Ultima nata della serie, conserva l'architettura a 32 bit, rimanendo in dimensioni molto ridotte: una scheda piccola, ma potente!

PIRANHA: LA FISHINO AGGRESSIVA!

di MASSIMO DEL FEDELE

Poco più di un anno fa abbiamo presentato la nostra prima scheda Arduino-like a 32 bit, chiamata Fishino32 (fascicoli n° 209 e 210) e caratterizzata dal formato della diffusissima Arduino UNO (ma anche della Fishino UNO), però equipaggiata con un microcontrollore a 32 bit capace di prestazioni eccezionali, se paragonate alla media delle schede di prototipazione del genere. Da parecchio tempo avevamo pronta anche la sua "sorellina minore", della quale abbiamo posticipato la pubblicazione per motivi di tempo e di sviluppo delle librerie a 32 bit (che si sono rivelate piuttosto impegnative...) ma ora è giunto il momento di metterla sotto ai riflettori: la scheda è la piccola "Piranha", un nome che è tutto un programma ed evidenzia la connotazione aggressiva, sul piano delle prestazioni. La scheda dispone infatti dello stesso microcontroller della

CARATTERISTICHE TECNICHE

- **Architettura a 32 bit**
- **Processore PIC32MX**
- **Clock a 120 MHz**
- **Formato Arduino MKR1000**
- **3 vie di alimentazione**
- **WiFi integrato**
- **USB integrato**
- **Supporto SD-Card**

Fishino32, ossia un PIC32MX con core MIPS, 128 kB di RAM e 512 kB di Flash, oltre ad un clock di ben 120 MHz, ma con le stesse dimensioni dell'Arduino MKR1000, ovvero con fattore di forma di poco superiore a quello della Arduino Nano e, come lunghezza, inferiore alla Fishino GUPPY, pur disponendo di un'invidiabile dotazione di periferiche incorporate. Abbiamo quindi a che fare con una scheda dotata di una potenza di calcolo notevole, però collocabile, in virtù del ridotto ingombro, quasi ovunque.
Facciamo dunque una panoramica d'apertura sulle caratteristiche della nostra Fishino Piranha:

- processore PIC32MX470F512;
- 512 kB di ROM;
- 128 kB di RAM;
- clock a 120 MHz;
- interfaccia USB nativa, sia device che host;
- RTC on board;
- lettore per schede microSD;
- transceiver WiFi on board;
- alimentazione a batteria, via USB ed esterna (plug) da 3 a 20 V;
- caricabatterie per LiPo 3,7V attivo se presente l'alimentazione esterna;
- possibilità di spegnere da software la scheda, lasciando alimentato solo il processore in standby;
- risveglio dallo standby tramite segnale esterno o a tempo;
- funzionamento logica interna a 3,3 V;
- pin digitali 5V-tolerant, quindi nessun rischio di danneggiare la scheda con periferiche a 5 V.

Come potete notare, l'unica cosa mancante è il codec audio, che non è stato previsto per motivi di spazio. Come per la Fishino32, la programmazione di questa scheda può essere eseguita sia tramite l'IDE di Arduino, sia mediante il nostro FishIDE; inoltre per i più esperti è possibile programmare la Piranha tramite i sistemi di sviluppo della Microchip (MPLAB IDE e PicKit3) che permettono anche di eseguire il debugging passo-passo del codice.
Grazie ad un buon numero di librerie software da noi scritte e/o adattate, già ben collaudate con la Fishino32, la scheda si programma come una normalissima Arduino UNO, tramite l'IDE e con la sintassi cui siamo abituati; basta eseguire una piccola procedura nell'IDE stesso, che permette di attivare gli strumenti di sviluppo per microcontrollori PIC, ed il gioco è fatto!
Per i componenti aggiuntivi abbiamo predisposto una serie di librerie, in continuo aggiornamento, che permetteranno di sfruttarli appieno.
Diamo adesso uno sguardo più attento all'hardware, riferendoci allo schema elettrico e analizzando i singoli blocchi che lo compongono. Lo stadio controller, il modulo WiFi e l'interfaccia per schede microSD sono identici a quelli della Fishino32.

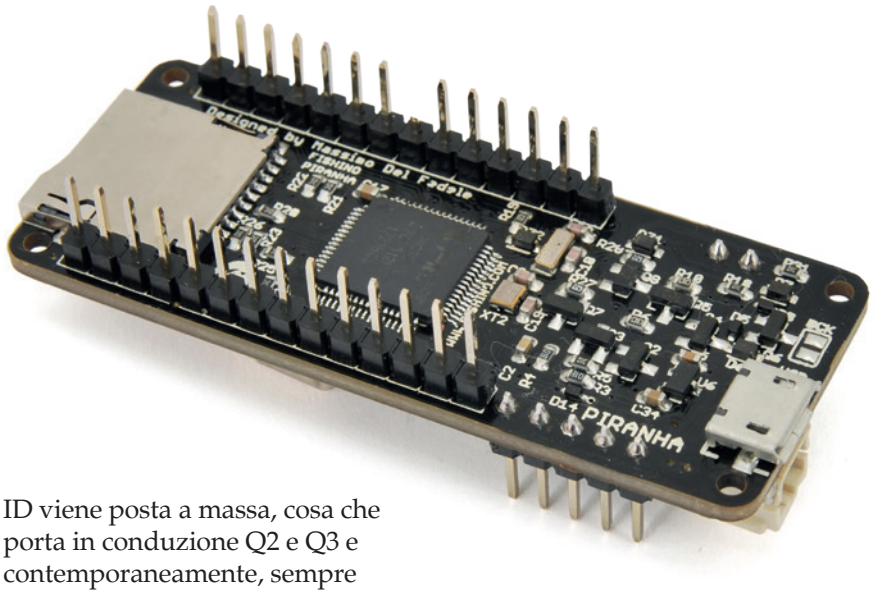
STADIO DI CONTROLLO DELL'ALIMENTAZIONE

Questo blocco è probabilmente quello di più difficile comprensione di tutta la scheda, quindi ne forniremo una descrizione completa, sebbene lo abbiamo già fatto per la Fishino32. Come potete notare dallo schema, questo stadio risulta decisamente più complesso rispetto ai modelli ad 8 bit e questo per le seguenti ragioni:

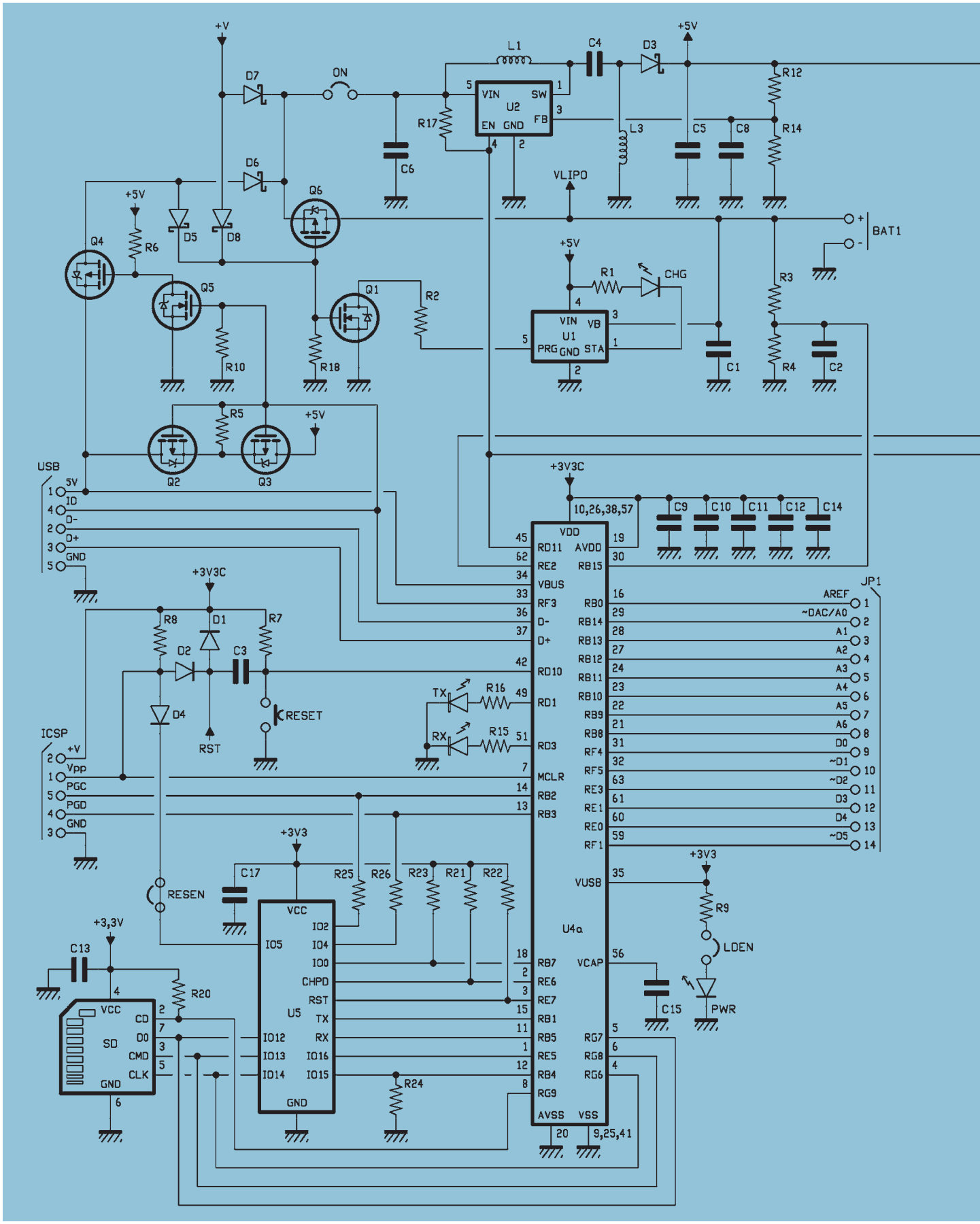
- la necessità di funzionare sia come host che come device USB;
- la tripla alimentazione;
- la possibilità di spegnimento software di tutto lo stadio di alimentazione;
- la possibilità di attivare esclusivamente l'alimentazione del controller in standby.

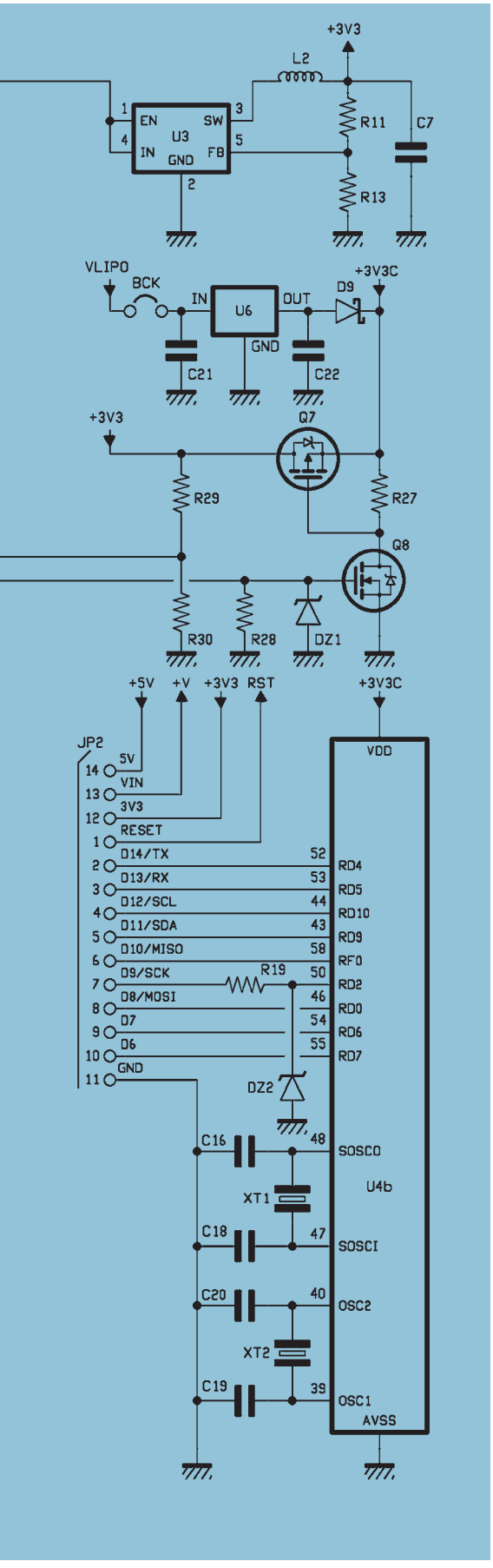
Le esigenze sono quindi parecchie. Iniziamo dalla sezione di controllo dell'alimentazione USB, la quale dev'essere in grado di prendere alimentazione (con cui far funzionare la Piranha) dal connettore USB se la board è connessa come device, mentre se funziona da host deve fornire un'adeguata alimentazione ad un device eventualmente connesso alla Piranha.
Per poter eseguire la commutazione ci viene in aiuto il piedino che fa capo alla linea USB ID, collegata sia al connettore USB che all'apposita porta del controller. I cavetti microUSB sono costituiti in modo da avere ad un estremo il pin corrispondente all'USB ID connesso a massa (tipo A), mentre all'altro estremo questo pin è scollegato (tipo B); l'apparecchiatura a cui è connesso l'estremità di tipo A assume il ruolo iniziale di host, e contemporaneamente fornisce l'alimentazione all'altra apparecchiatura, questa connessa all'estremo di tipo B, che assume il ruolo iniziale di device.
Abbiamo parlato di ruoli iniziali per un motivo ben preciso: in una connessione OTG i due device connessi possono cambiare ruolo; la specifica garantisce soltanto che la linea verrà alimen-

tata sempre dal device connesso all'estremità A.
Torniamo quindi al nostro schema e vediamo che alla linea USB ID è connessa una resistenza di valore elevato (R10, 100 kohm) verso massa, il cui scopo è non lasciare fluttuante la linea in caso di assenza di alimentazione, ed una resistenza, R5, di valore decisamente inferiore, (10 kohm) connessa in comune ai source dei MOSFET Q2 e Q3. Come si evince dallo schema, su questi source, grazie ai diodi interni ai MOSFET è sempre presente una tensione positiva (5V meno la caduta diretta sui diodi) proveniente o dalla presa USB o dall'alimentazione a 5V della scheda Piranha. Abbiamo quindi la certezza che la R5 sia connessa ai +5V in qualsiasi condizione di funzionamento. Se la linea USB ID è scollegata da massa (connessione tipo B) questa tensione positiva, attraverso R5, porta in conduzione il MOSFET a canale N Q5, il quale a sua volta porta in conduzione il MOSFET a canale P Q4; contemporaneamente, sempre attraverso R5 i due MOSFET Q4 e Q5 visti precedentemente vengono posti in interdizione ed essendo collegati in antiserie, non permettono il passaggio dell'alimentazione dai 5V di Fishino al connettore USB.
Sintetizzando, con USB ID fluttuante, quindi quando inseriamo un connettore di tipo B, la Fishino non fornisce alimentazione al connettore USB che, per contro, fornisce alimentazione al Fishino attraverso il MOSFET Q4. Otteniamo quindi la prima parte dello scopo desiderato, ovvero che con un connettore di tipo B Fishino ricava l'alimentazione dalla linea USB comportandosi da device vero e proprio.
Se inseriamo, per contro, un connettore di tipo A, la linea USB



