libreria, la quale ci permette di instaurare una comunicazione tra Fishino e un qualsiasi dispositivo che supporti Telegram tramite un bot. Perciò entreremo subito nel cuore del codice (scaricabile dal sito della rivista), naturalmente con le dovute spiegazioni.
Diversamente dal progetto della Notes Machine utilizzeremo un procedimento più semplice per memorizzare i vari comandi; in essa si era presentato l'uso dei templates, ma ci limiteremo a comparare alcune stringhe. Ovviamente sarà un codice meno elegante e performante rispetto al precedente, ma di immediata comprensione.
Prima di analizzare nel dettaglio la **FishgramHandler()** incentriamo l'attenzione sulle seguenti istruzioni presenti nel setup:

```
bool b = true;
FishGram.restrict(b);
FishGram.allow(myId);
```

Tramite le prime due andiamo ad attivare la whitelist messa a disposizione da Fishgram. Solamente gli ID presenti in questa lista saranno autorizzati a comunicare con il Fishino attraverso il bot. Per aggiungere un ID a questa lista si utilizza la terza istruzione, da ripetere nel caso di più ID.
Tra gli esempi messi a disposizione da questa libreria ce ne è uno che ci permette agilmente di conoscere sia il nostro ID che quello di chiunque scriva al bot controllato da Fishino, semplicemente aprendo il monitor seriale.
Consideriamo adesso la **FishgramHandler()**. Dopo quattro costrutti if che analizzeremo a breve, i quali controllano alcune variabili booleane, troviamo una seconda serie di cinque costrutti if con una condizione leggermente particolare.
La funzione **strcmp()** presente in ciascuno di essi

confronta il messaggio appena ricevuto con una stringa da noi decisa.
Nel caso in cui quest'ultima è esattamente uguale al messaggio la funzione **strcmp()** restituisce 0, per questo è necessario anteporre ad essa il punto esclamativo di negazione:

```
if (!strcmp(message, "/temp"))
```

Trovata l'uguaglianza, questi cinque costrutti if si limitano a creare ed inviare un messaggio di risposta per poi terminare la funzione **FishgramHandler()**:

```
String ans = F("La temperatura vale: ");
ans += temp;
ans += "°C";
FishGram.sendMessage(id, ans.c_str());
return true;
```

Questi ci permettono di conoscere l'ultimo valore di temperatura misurato, le due soglie di allerta in caso di malfunzionamento, la temperatura che abbiamo deciso di mantenere e il valore del differenziale, cioè la differenza tra la soglia superiore e quella inferiore del ciclo di isteresi della temperatura. La terza serie di sei costrutti if permette invece di impostare alcuni di questi valori, appoggiandosi ad alcune variabili booleane. I primi due sono speculari, infatti impostano o la stagione calda o la stagione fredda modificando il valore della variabile `hot`.
Per fare un piccolo esempio: in estate superati i 22°C sarà necessario attivare il condizionatore, mentre in inverno, superata la stessa soglia, sarà necessario disattivare l'impianto di riscaldamento. Questa variabile verrà sfruttata nella void **relay()** che discuteremo più avanti.
Gli ultimi quattro costrutti if della terza serie "attivano" ciascuno la propria variabile booleana, disattivando quella degli altri tre, per poi anch'essi inviare un breve messaggio e terminare la **FishgramHandler()**:

```
if (!strcmp(message, "/set_up"))
  {
    set_up = true;
    set_temp = false;
    set_diff = false;
    set_down = false;
    String ans = F("Imposta la soglia superiore:");
    FishGram.sendMessage(id, ans.c_str());
    return true;
  }
```

Il messaggio chiederà di inviare un valore numerico per impostare ad esempio la soglia di allerta superiore. Ciò significa che alla successiva chiamata della **FishgramHandler()** il messaggio non dovrà essere analizzato dalle varie **strcmp()**; dovrà invece andare a modificare il valore della variabile corrispondente all'opzione da impostare.
Per questo i quattro costrutti if sono stati inseriti per primi. Essendo `set_up = true` si eseguirà la seguente porzione di codice:

```
if (set_up)
  {
    if ((atoi(message) > (setTemp + diff / 2)))
    {
      sup = atoi(message);
      String ans = F("Soglia superiore impostata a ");
      ans += sup;
      ans += "°C";
      FishGram.sendMessage(id, ans.c_str());
      set_up = false;
    } else {
      String ans = F("Soglia superiore non valida, riprova");
      FishGram.sendMessage(id, ans.c_str());
    }
    return true;
  }
```

Il messaggio contenente il valore numerico verrà analizzato dalla funzione **atoi()**, la quale converte una stringa in un numero intero. Se questo valore è maggiore o uguale alla temperatura da mantenere sommata alla metà del differenziale, ovvero se è maggiore o uguale della soglia superiore del ciclo di isteresi, allora verrà impostata la variabile corrispondente e dopo un messaggio di conferma la variabile booleana di appoggio verrà "disattivata". Se invece il valore non è corretto l'utente verrà

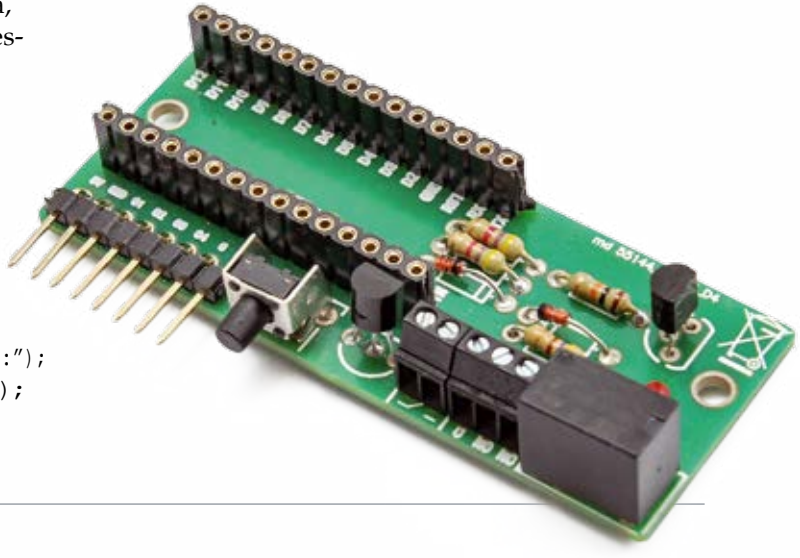


Fig. 7
Il menu creato.

Fig. 8
Il pulsante slash per consultare il menu.

avvisato e la variabile `set_up` rimarrà `true` per permettere un altro tentativo.
Adesso consideriamo meglio le condizioni da verificare prima di memorizzare un nuovo valore nella variabile contenente la soglia di allerta superiore (all'interno del codice troverete tutti i commenti che spiegano nel dettaglio tutti i test che vengono fatti ai valori prima di essere memorizzati).
Come già accennato la soglia di allerta superiore deve essere maggiore o uguale alla soglia superiore del ciclo di isteresi. Esempio chiarificatore. Se vogliamo mantenere 20°C, con un differenziale impostato a 2°C, la soglia superiore del ciclo di isteresi automaticamente sarà pari a 21°C, cioè la metà del differenziale sommata alla temperatura da mantenere. Per questo sarebbe inadeguata una soglia superiore di allerta per malfunzionamenti inferiore ai 21°C.
Lo stesso ragionamento si applica alla soglia di allerta inferiore.
Per impostazione predefinita, queste variabili vengono impostate a:

- 20°C per la temperatura da mantenere;
- 100°C per la soglia superiore;
- 0°C per la soglia inferiore;
- 2°C per l'isteresi.

Dal punto di vista dell'utente Telegram, potrebbe risultare difficile memorizzare tutti i vari comandi disponibili. Ecco che l'app ci viene in aiuto, perché infatti in un paio di passaggi è possibile creare un piccolo menu, con la lista dei comandi e le relative descrizioni (**Fig. 7**), che verrà visualizzato alla pressione dell'icona contenente uno slash (**Fig. 8**).
Infatti i comandi su Telegram iniziano proprio con uno slash "/", come ad esempio "/start" che serve ad avviare per la prima volta un bot. Per questo i comandi da noi scelti e inseriti nelle varie strcmp() iniziano tutti con "/".
In effetti non è obbligatorio, però non inserendo uno slash iniziale sacrificheremo l'opportunità di creare la lista di comandi.
Colleghiamoci quindi al "padre di tutti i bot", ovvero BotFather, come abbiamo fatto per creare un bot seguendo l'articolo di Febbraio 2017.
Inseriamo quindi il comando "/setcommands" e selezioniamo il bot interessato: **Doinobot**, nel nostro caso (**Fig. 9**).
A questo punto BotFather ci chiede di inserire i vari comandi in un messaggio, seguendo un preciso schema, senza però aggiungere lo slash iniziale (**Fig. 10**). Per il nostro progetto risponderemo quindi con il seguente messaggio (**Fig. 11**):

temp - visualizza l'ultima temperatura rilevata
up - visualizza la soglia superiore

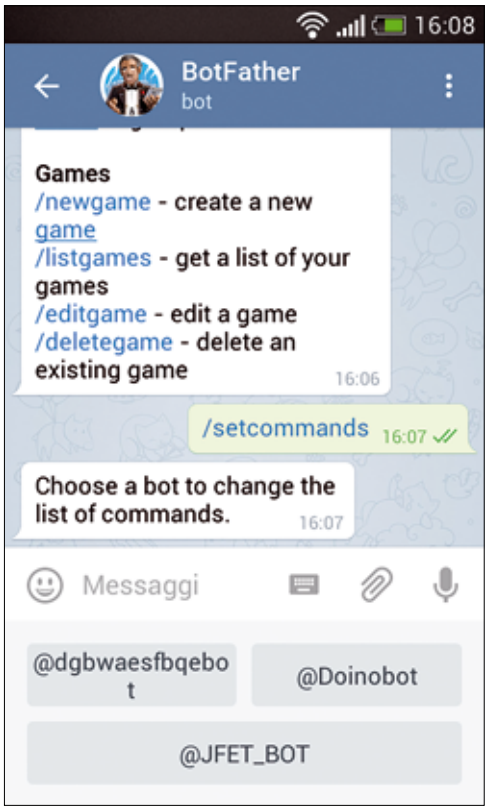


Fig. 9 *Invio del comando verso Doinobot.*

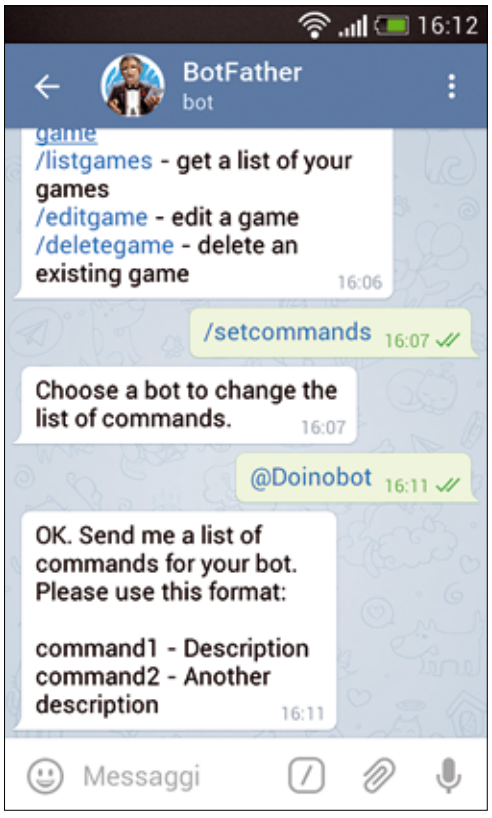


Fig. 10 *Telegram richiede l'inserimento di tutti i comandi in un unico messaggio.*

down - visualizza la soglia inferiore
stemp - visualizza la temperatura da mantenere
diff - visualizza il differenziale
set_hot - imposta stagione calda
set_cold - imposta stagione fredda
set_temp - imposta la temperatura da mantenere
set_diff - imposta il differenziale
set_up - imposta la soglia di allerta superiore
set_down - imposta la soglia di allerta inferiore

Se tutto è andato a buon fine, riceveremo il messaggio di conferma di **Fig. 12.** A questo punto torniamo nella chat personale con il bot e premiamo sull'icona contenente uno slash, evidenziata nella **Fig. 8**. La **Fig. 7** ci mostra il risultato ottenuto. Nella **Fig. 13** possiamo vedere la risposta che otteniamo se premiamo sul comando "/temp".
Infine non dimentichiamo di inserire l'istruzione `FishGram.loop();` all'interno della void loop(). Ricordiamo che essa deve durare il meno possibile, per garantire un corretto funzionamento della libreria Fishgram.

VOID RELAY()

Affrontiamo, infine, la spiegazione di questa procedura, che viene chiamata qualche istante dopo l'effettuazione di una nuova misura della temperatura.

```
if (hot)
  {
    if (temp > (setTemp + diff / 2))
    {
      digitalWrite(RELAY, HIGH);
    }
    if (temp < (setTemp - diff / 2))
    {
      digitalWrite(RELAY, LOW);
    }
  } else {
    if (temp > (setTemp + diff / 2))
    {
      digitalWrite(RELAY, LOW);
    }
    if (temp < (setTemp - diff / 2))
    {
      digitalWrite(RELAY, HIGH);
    }
  }
```

Il blocco di codice appena esposto controlla innanzitutto quale stagione è stata impostata, dopodiché nel caso in cui la temperatura superi la soglia superiore dell'isteresi (temperatura da mantenere + differenziale / 2) o scenda al di sotto di quella inferiore (temperatura da mantenere - differenziale / 2), rispettivamente attiverà o disattiverà il relé.

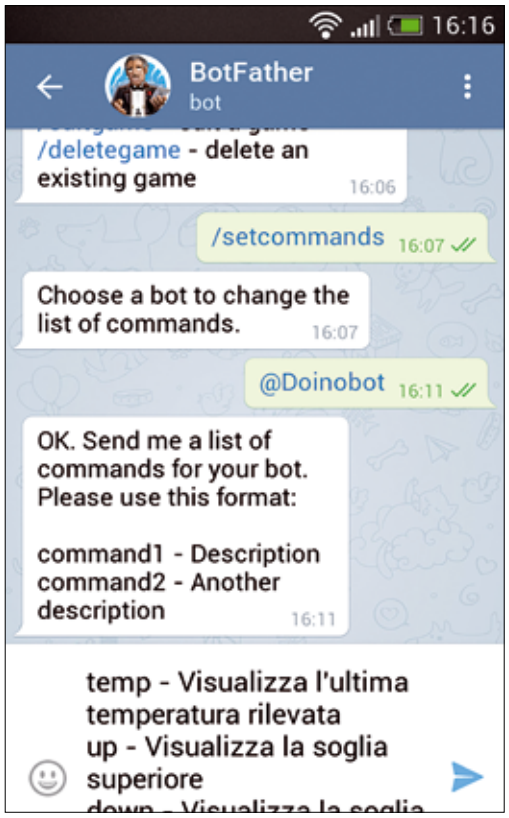


Fig. 11 *Il nostro messaggio di risposta.*

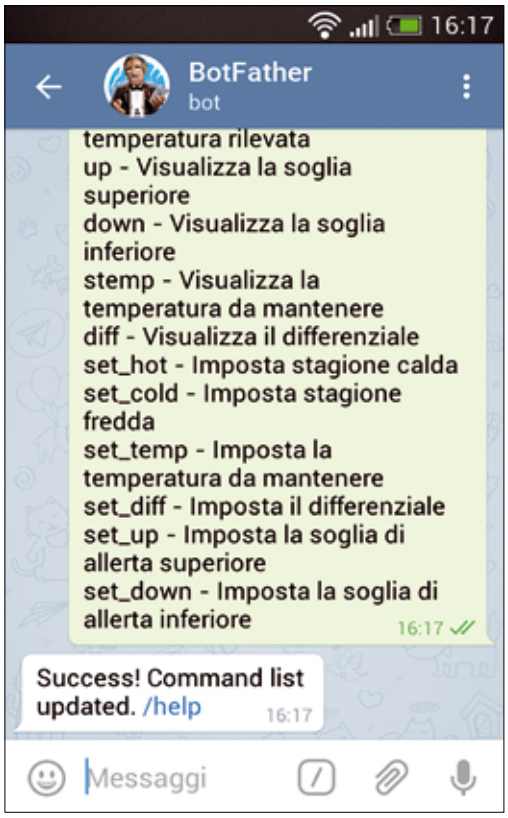


Fig. 12 *Messaggio di conferma da parte di BotFather.*

La macro RELAY è stata definita prima della **void setup()** assegnandole il valore 2, dato che utilizzando il circuito (ossia lo shield qui proposto) precedentemente descritto, il pin di attivazione del relé in esso presente si troverà connesso al pin 2 della Fishino Guppy.

```
if ((temp > sup) && (!supExceededMessage))
 {
   String ans = F("Attenzione!
   La temperatura e' …");
   FishGram.sendMessage(myId, ans.c_str());
   supExceededMessage = true;
 }
 if ((temp < inf) && (!infExceededMessage))
 {
   String ans = F("Attenzione!
   La temperatura e' …");
   FishGram.sendMessage(myId, ans.c_str());
   infExceededMessage = true;
 }
```

Il secondo blocco di codice (qui sopra) avverte di un possibile malfunzionamento, sfruttando i valori delle due soglie di allarme. La prima volta che la temperatura supera la soglia di avviso superiore o scende al di sotto di quella inferiore, essendo le variabili `supExceededMessage` e `infExceededMessage` settate a "false" per impostazione predefinita, all'utente verrà inviato un messaggio. Entrambi i costrutti if, prima di terminare, settano a "true" la rispettiva variabile booleana, in modo da non recapitare di nuovo il messaggio alla successiva chiamata della **void relay()**.

```
if ((supExceededMessage) && (temp <
sup))
  {
    String ans = F("Anormalita' risolta…");
    FishGram.sendMessage(myId, ans.c_str());
    supExceededMessage = false;
  }

  if ((infExceededMessage) && (temp > inf))
  {
    String ans = F("Anormalita' risolta…");
    FishGram.sendMessage(myId, ans.c_str());
    infExceededMessage = false;
  }
```

Quest'ultimo blocco di codice è utilizzato per resettare la condizione di allarme: appena la temperatura rientra nella norma, essendo `supExceededMessage` o `infExceededMessage` settate a "true", viene eseguito il corrispondente costrutto if che, sempre con un messaggio, avvisa l'utente che

Fig. 13 *Risposta al comando /temp.*

il problema è risolto. Le due variabili booleane vengono quindi reimpostate a "false" e il compito della **void relay()** termina qua.
Ovviamente sia la condizione di allarme che quella di reset dello stesso possono essere arricchite a piacere, ad esempio tramite l'accensione di LED o attraverso suoni di avviso.

