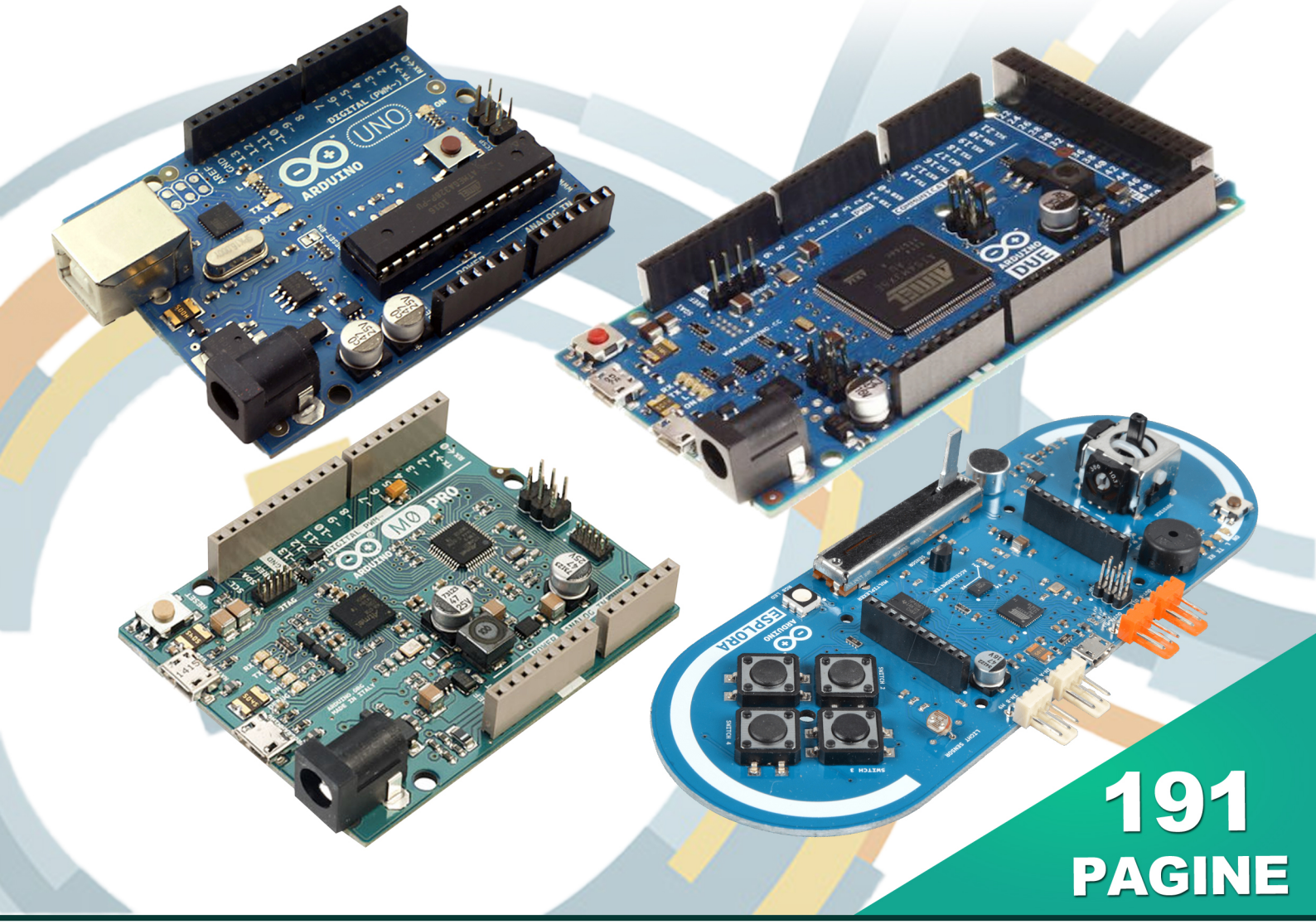
eOS Book @9

ARDUINO®
PROFESSIONALE

TUTORIAL E PROGETTI AVANZATI CON **ARDUINO UNO**, **ARDUINO DUE**, **ARDUINO ESPLORA** E **ARDUINO M0 PRO**



191 PAGINE

Elettronica Open Source

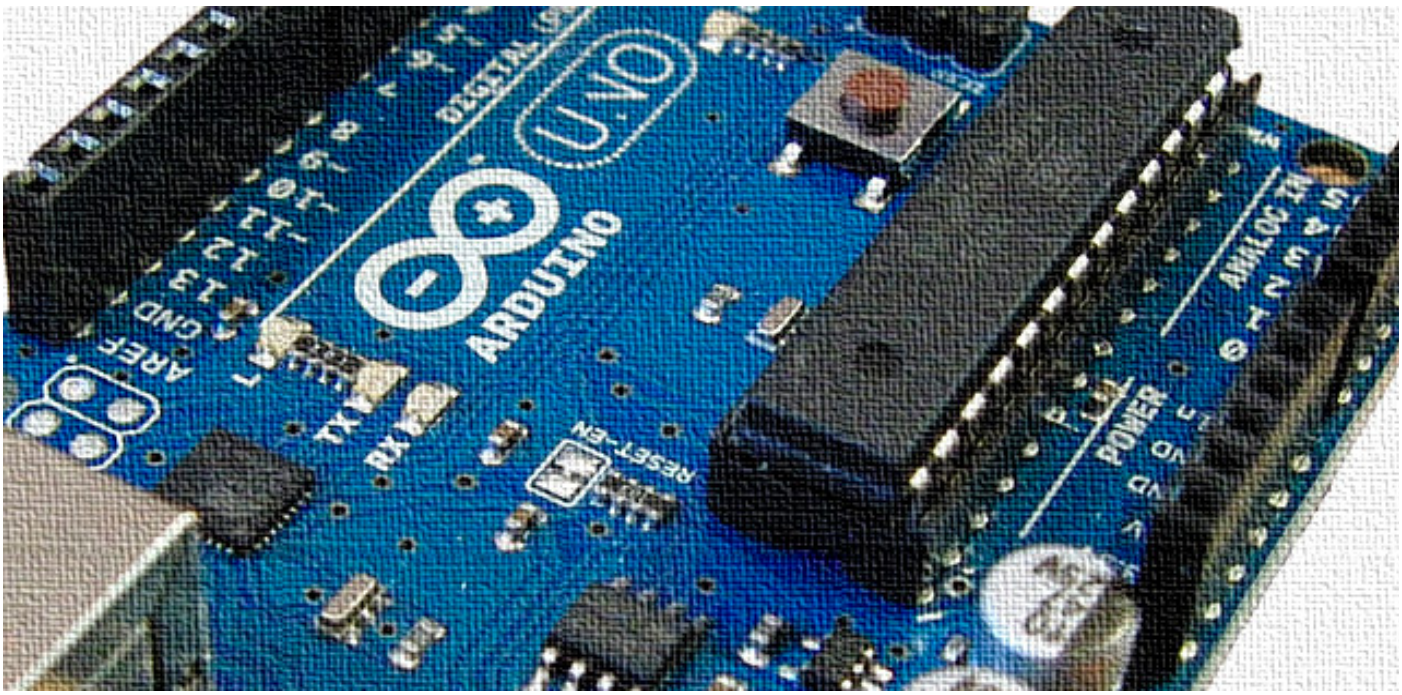
Arduino UNO

Arduino DUE

Arduino ESPLORA

Arduino M0 Pro

Arduino UNO



Arduino ai raggi X: cosa fare per renderlo professionale – Prima Parte

di Emanuele Bonanni

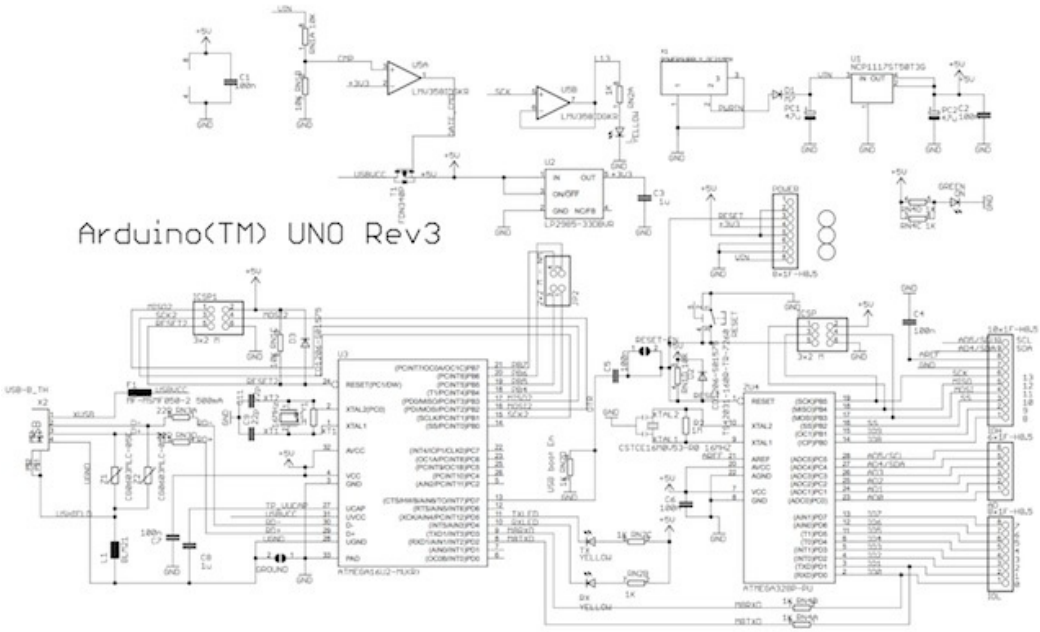
Arduino è una scheda molto semplice da programmare ed è per questo che è molto utilizzata. Però, come dicevamo in apertura, ci sono alcuni problemi che devono essere risolti per poterla rendere anche una scheda affidabile e davvero versatile quando i progetti debbano avere un ambito di applicazione più professionale.

I PROBLEMI DI ALIMENTAZIONE

Vi sarà certamente capitato di scrivere del codice "al volo", per poi rendervi conto che, dopo averlo caricato sulla scheda, aver guardato il LED lampeggiare ed aver atteso la conferma dell'upload, esso non funziona affatto. Controllare il codice, la sintassi, la corrispondenza del numero delle variabili e così via dicendo in queste situazioni non è servito a molto ed infatti il problema non avete capito quale fosse. Se vi siete trovati in questa situazione, sappiate che il vostro problema potrebbe essere di alimentazione!

La scheda lavora molto bene con segnali analogici e digitali ma probabilmente siete incappati in alcuni dei suoi “strani” problemi che, è innegabile, la affliggono e che sono di diversa natura.

Non appena alimentate la scheda, infatti, **nella sezione di alimentazione** della stessa (versione uno), per intenderci appena inserite il connettore coassiale, **l’intero dispositivo viene polarizzato**.



Analizzando lo schema elettrico (PDF) possiamo notare come Arduino possa essere alimentato sia direttamente dal cavo USB che dall’apposito plugin di alimentazione. I casi possibili quindi sono 3: alimentazione da USB, da plugin oppure entrambe.

Alimentazione USB: i 5V provenienti dalla connessione USB alimentano tutta la scheda tramite il mosfet T1 che viene attivato dal comparatore U5A perchè il segnale all’ingresso non-invertente non è presente. Questa condizione permane anche in caso di presenza dell’alimentazione da plugin, finchè quest’ultima non abbia raggiunto il valore di soglia impostato. Infatti il comparatore commutera quando (Vin/2 > 3V3). Quindi, considerando anche il diodo D1, l’alimentazione della scheda sarà collegata all’USB se la tensione Vin del plugin è inferiore a 7V3 (ricavato dalla formula (Vin-0,7)/2 > 3V3).

Alimentazione da plugin: questa alimentazione può variare da 6V a 20V (come riportato nelle specifiche di Arduino come valori limite e da 7V a 12V come valori raccomandati. Quando la scheda è sufficientemente alimentata dal plugin allora il mosfet T1 non sarà in conduzione, proteggendo la porta USB da ritorni di alimentazione.