ID viene posta a massa, cosa che porta in conduzione Q2 e Q3 e contemporaneamente, sempre attraverso Q5, in interdizione Q4; l'alimentazione fluisce quindi dal Fishino verso il connettore USB e non rientra "dalla porta di servizio" essendo Q4 interdetto.
È da notare, ovviamente, che nel secondo caso (connessione di tipo A) per poter fornire i 5V alla porta USB il Fishino dev'essere alimentato in qualche altro modo, ovvero dalla batteria o dal plug VIN, come vedremo in seguito.
Proseguendo verso il basso del circuito notiamo il complesso di diodi Schottky D5/D8 e D6/D7 rispettivamente, i primi due dei quali sono per piccole correnti, mentre i secondi sono diodi di potenza. Vediamo inoltre il MOSFET Q6, al quale giunge l'alimentazione della batteria.
Anche qui il funzionamento è abbastanza sottile; i diodi di potenza (D6 e D7), insieme al MOSFET Q6, costituiscono una porta OR di potenza, che confluisce nella linea VOR dello schema. Per i diodi il funzionamento è abbastanza semplice: la tensione maggiore tra VIN e VUSB viene trasmessa su VOR, mentre l'altra, a causa dell'interdizione del diodo corrispondente, che si trova polarizzato inversamente, viene isolata.
Ci si potrebbe chiedere perché sulla linea della batteria non abbiamo inserito semplicemente un diodo di potenza come per le altre due, cosa che effettivamente avrebbe semplificato il circuito, realizzando una porta OR di potenza con tre diodi. Questo è fattibile, ma c'è il solito problema della caduta di tensione sui diodi che, se per tensioni elevate è praticamente ininfluente, su una tensione ridotta come quella della batteria porta a perdite considerevoli: intuitivamente, si può dire che una caduta di tensione di 0,5V su 10 V corrisponde al 5%, mentre una caduta di 0,5V su 3,6V è pari a circa il 14%, cosa che si riflette sull'assorbimento in corrente come vedremo di seguito, e proprio dove servirebbe una maggior efficienza, ovvero nell'alimentazione a batteria.
Il MOSFET Q6 non ha questo problema, visto che non presenta una caduta di tensione fissa come un diodo, ma una resistenza di conduzione (particolarmente bassa nel modello da noi scelto); per contro, un MOSFET non ha il pregio del diodo di condurre solo quando serve, per cui occorre un sistema per mandarlo in conduzione o interdirlo a seconda delle necessità.
Per questo utilizziamo i due diodi per piccole correnti D5 e D8; il funzionamento del tutto è il seguente: se non è presente un'alimentazione su VIN e nemmeno


+V
D7
ON
L1
C4
D3
+5V
R12
VIN SW
U2
EN GND FB
R17
L3
C5
C8
R14
C6
D6
VLIPO
+5V
Q6
BAT1
R6
D5
D8
Q4
Q5
Q1
R2
R1
CHG
+5V
VIN VB
U1
PRG GND STA
R3
R10
R18
C1
R4
C2
R5
+5V
Q2
Q3
USB
5V
ID
D-
D+
GND
+3V3C
10,26,38,57
C9
C10
C11
C12
C14
VDD
RD11 AVDD
RE2 RB15
VBUS
RF3 RB0
D- RB14
D+ RB13
RB12
RD10 RB11
RB10
RD1 RB9
RB8
RD3 RF4
RF5
MCLK RE3
RB2 RE1
RB3 RE0
RF1
VUSB
VCAP
RB7
RE6
RE7
RB1
RB5 RG7
RE5 RG8
RB4 RG6
RG9
AVSS VSS
U4a
JP1
AREF
~DAC/A0
A1
A2
A3
A4
A5
A6
D0
~D1
~D2
D3
D4
~D5
+3V3C
R8
D1
R7
D2
C3
D4
RST
RESET
TX
R16
RX
R15
ICSP
+V
Vpp
PGC
PGD
GND
RESEN
+3V3
C17
R25
R26
R23
R21
R22
+3V3
R9
LDEN
PWR
C15
+3,3V
C13
R20
VCC
CD
DO
CMD
CLK
GND
SD
U5
VCC
IO2
IO5
IO4
IO0
CHPD
RST
TX
RX
IO12
IO13
IO16
IO14
IO15
GND
R24



attraverso VUSB, la resistenza R18 porta a massa il gate di Q6 che quindi conduce e permette il passaggio della tensione della batteria verso VOR. Invece se è presente una delle due (VIN o VUSB, o entrambe) uno dei diodi D5 e D8 conduce, polarizzando positivamente Q6 che passa in interdizione scollegando quindi la batteria. In breve, se abbiamo un'alimentazione esterna, che proviene dal plug VIN o dal connettore USB, la batteria viene scollegata; se invece manca l'alimentazione esterna entra in funzione la batteria.

Si può notare che la linea su cui confluiscono i diodi D5 e D8 ha un nome, BATOFF, perché questo segnale, che è a livello alto se è presente un'alimentazione esterna, verrà utilizzato anche in un'altra sezione del circuito.

Vediamo ora il gruppo di componenti che fanno capo ai MOSFET Q7 e Q8; questi servono per scollegare la linea +3V3, ovvero la linea principale a +3,3 V, dalla linea +3V3Core, destinata ad alimentare solo il controller. Ma a cosa servono? Semplicemente ad evitare che un'alimentazione destinata al solo controller, attivata quando tutto il resto è spento, e che ha lo scopo di mantenere attivo l'RTC interno ed altre funzionalità minime, confluisca nei circuiti che si vogliono tenere spenti.

Normalmente, se è presente tensione sui +3,3V, questa confluisce attraverso il diodo interno a Q7, e anche attraverso il Q7 stesso se questo è in conduzione, sull'alimentazione +3V3Core del controller; questo è il funzionamento normale. Se invece togliamo l'alimentazione sui +3,3V e contemporaneamente portiamo a livello basso la linea OFF, attraverso Q8 portiamo il Q7 all'interdizione e quindi impediamo ai +3V3Core di andare ad alimentare il resto della scheda. La linea OFF viene comandata dal controller, come vedremo in seguito, durante lo spegnimento programmato.

Sempre per motivi di spazio, nella Piranha non abbiamo inserito la pila a bottone al litio per il mantenimento (peraltro poco utile, essendo presente l'alimentazione a batteria).

Per ultima notiamo la linea PWRGOOD, connessa a un partitore resistivo; questa fornisce un segnale positivo ad un I/O del controller che indica la presenza di un'alimentazione adeguata sulla scheda e può venire utilizzato, appunto, per gestire la mancanza di questa e le funzioni di standby.

Come detto sopra, si tratta di una gestione piuttosto complessa, per la quale è disponibile un'apposita libreria software.

L'ultima parte della sezione in esame è quella che ruota attorno al regolatore lineare U6, che è un semplice regolatore a bassa corrente (max 100 mA circa) e bassissimo consumo, destinato a fornire la tensione di mantenimento al controller tramite la batteria LiPo esterna.

Questa sezione può essere disattivata tagliando il ponticello BCK, normalmente chiuso. Con il ponticello chiuso e la batteria connessa, occorre prestare attenzione, perché senza un apposito software installato il controller rischia di restare in funzione, prendendo l'alimentazione dalla batteria e prosciugandola in qualche ora, a meno di non aver messo il tutto in una delle modalità a basso consumo disponibili.

Essendo la libreria di gestione delle modalità a basso consumo (e di tutto lo stadio di alimentazione) piuttosto complessa, ne rimandiamo la descrizione ad un articolo successivo.

STADIO SWITCHING

Questo blocco è praticamente identico a quello della scheda Fishino MEGA già descritta nel fascicolo n° 204 (aprile 2016), quindi evitiamo di ripeterne la descrizione; basti sapere che viene utilizzato un convertitore switching ad alta efficienza in modalità SEPIC, in grado di accettare in ingresso una tensione da un minimo di 3 volt ad un massimo di 20 volt, fornendo in uscita i 5 volt necessari.

STADIO CARICABATTERIA

Anche questo stadio è già stato esaminato nelle schede precedenti, quindi rimandiamo, per i dettagli, ai relativi articoli. Fa capo all'onnipresente MCP73831, che è un completo regolatore di carica integrato per LiPo a cella singola. Qui l'unico punto degno di nota è costituito dal MOSFET Q1, un canale N, utilizzato per abilitare o meno la funzione di carica della batteria.
Se ricordiamo la descrizione del primo stadio (controllo alimentazioni), la linea BATOFF è a livello alto quando è presente un'alimentazione esterna; in questo caso il MOSFET Q1 va in conduzione, portando a massa la resistenza R2 ed attivando quindi la carica. In assenza di tensioni esterne (alimentazione a batteria) la linea BATOFF va a livello basso, disattivando la carica.
A completamento della descrizione notiamo le due resistenze R3 ed R4, che costituiscono un partitore di tensione in grado di fornire metà della differenza di potenziale della batteria LiPo alla linea RB15 del microcontrollore, il cui ADC la misura per determinare e segnalare lo stato di carica. Abbiamo utilizzato un partitore perché la tensione della batteria LiPo, a piena carica, supera i 3,3 volt tollerati dagli ingressi analogici del controller; di ciò occorre tenere conto nell'eseguire la misura con l'ADC.

