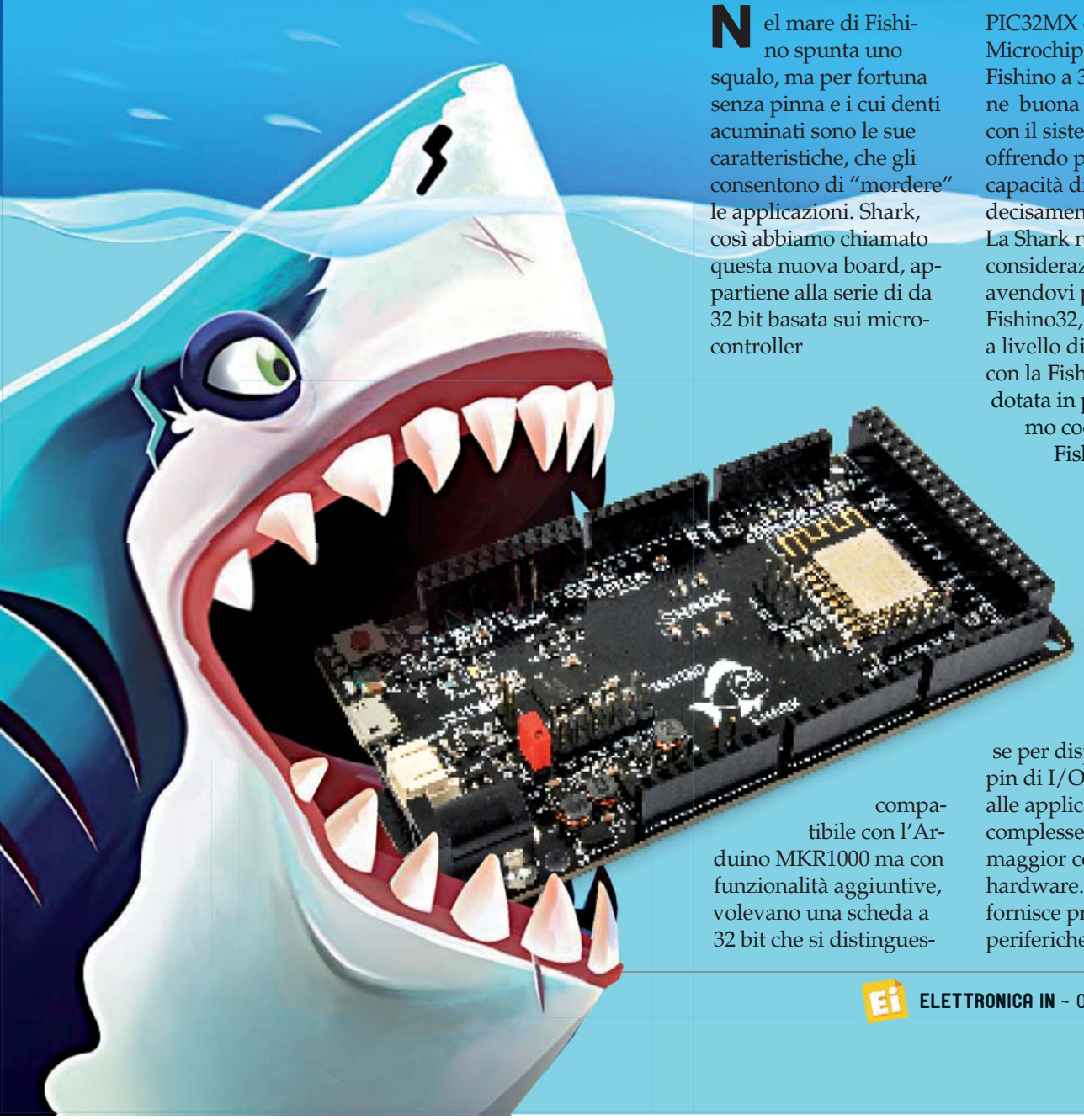
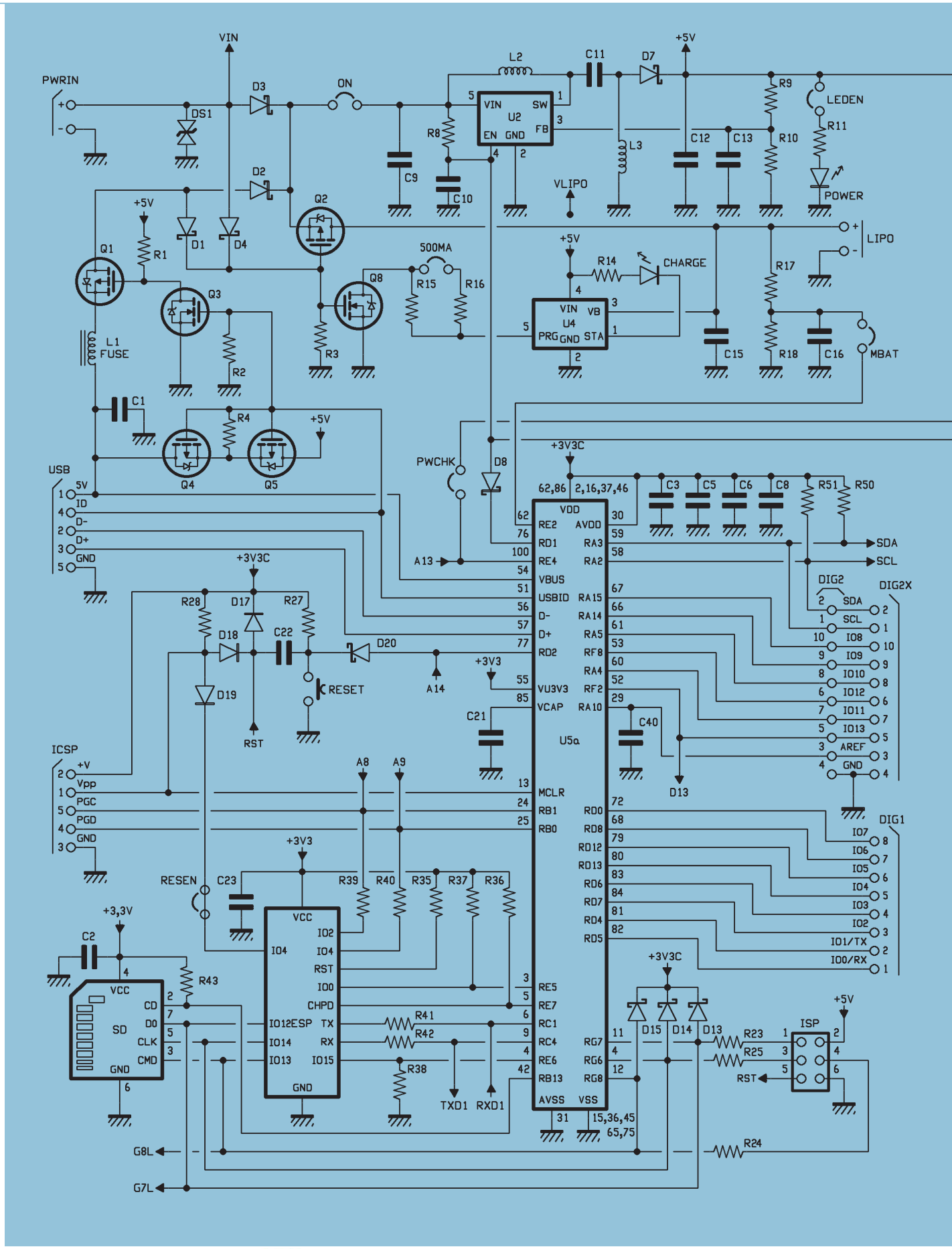
SHARK, A FISHINO SPUNTANO I DENTI