CONCLUSIONE

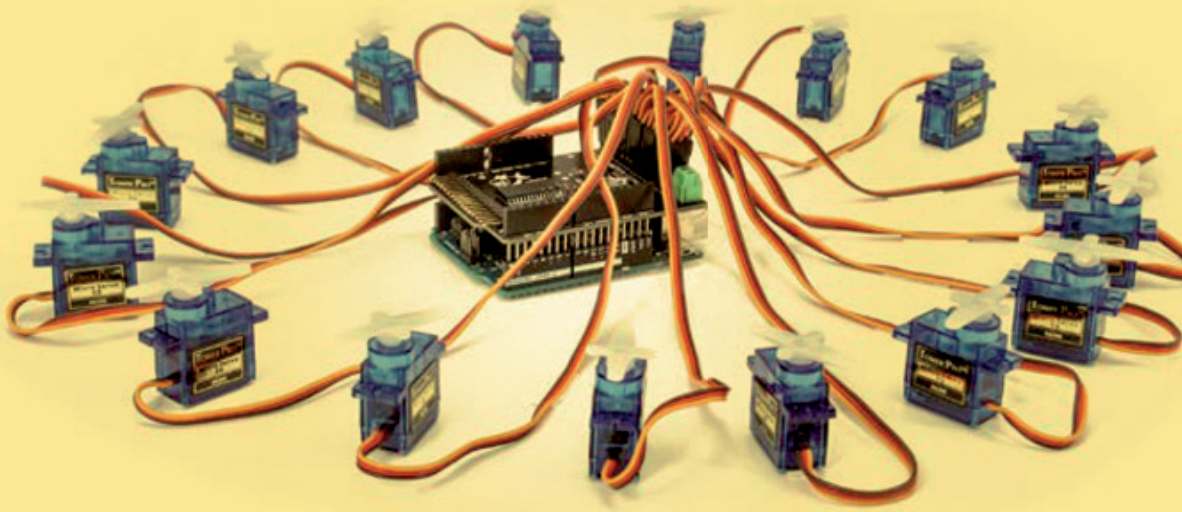
Un termostato è l'esempio ideale per mostrare sia le potenzialità della libreria Fishgram sia la versatilità delle schede Fishino, specialmente nell'ambito IoT. La connettività WiFi, unita alla capacità di sostenere scambio di dati tramite il protocollo HTTPS, è parte integrante di queste board, mantenendo la semplicità di utilizzo intrinseca del mondo Arduino.
Ci auguriamo di avervi fornito ancora una volta una prova di ciò con l'applicazione qui descritta. ■

per il MATERIALE

Tutti i componenti utilizzati in questo progetto sono di facile reperibilità. Il master del circuito stampato può essere scaricato dal sito della rivista (*www.elettronicain.it*). La scheda Fishino GUPPY (cod. GUPPY) è disponibile presso Futura Elettronica al prezzo di Euro 33,90. Il prezzo si intende IVA compresa.

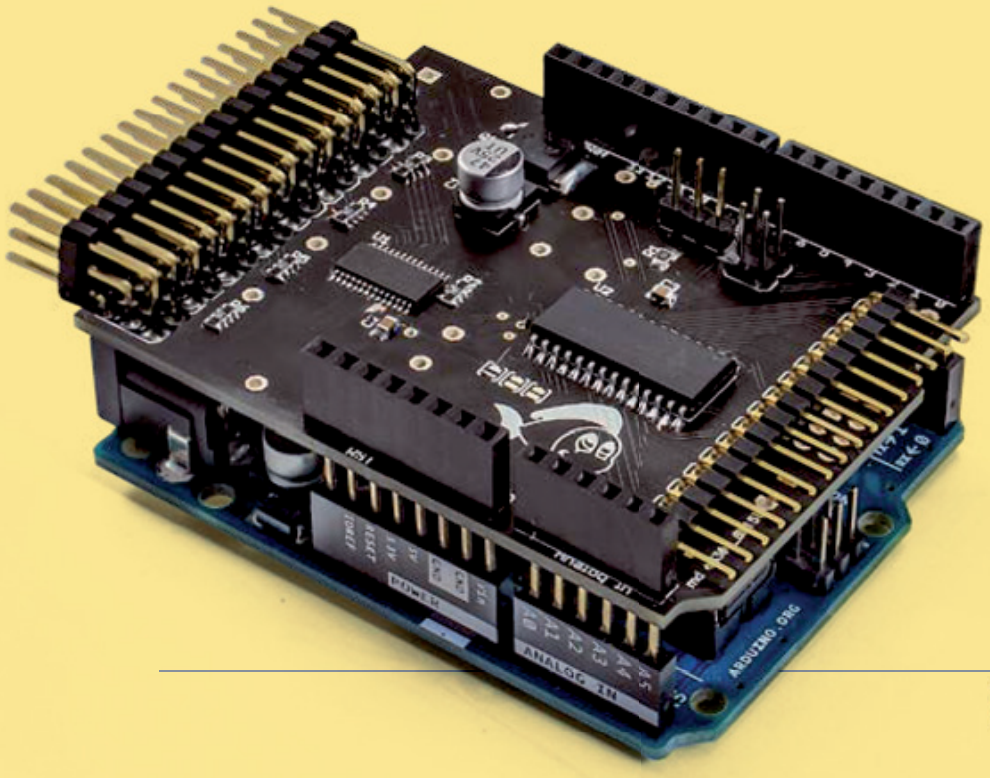
Il materiale va richiesto a:
Futura Elettronica, Via Adige 11,
21013 Gallarate (VA) - Tel: 0331-799775
http://www.futurashop.it

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.



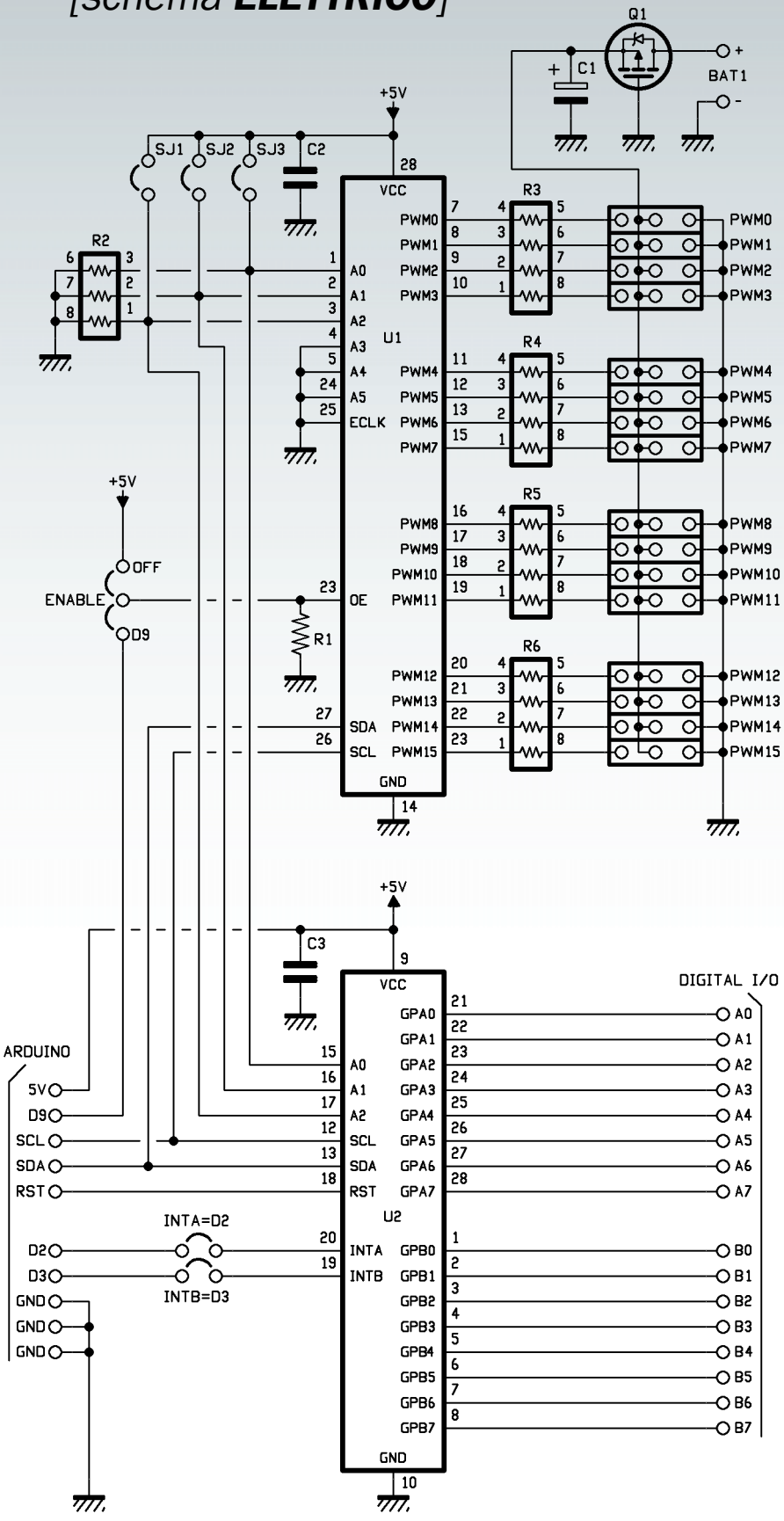
OCTOPUS, ESPANSIONE "TENTACOLARE" PER ARDUINO/FISHINO

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 registi (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.
Perchè 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 ininfluente 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:

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

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 µF, 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 le 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;
}
```

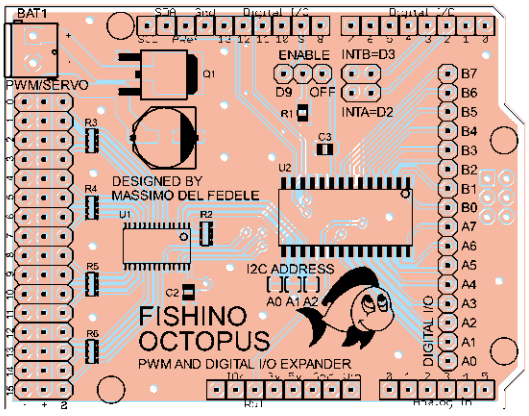
Questa modalità apparentemente strana di utilizzo della variabile **Octopus** permette di ovviare ad uno dei problemi del C++, ovvero dell'inizializzazione delle variabili globali che non avviene in un ordine predeterminato ma è casuale, e viene effettuata al caricamento dello sketch. Nel nostro caso,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

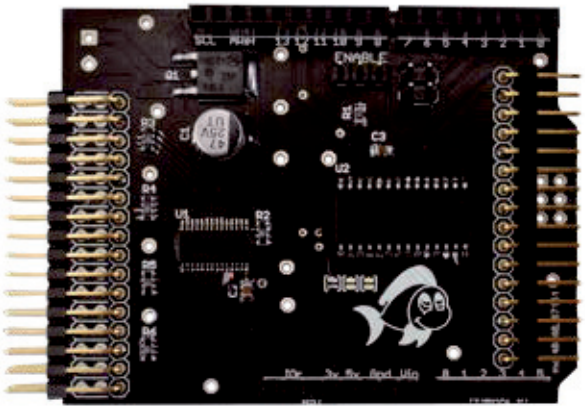
```
Wire.begin();
Wire.setClock(400000);
```

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 (DPACK)
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 in-
vert = 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.elettronicain.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'I²C, 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 ativa 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.elettronicain.it*. 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

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 messaggino 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, lasciarlo 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 disponibile (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 digitarne 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

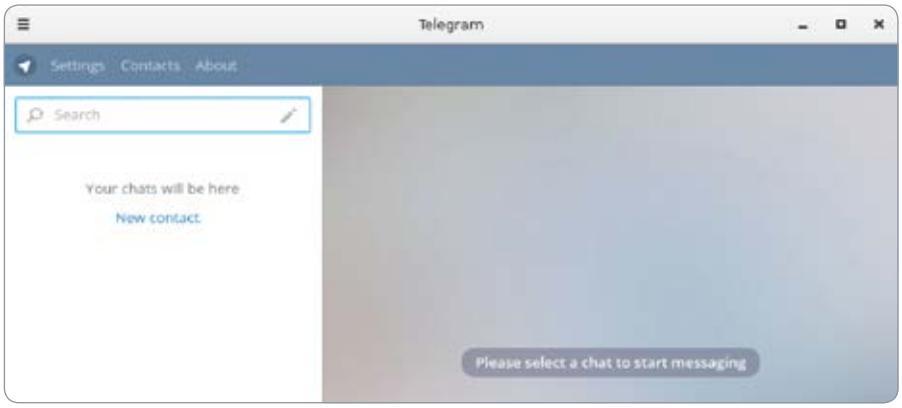


Fig. 1 *- Installazione di Telegram.*

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:
nnnnnnnnn:XXXXXXX-YYYYYYY_ZZZZZZZZZZZZZZZZZZZ

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 bellissimo 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.

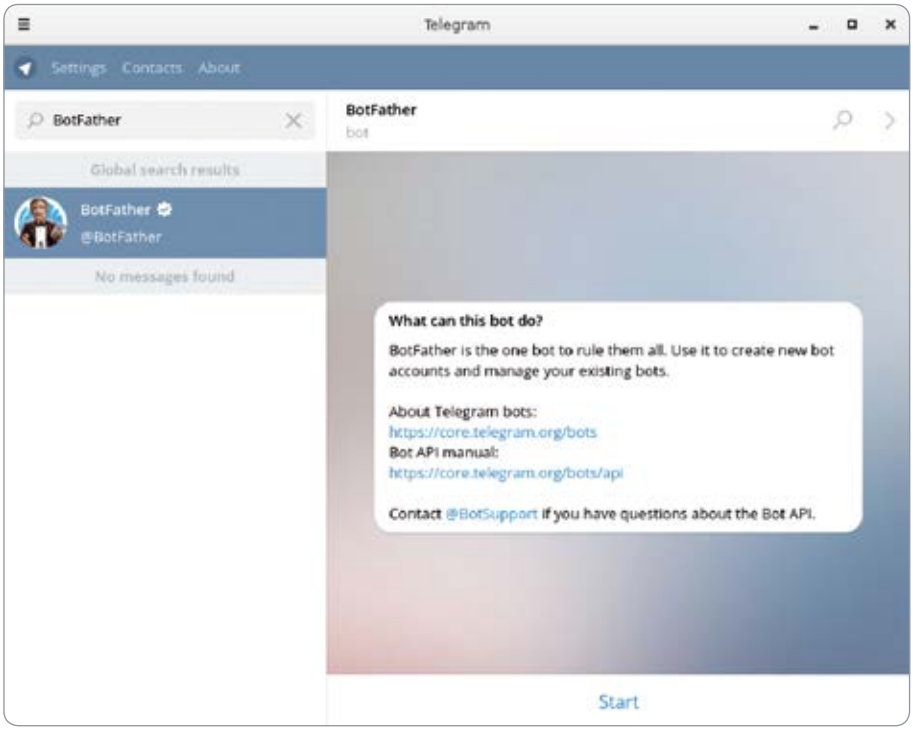


Fig. 2 *- Ricerca di BotFather.*

Possiamo provare il comando direttamente sul browser, immettendo nella casella dell'indirizzo il testo seguente (sostituite 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,"f
rom":{"id":nnnnnnnnn,"first_
name":"Massimo","last_name":"Del Fe
dele"},"chat":{"id":nnnnnnnnn,"fir
st_name":"Massimo","last_name":"Del
Fedele","type":"private"},"date":14
83380231,"text":"ciao Pippo"}}]}
```