STADIO MICROCONTROLLER

Attorno al microcontrollore PIC32MX470F512 notiamo due quarzi, uno da 20 MHz e l'altro da 32,768 kHz; il primo fornisce il clock principale, attraverso un PLL interno, che lo moltiplica fino a 120 MHz. Si tratta di un circuito non proprio a basso consumo, quindi da evitare nelle condizioni di risparmio energetico; per contro, il quarzo da 32,768 kHz serve esclusivamente per il Real Time Clock (RTC) interno al PIC, che è un circuito studiato appositamente per avere un bassissimo assorbimento e che può quindi essere lasciato in funzione anche in condizioni di standby.
Accettando una precisione minore sulla frequenza di clock si può omettere il quarzo e utilizzare l'oscillatore RC interno, che però non è in grado di portare il microcontrollore alla sua massima frequenza (120 MHz) a causa di vincoli del PLL interno, quindi il clock interno verrebbe limitato a 96 MHz.
Si notino, tra le svariate connessioni sul controller, le tre linee viste in precedenza, ovvero la OFF, che gli permette di togliere l'alimentazione principale, la PWRGOOD, che consente il controllo dello stato della medesima, e la linea MBAT che permette la misura della tensione della batteria.
Diamo adesso uno sguardo alla sezione reset, interessante perché utilizza un solo pulsante di reset sia per resettare il micro, sia per altre funzioni ottenute in abbinamento con un apposito bootloader sul microcontrollore. Notiamo subito tre linee che fanno capo allo stadio di reset:

- MCLR;
- REMOTERESET;
- LONGRESET.

La linea MCLR è quella di reset vero e proprio del microcontroller; portandola a massa, il medesimo viene riavviato. Notiamo subito che questo può avvenire sia premendo il pulsante di reset, il quale attraverso il condensatore C22 invia un breve impulso negativo alla linea MCLR, sia portando a massa la linea REMOTERESET, la quale fa capo al modulo WiFi e che consente quindi a questo di resettare il controller.
La linea LONGRESET, per contro, viene a trovarsi a massa per tutto il tempo in cui il pulsante di reset viene mantenuto premuto, consentendo quindi al bootloader di conoscere le nostre intenzioni.
In sintesi:

- una breve pressione del pulsante reset (inferiore ai 2 secondi) o un segnale negativo sulla linea REMOTERESET riavviano la scheda, come d'abitudine;
- mantenendo premuto il pulsante di reset per un tempo tra 2 e 4 secondi viene avviato il bootloader (cosa evidenziata dal lampeggio del LED arancione), mettendo quindi la scheda in modalità caricamento sketch;
- mantenendo premuto il pulsante per un tempo superiore ai 4 secondi si avvia la procedura di upgrade del firmware del modulo WiFi, cosa evidenziata dal lampeggio del LED blu.

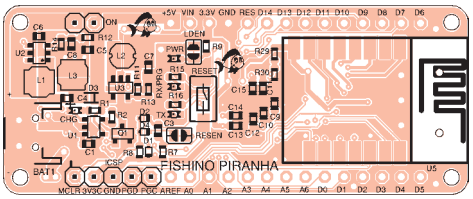
È comunque possibile, una volta entrati nelle due modalità del bootloader, riavviare semplicemente la scheda ed eseguire lo sketch precaricato ripremendo brevemente il pulsante di reset.
Per quanto riguarda la sezione led, si tratta semplicemente di

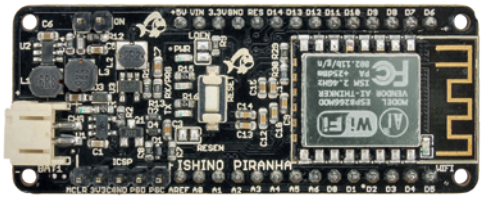
Elenco Componenti:

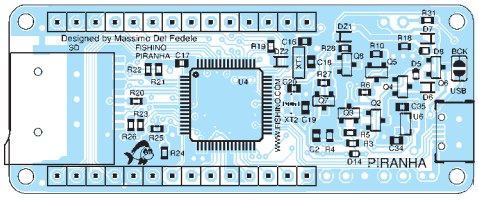
C1, C17: 4,7 µF ceramico (0603)
C2, C9÷C14: 100 nF ceramico (0603)
C3, C15, C21, C22: 1 µF ceramico (0603)
C4: 4,7 µF 25 VL ceramico (0603)
C5: 22 µF ceramico (0603)
C6: 22 µF 25 VL ceramico (0603)
C7: 10 µF ceramico (0603)
C8: 360 pF ceramico (0603)
C16, C18÷C20: 22 pF ceramico (0603)
D1, D2, D4, D5, D8, D9: RB521S
D3, D6, D7: PMEG3020EJ
DZ1, DZ2: MM3Z3V3
Q1, Q5, Q8: 2N7002
Q2÷Q4: FDN340P
Q6, Q7: NTR4171P
R1, R15, R16: 470 ohm (0603)
R2: 3,3 kohm (0603)
R3, R4: 1 Mohm (0603)
R5, R8, R17, R18, R20÷R26, R29: 10 kohm (0603)
R6, R10: 100 kohm (0603)
R7, R9: 1 kohm (0603)
R11: 475 kohm (0603)
R12: 97,6 kohm (0603)
R13: 105 kohm (0603)
R14: 13,3 kohm (0603)
R19, R27, R28, R30: 220 kohm (0603)

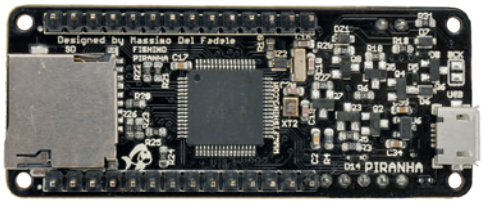
U1: MCP73831
U2: SX1308
U3: LC3406
U4: PIC32MX470F512HI/MR
U5: Modulo ESP12 con ESP8266
U6: XC6206P33
XT1: 32.768 KHz
XT2: 20 MHz
TX: LED blu (0603)
RX/PRG: LED giallo (0603)
PWR: LED verde (0603)
CHG: LED rosso (0603)
L1, L3: 6.8 µH
L2: 2.2 µH
USB: Connettore micro-USB
SD: Connettore micro-SD
LIPO: Connettore JST 2 vie 2.54 mm
RESET: Microswitch

Varie:
- Strip maschio 14 vie (2 pz.)
- Strip maschio 5 vie
- Strip maschio 2 vie
- Jumper
- Circuito stampato S1281 (62x25 mm)









due led connessi ad altrettante porte del controller, utilizzabili per qualsiasi compito (indicati sulla scheda come RX/PRG e TX, rispettivamente); su questa scheda è assente, per motivi di spazio, l'abituale LED13 utilizzato, per esempio, nei test con lo sketch BLINK. Questo è comunque sostituito egregiamente dai 2 led di cui sopra, accessibili tramite gli I/O digitali 25 e 26.
Passiamo adesso alla sezione WiFi, simile a quella delle Fishino precedenti, perché il modulo ha sempre un firmware da noi realizzato, in grado di comunicare ad alta velocità tramite la porta SPI e non la porta seriale.
In questa scheda è stato eliminato il connettore ESPCONN al quale eravamo abituati nelle board precedenti, ed al quale facevano capo alcuni I/O e pin di controllo del modulo sfruttati sia per programmarlo che, per alcuni di essi, come I/O aggiuntivi. Questo perché nel Fishino Piranha abbiamo già liberato tutti gli I/O che venivano usati per comunicare con il modulo e con la scheda SD, che erano il 4, 7 e 10 su UNO e Mega; abbiamo inoltre, rispetto a Fishino32, separato anche l'interfaccia SPI utilizzando quella che in quest'ultimo veniva dedicata al codec audio. Tutte le porte disponibili sugli headers del Piranha sono quindi libere da qualsiasi vincolo e/o interferenza con le periferiche interne.
Non solo, due porte sul connettore ICSP sono sfruttabili come porte digitali aggiuntive (ne parleremo in dettaglio descrivendo il software), quindi abbiamo preferito sfruttare le connessioni con il modulo ESP in altro modo. Infatti, tutte le linee dell'ESP (salvo l'ingresso analogico) sono ora sotto controllo del processore principale, cosa che ci permette di effettuare le seguenti operazioni direttamente da software, senza impegnare alcun I/O:

- resettare il modulo ESP via software;
- spegnere il modulo da software (cosa che in precedenza richiedeva l'uso di un I/O di Fishino);
- caricare il firmware sul modulo senza dover effettuare collegamenti strani né caricare sketch particolari;
- ridirigere l'output seriale del modulo sulla seriale USB o su qualsiasi altra seriale, sia software che hardware, disponibile sul Fishino.