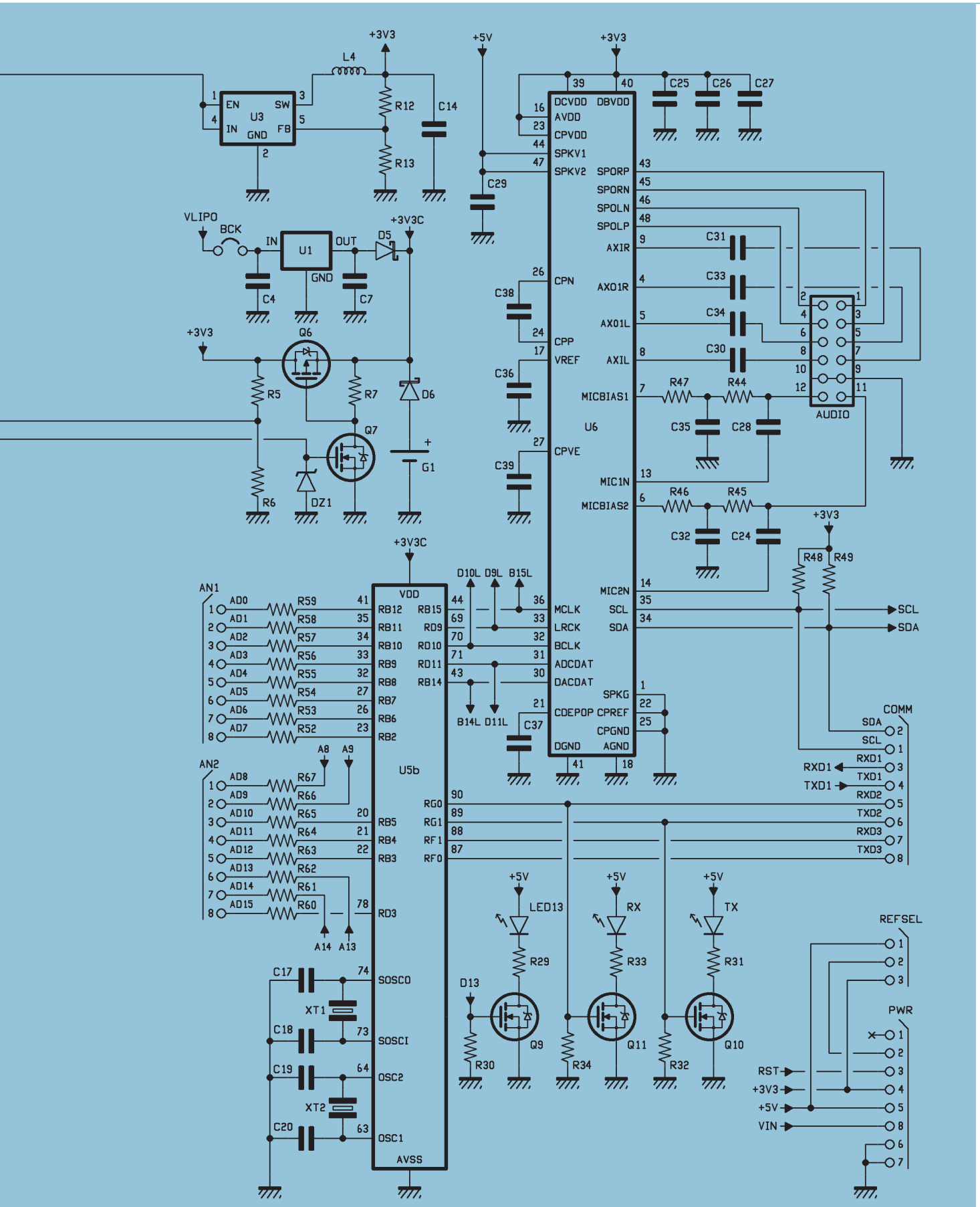
Una nuova scheda Arduino-like appare all'orizzonte ed ha prestazioni tali da poter "divorare" le concorrenti: non a caso è stata chiamata Shark, ossia squalo! Conosciamone le caratteristiche e le novità. Prima puntata.

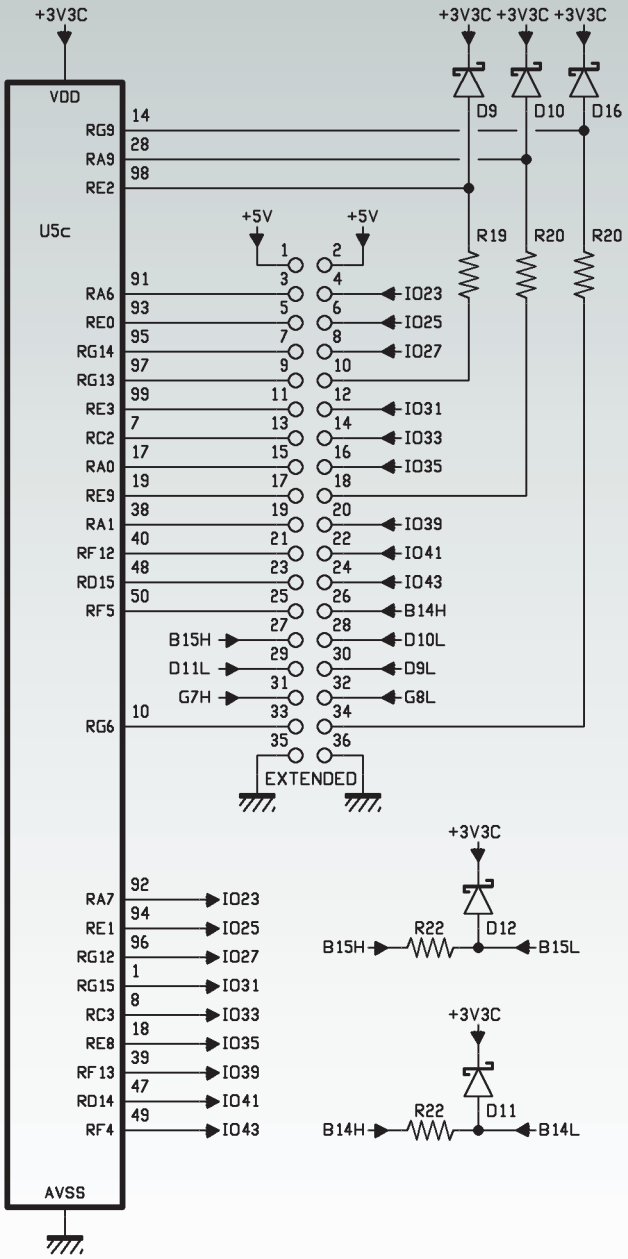
di MASSIMO DEL FEDELE



Nel mare di Fishino spunta uno squalo, ma per fortuna senza pinna e i cui denti acuminati sono le sue caratteristiche, che gli consentono di "mordere" le applicazioni. Shark, così abbiamo chiamato questa nuova board, appartiene alla serie di da 32 bit basata sui microcontroller PIC32MX della Microchip. Come le altre Fishino a 32 bit, mantiene buona compatibilità con il sistema di Arduino, offrendo prestazioni e capacità di memoria decisamente superiori. La Shark nasce dalla considerazione che, pur avendovi proposto la Fishino32, compatibile a livello di connettori con la Fishino UNO ma dotata in più di un ottimo codec audio, e la Fishino Piranha, compatibile con l'Arduino MKR1000 ma con funzionalità aggiuntive, volevano una scheda a 32 bit che si distinguesse per disponibilità di pin di I/O, da destinare alle applicazioni più complesse e richiedenti maggior connettività hardware. Fishino Shark fornisce prestazioni e periferiche di una 32 bit


