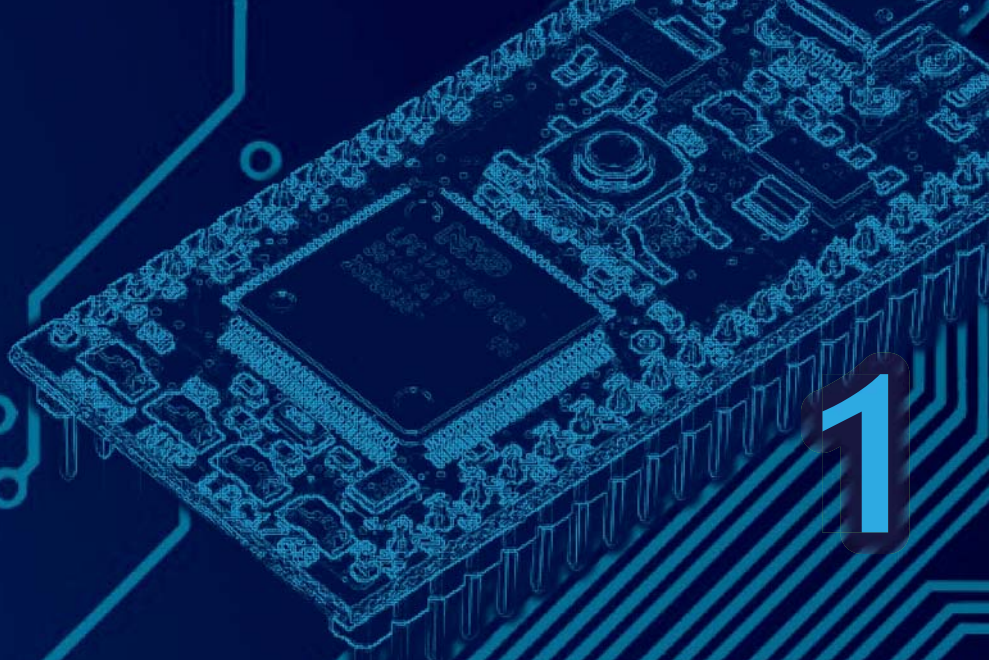


È un'alternativa più avanzata alla popolare scheda embedded ad 8 BIT (Arduino) e può contare sulla potenza di calcolo di MCU ad architettura ARM Cortex; più compatta e prestante, vi mostreremo passo passo come è possibile importare il codice scritto per Arduino e come usarne gli shield. Prima puntata.



Conoscere e usare

mbed

dell'ing. Vincenzo Mendola

Negli ultimi due anni si è fatto un gran parlare di Arduino, tanto che chi ha approcciato il mondo dell'elettronica con questa board o coloro che l'hanno conosciuta per hobby, probabilmente non si sono accorti che sulla scia di Arduino sono nate tante altre schede embedded; non parliamo degli innumerevoli cloni e varianti più o meno compatibili (Microchip, ad esempio), ma di un sistema di sviluppo e prototipazione che non tutti conoscono ma che è molto più potente e potenzialmente più versatile di Arduino, anche se magari meno immediato da utilizzare in maniera proficua da parte dei neofiti: mbed. Disponibile da alcuni anni e

sicuramente meno pubblicizzata, mbed è nata da un'idea (parliamo del 2005) di due dipendenti di ARM, anch'essi spinti dal desiderio, come nel caso del team di Arduino, di rendere disponibile a tutti gli appassionati di elettronica e autocostruzione dotati di conoscenze tecniche e di programmazione essenziali, un solido supporto hardware su cui sviluppare in maniera rapida ed efficace le proprie idee; da qui deriva la definizione da essi utilizzata di "tool for rapid prototyping with microcontrollers" (punto [1] dei Riferimenti).

Nel corso vi mostreremo come utilizzare gli shield di Arduino e come fare il porting del relativo codice su questa piattaforma.

mbed è attualmente disponibile in due versioni,

Fig. 1

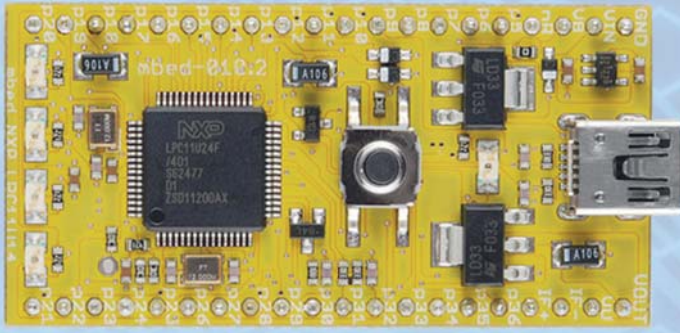
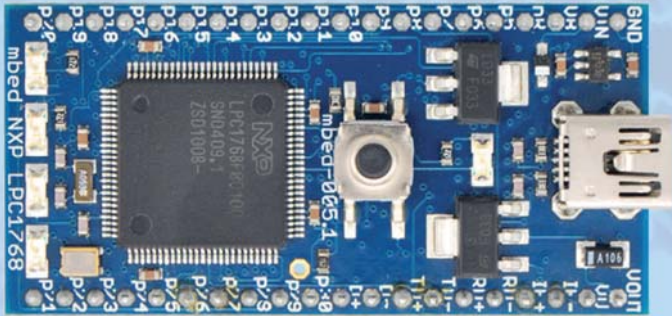


Fig. 2



una con microprocessore Cortex M0 (<http://mbed.org/handbook/mbed-NXP-LPC11U24>) visibile in Fig.1 e l'altra con Cortex M3 (<http://mbed.org/handbook/mbed-NXP-LPC1768>) visibile in Fig. 2.

A queste si è aggiunta recentemente una terza versione, il cui hardware non è stato sviluppato direttamente dal team mbed ma dalla Freescale: questa versione è commercializzata come "mbed ready", dato che per utilizzarla come una board mbed è sufficiente aggiornarne il firmware.

Questa scheda ha un form factor differente dalle altre due, dalle quali differisce per alcune peculiarità; per questa ragione abbiamo

NXP LPC810: il primo Cortex M0+ in un package 8 DIP, al costo di una MCU a 8 BIT come l'ATTINY 85.



deciso di dedicarle un'intera puntata delle successive che formano questo corso.

PERCHÉ 32 BIT

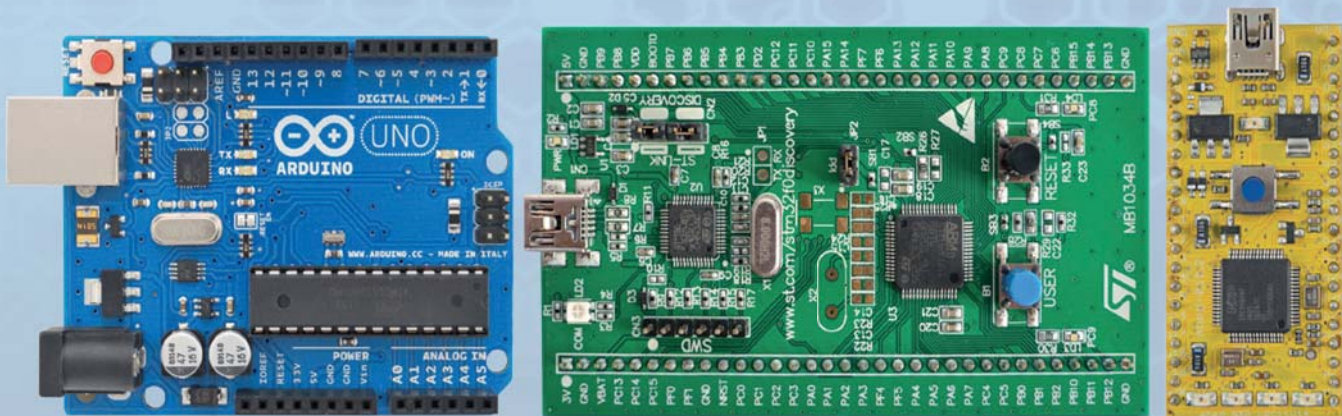
La drastica diminuzione dei costi dei semiconduttori e dei microcontrollori in particolare, ha determinato la rapida diffusione di vari prodotti, rendendo disponibile la potenza dei micro a 32 bit anche agli appassionati, oltreché ai professionisti.

Ma quali sono realmente i vantaggi dei microcontrollori a 32 bit? Ebbene, non esiste una risposta univoca, perché nel confrontare i sistemi a 8 e 16 bit con quelli a 32 andrebbero bilanciati i vantaggi con gli svantaggi delle differenti architetture, perciò volendo rispondere in maniera un po' semplicistica e grossolana, possiamo prendere spunto da quanto riportato dalla stessa ARM (punto [2] dei Riferimenti) nella pagina di presentazione dei propri microcontrollori Cortex-M (le ultime due considerazioni sono chiaramente dirette più alla tecnologia ARM che all'architettura a 32 bit).

In particolare, l'architettura a 32 bit offre:

- maggiore efficienza energetica; essendo più veloci, i microcontrollori possono eseguire le stesse operazioni richieste dai micro a 8 e 16 bit in meno tempo e quindi tornare più rapidamente e per tempi più lunghi in una condizione di funzionamento a basso consumo (sleep mode) ;
- codice più compatto, grazie all'alta densità del set di istruzioni, in grado di effettuare un maggior numero di operazioni per byte;

Fig. 3



- facilità d'uso, in virtù della portabilità del codice, reso compatibile con una vasta gamma di prodotti di brand differenti (grazie alla standardizzazione dell'architettura assicurata dalla ARM, che ha permesso anche alle software-house che si occupano della realizzazione dei tool di sviluppo software l'unificazione di molti dispositivi in un unico prodotto);
- migliori prestazioni garantite dalla potenza dei Cortex-M, caratterizzati da una più elevata potenza di calcolo per MHz di clock rispetto ai micro a 8 e 16 bit.

A queste motivazioni va aggiunto il fatto che oggi possiamo contare su un'ampia disponibilità a costi relativamente bassi dei micro a 32 bit, il che ha reso conveniente usare microcontrollori del genere anche in ambiti fino a poco tempo fa tipicamente adatti ai micro ad 8 bit.

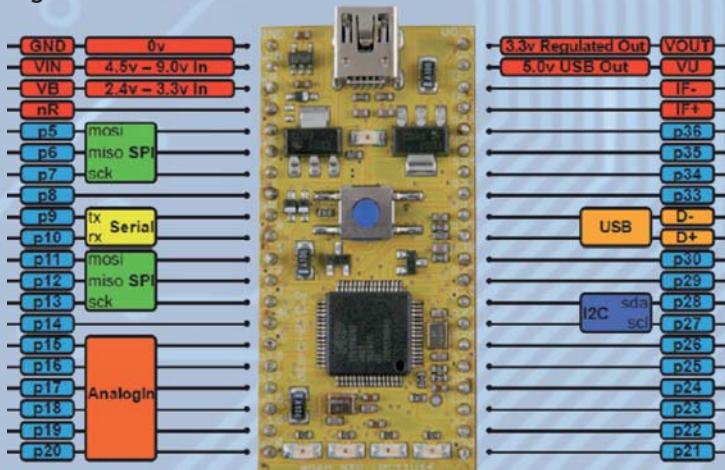
Come si vede in Fig. 3, la scheda mbed è grande meno di 1/3 della Discovery e dell'Arduino, grazie ad una miniaturizzazione spinta e ad un'accurata ingegnerizzazione. Questa caratteristica (non è di certo l'unica) rientra tra gli elementi che determinano la grande differenza di prezzo tra una Discovery F0 con microcontrollore ST STM32F051 e l'MBED LCP11U24, entrambe dotate di Cortex-M0 e quindi con prestazioni e caratteristiche comparabili.

Come molti di voi sanno, ARM non produce direttamente i propri processori, ma ne sviluppa la tecnologia che poi viene venduta sotto licenza ai produttori di semiconduttori, i quali la implementano nei propri prodotti: per questo, in alcuni microcontrollori leggiamo, tra le caratteristiche, "basato su tecnologia ARM" o "con tecnologia ARM".

Tra i produttori che usano ARM possiamo citare Texas Instruments, Toshiba, Analog Devices, NXP, ST, Atmel, Samsung, Freescale, Fujitsu, Renesas e Broadcom.

Per l'mbed sono state scelte due MCU della NXP, che in entrambi i casi danno il nome al dispositivo: si tratta delle LPC11U24 e LPC1768. La NXP LPC11U24 si basa "sola-mente" su di un core ARM® Cortex™-M0 low-power a 48 MHz, con 8 kB di RAM e 32 kB di memoria Flash; le periferiche in dotazione (Fig. 4 e punto [3] dei Riferimenti) sono

Fig. 4



un USB di tipo esclusivamente "Device" (cioè può solo essere gestita da un host e non collegarsi ad altri apparati USB analoghi), 2 bus SPI, una I2C, un'UART, 6 ADC a 12 bit, più 30 pin che possono lavorare come GPIO (General Purpose Input/Output).

Il "cuore" dell'mbed NXP LPC1768 MCU è invece costituito da un ARM® Cortex™-M3, Core a 96 MHz, dotato di 32 kB di RAM e di 512 kB di Flash. Le periferiche disponibili in questo caso (Fig. 5) includono anche una porta Ethernet, USB funzionanti sia come Host che come Device, 2 bus SPI, 2 I2C, 3 UART, una porta CAN bus, 6 moduli PWM e altrettanti ADC, DAC, più 26 GPIO (le caratteristiche complete si possono consultare alla pagina web raggiungibile dal link al punto [4] dei Riferimenti). La MCU LPC1768 ha anche un Real Time Clock integrato.

Entrambe le MCU usate nell'mbed hanno un package DIP a 40 pin, con passo 0,1", che misura soltanto 54x26mm. Per ulteriori dettagli

Fig. 5

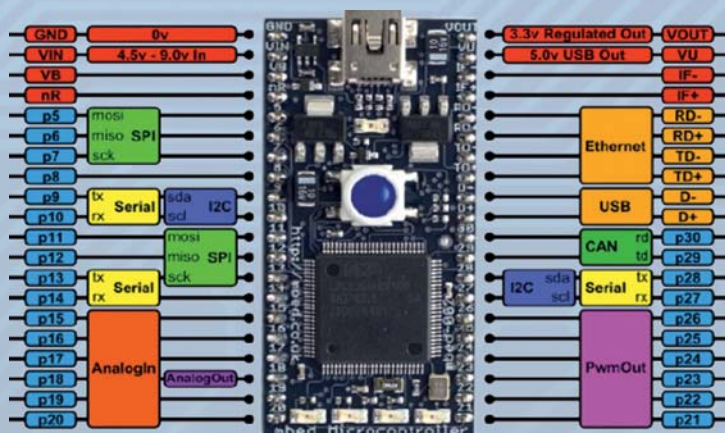
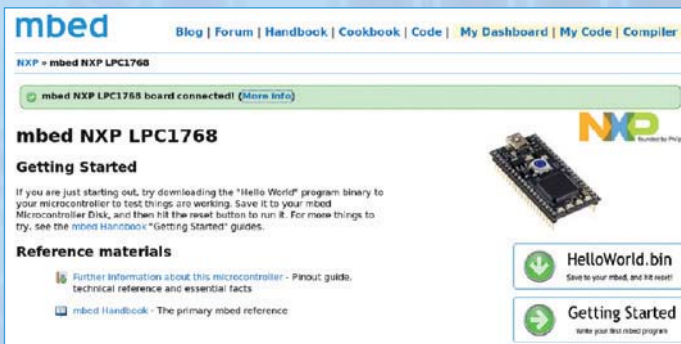


Fig. 6



Fig. 7



e specifiche inerenti l'mbed si faccia riferimento al sito ufficiale e ai link web presenti ai punti [5] [6] [7] dei Riferimenti. La LPC11U24 è principalmente adatta ad applicazioni che richiedono bassi consumi (la MCU usata consuma 1÷16 mA) mentre la seconda è consigliata in tutte quelle applicazioni in cui le prestazioni e la connettività sono caratteristiche di primaria importanza e i consumi (contenuti in 60÷120 mA) non costituiscono un fattore critico. Il maggior consumo del chip NXP LCP1768 si fa notare, durante il normale funzionamento, allorché è facilmente riscontrabile un leggero aumento di temperatura della superficie, dovuto al gran numero di periferiche in funzione, le quali determinano un complessivo, discreto consumo di corrente. Alla luce di quanto detto, prima di scegliere la propria versione di mbed consigliamo di

valutare bene le proprie necessità e considerare le possibili esigenze future, tenendo presente che la maggiore completezza e complessità circuitale dell'LPC1768, caratterizzata dalla disponibilità di molte funzionalità, è controbilanciata da maggiori consumi, fattore cruciale nelle applicazioni portatili in cui l'autonomia è un elemento determinante. La valutazione dei futuri sviluppi di un'applicazione è anch'essa un fattore chiave, perché magari scegliendo la versione basata su LPC11U24 si spende inizialmente meno, ma se un'applicazione richiede l'aggiunta di una periferica non implementata nella MCU, che va quindi aggiunta a parte (ad esempio un'interfaccia Ethernet, una connessione USB di tipo Host invece che Device, oppure un RTC) è facile che i chip e il circuito da aggiungere facciano costare il tutto più della versione mbed più completa.

Le schede mbed sono entrambe plug and play: basta collegarle al PC mediante il cavo USB in dotazione e sono subito pronte a funzionare. Appena connesse, un LED blu posizionato vicino al connettore mini-USB inizierà a lampeggiare velocemente, per poi rimanere acceso una volta effettuato in maniera stabile il collegamento con il computer. Le mbed vengono riconosciute dal PC (lo abbiamo testato sia su Windows che su Linux) come memoria flash (USB Mass Storage Device) da 2 MB di capacità. Esplorandone il contenuto, vediamo che in questa memoria è presente il file *mbed.htm*, aprendo il quale il browser del sistema operativo darà accesso a una pagina web (Fig. 6) dalla quale potrete registrare gratuitamente il vostro dispositivo (<https://mbed.org/account/login/?next=/>); la procedura richiede, chiaramente, la connessione del PC ad Internet.

A questo riguardo va detto che, come segnalato anche dallo stesso team di sviluppo di mbed (punto [8] dei riferimenti) e personalmente sperimentato dall'autore, in alcuni casi il cavo USB fornito di serie nella confezione può creare problemi come il mancato ricono-

Fig. 8



Fig. 9



Fig. 10

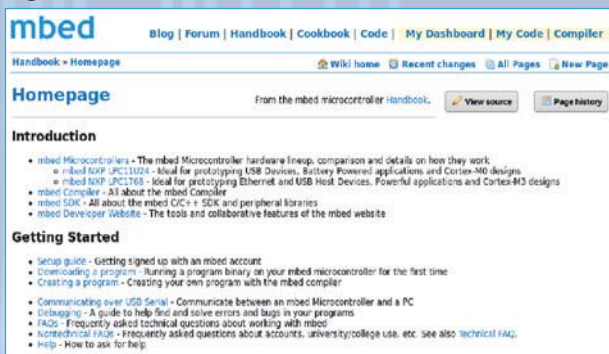


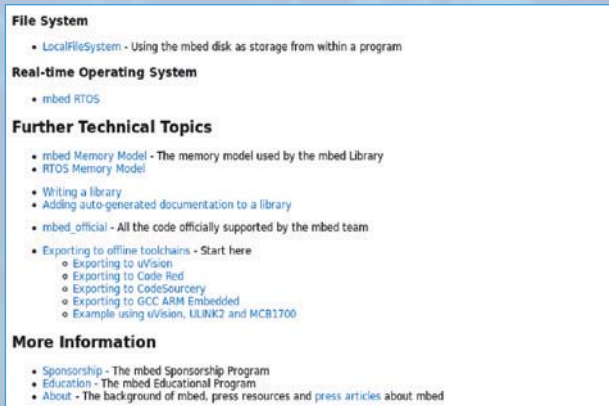
Fig. 11



Fig. 12



Fig. 13



scimento della memoria Flash dell'mbed: ciò può dipendere dal chipset USB presente sulla scheda madre del computer, in quanto alcuni modelli sono particolarmente sensibili alla qualità del cavo. In questo caso è necessario cambiare cavo, optando per uno più corto e munito della ferrite di blocco della RF, ma soprattutto di qualità elevata (per qualità, si intende a bassa attenuazione e diafonia). Una volta effettuata la registrazione ed effettuato l'accesso, ci ritroveremo nella pagina mostrata in Fig. 7, nella quale in alto è indicato il modello di mbed che stiamo utilizzando (nel caso specifico, una LPC1768); inoltre ci viene segnalato che la scheda mbed è correttamente connessa al PC. Cliccando su *more info* ci verranno mostrate ulteriori informazioni sulla nostra scheda: revisione, numero di serie, batch e versione firmware, oltre all'indicazione sullo stato di aggiornamento di quest'ultimo (Fig. 8).