La tensione di ingresso, dal plugin, giunge al diodo D1 e poi al regolatore U1 dal quale escono appunto i 5V regolati necessari per alimentare l’intera scheda ed i suoi componenti.

Alimentazione da entrambe le fonti: in questo caso l’alimentazione sarà presa dalla fonte che prevarrà, infatti come abbiamo visto sopra, il comparatrore U5A gestirà il mosfet, usato come switch, che si occupa di connettere o meno i 5V USB alla scheda.

Questo è quindi il principio di funzionamento dell’alimentazione della scheda Arduino Uno Rev.3.

Parlando di tensioni non possiamo non notare che **proprio accanto al connettore coassiale sono presenti due condensatori elettrolitici da 47 uF**, utile per immagazzinare energia e quindi livellare la tensione, quando la scheda è alimentata.

Sempre a proposito dell’alimentazione, le specifiche riportano che la massima tensione supportata in ingresso è pari a 20V mentre il limite

di questi condensatori è di solo 16V. **Questa è una discrepanza** notevole e **che potrebbe avere gravi implicazioni sul funzionamento della scheda**.

I condensatori elettrolitici, infatti, corrono rischi notevoli se vengono spinti anche soltanto al limite delle loro possibilità. Figuriamoci se queste vengono superate.

Prima di procedere, vale la pena di riportare qui direttamente le caratteristiche riportate sulle specifiche di Arduino:

- **tensione operativa:** 5 V;
- **tensione d'ingresso raccomandata:** compresa tra 7 e 12 V;
- **tensione di ingresso (limiti):** compresa tra 6 e 20 V;
- **numero di pin di I/O digitali:** 14 (di cui 6 PWM);
- **numero di pin analogici di ingresso:** 6;
- **corrente DC per pin:** 40 mA;
- **corrente DC per pin a 3.3 V:** 50 mA.

Quindi è fondamentale tenere a mente il fatto che **esistono** dei **limiti massimi** e delle caratteristiche operative consigliate che differiscono ovviamente dal massimo perché si trovano ampiamente all'interno del range massimo segnalato.

C'è scritto, inoltre, nelle specifiche di Arduino, che se si utilizzano più di 12 V, il regolatore di tensione potrebbe riscaldarsi fino al punto da danneggiare la scheda. Questo, nei fatti, vuol dire che è necessario lavorare sulla temperatura, considerando che siccome essa aumenta più o meno linearmente così come specificato nella seguente equazione

$$Tj=P \times \theta ja + Ta$$

allora c'è da valutare e pesare tutti i componenti di quest'equazione, ovvero i coefficienti della stessa. Qui il parametro che si vuole calcolare è la temperatura di giunzione (Tj) che aumenta in maniera proporzionale alla potenza in gioco (P) che viene moltiplicata per un fattore che è, in realtà, il coefficiente di resistenza termica tra la giunzione e l'ambiente circostante (θja) al quale valore viene sommato la temperatura attuale di lavoro (Ta).