Come si può notare, il testo di risposta contiene parecchie informazioni, tra cui il mittente del messaggio, l'id del messaggio (tramite il quale, per esempio, in una chiamata successiva inviando 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

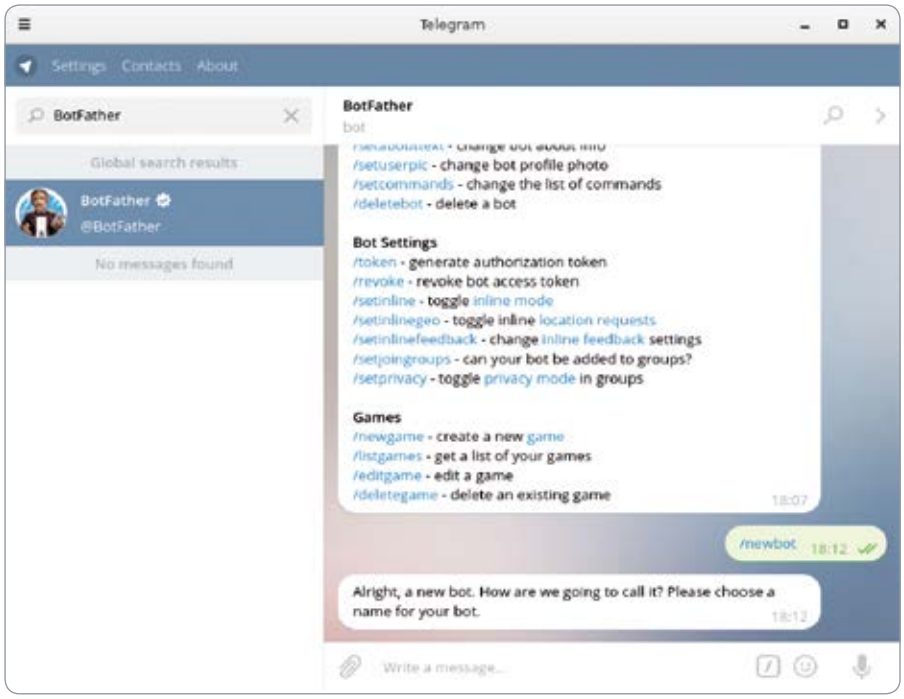


Fig. 4 *- Creazione del nostro Bot.*

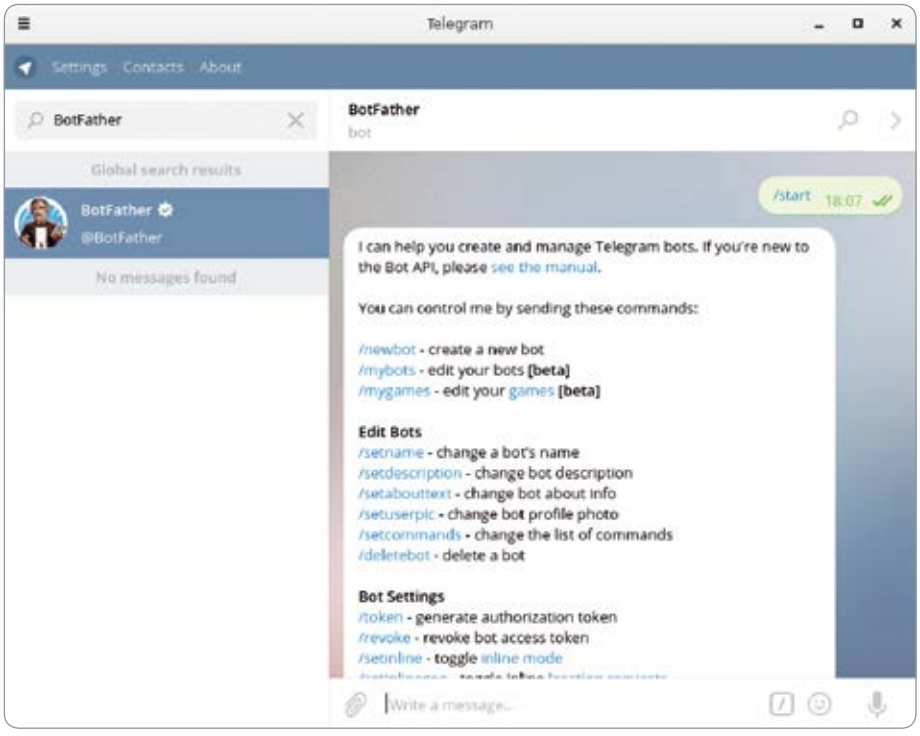


Fig. 3 *- Risposta di BotFather.*

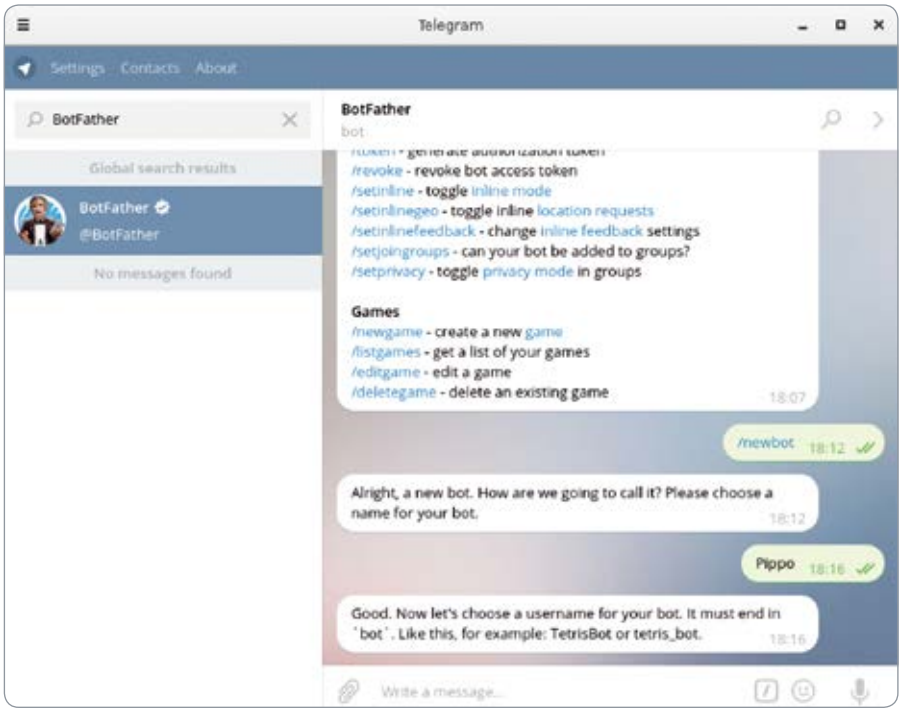


Fig. 5 *- Assegnazione del nome al Bot.*

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 *la-
stName, const char *message)
```

Il secondo è "inizializzazione di FishGram":

```
        // start FishGram
        FishGram.
event(FishGramHandler);
        FishGram.begin(F(MY_TELE-
GRAM_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

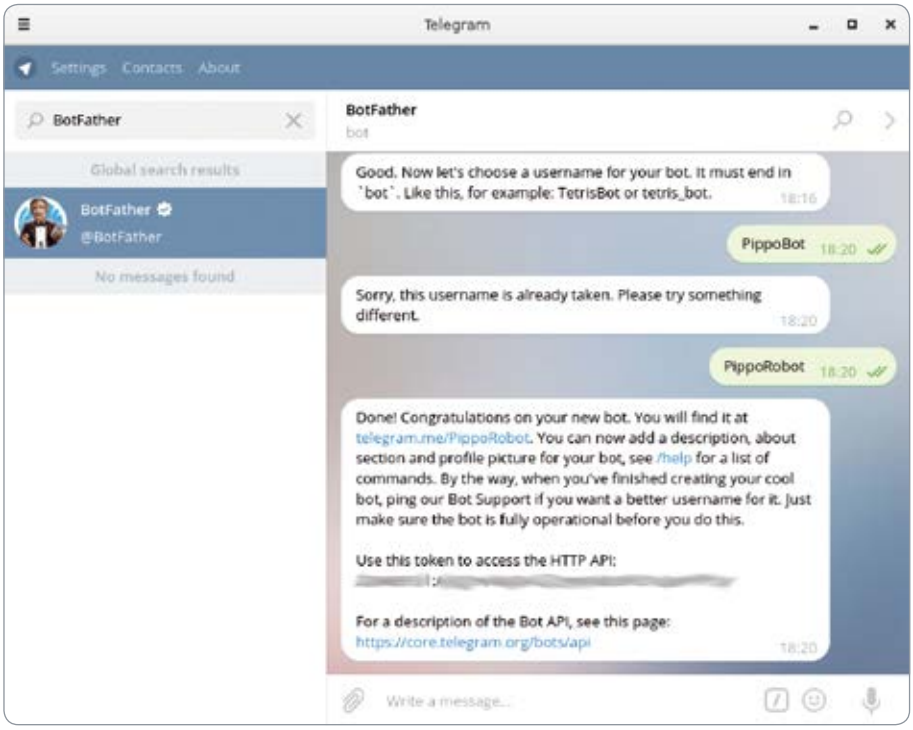


Fig. 6 *- Completamento del bot.*

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 nè bloccare l'esecuzione del **loop()** nè impiegare troppo tempo. Quindi, niente **delay()**, niente **while()** con durate lunghe e/o attesa di pul-

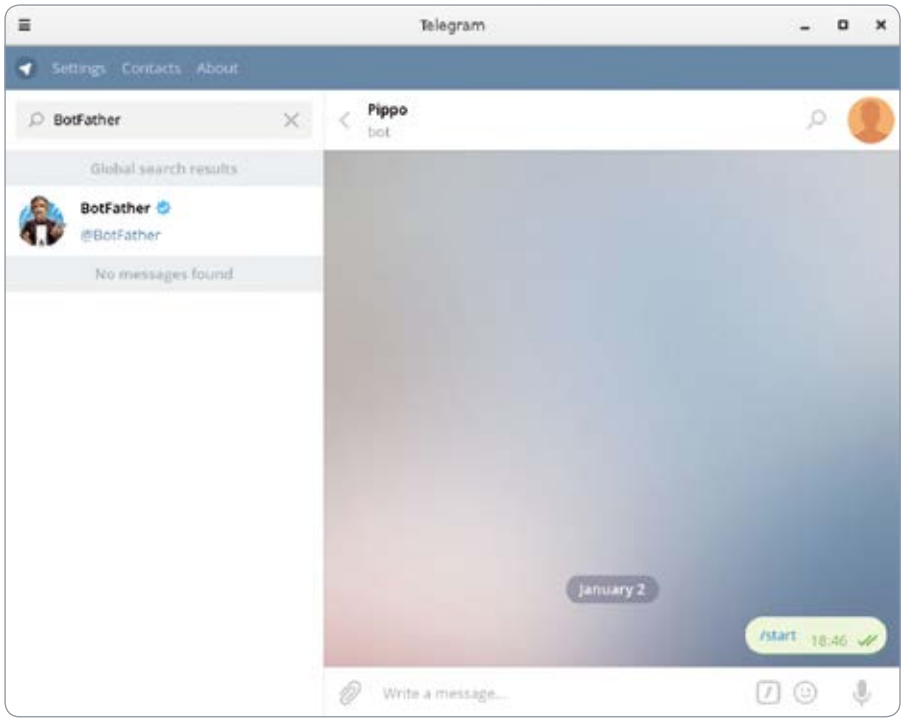


***Fig. 7** - Avvio del bot.*

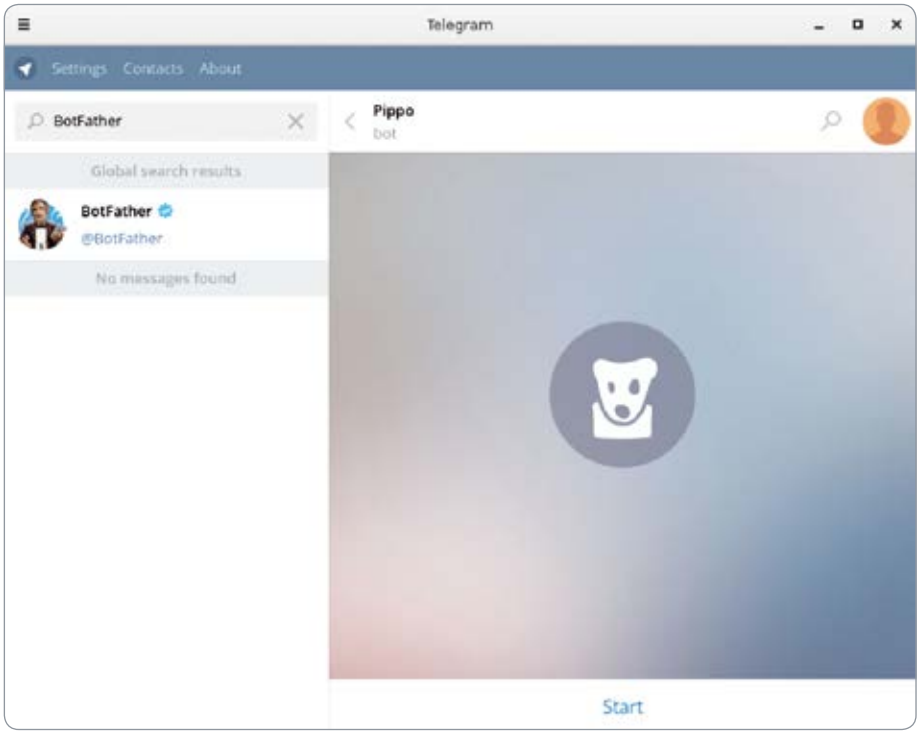


***Fig. 8** - Messaggio iniziale del bot.*

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 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";
        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 si... 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 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**.

Il sistema in versione San Valentino: la stampante è montata in un quadro in tema.

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

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 x 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.

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 recupererà i 10 messaggi precedenti:

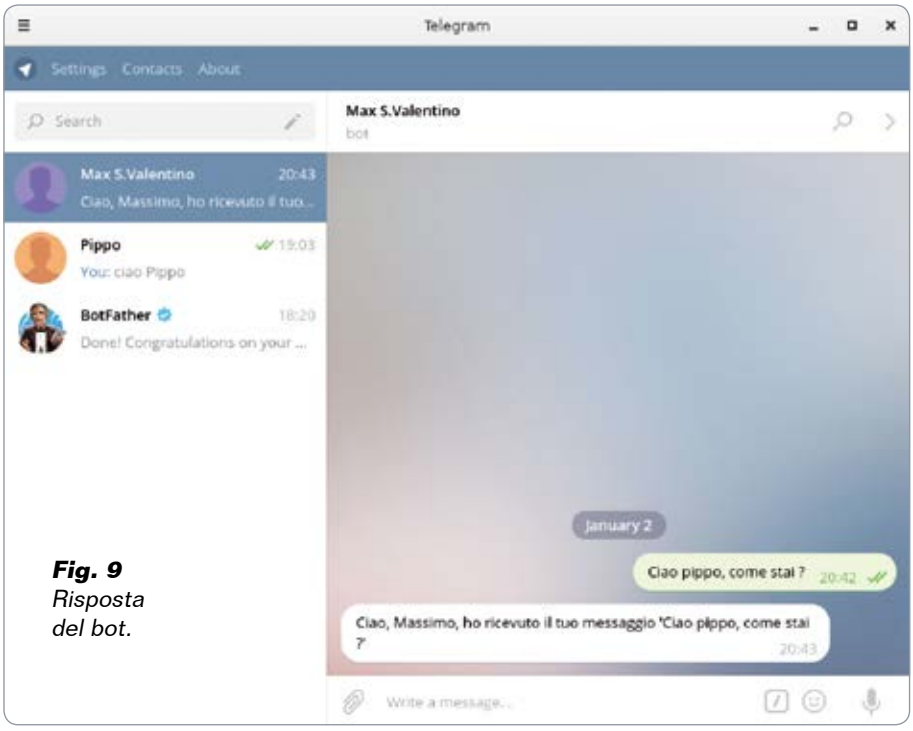


Fig. 9 *Risposta del bot.*

```
// 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 spe-
cifico pezzo per pezzo
bool startMessage(uint32_t const &id,
uint16_t len);
bool contMessage(const char *msg);
bool contMessage(const __FlashStrin-
gHelper *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";

        String s;
        s = "Ciao, ";
        s += firstName;
        s += ", ho ricevuto il tuo messaggio '";
        s += message;
        s += "'";
        FishGram.sendMessage(id, s.c_str());
        return true;
}
```

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 messaggino romantico;
- stampare una citazione o un messaggino 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).