Torniamo ora alla pagina di Fig. 7: da qui possiamo raggiungere le principali sezioni del sito ed iniziare a lavorare con la nostra mbed. Per prima cosa proviamo a caricare il nostro primo programma tramite il link visibile in Fig. 9; una volta salvato il file "HelloWorld.bin" nella memoria del micro, resettiamo quest'ultimo premendo il pulsante presente sul lato superiore della scheda. Dopo pochi istanti, il primo LED in basso a sinistra inizierà a lampeggiare. La procedura di download qui descritta è dettagliata anche alla pagina web <https://mbed.org/handbook/Downloading-a-program>.

Seguendo il link "Getting Started" posizionato subito sotto quello appena utilizzato per caricare HelloWorld.bin, ci verranno fornite maggiori informazioni circa l'uso del nostro sistema (Fig. 10, 11, 12, 13).

Procediamo per gradi: l'IDE degli mbed è rilasciato solo in versione cloud (gratuita), una volta effettuato il login; tale scelta pare essere stata inizialmente dettata dalle difficoltà incontrate dal team di sviluppo nell'installare il software necessario per la programmazione e lo sviluppo del codice sui computer delle scuole e delle università, a causa delle forti restrizioni che spesso vengono implementate dai responsabili IT per garantire la sicurezza dei sistemi informatici, utilizzati da giovani curiosi e con tanta voglia di sperimentare,

che a volte possono causare inconsapevolmente gravi problemi. L'IDE è raggiungibile all'indirizzo <https://mbed.org/compiler/>, oppure più semplicemente, servendosi del link in alto a destra indicato come "Compiler". Nella pagina "Creating a program" (<https://mbed.org/handbook/Creating-a-program>) sono riassunti i principali passaggi da seguire per creare e caricare in mbed il nostro primo programma ed è mostrato il codice di "HelloWorld" (Fig. 14).

In alto a destra c'è un link chiamato "Import this program" che ci permette di caricare il codice nell'IDE. Nella colonna di sinistra "Program workspace" troviamo il programma appena caricato, il cui codice è visibile nell'area centrale dell'IDE, selezionando *main.cpp* (Fig. 15).

Il file "mbed", visibile nello spazio di lavoro sotto a *main.cpp*, e identificato tramite un'icona a forma di ingranaggio, è la libreria che contiene tutte le istruzioni per controllare e far funzionare mbed.

Veniamo al codice che fa lampeggiare il LED1 (Fig. 16): la prima riga contiene l'istruzione "#include mbed.h" che, come abbiamo appena visto, dichiara al compilatore di includere la libreria delle funzioni base del nostro dispositivo; la riga successiva contiene il codice "DigitalOut myled(LED1);" che definisce il pin collegato al LED1 come uscita digitale, associandogli l'etichetta "myled".

Le righe contenute all'interno di "int main()" tra le due parentesi graffe sono quelle che fanno lampeggiare il LED: "while(1)" definisce un loop infinito per le istruzioni seguenti

che sono "myled = 1;" che accende il LED1, mentre "myled = 0;" riporta l'uscita a questo associato a un livello basso, spegnendolo. Questi due comandi sono intervallati da "wait(0.2);" che introduce un ritardo di 0,2 secondi. Terminata l'esecuzione dell'ultima istruzione, il codice si ripete indefinitamente fintantoché l'mbed è alimentato. Il codice appena visto è funzionalmente equivalente a quello utilizzato da Arduino per far lampeggiare il LED 13 presente sulla scheda (<http://arduino.cc/en/Tutorial/Blink>), come si vede nel Listato 1.

Notiamo subito che nel codice di "HelloWorld" manca il "void setup()" che in questo ambiente non viene utilizzato, mentre il "void loop()" è diventato "int main()" il cui contenuto viene eseguito in loop dall'istruzione C "while(1){}"; notiamo anche che "pinMode(led, OUTPUT);" diventa l'equivalente "DigitalOut myled(LED1);" e che "digitalWrite(led, HIGH);" corrisponde a "myled = 1;". Inoltre, "digitalWrite(led, LOW);" è diventato "myled = 0;". Infine, il comando "delay(1000);" è stato tradotto in "wait(1);". L'mbed utilizza la funzione "wait()" il cui argomento rappresenta il numero di secondi di ritardo da introdurre (con risoluzione al microsecondo), mentre "wait_ms" e "wait_us" hanno come argomento rispettivamente i millisecondi o i microsecondi che verranno lasciati trascorrere prima di eseguire l'istruzione successiva.

VISUALIZZARE I DATI SULLA COM VIRTUALE

Prima di passare ad un esempio più comples-

Fig. 14

```

HelloWorld - main.cpp
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}

```

Fig. 15

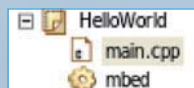


Fig. 16

```

main.cpp x
1 #include "mbed.h"
2
3 DigitalOut myled(LED1);
4
5 int main() {
6     while(1) {
7         myled = 1;
8         wait(0.2);
9         myled = 0;
10        wait(0.2);
11    }
12 }
13

```

Listato 1

```

void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}

```

so, che nel nostro caso prevede l'acquisizione del segnale di un trasduttore di temperatura analogico tramite l'ADC del micro, spieghiamo come si possono visualizzare i dati che viaggiano sulla seriale virtuale del nostro mbed, emulata dalla connessione USB. A riguardo, sul sito mbed esistono numerose pagine dedicate che spiegano nel dettaglio come utilizzare la seriale via USB: le trovate ai link descritti nei punti [9] e [10] dei *Riferimenti*. Questa operazione non può essere eseguita direttamente tramite l'IDE, come siamo abituati a fare con Arduino, ma richiede un ulteriore software emulatore di terminale da far girare sul PC; l'handbook consiglia di usare Teraterm, ottimo e versatile, nonché il diffusissimo e conosciutissimo PuTTY, ovvero, anche Terminal by Bray (punto [11] dei *Riferimenti*). Sono tutti e tre adatti allo scopo e l'utilizzo di uno rispetto all'altro è da ricondursi esclusivamente a preferenze personali. In realtà la scelta non è limitata a questi tre software, perché potete usare qualunque altro programma in grado di gestire le connessioni e le comunicazioni sulle seriali (virtuali o fisiche) del PC.

Vediamo più nel dettaglio i primi due, i quali sono certamente quelli che offrono l'interfaccia utente migliore; quanto a Terminal di Bray, funziona in modo analogo ai precedenti: si seleziona la COM da monitorare e i dati seriali vengono mostrati in un'apposita area presente nella schermata principale del software, nella quale si impostano i vari parametri della connessione.

TERATERM

Accedendo al link indicato al punto [11] dei *Riferimenti*, scarichiamo l'ultima versione del software da sourceforge (<http://sourceforge.jp/projects/ttssh2/releases/>). Il programma è

disponibile in due versioni: quella eseguibile, da installare nel PC, e la versione compressa, che non necessita di installazione. Spieghiamo come servirsi della seconda, dato che la prima ha una procedura di installazione guidata e quindi si spiega da sé: una volta scompattato in una cartella di nostra scelta (mantenendo comunque la struttura originale delle sottocartelle), eseguiamo cliccando due volte sull'icona del file *ttermpro.exe*, allorché si aprirà la finestra riportata in Fig. 17, dove selezioniamo "Serial" e la corrispondente COM (parliamo della COM virtuale corrispondente all'USB) cui è assegnata la nostra mbed. In caso di dubbio è possibile verificare quale è la COM virtuale, accedendo alla pagina del sistema operativo che ci elenca le periferiche del computer (su Windows 7 è raggiungibile da *Pannello di controllo\Sistema e Sicurezza\Sistema* selezionando *Gestione dispositivi* e da qui *Porte COM e LPT*). La scheda mbed viene identificata come *mbed Virtual Serial Port (COMx)*, dove *x* è il numero assegnato alla COM virtuale e corrisponde alla USB presente ai terminali 31 e 32 degli mbed.



Fig. 17

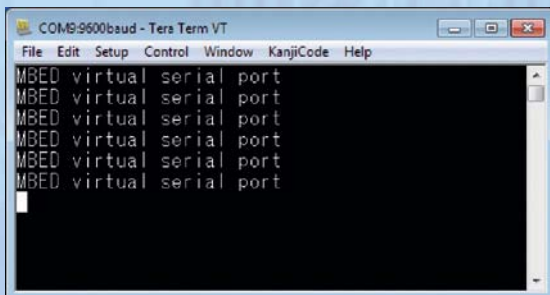


Fig. 18

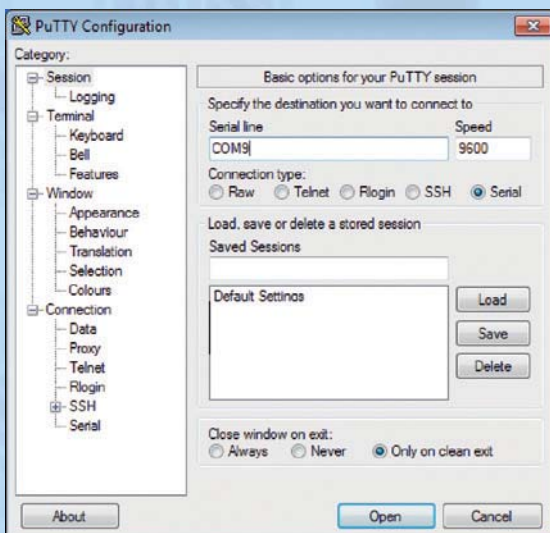


Fig. 19

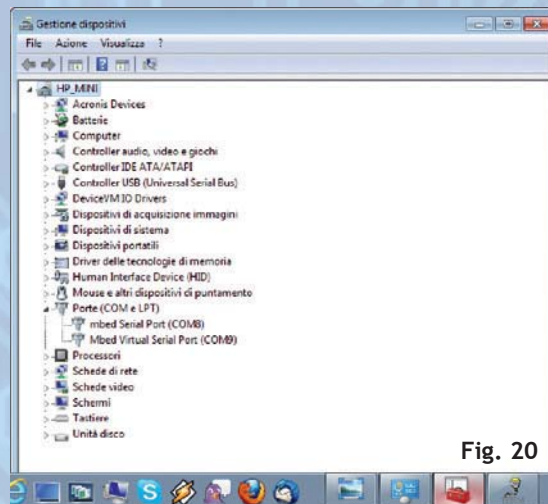


Fig. 20

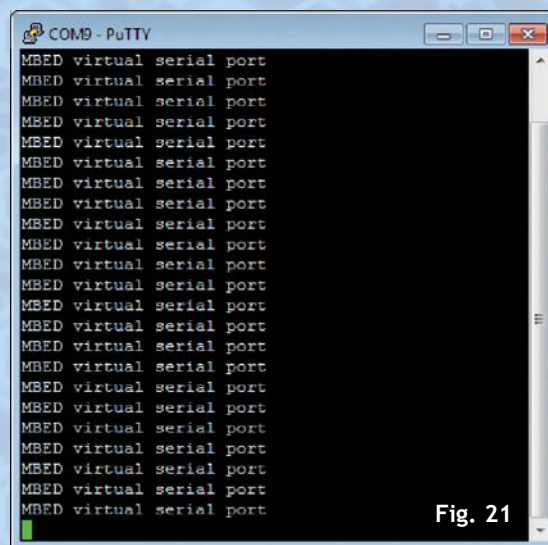


Fig. 21

Ora carichiamo sull'mbed il semplice codice mostrato nel **Listato 2**.

Sulla schermata del terminale avremo quanto riportato nella **Fig. 18**. La scritta si ripeterà finché non chiuderemo la connessione o non toglieremo alimentazione all'mbed.

Per iniziare una nuova connessione con TeraTerm si deve selezionare nel menu "File" "New connection", mentre per terminarla basta selezionare "Disconnect" (sempre da "File").

PuTTY

PuTTY è sicuramente il più conosciuto tra i client telnet e SSH del mondo open source; si scarica in FTP dal sito <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>, lo stesso nel quale si possono trovare anche i file sorgente. Lanciando "putty.exe" con un semplice doppio clic (non è necessaria alcuna installazione) si apre la finestra di dialogo, nella quale basta selezionare la COM in maniera analoga a quanto fatto precedentemente con Teraterm (la **Fig. 19** mostra le impostazioni del caso) e il programma è pronto per svolgere il proprio compito. La **Fig. 21** mostra una tipica schermata di ter-

Listato 2

```
#include "mbed.h"
#include "USBSerial.h"

//Virtual serial port over USB
USBSerial serial;

int main(void) {

    while(1)
    {
        serial.printf("MBED virtual serial port\r\n");
        wait(1);
    }
}
```

Listato 3

```
#include "mbed.h"

Serial pc(USBTX, USBRX); // tx, rx

int main(void) {while(1){
    pc.printf("MBED serial port\r\n");
    wait(1); }
}
```


Listato 4

```
#include "mbed.h"

Serial pc(USBTX, USBRX); // tx, rx
PwmOut led(LED1);

float brightness = 0.0;

int main() {
    pc.printf("Press 'u' to turn LED1 brightness up, 'd' to turn it down\n");

    while(1) {
        char c = pc.getc();
        if((c == 'u') && (brightness < 0.5)) {
            brightness += 0.01;
            led = brightness;
        }
        if((c == 'd') && (brightness > 0.0)) {
            brightness -= 0.01;
            led = brightness;
        }
    }
}
```

minale del software PuTTY, con il dettaglio dei dati scambiati sulla seriale. Se colleghiamo anche la mini-USB di cui ci serviamo per caricare il codice sul nostro mbed, vediamo che questa verrà riconosciuta come un'ulteriore porta COM (Fig. 20) virtuale, identificata in questo caso come *mbed Serial Port*.

Vediamo dunque come utilizzare anche questa seriale: possiamo farlo mediante il semplice codice d'esempio riportato nel **Listato 3**. Otteniamo così un risultato analogo a quello del caso precedente (**Listato 2**) ma questa volta non abbiamo fatto uso della libreria "USBSerial.h", di conseguenza il comando `serial.printf()` è diventato `pc.printf()` ("r" e "\n" hanno la funzione rispettivamente di "ritorno carrello" e "nuova riga").

Prima di chiudere questa prima puntata del corso, proviamo ad eseguire un programma più articolato (**Listato 4**).

Questo codice, presentato nella pagina dedicata alla seriale del mbed (link di cui al punto [9] dei *Riferimenti*) ci mostra un esempio di comunicazione bidirezionale sulla COM del nostro mbed; come indicato sul terminale, premendo il tasto "u" si aumenta la luminosità (controllata in PWM) del LED1, mentre premendo "d" la luminosità stessa diminuisce. Bene, qui si conclude la prima parte del corso su mbed; in attesa di riprendere il discorso (lo faremo nel prossimo fascicolo di *Elettronica In*) speriamo di aver stimolato sufficientemente la vostra curiosità spingendovi a provare questa interessante scheda embedded.

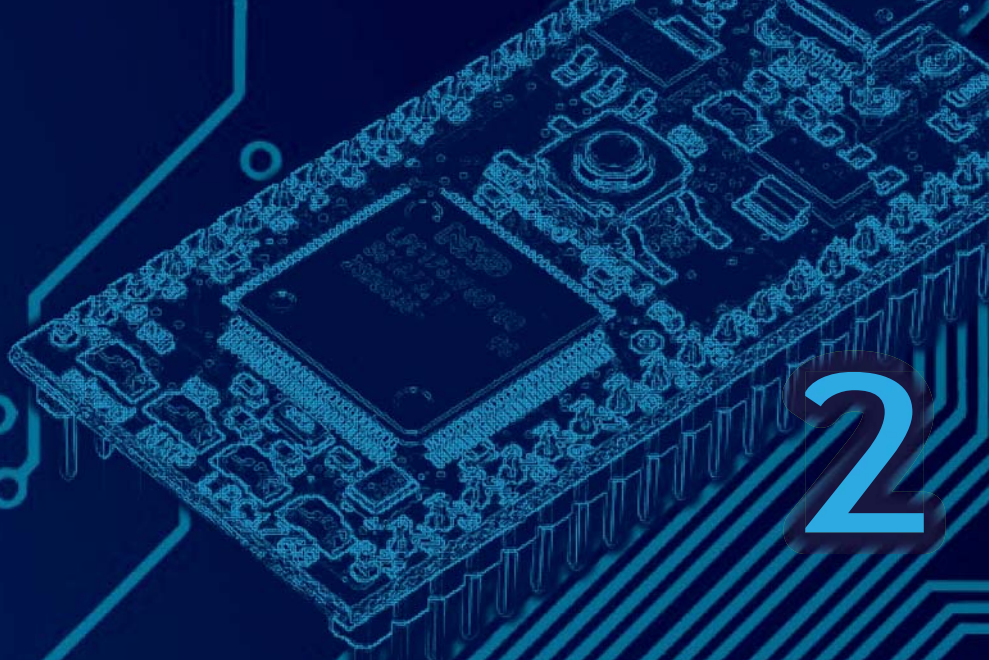
Per ulteriori dettagli relativi alle istruzioni, alle librerie e alle funzionalità e su come implementarle, è possibile fare riferimento alla sezione "Handbook" (<https://mbed.org/handbook/Homepage>), al "Cookbook" (<https://mbed.org/cookbook/Homepage>) e al forum (<https://mbed.org/forum/>).

Nella prossima puntata illustreremo ulteriori dettagli del codice e presenteremo altri esempi pratici che mostreranno come sia possibile effettuare il porting (cioè la portabilità) del codice di Arduino in ambiente mbed e in che modo utilizzare gli shield di Arduino con i nostri dispositivi basati sui processori Cortex NXP. ■

Riferimenti

- [1] <http://mbed.org/handbook/Founders-interview>
- [2] <http://www.arm.com/products/processors/cortex-m/index.php>
- [3] http://mbed.org/users/alexan_e/notebook/pinout-of-the-mbed-lpc11u24/
- [4] <http://mbed.org/users/fraserphillips/notebook/mbed-gpio-pin-table/>
- [5] <http://mbed.org/handbook/mbed-Microcontrollers>
- [6] <http://mbed.org/handbook/mbed-NXP-LPC11U24>
- [7] <http://mbed.org/handbook/mbed-NXP-LPC1768>,
- [8] <http://mbed.org/blog/entry/mbed-Windows-7-and-USB-cables/>
- [5] <http://mbed.org/handbook/USBSerial>
- [9] <https://mbed.org/handbook/SerialPC>
- [10] <https://mbed.org/handbook/Windows-serial-configuration>
- [11] <https://mbed.org/handbook/Terminals>

Scopriamo una scheda di interfaccia e prototipazione che permette di utilizzare con mbed gli shield pensati per Arduino. 2^ puntata.