VIN
PWRIN
DS1
D3
D2
D1
D4
ON
L2
C11
D7
+5V
R9
LEDEN
R11
POWER
U2
VIN SW
EN GND FB
R8
C9
C10
L3
C12
C13
R10
VLIPO
LIPO
Q1
Q2
Q3
Q8
R1
R2
R3
R15
R16
500MA
R14
CHARGE
U4
VIN VB
PRG GND STA
R17
R18
C15
C16
MBAT
L1
FUSE
C1
R4
Q4
Q5
USB
5V
ID
D-
D+
GND
PWCHK
D8
+3V3C
62,86
2,16,37,46
C3
C5
C6
C8
R51
R50
SDA
SCL
A13
VDD
RE2
RD1
RE4
VBUS
USBID
D-
D+
RD2
VU3V3
VCAP
MCLR
RB1
RB0
AVDD
RA3
RA2
RA15
RA14
RA5
RF8
RA4
RF2
RA10
RE5
RE7
RC1
RC4
RE6
RB13
AVSS
VSS
RD0
RD8
RD12
RD13
RD6
RD7
RD4
RD5
U5a
DIG2
DIG2X
IO8
IO9
IO10
IO12
IO11
IO13
AREF
DIG1
IO7
IO6
IO5
IO4
IO3
IO2
IO1/TX
IO0/RX
R28
D17
R27
D18
C22
D20
D19
RESET
RST
A14
+3V3
C21
C40
D13
ICSP
+V
Vpp
PGC
PGD
A8
A9
RESEN
C23
R39
R40
R35
R37
R36
VCC
IO2
IO4
RST
IO0
CHPD
IO12ESP TX
IO14 RX
IO13 IO15
GND
+3,3V
C2
R43
SD
CD
D0
CLK
CMD
R41
R42
R38
TXD1
RXD1
D15
D14
D13
ISP
R23
R25
RST
R24
G8L
G7L
15,36,45
65,75


+3V3
L4
U3
EN
SW
IN
GND
FB
R12
R13
C14
VLIPO
BCK
U1
IN
OUT
GND
D5
+3V3C
C4
C7
+3V3
Q6
R5
R7
D6
Q7
R6
DZ1
G1
+5V
+3V3
C25
C26
C27
DCVDD
DBVDD
AVDD
CPVDD
SPKV1
SPKV2
C29
SPORP
SPORN
SPOLN
SPOLP
AXIR
AXO1R
AXO1L
AXIL
C31
C33
C34
C30
C38
CPN
CPP
VREF
C36
C39
CPVE
MICBIAS1
MICBIAS2
U6
MIC1N
MIC2N
R47
R44
C35
C28
R46
R45
C32
C24
AUDIO
+3V3
R48
R49
SCL
SDA
MCLK
LRCK
BCLK
ADCDAT
DACDAT
SPKG
CDEPOP
CPREF
CPGND
DGND
AGND
C37
+3V3C
AN1
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
R59
R58
R57
R56
R55
R54
R53
R52
VDD
RB12
RB11
RB10
RB9
RB8
RB7
RB6
RB2
RB15
RD9
RD10
RD11
RB14
D10L
D9L
B15L
B14L
D11L
U5b
AN2
A8
A9
AD8
AD9
AD10
AD11
AD12
AD13
AD14
AD15
R67
R66
R65
R64
R63
R62
R61
R60
A14
A13
RB5
RB4
RB3
RD3
RG0
RG1
RF1
RF0
COMM
RXD1
TXD1
RXD2
TXD2
RXD3
TXD3
C17
C18
C19
C20
XT1
XT2
SOSCO
SOSCI
OSC2
OSC1
AVSS
+5V
LED13
RX
TX
R29
R33
R31
D13
Q9
Q11
Q10
R30
R34
R32
REFSEL
PWR
RST
+3V3
+5V
VIN



nel format di un Arduino/Fishino MEGA. Grazie a uno specifico bootloader, per aggiornare il firmware del modulo WiFi basta una pressione prolungata del tasto reset e la scheda si predisporrà automaticamente in modalità aggiornamento da connettore USB.
Se siete preoccupati per la programmazione di questa schede, potete tranquillizzarvi; un po' grazie alla versatilità dell'attuale IDE Arduino, un po' per il lavoro del team di Chipkit che abbiamo parzialmente sfruttato (anche se con parecchie modifiche), ma anche grazie a librerie software da noi scritte e/o adattate, la programmazione si esegue come su una comune Arduino UNO, ossia tramite l'IDE, nel quale dovete attivare gli strumenti di sviluppo per le PIC ed il gioco è fatto!
Per i componenti aggiuntivi abbiamo predisposto una serie di librerie, alcune delle quali sono in continuo sviluppo, ragion per cui consigliamo di visitare periodicamente il sito web *www.fishino.it*.

SCHEMA ELETTRICO

Lo schema ricalca in gran parte quello della Fishino32, rispetto alla quale utilizziamo un micro con maggior numero di I/O che è pressoché identico a quello della Fishino32, solo che qui è a 100 pin mentre nel modello più "piccolo" è a 64 pin.
Lo stesso microcontroller è disponibile con frequenze di clock che vanno dagli 80 MHz ai 120 MHz, però mentre sulla Fishino32 siamo riusciti a sfruttarlo a 120 MHz, sulla Fishino Shark abbiamo previsto la possibilità di selezionare la velocità tra 5 e 105MHz.
Lo schema elettrico è molto simile a quello della Fishino32, analizziamolo e vediamo le differenze.

STADIO DI CONTROLLO DELL'ALIMENTAZIONE

Questo blocco risulta decisamente più complesso rispetto ai modelli precedenti, per le seguenti ragioni:

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

Iniziamo dalla sezione di controllo dell'alimentazione USB, la quale dev'essere in grado di fornire tensione alla Shark attraverso il connettore USB se connessi come device, mentre deve fornire un'adeguata alimentazione a un device eventualmente connesso alla scheda quando essa opera in modalità host USB. Per la commutazione ci viene in aiuto il piedino che fa capo alla linea USBID, collegata sia al connettore USB che all'apposita porta del controller. Ricordate che i cavetti microUSB sono cablati in modo da avere ad un estremo il pin corrispondente all'USBID connesso a massa (tipo A) e all'altro lo stesso pin scollegato (tipo B); l'apparecchiatura cui si connette l'estremità di tipo A assume il ruolo iniziale di host e contemporaneamente fornisce l'alimentazione all'altra apparecchiatura, connessa, questa, all'estremo di tipo B (che assume il ruolo iniziale di device). L'espressione "ruoli iniziali" è d'obbligo perché in una connessione OTG, i due dispositivi USB possono cambiare ruolo; la specifica garantisce soltanto che la linea verrà alimentata sempre dal device connesso all'estremità A.