È necessario, pertanto, fare una valutazione operativa completa sull'incidenza della temperatura sulla scheda.

In linea generale, noi sappiamo che **uno dei grossi problemi**, probabilmente il più grosso, **della tecnologia allo stato solido è il fatto che riscalda** e i dispositivi vanno raffreddati in maniera opportuna, pena degrado delle prestazioni, nella migliore delle ipotesi. Ed è proprio di questo che parleremo adesso.

La cosa migliore, per tutti sempre, è quella di utilizzare **il proprio pollice**. L'idea è: se lo puoi toccare, non è troppo caldo.

Potete quindi mettere il pollice sopra l'integrato di turno e cominciare a fornire tensioni crescenti, ovviamente sempre e soltanto entro i limiti!

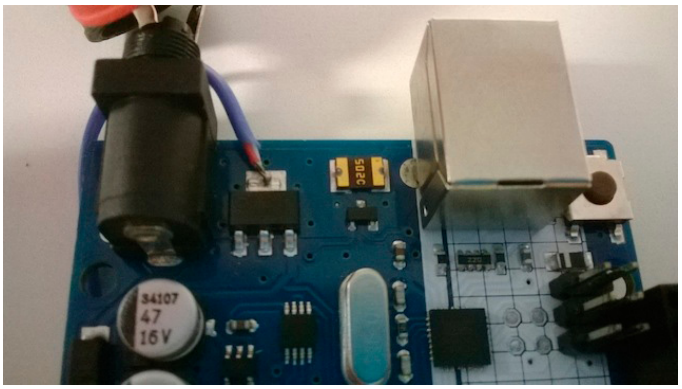
Facendo variare la tensione di esercizio per diversi intervalli temporali si nota il generale riscaldamento dell'integrato.

Dopodiché, è anche importante caratterizzare quanto tempo l'integrato resta in funzione anche per capire quanto "regge" all'aumento di temperatura nelle prestazioni e quindi correlare la capacità di operare a temperature più elevate con la sua resa effettiva.

Si nota chiaramente che esiste un limite, in particolare ci possiamo aspettare che questo sia intorno ai 65°, oltre il quale le prestazioni degradano ma soprattutto si va incontro al rischio di danneggiamento dell'integrato prima della scheda.

Ovviamente questa è una prova spannometri-

ca, per avere una risposta precisa bisogna applicare la formula utilizzando le specifiche del componente fornite dal datasheet e, sempre consigliato, verificare anche nella realtà con un alimentatore variabile ed una sonda di temperatura applicata sul regolatore.



Noi la prova l'abbiamo fatta. Considerando 30minuti come tempo per stabilizzare la temperatura ed una temperatura ambiente di 22 C° (quindi ottimale) abbiamo ottenuto questi gradi in relazione alle varie tensioni applicate e correnti erogate.

Assorbimento Base di 50mA - Temperatura esterna, sul case (la temperatura di giunzione è decisamente più alta)

12V - 55 °C

14V - 64 °C

16V - 72 °C

Assorbimento Base di 100mA - Temperatura esterna, sul case (la temperatura di giunzione è decisamente più alta)

12V - 74°C

14V - 85°C

16V - XX°C *(prova non effettuata per non danneggiare il regolatore)*

Considerando che Arduino è una scheda pensata per connettersi alle shield e nella maggior parte dei casi alimentarle, direi che la sua alimentazione è insufficiente per tale compito, basti pensare che la sola shield ethernet assorbe circa 200mA. Valore difficilmente sopportabile dal regolatore U1.

Inoltre, se avete un progetto con molti led, come se ne vedono tanti in rete, dovete tener presente che oltre un certo numero led (dipende dal singolo assorbimento) potreste aver problemi di assorbimento sull'alimentazione e quindi mandare in reset o addirittura bruciare il regolatore e/o il microcontrollore. Andrà quindi fatto un calcolo preciso sulla corrente assorbita che deve essere rapportata sia alla corrente massima erogabile dal micro che alla dissipazione del regolatore U1.

ATTENZIONE: non prendete queste righe come un'incitazione a provare senza criterio!

Di fatto, tornando a noi, la temperatura operativa ottimale può essere considerata valida intorno ai 50°. Questo vuol dire che, dal momento in cui è fissato l'importo di corrente e di tensione continua che si sta fornendo scheda, bisogna considerare che la corrente massima è pari a:

Imax = P / V = 500 mW / (Vin - 5V)

Una potenza di 500 mW vuol dire che, se abbiamo 5 V in ingresso, stiamo lavorando con 100 mA. Se la tensione in ingresso è pari a 12 V, ne risulta una corrente pari a 70 mA e se questo vi sembra strano, effettivamente avete ragione.

Questo valore non è accettabile.

Ovviamente questi limiti devono essere considerati in relazione a queste cifre. Dopodiché, caso per caso, progetto del progetto, bisogna confrontare i valori che si hanno con quelli teorici massimi supportati dalla scheda ed è proprio di questo che dobbiamo imparare a tener conto.

Per risolvere il problema del fatto che l'integrato si riscalda, basta pensare a raffreddarlo. Questa affermazione lapalissiana non è poi così evidente dal momento che la progettazione di un dissipatore, lo abbiamo visto poco tempo

fa sul proprio su queste pagine è piuttosto delicata, complessa e deve tener conto di diversi fattori per garantire la più efficiente delle attività di dissipazione del calore. Questa soluzione dovrà tenere poi conto anche del tipo di regolatore di tensione con quale abbiamo a che fare, non ultimo per via del fatto che non tutti regolatori forniscono lo stesso livello di tensione.

Vale la pena precisare che **12 V è un valore piuttosto comodo per pilotare una serie di dispositivi circuiti integrati** disponibili commercialmente e che possono avere diverse funzioni. Pertanto, potremmo non voler rinunciare all'utilizzo di questo valore di tensione. **Come si fa a renderlo pienamente compatibile allora con Arduino se c'è questa che sembra rendere le due grandezze incompatibili?**

Se non si desidera utilizzare il dissipatore, è comunque possibile pensare ad una scheda esterna. Il nuovo regolatore che lavori con questo valore di tensione potrebbe essere, di fatto, uno **stadio di alimentazione esterno ad Arduino**, perfettamente configurato e che si limiti a fornire i 5 V di cui la scheda bisogna ed i 12, invece, al resto del circuito da alimentare.

In questo modo la scheda non verrebbe sovraccaricata.

Sarebbe stato quindi anche opportuno che il regolatore di tensione utilizzato (NCP1117) fosse stato in un contenitore differente, rispetto all'attuale SOT223 montato sulla scheda Arduino. Dal link precedente potete trovare il relativo datasheet e quindi verificare che esistono package con una dissipazione migliore, come il DPAK sempre a montaggio superficiale.

Inoltre un regolatore compatibile, e ce ne sono molti, con un basso dropout (LDO) avrebbe contribuito ad un ulteriore abbassamento della dissipazione.