Conoscere e usare

mbed

dell'ing. Vincenzo Mendola

Come abbiamo già spiegato nella prima puntata di questo corso, l'mbed ha un package DIP di 40 pin, con passo 0.1", che misura soltanto 54x26mm, quindi non molto più grande di un Arduino Nano. Per poter utilizzare in maniera proficua tutte le caratteristiche del nostro sistema di prototipazione, è conveniente disporre di un circuito stampato cui poter connettere in maniera stabile e affidabile il nostro mbed e che sia in grado inoltre di fornirci

un supporto meccanico adeguato. Il codice per sviluppare applicazioni con l'mbed è diverso da quello utilizzato da Arduino, tuttavia nella prima puntata abbiamo mostrato che è possibile eseguire in maniera equivalente le stesse funzioni di gestione dei GPIO, ADC, ecc. Allora, perché non realizzare un unico PCB che, oltre a fornire un utile supporto alle connessioni del nostro mbed, ci permetta anche di impiegare gli innumerevoli shield di Arduino? Ci siamo posti questa domanda e in risposta ab-

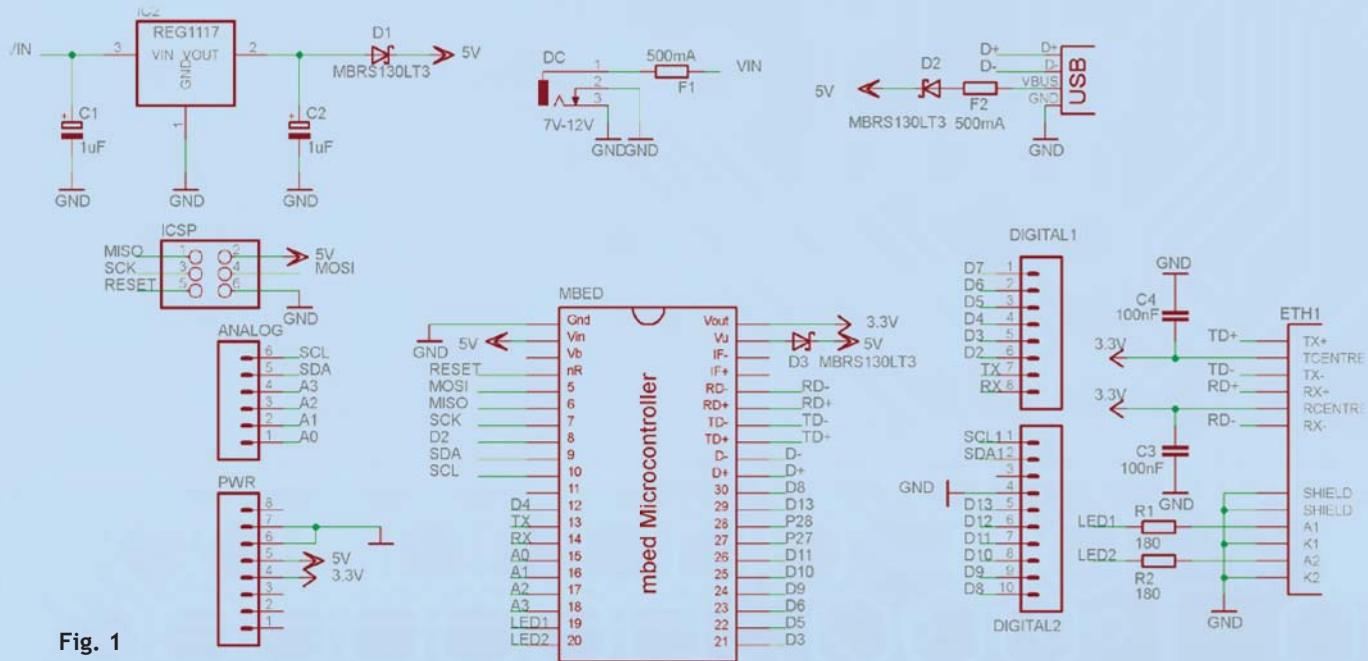


Fig. 1

biamo progettato un circuito stampato di supporto e interconnessione, che soddisfa tutti i requisiti richiesti: compatibilità con gli shield già esistenti e facilità d'accesso alle principali periferiche dell'mbed, tra cui l'ethernet e l'USB. Vediamolo nel dettaglio: in Fig. 1 è riportato lo schema elettrico, mentre in Fig. 2 è visibile una possibile realizzazione della

scheda di questo circuito, che possiamo ritenere un vero e proprio adattatore. Sono presenti gli strip ai quali connettere direttamente l'mbed, gli strip per gli shield di Arduino con il loro caratteristico passo, un connettore USB di tipo B, un connettore RJ45 per interfacciarsi alla rete, i terminali ICSP ed un plug a cui connettere un alimentatore esterno in continua che fornisca una tensione di valore compreso tra 7 e 12 V; quando si alimenta il circuito tramite il plug, si consiglia di non eccedere i 7 V per evitare di dissipare inutilmente potenza nel regolatore lineare a tre terminali a basso dropout (Low Drop Output regulator) REG1117-5.

Questo integrato, prodotto dalla Texas Instruments, è disponibile in due varianti:

- la versione "base", caratterizzata da un drop-out di 1,2 V (per funzionare correttamente fornendo in uscita i 5 V previsti deve avere all'ingresso una tensione di almeno 6,2 V) e da una corrente massima erogabile di 800 mA, che vengono progressivamente ridotti automaticamente superando i 10 V all'ingresso per evitare un eccessivo riscaldamento del chip;
- la versione "advance", indicata con l'aggiunta di una "A" nel codice di identificazione del prodotto (REG1117-5A), che ha un drop-out di 1,3 V alla massima corrente erogabile (pari ad 1A), anche in



Fig. 2

Listato 1

```

#include "mbed.h"
#include "ds1307.h"

Serial pc(USBTX, USBRX);

DS1307 my1307(p9,p10);

int sec = 00;      // Values to set time
int min = 00;
int hours = 12;
int day = 7;
int date = 3;
int month = 3;
int year = 13;

void test_rw(int test)
{
    if (test == 0)
    {pc.printf("%.2D:",hours);
    pc.printf("%.2D:",min);
    pc.printf("%.2D \n\r",sec);
    pc.printf("%.2D/",date);
    pc.printf("%.2D/",month);
    pc.printf("%.2D \n\r",year);
    pc.printf("\n\r");}

    else pc.printf("Last R/W operation failed!\n\r");

    wait(1);
}
int main ()
{
    //test_rw(my1307.settime( sec, min, hours, day, date, month, year)); wait(3); // Required only during
the first RTC connection to set the RTC.

while(1){
test_rw(my1307.gettime( &sec, &min, &hours, &day, &date, &month, &year)); // Read the time and display on
screen
}
}

```

questo caso automaticamente ridotta per tensioni d'ingresso maggiori di 10 V.

In entrambi i casi la massima tensione applicata all'ingresso del regolatore non può eccedere i 15 V, pena il danneggiamento irreversibile del chip. Nel nostro circuito possono essere utilizzate entrambe le versioni, dato che hanno la stessa piedinatura; per quanto riguarda la corrente di esercizio, difficilmente verrà superato il valore di 500 mA, che poi è la corrente standard erogabile da una comune porta USB. Per la stessa ragione abbiamo scelto il più compatto case SOT223, che difficilmente sarà sottoposto a grandi stress termici, meglio tollerati dalla più massiccia versione in contenitore DDPAK.

Come è visibile dallo schema elettrico, sia il plug DC che la USB sono protetti dai cortocircuiti mediante protezioni autoripristinanti ottenute mediante l'uso

di poliswitch da 500 mA, che abbiamo voluto inserire anche se generalmente le porte USB dei PC sono già munite di una loro protezione; in questo modo si può stare tranquilli anche qualora si decida di alimentare il circuito mediante USB non standard o presenti su alimentatori autocostruiti come quelli che presenteremo nei prossimi numeri della rivista. I diodi Schottky D1 e D2 servono per separare le sorgenti di alimentazione dato che, come abbiamo visto, il nostro mbed può essere alimentato sia tramite USB, sia tramite il DC plug.

Sempre sulla linea di alimentazione, possiamo notare D3, il quale, in maniera analoga a quanto appena visto, ha la funzione di garantire che il terminale d'uscita a 5 V del regolatore LDO presente sul PCB dell'mbed possa solo erogare corrente e non diventare un carico. I catodi di questi tre diodi sono fra loro connessi, garantendo

Listato 2

```
#include "mbed.h"

int main() {

    // get the current time from the terminal
    struct tm t;
    printf("Enter current date and time:\n\r");
    printf("YYYY MM DD HH MM SS[enter]\n\r");
    scanf("%d %d %d %d %d %d", &t.tm_year, &t.tm_mon, &t.tm_mday, &t.tm_hour, &t.tm_min, &t.tm_sec);

    // adjust for tm structure required values
    t.tm_year = t.tm_year - 1900;
    t.tm_mon = t.tm_mon - 1;

    // set the time
    set_time(mktime(&t));

    // display the time
    while(1) {
        time_t seconds = time(NULL);
        printf("%s \r", ctime(&seconds));
        wait(1);
    }
}
```

do in questo modo una corretta tensione di alimentazione al pin Vin del mbed. Alla luce di quanto appena visto, al fine di ridurre le cadute di tensione ai loro capi, i tre diodi devono essere necessariamente di tipo Schottky, come ad esempio l'PME-G2005AEA della NXP da noi scelto, che

garantisce una piccola caduta di tensione (400mV@IF=500mA); C1 e C2 sono i due condensatori di filtro al tantalio da 10 µF, della stessa tipologia consigliata dal datasheet; R1 e R2 (da 180 ohm) sono le due resistenze di limitazione della corrente che scorre nei LED inseriti all'interno

Listato 3

```
#include "mbed.h"
#include "EthernetInterface.h"
#include "NTPClient.h"

EthernetInterface eth;
NTPClient ntp;

int main()
{
    eth.init(); //Use DHCP

    eth.connect();

    printf("Trying to update time...\r\n");
    if (ntp.setTime("ntp1.inrim.it") == 0) // INRIM italian NTP server
    {
        printf("Set time successfully\r\n");
        time_t ctTime;
        ctTime = time(NULL);
        printf("Time is set to (UTC): %s\r\n", ctime(&ctTime));
    }
    else
    {
        printf("Error\r\n");
    }

    eth.disconnect();

    while(1) {
    }
}
```

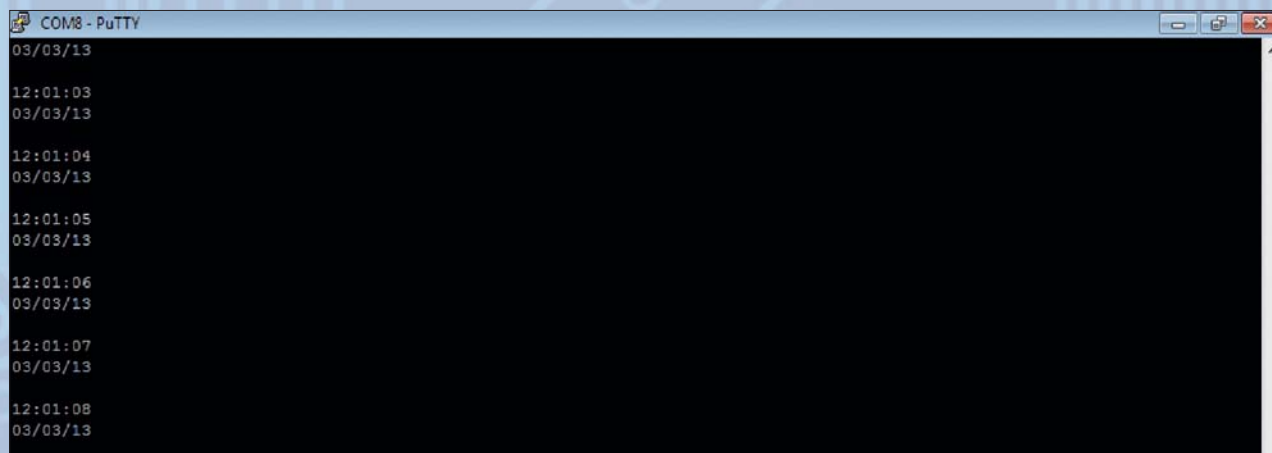


Fig. 3

del connettore RJ45 dotato di filtri magnetici (si può acquistare da Futura Elettronica, www.futurashop.it, con il codice 7300-RJ45EM) che possono essere utilizzati per segnalare la connessione dell'mbed ad una rete ethernet e per evidenziare lo scambio di dati su di essa. C3 e C4 servono per disaccoppiare la tensione di 3,3 V come riportato nella pagina del cookbook dedicata (punto [1] dei Riferimenti).

Il pcb è stato disegnato per ottenere la massima compatibilità con gli shield di Arduino; la **Tabella 1** illustra la corrispondenza tra i pin dell'mbed con quelli di Arduino, un'informazione necessaria, durante la scrittura del codice, per garantire una perfetta equivalenza delle funzionalità con gli shield già esistenti.

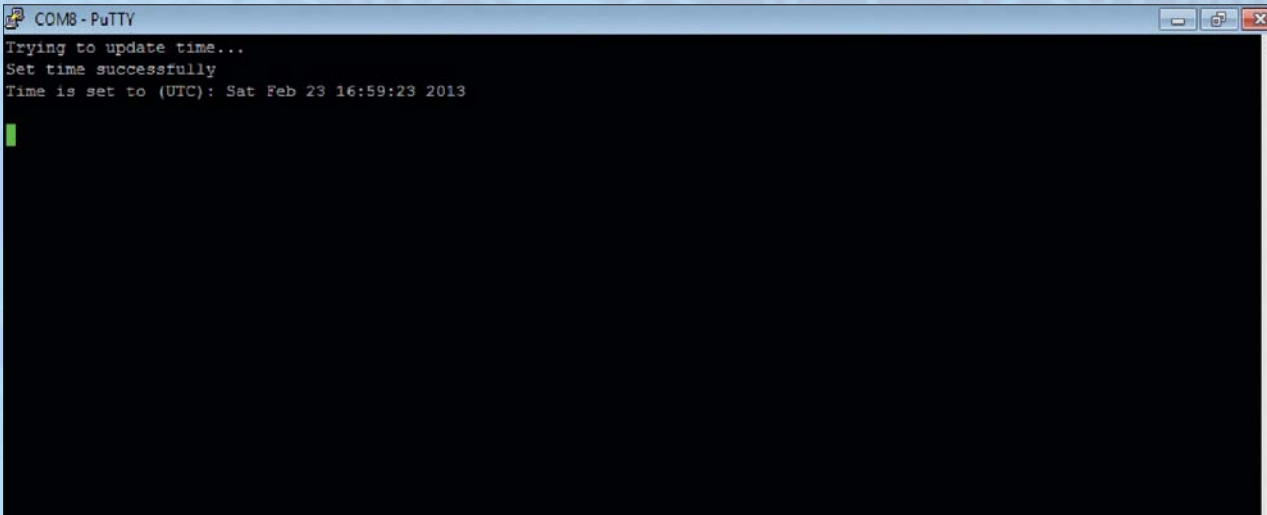
Come si può notare, sono disponibili tutti i GPIO dell'Arduino Uno (D2-D13), i sei PWM (D3, D5, D6, D9, D10, D11), la seriale (D0, D1), la I2C (A4-A5), le connessioni all'ICSP e 4 ADC (A0-A3), tutti disponibili nella stessa posizione in cui lo sono sull'Arduino. Per l'mbed LCP1768 è anche disponibile il DAC, che risulta connesso al pin 18 dell'mbed e all'A3 del nostro PCB. Ora che abbiamo a disposizione questo comodo circuito, che ci permette di utilizzare con facilità gli shield di Arduino, possiamo ad esempio collegare all'mbed l'RTC realizzato per Arduino e presentato nel Fascicolo n° 163 (febbraio 2012) della nostra rivista, con l'unica accortezza di ponticellare con dei fili le due connessioni a SDA e SCL (replicate nelle ultime revisioni di Arduino UNO alle estremità del connettore dei GPIO, oltre il D13) con i

pin analogici A4 (SDA) e A5 (SCL) della Arduino stessa.

Ora, prima di procedere è importante fare una precisazione: gli mbed sono 5V-tolerant (cioè accettano livelli TTL, pur

ARDUINO	MBED	FUNCTION
D0	p14	
D1	p13	
D2	p8	
D3	p21	PWM
D4	p12	
D5	p22	PWM
D6	p23	PWM
D7	p27	
D8	p30	
D9	p24	PWM
D10	p25	PWM
D11	p26	PWM
D12	p28	
D13	p29	
A0	p15	
A1	p16	
A2	p17	
A3	p18	DAC
A4	P9	SDA
A5	P10	SCL
ICSP	P5	MOSI
ICSP	P6	MISO
ICSP	P7	SCK

Tabella 1



```

COM8-PuTTY
Trying to update time...
Set time successfully
Time is set to (UTC): Sat Feb 23 16:59:23 2013

```

Fig. 4

prediligendo quelli 0/3,3V) ma, per essere certi di evitare danni irreparabili al chip, è sempre consigliabile utilizzare un traslatore di livello tipo quello presentato per Raspberry Pi nel fascicolo di febbraio di quest'anno, oppure lavorare direttamente con logica a 3,3 V, ovviamente anche per il Bus I²C.

Lo shield RTC di Arduino lavora con un I²C a livello TTL (0/5V) per cui è necessaria molta attenzione qualora lo si voglia connettere direttamente all'mbed.

Sappiamo che l'LCP1768, a differenza della versione LPC11U24, incorpora già un modulo RTC, ma comunque ad entrambi è possibile connettere un DS1307 della Maxim, servendosi di un'apposita libreria (ad esempio quella disponibile all'indirizzo http://mbed.org/users/harrypowers/code/DS1307/docs/c3e4da8feb10/ds1307_8h_source.html) e del semplice codice riportato nel **Listato 1**.

Nella **Fig. 3** è visibile la schermata di terminale della seriale, contenente i dati in uscita dall'RTC. Notate che è possibile utilizzare anche l'RTC a bordo dell'LPC1768 e del nuovo modello FRDM-KL25Z, servendosi del codice (punto [2] dei Riferimenti) contenuto nel **Listato 2**.

Come si vede dalla lettura delle poche istruzioni necessarie, l'RTC on-board di mbed viene impostato direttamente inserendo la data e l'ora tramite seriale, sulla quale viene costantemente aggiornata e visualizzata ogni secondo.

Il nostro PCB di adattamento dispo-

ne anche di un connettore ethernet che consente di connettere l'LPC1768 alla rete LAN (Local Area Network) e WAN (Wide Area Network); perciò vi proponiamo un semplice esempio di utilizzo della libreria ETHERNET che permette di determinare l'ora esatta tramite un server NTP (Network Time Protocol) qualora abbiate connesso il vostro mbed alla rete INTERNET e vogliate sfruttare l'altissima precisione fornita dagli orologi atomici (il codice è illustrato nel **Listato 3**). Per i nostri test ci siamo serviti del primo dei due server NTP primari installati nel Laboratorio di Tempo e Frequenza Campione dell'INRIM, accessibili liberamente. Ulteriori informazioni sono disponibili alla pagina http://www.inrim.it/ntp/index_i.shtml. La libreria ed il relativo codice sono disponibili nella pagina http://mbed.org/users/donatien/code/NTPClient_HelloWorld/.

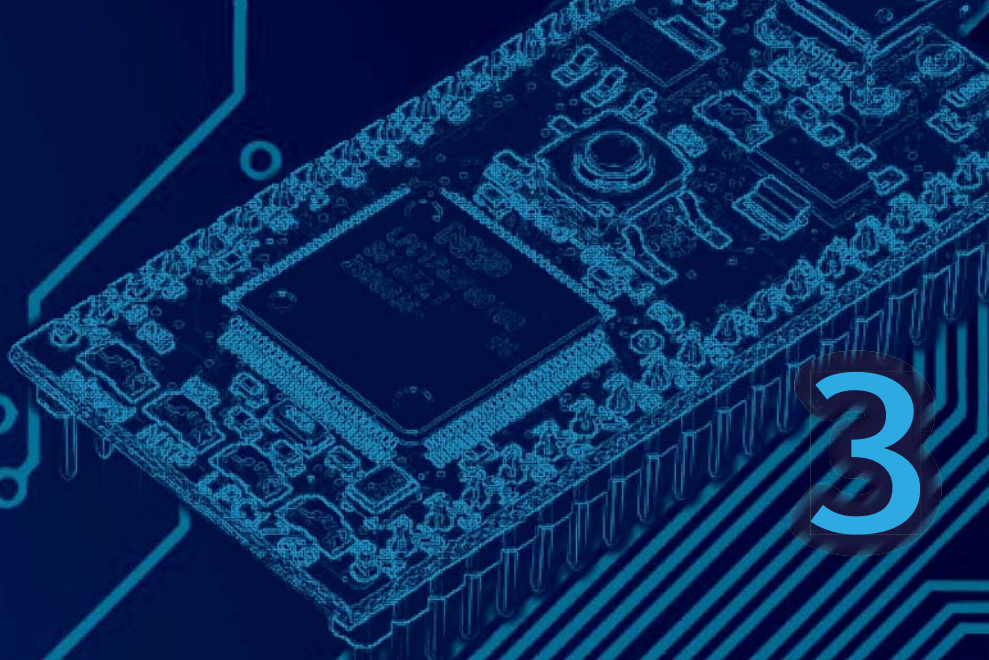
Nella **Fig. 4** si vede il risultato della sincronizzazione con il server NTP.