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);
```

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

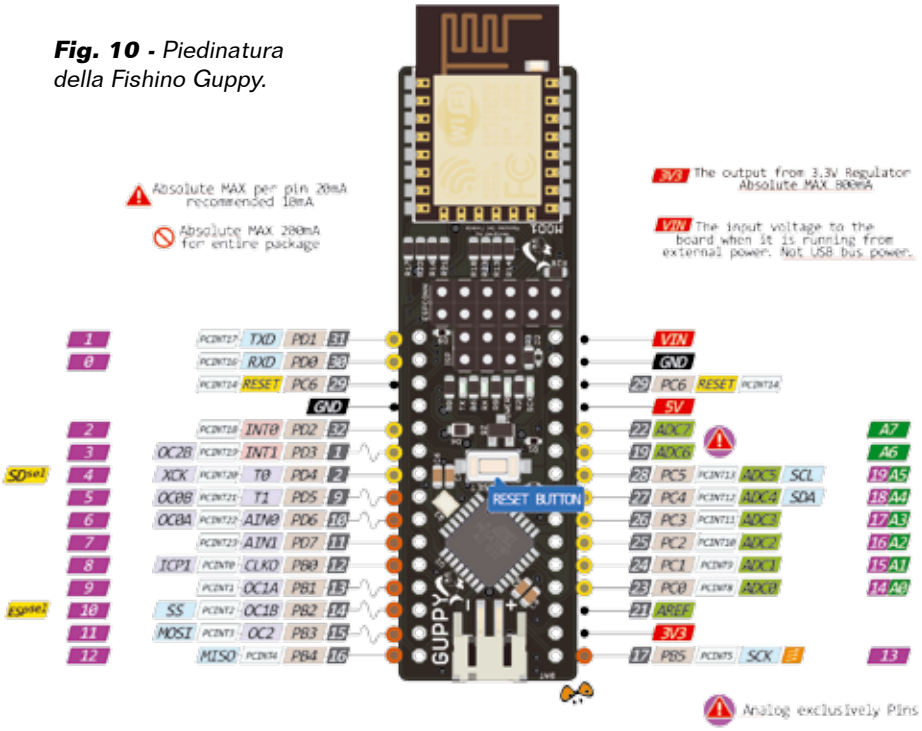


***Fig. 10** - Piedinatura della Fishino Guppy.*

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 desiderato, 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 nè 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

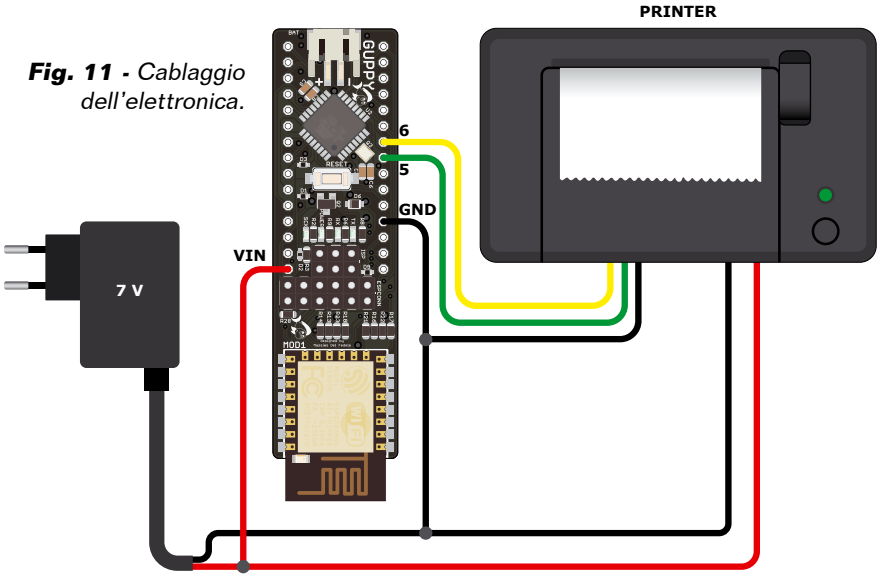


Fig. 11 *- Cablaggio dell'elettronica.*

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 mesaggio (**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 daremo 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.elettronicain.it* e, in seguito, verrà incluso tra gli esempi nelle librerie di Fishino.
Le citazioni (e le frasi romantiche) 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? Si, 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
```

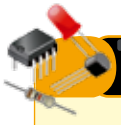
dove **ID** è un numero identificativo della citazione, per poterla "ripescare", ad esempio, per mandare al mittente la conferma di quanto si è stampato; **LEN** è la lunghezza del testo della citazione stessa, necessaria per poter utilizzare le funzioni **startMessage()** e annesse della libreria FishGram (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.

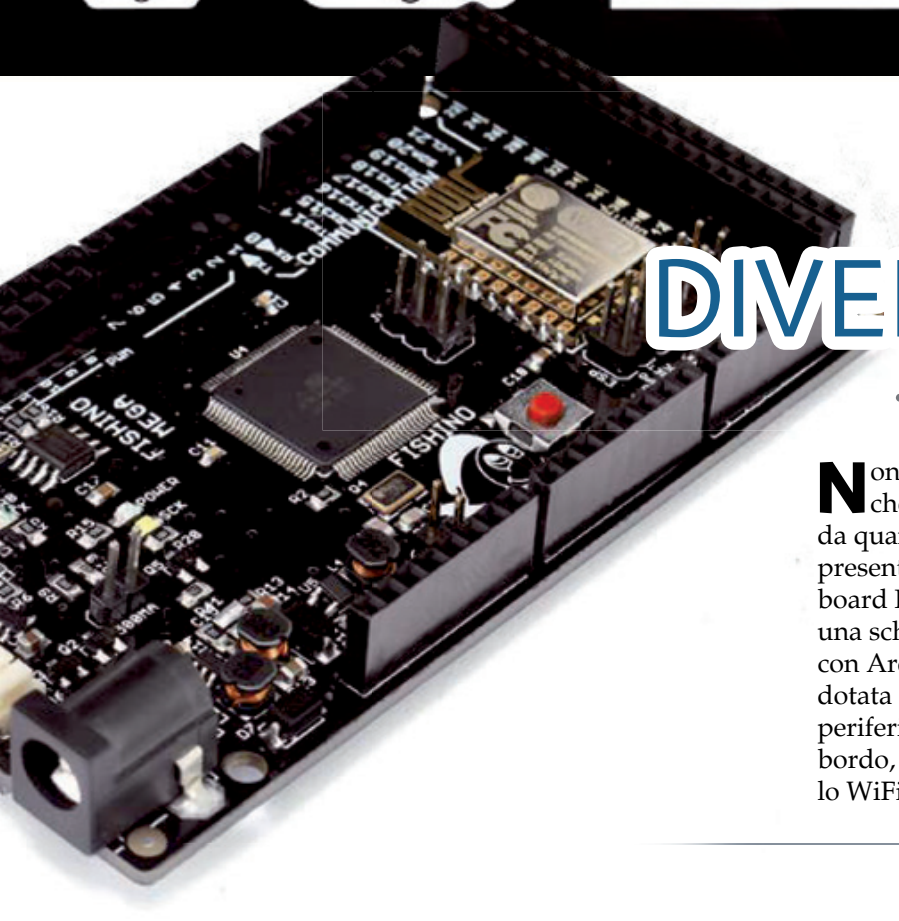


per il MATERIALE

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

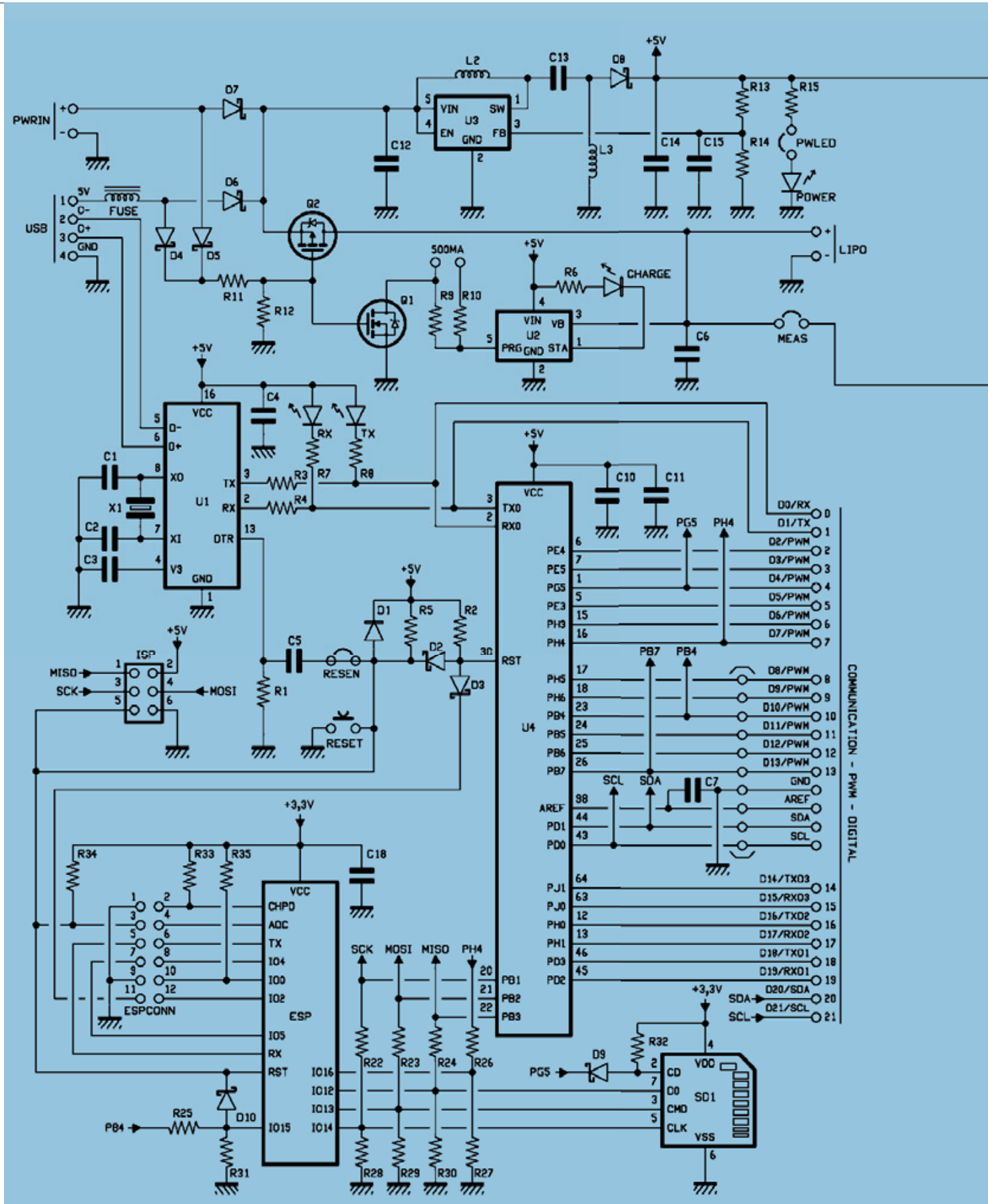
La nostra versione della popolare e potente Arduino MEGA, basata su un hardware rivisitato specialmente nella sezione di alimentazione.

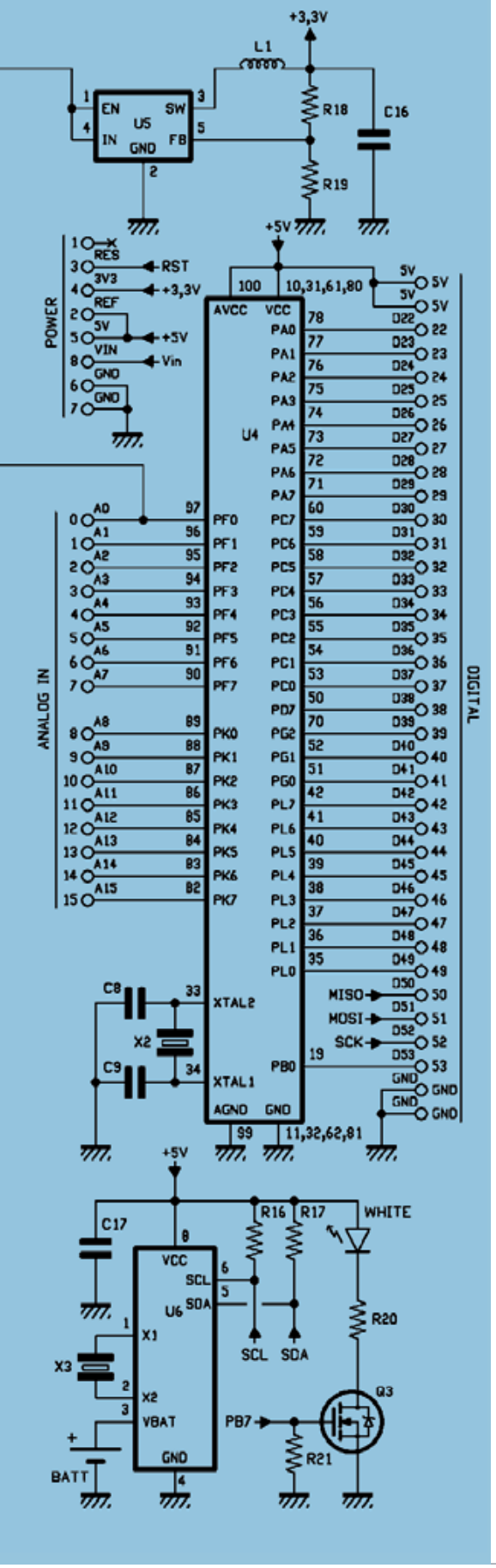


FISHINO DIVENTA MEGA

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


PWRIN
USB
FUSE
D4
D5
D6
D7
Q2
Q1
R11
R12
L2
C13
D8
U3
VIN
SW
EN
GND
FB
C12
L3
C14
C15
R13
R14
R15
PWLED
POWER
LIPO
500MA
R9
R10
U2
PRG
STA
R6
CHARGE
C6
MEAS
+5V
+3,3V
U1
VCC
GND
XO
XI
V3
DTR
TX
RX
C1
C2
C3
X1
C4
R3
R4
R7
R8
RESEN
RESET
D1
D2
D3
R5
R2
R1
C5
ISP
MISO
SCK
MOSI
U4
TXO
RXO
RST
C10
C11
PG5
PH4
PB7
PB4
SCL
SDA
C7
AREF
D0/RX
D1/TX
D2/PWM
D3/PWM
D4/PWM
D5/PWM
D6/PWM
D7/PWM
D8/PWM
D9/PWM
D10/PWM
D11/PWM
D12/PWM
D13/PWM
GND
AREF
SDA
SCL
D14/TXD3
D15/RXD3
D16/TXD2
D17/RXD2
D18/TXD1
D19/RXD1
D20/SDA
D21/SCL
COMMUNICATION - PWM - DIGITAL
ESP
ESPCONN
CHPD
ADC
TX
IO4
IO0
IO2
IO5
RX
RST
IO15
IO16
IO12
IO13
IO14
R34
R33
R35
C18
R22
R23
R24
R26
R28
R29
R30
R27
R25
R31
D10
PB4
D9
R32
SD1
VDD
CD
DO
CMD
CLK
VSS



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:

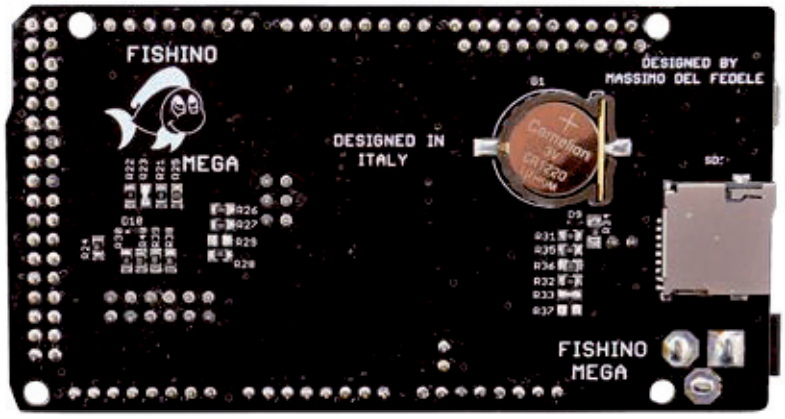
- compatibile al 100% con Arduino MEGA;
- modulo WiFi a bordo, con dotazione completa di librerie da noi sviluppate;
- lettore di microSD a bordo;
- modulo RTC a bordo;
- alimentazione switching integrale, in grado di fornire 800 mA-1A circa (complessivi tra i 3 ed i 5 volt), a partire da una tensione 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.