Torniamo quindi al nostro schema; vediamo che alla linea USBID è connessa una resistenza di valore elevato (R2, 100 KOhm) verso massa, il cui scopo è solo non lasciare fluttuante la linea in caso di assenza di alimentazione, ed una resistenza (R4) di valore decisamente inferiore (10 KOhm) connessa tra i source dei MOSFET Q4 e Q5. Su questi, grazie ai diodi interni ai MOSFET, è sempre presente una tensione positiva (5V meno la caduta diretta sui diodi) proveniente o dalla presa USB o dall'alimentazione a 5V della scheda Fishino. Abbiamo quindi la certezza che la R4 sia connessa ai +5V in qualsiasi condizione di funzionamento. Se la linea USBID è scollegata da massa (connessione tipo B) questa tensione positiva, attraverso R4, porta in conduzione il MOSFET a canale N Q3, il quale a sua volta porta in conduzione il Q1; contemporaneamente, sempre attraverso R4 i due MOSFET Q4 e Q5, vengono posti in interdizione ed essendo collegati in antiserie, non permettono il passaggio dell'alimentazione dai 5V di Fishino al connettore USB.
Sintetizzando, a USBID fluttuante, quindi quando inseriamo un connettore di tipo B, Fishino non fornisce alimentazione al connettore USB che, per contro, alimenta Fishino attraverso il MOSFET Q1. Otteniamo quindi la prima parte dello scopo desiderato, ovvero che con un connettore di tipo B Fishino ricava l'alimentazione dalla linea USB comportandosi da device vero e proprio. Se invece inseriamo un connettore di tipo A, la linea USBID viene posta a massa, cosa che porta in conduzione Q4 e Q5 e contemporaneamente, sempre attraverso Q3, in interdizione Q1; l'alimentazione fluisce quindi da Fishino al connettore USB e non rientra "dalla porta di servizio" essendo Q1 interdetto.
Notate che nel secondo caso (connessione di tipo A) per poter fornire i 5V alla porta USB, Fishino deve essere alimentata dalla batteria o dal plug VIN, come vedremo in seguito.
Proseguendo l'analisi, notiamo le serie di diodi D1+D4 e D2+D3: i primi due sono per piccole correnti, mentre i secondi sono di potenza. Vediamo inoltre il MOSFET Q2, al quale giunge l'alimentazione della batteria. I diodi D2 e D3, insieme al MOSFET Q1, costituiscono una porta OR di potenza che confluisce nella linea VOR dello schema. Per i diodi, il funzionamento è questo: 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 resta il 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: si può dire che una caduta di tensione di 0,5V su 10V corrisponde al 5%, mentre la stessa caduta su 3,6V incide per circa il 14%, cosa che si riflette sull'assorbimento in corrente come vedremo di seguito, e proprio dove servirebbe una maggior efficienza, ovvero nell'alimentazione a batteria.
Il MOSFET Q2 non ha questo problema, perché diversamente dal diodo non presenta una caduta di tensione minima ma una resistenza di conduzione, molto bassa nel modello utilizzato; 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. Per questo utilizziamo i due diodi per piccole correnti D1 e D4; il funzionamento è il seguente:

- se non è presente un'alimentazione su VIN e nemmeno attraverso VUSB, la resistenza R3 porta a massa il gate di Q1 che quindi conduce e permette il passaggio della tensione della batteria verso VOR;
- se invece è presente una delle due (VIN o VUSB, o entrambe) uno dei diodi D1 e D4 conduce, polarizzando positivamente Q2 che passa in interdizione scollegando quindi la batteria.

CARATTERISTICHE TECNICHE

- **Formato standard Arduino MEGA**
- **Processore PIC32MX470F512**
- **512 kB di ROM**
- **128 kB di RAM**
- **Frequenza di clock di 105 MHz**
- **Interfaccia USB nativa, sia device che host**
- **RTC incorporato**
- **Lettore per schede microSD**
- **Modulo WiFi integrato**
- **Codec audio**
- **Tensione di alimentazione: 3÷20Vcc**
- **Alimentazione a batteria, plug e/o via connettore USB**
- **Funzionamento interno a 3,3 V**
- **Pin DIGITALI 5V-tolerant**
- **Ricarica automatica di una batteria LiPo**
- **Spegnimento da software con processore in standby**
- **Wake-up tramite pin esterno o in tempi prefissati**

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 D1 e D4 ha un nome, BATOFF, perché questo segnale, che è a livello alto se è presente un'alimentazione esterna, verrà utilizzato anche in un'altra sezione del circuito.
Passiamo alla sezione circuitale facente capo ai MOSFET Q6 e Q7; questi servono per scollegare la linea +3V3, ovvero la linea principale a +3,3V, dalla linea +3V3Core, destinata ad alimentare solo il controller. A cosa servono? Semplicemente ad evitare che un'alimentazione destinata al solo microcontrollore, attivata quando tutto il resto è spento allo scopo di mantenere attivo l'RTC interno ed altre funzionalità minime, confluisca nei circuiti che intendiamo tenere spenti. Normalmente, se è presente tensione sui +3,3V, questa confluisce attraverso il diodo interno a Q6, e anche attraverso il Q6 stesso se questo è in conduzione, sull'alimentazione +3V3Core del controller; è questo il funzionamento normale. Invece se togliamo l'alimentazione sui +3,3V, e contemporaneamente portiamo a livello basso la linea OFF, attraverso Q7 portiamo il Q6 ad interdizione e quindi impediamo ai +3V3Core di andare ad alimentare il resto della scheda. La linea OFF viene comandata dal controller, come vedremo in seguito, durante lo spegnimento programmato.
Si possono poi notare la pila al litio di mantenimento G1 ed il diodo D6, che servono a fornire tensione al controller quando manca qualsiasi altra cosa in grado di alimentarlo. Per questo occorre fare attenzione : se togliamo completamente l'alimentazione al Fishino, questa piletta finisce per alimentare tutta la scheda, a meno di non "prendere provvedimenti" attraverso il software portando a livello basso la linea OFF. A differenza quindi delle schede ad 8 bit, dove la piletta serviva solo per la tensione di mantenimento dell'RTC, e la cosa veniva gestita in modo automatico, qui è necessario un intervento del software a supporto, altrimenti la piletta si scaricherebbe in pochi minuti.
Consigliamo quindi di non utilizzarla, e lasciare il mantenimento alla batteria LiPo ricaricabile, a meno che non si sappia come gestire il risparmio energetico della scheda via software.
Per ultima notiamo la linea PWRGOOD, connessa ad un partitore con la massa; questa fornisce un segnale positivo ad 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 stand-by.
Come detto sopra si tratta di una gestione piuttosto complessa, per la quale è stata predisposta un'apposita libreria software, tuttora in fase di sviluppo, per gestire le modalità di stand-by.
Vi consigliamo di guardare il codice e gli esempi della libreria FishinoLowPower per capirne il funzionamento!
L'ultima parte della sezione in esame è quella che ruota attorno al regolatore lineare U1, un semplice regolatore a bassa corrente (max 100 mA circa) e bassissimo consumo, denstinato a fornire la tensione di mantenimento al controller tramite la batteria LiPo esterna.
Questo ha lo stesso effetto della piletta al litio G1, con l'indubbio vantaggio di utilizzare una batteria ricaricabile in grado di fornire una durata estremamente lunga anche in presenza di consumi non proprio trascurabili.
Questa sezione è disattivabile tagliando il ponticello BCK, normalmente connesso. Anche qui valgono (anche se in modo meno pressante) i discorsi sul risparmio energetico fatti prima : se spegniamo via software la scheda, il controller prenderà alimentazione dalla batteria LiPo che quindi tenderà a scaricarsi; inoltre, se superassimo i 100 mA di assorbimento, il piccolo regolatore lineare si "siederebbe", annullando la sua tensione in uscita. È quindi sempre compito del software fare in modo che il controller venga posto in standby o comunque in una delle modalità a basso consumo disponibili.