Bene, con questo abbiamo concluso anche questa puntata; nella prossima mostreremo ulteriori esempi di utilizzo del nostro circuito di adattamento e approfondiremo la conoscenza delle schede di prototipazione per mbed. ■

Riferimenti

- [1] <http://mbed.org/cookbook/Ethernet-RJ45>
- [2] <http://mbed.org/blog/entry/103/>

Utilizziamo la nostra piattaforma di prototipazione per pilotare un display LCD mediante uno shield Arduino in modo da visualizzare scritte a piacere e l'ora dell'RTC, ma anche per generare forme d'onda usando il DAC del LPC176. Terza puntata.



Conoscere e usare

mbed

dell'ing. Vincenzo Mendola

Nelle puntate precedenti abbiamo introdotto le schede di prototipazione mbed, mostrandovi le nozioni base per il loro utilizzo per aiutarvi a prendere confidenza con il relativo hardware; abbiamo inoltre introdotto un comodo circuito adattatore che ci permette di utilizzare con facilità gli shield di Arduino sulla piattaforma mbed a 32 bit. Con questo adattatore abbiamo iniziato a realizzare alcune applicazioni, descrivendo di volta in volta il relativo codice. In questa puntata prosegua-

mo la descrizione delle applicazioni e lo facciamo proponendo la gestione di vari display standard (a controllo parallelo) a cristalli liquidi, ma anche la generazione di segnali di forma d'onda più comune. Partiamo con il primo esempio applicativo, utilizzando l'LCD shield per Arduino, presentato sul numero 174 di marzo 2013 di Elettronica In (e commercializzato da Futura Elettronica con la sigla 7100-FT1030K) per mostrare qualche esempio di codice che permette ad mbed di interfacciarsi ad un comune LCD. Per prima cosa provvediamo alla

Listato 1

```
#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p30, p24, p12, p22, p23, p27,
TextLCD::LCD16x2);

int main()
{
    lcd.locate(0,0);
    lcd.printf("MBED SHIELD");
    lcd.locate(0,1);
    lcd.printf("ELETTRONICA IN");
}
```

visualizzazione delle scritte "MBED SHIELD" e "ELETTRONICA IN" (**Listato 1**).

Com'è facile intuire, la libreria *TextLCD* serve per rappresentare caratteri alfanumerici su LCD basati sul diffusissimo driver HD44780 o controller compatibili. Fra parentesi sono riportati, nell'ordine, i pin dell'mbed connessi ai terminali "RS", "E", "D4-D7" del modulo LCD, mentre "TextLCD::LCD16x2" serve per definire la tipologia del modulo connesso. Attualmente sono supportati i display LCD, elencati nella **Tabella 1**, ognuno con la relativa istruzione da includere nel codice. Tornando al **Listato 1**, l'istruzione "lcd.locate(0,1);" analogamente a quanto avviene per Arduino, indica la colonna e la riga in cui visualizzare il testo tra le virgolette del comando "lcd.printf("");".

In maniera analoga al codice di Arduino, la riga #include "TextLCD.h" richiama la libreria per la scrittura di caratteri sugli LCD, mentre la riga "TextLCD lcd(p27, p23, p22, p12, p21, p8, TextLCD::LCD16x2);" serve per definire i collegamenti tra l'mbed (connesso in modalità a 4 BIT) e il display, oltre che la tipologia (due righe e 16 caratteri per riga); il comando "lcd.printf("ELETTRONICA IN\n");" serve per scrivere il testo desiderato. Come per tutte le funzioni e le librerie di mbed, è possibile approfondire l'argomento mediante il cookbook (<https://mbed.org/cookbook/Homepage>) e l'handbook (<https://mbed.org/handbook/Homepage>).

Listato 2

```
#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p27, p23, p22, p12, p21, p8, TextLCD::LCD16x2); //meteo shield

int main() {
    lcd.printf("ELETTRONICA IN\n");
}
```

Tabella 1

Tipo di display	Istruzione
8x1 LCD panel	TextLCD::LCD8x1
8x2 LCD panel	TextLCD::LCD8x2
16x1 LCD panel	TextLCD::LCD16x1
16x4 LCD panel	TextLCD::LCD16x4
24x2 LCD panel	TextLCD::LCD24x2
24x4 LCD panel	TextLCD::LCD24x4
40x2 LCD panel	TextLCD::LCD40x2
40x4 LCD panel	TextLCD::LCD40x4

ge). Qualora si volesse realizzare un proprio LCD shield, basta semplicemente modificare i parametri riportati fra le parentesi del comando TextLCD in modo da rispettare le connessioni utilizzate nel proprio PCB. Per esempio possiamo servirci del meteo shield (fascicolo n° 171 di Elettronica In) ed utilizzarlo per scrivere del testo a piacimento (**Fig. 1**). Il relativo codice è quello nel **Listato 2**.

Passiamo ora ad un esempio un po' più complesso: proviamo a visualizzare l'ora e la data del nostro RTC shield su un comune display LCD 16x2; il codice è quello mostrato nel **Listato 3**.

Ricordiamo che l'RTC shield e l'RTC presente sul meteo shield, utilizzano entrambi il DS1307, che ha una logica operante a 5V, mentre gli mbed lavorano a 3,3V; anche se sono "5V tolerant" (possono accettare al loro ingresso tensioni di 5V e leggerle correttamente come livello alto senza danneggiarsi) si



Fig. 1

Listato 3

```

#include "mbed.h"
#include "TextLCD.h"
#include "ds1307.h"

TextLCD lcd(p27, p23, p22, p12, p21, p8, TextLCD::LCD16x2); //meteo shield
Serial pc(USBTX, USBRX);
DS1307 my1307(p9,p10);

int sec = 0;
int min = 55;
int hours = 20;
int day = 7;
int date = 3;
int month = 3;
int year = 13;

void test_rw(int test)
{
    if (test == 0) pc.printf("Last R/W operation passed!\n\r");
    else pc.printf("Last R/W operation failed!\n\r");
}

int main()
{test_rw(my1307.settime( sec, min, hours, day, date, month, year));// to set RTC
while(1){ test_rw(my1307.gettime( &sec, &min, &hours, &day, &date, &month, &year));
    lcd.locate(0,0);
    lcd.printf("% .2D",hours);
    lcd.printf(":% .2D",min);
    lcd.printf(":% .2D",sec);

    lcd.locate(0,1);
    switch(day){
    case 1:
        lcd.printf("LUN");
        break;
    case 2:
        lcd.printf("MAR");
        break;
    case 3:
        lcd.printf("MER");
        break;
    case 4:
        lcd.printf("GIO");
        break;
    case 5:
        lcd.printf("VEN");
        break;
    case 6:
        lcd.printf("SAB");
        break;
    case 7:
        lcd.printf("DOM");
        break;
    }

    lcd.printf(" % .2D",date);
    lcd.printf("/% .2D",month);
    lcd.printf("/% .2D",year);
}
}

```

consiglia di utilizzare un traslatore di livello (5V/3,3V) bidirezionale, ossia un circuito in grado di trasformare in 0/3,3V i livelli TTL in ingresso e di elevare a 5 volt i livelli logici alti in uscita dai pin dell'mbed. Il circuito può essere qualcosa di analogo allo shield suggerito nel fascicolo n° 173 della rivista, per adattare RaspberryPi agli shield per Arduino. La riga "switch-case" del Listato 3 può anche essere sostituita dalla seguente porzione di

codice (il risultato resta invariato):

```

if (day == 1)
    lcd.printf("LUN");
if (day == 2)
    lcd.printf("MAR");
if (day == 3)
    lcd.printf("MER");
if (day == 4)
    lcd.printf("GIO");
if (day == 5)
    lcd.printf("VEN");
if (day == 6)
    lcd.printf("SAB");

```

Listato 4

```

#include "mbed.h"
#include "TextLCD.h"
#include "ds1307.h" //http://mbed.org/users/harrypowers/code/DS1307/docs/c3e4da8feb10/ds1307_8h_source.html

Serial pc(USBTX, USBRX);
TextLCD lcd(p27, p23, p22, p12, p21, p8, TextLCD::LCD16x2);

DS1307 my1307(p9,p10);

int sec = 0;
int min = 00;
int hours = 16;
int day = 7;
int date = 23;
int month = 2;
int year = 13;

void test_rw(int test)
{
    if (test == 0) pc.printf("Last R/W operation passed!\n\r");
    else pc.printf("Last R/W operation failed!\n\r");
}

float VADC= 3.31;
float Vcc=4.21;

int DPR = 0;
int RHCORR = 0;
int PCORR = 0;

float STAMPA_T = 0;
float STAMPA_U = 0;
float STAMPA_P = 0 ;

AnalogIn val(p17);
AnalogIn UM(p16);
AnalogIn Pascal(p15);

float temp()
{
    float T=0.0;
    float nread = 100.0;          // NUMBER OF READINGS
    float somma = 0.0;
    for (int i=0; i<nread; i++) {
        float TADC=val.read();
        T = (((VADC*TADC)-0.5)* 100); //TEMPERATURE
        somma += T;
    }
    wait_ms(100);
    return (somma/nread);
}

float readUMID()
{
    float RHout=0.0;
    float nread = 100.0;          // NUMBER OF READINGS
    float somma = 0.0;
    for (int i=0; i<nread; i++) {
        float UMADC = UM.read();
        RHout=(((UMADC)-0.1515)/0.00636)+RHCORR; //HUMIDITY
        somma += RHout;
    }
    wait_ms(100);
    return (somma / nread);
}

float pressure()
{
    float P=0.0;
    float nread = 100.0;          // NUMBER OF READINGS
    float somma = 0.0;
    for (int i=0; i<nread; i++) {
        float PascalADC= Pascal.read();
        P=(((PascalADC*VADC)/Vcc+0.095)/0.009)*10+DPR+PCORR; //PRESSURE TRANSFERT FUNCTION
        somma += P;
    }
}

```

```

    wait_ms(100);
    return ( somma / nread );
}

int main()
{
//test_rw(my1307.settime( sec, min, hours, day, date, month, year)); //REGOLA ORARIO E DATA
while (true) {

    STAMPA_T= (temp());
    STAMPA_U= (readUMID());
    STAMPA_P = (pressure());
    lcd.locate(0, 0);
    lcd.printf("%2.1f",STAMPA_T); //SHOW ONLY THE FIRST DECIMAL
    wait_ms(200);
    lcd.locate(4,0);
    lcd.printf("C");
    lcd.locate(6, 0);
    lcd.printf("%2.1f",STAMPA_U); //SHOW ONLY THE FIRST DECIMAL
    lcd.locate(10,0);
    lcd.putc(37);
    wait_ms(200);

    lcd.locate(12, 0);
    lcd.printf("%4.0f",STAMPA_P); //SHOW ONLY THE INTEGER PART
    wait_ms(200);

    test_rw(my1307.gettime( &sec, &min, &hours, &day, &date, &month, &year));
    lcd.locate(0,1); // Print and refresh data on line 2 of the LCD display
    lcd.printf("%2.2D",hours);
    lcd.printf("%2.2D",min);
    //lcd.printf("%2.2D",sec);
    //lcd.printf("%1D",DAY);

    if (day == 1)
        lcd.printf(" D");
    if (day == 2)
        lcd.printf(" L");
    if (day == 3)
        lcd.printf(" Ma");
    if (day == 4)
        lcd.printf(" Me");
    if (day == 5)
        lcd.printf(" G");
    if (day == 6)
        lcd.printf(" V");
    if (day == 7)
        lcd.printf(" S");

    lcd.printf(" %2.2D",date);
    lcd.printf("/%2.2D",month);
    lcd.printf("/%2.2D",year);

}
}

```

```

if (day == 7)
    lcd.printf("DOM");

```

A questo punto vediamo come modificare il codice scritto per Arduino allo scopo di utilizzare il meteo shield con l'mbed. Per prima cosa ricordiamo che sul PCB dello shield sono presenti due ponticelli che permettono di selezionare la corretta tensione di alimentazione del sensore di pressione, previsti a suo tempo per facilitare l'uso del circuito anche con i sensori a 3,3V (ad esempio il MP3H6115A), tensione usata dai microcontrollori di mbed e da Arduino Due. Prima di mostrare una possibile versione del codice per mbed, ricordiamo che il modello LCP1768 incorpora già un RTC, mentre la versione LPC11U24 no, per cui abbiamo inserito anche la libreria per il DS1307, scaricabile all'indirizzo web <http://mbed>.

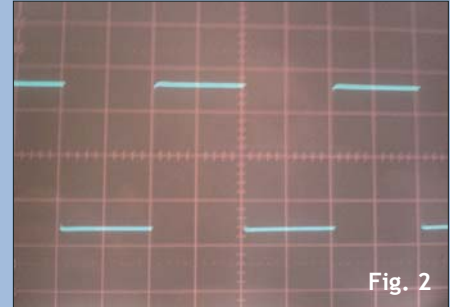


Fig. 2

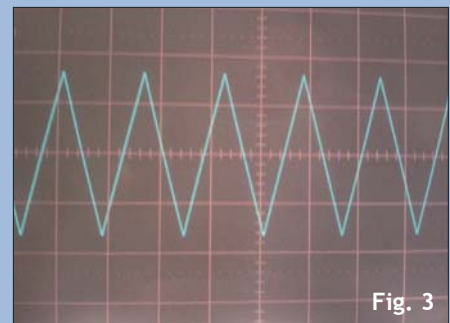


Fig. 3

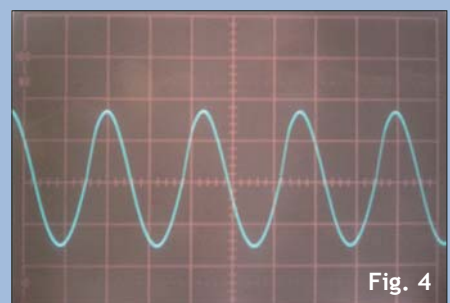


Fig. 4

Listato 5

```
#include "mbed.h"

PwmOut led(LED1);

int main() {
    while(1) {
        for(float p = 0.0; p < 1.0; p += 0.1) {
            led = p;
            wait(0.1);
        }
    }
}
```

Listato 6

```
#include "mbed.h"

PwmOut leds[] = {LED1, LED2, LED3, LED4};
int i;

float p=0.0;

int main()
{
    while(1) {

        for (i=0; i<4; i++) {
            for(p = 0.0; p <= 1.0; p += 0.01) {
                leds[i] = p;
                wait(0.01);
            }
            for (i = 3; i>=0; i--) {
                for(p = 1.0; p >= 0.0; p -= 0.01) {
                    leds[i] = p;
                    wait(0.01);}
            }

            wait(0.1);
        }
    }
}
```

Listato 7

```
#include "mbed.h"

AnalogOut DA(p18);

int main()
{
    while(1) {
        DA=1; //ONDA QUADRA
        wait_us(100);
        DA=0;
        wait_us(100);
    }
}
```

org/users/harrypowers/code/DS1307/docs/c3e4da-8feb10/ds1307_8h_source.html. Si noti che tale libreria può comunque anche essere utilizzata con l'mbed basato sul processore Cortex M3. Il codice mbed è quello contenuto nel **Listato 4**. Un'altra funzionalità molto utile nei sistemi basati su microcontrollore è la creazione di segnali PWM, solitamente ottenuta da appositi moduli PWM integrati nei micro stessi. Nel

Listato 8

```
#include "mbed.h"

float i;

AnalogOut DA(p18);

int main() {
    for (i=0; i<1;i+=0.01) //ONDA TRIANGOLARE
        {DA=i;
        wait_us(1);}

    for (i=1; i>0; i=i-0.01)
        {DA=i;
        wait_us(1);}

}
}
```

Listato 9

```
#include "mbed.h"

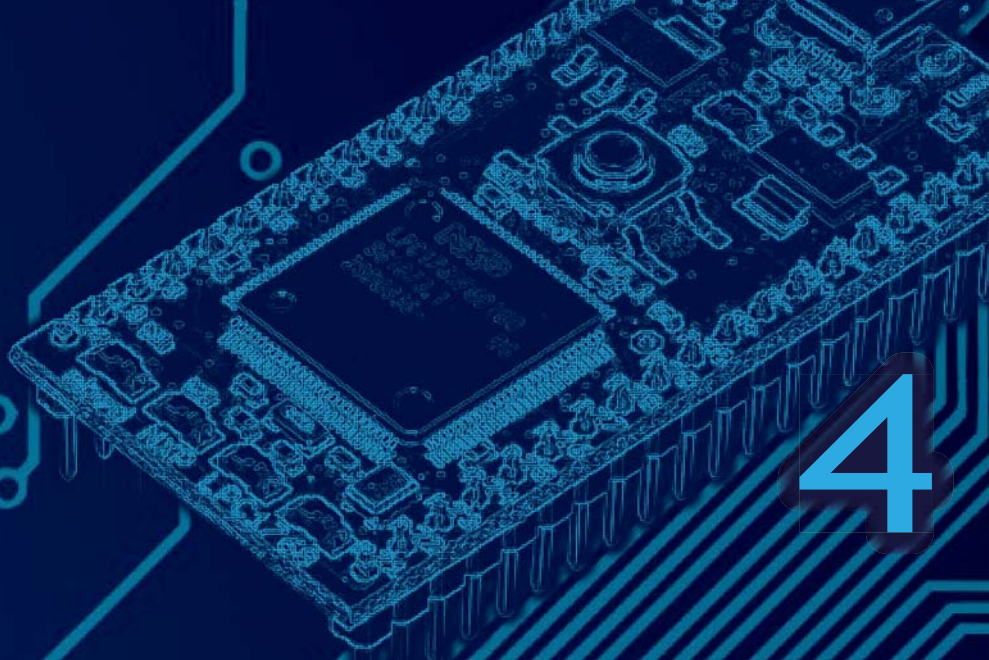
float i;

AnalogOut DA(p18);

int main() {
    for (float i=0; i<360; i++) { //ONDA SINUSOIDALE
        DA = sin(i/180*3.14)*0.5+0.5;
        wait_us(1);
    }
}
```

caso della nostra piattaforma mbed, la generazione del PWM è implementata attraverso la funzione PwmOut (<https://mbed.org/handbook/PwmOut>). Nel **Listato 5** e nel **Listato 6** trovate due semplici applicazioni pratiche che sfruttano i quattro LED a bordo dell'mbed LPC1768. Vediamo ora come generare delle semplici forme d'onda utilizzando il DAC (Digital to Analog converter) presente nell'mbed LPC1768. Cominciamo con la generazione di un'onda rettangolare come quella visibile in **Fig. 2**, la quale può essere generata senza alcuna difficoltà con il codice che trovate nel **Listato 7**. Con le piattaforme mbed non ci si limita alla generazione della classica onda quadra, perché è anche possibile generare forme d'onda più complesse come la triangolare (**Fig. 3**) e la sinusoidale (**Fig. 4**), ottenibili rispettivamente con i codici che trovate nei **Listati 8** e **9**. Come sempre, consigliamo di leggere attentamente la documentazione ufficiale per ulteriori dettagli e per imparare a sfruttare tutte le caratteristiche delle funzioni e delle librerie disponibili, che qui per ragioni di spazio, non abbiamo potuto descrivere. Bene, con questo abbiamo concluso. Riprenderemo il discorso nel prossimo fascicolo. ■

Realizziamo la nostra prima applicazione con una board mbed ready: la KL25Z prodotta da Freescale. Quarta ed ultima puntata.