Ma abbiamo l'NCP1117 e quindi una soluzione pratica potrebbe essere di alimentare la scheda a 9V oppure interfacciare l'alimentazione con un ulteriore regolatore che fornisca 9V appunto, o meglio ancora 8V.

LE USCITE ANALOGICHE

Quello che andiamo ad analizzare della sezione dei pin analogici riguarda aspetti leggermente più complicati rispetto a quelli che concernono (o intervengono) in semplici progetti. **Questo ed il prossimo paragrafo, avrete modo di accorgervene, parlano di aspetti dei quali è molto difficile che si accorgano gli utenti alle prime armi, quelli molto inesperti.** Questo perché lo scopo dell'articolo è cercare di rendere Arduino più professionale analizzando soprattutto problemi dei quali si accorgano utenti che cominciano, come si suol dire, a "contare i peli" alla scheda e a spingerla oltre i suoi limiti. Programmatori certamente più smaliziati o elettronici di lungo corso, soprattutto esperti di elettronica analogica. D'altronde, però, sapere prima è molto meglio che rimanere delusi poi, non trovate?

Ci sono **i pin 3, 5, 6, 9, 10 e 11** della sezione digitale che sono **marchiati con una** ~, a differenza degli altri che non la riportano. Questo perchè che **quelli**, e soltanto quelli, "marchiati" **possono produrre in uscita segnali PWM** (Pulse Width Modulation). Anche se, in realtà, il controllo di questi pin può essere effettuato mediante la funzione

```
analogwrite(pin, value)
```

il cui nome suggerirebbe che ci sia un qualche modo "analogico" per controllarli, **questi segnali NON hanno a che fare con un convertitori**

DAC. Ecco, questi, in realtà, forniscono uscite digitali e, quindi, quantizzate, non davvero analogiche.

Quando questo diventa importante?

Beh, è semplice: il convertitore è a 10 bit quindi questo vuol dire che i valori possibili vanno da 0 a 1023. E allora, se il numero di ampiezze "utili" per noi e per la nostra applicazione, è estremamente contenuto, non ci accorgeremo mai di questa limitazione. Se invece i valori di cui abbiamo necessità dovessero aumentare e diventarre confrontabili con la risoluzione, sentiremmo il problema.

Non soltanto "il numero di numeri" necessari è importante ma anche la precisione con la quale generiamo il numero. Per esempio, **anche soltanto nel lampeggio di un LED** (anche riferendoci al valore massimo di corrente di cui abbiamo parlato in precedenza) **possiamo aver necessità di essere estremamente precisi**.

La ragione per la quale necessario stare attenti è molto semplice: **i pin con cui è possibile lavorare in PWM implementano in hardware un counter-timer che automaticamente genera forme d'onda senza necessità di interventi da parte del programmatore** (tramite codice).

A differenza di quello che succede quando il PWM viene comandato via codice, il timer hardware produce esattamente impulsi che hanno quella temporizzazione con periodo stabile. Ora, il parametro *value* imposta il *duty-cycle* per, poi, moltiplicarlo per un numero complesso fra 0 e 255 (il che vuol dire che siamo parlando di 8 bit) che è un intero senza segno.

Quando parliamo del *duty-cycle* c'è da intendersi anche sulla notazione perché è possibile anche che siate abituati a riferirvi a questa cifra in percentuale normalizzata (per intenderci con un minimo di 0.0 ed un massimo di 1.0). Ad ogni modo, il concetto resta perfettamente identico.

Supponiamo di volere un duty-cycle pari al 30%:

0,3 x 255 = 76,5

Ci si pone, di fatto, il problema di sempre: **qual è il valore più vicino?** In questo caso poi siamo nella condizione di scegliere **tra 76 e 77** quasi in maniera indifferente. Ci rendiamo conto che scegliere 76 implicherebbe, in realtà, un valore di 0.298 mentre scegliere 77 vorrebbe dire 0.30196. E allora? Questi valori vanno bene o vanno male?

Qui ci colleghiamo alla premessa iniziale: la risposta è che **dipende dalla precisione**, o meglio dall'imprecisione, che siete disposti a tollerare. Quanto potete essere di "bocca buona" dipende dal grado di precisione che il vostro specifico progetto richiede.

Siccome c'è un'incertezza sui valori misurati pari allo 0,4% del valore di fondo scala, c'è anche da valutare se tale incertezza sia effettivamente compatibile con il tipo di progetto che state portando avanti.

Per garantire la corretta alimentazione e la scrittura del valore desiderato, è possibile seguire un primo approccio,in cui imponiamo semplicemente valori alto o basso, chiaramente inseriti in un loop con delle temporizzazioni precise:

```
pinMode(10, OUTPUT);
analogWrite(11,LOW);
analogWrite(11,HIGH);
```

oppure, un secondo approccio in cui andiamo a specificare esattamente il valore che vogliamo inserire:

```
pinMode(10, OUTPUT);
analogWrite(11,130);
```

In questa seconda modalità PWM ci accorgeremo che **i LED rispondono comunque bene**.

Questo perché la loro luminosità, o meglio **l'intensità luminosa, dipende criticamente proprio dal valore che andiamo a scrivere sul pin**. Tuttavia resta un fatto: l'occhio umano risponde, alla stimolazione luminosa, in maniera pressoché logaritmica e pertanto l'intensità luminosa del LED, in realtà, va misurata in ottave.
Tutto questo va riferito anche al fatto che, **siccome il segnale (digitale) proviene da un PWM e potremmo utilizzare un filtro RC molto semplice per estrarre il valore medio dal treno di impulsi**, allora **nasce la presenza di un polo**.
La frequenza caratteristica è pari a
f=1/(2 π RC)
il che implica anche la figura del roll-off è fondamentale. Non è questo, però, l'aspetto fondamentale bensì il fatto che la **frequenza può essere modulata, modificata** (come effetto delle modifiche fatte in hardware) **fino ad ottenere dei risultati migliori**. Su che cosa? È semplice: in generale, aumentare la frequenza risolve la maggior parte dei problemi che stiamo sperimentando. Soprattutto non richiede ulteriore hardware aggiuntivo.
Il firmware, invece, se aumentare la frequenza è quello che vogliamo, non risulta di alcuna utilità, a meno di non pilotare direttamente l'uscita come abbiamo visto precedentemente, ma anche in questo caso avremo i limiti imposti dalla frequenza di lavoro del microcontrollore.
Non c'è modo quindi di farlo adeguatamente tramite software.
Sostanzialmente la cosa è trasparente all'utente.
Quindi, se abbiamo l'esigenza di farlo in software, l'unica è interpellare direttamente i Timer dei microcontrollori. In particolare stiamo parlando di *Timer0, Timer1* e *Timer2*. Questi timer infatti sono la base dei tempi per i rispettivi PWM.
Tuttavia il filtraggio analogico non riduce l'errore, o meglio l'imprecisione, dello 0,4% su passi successivi di cui abbiamo parlato in precedenza.
Sia l'ATmega168 sia l'ATmega328 hanno counter-timer e hardware PWM che includono la maggior parte delle opzioni e delle features interessanti che l'interfaccia di Arduino mostra. Pertanto, vale la pena di leggere la documentazione ufficiale della Atmel in maniera tale da comprendere meglio come funziona e perché.
In realtà, sulla nuova versione della scheda, Arduino DUE, che è basata sul microcontrollore a 32 bit SAM3X8E, ci sono **convertitori** che hanno **12 bit che producono anche la bellezza di 12 output PWM** ulteriori e che, differentemente da quanto succede per le uscite filtrate digitalmente, possono produrre più alte risoluzioni per l'estensione analogica con bande decisamente più accettabili. Questo si traduce nel fatto che si può lavorare molto meglio con l'audio rispetto a quello che si può fare con la UNO.
Ancora una volta, quindi, l'alimentazione è fondamentale. Questo perché i valori DC forniti sono proporzionali alle tensioni di alimentazione ai convertitori.
Vale la pena di sottolineare, a questo proposito, che la maggior parte dei progetti, specie quelle amatoriali, si basa certamente sull'alimentazione che proviene dall'USB dei computer. Questa non è necessariamente una soluzione ottimale perché spesso siamo solo vicini ai 5 V e comunque non è possibile effettuare un controllo estremamente fine della corrente che viene erogata tramite queste connessioni.