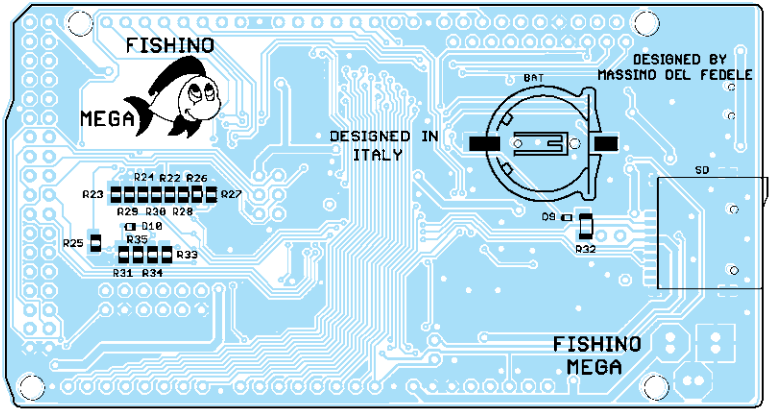
SCHEMA ELETTRICO

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

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 interdicendosi.
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 risultare 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:

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

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

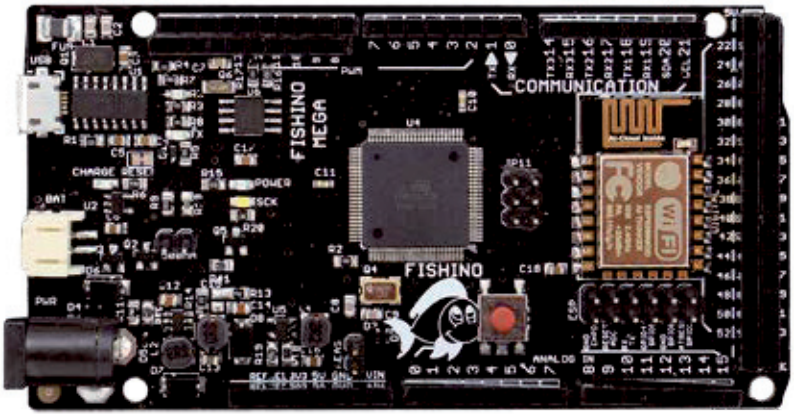
$$E = \frac{V_{OUT} \cdot I_{OUT}}{V_{IN} \cdot I_{IN}} = \frac{5V \cdot 800\,mA}{15V \cdot 306.5\,mA} = 87\%$$

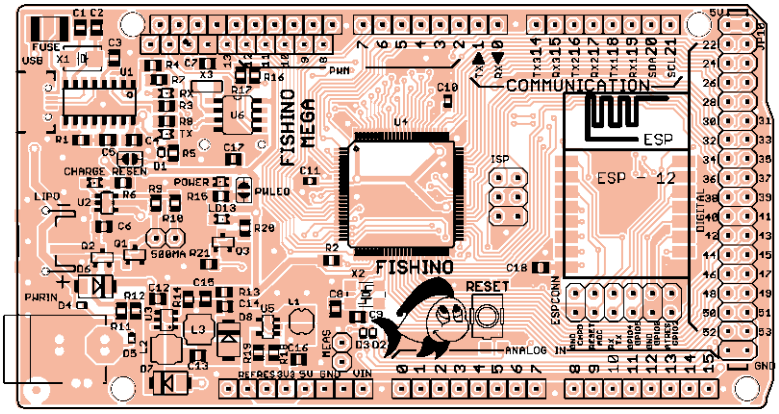
contro il 90% teorico dato dal convertitore. Il diodo causa quindi una perdita di efficienza di un 3%, che possiamo considerare trascurabile. Dalla formula si nota

(0805)
C15: 100 pF ceramico (0805)
C16: 10 µF ceramico (0805)
C17: 100 nF ceramico (0805)
C18: 1 µF 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 µH
L2: 6,8 µH
L3: 6,8 µH
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):

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

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

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

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:

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

che è un'espressione di secondo grado in ILiPo; inserendo i valori e risolvendola, si ottiene:

$$I_{LiPo} = 1280\ mA$$

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

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

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 (RdsON 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 Shottky 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 funzionamento, come si vedrà negli esempi presentati in seguito.

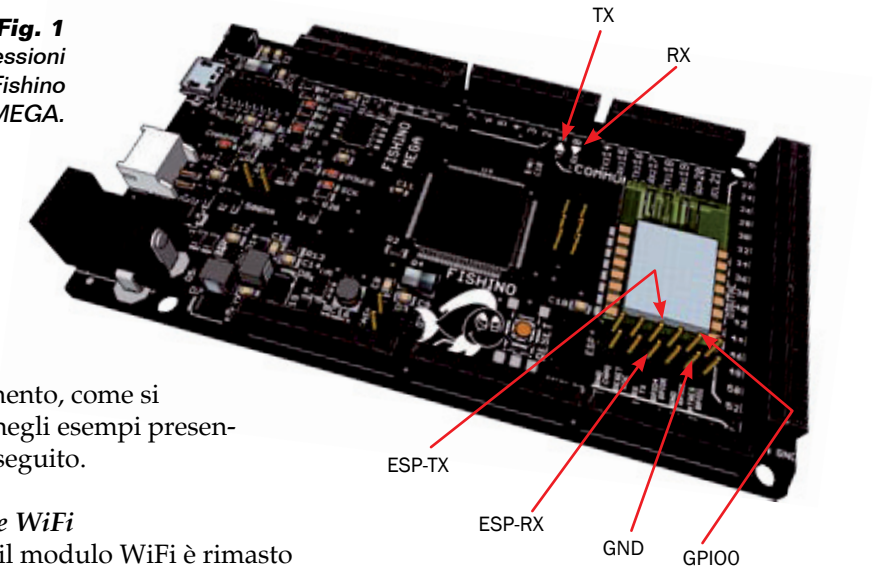


***Fig. 1** Connessioni della Fishino MEGA.*

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 autoresettarsi.

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 µF 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 riprogrammazione 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 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 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

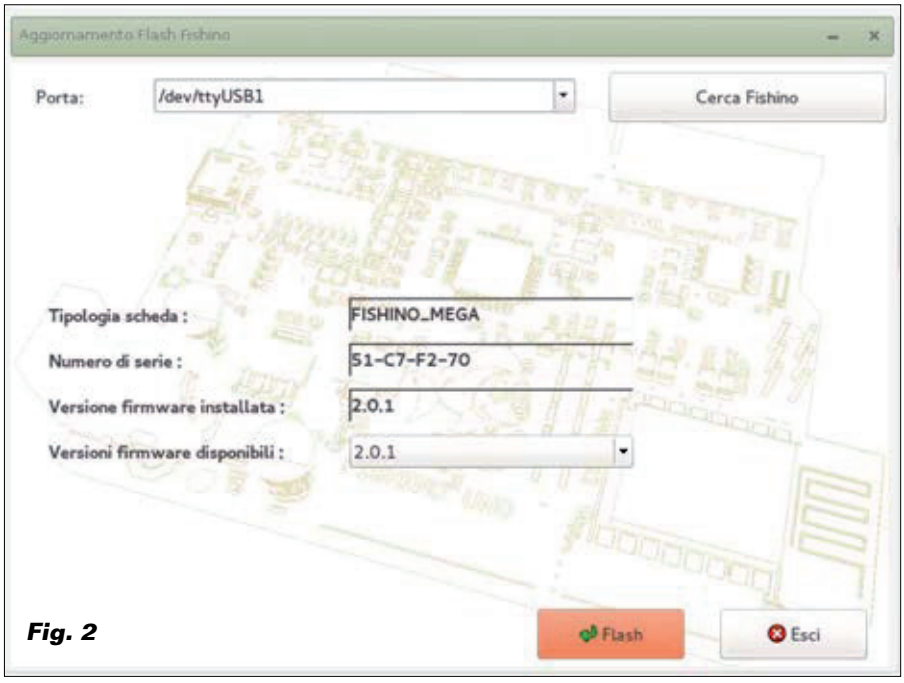
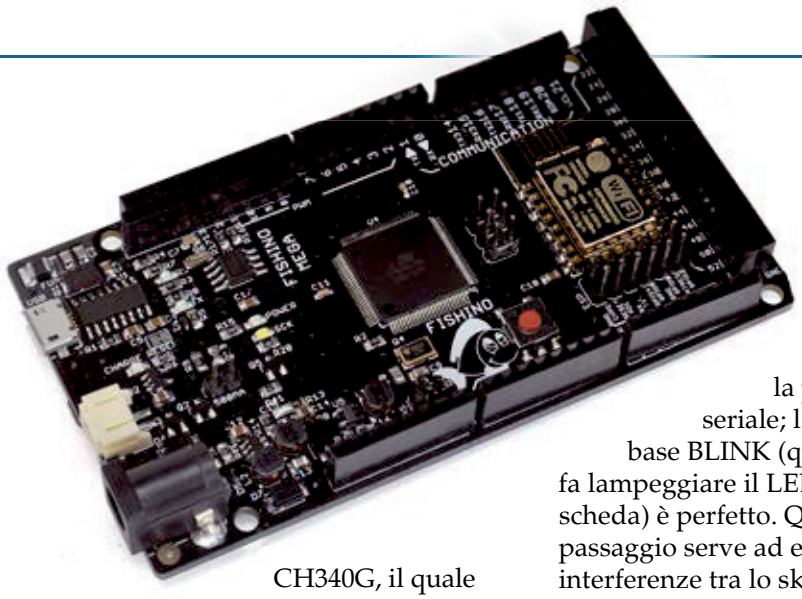


Fig. 2



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 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.

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

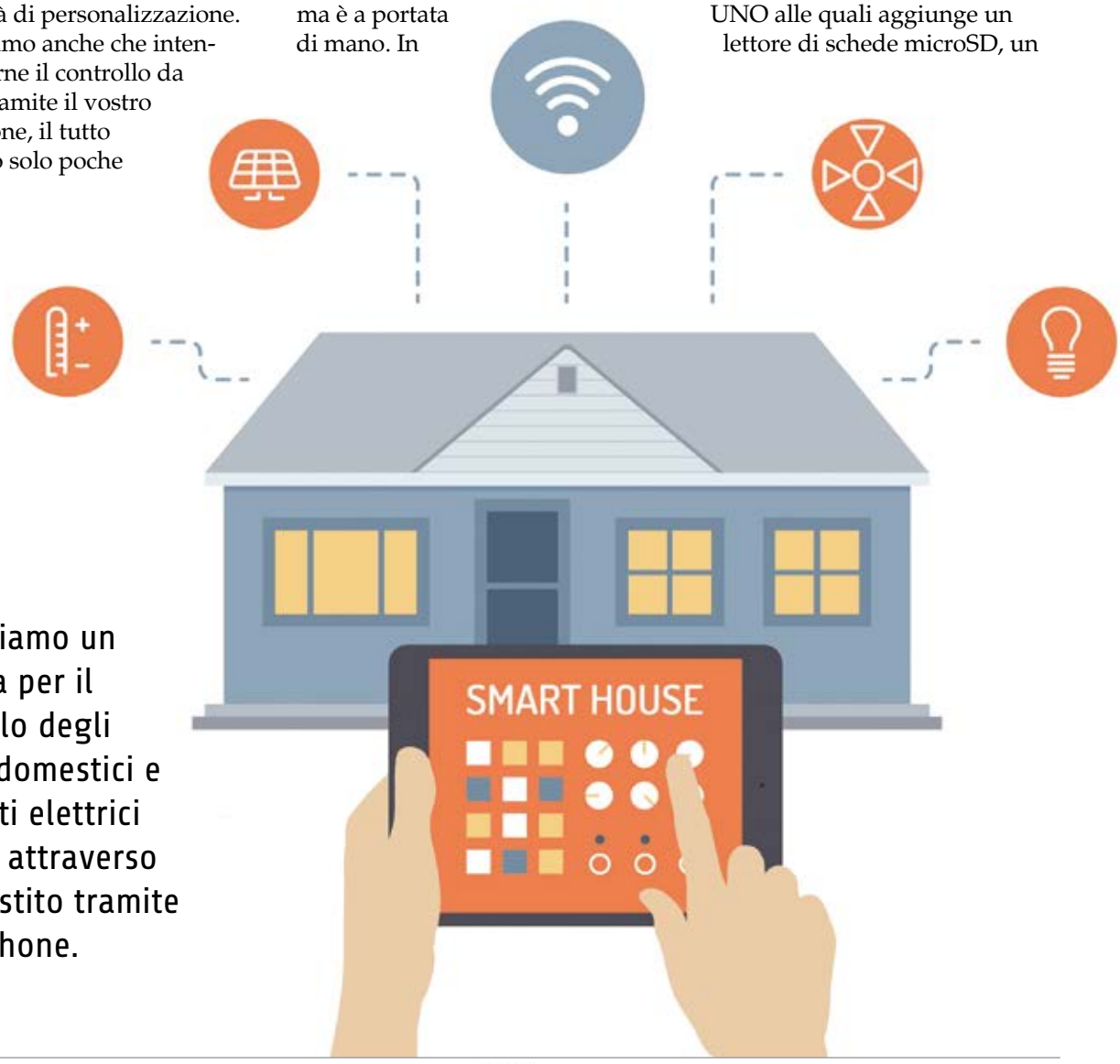
La board Fishino MEGA (cod. FISHINOMEGA) viene fornita montata e collaudata. Può essere acquistata presso Futura Elettronica al prezzo di Euro 49,90. Il prezzo si intende IVA compresa.

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

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.

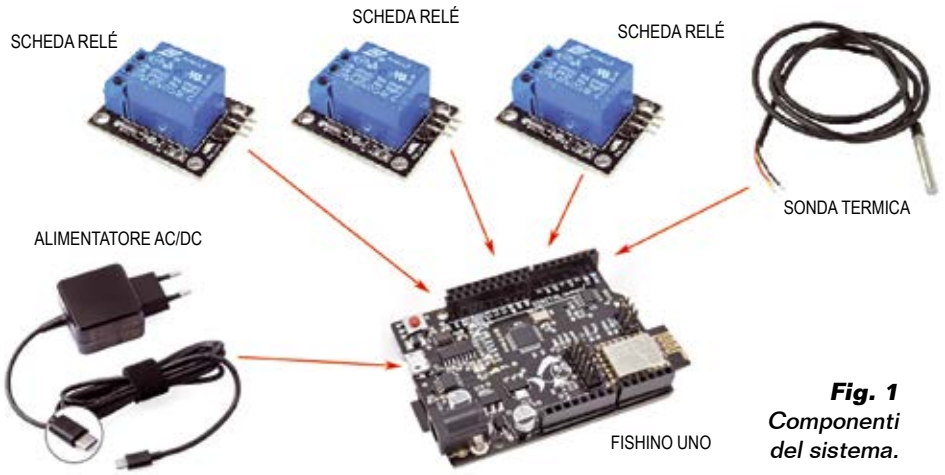


***Fig. 1** Componenti del sistema.*

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 contenente 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'occorrente 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-FISHINOUNO);
- 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

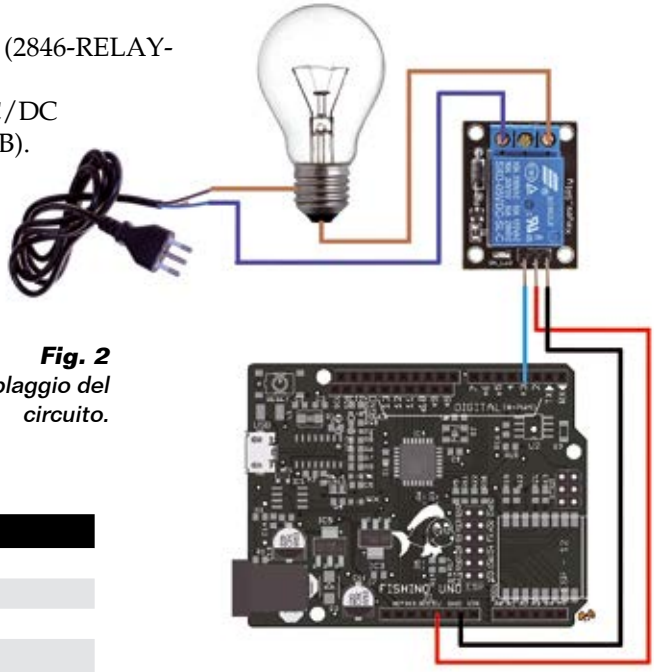


***Fig. 2** Cablaggio del circuito.*

***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.*

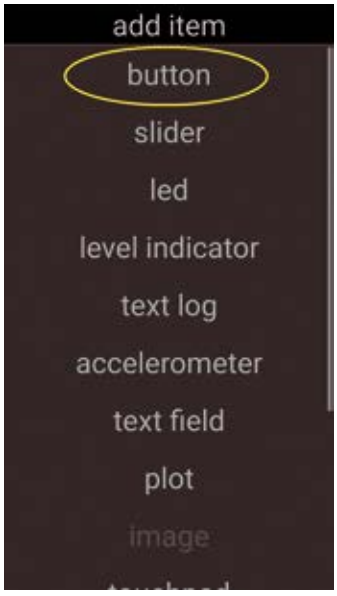


***Fig. 4** - RoboRemo aggiunta di un pulsante.*

***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. Scartiamo 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 trovate 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>inclusione 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.*