Conoscere e usare

mbed

dell'ing. Vincenzo Mendola

Nell'ultimo anno abbiamo assistito alla nascita di tantissime nuove board embedded destinate al mondo hobbystico o comunque al mercato di fascia medio-bassa: sulla scia di Arduino e di Raspberry Pi hanno fatto la loro comparsa PcDuino, BeagleBoard, Cubieboard, Odroid, UDOO e tante altre ancora. Schede via via sempre più potenti, addirittura multiprocessore, ma che rimangono in quella fascia di prezzo (massimo 50-100 Euro) alla portata un po' di tutti, studenti, hobbysti e semplici appassionati. Un fiorire di nuovi prodotti che ha visto una convergenza tra il mondo Arduino e il sistema operati-

vo GNU/Linux all'insegna dell'open hardware e dell'open software. La maggior parte delle board presenta infatti la facilità della gestione dei GPIO e delle periferiche tipica di Arduino, mentre la connessione alla rete e quindi ad Internet e la gestione complessiva della board, è garantita da un sistema operativo basato su GNU/Linux. Questa è anche la logica progettuale delle nuove board immesse sul mercato o semplicemente annunciate dal team di Arduino. Parliamo, in primo luogo, di Arduino Yún, la prima scheda di casa Arduino che fa ricorso ad un sistema operativo GNU/Linux denominato Linino, ma anche di Arduino TRE che dovrebbe vedere la luce nella primavera dell'an-

Listato 1

```
#include "mbed.h"

int main() {

    DigitalOut ledG(LED_GREEN);

    while (true) {

        ledG = 1; // off
        wait(0.5);
        ledG = 0; // on
        wait(0.5);

    }

}
```

no prossimo e che utilizzerà un processore Sitara di Texas Instruments ed un sistema operativo GNU/Linux.

Leggermente diversa è invece l'impostazione della board Intel Galileo che utilizza un processore x86 e nella quale lo sketch di Arduino "gira" all'interno del sistema GNU/Linux come un processo utente.

Indubbiamente la comparsa e la disponibilità di una scheda a bassissimo costo basata su GNU/Linux come Raspberry Pi (ad oggi sono state vendute oltre due milioni di board) ha fatto capire che, il vantaggio di disporre on board di un sistema operativo flessibile (e gratuito!) come Linux sono di gran lunga superiori agli svantaggi che comporta una programmazione un po' più complessa. Tanto fervore con tante nuove schede ha anche avuto il benefico effetto di accentuare la concorrenza tra i produttori di semiconduttori, che si sono prodigati nell'immettere sul mercato prodotti sempre più performanti e sempre più economici. Tutto ciò si è tradotto in una riduzione dei prezzi delle board che ha generato una spirale positiva che ha fatto sì che anche le tecnologie più avanzate potessero risultare disponibili a prezzi incredibilmente bassi.

LA SCHEDA MBED READY KL25Z

Da questa corsa contro il tempo non resta fuori nemmeno il team mbed, che negli ultimi mesi ha introdotto, dopo anni di apparente staticità, numerose innovazioni, passando dalla riduzione generalizzata dei costi dell'hardware, all'aggiunta di nuovi modelli, non direttamente progettati dal team ufficiale, ma compatibili. Dunque, nell'universo mbed, prima conosciuto per i suoi modelli "storici" che ne hanno decretato il successo

Fig. 1



(parliamo della LPC1768 e della LPC1114, di cui al punto [1] dei riferimenti).

Tra le nuove schede della community mbed, quella che ha attirato subito la nostra attenzione è stata la KL25Z sviluppata da Freescale (Fig. 1): venduta al pubblico ad un prezzo veramente concorrenziale (si parla di circa 12 euro) per un prodotto di questo tipo e

Listato 2

```
#include "mbed.h"

int main() {

    DigitalOut ledG(LED_GREEN);
    DigitalOut ledR(LED_RED);
    DigitalOut ledB(LED_BLUE);

    //ALL COLORS OFF
    ledR=1;
    ledG=1;
    ledB=1;

    while (true) {

        //GREEN

        ledG = 0; // on
        wait(0.25);

        ledG = 1; // off
        wait(0.25);

        //RED

        ledR = 0; // on
        wait(0.25);

        ledR = 1; // off
        wait(0.25);

        //BLUE

        ledB = 0; // on
        wait(0.25);

        ledB = 1; // off
        wait(0.25);

    }

}
```

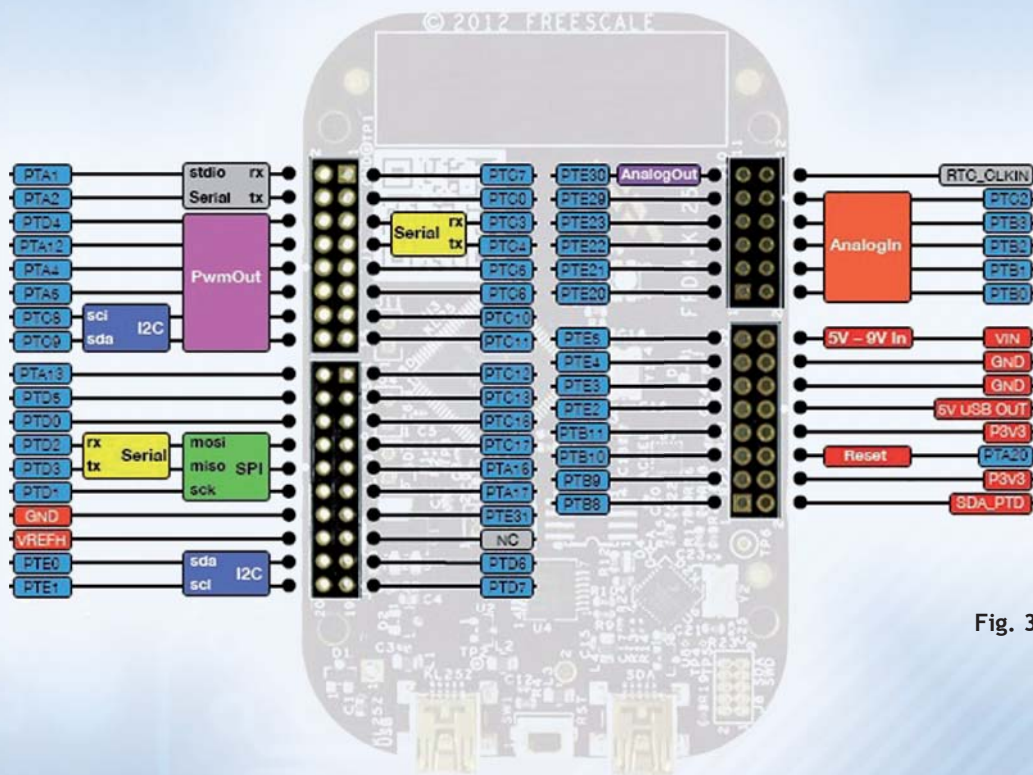


Fig. 3

con queste caratteristiche, la KL25Z è sostanzialmente una board “mbed ready” (così definita per via della possibilità di utilizzare le librerie sviluppate per le board mbed tramite un semplice upload attraverso la presa USB, come dettagliato sul sito ufficiale) dalle interessantissime e peculiari caratteristiche, le quali la differenziano da quelle delle due versioni presentate nelle precedenti puntate. La scheda di Freescale è facilmente upgradabile ad mbed, semplicemente aggiornandone il firmware, come dettagliato nella pagina descrittiva della board di cui al punto [2] dei Riferimenti. Una volta fatto l’aggiornamento, il sistema ci confermerà che la procedura è andata correttamente a termine (con un messaggio come quello riportato nella Fig. 2). La prima cosa che salta subito all’occhio è il form factor di questa nuova board, che è compatibile con quello delle board Arduino; inoltre la KL25Z di Freescale dispone di pad forati sui lati come Arduino, quindi in pratica, basta aggiungere gli header e si possono montare su di esso gli stessi shield facendo

però attenzione al fatto che gli mbed lavorano con logica a 3,3V. Questa caratteristica costituisce un vantaggio importante, in quanto non si può negare che la vastità di shield ed applicazioni disponibili per Arduino è molto allettante e che uno degli elementi che uno sperimentatore spesso valuta prima dell’acquisto di una scheda

Device 'kl25z' connected and added to your account.

Fig. 2

Listato 3

```
#include "mbed.h"

BusOut RGB(PTB18, PTB19, PTD1);

int main() {
    //ALL COLORS OFF

    RGB=0x07;

    while (true) {

        RGB=0x06;
        wait(0.25);
        RGB=0x05;
        wait(0.25);
        RGB=0x03;
        wait(0.25);
    }
}
```


Listato 4

```
#include "mbed.h"

DigitalOut ledR(LED_RED);
Serial pc(USBTX,USBRX);

int main() {
    int i=1;
    pc.printf("Hello World!\r\n");

    while (true) {
        wait(1);
        pc.printf("%d\r\n",i);
        i++;
        ledR = !ledR;
    }
}
```

Listato 5

```
#include "mbed.h"
#include "MMA8451Q.h"

#define MMA8451_I2C_ADDRESS (0x1d<<1)

int main(void) {
    MMA8451Q acc(PTE25, PTE24, MMA8451_I2C_
ADDRESS);
    PwmOut rled(LED_RED);
    PwmOut gled(LED_GREEN);
    PwmOut bled(LED_BLUE);

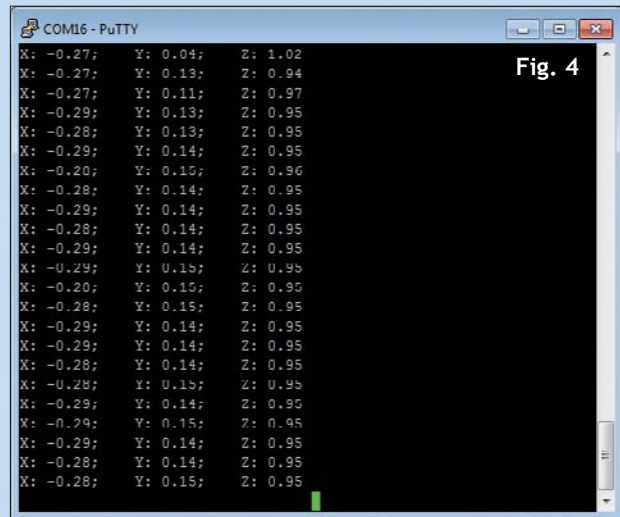
    while (true) {
        rled = 1.0 - abs(acc.getAccX());
        gled = 1.0 - abs(acc.getAccY());
        bled = 1.0 - abs(acc.getAccZ());

        printf("X: %1.2f;   ", acc.getAccX());
        printf("Y: %1.2f;   ", acc.getAccY());
        printf("Z: %1.2f \n\r", acc.getAccZ());
        wait(0.1);
    }
}
```

di prototipazione è la possibilità di utilizzare questo "bagaglio". Ecco perché la compatibilità con il mondo Arduino è attualmente (a ragione o meno) un fattore tenuto molto in considerazione da chi realizza nuove schede di sviluppo e prototipazione.

Vediamo dunque le caratteristiche della scheda Freescale (reperibili sul web alla pagina indicata al punto [3] dei Riferimenti): il cuore della KL25Z è un processore ARM® Cortex™-M0+ Core operante con un clock di 48MHz, con 16 kB RAM e 128 kB di Flash; oltre alle periferiche standard per questa tipologia di tool (2 bus SPI, 2 I²C-Bus, 3 UART, 6 moduli PWM, 6 ADC, vari GPIO), saltano all'occhio delle funzionalità avanzate e di sicuro interesse: a bordo si trovano un accelerometro a tre assi (si tratta di un MMA8451Q della Freescale), un LED RGB controllato in PWM e un touch sensor di tipo capacitivo.

Prima di proseguire è d'obbligo un accenno



alle specifiche elettriche: la tensione di alimentazione della board è compresa tra 5 e 9 volt (in continua), con un assorbimento massimo dichiarato di 100 mA; un regolatore sulla scheda riduce la tensione adattandola a quella richiesta dalla logica a 3,3 V, che caratterizza da tempo la maggior parte dei dispositivi simili al Freescale. Ciascun terminale di I/O della board è in grado di erogare 4 mA. In Fig. 3 è visibile il dettaglio del pinout del KL25Z.

Proviamo a prendere confidenza con la KL25Z partendo da alcuni semplici esempi applicativi. Il primo riguarda la gestione del LED RGB disponibile sul PCB, che otteniamo con un semplice esempio che ci permette di far lampeggiare il solo colore verde: lo trovate nel Listato 1. Se guardate lo schema elettrico del KL25Z (punto [4] dei Riferimenti) noterete che il LED RGB sulla scheda è di tipo ad anodo comune, quindi per pilotarlo è necessario fornire un livello basso al singolo LED da accendere; in questo caso è stato scelto il verde (LED_GREEN). Come vedete, il codice è veramente semplice ed auto esplicativo.

Vediamo ora, nel Listato 2, come far lampeggiare tutti e tre i colori che il LED montato sulla scheda mbed può produrre.

L'applicazione proposta non è affatto complessa; è solo un esempio di base.

La stessa cosa si sarebbe potuta ottenere servendosi delle istruzioni contenute nel Listato 3, dedotte osservando in dettaglio lo schema elettrico; guardare lo schema è una cosa che consigliamo sempre di fare, perché vi permette di capire maggiormente le caratteristiche dell'hardware che stiamo utilizzando, permettendoci di sfruttarne al meglio le caratteristiche.

A questo punto del corso riteniamo non sia più necessario spiegare le istruzioni riga per riga,

in quanto dovrete ormai avere il background che vi permette di comprenderle. Ad ogni modo, per chi volesse approfondire, può trovare ulteriori dettagli inerenti a "BusOut" visitando la pagina web <http://mbed.org/handbook/BusOut>.

Bene, detto ciò, alla nostra applicazione andiamo ad aggiungere la comunicazione con la seriale, la quale viene implementata grazie al codice che trovate nel **Listato 4**.

Passiamo quindi ad un esempio che combina l'uso dell'accelerometro con quello del LED RGB: il valore dell'accelerazione sui 3 assi è visibile sia in maniera numerica tramite la seriale del nostro mbed (**Fig. 4**, sia direttamente tramite il colore del LED sulla scheda, che varia in base al valore misurato. Nel **Listato 5** trovate il codice che permette di ottenere quanto appena detto. Il team di mbed ha creato una libreria per utilizzare in maniera immediata lo slide presente sulla board. Vediamo due semplici applicazioni che lo riguardano, il primo dei quali si basa sul codice che trovate nel **Listato 6**. Il secondo esempio si basa invece sul codice descritto nel **Listato 7**.

Nel primo caso, il LED mantiene visibile il livello di luminosità selezionato tramite lo slide, mentre il codice del **Listato 7** lo accende solo quando toccherete lo slide e la luminosità varierà fintanto che farete scorrere il dito. Per finire la lezione di quest'ultima puntata del corso, ecco un esempio prettamente "scenografico" tratto dalla pagina ufficiale del KL25Z (punto [5] dei Riferimenti) nella quale potrete trovare altri esempi applicativi. Il codice corrispondente da caricare in mbed è quello contenuto nel **Listato 8**.

Bene, ora che avete appreso i rudimenti per lavorare in maniera proficua con le schede mbed, potete proseguire da soli nell'uso e arricchire il vostro bagaglio di applicazioni in base alle vostre esigenze, ricordando che potrete sempre

Listato 6

```
#include "mbed.h"
#include "TSSensor.h"

int main(void) {
    PwmOut led(LED_BLUE);
    TSSensor tsi;

    while (true) {
        while(tsi.readPercentage()>0) {
            led = 1.0 - tsi.
readPercentage();
            wait(0.1);}
    }
}
```

Listato 7

```
#include "mbed.h"
#include "TSSensor.h"

int main(void) {
    PwmOut led(LED_RED);
    TSSensor tsi;

    while (true) {

        led = 1.0 - tsi.readPercentage();
        wait(0.1);
    }
}
```

Listato 8

```
#include "mbed.h"

PwmOut r (LED_RED);
PwmOut g (LED_GREEN);
PwmOut b (LED_BLUE);

int main() {
    r.period(0.001);
    g.period(0.001);
    b.period(0.001);

    while (true) {
        for (float i = 0.0; i < 1.0 ; i += 0.001) {
            float p = 3 * i;
            r = 1.0 - ((p < 1.0) ? 1.0 - p : (p > 2.0) ? p - 2.0 : 0.0);
            g = 1.0 - ((p < 1.0) ? p : (p > 2.0) ? 0.0 : 2.0 - p);
            b = 1.0 - ((p < 1.0) ? 0.0 : (p > 2.0) ? 3.0 - p : p - 1.0);
            wait (0.0025);
        }
    }
}
```

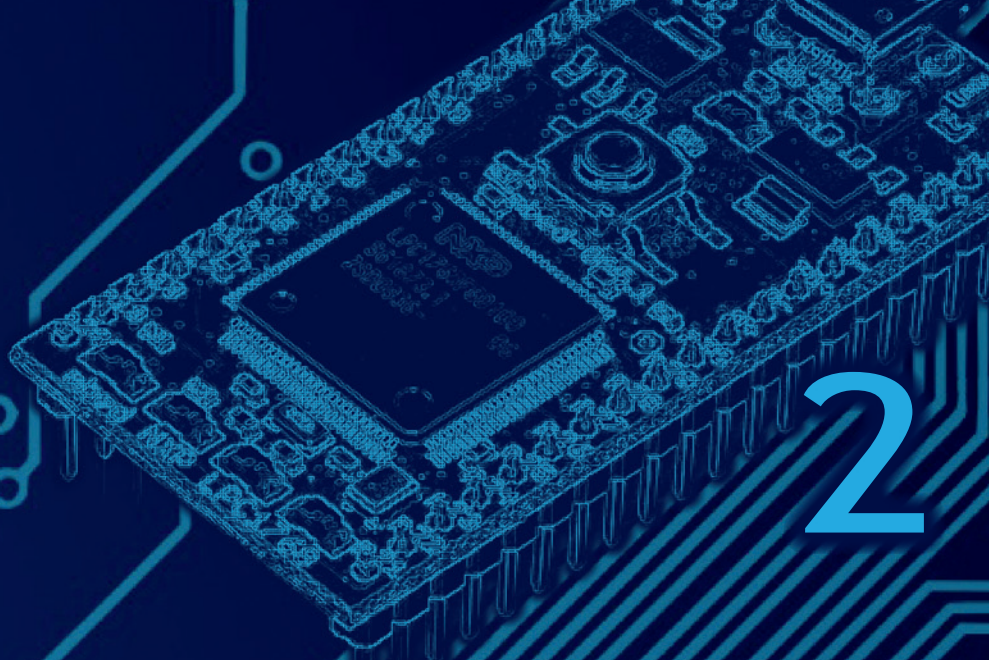
contare sull'aiuto dell'handbook, del cookbook e del forum dedicato al mondo mbed.

Con questo corso abbiamo concluso; torneremo sull'argomento quando la comunità di sviluppatori mbed sfornerà novità rilevanti. Comunque non mancheremo di pubblicare, in futuro, ulteriori approfondimenti che vi permettano di tenervi aggiornati sul mondo degli mbed che, per quanto viva un percorso diverso e lontano dalle "luci della ribalta" come Arduino o Raspberry Pi, è ugualmente interessante e meritevole di attenzione. ■

Riferimenti

- [1] <http://mbed.org/platforms/>
- [2] <http://mbed.org/platforms/KL25Z/>
- [3] <http://mbed.org/handbook/mbed-FRDM-KL25Z>
- [4] http://cache.freescale.com/files/soft_dev_tools/hardware_tools/schematics/FRDM-KL25Z_SCH.pdf?fp=1
- [5] <http://mbed.org/handbook/mbed-FRDM-KL25Z-Examples>

Scopriamo una scheda di interfaccia e prototipazione che permette di utilizzare con mbed gli shield pensati per Arduino. 2^a puntata.