I PIN DIGITALI E ANALOGICI

E' necessario connettere, in maniera corretta,

input e output del microcontrollore in maniera tale che il progetto possa funzionare al meglio. A differenza di quanto succede nel mondo ideale, **il mondo reale impone dei requisiti sia sulla circuiteria esterna sia su Arduino che** troppo spesso **vengono ignorati**. Ed, esattamente come per le altre considerazioni fatte, è possibile verificare effetti indesiderati sia sui dati sia sulla scheda che rischia, tra le altre cose, potenzialmente, di essere danneggiata.

A questo punto, vale la pena di analizzare i pin in maniera tale da valorizzare anche il progetto. **I 6 pin analogici d'ingresso di Arduino**, ovvero quelli numerati **da A0 ad A5, possono** anche **servire da uscite**, dal momento che anche questi sono programmabili e potrebbero essere impiegati come pin di I/O general purpose.

Molti dei problemi che si verificano con i progetti su Arduino vengono fuori da eventuali interazioni tra il circuito esterno e l'hardware provvisto per microcontrollori sulla scheda.

Per esempio, anche se generalmente pensiamo che i pin digitali abbiano sempre la possibilità di lavorare fra una tensione minima pari a 0 V ed una massima pari a VCC = 5V, in realtà non è esattamente così. **Soprattutto non si riesce ad andare con precisione rail-to-rail.**

Facendo test e misure si riesce a scoprire che, in particolare, **il limite minimo sembra essere 0,5 V** mentre per quanto riguarda il **limite superiore** parliamo di un massimo di **5,5 V**.

L'intervallo totale effettivo rilevabile risulta essere compreso tra **-0.5 e 5.5 V**. Questo può causare dei problemi quando abbiamo il livello **0 V come riferimento**. In particolare è possibile che ci siano feedback di correnti negative che possono anche creare dei danni non indifferenti alla scheda.

Se utilizziamo una sezione di alimentazione per un'eventuale espansione del nostro progetto, sarà necessario utilizzare dei diodi, magari degli Zener, per limitare questi fenomeni al massimo.

In realtà uno degli aspetti più interessanti è rappresentato anche dal ruolo di ciascun pin. Come abbiamo detto, scrivendo un firmware opportuno, la funzione di tutti gli I/O può essere facilmente personalizzata. Tuttavia, **quando interviene un** evento di **RESET**, salvo specifiche condizioni, **ciascuno di essi torna ad avere la funzione che aveva originariamente**. Ergo è assolutamente indispensabile essere sicuri di quale funzione ciascun pin stava svolgendo prima di adoperare il circuito.

Le condizioni di RESET, in realtà, possibili sono **tre**: la prima occorre quando **viene meno l'alimentazione** del circuito, ovviamente. La seconda è quando si effettui un **reset via software** mentre la terza corrisponde alla **pressione del pulsante** predisposto sulla scheda.

Come tutti circuiti digitali, esistono **valori di tensione notevoli** relativi alla specifica famiglia logica. In particolare stiamo parlando di **VIL, VIH, VOL e VOH**. Sono valori caratteristici che corrispondono ai minimi e massimi delle tensioni in ingresso ed in uscita corrispondenti ad una commutazione. Talvolta può capitare che ci sia corrente all'interno anche se non c'è niente connesso alla scheda.

Considerati i limiti delle famiglie logiche, le loro caratteristiche e quindi le specifiche tensioni già nominate, le correnti di leakage in input potrebbero esserci anche per pin impostati per valori logici "LOW". A maggior ragione questo vale se invece il pin è "alto". **Un valore notevole, ed anche frequente, per questa condizione potrebbe essere 1 uA**; ancora una volta, rispetto allo specifico sistema, questo potrebbe essere troppo grande oppure assolutamente ininfluen-

te.

Una sorgente di questo genere di interferenze potrebbero essere i **condensatori**. Anche piccole cariche accumulate all'interno dei condensatori elettrolitici possono risultare da un campo elettrostatico esterno, dall'interruzione brusca di corrente oppure ancora da un forte segnale a radiofrequenze, da interferenze, ancorché a bassa frequenza (linee di segnale a 50 Hz, per esempio).

La documentazione, ma l'esperienza soprattutto, suggeriscono di utilizzare dei semplici fotodiodi oppure delle capacità commutate per misurare valori di corrente effettivamente presenti.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-ai-raggi-x-cosa-fare-renderlo-professionale-prima-parte

Arduino ai raggi X: rendiamolo professionale – Seconda Parte

di Piero Boccadoro

LA QUESTIONE DELL'LMV358

Un altro problema di Arduino è la tensione di riferimento sull'LMV358 (U5), il comparatore che si occupa di controllare l'alimentazione generale (da connettore o da usb). Infatti sul pin 3 abbiamo in ingresso la tensione Vin/2 tramite un partitore di tensione. Ipotizzando una tensione di alimentazione uguale a 13V, sottratta della caduta di tensione sul diodo di ingresso D1, abbiamo 12.3V che applicati al partitore portano ben 6.15V in ingresso al comparatore. **Questa tensione non è accettabile perché supera l'absolute maximum rating dichiarato dal datasheet che è di 5.5V**. La soluzione è NON superare mai quindi l'alimentazione di 12V indipendentemente da quello che leggete sul sito di… arduino!

TEMPERATURA

Una nota sulla temperatura. Di seguito i componenti sensibili e la loro temperatura operativa indicata dal datasheet:

LMV358IDGKR -> -40°C + 125°C
LP2985-33DBVR -> -40°C + 125°C
NCP1117ST50T3G -> 0°C + 125°C
ATMEGA16U-MU -> -40°C + 85°C
ATMEGA328P-PU -> -40°C + 85°C

Si può notare che rientrano tutti nel range industriale tranne il regolatore NCP1117. Fermo restando che nel funzionamento difficilmente può creare problemi a basse temperature, cè da chiedersi perché non è stato utilizzato anche questo integrato con range di temperatura industriale?

USB

Sull'alimentazione proveniente dalla seriale USB (USBVCC) è stato previsto, giustamente un fusibile autoripristinante da 500mA ma non è stata inserita una induttanza di filtro, la classica BLM21 che si vede in quasi tutte le applicazioni USB. Tale induttanzina invece è stata inserita tra la massa e lo shield del connettore USB. Trovo tale soluzione discutibile, sia positivamente che negativamente, quindi entriamo nel campo delle scuole di pensiero contro i disturbi. Però sulla linea USBVCC un filtraggio sarebbe stato opportuno.