STADIO DI ALIMENTAZIONE SWITCHING

Questo stadio fa capo agli integrati U2 (SX1308), un convertitore asincrono step-up qui utilizzato in modalità SEPIC in grado di fornire un'uscita a 5V

con tensioni in ingresso variabili da circa 3 a 20 V, ed all'integrato U3, che è un convertitore sincrono step-down, utilizzato per ricavare i 3,3 V di alimentazione per la maggior parte del circuito.
Entrambi questi stadi sono già stati ampiamente descritti per le schede Fishino Mega e Guppy e, per quanto riguarda il funzionamento del SEPIC, nella scheda Torpedo. Le uniche differenze consistono nella presenza del jumper ON, che permette l'inserimento di un interruttore di spegnimento in grado di togliere l'alimentazione a tutta la scheda (fatto salvo quella di mantenimento eventualmente abilitata, come spiegato nel paragrafo precedente), e la linea OFF, già vista prima, in grado di spegnere, se portata a massa, il convertitore switching.
Questa linea è sotto diretto controllo del processore, attraverso un I/O e permette quindi sia lo spegnimento completo della scheda via software, sia l'isolamento della sezione di alimentazione del solo controller dal resto del circuito.
Riassumendo, ponendo la linea OFF a massa viene tolta tutta l'alimentazione alla scheda; ovviamente questa linea deve essere mantenuta a massa dal controller che quindi deve rimanere operativo attraverso una delle due sorgenti di mantenimento (LiPo o piletta al litio); in assenza di queste, portando la linea OFF a massa otterremo solo un reset della scheda ed il conseguente riavvio del tutto.

STADIO CARICABATTERIA

Anche questo stadio, come nelle board precedenti, fa capo all'integrato MCP73831, un completo regolatore di carica integrato per LiPo a cella singola. Qui, l'unico elemento degno di nota è il mosfet Q8, 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 Q8 va in conduzione, portando a massa la resistenza R15 (ed eventualmente la R16 se abbiamo selezionato la carica a 500 mA) attivando quindi la carica. In assenza di tensioni esterne (alimentazione a batteria) la linea BATOFF va a livello basso disattivando la carica.
A completamento della descrizione notiamo le resistenze R17 ed R18, che costituiscono un partitore di tensione in grado di fornire metà della tensione della batteria LiPo, misurabile attraverso una linea del controller per poter segnalare lo stato di carica della stessa. Abbiamo utilizzato un partitore perché la tensione della batteria LiPo, a piena carica, supera i 3,3 V tollerati dagli ingressi analogici del microcontrollore; occorre tenerne quindi conto nella misura.

SEZIONE MICROCONTROLLORE

Arriviamo finalmente al PIC32MX470F512L, un microcontroller a 32 bit capace di elevate prestazioni, consumi decisamente ridotti e con una ricca dotazione di periferiche.
Notiamo innanzitutto due quarzi, uno da 20 MHz e l'altro da 32,768 KHz. Ma perché due quarzi? Ebbene, perché perché il primo fornisce il clock principale, attraverso un PLL interno che lo moltiplica fino a 105 MHz; si tratta di un circuito non proprio a basso consumo, quindi da evitare nelle condizioni di risparmio energetico. Il quarzo da 32.768 Hz 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 stand-by.
Tornando al primo quarzo, accettando una precisione minore sulla frequenza questo avrebbe potuto essere rimosso perché il PIC integra un oscillatore RC piuttosto preciso, ma che ha due limiti:

- non fornisce la precisione del quarzo;
- non è in grado di portare il PIC alla sua massima frequenza di 105 MHz, a causa di vincoli del PLL interno che limitano a 96 MHz.

Il primo punto ha importanza solo se utilizziamo la scheda per misure precise di tempi brevi: ad esempio come frequenzimetro o periodimetro; per i tempi lunghi basterebbe l'altro quarzo.
Il secondo punto è vincolante se vogliamo "spremere" il controller al massimo delle sue possibilità.
Del micro notate le già accennate linee OFF, che gli permette di togliere l'alimentazione principale, PWRGOOD, che consente il controllo dello stato della medesima, ed MBAT che permette la misura della tensione della batteria.
In questa scheda, visto che gli I/O accessibili esternamente sono parecchi, abbiamo dovuto "multiplexare", ovvero sovrapporre, alcune funzionalità su singoli pin: la PWRGOOD, ad esempio, è sovrapposta all'ingresso analogico AD13; se si utilizza quest'ultimo la funzionalità di PWRGOOD non è disponibile e viceversa.

STADIO RESET E LED

Questo stadio è piuttosto interessante perché ci ha permesso di utilizzare un solo pulsante di reset per svariate funzioni, in abbinamento con un apposito bootloader sul controller.
Notiamo subito tre linee che fanno capo allo stadio

di reset:
- MCLR
- REMOTERESET
- LONGRESET

La linea MCLR è il reset vero e proprio e portandola a massa il microcontrollore viene riavviato. Notiamo subito che questo può avvenire sia premendo il pulsante di reset (attraverso il condensatore C22 che trasferisce un breve impulso a zero logico alla linea MCLR), sia portando a massa la linea REMOTERESET, proveniente dal modulo WiFi e che consente quindi a questo di resettare il micro.
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 uno zero logico sulla linea REMOTERESET riavviano la scheda;
- 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 premendo nuovamente e brevemente il pulsante di reset.
Anche qui, per scarsità di pin, abbiamo dovuto condividere la linea LONGRESET con un I/O analogico (l' AD14); nessun problema, in questo caso, visto che premendo il reset l'ingresso non serve!
Al microcontrollore abbiamo assegnato due LED (pilotati attraverso i MOSFET Q10 e Q11 per non caricare eccessivamente gli I/O del PIC), multiplexati con la porta seriale 2, utilizzabili per qualsiasi compito (indicati sulla scheda come RX/PRG e TX, rispettivamente); utilizzando la porta seriale 2, i LED pulseranno in analogia con i dati in ingresso/uscita. Se non usiamo la seriale 2, possiamo gestire i LED via software.
Al microcontrollore troviamo inoltre collegato il solito LED connesso alla porta 13 che viene utilizzato nei test della scheda (programma BLINK), attraverso il mosfet Q9 per non caricare l'uscita D13. A differenza della Fishino32, l'uscita D13 è un normale I/O e NON ha la funzionalità aggiuntiva di clock per l'interfaccia SPI, che è, come per l'Arduino/Fishino MEGA, su altri pin.