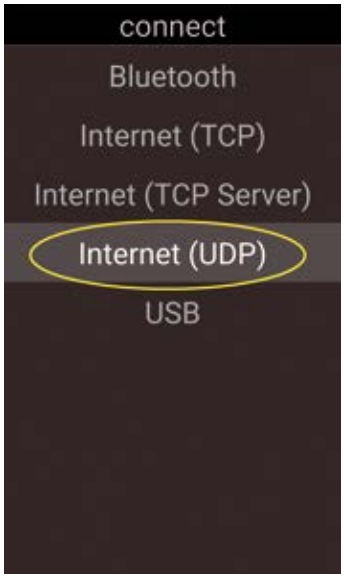


***Fig. 8** - RoboRemo: configurazione comunicazione.*

***Fig. 9** - RoboRemo: impostazione indirizzo IP.*

***Fig. 10** - RoboRemo: comando luci*

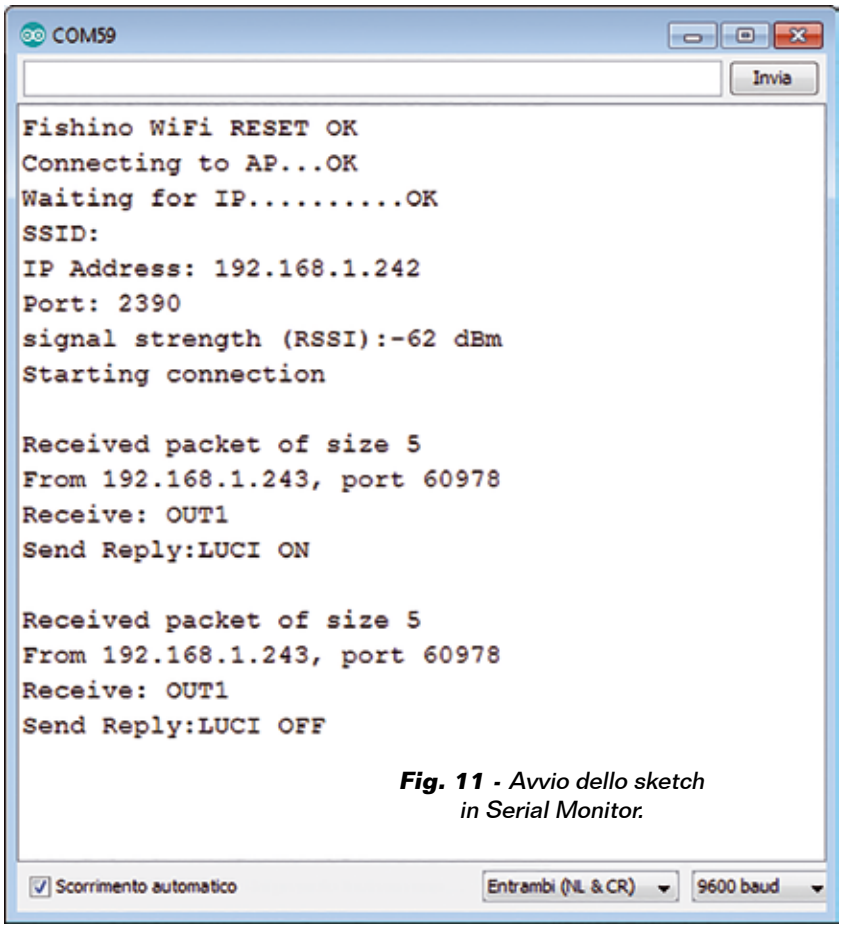


***Fig. 11** - Avvio dello sketch in Serial Monitor.*

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

***Fig. 12** - QR-Code per scaricare **RoboRemo** dal Play Store Android.*

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";
    }
  }
```

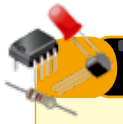
Listato 2

```
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 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.



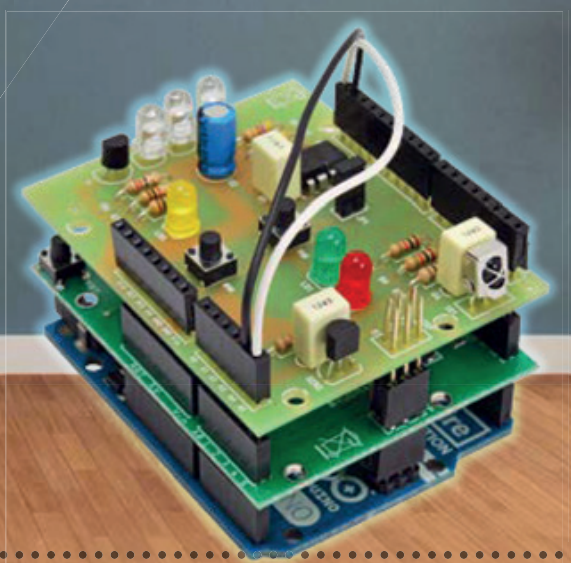
per il MATERIALE

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

Permette di gestire via Bluetooth uno o più utilizzatori elettrici controllabili tramite infrarossi.

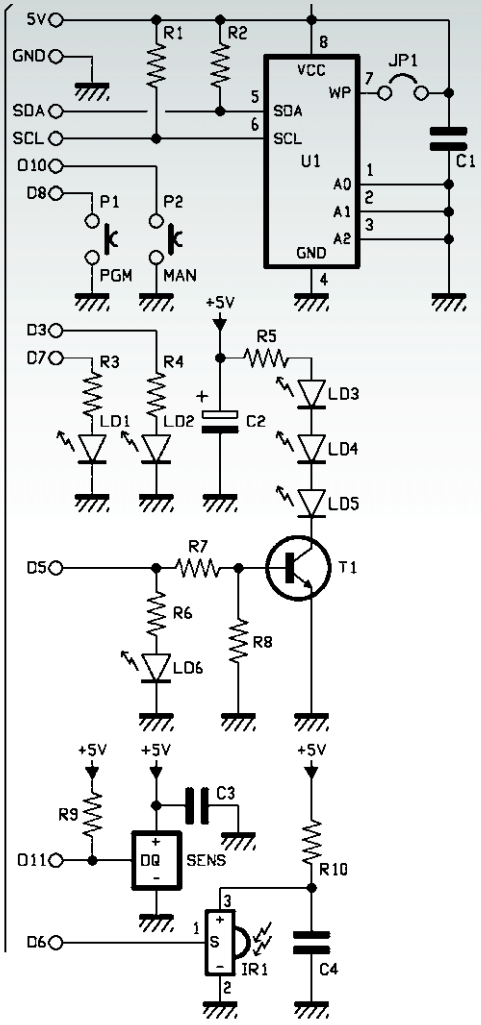
TELECOMANDO BLUETOOTH



Ing. OSVALDO SANDOLETTI

Nel fascicolo 197 avevamo presentato il progetto di un emulatore di telecomandi a raggi infrarossi costituito dalla board RandA (una Arduino con da un lato i connettori per interfacciarsi a Raspberry Pi e dall'altro quelli per gli shield Arduino) ed un apposito shield (ArdIR), che permetteva di controllare gli elettrodomestici di casa riproducendo i segnali emessi dai relativi telecomandi.
La peculiarità di tale progetto era la possibilità di effettuarne il controllo anche a distanza, attraverso Internet, tramite un'apposita interfaccia web accessibile dal browser dei dispositivi che vi si collegavano. Siccome in quel progetto avevamo utilizzato uno shield per Arduino, ci siamo chiesti perché non riutilizzarlo rivedendo il progetto originale in chiave Arduino, ossia togliendo Raspberry Pi e RandA e applicando lo shield direttamente ad una scheda Arduino Uno. Detto...fatto! In queste pagine vedremo come controllare lo shield ArdIR attraverso un collegamento di tipo Bluetooth, dunque a corto raggio; ciò significa che il dispositivo "remoto" *controllante* dovrà operare nello stesso edificio del sistema, ad una distanza non superiore a pochi metri, specialmente se vi sono grossi muri interpo-

[schema **ELETTRICO ArdIR**]



sti fra trasmettitore e ricevitore. Tale requisito appare come una limitazione che ci fa interrogare sul perché scegliere questo sistema. Ebbene, se si ha l'esigenza di gestire dispositivi all'interno delle mura domestiche pur essendo al di fuori della portata ottica dei classici telecomandi IR (ad esempio se ascoltate musica dall'impianto stereo posto in un'altra stanza, e volete poterne controllare il volume, il brano, ecc.), la comunicazione Bluetooth può risultare la più conveniente perché non necessita della disponibilità di una LAN attiva (e completa di router WiFi per collegarsi a dispositivi mobili); il collegamento tra i dispositivi è diretto (*punto-punto*), a patto che ciascuno sia dotato ovviamente di un'interfaccia Bluetooth. Questo è ormai (quasi) sempre vero per tablet e smartphone, che in questo progetto giocano il ruolo di dispositivi *controllanti*; nel dispositivo *controllato*, ovvero il sistema formato da Arduino e dello shield ArdIR, non è così, perché di norma Arduino non è dotata di tale interfaccia. Ci siamo quindi trovati di fronte al dilemma di cercare una versione di Arduino equipaggiata con transceiver Bluetooth o applicare al nostro sistema uno shield dotato di tale connettività. Alla fine abbiamo optato per la seconda, utilizzando uno shield disponibile come kit sul catalogo di Futura Elettronica, siglato FT1032M "Shield Bluetooth con RN-42". Tale shield era stato presentato sul numero 174 di Elettronica In, all'interno del corso su Android (liberamente scaricabile dal nostro sito *www.elettronicain.it*).

Nella **Fig. 1** è riassunto il nostro telecomando Bluetooth: l'utente può utilizzare un qualsiasi dispositivo mobile (dotato di interfaccia Bluetooth) per impartire, tramite un'apposita app che analizzeremo in seguito, i comandi desiderati; questi verranno trasferiti via radio allo shield Bluetooth, rendendoli così disponibili ad Arduino, il quale, tramite l'apposito sketch, gestirà lo shield ArdIR per emettere il segnale a raggi infrarossi verso il dispositivo da controllare.

A livello logico, la **Fig. 2** evidenzia invece "le entità" hardware/software coinvolte nello scambio di dati tra i dispositivi.

Stabilito qual è lo scenario, passiamo ad esaminare i software applicativi (app e sketch) che presiedono al funzionamento delle rispettive parti. Per quanto riguarda la descrizione dettagliata dei due shield (e anche per conoscerne pregi e limiti) rimandiamo ai fascicoli 197 e 174.

L'APPLICAZIONE ARDIRAPP

Questo applicativo è stato sviluppato per sistemi Android, pertanto potrà essere installato sui dispositivi basati su tale piattaforma, a partire dalla versione 4.1 (nome in codice: Jelly Bean).

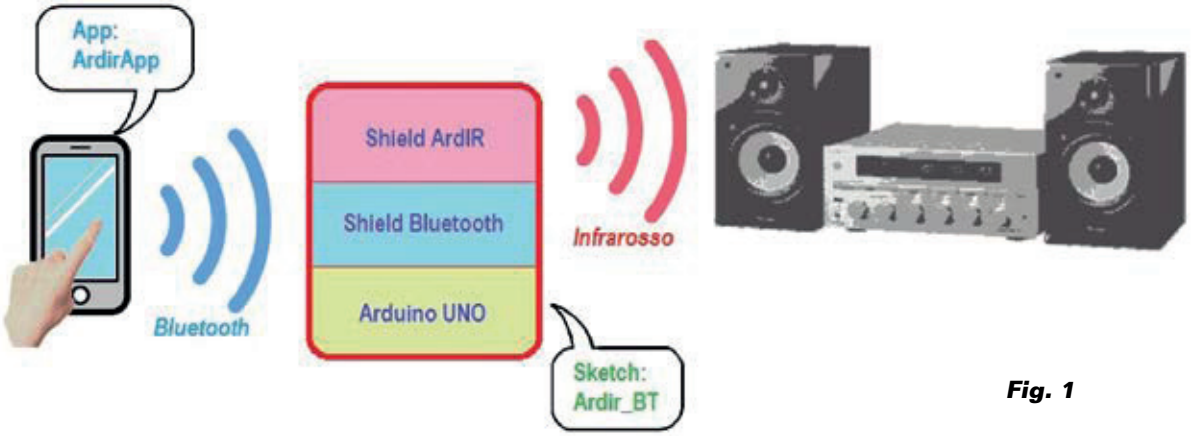


Fig. 1

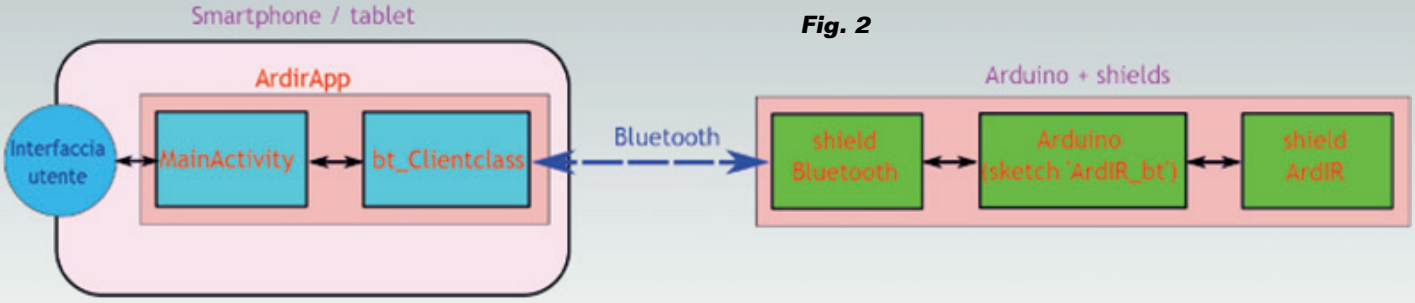


Fig. 2

L'ambiente di sviluppo utilizzato è Android Studio, recentemente sponsorizzato da Google come software ufficiale per lo sviluppo di applicazioni Android in luogo di Eclipse.
Dal punto di vista dell'utilizzatore, la nostra app richiede la gestione di due fasi:

1) ricerca e connessione al dispositivo Bluetooth desiderato, ovvero lo shield FT1032M;
2) selezione del canale da attivare, per la trasmissione del codice verso lo shield FT1032M.

Dal punto di vista del programmatore, per eseguire quanto sopra bisogna strutturare un programma composto (almeno) dalla classe principale, cui sono demandate le attività di partenza (inizializzazione di grafica e risorse) e di gestione degli eventi (callback dei pulsanti dell'interfaccia utente) che abbiamo chiamato "MainActivity". La gestione del Bluetooth è stata invece affidata, per questione di pulizia e manutenibilità del codice, ad una classe separata chiamata "bt_Clientclass".
Il flusso logico del comportamento della "MainActivity" è schematizzato in **Fig. 3**: all'avvio, dopo aver istanziato ed inizializzato le varie risorse, il programma verifica subito la presenza dell'interfaccia Bluetooth; se essa non è abilitata, viene chiesto all'utente di farlo. Se l'utente rifiuta, oppure se tale interfaccia non viene rilevata sul dispositivo che si sta utilizzando, il programma termina. Viceversa, una volta che l'interfaccia è abilitata il programma permette di accedere alla lista dei dispositivi Bluetooth attivi nelle vicinanze, ed è compito dell'utente selezionare quello di suo interesse: in questo caso si tratta del modulo RN42 dello shield Bluetooth facente capo al sistema cui è collegato ArdIR. Fatto ciò, il programma apprende l'indirizzo fisico (MAC) di tale dispositivo e può effettuarne la connessione tramite il metodo connect() dichiarato nella classe "bt_Clientclass".
Una volta instaurato il collegamento con lo shield Bluetooth, la nostra app resta in attesa che l'utente selezioni uno dei canali resi disponibili dall'interfaccia grafica, che dovranno poi venire trasmessi in infrarosso dallo shield ArdIR. Quando ciò avviene, si compone un messaggio formato da una precisa sequenza di dati numerici fra i quali è con-