LA STABILITÀ MECCANICA

C'è poi un problema, tutt'altro che trascurabile ma sul quale apriamo soltanto una breve parentesi, riguardo la struttura dell'espandibilità prevista dalla scheda. **Il sistema degli shield è semplicemente geniale**. L'espandibilità è modulare, garantita per un numero di espansioni assolutamente notevole sia per numero di espansioni possibili contemporaneamente sia per varietà.

Arduino dimostra di essere molto versatile per realizzare praticamente qualunque tipo di sistema.

Dopodiché c'è un problema: se volessimo utilizzare questo sistema, espanso con uno shield opportunamente progettato, sarebbe compatibile con l'impiego in ambienti ostili o che preve-

dano particolari condizioni operative come forti vibrazioni? Che impatto avrebbero queste sollecitazioni sulla struttura?

La scheda è resistente a sollecitazioni per esempio nel campo automotive o avionico? E come reagirebbe lo shield? Qual è il rischio che si stacchi, per esempio? Quale quello che si danneggi per inopportuna o sottodimensionata resistenza a shock meccanici?

E come si affronta/risolve la questione della certificazione?

È necessario fare una seria riflessione su questo. Mancano, in realtà, i casi pratici di studio in cui si trova a testare questo aspetto ma per poter garantire al progettista che anche una semplice applicazione possa davvero beneficiare dell'utilizzo di Arduino sarebbe necessario un ulteriore, forse notevole, sviluppo in questa direzione.

Attualmente l'unica soluzione possibile è di scegliere shield che abbiamo i fori di ancoraggio in asse con Arduino per poter inserire adeguate viti e bulloni.

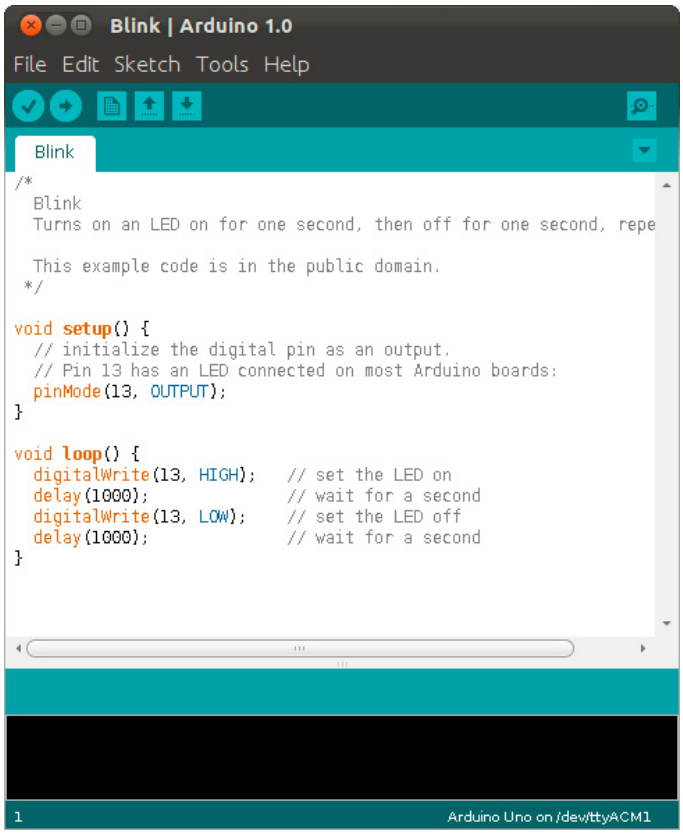
IL SOFTWARE

Riguardo questo aspetto ci sono diverse voci a sostegno della tesi per la quale **il software** che permette la programmazione di Arduino **è da rivedere**.

Si tratta indubbiamente di un programma molto snello, efficace ed intuitivo. Il problema vero, però, è che questa incisività, **questa semplicità, non ripaga il programmatore più smaliziato dell'averlo voluto scegliere perché non gli permette di lavorare scendendo più a basso livello**.

È davvero molto comodo poter imparare i rudimenti della programmazione con questa GUI, che riesce a proporre una sintesi delle principali funzioni che il C ed il C++ richiedono all'utilizzatore.

Tuttavia, **sussistono una serie di problemi che rendono questa IDE particolarmente insoddisfacente.**



La prima questione che resta aperta e la possibilità di effettuare debug riga per riga.

Il codice non necessariamente funziona al primo caricamento ed anche il programmatore esperto sperimenta dei problemi. E sarebbe molto interessante che ci fosse tutto un sistema di warning o avvisi che segnalino evidenti di conflitti tra assegnazioni di variabili, oppure allocazioni di vettori che cambiano dimensione nel corso dell'esecuzione. A dire il vero, quest'ultima feature è presente ma purtroppo non va di pari passo con la possibilità di risolvere il problema grazie all'allocazione dinamica.

I programmatori più esperti conosceranno certamente la keyword "*new*" e con essa la funzione *malloc*.

Mi è capitato di dover ridefinire due variabili globali all'interno dell'esecuzione del *loop()*. Pur-

troppo gli errori segnalati non erano chiari ed alla fine ho semplicemente intuito quale fosse il problema.

Non tutte le funzioni di libreria possono essere facilmente incluse e molte rischiano di dover essere scritte apposta. Questo da un lato è certamente un ottimo esercizio di programmazione ma dall'altro un'enorme perdita di tempo ed una complicazione notevole. Questo a maggior ragione per via del fatto che non esistendo una seria suite di debug integrata, eventuali errori proprio nella scrittura di questa libreria, **sarebbero pressoché impossibili da trovare**.

Queste problematiche portano semplicemente il programmatore a scegliere di scrivere con un'altra interfaccia e, di conseguenza, cambiare scheda sulla quale lavorare.

L'unica soluzione, a questo scopo, sarebbe di utilizzare un'interfaccia JTAG e scrivere direttamente in Assembly.

La domanda, pertanto diventa: **ma davvero chi è in grado di fare una cosa del genere sceglie Arduino?**

Se la risposta a questa domanda sembrerebbe essere no, è comunque vero che a questo punto Arduino mostra tutti i suoi limiti e diventa non più una scheda per "tutti" ma per "chi comincia o poco più".

Tornando un attimo al discorso delle variabili globali, non è sempre vero che la riallocazione non funziona; anzi, proprio per verificare che la cosa funzionasse, ho creato un programma base che vi mostro qui di seguito:

```
int i=0;

void setup() {
Serial.begin(9600);
i=1;
Serial.println(i);
}
void loop() {
if (i<10) {
Serial.println(i);
}
else if(i>10 && i<20) {
Serial.print(i);
Serial.println("Non ancora");
}
else if (i>20) {
Serial.println(i);
}
i=i+1;
}
```

ecco, in questo semplice sketch l'override funziona!

Ecco, quindi, che la domanda diventa: quand'è che questa operazione smette di funzionare? E perché?

Io personalmente a queste le domande non ho trovato risposta. Il sospetto è che non si possano trovare con grande facilità proprio per il fatto che non esiste una funzione di debug puntuale che sottolinei i vari errori del compilatore, eventuali problemi nell'accesso dei registri o similari. Sarà davvero difficile riuscire a trovare una "quadra" per dare una risposta a questo dubbio.

Tutto sommato, però, bisogna anche ammettere che questo genere di problematiche vien fuori solamente quando il progetto è di una certa complessità, il codice diventa piuttosto impegnativo e viene da pensare che a quel punto l'utilizzo di una scheda come Arduino sia da sostituire all'impiego di qualcosa di più performante e professionale.