Per il resto, nulla cambia rispetto

alle schede precedenti; la comunicazione avviene attraverso la porta SPI interna che, come accennato sopra, è separata da quella che fa capo agli headers e quindi accessibile esternamente. Questo è stato possibile data l'assenza del codec audio, presente invece sulla Fishino32, che richiede una porta SPI dedicata. L'interfaccia verso la microSD è identica a quella delle schede già viste; sfrutta l'interfaccia SPI interna come il modulo WiFi, mentre anche qui per la selezione della scheda viene utilizzato un I/O interno, lasciando libero quindi l'I/O D4, che era necessario a questo scopo nelle schede ad 8 bit.
Potremo quindi utilizzare qualsiasi periferica SPI esterna e, anche se faremo qualche errore nei nostri sketch, questo non provocherà il blocco nè del modulo WiFi nè della scheda microSD.
Come ciliegina sulla torta abbiamo 2 ulteriori porte digitali (questa volta NON 5V tolerant) a disposizione sullo header ICSP, quando questo non è utilizzato nella programmazione del bootloader. L'unica grossa differenza rispetto alle schede precedenti è l'assenza degli stadi di conversione livelli (da 5 a 3,3 V) visto che il PIC utilizza i 3,3 V come alimentazione, rendendo quindi possibile (e più veloce!) un collegamento diretto. Unico accorgimento da tenere, nell'esecuzione di sketch esistenti: il pin di selezione scheda (CS) non è più il 4 (o il 10 in alcune versioni di Arduino) ma un pin interno denominato SDCS; è quindi sufficiente sostituire tutti i riferimenti all'I/O 4 con l'I/O SDCS (scritto proprio così, visto che può cambiare fisicamente da scheda a scheda).

HEADER

In questa sezione sono raggruppati tutti i connettori che portano i segnali verso l'esterno della scheda, quindi gli header laterali il connettore ICSP (In Circuit Serial Programming) che serve per programmare la scheda senza passare per il bootloader (ad esempio per programmare il bootloader stesso), e sul quale sono disponibili due I/O digitali aggiuntivi, ed il connettore USB.
Come accennato parlando delle caratteristiche della scheda, la PIC ha molti, ma non tutti, i pin I/O 5V tolerant, ovvero in grado di accettare 5V in ingresso anche se il controller è alimentato a 3,3 V senza danneggiarsi. Dove possibile abbiamo sfruttato quei pin portandoli all'esterno come I/O digitali; purtroppo in alcuni casi questo non è stato possibile, ed abbiamo quindi inserito una protezione in ingresso tramite resistenze e diodi.
La protezione consiste in una resistenza da 220 ohm in serie all'ingresso ed un diodo in grado di scaricare l'eccedenza sulla linea a 3,3 V, limitando quindi la tensione in ingresso a tale valore.
Sui pin analogici, che non sono 5V-tolerant (proprio a causa della circuiteria analogica interna al controller) non è presente alcuna protezione. Attenzione, quindi, a non inserire nei medesimi delle tensioni superiori a 3,3 Volt, pena il rischio di danneggiamento del chip. L'integrato dispone, infatti, di diodi di protezione al suo interno, che però sono in grado di sopportare correnti bassissime e/o per brevi periodi di tempo, ovvero scariche elettrostatiche, quindi non tenetene conto!
Per concludere questa sezione, il layout dei connettori esterni è praticamente identico a quello dell'Arduino MKR1000, salvo l'output analogico (DAC) non presente nella PIC utilizzata; potete quindi realizzare quasi tutti i progetti disponibili per la MKR1000 con la nostra Piranha, con il vantaggio delle prestazioni decisamente superiori!

REALIZZAZIONE PRATICA

Piranha è una scheda realizzata completamente con componenti SMD, alcuni di dimensioni particolarmente ridotte, e popolata su entrambi i lati. Il circuito stampato è a doppia faccia con fori metallizzati e potrebbe anche essere autocostruito con metodi artigianali, tuttavia potrebbe essere difficile realizzare le vie di interconnessione tra le piste delle due facce; peraltro alcune piste sono particolarmente sottili, il che esige un'incisione perfetta.
Il montaggio richiede molta attenzione e un'attrezzatura specifica composta da un saldatore a punta finissima e filo di lega saldante del diametro massimo di 0,5 mm; della pasta flussante sparsa sulle piazzole può aiutare la saldatura e una lente d'ingrandimento consente di vedere se le saldature sono ben fatte, eccedenti o se i componenti sono spostati.
Meglio sarebbe, per alcuni componenti, ricorrere a una stazione ad aria calda e utilizzare del flussante (da cospargere sulle piazzole, dopo aver sciolto su queste ultime un velo di stagno) per agevolare la saldatura e impedire che la lega saldante cortocircuiti pad adiacenti.
Suggeriamo quindi, ai meno esperti, di acquistare la scheda già pronta all'uso, visto che a nostro avviso il tempo ed il costo dei materiali per una realizzazione casalinga non vengono compensati dal risparmio che se ne potrebbe ottenere. Sul nostro sito web *www.elettronicain.it* mettiamo comunque a disposizione tutti i file necessari per

poterlo fare, come d'abitudine; poi decidete voi. Per tutte le fasi del montaggio, seguite i disegni nella pagina accanto.

MESSA IN FUNZIONE

Come abbiamo detto ad inizio articolo, la programmazione di Piranha si può fare tranquillamente tramite l'IDE di Arduino, avendo l'accortezza di installare il package da noi fornito sul sito *www.fishino.it*, contenente tutti i programmi aggiuntivi necessari. L'installazione del package, grazie al lavoro del team di Arduino, è decisamente semplice: è sufficiente aprire l'IDE, selezionare dal menu File il sottomenu Impostazioni e nella riga intitolata "Url aggiuntive per il Gestore schede" inserire il seguente percorso: *www.fishino.it/arduinoide/package_fishino_index.json*.
Come si vede nella schermata mostrata in **Fig. 1** è possibile inserire più percorsi, facendo clic sul pulsante tramite il quale si aprirà la finestra in **Fig. 2**.
Una volta inserito il percorso di Fishino, è sufficiente entrare nel menu 'Strumenti-Scheda-Gestore schede...', tramite il quale si avrà accesso al pannello di gestione delle schede aggiuntive; verso la fine troverete il pacchetto *fishino_pic32 by Fishino* (**Fig. 3**).
Potete notare il pulsante *Installa* posto sulla destra, insieme a una casella per selezionare il numero di versione.
Al momento della stesura di questo articolo il numero di versione è 1.0.1, ma il pacchetto è in continuo aggiornamento. Fate clic sul pulsante e verrà avviato lo scaricamento dei pacchetti, che potrà durare anche parecchi minuti; non preoccupatevi comunque, perché questa operazione va eseguita solo una volta, o comunque per aggiornarlo, nel qual caso verranno scaricati solo gli elementi modificati.
Una volta terminata l'installazione, la schermata si presenterà come in **Fig. 4**.
Notate che adesso il pulsante