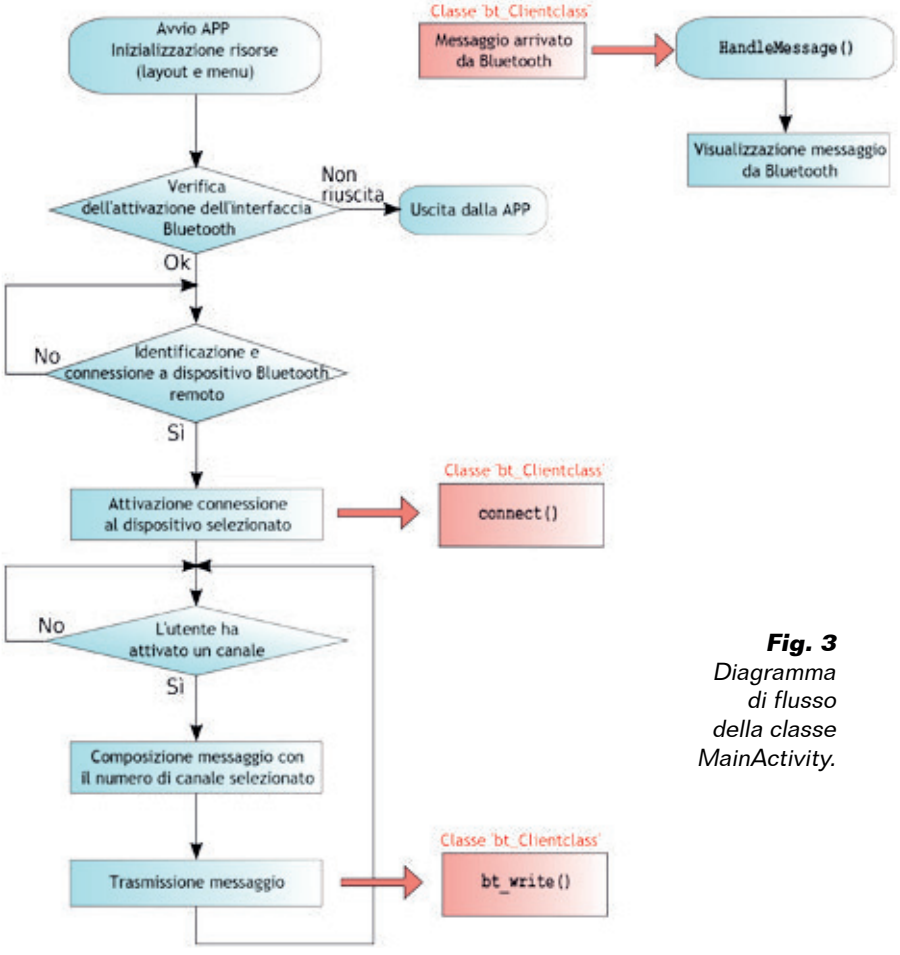
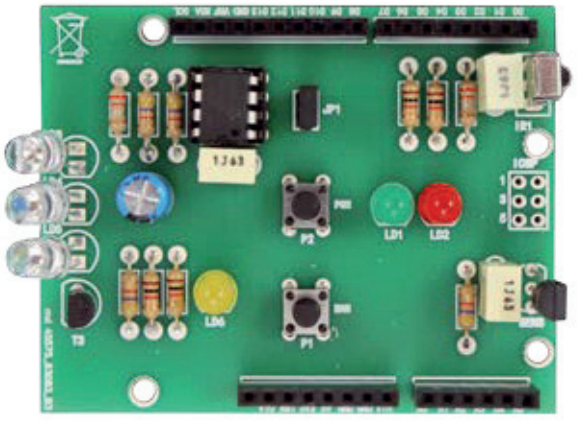
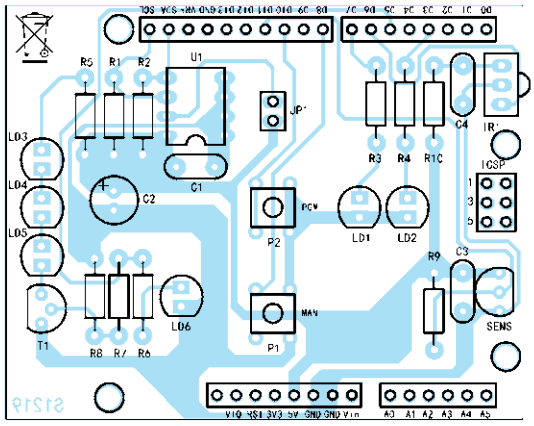


Fig. 3
Diagramma di flusso della classe MainActivity.





Elenco Componenti:

R1: 4,7 kohm
R2: 4,7 kohm
R3: 1 kohm
R4: 1 kohm
R5: 22 ohm
R6: 1 kohm
R7: 1 kohm
R8: 4,7 kohm
R9: 4,7 kohm
R10: 22 ohm
C1: 1 µF 63 VL poliestere
C2: 47 µF 16 VL elettrolitico
C3: 1 µF 63 VL poliestere
C4: 1 µF 63 VL poliestere
LD1: LED verde 5 mm
LD2: LED rosso 5 mm
LD3: LED IR 5 mm
LD4: LED IR 5 mm
LD5: LED IR 5 mm
LD6: LED giallo 5 mm
IR1: IR38DM
SENS: DS18B20+
T1: BC337
U1: 24LC256
P1: Microswitch
P2: Microswitch

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

tenuto anche il codice del canale scelto, e nel contempo si chiama il metodo **write()** della classe "bt_Clientclass", preposto alla trasmissione di tali dati tramite Bluetooth.
Ciò fatto, il programma è già pronto per gestire un successivo comando impartito dall'utente e rimane contestualmente in ascolto di possibili messaggi ritornati dal dispositivo Bluetooth remoto. Nel nostro caso (analogamente a quanto avveniva sulla versione controllata via web) tali dati possono arrivare da Arduino, il quale, dopo aver ricevuto ed elaborato il comando, oltre ad attivare lo shield ArdIR ritrasmette su Bluetooth l'esito dell'operazione attraverso stringhe di caratteri ("OK", "Fail" oppure il valore di temperatura se era stato richiesto). Pertanto queste stringhe saranno ricevute dalla app e visualizzate direttamente sull'interfaccia utente.
Notate che l'operazione di lettura del messaggio è separata dal flusso principale del programma: in tal modo si evita che esso si blocchi nel caso che il messaggio di ritorno non venisse per qualche motivo ricevuto. Più nel dettaglio, utilizziamo un meccanismo di Android previsto per lo scambio di dati tra threads diversi, ovvero la coda di messaggi (MessageQueue): nel caso specifico, nella MainActivity abbiamo istanziato un oggetto di tipo handler, passandone poi il riferimento alla classe "bt_Clientclass" (quando viene creato l'oggetto); quest'ultima lo userà per trasmettere i messaggi con le informazioni ed i dati in arrivo dal Bluetooth, scatenando automaticamente la chiamata del metodo **handleMessage()** dell'handler, e rendendo così il messaggio disponibile alla MainActivity che provvederà infine alla visualizzazione sullo schermo.
Passiamo ad analizzare la funzionalità della classe "bt_Clientclass" che sovrintende alla comunicazione Bluetooth verso Arduino configurando il dispositivo (come dice il nome stesso) come periferica client, mentre il lato Arduino giocherà il ruolo di server (per inciso, l'architettura di base è stata mutuata dal progetto "BluetoothChat", una demo sulla comunicazione Bluetooth disponibile dal sito web *www.developer.android.com*). Lo schema logico

del funzionamento è visibile in **Fig. 4**; essendo una classe "di servizio" all'applicazione principale, ne abbiamo illustrato solo i metodi di interfaccia che fanno riferimento al diagramma di flusso riportato nella **Fig. 3**.
Chiamando dalla MainActivity il metodo **connect()** si instaura la comunicazione Bluetooth che in Android viene effettuata, una volta identificato il dispositivo cui connettersi (ed il relativo indirizzo), in due passi successivi:

1) apertura di un canale di comunicazione (socket) verso il server;
2) connessione al socket.

Nella prima fase si tenta di creare il collegamento al dispositivo relativo all'indirizzo specificato, usando il metodo **createRfcommSocketToServiceRecord()** della classe "android.bluetooth.BluetoothDevice"; tale metodo richiede di passare un identificativo univoco universale (UUID = universally unique identifier), ovvero un numero a 128 bit, univoco per ciascun dispositivo nella rete Bluetooth in cui si opera. Nel nostro caso, per connettersi al modulo RN-42 viene utilizzato un UUID "generico" prestabilito. Come si evince dal nome, il canale che chiediamo di aprire è di tipo RFCOMM (un protocollo di comunicazione che "simula" una porta seriale RS232), su cui si basa il profilo Bluetooth SPP (Serial Port Profile). Questo tipo di protocollo è tra i più comunemente usati per la trasmissione dati tra dispositivi tramite Bluetooth, in quanto viene sfruttato da tante applicazioni che erano state concepite per utilizzare la comunicazione su porta seriale ed è supportato da tutti i principali sistemi operativi su dispositivi mobili e non.
Tornando al nostro programma,

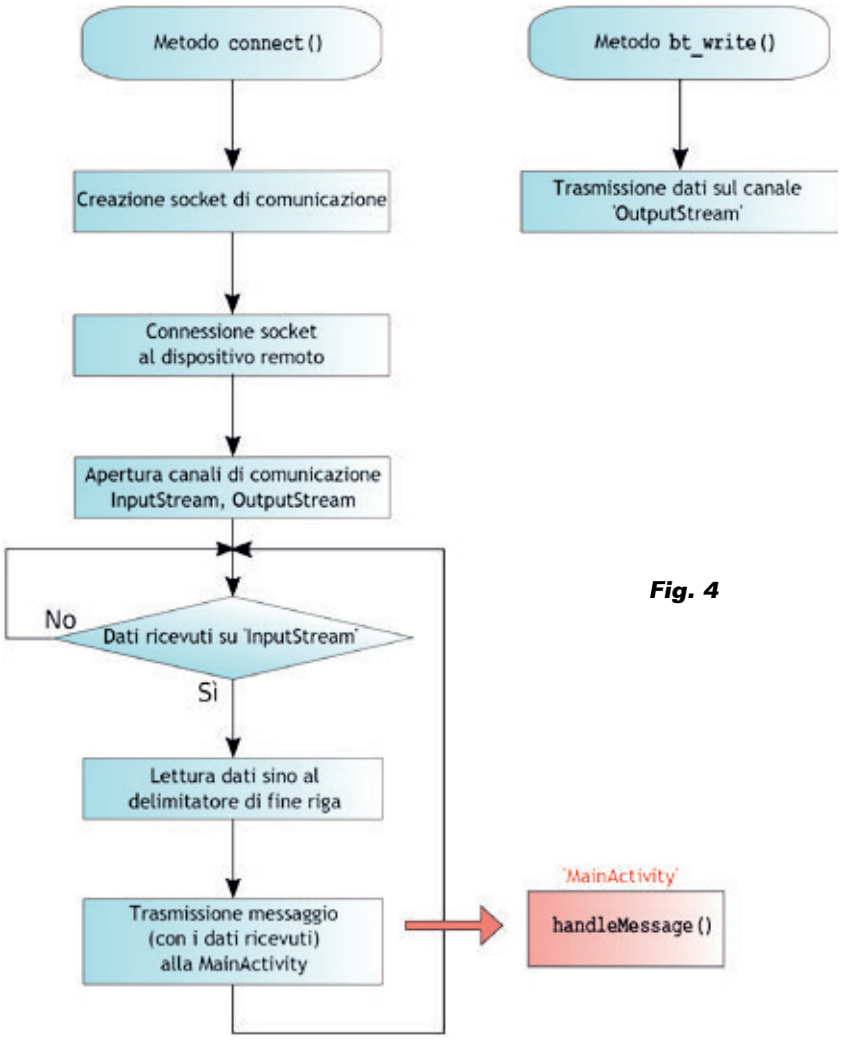


Fig. 4

dopo aver creato e aperto con successo il socket, si invoca il metodo **connect()** per attivare la connessione al server; se anche questo ha successo, la comunicazione è stabilita e pronta per effettuare uno scambio di dati tra le parti. Per far ciò, si associano al socket due oggetti di tipo "InputStream" e "OutputStream" sui quali si lavorerà per leggere e scrivere i dati.
Dando uno sguardo al codice, si nota che queste procedure vengono eseguite all'interno di due classi annidate, chiamate "ConnectThread" e "ConnectedThread". Queste due, che estendono la classe Thread, sono state utilizzate appunto per gestire le fasi di connessione e di comunicazione all'interno dei due rispettivi threads, la cui esecuzione rimane slegata da quella del programma principale: in tal modo si evita che questo si blocchi in caso di problemi di connessione o di comunicazione col server Bluetooth. Per tale motivo inoltre, come si nota sempre nel diagramma di flusso di **Fig. 4**, terminata la fase di connessione il codice rimane "ibernato" in attesa di ricevere dati dal Bluetooth, sull'oggetto "InputStream"; essendo eseguito all'interno di un thread, questo non pregiudica il funzionamento del programma principale. Quindi, possiamo dire che la logica di ricezione dati viene gestita da un processo che gira in background, in modo asincrono e indipendente dall'applicazione principale. Quando poi il server (gestito da

Aggiungere canali al **telecomando**

Se nella "versione web" bastava editare una pagina HTML, per modificare il layout e il numero di canali nel programma "ArdirApp" è necessario modificarne i sorgenti del progetto, ricompilarli ed installare nuovamente la app sul nostro dispositivo, pertanto è un'attività consigliabile solo da chi già mastica Android come programmatore o che comunque abbia almeno avuto un'infarinatura su questa piattaforma, rudimenti di Java compresi, ad esempio leggendosi il corso che era stato presentato sulla rivista a partire dal numero 169, e liberamente scaricabile dal sito di Elettronica In (sul quale viene presentato anche lo shield Bluetooth qui utilizzato).
Il progetto completo è scaricabile direttamente dal sito di Elettronica In, e dev'essere scompattato ed importato nell'ambiente di sviluppo Android Studio. Il file di descrizione del layout si chiama "activity_main.xml" e si trova nella sottocartella res/layout: per modificarlo si può agire direttamente sul suo contenuto testuale (in linguaggio XML) oppure graficamente sulla vista dell'anteprima. In ogni caso, a ciascun pulsante creato bisognerà associarvi univoci attributi id e onClick, proseguendo con la numerazione progressiva già utilizzata (quindi ad es. gli attributi onClick dovranno essere: cbP5, cbP6, cbP7, ...). Naturalmente, la disposizione grafica e l'impaginazione dei tasti è lasciata alla libertà del programmatore: chi decidesse di utilizzare un tablet, in virtù dell'ampio schermo disponibile, potrebbe disegnare un completo telecomando virtuale, magari con comandi associati a diversi dispositivi: spazio alla fantasia!
In ogni caso, una volta inseriti tutti i canali desiderati, bisogna implementare i rispettivi metodi di "callback" (specificati con l'attributo onClick) nella classe MainActivity; di fatto si può fare un copia-incolla dei metodi già esistenti cambiandone solo il nome. Il corpo della funzione infatti estrae proprio dal nome il numero di canale e lo usa, nella funzione TxBtChan(), per costruire il vettore out_buf[] da trasmettere ad Arduino.
Nella MainActivity bisogna anche specificare il numero di canali utilizzati in totale nella costante di tipo intero "NCHANNELS", dichiarata subito all'inizio del corpo della classe: il numero attuale è 5 perché tanti sono i canali in uso nel programma di esempio (ricordiamo anche che la memoria EEPROM di ArdIR consente la memorizzazione di un massimo di 127 canali). Infine, si può ricompilare il progetto e re-installare sul dispositivo il nuovo file ".apk" così creato.
Lo stesso numero "NCHANNELS" visto sopra dev'essere poi anche specificato all'interno dello sketch "ArdIR_BT", nella riga apposita:

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

Dopo la modifica, lo sketch andrà pertanto ricaricato su Arduino (ricordatevi di togliere gli shield prima di farlo).

Arduino) trasmetterà la stringa di dati, essa verrà ricomposta dal codice sino a ricevere il carattere di fine linea. A questo punto il nostro processo utilizza l'handler fornitogli dalla MainActivity per trasmettergli la stringa ricevuta, nel seguente modo:

- crea un messaggio chiamando **obtainMessage()** e passandovi la stringa di dati;
- dall'istanza del messaggio appena creato, chiama il metodo **SendToTarget()**; questo scatenerà, come anzidetto, l'attivazione della "controparte" (il metodo **handleMessage()**) dell'handler, che essendo implementato nella MainActivity rende i dati disponibili in tale classe.

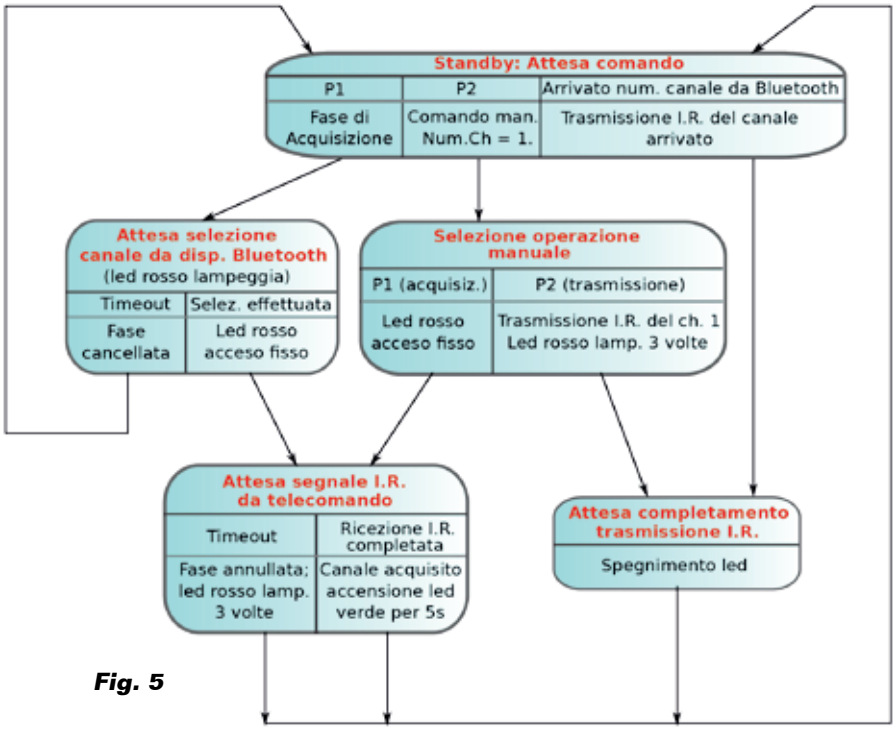


Fig. 5

Invece, per trasmettere i dati verso Arduino, la MainActivity deve semplicemente invocare il metodo **bt_write()**, passando come argomento il vettore di dati che si vuole inviare; questi verrà poi trasmesso alla periferica Bluetooth tramite il metodo **write()** dell'oggetto "OutputStream" precedentemente inizializzato.