Un vero peccato è che praticamente la stessa IDE venga utilizzata anche per Arduino DUE, una scheda della quale abbiamo già parlato

ampiamente e che abbiamo visto, nel dettaglio, essere una soluzione straordinariamente più interessante e molto più marcatamente votata all'impiego in applicazioni ed ambiti certamente non da novizi.

Ricordo inoltre che la programmazione di Arduino, tramite la IDE ufficiale, avviene in C/C++, quindi NON in assembler e nemmeno in C ma è in C/C++ che molti sketch vengono scritti. Da una parte questo porta una semplificazione non indifferente del programma ma dall'altra anche una NON conoscenza e quindi NON controllo di quanto avviene realmente sul microcontrollore ATMega328. A tal proposito invito a leggere questo articolo. Un esempio classico di problema che emerge con questo tipo di approccio è quello della gestione di più eventi. Se ad esempio dovete gestire una comunicazione seriale RS232 in full-duplex e scrivere i dati su un display LCD non potrete farlo senza che il display visualizzi un fastidioso sfarfallio. L'evidenza di tale difetto è proporzionale all'attività della porta seriale ma con una seriale simulata via software (la periferica UART del micro è impegnata per la comunicazione con la IDE) di meglio non si può fare. A meno di non avere un approccio alla comunicazione in bit-banging ma in questo modo ci perdiamo tutta quella semplicità di cui sopra. Insomma, un semplice progetto (RS232 + LCD) già potrebbe avere problemi, figuriamoci progetti più complessi, se non gestiti correttamente, che risultati possono avere.

DELLE PROPOSTE

Concludiamo questo articolo con un breve accenno ad alcune ulteriori accortezze che si potrebbero avere per migliorare la qualità complessiva totale di Arduino e renderlo un prodotto largamente appetibile ed interessante per chi ne debba fare un uso professionale o anche semplicemente più variegato.

1. innanzitutto si potrebbe rendere la corrente massima erogabile dall'output a 3.3 V più alta dei soli 50 mA;
2. si potrebbero impiegare delle protezioni dagli overshoot di corrente per ciascun pin della scheda;
3. si possono implementare delle misure di limitazione nella dissipazione di potenza per proteggere i microcontrollori, per esempio implementando un integrato switch per la distribuzione dell'alimentazione, ce ne sono in contenitori piccolissimi, questo limiterebbe la corrente in maniera tale da ridurre automaticamente il voltaggio per limitare la corrente al valore di 150 o 200 mA massimo;

Queste sono soltanto alcune delle migliorie che si potrebbero certamente e semplicemente, soprattutto, adottare sulla scheda e la renderebbero un po' più completa.

TIRIAMO LE SOMME

Questa versione della scheda in realtà, rappresenta un punto di evoluzione dell'intera serie perché, specie la terza revisione, presenta un grounding molto migliorato rispetto alle precedenti il che, come abbiamo visto, rappresenta un ottimo punto di partenza. I problemi sperimentati possono essere molti meno, specie per segnali analogici di bassa entità anche se è sempre

necessario porre particolare attenzione alle connessioni a massa.

La panoramica odierna è servita a dare un'idea di quali sono i problemi che Arduino ha ma soprattutto per quale motivo essa non è del tutto compatibile con l'uso professionale.

Per quanto concerne l'alimentazione, come abbiamo visto, la scheda è assolutamente autosufficiente e funzionale. L'alimentazione deve essere fornita da Arduino sempre e soltanto rispettando quei limiti ma rimanere in valori più contenuti può essere molto utile ed addirittura santifico per la scheda.

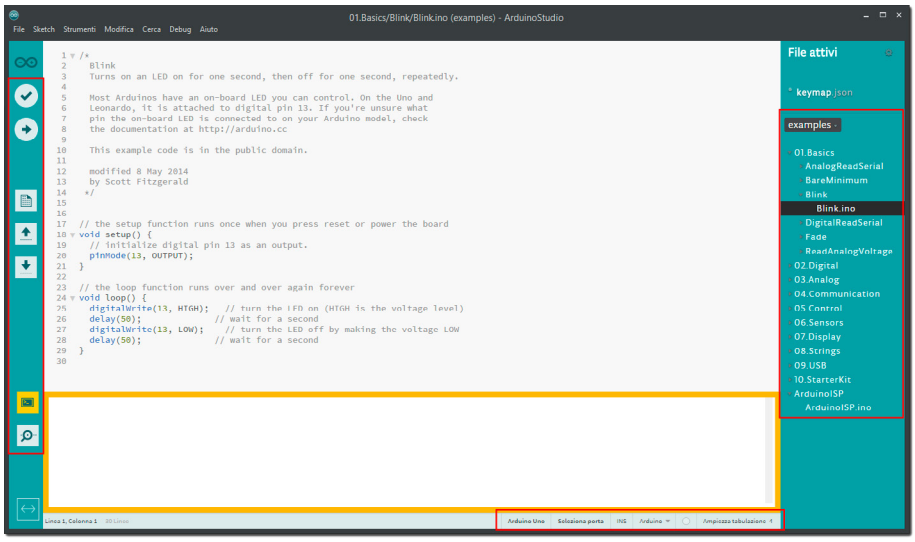
Image Credits | www.goeffect.com

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-ai-raggi-x-rendiamolo-professionale-seconda-parte

Arduino Studio: getting started

di Ernesto Sorrentino

Arduino Studio è un **nuovo ambiente di sviluppo open source** per il linguaggio di programmazione di Arduino. Invece di una architettura monolitica e un modello di sviluppo centralizzato, Arduino Studio sfrutta **Adobe Brackets; un editor scritto in JavaScript, HTML e CSS** che si concentra sulla progettazione di applicazioni web. Brackets è dotato di caratteristiche uniche come **modifica rapida, anteprima live** su browser e altre particolarità introvabili in altri ambienti di sviluppo. Attualmente la versione rilascia di Arduino Studio è Alpha ma il Team sta lavorando duramente per consentire agli utenti di sfruttare questo tool come stand-alone, web/cloud-based ed editor tutto integrato. **Un solo programma per tutti gli ambienti**, distribuito sul sito Arduino.org, in versione per Linux, Mac OS X o Windows.

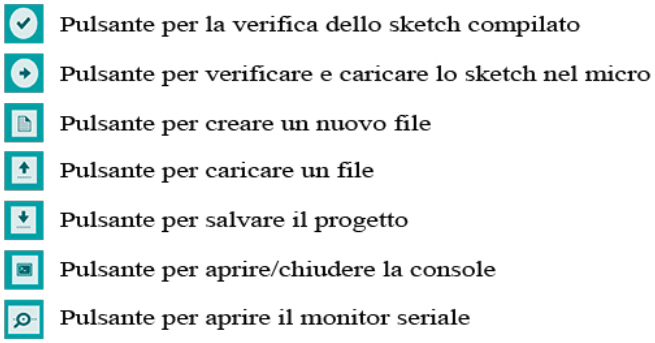


INSTALLAZIONE E PANORAMICA

Scaricato la versione idonea al proprio sistema operativo, scompattare l'archivio nella root di "C:" o in una directory a vostra scelta; **il tool non necessita l'installazione**, si avvia dall'eseguibile "**ArduinoStudio.exe**".