Conoscere e usare

mbed

dell'ing. Vincenzo Mendola

Come abbiamo già spiegato nella prima puntata di questo corso, l'mbed ha un package DIP di 40 pin, con passo 0.1", che misura soltanto 54x26mm, quindi non molto più grande di un Arduino Nano. Per poter utilizzare in maniera proficua tutte le caratteristiche del nostro sistema di prototipazione, è conveniente disporre di un circuito stampato cui poter connettere in maniera stabile e affidabile il nostro mbed e che sia in grado inoltre di fornirci

un supporto meccanico adeguato. Il codice per sviluppare applicazioni con l'mbed è diverso da quello utilizzato da Arduino, tuttavia nella prima puntata abbiamo mostrato che è possibile eseguire in maniera equivalente le stesse funzioni di gestione dei GPIO, ADC, ecc. Allora, perché non realizzare un unico PCB che, oltre a fornire un utile supporto alle connessioni del nostro mbed, ci permetta anche di impiegare gli innumerevoli shield di Arduino? Ci siamo posti questa domanda e in risposta ab-

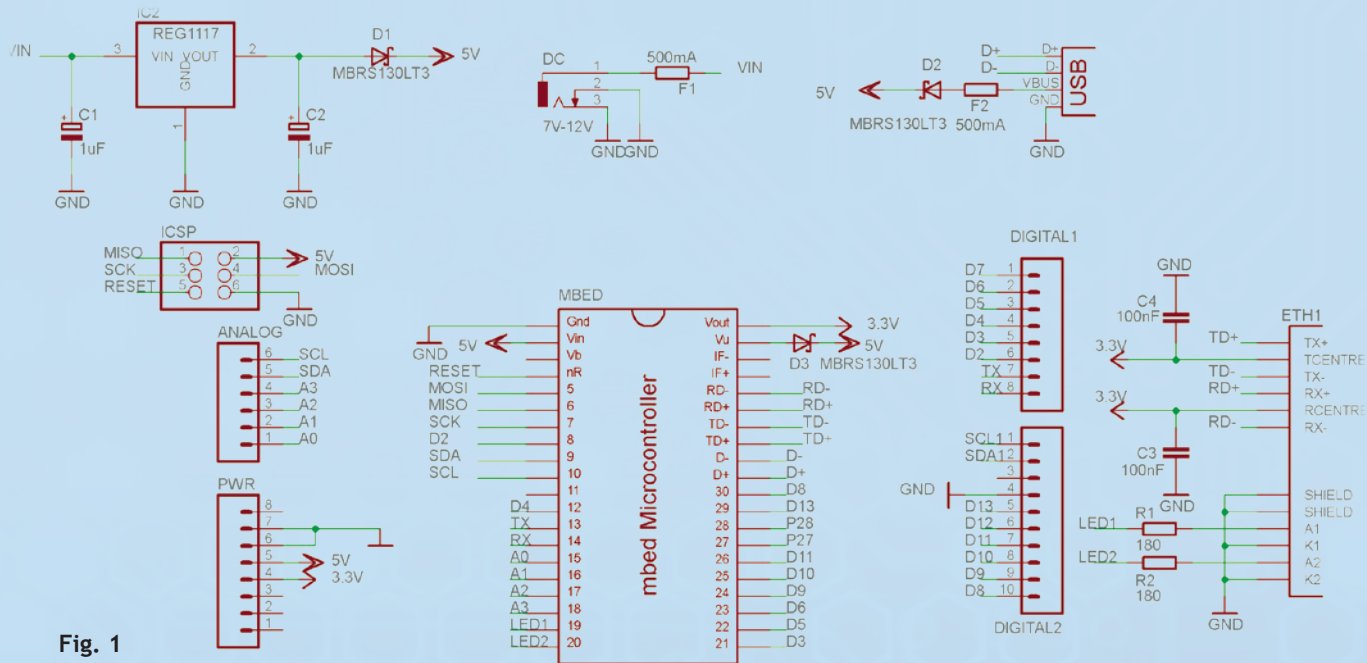


Fig. 1

biamo progettato un circuito stampato di supporto e interconnessione, che soddisfa tutti i requisiti richiesti: compatibilità con gli shield già esistenti e facilità d' accesso alle principali periferiche dell'mbed, tra cui l'ethernet e l'USB. Vediamolo nel dettaglio: in Fig. 1 è riportato lo schema elettrico, mentre in Fig. 2 è visibile una possibile realizzazione della

scheda di questo circuito, che possiamo ritenere un vero e proprio adattatore. Sono presenti gli strip ai quali connettere direttamente l'mbed, gli strip per gli shield di Arduino con il loro caratteristico passo, un connettore USB di tipo B, un connettore RJ45 per interfacciarsi alla rete, i terminali ICSP ed un plug a cui connettere un alimentatore esterno in continua che fornisca una tensione di valore compreso tra 7 e 12 V; quando si alimenta il circuito tramite il plug, si consiglia di non eccedere i 7 V per evitare di dissipare inutilmente potenza nel regolatore lineare a tre terminali a basso dropout (Low Dropout Output regulator) REG1117-5.

Questo integrato, prodotto dalla Texas Instruments, è disponibile in due varianti:

- la versione "base", caratterizzata da un drop-out di 1,2 V (per funzionare correttamente fornendo in uscita i 5 V previsti deve avere all'ingresso una tensione di almeno 6,2 V) e da una corrente massima erogabile di 800 mA, che vengono progressivamente ridotti automaticamente superando i 10 V all'ingresso per evitare un eccessivo riscaldamento del chip;
- la versione "advance", indicata con l'aggiunta di una "A" nel codice di identificazione del prodotto (REG1117-5A), che ha un drop-out di 1,3 V alla massima corrente erogabile (pari ad 1A), anche in

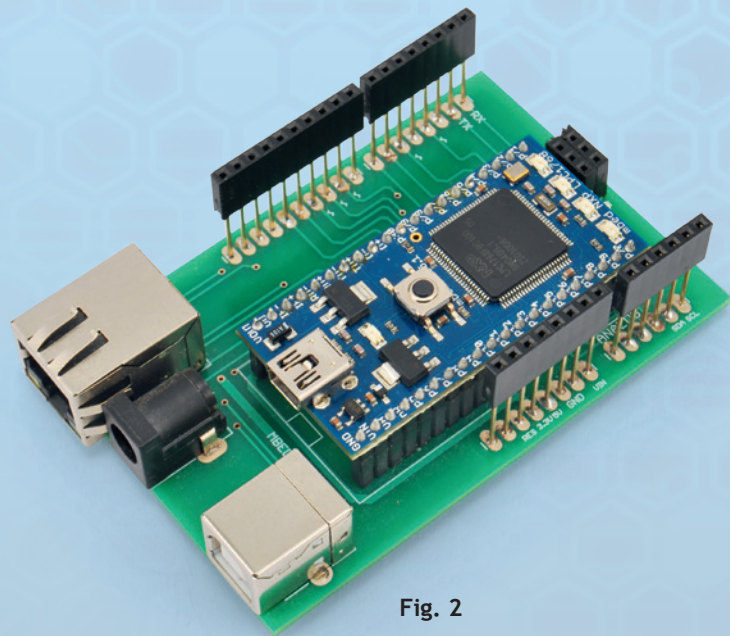


Fig. 2

Listato 1

```

#include "mbed.h"
#include "ds1307.h"

Serial pc(USBTX, USBRX);

DS1307 my1307(p9,p10);

int sec = 00;          // Values to set time
int min = 00;
int hours = 12;
int day = 7;
int date = 3;
int month = 3;
int year = 13;

void test_rw(int test)
{
    if (test == 0)
    {pc.printf("%.2D:",hours);
    pc.printf("%.2D:",min);
    pc.printf("%.2D \n\r",sec);
    pc.printf("%.2D/",date);
    pc.printf("%.2D/",month);
    pc.printf("%.2D \n\r",year);
    pc.printf("\n\r");}

    else pc.printf("Last R/W operation failed!\n\r");

    wait(1);
}
int main ()
{
    //test_rw(my1307.settime( sec, min, hours, day, date, month, year)); wait(3); // Required only during
    the first RTC connection to set the RTC.

    while(1){
    test_rw(my1307.gettime( &sec, &min, &hours, &day, &date, &month, &year)); // Read the time and display on
    screen
    }
}

```

questo caso automaticamente ridotta per tensioni d'ingresso maggiori di 10 V.

In entrambi i casi la massima tensione applicata all'ingresso del regolatore non può eccedere i 15 V, pena il danneggiamento irreversibile del chip. Nel nostro circuito possono essere utilizzate entrambe le versioni, dato che hanno la stessa piedinatura; per quanto riguarda la corrente di esercizio, difficilmente verrà superato il valore di 500 mA, che poi è la corrente standard erogabile da una comune porta USB. Per la stessa ragione abbiamo scelto il più compatto case SOT223, che difficilmente sarà sottoposto a grandi stress termici, meglio tollerati dalla più massiccia versione in contenitore DDPACK.

Come è visibile dallo schema elettrico, sia il plug DC che la USB sono protetti dai cortocircuiti mediante protezioni autoripristinanti ottenute mediante l'uso

di poliswitch da 500 mA, che abbiamo voluto inserire anche se generalmente le porte USB dei PC sono già munite di una loro protezione; in questo modo si può stare tranquilli anche qualora si decida di alimentare il circuito mediante USB non standard o presenti su alimentatori autocostruiti come quelli che presenteremo nei prossimi numeri della rivista. I diodi Schottky D1 e D2 servono per separare le sorgenti di alimentazione dato che, come abbiamo visto, il nostro mbed può essere alimentato sia tramite USB, sia tramite il DC plug.

Sempre sulla linea di alimentazione, possiamo notare D3, il quale, in maniera analoga a quanto appena visto, ha la funzione di garantire che il terminale d'uscita a 5 V del regolatore LDO presente sul PCB dell'mbed possa solo erogare corrente e non diventare un carico. I catodi di questi tre diodi sono fra loro connessi, garantendo

Listato 2

```
#include "mbed.h"

int main() {

    // get the current time from the terminal
    struct tm t;
    printf("Enter current date and time:\n\r");
    printf("YYYY MM DD HH MM SS[enter]\n\r");
    scanf("%d %d %d %d %d %d", &t.tm_year, &t.tm_mon, &t.tm_mday, &t.tm_hour, &t.tm_min, &t.tm_sec);

    // adjust for tm structure required values
    t.tm_year = t.tm_year - 1900;
    t.tm_mon = t.tm_mon - 1;

    // set the time
    set_time(mktime(&t));

    // display the time
    while(1) {
        time_t seconds = time(NULL);
        printf("%s \r", ctime(&seconds));
        wait(1);
    }
}
```

do in questo modo una corretta tensione di alimentazione al pin Vin del mbed. Alla luce di quanto appena visto, al fine di ridurre le cadute di tensione ai loro capi, i tre diodi devono essere necessariamente di tipo Schottky, come ad esempio l'PME-G2005AEA della NXP da noi scelto, che

garantisce una piccola caduta di tensione (400mV@IF=500mA); C1 e C2 sono i due condensatori di filtro al tantalio da 10 µF, della stessa tipologia consigliata dal datasheet; R1 e R2 (da 180 ohm) sono le due resistenze di limitazione della corrente che scorre nei LED inseriti all'interno

Listato 3

```
#include "mbed.h"
#include "EthernetInterface.h"
#include "NTPClient.h"

EthernetInterface eth;
NTPClient ntp;

int main()
{
    eth.init(); //Use DHCP

    eth.connect();

    printf("Trying to update time...\r\n");
    if (ntp.setTime("ntp1.inrim.it") == 0) // INRIM italian NTP server
    {
        printf("Set time successfully\r\n");
        time_t ctTime;
        ctTime = time(NULL);
        printf("Time is set to (UTC): %s\r\n", ctime(&ctTime));
    }
    else
    {
        printf("Error\r\n");
    }

    eth.disconnect();

    while(1) {
    }
}
```

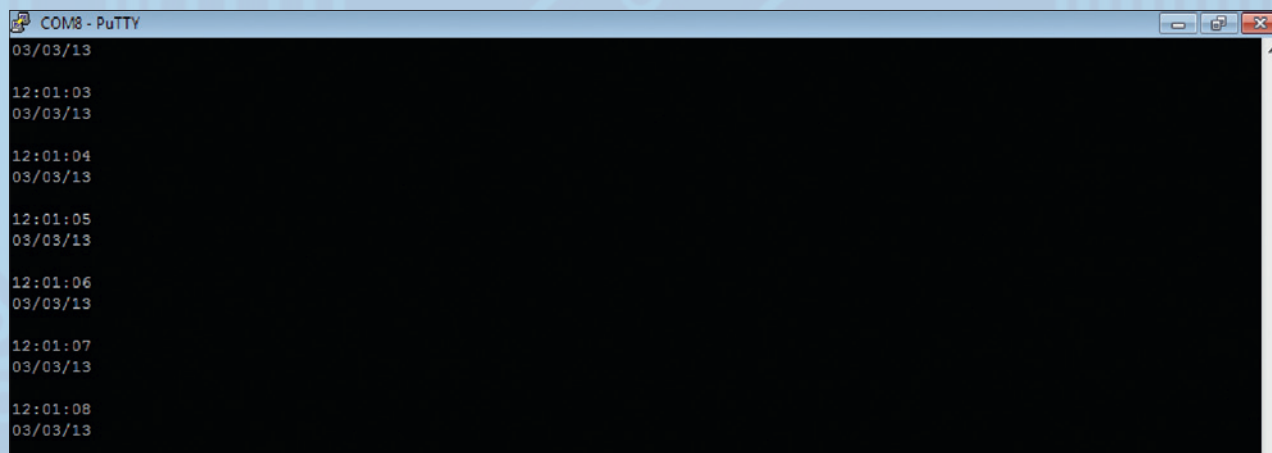


Fig. 3

del connettore RJ45 dotato di filtri magnetici (si può acquistare da Futura Elettronica, www.futurashop.it, con il codice 7300-RJ45EM) che possono essere utilizzati per segnalare la connessione dell'mbed ad una rete ethernet e per evidenziare lo scambio di dati su di essa. C3 e C4 servono per disaccoppiare la tensione di 3,3 V come riportato nella pagina del cookbook dedicata (punto [1] dei Riferimenti).

Il pcb è stato disegnato per ottenere la massima compatibilità con gli shield di Arduino; la **Tabella 1** illustra la corrispondenza tra i pin dell'mbed con quelli di Arduino, un'informazione necessaria, durante la scrittura del codice, per garantire una perfetta equivalenza delle funzionalità con gli shield già esistenti.

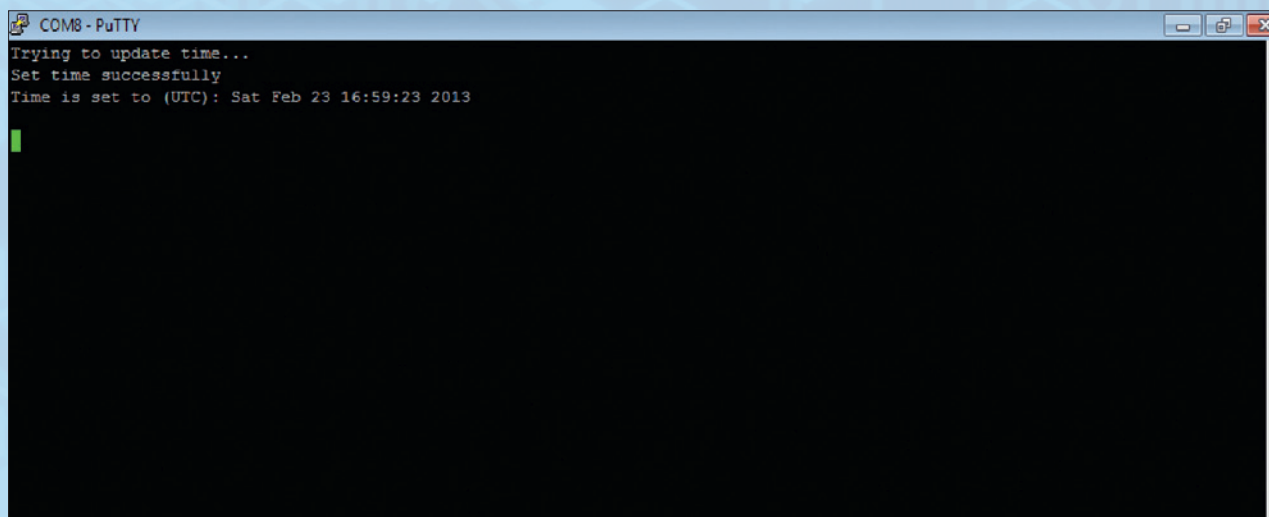
Come si può notare, sono disponibili tutti i GPIO dell'Arduino Uno (D2-D13), i sei PWM (D3, D5, D6, D9, D10, D11), la seriale (D0, D1), la I2C (A4-A5), le connessioni all'ICSP e 4 ADC (A0-A3), tutti disponibili nella stessa posizione in cui lo sono sull'Arduino. Per l'mbed LCP1768 è anche disponibile il DAC, che risulta connesso al pin 18 dell'mbed e all'A3 del nostro PCB. Ora che abbiamo a disposizione questo comodo circuito, che ci permette di utilizzare con facilità gli shield di Arduino, possiamo ad esempio collegare all'mbed l'RTC realizzato per Arduino e presentato nel Fascicolo n° 163 (febbraio 2012) della nostra rivista, con l'unica accortezza di ponticellare con dei fili le due connessioni a SDA e SCL (replicate nelle ultime revisioni di Arduino UNO alle estremità del connettore dei GPIO, oltre il D13) con i

pin analogici A4 (SDA) e A5 (SCL) della Arduino stessa.

Ora, prima di procedere è importante fare una precisazione: gli mbed sono 5V-tolerant (cioè accettano livelli TTL, pur

ARDUINO	MBED	FUNCTION
D0	p14	
D1	p13	
D2	p8	
D3	p21	PWM
D4	p12	
D5	p22	PWM
D6	p23	PWM
D7	p27	
D8	p30	
D9	p24	PWM
D10	p25	PWM
D11	p26	PWM
D12	p28	
D13	p29	
A0	p15	
A1	p16	
A2	p17	
A3	p18	DAC
A4	P9	SDA
A5	P10	SCL
ICSP	P5	MOSI
ICSP	P6	MISO
ICSP	P7	SCK

Tabella 1



```

COM8-PuTTY
Trying to update time...
Set time successfully
Time is set to (UTC): Sat Feb 23 16:59:23 2013

```

Fig. 4

prediligendo quelli 0/3,3V) ma, per essere certi di evitare danni irreparabili al chip, è sempre consigliabile utilizzare un traslatore di livello tipo quello presentato per Raspberry Pi nel fascicolo di febbraio di quest'anno, oppure lavorare direttamente con logica a 3,3 V, ovviamente anche per il Bus I²C.

Lo shield RTC di Arduino lavora con un I²C a livello TTL (0/5V) per cui è necessaria molta attenzione qualora lo si voglia connettere direttamente all'mbed.

Sappiamo che l'LCP1768, a differenza della versione LPC11U24, incorpora già un modulo RTC, ma comunque ad entrambi è possibile connettere un DS1307 della Maxim, servendosi di un'apposita libreria (ad esempio quella disponibile all'indirizzo http://mbed.org/users/harrypowers/code/DS1307/docs/c3e4da8feb10/ds1307_8h_source.html) e del semplice codice riportato nel **Listato 1**.

Nella **Fig. 3** è visibile la schermata di terminale della seriale, contenente i dati in uscita dall'RTC. Notate che è possibile utilizzare anche l'RTC a bordo dell'LPC1768 e del nuovo modello FRDM-KL25Z, servendosi del codice (punto [2] dei Riferimenti) contenuto nel **Listato 2**.

Come si vede dalla lettura delle poche istruzioni necessarie, l'RTC on-board di mbed viene impostato direttamente inserendo la data e l'ora tramite seriale, sulla quale viene costantemente aggiornata e visualizzata ogni secondo.

Il nostro PCB di adattamento dispo-

ne anche di un connettore ethernet che consente di connettere l'LPC1768 alla rete LAN (Local Area Network) e WAN (Wide Area Network); perciò vi proponiamo un semplice esempio di utilizzo della libreria ETHERNET che permette di determinare l'ora esatta tramite un server NTP (Network Time Protocol) qualora abbiate connesso il vostro mbed alla rete INTERNET e vogliate sfruttare l'altissima precisione fornita dagli orologi atomici (il codice è illustrato nel **Listato 3**). Per i nostri test ci siamo serviti del primo dei due server NTP primari installati nel Laboratorio di Tempo e Frequenza Campione dell'INRIM, accessibili liberamente. Ulteriori informazioni sono disponibili alla pagina http://www.inrim.it/ntp/index_i.shtml. La libreria ed il relativo codice sono disponibili nella pagina http://mbed.org/users/donatien/code/NTPClient_HelloWorld/.