SKETCH ARDIR_BT

Dal lato Arduino ritroviamo, come accennato, lo shield ArdIR, che viene gestito attraverso un firmware in larga parte derivato dallo sketch *ArdIR.ino*, spiegato anch'esso nell'articolo relativo allo shield anzidetto; per questa ragione qui focalizzeremo la nostra attenzione sulle modifiche fatte rispetto a tale sketch.

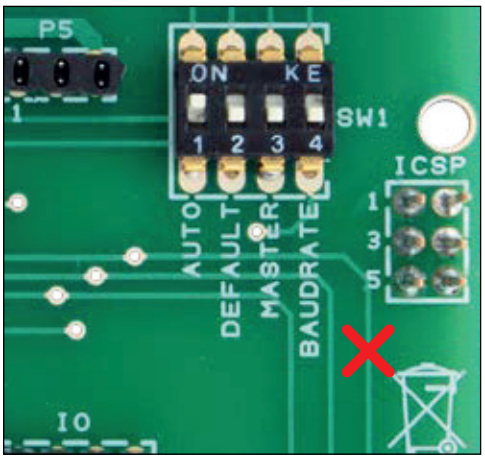


Fig. 6 *- Impostazione dei dip-switch e pista da tagliare sullo shield bluetooth.*

Questa volta, la seriale di Arduino è collegata al modulo RN42 dello shield Bluetooth, pertanto, bisogna gestire coerentemente la comunicazione con quest'ultimo, la quale prevede una velocità di comunicazione di 115.200 Baud, nonché un'eventuale fase di inizializzazione. Per tale motivo, nella funzione **setup()** dello shield viene opportunamente inizializzata l'interfaccia seriale e trasmesse quindi le informazioni "minimali" all'RN42 necessarie per poterlo fare lavorare correttamente, ovvero impostandolo nel modo SM0 (si veda il manuale del modulo per ulteriori dettagli) in modo tale che giochi il ruolo di Server nella comunicazione verso i dispositivi Bluetooth.
Anche dal punto di vista funzionale, il programma ricalca perfettamente il comportamento dell'originale "ArdIR.ino"; ne riportiamo il diagramma degli stati in **Fig. 5**.
Riassumiamo brevemente qual è la logica di funzionamento: si parte dallo stato di attesa, in cui entrambi i LED dello shield rimangono spenti, ove si attende il verificarsi di un'azione che può essere:

- pressione del tasto P1 dello shield, nel qual caso si attiva la fase di acquisizione del codice di un canale del telecomando; in tal modo si effettua l'associazione tra il tasto (il canale) attivato dalla app ed il codice infrarosso che si vuole trasmettere al ricevitore (televisore, ecc.); più avanti descriveremo le operazioni da compiere in pratica;
- pressione del tasto P2 dello shield; si gestisce l'acquisizione e la trasmissione del primo canale "in locale", ovvero senza l'ausilio del dispositivo Bluetooth remoto;
- ricezione del numero di un canale dal modulo Bluetooth (e quindi da uno smartphone/tablet dotato dell'app discussa sopra); questa è la fase di funzionamento "normale" del sistema, in questo caso viene attivata la trasmissione a infrarossi del codice associato al canale del telecomando precedentemente acquisito.

DALLA TEORIA ALLA PRATICA

Passiamo adesso alla preparazione e messa in funzione del nostro sistema, per la realizzazione del quale sono necessari:

- scheda Arduino Uno o equivalente (ad esempio Fishino Uno);
- shield ArdIR (FT1219K);
- shield Bluetooth (FT1032M);
- smartphone oppure Tablet con sistema operativo Android (dalla versione 4.1 in poi), ovviamente dotato di interfaccia Bluetooth.

Prima di collegare Arduino agli shield bisogna caricarvi lo sketch, in quanto tale operazione viene eseguita attraverso le linee della porta seriale hardware (D0 e D1), che sono però le stesse utilizzate dallo shield Bluetooth: se questo fosse già inserito, vi sarebbe quindi un conflitto che renderebbe impossibile la programmazione. Sullo shield Bluetooth bisogna anche

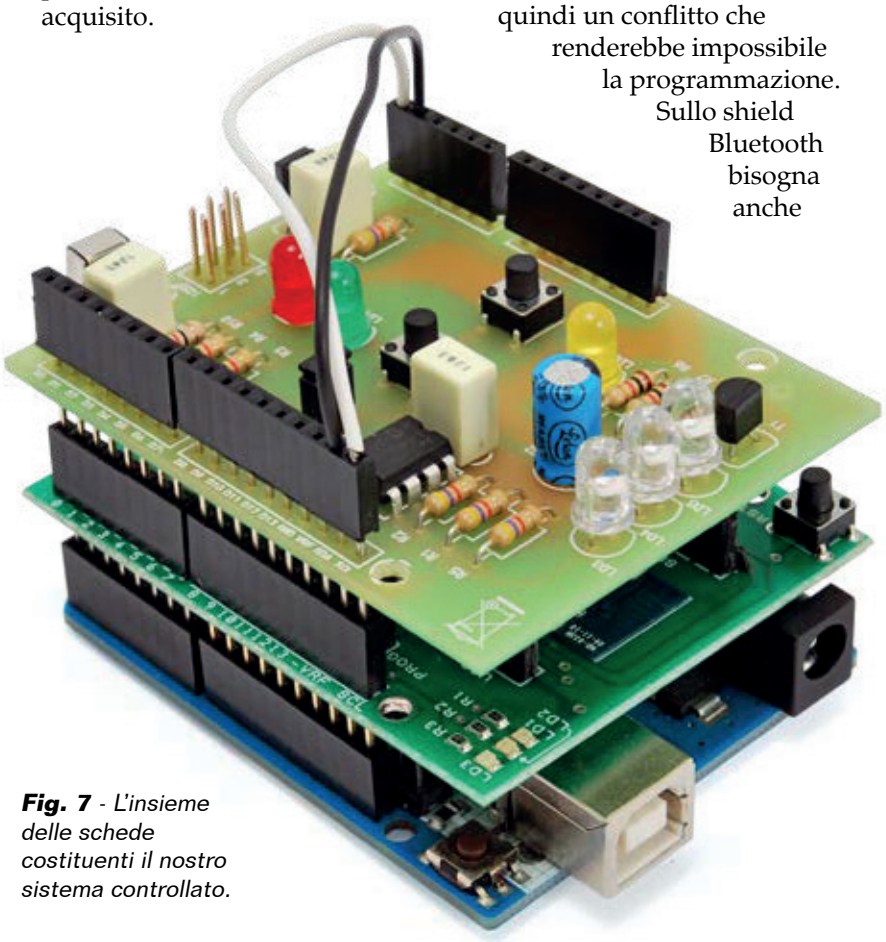


Fig. 7 *- L'insieme delle schede costituenti il nostro sistema controllato.*

La multimedialità del **televisore**

Una delle più utili interfacce di cui sono dotati i moderni televisori è sicuramente la porta HDMI (High-Definition Multimedia Interface): pensata principalmente per accettare dati digitali audio/video e quindi utilizzare il televisore come una periferica di visualizzazione per i dispositivi dotati di tale interfaccia (PC, fotocamere, ecc.), nel tempo si è "evoluta" dotandosi di funzionalità aggiuntive, per fornire l'intercomunicabilità tra i dispositivi collegati e anche la connettività su rete Ethernet (ad esempio, tramite tale interfaccia si può cablare un intero sistema Home-Theatre (http://it.wikihow.com/Collegare-un-Cavo-HDMI). Contestualmente, sono stati sviluppati diversi gadget da collegare a tale porta per sfruttarne le potenzialità. Uno di essi è il Google Chromecast (www.google.com/intl/it_it/chromecast), un piccolo dispositivo del costo di poche decine di euro che consente di collegare al televisore, tramite Wi-Fi, PC e dispositivi mobili su piattaforma Windows, Android e Mac OS: in tal modo si può sfruttare lo schermo TV per "estendere" il monitor del telefonino/tablet/PC e vedersi i video preferiti in streaming, anche in HD (in base a quanto enunciato si dovrebbe arrivare, nel migliore dei casi, a 1080p). Ma non è tutto: Google mette a disposizione il Google Cast Software Development Kit (https://developers.google.com/cast/) per consentire agli sviluppatori di App una rapida integrazione tra i propri programmi e Chromecast. Ulteriori potenzialità dell'utilizzo di questo strumento sono descritte nel sito web www.navigaweb.net/2015/01/5-modi-di-usare-il-chromecast-che.html, su cui si trovano anche informazioni sulle funzionalità "HDMI-CEC" (HDMI Consumer Electronics Control) che consentono alla porta HDMI, se il televisore è predisposto, di assumerne un controllo più completo (es. accendere/spegnere, controllare il volume,...) oppure, viceversa, di consentire al televisore stesso di controllare, tramite il relativo telecomando (oppure ArdIR!) dispositivi connessi sempre sulla porta HDMI (ad esempio

***Fig. 9** - Una versione di Chromecast.*

un lettore Blu-Ray o anche un sistema Home-Theatre).
Persino la Apple ha provato ad inserirsi in questo filone, proponendo la sua "Apple TV": un dispositivo collegato al televisore (sempre sulla porta HDMI) ed interfacciato ad una rete locale (tramite cavo o Wi-Fi). Se tale rete è connessa ad Internet, si potrà accedere dal TV ai contenuti di iTunes, altrimenti si potrà comunque utilizzarlo trasferendoli da iPAD o iPhone (almeno 4S) tramite il protocollo "AirPlay" di Apple. Malgrado la "Apple TV" non abbia ad oggi riscosso un grande successo (anche se la prima generazione risale al 2007, ed in ottobre 2015 è uscita la quarta), Apple continua a crederci: lo dimostra la presenza di un "App store" con applicazioni dedicate (soprattutto giochi, probabilmente per insidiare la concorrenza più agguerrita nel settore delle consolle...) e lo sviluppo di un proprio sistema operativo (tvOS) per gestirla. Insieme al dispositivo viene inoltre fornito un telecomando dedicato, che nell'ultima versione è completo di superficie tattile per la gestione dei comandi tramite lo sfioramento delle dita, come ormai ci hanno abituato a fare da tempo i nostri telefonini.

***Fig. 10** - Apple TV di quarta generazione.*

scollegare, magari incidendola con un taglierino, la pista che va dal segnale A4 del connettore Arduino, al piedino PIO9 del modulo RN42 (vedere **Fig. 6**), in quanto interferisce con il bus I²C utilizzato da Arduino per interfacciarsi alla EEPROM dello shield ArdIR. Nella stessa figura si nota anche l'impostazione dei quattro dip-switch: "AUTO" dev'essere posizionato su ON, mentre gli altri rimangono su OFF.
Per quanto riguarda lo shield ArdIR, se siete in possesso di una scheda Arduino Uno revisione 1 o 2, dovrete collegare tramite due fili "volanti" i segnali A4 e A5 con SDA e SCL rispettivamente. La **Fig. 7** mostra il sistema completo di questi "ponticelli" sullo shield ArduIR. Notate, infine, che lo shield Bluetooth va montato sotto ArdIR.
Una volta "assemblato" il sistema, si può alimentare Arduino tramite la presa USB oppure un alimentatore esterno. Immediatamente si vedrà il LED LD1 (su ArdIR) lampeggiare tre volte ad indicazione che il sistema è pronto per l'utilizzo. Inoltre, sullo shield Bluetooth, dovrete osservare il lampeggio continuo

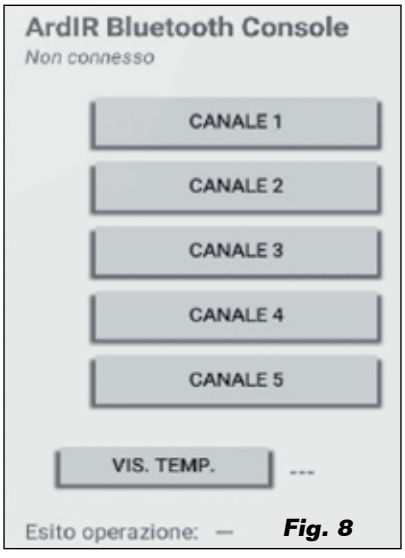


Fig. 8

a circa 1 Hz del led rosso (pure lui si chiama LD1) ad indicare che è in attesa di essere rilevato da un altro dispositivo Bluetooth.
Installiamo ed attiviamo dunque anche la "ArdirApp" sul nostro smartphone Android preferito; all'avvio vi sarà richiesto, se non ancora fatto, di attivare la periferica Bluetooth del dispositivo: naturalmente confermate.
Compare quindi la schermata visibile in **Fig. 8**, ove la scritta "Non connesso" sotto al titolo indica che dobbiamo ancora stabilire il collegamento con il modulo su Arduino. Entriamo allora nel menu (icona della lente d'ingrandimento a fianco del titolo) che mostra l'elenco di tutti i dispositivi che già erano stati "accoppiati" in passato. La prima volta, quindi, non vi sarà il nostro, per cui agiamo sul comando di "Scansione dispositivi"; dovrebbe così comparire anche la voce relativa allo shield Bluetooth, riportandone il nome e l'indirizzo "MAC" del modulo RN-42, ad esempio:

```
FireFly-E6CB
00:06:66:4F:E6:CB
```

dove "FireFly" è il nome locale assegnato da Microchip ai suoi moduli Bluetooth di derivazione Roving Networks (volendo, il nome del modulo si può anche modificare; rimandiamo però chi fosse interessato a farlo, a consultare l'apposita documentazione del modulo).
Selezionando tale voce sul menu, il programma tenta di stabilire la connessione (accoppiamento o pairing) con il modulo; la prima volta verrà chiesta l'autenticazione tramite passkey che per default è "1234". Se l'operazione va a buon fine, sotto il titolo deve apparire la scritta "Connesso a (nome del modulo)".
Possiamo allora procedere con l'apprendimento di un codice del telecomando, ad esempio il tasto di azzeramento volume del televisore, che assegneremo al Canale 1 sulla nostra app.
Nota: se la fase di apprendimento dello shield ArdIR era già precedentemente stata effettuata, magari anche tramite pagina web (utilizzando RandA), tale procedura non è più necessaria in quanto i codici acquisiti dal telecomando vengono scritti nella memoria eeprom di ArdIR e sono dunque salvati in modo permanente (salvo che non si voglia riscriverli).
Dopo esserci assicurati che il ponticello JP1 sullo shield ArdIR NON sia inserito (altrimenti la EEPROM rimane protetta in scrittura), premiamo P1 per entrare in modalità di acquisizione da telecomando: il LED rosso LD1 di ArdIR lampeggia, segnalando che è in attesa dell'azionamento di uno dei canali sulla videata della app; selezioniamo il Canale 1. Il LED si accende fisso a segnalare che bisogna attivare il tastino di azzeramento volume sul telecomando (dopo averlo "puntato" verso il ricevitore a infrarossi IR1 a circa 10 cm di distanza) per trasmettere il codice da acquisire. Ciò fatto, dovrebbe accendersi il LED verde per cinque secondi a segnalare il corretto apprendimento del nuovo codice. Possiamo quindi testarlo, puntando i LED infrarossi di ArdIR verso il televisore e sperimentando che, attivando il tasto "Canale 1" sullo smartphone, si ammutolisca il volume del televisore.
Si può dunque procedere con la memorizzazione di altri comandi su altrettanti canali che, come già suggerito nella versione su pagine HTML, si possono aumentare a piacere (compatibilmente con le capacità di visualizzazione dello schermo del telefonino), come discusso nell'apposito riquadro in questo stesso articolo.
Anche stavolta abbiamo la possibilità di interrogare il sensore DS18B20 montato su ArdIR per leggere la temperatura ambiente, cliccando sul tasto "Vis. Temp". Ricordiamo che, alla prima interrogazione, la temperatura restituita sarà di 85°C (valore predefinito di lettura del sensore DS18B20) in quanto non saranno stati precedentemente eseguiti comandi di conversione.

CONCLUSIONI

Questo articolo descrive come aggiungere la connettività Bluetooth allo shield ArdIR, utilizzando un altro shield già presentato sulla rivista ed una "normale" scheda Arduino Uno. In questo modo, utilizzando un telefonino o tablet, si può gestire il controllo di apparecchi domestici anche al di fuori della portata ottica dei normali telecomandi a raggi infrarossi, e senza la necessità di avere a disposizione una rete WiFi attiva. ■

per il MATERIALE

La board e gli shield utilizzati in questo progetto sono disponibili presso Futura Elettronica. La board Arduino Uno REV3 (cod. ARDUINOUNOREV3) è in vendita al prezzo di Euro 24,50. L'Ardir Shield (cod. FT1219K) costa Euro 18,00 mentre lo Shield bluetooth con RN-42 (cod. FT1032) si può acquistare a Euro 34,00. Tutti i prezzi si intendono IVA compresa.

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