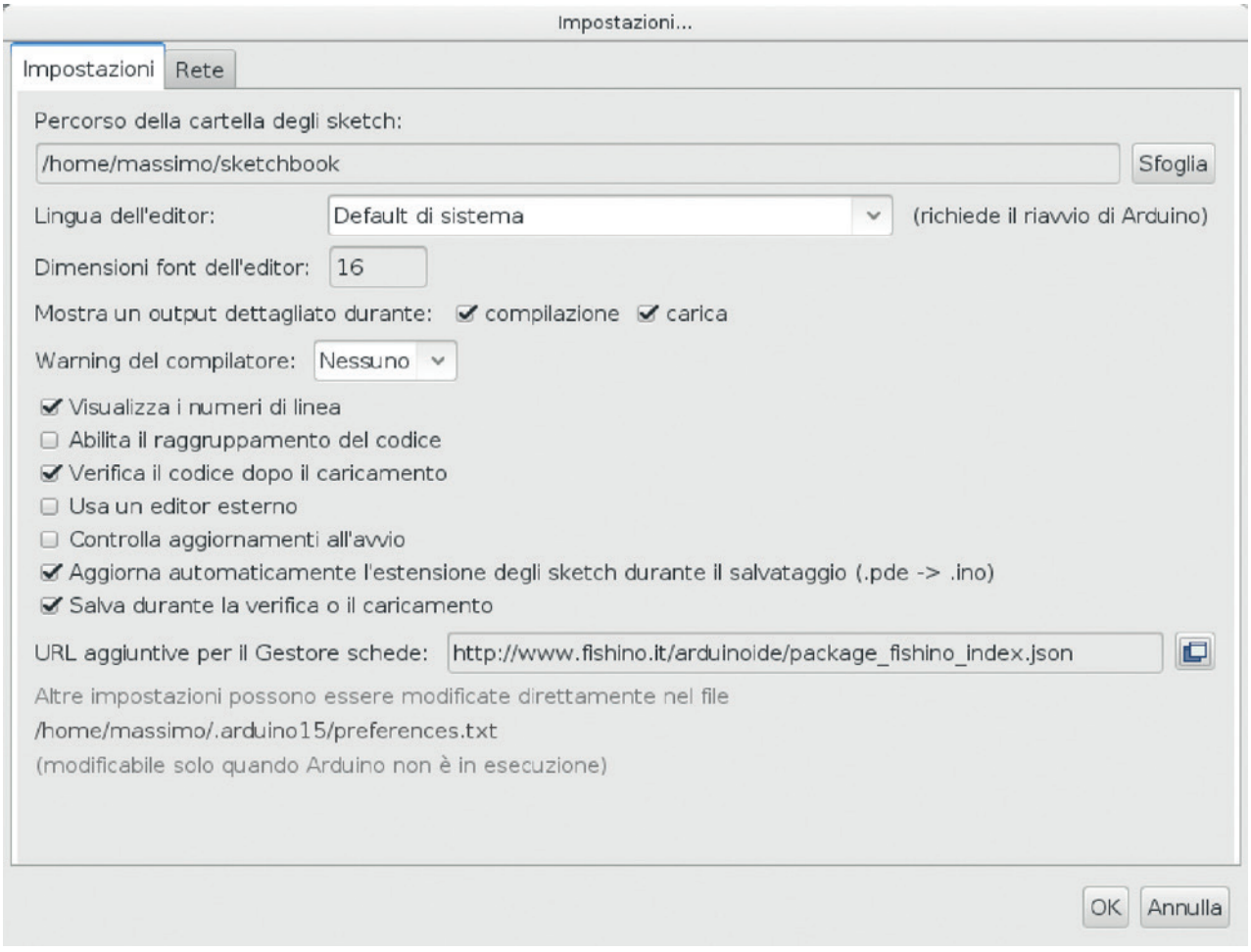


***Fig. 1** - Indicazione del percorso di ricerca del package.*

Installa è sostituito dal pulsante Rimuovi ed è apparso l'elenco delle schede disponibili; non preoccupatevi se sarà diverso da quanto appare nella **Fig. 5**, stiamo studiando ulteriori schede! Una volta completata questa operazione, nel menu di selezione della scheda troverete la nostra Fishino32, come visibile nell'immagine a fianco.
In alternativa potete utilizzare la nostra FishIDE, un'IDE da noi sviluppata non solo per le schede di Fishino ma anche per tutte le schede compatibili con l' IDE di Arduino, e dotata di numerose funzionalità aggiuntive.
Anche qui, per la descrizione rimandiamo al relativo articolo già apparso sulla rivista e/o al sito *www.fishino.it* dove potrete trovare tutti i dettagli.
Ecco fatto: La vostra Piranha è quasi operativa! Scriviamo "quasi" perché mancano ancora i driver USB, che per alcuni sistemi operativi è necessario installare; non è il caso di Linux, nel quale sono inclusi e nemmeno di OS/X. Intendiamo Microsoft Windows, che purtroppo da solo non ce la fa. Per esso occorre procedere come spiegato nella sezione qui di seguito.

INSTALLAZIONE DRIVER USB PER WINDOWS

I driver vengono scaricati automaticamente insieme al pacchetto software contenente le descrizioni delle nuove schede che abbiamo appena installato; purtroppo non vengono installati automaticamente nel sistema operativo Windows per motivi che non dipendono da noi.
Per l'installazione è sufficiente, una volta collegato Piranha al PC, seguire le istruzioni che appariranno sullo schermo, cercando i driver manualmente nel seguente percorso:

```
C:\Users\Massimo\AppData\Local\
Arduino15\packages\fishino\tools\
pic32-drivers-windows\1.0.0\chipKIT
Drivers
```

Nel vostro sistema il percorso sarà probabilmente differente, soprattutto il nome dopo 'Users' (che sarà quello del vostro utente) e probabilmente la versione o altro.
Una volta selezionato il file *Stk500v2.inf* il driver verrà installato ed apparirà la porta nell'IDE.

```
<inserire un eventuale dettaglio/im-
magini dell'installazione dei dri-
vers e/o modificare le istruzioni
fornendo direttamente il driver sca-
ricabile.......>
```

Una volta installati (se necessario) i driver, il sistema è pronto all'uso. Occorre selezionare la porta corretta nell'IDE, che sarà una COMxx su Windows o una TTYACMxx su Linux.

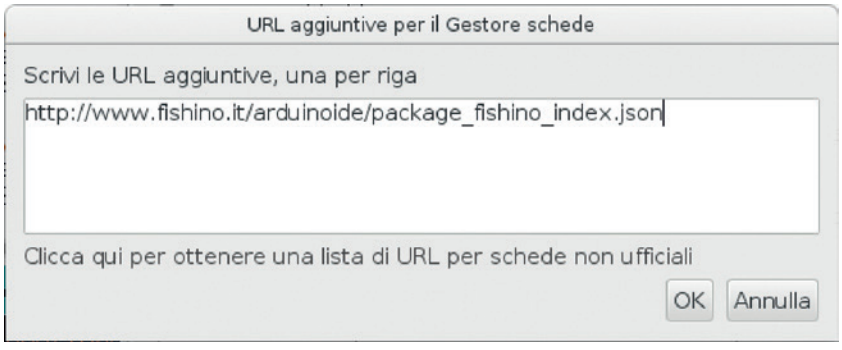


***Fig. 2** - Inserimento URL aggiuntivi.*

ESECUZIONE DEL PRIMO SKETCH

Eccoci pronti per il collaudo della nostra scheda!
Qui troviamo subito una piccola differenza, che può sembrare una scomodità, rispetto ad Arduino e alle schede Fishino ad 8 bit, ma che in realtà spesso si traduce in un vantaggio: la scheda attualmente non dispone di un autoreset che la mette automaticamente in modalità caricamento sketch; occorre quindi premere il pulsante di reset, tenendolo premuto per circa 2 secondi, finché non lampeggia il LED arancione di programmazione, ma non più di 4 secondi, dopodiché viene impostata la modalità di aggiornamento del firmware del modulo WiFi, cosa evidenziata dal lampeggio del LED blu.
Se ci è scappato il tempo, poco male... rilasciamo il tasto reset e ricominciamo!
Perché questa scelta? Semplicemente perché, disponendo la scheda di un USB nativo, questo è legato al funzionamento del controller stesso, come nel caso delle schede Leonardo di Arduino. Se il controller si blocca (basta uno sketch che si blocchi) la porta USB non risponde più ai comandi, nemmeno a quelli di reset. Abbiamo preferito quindi al momento un sistema manuale che eviti, ad esempio, la famigerata "manovra di emergenza" necessaria in alcuni casi sulle schede Arduino, ovvero la pressione del tasto reset "un istante" dopo aver lanciato la programmazione dall'IDE Arduino/Fishino.
Non escludiamo comunque di inserire, in una prossima versione del bootloader, un sistema di reset automatico, anche se i vincoli