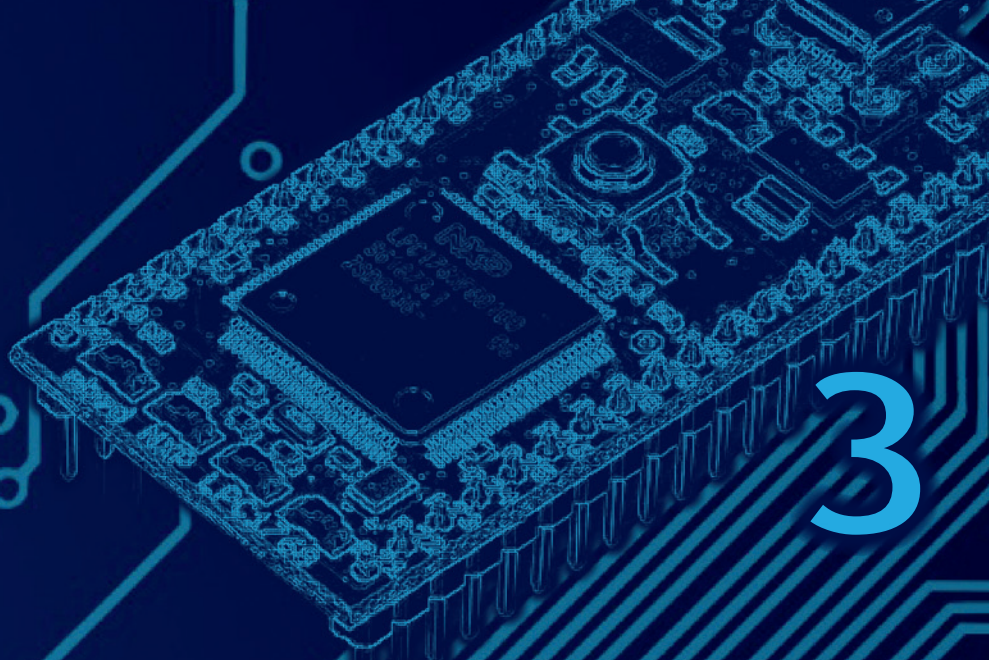
Nella **Fig. 4** si vede il risultato della sincronizzazione con il server NTP.

Bene, con questo abbiamo concluso anche questa puntata; nella prossima mostreremo ulteriori esempi di utilizzo del nostro circuito di adattamento e approfondiremo la conoscenza delle schede di prototipazione per mbed. ■

Riferimenti

- [1] <http://mbed.org/cookbook/Ethernet-RJ45>
- [2] <http://mbed.org/blog/entry/103/>

Utilizziamo la nostra piattaforma di prototipazione per pilotare un display LCD mediante uno shield Arduino in modo da visualizzare scritte a piacere e l'ora dell'RTC, ma anche per generare forme d'onda usando il DAC del LPC176. Terza puntata.



3

Conoscere e usare

mbed

dell'ing. Vincenzo Mendola

Nelle puntate precedenti abbiamo introdotto le schede di prototipazione mbed, mostrandovi le nozioni base per il loro utilizzo per aiutarvi a prendere confidenza con il relativo hardware; abbiamo inoltre introdotto un comodo circuito adattatore che ci permette di utilizzare con facilità gli shield di Arduino sulla piattaforma mbed a 32 bit. Con questo adattatore abbiamo iniziato a realizzare alcune applicazioni, descrivendo di volta in volta il relativo codice. In questa puntata prosegua-

mo la descrizione delle applicazioni e lo facciamo proponendo la gestione di vari display standard (a controllo parallelo) a cristalli liquidi, ma anche la generazione di segnali di forma d'onda più comune. Partiamo con il primo esempio applicativo, utilizzando l'LCD shield per Arduino, presentato sul numero 174 di marzo 2013 di Elettronica In (e commercializzato da Futura Elettronica con la sigla 7100-FT1030K) per mostrare qualche esempio di codice che permette ad mbed di interfacciarsi ad un comune LCD. Per prima cosa provvediamo alla

Listato 1

```
#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p30, p24, p12, p22, p23, p27,
TextLCD::LCD16x2);

int main()
{
    lcd.locate(0,0);
    lcd.printf("MBED SHIELD");
    lcd.locate(0,1);
    lcd.printf("ELETTRONICA IN");
}
```

visualizzazione delle scritte "MBED SHIELD" e "ELETTRONICA IN" (**Listato 1**).

Com'è facile intuire, la libreria *TextLCD* serve per rappresentare caratteri alfanumerici su LCD basati sul diffusissimo driver HD44780 o controller compatibili. Fra parentesi sono riportati, nell'ordine, i pin dell'mbed connessi ai terminali "RS", "E", "D4-D7" del modulo LCD, mentre "*TextLCD::LCD16x2*" serve per definire la tipologia del modulo connesso. Attualmente sono supportati i display LCD, elencati nella **Tabella 1**, ognuno con la relativa istruzione da includere nel codice. Tornando al **Listato 1**, l'istruzione "*lcd.locate(0,1);*" analogamente a quanto avviene per Arduino, indica la colonna e la riga in cui visualizzare il testo tra le virgolette del comando "*lcd.printf(*\"*\"*\");".

In maniera analoga al codice di Arduino, la riga *#include "TextLCD.h"* richiama la libreria per la scrittura di caratteri sugli LCD, mentre la riga "*TextLCD lcd(p27, p23, p22, p12, p21, p8, TextLCD::LCD16x2);*" serve per definire i collegamenti tra l'mbed (connesso in modalità a 4 BIT) e il display, oltre che la tipologia (due righe e 16 caratteri per riga); il comando "*lcd.printf("ELETTRONICA IN\n");*" serve per scrivere il testo desiderato. Come per tutte le funzioni e le librerie di mbed, è possibile approfondire l'argomento mediante il cookbook (<https://mbed.org/cookbook/Homepage>) e l'handbook (<https://mbed.org/handbook/Homepage>).

Listato 2

```
#include "mbed.h"
#include "TextLCD.h"

TextLCD lcd(p27, p23, p22, p12, p21, p8, TextLCD::LCD16x2); //meteo shield

int main() {
    lcd.printf("ELETTRONICA IN\n");
}
```

Tabella 1

Tipo di display	Istruzione
8x1 LCD panel	TextLCD::LCD8x1
8x2 LCD panel	TextLCD::LCD8x2
16x1 LCD panel	TextLCD::LCD16x1
16x4 LCD panel	TextLCD::LCD16x4
24x2 LCD panel	TextLCD::LCD24x2
24x4 LCD panel	TextLCD::LCD24x4
40x2 LCD panel	TextLCD::LCD40x2
40x4 LCD panel	TextLCD::LCD40x4

ge). Qualora si volesse realizzare un proprio LCD shield, basta semplicemente modificare i parametri riportati fra le parentesi del comando *TextLCD* in modo da rispettare le connessioni utilizzate nel proprio PCB. Per esempio possiamo servirci del meteo shield (fascicolo n° 171 di *Elettronica In*) ed utilizzarlo per scrivere del testo a piacimento (**Fig. 1**). Il relativo codice è quello nel **Listato 2**.

Passiamo ora ad un esempio un po' più complesso: proviamo a visualizzare l'ora e la data del nostro RTC shield su un comune display LCD 16x2; il codice è quello mostrato nel **Listato 3**.

Ricordiamo che l'RTC shield e l'RTC presente sul meteo shield, utilizzano entrambi il DS1307, che ha una logica operante a 5V, mentre gli mbed lavorano a 3,3V; anche se sono "5V tolerant" (possono accettare al loro ingresso tensioni di 5V e leggerle correttamente come livello alto senza danneggiarsi) si



Fig. 1

Listato 3

```

#include "mbed.h"
#include "TextLCD.h"
#include "ds1307.h"

TextLCD lcd(p27, p23, p22, p12, p21, p8, TextLCD::LCD16x2); //meteo shield
Serial pc(USBTX, USBRX);
DS1307 my1307(p9,p10);

int sec = 0;
int min = 55;
int hours = 20;
int day = 7;
int date = 3;
int month = 3;
int year = 13;

void test_rw(int test)
{
    if (test == 0) pc.printf("Last R/W operation passed!\n\r");
    else pc.printf("Last R/W operation failed!\n\r");
}

int main()
{test_rw(my1307.settime( sec, min, hours, day, date, month, year)); // to set RTC
while(1){ test_rw(my1307.gettime( &sec, &min, &hours, &day, &date, &month, &year));
    lcd.locate(0,0);
    lcd.printf("% .2D", hours);
    lcd.printf(" : %.2D", min);
    lcd.printf(" : %.2D", sec);

    lcd.locate(0,1);
    switch(day){
    case 1:
        lcd.printf("LUN");
        break;
    case 2:
        lcd.printf("MAR");
        break;
    case 3:
        lcd.printf("MER");
        break;
    case 4:
        lcd.printf("GIO");
        break;
    case 5:
        lcd.printf("VEN");
        break;
    case 6:
        lcd.printf("SAB");
        break;
    case 7:
        lcd.printf("DOM");
        break;
    }

    lcd.printf(" % .2D", date);
    lcd.printf("/%.2D", month);
    lcd.printf("/%.2D", year);
}
}

```

consiglia di utilizzare un traslatore di livello (5V/3,3V) bidirezionale, ossia un circuito in grado di trasformare in 0/3,3V i livelli TTL in ingresso e di elevare a 5 volt i livelli logici alti in uscita dai pin dell'mbed. Il circuito può essere qualcosa di analogo allo shield suggerito nel fascicolo n° 173 della rivista, per adattare RaspberryPi agli shield per Arduino. La riga "switch-case" del Listato 3 può anche essere sostituita dalla seguente porzione di

codice (il risultato resta invariato):

```

if (day == 1)
    lcd.printf("LUN");
if (day == 2)
    lcd.printf("MAR");
if (day == 3)
    lcd.printf("MER");
if (day == 4)
    lcd.printf("GIO");
if (day == 5)
    lcd.printf("VEN");
if (day == 6)
    lcd.printf("SAB");

```

Listato 4

```

#include "mbed.h"
#include "TextLCD.h"
#include "ds1307.h" //http://mbed.org/users/harrypowers/code/DS1307/docs/c3e4da8feb10/ds1307_8h_source.html

Serial pc(USBTX, USBRX);
TextLCD lcd(p27, p23, p22, p12, p21, p8, TextLCD::LCD16x2);

DS1307 my1307(p9,p10);

int sec = 0;
int min = 00;
int hours = 16;
int day = 7;
int date = 23;
int month = 2;
int year = 13;

void test_rw(int test)
{
    if (test == 0) pc.printf("Last R/W operation passed!\n\r");
    else pc.printf("Last R/W operation failed!\n\r");
}

float VADC= 3.31;
float Vcc=4.21;

int DPR = 0;
int RHCORR = 0;
int PCORR = 0;

float STAMPA_T = 0;
float STAMPA_U = 0;
float STAMPA_P = 0 ;

AnalogIn val(p17);
AnalogIn UM(p16);
AnalogIn Pascal(p15);

float temp()
{
    float T=0.0;
    float nread = 100.0;          // NUMBER OF READINGS
    float somma = 0.0;
    for (int i=0; i<nread; i++) {
        float TADC=val.read();
        T = ((VADC*TADC)-0.5)* 100; //TEMPERATURE
        somma += T;
    }
    wait_ms(100);
    return (somma/nread);
}

float readUMID()
{
    float RHout=0.0;
    float nread = 100.0;          // NUMBER OF READINGS
    float somma = 0.0;
    for (int i=0; i<nread; i++) {
        float UMADC = UM.read();
        RHout=((UMADC)-0.1515)/0.00636)+RHCORR; //HUMIDITY
        somma += RHout;
    }
    wait_ms(100);
    return (somma / nread);
}

float pressure()
{
    float P=0.0;
    float nread = 100.0;          // NUMBER OF READINGS
    float somma = 0.0;
    for (int i=0; i<nread; i++) {
        float PascalADC= Pascal.read();
        P=(((PascalADC*VADC)/Vcc+0.095)/0.009)*10+DPR+PCORR; //PRESSURE TRANSFERT FUNCTION
        somma += P;
    }
}

```

```

    wait_ms(100);
    return ( somma / nread );
}

int main()
{
//test_rw(my1307.settime( sec, min, hours, day, date, month, year)); //REGOLA ORARIO E DATA
while (true) {

    STAMPA_T= (temp());
    STAMPA_U= (readUMID());
    STAMPA_P = (pressure());
    lcd.locate(0, 0);
    lcd.printf("%2.1f",STAMPA_T); //SHOW ONLY THE FIRST DECIMAL
    wait_ms(200);
    lcd.locate(4,0);
    lcd.printf("C");
    lcd.locate(6, 0);
    lcd.printf("%2.1f",STAMPA_U); //SHOW ONLY THE FIRST DECIMAL
    lcd.locate(10,0);
    lcd.putc(37);
    wait_ms(200);

    lcd.locate(12, 0);
    lcd.printf("%4.0f",STAMPA_P); //SHOW ONLY THE INTEGER PART
    wait_ms(200);

    test_rw(my1307.gettime( &sec, &min, &hours, &day, &date, &month, &year));
    lcd.locate(0,1); // Print and refresh data on line 2 of the LCD display
    lcd.printf("%.2D",hours);
    lcd.printf("%.2D",min);
    //lcd.printf("%.2D",sec);
    //lcd.printf("%.1D",DAY);

    if (day == 1)
        lcd.printf(" D");
    if (day == 2)
        lcd.printf(" L");
    if (day == 3)
        lcd.printf(" Ma");
    if (day == 4)
        lcd.printf(" Me");
    if (day == 5)
        lcd.printf(" G");
    if (day == 6)
        lcd.printf(" V");
    if (day == 7)
        lcd.printf(" S");

    lcd.printf(" %.2D",date);
    lcd.printf("/%.2D",month);
    lcd.printf("/%.2D",year);

}
}

```

```

if (day == 7)
    lcd.printf("DOM");

```

A questo punto vediamo come modificare il codice scritto per Arduino allo scopo di utilizzare il meteo shield con l'mbed. Per prima cosa ricordiamo che sul PCB dello shield sono presenti due ponticelli che permettono di selezionare la corretta tensione di alimentazione del sensore di pressione, previsti a suo tempo per facilitare l'uso del circuito anche con i sensori a 3,3V (ad esempio il MP3H6115A), tensione usata dai microcontrollori di mbed e da Arduino Due. Prima di mostrare una possibile versione del codice per mbed, ricordiamo che il modello LCP1768 incorpora già un RTC, mentre la versione LPC11U24 no, per cui abbiamo inserito anche la libreria per il DS1307, scaricabile all'indirizzo web <http://mbed>.

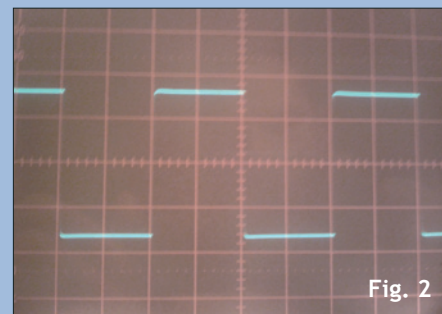


Fig. 2

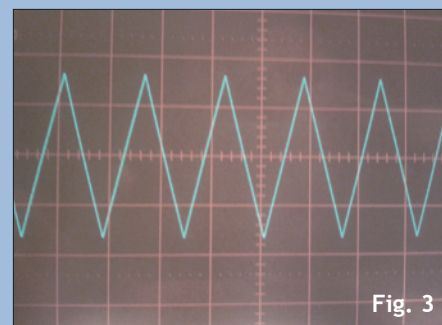


Fig. 3

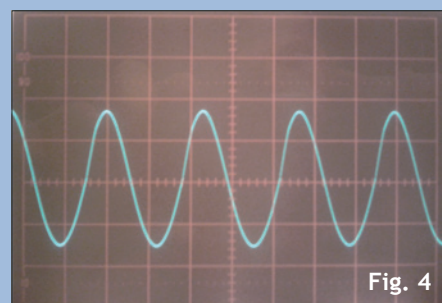
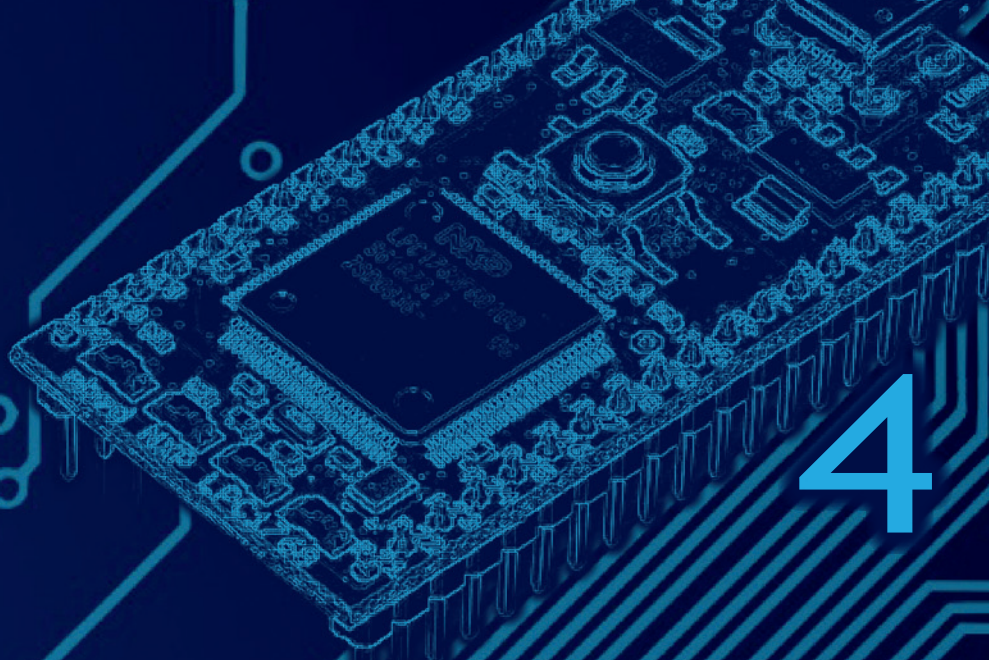


Fig. 4

Realizziamo la nostra prima applicazione con una board mbed ready: la KL25Z prodotta da Freescale. Quarta ed ultima puntata.



Conoscere e usare

mbed

dell'ing. Vincenzo Mendola

Nell'ultimo anno abbiamo assistito alla nascita di tantissime nuove board embedded destinate al mondo hobbystico o comunque al mercato di fascia medio-bassa: sulla scia di Arduino e di Raspberry Pi hanno fatto la loro comparsa PcDuino, BeagleBoard, Cubieboard, Odroid, UDOO e tante altre ancora. Schede via via sempre più potenti, addirittura multiprocessore, ma che rimangono in quella fascia di prezzo (massimo 50-100 Euro) alla portata un po' di tutti, studenti, hobbysti e semplici appassionati. Un fiorire di nuovi prodotti che ha visto una convergenza tra il mondo Arduino e il sistema operati-

vo GNU/Linux all'insegna dell'open hardware e dell'open software. La maggior parte delle board presenta infatti la facilità della gestione dei GPIO e delle periferiche tipica di Arduino, mentre la connessione alla rete e quindi ad Internet e la gestione complessiva della board, è garantita da un sistema operativo basato su GNU/Linux. Questa è anche la logica progettuale delle nuove board immesse sul mercato o semplicemente annunciate dal team di Arduino. Parliamo, in primo luogo, di Arduino Yún, la prima scheda di casa Arduino che fa ricorso ad un sistema operativo GNU/Linux denominato Linino, ma anche di Arduino TRE che dovrebbe vedere la luce nella primavera dell'an-

Listato 1

```
#include "mbed.h"

int main() {

    DigitalOut ledG(LED_GREEN);

    while (true) {

        ledG = 1; // off
        wait(0.5);
        ledG = 0; // on
        wait(0.5);

    }

}
```

no prossimo e che utilizzerà un processore Sitara di Texas Instruments ed un sistema operativo GNU/Linux.