STADIO WIFI

Anche questo stadio è già stato descritto nelle precedenti schede, ma qui troviamo una differenza piuttosto consistente, quindi lo rivediamo in dettaglio.
Il modulo sfrutta sempre un firmware da noi realizzato, in grado di comunicare ad alta velocità tramite la porta SPI e non sulla seriale, come d'abitudine nei sistemi che utilizzano l'ESP8266; la differenza di prestazioni, come già notato, è notevole.
In questa scheda, per contro, non vediamo più il connettore ESPCONN al quale eravamo abituati nelle board precedenti ad 8 bit 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.
Come mai questa scelta? Semplicemente perché nella Fishino Shark abbiamo già liberato alcuni degli I/O che venivano usati per comunicare con il modulo e con la scheda SD, che erano il 4, 7 e 10; abbiamo inoltre separato la porta I²C dai due ingressi analogici A4 ed A5, ottenendo quindi 3 porte digitali e due porte analogiche in più a disposizione. Non solo, ma due porte sul connettore ICSP sono anch'esse 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, quasi tutte le linee dell'ESP (salvo l'ingresso analogico) sono ora sotto controllo del processore principale, cosa che ci permette di effettuare le seguenti operazioni direttamente da software, senza impegnare alcun I/O :
- resettare il modulo da software;
- spegnere il modulo da software (cosa che in precedenza richiedeva l'uso di un I/O del Fishino);
- caricare il firmware sul modulo senza dover effettuare collegamenti strani nè caricare sketch particolari;
- ridirigere l'output seriale del modulo sulla seriale USB o su qualsiasi altra seriale, sia software che hardware, disponibile sul Fishino.

Ma perché "quasi" tutte le linee? Anche qui, per risparmiare un I/O sul controller, abbiamo evitato di collegare la linea di RESET dell' ESP8266; la stessa funzionalità si può ottenere tranquillamente tramite la linea ESP_CHPD, che serve anche a mandare in modalità low-power il modulo.
Le nostre librerie software tengono già in conto tutto

questo.
Per il resto, cambia praticamente nulla rispetto alle schede precedenti: la comunicazione avviene attraverso la porta SPI, che fa capo anche alla scheda SD ed ai pins digitali D50, D51 e D53 sul connettore grande in fondo alla scheda.
Visto che la PIC utilizzata ha due interfacce SPI incorporate, verrebbe da chiedersi perché non è stata dedicata una di queste ai moduli inclusi (WiFi e SD) liberando così anche gli I/O D50, D51 e D53 completamente; questo è stato fatto perché la seconda interfaccia SPI è dedicata, in modalità I²S, al codec audio che quando è in funzione la impegna a tempo pieno. In un primo tempo avevamo scelto quella strada, salvo accorgerci in seguito che risultava impossibile, ad esempio, riprodurre un file audio letto da scheda SD, visto che durante la riproduzione l'interfaccia era totalmente impegnata con il codec senza possibilità di accedere alla SD.
Va detto che nelle schede in formato Arduino MEGA i tre I/O suddetti sono comunque dedicati alla porta SPI, e comunque se necessario abbiamo a disposizione i due I/O sulla porta ICSP di cui parlavamo prima, ed eventualmente tutti gli I/O analogici che sono comunque utilizzabili anche in digitale.

STADIO INTERFACCIA MICROSD

Questo stadio è assolutamente identico a quello delle schede già viste; sfrutta l'interfaccia SPI, come il modulo WiFi, mentre anche qui per la selezione della scheda viene utilizzato un I/O interno, lasciando libero quindi l'I/O D4 che prima era necessario per questo scopo.
L'unica grossa differenza rispetto alle schede precedenti è l'assenza degli stadi di conversione livelli (da 5 Volt a 3,3 V) visto che la PIC utilizza i 3,3 volt come alimentazione, rendendo quindi possibile (e più veloce!) un collegamento diretto. L'unico dettaglio da tenere presente nell'esecuzione di sketch esistenti è che 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).

STADIO CODEC AUDIO

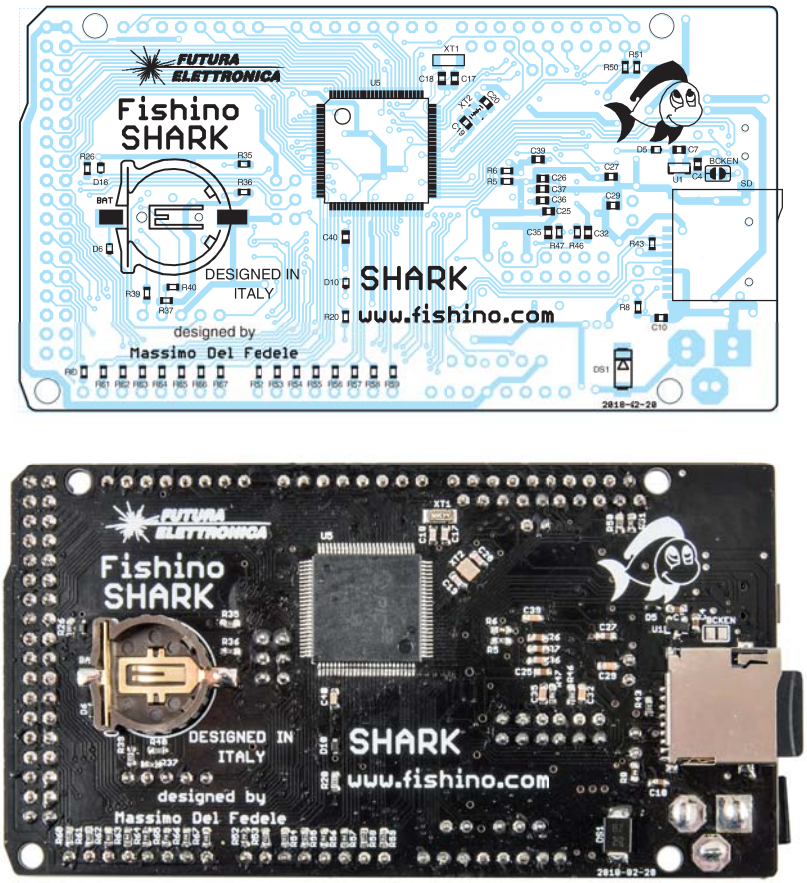
Ed eccoci arrivati alla succosa novità presente nel Fishino32, il codec audio!
Questo si basa su un chip, l'ALC5631Q, utilizzato in molti prodotti consumer (tablet, riproduttori audio, eccetera) e quindi dotato di prestazioni notevolissime pur conservando un prezzo più che accettabile.
Il codec ha le seguenti caratteristiche:

- convertitore D/A (DAC) stereo a 24 bit con un rapporto segnale/disturbo di 100 dbA ed una distorsione totale a -90 db;
- convertitore A/D (ADC) stereo a 24 bit con un rapporto segnale/disturbo di 93 dbA ed una distorsione totale a -88 dB;
- un amplificatore stereo in classe D in grado di erogare 650 mW per canale se alimentato a 3,3V su un'impedenza di 8 ohm e fino a 2,8 W per canale se alimentato (come nel nostro caso) a 5 volt, su un'impedenza di 4 ohm;
- un amplificatore per cuffie stereo, in grado di erogare 45 mW su 16 ohm, che purtroppo non utilizzeremo nella nostra scheda per motivi di spazio;
- 3 ingressi analogici differenziali o un ingresso analogico stereo (noi utilizziamo quest'ultimo);
- 2 ingressi microfonici analogici preamplificati, sia in modalità differenziale (da noi non usata) sia in modalità single-ended;
- un ingresso per microfono stereo digitale (anche questo, non utilizzato sulla nostra scheda sempre per motivi di spazio);
- un'uscita analogica di linea, alla quale è possibile connettere un amplificatore esterno;
- un completo sistema di power management che permette di spegnere i singoli componenti del codec se non utilizzati;
- un equalizzatore digitale a 6 bande audio;
- effetti di pseudo-stereo e di stereo spaziale;
- circuiti di rimozione dei disturbi di accensione e spegnimento negli altoparlanti;
- gestione automatica di livelli (ALC) sia in riproduzione che in registrazione;
- interfaccia standard I²S ad alta velocità (24 bit/ da 8 a 192 KHz sul DAC in riproduzione e 24 bit/ da 8 a 96 KHz sull'ADC in registrazione).

Come si può notare, una serie di caratteristiche impressionanti, che permetteranno di realizzare, senza alcun hardware aggiuntivo che non sia un altoparlante o un microfono, moltissimi progetti che coinvolgono l'audio digitale. Tra questi:

- player audio (wav e Mp3);
- registratore audio;
- riconoscimento vocale;
- sintesi musicale;
- sintesi vocale.

Come vedrete in seguito, abbiamo già implementato un player per files **.wav**, **.mp3** e **.ogg**/**.ogb**; per la registrazione abbiamo a disposizione una libreria in grado di creare files **.wav**. Esiste anche un encoder



Elenco Componenti:

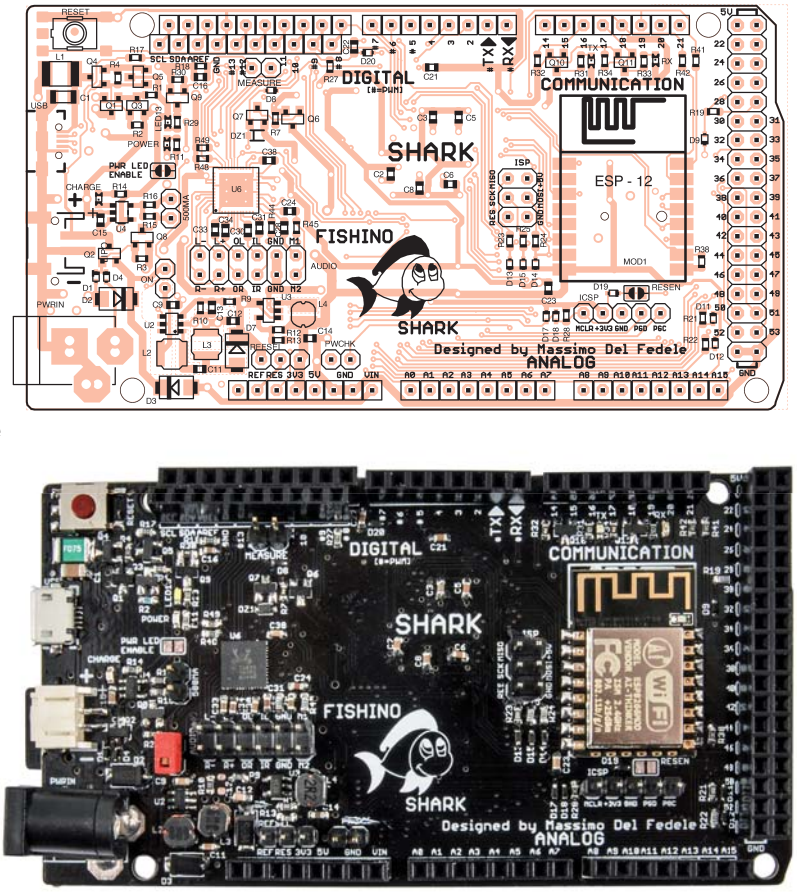
C1: 100 μF ceramico
C2, C3, C5, C6, C8, C10, C16, C40: 100 nF ceramico
C4, C7, C21, C22, C24, C25, C26,
C27, C28, C30, C31, C33, C34, C37: 1 μF ceramico
C9: 22 μF 25 VL ceramico
C11: 4,7 μF 25 VL ceramico
C12: 22 μF ceramico
C13: 360 pF ceramico
C14: 10 μF ceramico
C15, C23, C32, C35, C36: 4,7 μF ceramico
C17÷C20: 22 pF ceramico
C29: 22 μF ceramico
C38, C39: 2,2 μF ceramico
D1: RB521S
D2. D3: SS34SMA
D4÷D6, D8÷D20: RB521S
D7: SS34SMA
DZ1: MM3Z3V3
DS1: SMBJ24CA
Q1: FDN340P
Q2: NTR4171P
Q3: 2N7002
Q4, Q5: FDN340P
Q6: NTR4171P
Q7÷Q11: 2N7002
R1, R2: 100 kohm
R3÷R5, R15, R28, R35÷R38, R43, R48÷R51: 10 kohm
R6÷R8: 220 kohm
R9: 97,6 kohm
R10: 13,3 kohm

MP3, ma le sue prestazioni sono decisamente insufficienti ed è stato incluso solo per curiosità e per poter fare alcune prove; la compressione MP3 è infatti troppo onerosa per essere effettuata in real time dal nostro pur potentissimo controller.
Come già accennato parlando del modulo WiFi, il codec sfrutta un'interfaccia SPI dedicata, utilizzata in modalità I²S, per la gestione della quale abbiamo scritto un'apposita libreria software.
Il codec ha inoltre bisogno di una porta I²C (non necessariamente dedicata solo a lui, ed infatti abbiamo utilizzato la porta I²C standard del Fishino) per controllare tutte le varie funzioni disponibili.
Anche qui abbiamo creato un'apposita libreria in grado di controllare il funzionamento del codec; tramite questa si può, per esempio, decidere se registrare o riprodurre, modificare il volume audio, regolare l'equalizzatore, impostare l'ALC, eccetera.
Come potrete ben immaginare, i controlli sono numerosissimi, quindi la libreria verrà estesa man mano che se ne presenterà la necessità.
Sullo schema elettrico della sezione c'è pochissimo da aggiungere: come potete vedere, oltre al codec abbiamo qualche condensatore di disaccoppiamento sulle alimentazioni e sugli I/O, un condensatore per un convertitore interno destinato ad aumentare la tensione per l'amplificatore delle cuffie (C38) ed un paio di resistenze per la tensione di bias (polarizzazione) dei microfoni. Null'altro!
Torniamo ora un momento sull' interfaccia I²S: si tratta di un'interfaccia basata sull'SPI, prevista espressamente per la trasmissione di flussi audio stereo ad alta velocità; a differenza della SPI e a causa del fatto che i segnali che vi transitano devono essere in tempo reale, è praticamente impossibile utilizzarla in contemporanea su più apparecchiature. L'interfaccia utilizza gli usuali pin SDI (MISO), SDO (MOSI) e SCK (clock) per i dati, mentre il pin di selezione viene utilizzato in modo differente per selezionare il canale audio tra i due trasmessi; è inoltre presen-