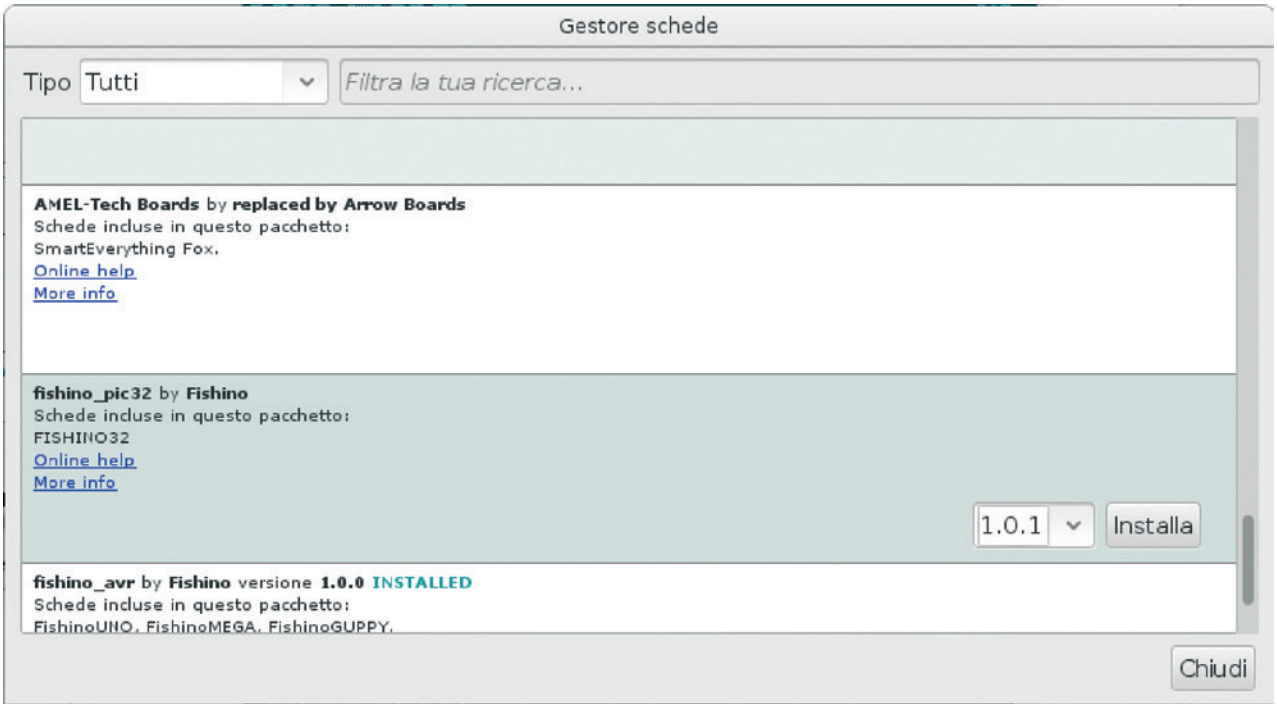


Fig. 3 *- Ricerca del package da installare.*

di cui abbiamo parlato restano. Torniamo quindi al nostro sketch; carichiamo l'esempio BLINK dall'IDE, quello che fa lampeggiare il LED13 (che corrisponde al LED bianco sulla scheda), teniamo premuto il tasto reset per circa 2 secondi e, quando inizia a lampeggiare il LED arancione, premiamo il pulsante di caricamento sketch dell'IDE.
Dopo pochi secondi (dipende dalla velocità del PC, dalla complessità dello sketch, dal numero di librerie utilizzate, eccetera...) il caricamento sarà completato, cosa evidenziata dallo spegnimento del LED arancione e dal

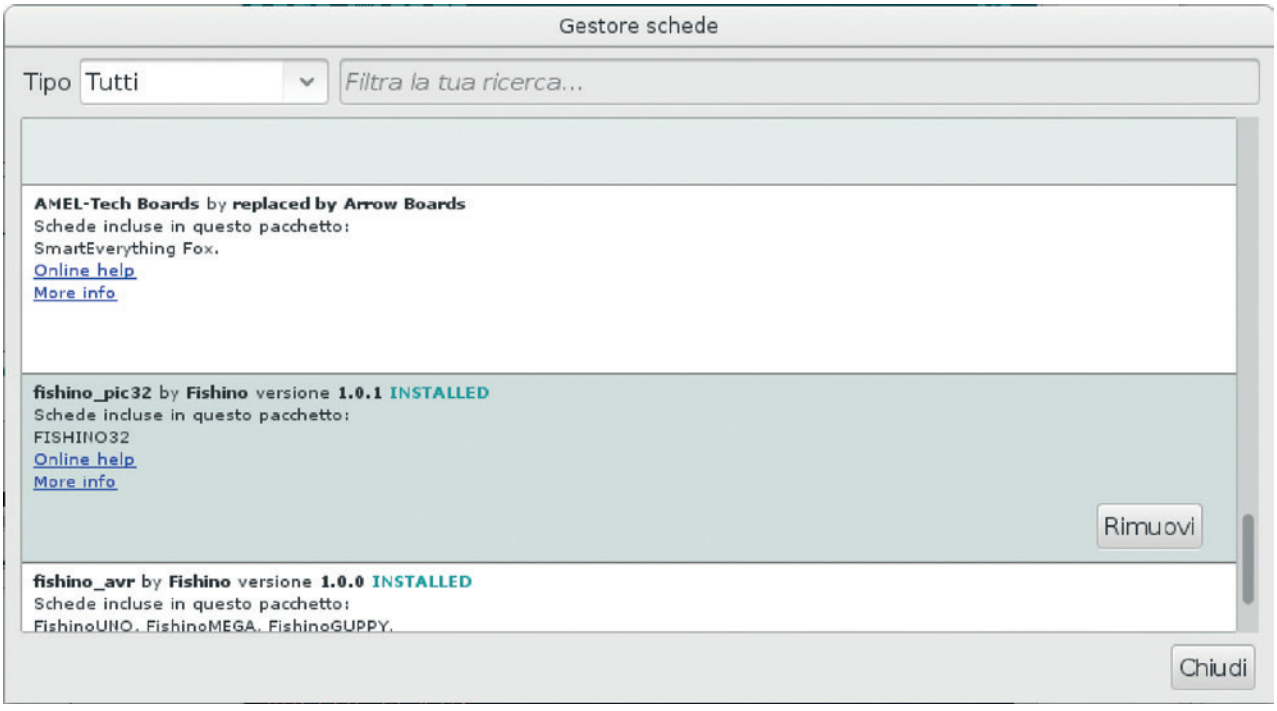


Fig. 4 *- Installazione completata.*

***Fig. 5** - Elenco delle schede installate.*

contemporaneo lampeggiamento del LED bianco.
Ecco fatto! Scheda collaudata e funzionante! Ed ora ?

AGGIORNIAMO IL MODULO WIFI

Se avete seguito gli articoli sulle schede Fishino ad 8 bit, ricorderete sicuramente che la procedura di aggiornamento firmware richiedeva alcune connessioni volanti tra gli I/O della scheda e la porta ESPCONN, il caricamento preventivo di uno sketch "neutro", ad esempio il blink, eccetera.
Nulla di tutto ciò è più necessario sul Fishino32: la scheda viene posta in modalità aggiornamento firmware utilizzando semplicemente il pulsante di reset!
La procedura è la seguente:

1. premere e mantenere premuto il tasto reset finché non inizia a lampeggiare il LED blu; occorrono 4 secondi (dopo i primi 2 inizierà a lampeggiare il LED arancione, ad indicare la modalità di caricamento sketch; dopo altri 2 secondi, inizierà a lampeggiare il LED blu);
2. a questo punto è sufficiente lanciare il programma FishinoFlasher, avendo l'accortezza di avere una connessione Internet funzionante sul proprio computer, visto che il firmware verrà scaricato dalla rete.