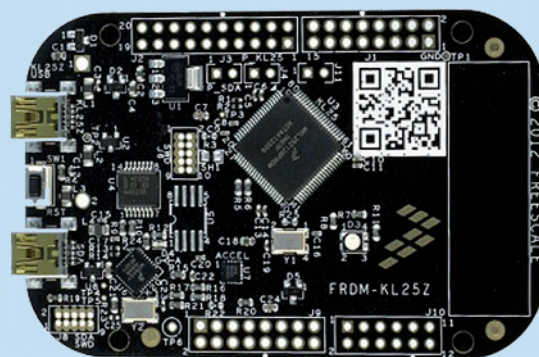
Leggermente diversa è invece l'impostazione della board Intel Galileo che utilizza un processore x86 e nella quale lo sketch di Arduino "gira" all'interno del sistema GNU/Linux come un processo utente.

Indubbiamente la comparsa e la disponibilità di una scheda a bassissimo costo basata su GNU/Linux come Raspberry Pi (ad oggi sono state vendute oltre due milioni di board) ha fatto capire che, il vantaggio di disporre on board di un sistema operativo flessibile (e gratuito!) come Linux sono di gran lunga superiori agli svantaggi che comporta una programmazione un po' più complessa. Tanto fervore con tante nuove schede ha anche avuto il benefico effetto di accentuare la concorrenza tra i produttori di semiconduttori, che si sono prodigati nell'immettere sul mercato prodotti sempre più performanti e sempre più economici. Tutto ciò si è tradotto in una riduzione dei prezzi delle board che ha generato una spirale positiva che ha fatto sì che anche le tecnologie più avanzate potessero risultare disponibili a prezzi incredibilmente bassi.

LA SCHEDA MBED READY KL25Z

Da questa corsa contro il tempo non resta fuori nemmeno il team mbed, che negli ultimi mesi ha introdotto, dopo anni di apparente staticità, numerose innovazioni, passando dalla riduzione generalizzata dei costi dell'hardware, all'aggiunta di nuovi modelli, non direttamente progettati dal team ufficiale, ma compatibili. Dunque, nell'universo mbed, prima conosciuto per i suoi modelli "storici" che ne hanno decretato il successo

Fig. 1



(parliamo della LPC1768 e della LPC1114, di cui al punto [1] dei riferimenti).

Tra le nuove schede della community mbed, quella che ha attirato subito la nostra attenzione è stata la KL25Z sviluppata da Freescale (Fig. 1): venduta al pubblico ad un prezzo veramente concorrenziale (si parla di circa 12 euro) per un prodotto di questo tipo e

Listato 2

```
#include "mbed.h"

int main() {

    DigitalOut ledG(LED_GREEN);
    DigitalOut ledR(LED_RED);
    DigitalOut ledB(LED_BLUE);

    //ALL COLORS OFF
    ledR=1;
    ledG=1;
    ledB=1;

    while (true) {

        //GREEN

        ledG = 0; // on
        wait(0.25);

        ledG = 1; // off
        wait(0.25);

        //RED

        ledR = 0; // on
        wait(0.25);

        ledR = 1; // off
        wait(0.25);

        //BLUE

        ledB = 0; // on
        wait(0.25);

        ledB = 1; // off
        wait(0.25);

    }

}
```

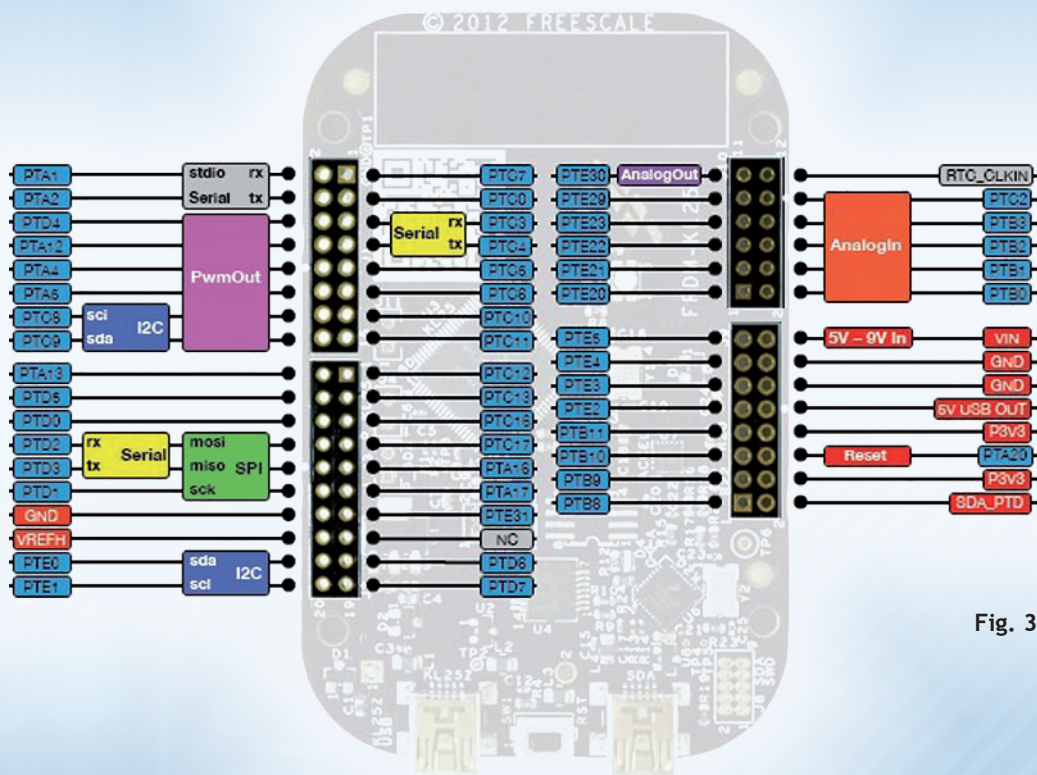


Fig. 3

con queste caratteristiche, la KL25Z è sostanzialmente una board “mbed ready” (così definita per via della possibilità di utilizzare le librerie sviluppate per le board mbed tramite un semplice upload attraverso la presa USB, come dettagliato sul sito ufficiale) dalle interessantissime e peculiari caratteristiche, le quali la differenziano da quelle delle due versioni presentate nelle precedenti puntate. La scheda di Freescale è facilmente upgradabile ad mbed, semplicemente aggiornandone il firmware, come dettagliato nella pagina descrittiva della board di cui al punto [2] dei Riferimenti. Una volta fatto l’aggiornamento, il sistema ci confermerà che la procedura è andata correttamente a termine (con un messaggio come quello riportato nella Fig. 2). La prima cosa che salta subito all’occhio è il form factor di questa nuova board, che è compatibile con quello delle board Arduino; inoltre la KL25Z di Freescale dispone di pad forati sui lati come Arduino, quindi in pratica, basta aggiungere gli header e si possono montare su di esso gli stessi shield facendo

però attenzione al fatto che gli mbed lavorano con logica a 3,3V. Questa caratteristica costituisce un vantaggio importante, in quanto non si può negare che la vastità di shield ed applicazioni disponibili per Arduino è molto allettante e che uno degli elementi che uno sperimentatore spesso valuta prima dell’acquisto di una scheda

Device 'kl25z' connected and added to your account.

Fig. 2

Listato 3

```
#include "mbed.h"

BusOut RGB(PTB18, PTB19, PTD1);

int main() {
//ALL COLORS OFF

RGB=0x07;

while (true) {

    RGB=0x06;
    wait(0.25);
    RGB=0x05;
    wait(0.25);
    RGB=0x03;
    wait(0.25);
}
}
```


Listato 4

```
#include "mbed.h"

DigitalOut ledR(LED_RED);
Serial pc(USBTX, USBRX);

int main() {
    int i=1;
    pc.printf("Hello World!\r\n");

    while (true) {
        wait(1);
        pc.printf("%d\r\n", i);
        i++;
        ledR = !ledR;
    }
}
```

Listato 5

```
#include "mbed.h"
#include "MMA8451Q.h"

#define MMA8451_I2C_ADDRESS (0x1d<<1)

int main(void) {
    MMA8451Q acc(PTE25, PTE24, MMA8451_I2C_
ADDRESS);
    PwmOut rled(LED_RED);
    PwmOut gled(LED_GREEN);
    PwmOut bled(LED_BLUE);

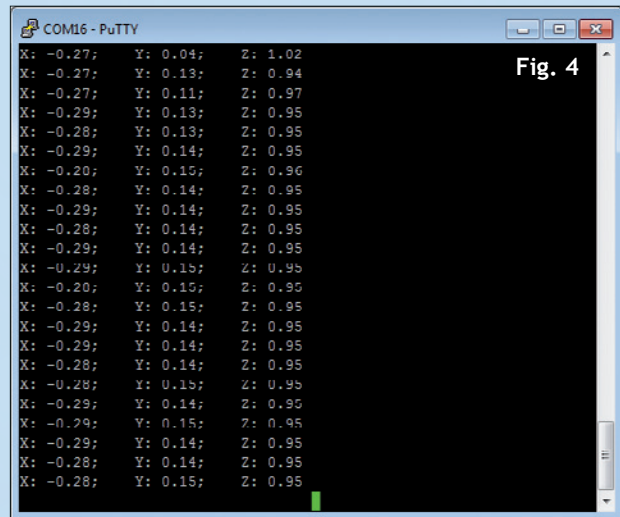
    while (true) {
        rled = 1.0 - abs(acc.getAccX());
        gled = 1.0 - abs(acc.getAccY());
        bled = 1.0 - abs(acc.getAccZ());

        printf("X: %1.2f;    ", acc.getAccX());
        printf("Y: %1.2f;    ", acc.getAccY());
        printf("Z: %1.2f \n\r", acc.getAccZ());
        wait(0.1);
    }
}
```

di prototipazione è la possibilità di utilizzare questo "bagaglio". Ecco perché la compatibilità con il mondo Arduino è attualmente (a ragione o meno) un fattore tenuto molto in considerazione da chi realizza nuove schede di sviluppo e prototipazione.

Vediamo dunque le caratteristiche della scheda Freescale (reperibili sul web alla pagina indicata al punto [3] dei Riferimenti): il cuore della KL25Z è un processore ARM® Cortex™-M0+ Core operante con un clock di 48MHz, con 16 kB RAM e 128 kB di Flash; oltre alle periferiche standard per questa tipologia di tool (2 bus SPI, 2 I²C-Bus, 3 UART, 6 moduli PWM, 6 ADC, vari GPIO), saltano all'occhio delle funzionalità avanzate e di sicuro interesse: a bordo si trovano un accelerometro a tre assi (si tratta di un MMA8451Q della Freescale), un LED RGB controllato in PWM e un touch sensor di tipo capacitivo.

Prima di proseguire è d'obbligo un accenno



alle specifiche elettriche: la tensione di alimentazione della board è compresa tra 5 e 9 volt (in continua), con un assorbimento massimo dichiarato di 100 mA; un regolatore sulla scheda riduce la tensione adattandola a quella richiesta dalla logica a 3,3 V, che caratterizza da tempo la maggior parte dei dispositivi simili al Freescale. Ciascun terminale di I/O della board è in grado di erogare 4 mA. In Fig. 3 è visibile il dettaglio del pinout del KL25Z.

Proviamo a prendere confidenza con la KL25Z partendo da alcuni semplici esempi applicativi. Il primo riguarda la gestione del LED RGB disponibile sul PCB, che otteniamo con un semplice esempio che ci permette di far lampeggiare il solo colore verde: lo trovate nel Listato 1. Se guardate lo schema elettrico del KL25Z (punto [4] dei Riferimenti) noterete che il LED RGB sulla scheda è di tipo ad anodo comune, quindi per pilotarlo è necessario fornire un livello basso al singolo LED da accendere; in questo caso è stato scelto il verde (LED_GREEN). Come vedete, il codice è veramente semplice ed auto esplicativo.

Vediamo ora, nel Listato 2, come far lampeggiare tutti e tre i colori che il LED montato sulla scheda mbed può produrre.

L'applicazione proposta non è affatto complessa; è solo un esempio di base.

La stessa cosa si sarebbe potuta ottenere servendosi delle istruzioni contenute nel Listato 3, dedotte osservando in dettaglio lo schema elettrico; guardare lo schema è una cosa che consigliamo sempre di fare, perché vi permette di capire maggiormente le caratteristiche dell'hardware che stiamo utilizzando, permettendoci di sfruttarne al meglio le caratteristiche.

A questo punto del corso riteniamo non sia più necessario spiegare le istruzioni riga per riga,

in quanto dovrete ormai avere il background che vi permette di comprenderle. Ad ogni modo, per chi volesse approfondire, può trovare ulteriori dettagli inerenti a "BusOut" visitando la pagina web <http://mbed.org/handbook/BusOut>.

Bene, detto ciò, alla nostra applicazione andiamo ad aggiungere la comunicazione con la seriale, la quale viene implementata grazie al codice che trovate nel **Listato 4**.

Passiamo quindi ad un esempio che combina l'uso dell'accelerometro con quello del LED RGB: il valore dell'accelerazione sui 3 assi è visibile sia in maniera numerica tramite la seriale del nostro mbed (**Fig. 4**, sia direttamente tramite il colore del LED sulla scheda, che varia in base al valore misurato. Nel **Listato 5** trovate il codice che permette di ottenere quanto appena detto. Il team di mbed ha creato una libreria per utilizzare in maniera immediata lo slide presente sulla board. Vediamo due semplici applicazioni che lo riguardano, il primo dei quali si basa sul codice che trovate nel **Listato 6**. Il secondo esempio si basa invece sul codice descritto nel **Listato 7**.

Nel primo caso, il LED mantiene visibile il livello di luminosità selezionato tramite lo slide, mentre il codice del **Listato 7** lo accende solo quando toccherete lo slide e la luminosità varierà fintanto che farete scorrere il dito. Per finire la lezione di quest'ultima puntata del corso, ecco un esempio prettamente "scenografico" tratto dalla pagina ufficiale del KL25Z (punto [5] dei Riferimenti) nella quale potrete trovare altri esempi applicativi. Il codice corrispondente da caricare in mbed è quello contenuto nel **Listato 8**.

Bene, ora che avete appreso i rudimenti per lavorare in maniera proficua con le schede mbed, potete proseguire da soli nell'uso e arricchire il vostro bagaglio di applicazioni in base alle vostre esigenze, ricordando che potrete sempre

Listato 6

```
#include "mbed.h"
#include "TSSensor.h"

int main(void) {
    PwmOut led(LED_BLUE);
    TSSensor tsi;

    while (true) {
        while(tsi.readPercentage()>0) {
            led = 1.0 - tsi.
readPercentage();
            wait(0.1);}
    }
}
```

Listato 7

```
#include "mbed.h"
#include "TSSensor.h"

int main(void) {
    PwmOut led(LED_RED);
    TSSensor tsi;

    while (true) {

        led = 1.0 - tsi.readPercentage();
        wait(0.1);
    }
}
```

Listato 8

```
#include "mbed.h"

PwmOut r (LED_RED);
PwmOut g (LED_GREEN);
PwmOut b (LED_BLUE);

int main() {
    r.period(0.001);
    g.period(0.001);
    b.period(0.001);

    while (true) {
        for (float i = 0.0; i < 1.0 ; i += 0.001) {
            float p = 3 * i;
            r = 1.0 - ((p < 1.0) ? 1.0 - p : (p > 2.0) ? p - 2.0 : 0.0);
            g = 1.0 - ((p < 1.0) ? p : (p > 2.0) ? 0.0 : 2.0 - p);
            b = 1.0 - ((p < 1.0) ? 0.0 : (p > 2.0) ? 3.0 - p : p - 1.0);
            wait (0.0025);
        }
    }
}
```

contare sull'aiuto dell'handbook, del cookbook e del forum dedicato al mondo mbed.

Con questo corso abbiamo concluso; torneremo sull'argomento quando la comunità di sviluppatori mbed sfornerà novità rilevanti. Comunque non mancheremo di pubblicare, in futuro, ulteriori approfondimenti che vi permettano di tenervi aggiornati sul mondo degli mbed che, per quanto viva un percorso diverso e lontano dalle "luci della ribalta" come Arduino o Raspberry Pi, è ugualmente interessante e meritevole di attenzione.

Riferimenti

- [1] <http://mbed.org/platforms/>
- [2] <http://mbed.org/platforms/KL25Z/>
- [3] <http://mbed.org/handbook/mbed-FRDM-KL25Z>
- [4] http://cache.freescale.com/files/soft_dev_tools/hardware_tools/schematics/FRDM-KL25Z_SCH.pdf?fp=1
- [5] <http://mbed.org/handbook/mbed-FRDM-KL25Z-Examples>

Listato 5

```
#include "mbed.h"

PwmOut led(LED1);

int main() {
    while(1) {
        for(float p = 0.0; p < 1.0; p += 0.1) {
            led = p;
            wait(0.1);
        }
    }
}
```

Listato 6

```
#include "mbed.h"

PwmOut leds[] = {LED1, LED2, LED3, LED4};
int i;

float p=0.0;

int main()
{
    while(1) {

        for (i=0; i<4; i++) {
            for (p = 0.0; p <= 1.0; p += 0.01) {
                leds[i] = p;
                wait(0.01);
            }
            for (i = 3; i>=0; i--) {
                for (p = 1.0; p >= 0.0; p -= 0.01) {
                    leds[i] = p;
                    wait(0.01);}

                wait(0.1);    }
            }
    }
}
```

Listato 7

```
#include "mbed.h"

AnalogOut DA (p18);

int main()
{
    while(1) {
        DA=1; //ONDA QUADRA
        wait_us(100);
        DA=0;
        wait_us(100);
    }
}
```

org/users/harrypowers/code/DS1307/docs/c3e4da-8feb10/ds1307_8h_source.html. Si noti che tale libreria può comunque anche essere utilizzata con l'mbed basato sul processore Cortex M3. Il codice mbed è quello contenuto nel **Listato 4**. Un'altra funzionalità molto utile nei sistemi basati su microcontrollore è la creazione di segnali PWM, solitamente ottenuta da appositi moduli PWM integrati nei micro stessi. Nel

Listato 8

```
#include "mbed.h"

float i;

AnalogOut DA (p18);

int main() {
    for (i=0; i<1;i=i+0.01) //ONDA TRIANGOLARE
    {DA=i;
    wait_us(1);}

    for (i=1; i>0; i=i-0.01)
    {DA=i;
    wait_us(1);}

}
}
```

Listato 9

```
#include "mbed.h"

float i;

AnalogOut DA (p18);

int main() {
    for (float i=0; i<360; i++) { //ONDA SINUSOIDALE
        DA = sin(i/180*3.14)*0.5+0.5;
        wait_us(1);
    }
}
```

caso della nostra piattaforma mbed, la generazione del PWM è implementata attraverso la funzione PwmOut (<https://mbed.org/handbook/PwmOut>). Nel **Listato 5** e nel **Listato 6** trovate due semplici applicazioni pratiche che sfruttano i quattro LED a bordo dell'mbed LPC1768. Vediamo ora come generare delle semplici forme d'onda utilizzando il DAC (Digital to Analog converter) presente nell'mbed LPC1768. Cominciamo con la generazione di un'onda rettangolare come quella visibile in **Fig. 2**, la quale può essere generata senza alcuna difficoltà con il codice che trovate nel **Listato 7**. Con le piattaforme mbed non ci si limita alla generazione della classica onda quadra, perché è anche possibile generare forme d'onda più complesse come la triangolare (**Fig. 3**) e la sinusoidale (**Fig. 4**), ottenibili rispettivamente con i codici che trovate nei **Listati 8** e **9**. Come sempre, consigliamo di leggere attentamente la documentazione ufficiale per ulteriori dettagli e per imparare a sfruttare tutte le caratteristiche delle funzioni e delle librerie disponibili, che qui per ragioni di spazio, non abbiamo potuto descrivere. Bene, con questo abbiamo concluso. Riprenderemo il discorso nel prossimo fascicolo. ■