R11, R27,
R41, R42: 1 kohm
R12: 475 kohm
R13: 105 kohm
R14, R29,
R31, R33: 470 ohm
R16: 2,7 kohm
R17, R18: 1 Mohm
R19÷R26: 680 ohm
R30, R32, R34: 220 kohm
R39: 3,3 kohm
R40: 3,3 kohm
R44, R45: 680 ohm
R46, R47: 2,2 kohm
R52÷R67: 220 ohm
U1: XC6206P33
U2: SX1308
U3: LC3406
U4: MCP73831
U5: PIC32MX470F512L-120I/PF
U6: ALC5631Q
XT1: 32.768 KHz
XT2: 20MHz
TX: LED blu
RX/PRG: LED arancione
LED13: LED bianco
POWER: LED verde
CHARGE: LED rosso
L1: MF-MSMF050-2

500mA
L2: 6.8 μH
L3: 6.8 μH
L4: 2.2 μH
USB: Connettore micro-USB
SD: Connettore micro-SD
MOD1: Modulo ESP12 con ESP8266
LIPO: Connettore JST 2 vie 2.54 mm
BAT: Porta batterie Ø12 mm per CR1220
PWRIN: Plug alimentazione
RESET: Microswitch

Varie:
- Strip femmina 8 vie (5 pz.)
- Strip femmina 10 vie (2 pz.)
- Strip femmina 2x18 vie
- Strip maschio 2x3 vie
- Strip maschio 2x6 vie
- Strip maschio 2 vie (4 pz.)
- Strip maschio 3 vie
- Strip maschio 5 vie
- Jumper (4 pz.)
- Circuito stampato S1408



te una linea addizionale di clock, utilizzata come riferimento.
Quindi abbiamo, nominando i pin dal lato master, ovvero dal punto di vista del controller:

- MISO o SDI --> ADCDAT, ovvero i dati in ingresso dal convertitore A/D al controller;
- MOSI o SDO --> DACDAT, ovvero i dati in uscita dal convertitore D/A verso la PIC;
- SCK --> BLCK, ovvero il clock per i singoli bit trasmessi;
- SS --> LRCK, ovvero il clock per differenziare canale sinistro e destro;
- MCLK --> clock di riferimento, abitualmente 4 volte il SCK.

Anche qui abbiamo dovuto "multiplexare" gli I/O del codec con altrettanti I/O disponibili sulla porta esterna in fondo al controller, e precisamente gli I/O dal 45 al 49.
Utilizzando l'audio questi I/O non saranno quindi disponibili per le proprie applicazioni; per contro, non utilizzando il codec avremo quegli I/O liberi e, soprattutto, potranno essere utilizzati come un'ulteriore interfaccia SPI (chiamata SPI0 da software) totalmente indipendentemente dalla SPI standard.
I segnali che transitano da/verso il codec sono visibili nella **Fig. 1**. In quest'immagine manca il clock di riferimento (MCLK) che è comunque un clock fisso di frequenza quadrupla rispetto al BLCK.
Lasciamo per il momento la descrizione del codec, che riprenderemo quando tratteremo le librerie software.

GLI HEADER

In questa sezione sono raggruppati tutti i connettori che portano i segnali verso l'esterno della scheda, quindi gli header laterali a standard Arduino , quello doppio in fondo alla scheda, il connettore ISP (che su Fishino Shark non è utilizzato per programmare la scheda, ma inserito per compatibilità con gli

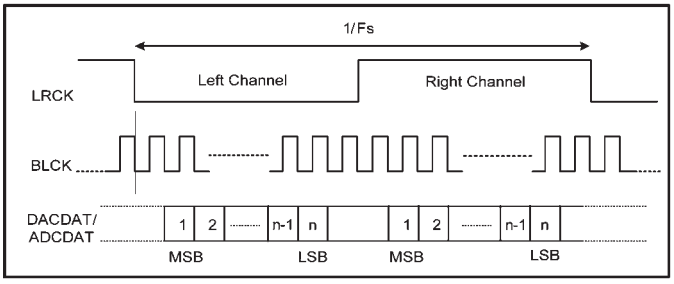


***Fig. 1** - Temporizzazioni dei segnali del codec.*

shields esistenti), il connettore ICSP (In Circuit Serial Programming) che serve per programmare la scheda senza passare per il bootloader (ad esempio per programmare il bootloader stesso), e sul quale sono disponibili due I/O digitali aggiuntivi, ed il connettore USB.
Come accennato elencando le caratteristiche della scheda, il PIC impiegato nella Shark ha molti, ma non tutti, i pin di I/O 5V tolerant, ovvero in grado di accettare 5V in ingresso.
Dove è stato 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 680 ohm in serie all'ingresso e in un diodo in grado di scaricare l'eccedenza di tensione sulla linea a 3,3 volt, limitando quindi la tensione in ingresso a tale valore.
Sui pin analogici, che non sono 5V tolerant (proprio a causa della circuiteria analogica interna al controller) abbiamo inserito solamente delle resistenze in serie per limitare un eventuale sovracorrente nel caso vi si inseriscano per errore dei valori superiori ai 3,3 volt.
Attenzione, questo non significa che è possibile inserire 5 volt su quel connettore! Le resistenze offrono un minimo grado di protezione, che dovrebbe proteggere il controller per brevi periodi se per errore vi si inseriscono i 5 volt, ma non bisogna farne conto ed utilizzarli stabilmente con quei valori di tensione!
L'integrato dispone infatti di diodi di protezione al suo interno, che fanno le veci di quelli da noi inseriti esternamente nei pin digitali protetti; questi diodi sono però in grado di assorbire una corrente molto limitata e per brevi periodi di tempo, quindi non ci si deve fare assolutamente conto.
Nella sezione header troviamo anche un connettore denominato REFSEL, che serve per selezionare la tensione che verrà connessa al pin VREF; è infatti possibile farvi giungere i 5V, nel caso si utilizzino shields "normali" funzionanti a quella tensione, oppure i 3,3V destinati a shield a bassa tensione come quelli utilizzati nell'Arduino DUE.

REALIZZAZIONE PRATICA

Fishino Shark è una scheda realizzata completamente con componenti SMD, alcuni di dimensioni particolarmente ridotte e popolata su entrambi i lati.
Pur utilizzando un PCB a due facce, quindi teoricamente realizzabile con metodi artigianali, ha molte piste sottili e moltissime vie (connessioni tra le due facce) che ne rendono la realizzazione.
Invece il montaggio non è particolarmente complesso, salvo per il codec audio ALC5631Q, che, essendo in package QFN, ha i piedini sotto il corpo, anche se lungo il bordo ed è praticamente impossibile da saldare con un comune saldatore a stilo, ma richiede la stazione ad aria calda.
Suggeriamo quindi ai meno esperti di acquistare la scheda già pronta all'uso, perché è più conveniente.
Sul nostro sito mettiamo comunque a disposizione tutti i file necessari a realizzare il PCB.

CONCLUSIONI

Concludiamo qui questa prima puntata dove abbiamo introdotto le peculiarità della nostra scheda Fishino Shark, descrivendone l'hardware ed il funzionamento.
Nella prossima puntata vi spiegheremo come gestirla dall'IDE di Arduino e vi proporremo un primo sketch di prova per testarne le potenzialità.

per il MATERIALE

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

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