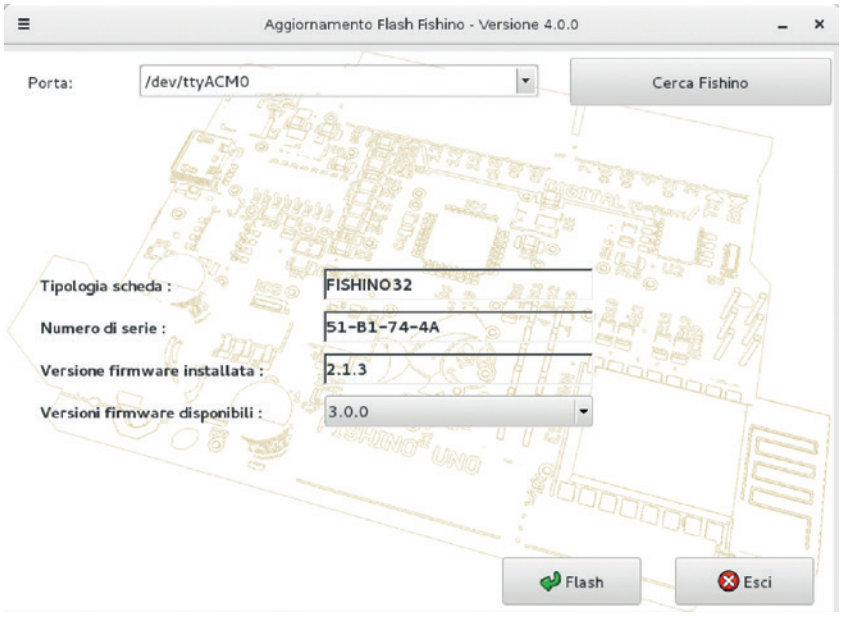


***Fig. 6** - Schermata di FishinoFlasher.*

Il programma FishinoFlasher si presenterà con la schermata proposta in **Fig. 6**.
Nella schermata si possono notare alcuni elementi, tra i quali la tipologia della scheda rilevata (Fishino32), il numero di serie del modulo WiFi e la versione attualmente installata del firmware (2.1.3 in questo caso).
Nella casella sottostante è possibile scegliere una nuova versione da installare; viene proposta l'ultima versione disponibile (3.0.0 in questo caso), ma nulla vieta di installare una versione precedente, anche se normalmente non è consigliato.
Fate attenzione che ad un cambio di numero di versione "major" (la prima cifra a sinistra, ossia 3 in questo caso) corrisponde una non retrocompatibilità con le librerie precedenti; in pratica, una libreria installata e funzionante con la versione 2.5.5 funzionerà anche con un firmware di versione 2.5.7 e 2.9.9, per esempio, anche se non ne sfrutterà le migliorie aggiuntive (non è vero il contrario, ovvero una libreria fatta per la versione 2.5.7 richiederà un firmare di versione almeno pari a 2.5.7!); per contro, quella libreria non funzionerà con un firmware 3.0.0.
In sintesi: una libreria fatta per un firmware avente un numero di versione major X funzionerà con tutti i firmware successivi aventi lo stesso numero di versione major, ma non con i precedenti, né con i successivi aventi numero di versione major differente.
Detto questo, scegliamo la nostra

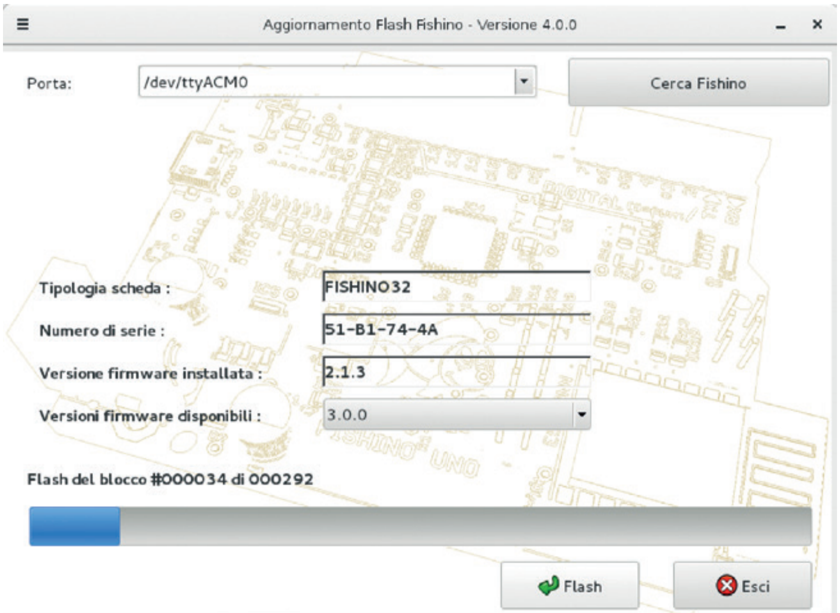


Fig. 7 *- Aggiornamento in corso.*

versione e premiamo il pulsante Flash; dopo pochi istanti inizierà l'aggiornamento vero e proprio (**Fig. 7**) il cui stato di avanzamento è mostrato nella barra orizzontale in basso, sopra i pulsanti Flash ed Esci.
Al termine della procedura apparirà una finestrella di conferma dell'avvenuto aggiornamento, come mostrato dalla **Fig. 8**. A questo punto è sufficiente premere Ok, chiudere il programma FishinoFlasher e premere reset sulla scheda per uscire dalla modalità aggiornamento.

CONCLUSIONI

Da quando abbiamo presentato la scheda Fishino32 abbiamo aggiunto un numero di librerie tale da renderne improponibile la descrizione in un articolo della rivista; per alcune di esse (in particolare per la libreria di gestione dell'alimentazione) presenteremo degli appositi articoli in seguito.
Concludiamo qui la presentazione della nostra nuova scheda Piranha; ovviamente nei prossimi mesi vi proporremo numerose applicazioni basate su di essa ed inizieremo a descrivere in dettaglio le librerie e il funzionamento hardware dei componenti interni, quando necessario.
Tra gli approfondimenti che vi proporremo ne dedicheremo uno particolare ad un "nuovo" modo di programmare le nostre schede a 32 bit, attraverso un linguaggio semi-interpretato, chiamato Squirrel, che offre nuove ed interessanti possibilità.

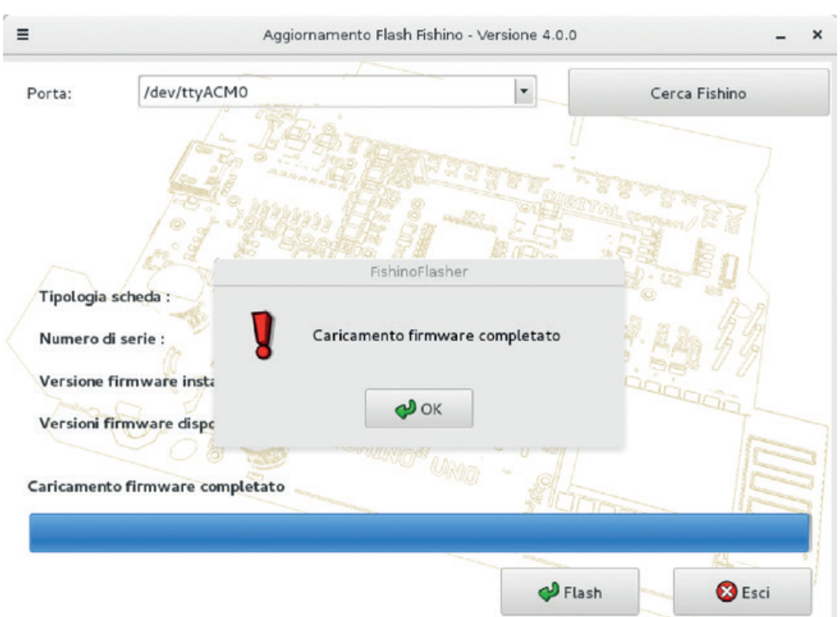


Fig. 8 *- Aggiornamento completato.*

per il **MATERIALE**

Tutti i componenti utilizzati in que-
s
b
p
c
u
c
r

Sostituire testo

Euro mentre l'integrato Microchip MCP3905A è disponibile 4,20

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