La nuova interfaccia grafica si presenta pulita ed elegante con l'aggiunta dei pulsati, sulla colonna di sinistra, per i comandi dedicati alle funzioni di Arduino; sulla colonna di destra è presente l'albero di esplorazione percorso mentre in basso ci sono i comandi per la configurazione della board e la console per i dettagli.

I comandi sono identici al vecchio editor, con precisione:



In aggiunta, nella parte inferiore della finestra, troviamo il menù rapido per i comandi più utilizzati come (in ordina da sinistra) la scelta della board, la porta di comunicazione, la modalità del cursore, il linguaggio di programmazione e la scelta selettiva tra spazi/tabulazione.

PROGRAMMARE L'ARDUINO UNO

Essendo una **versione ancora alfa l'unica bo-**

ard programmabile è l'Arduino UNO. Collegata al PC attendere il riconoscimento del micro; se non sono presenti i driver è possibile installarli dal menù:

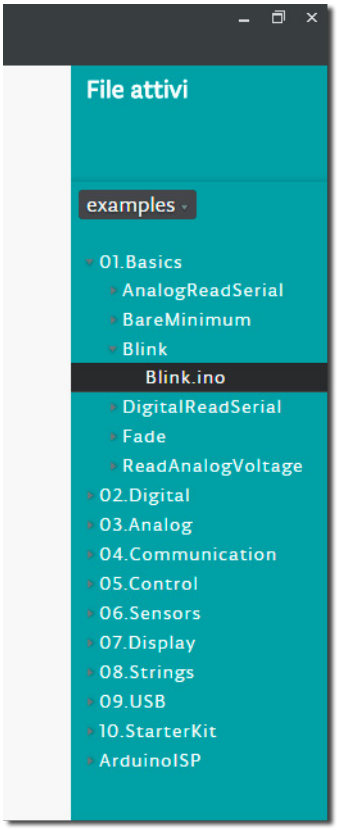
```
Aiuto\Installa driver\Arduino
```

Come esempio di prova utilizzeremo il famoso sketch "Blink", dal menù:

```
File\Apri esempi
```

Sulla destra della finestra si aprirà l'esploso della cartella esempi, dove sarà possibile caricare il file dal percorso:

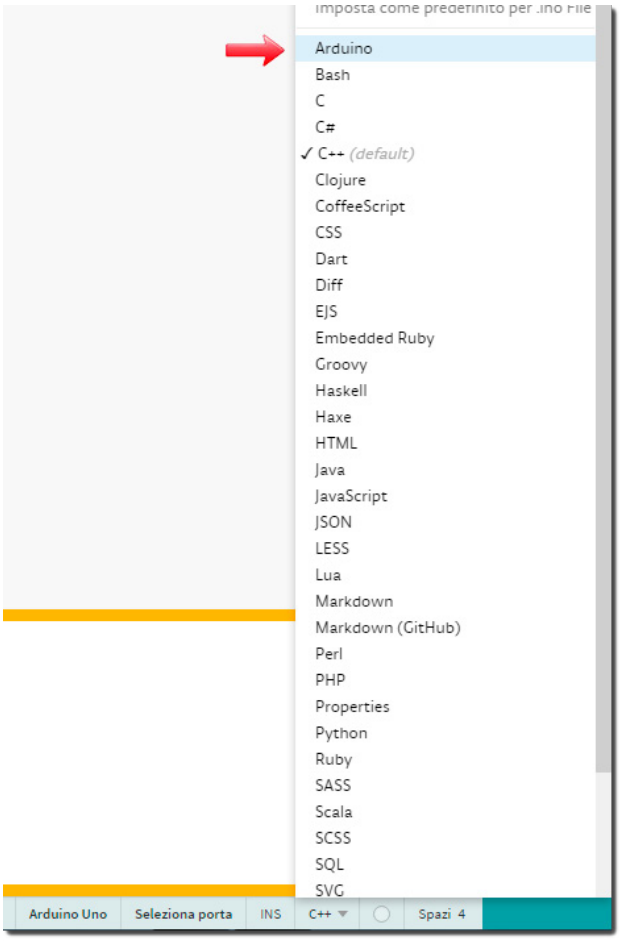
```
01.Basic\Blink\Blink.ino
```



Non resta che settare i parametri della board e dell'editor dal menù rapido in fondo alla finestra. Selezionare il tipo di Arduino cliccando sul primo pulsante del menù;

la porta seriale di comunicazione cliccando sul secondo pulsante;

e il tipo di linguaggio in uso cliccando sul quarto pulsante. L'impostazione base l'editor è settato su **C++** e come avrete notato, **le istruzioni Wiring non sono colorate**. Questo non pregiudica il funzionamento dello sketch ma risulta disabilitato l'auto-compilazione; per ottimizzare l'utilizzo del programma è consigliato selezionare **"Arduino"** come linguaggio.



Infine non resta che compilare lo sketch e programmare la board cliccando sul secondo pulsante del menù a sinistra della finestra o dal percorso

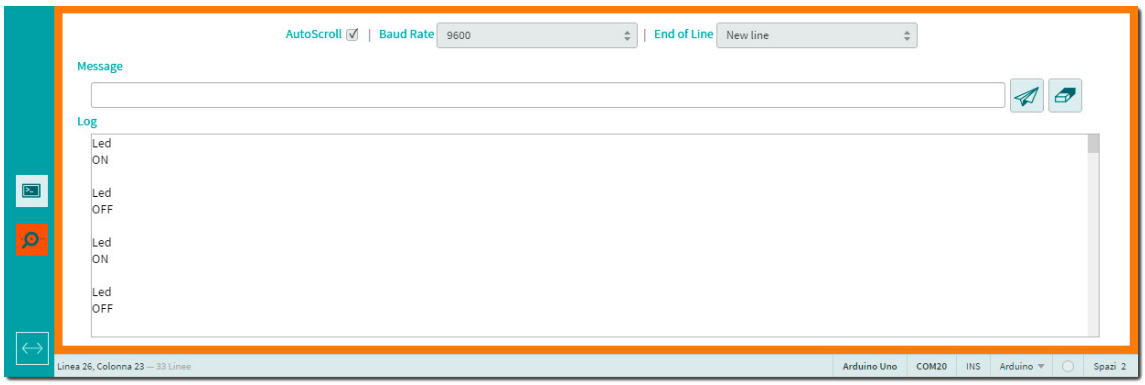
```
File\Carica
```

Durante la compilazione si può notare che i **dettagli nella console sono più ordinati** rispetto alla vecchia IDE, agevolando notevolmente la ricerca della cartella **"temp" per prelevare i file "core.a" e ".elf"** come visto in un articolo precedente. Se tutto è andato a buon fine dovreste vedere il led lampeggiare sulla scheda.

Un'ultima funzione da provare è il **"monitor seriale"**. A parte la grafica, e un piccolo bug nella gestione del fine linea, l'unica differenza riscontrata è **l'aggiunta di un pulsante per pulire il log**. Per effettuare una prova basta inserire il comando **"Serial"** di start in **"Setup()"** e quello di **"print"** in **"loop()"**, come di seguito:

```
void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  Serial.println("Led ON");
  digitalWrite(13, HIGH);
  delay(20);
  Serial.println("Led OFF");
  digitalWrite(13, LOW);
  delay(250);
}
```

Riprogrammare la board e aprire il **monitor seriale cliccando sul pulsante con l'icona della lente** sulla sinistra della finestra. Il risultato sarà il seguente:



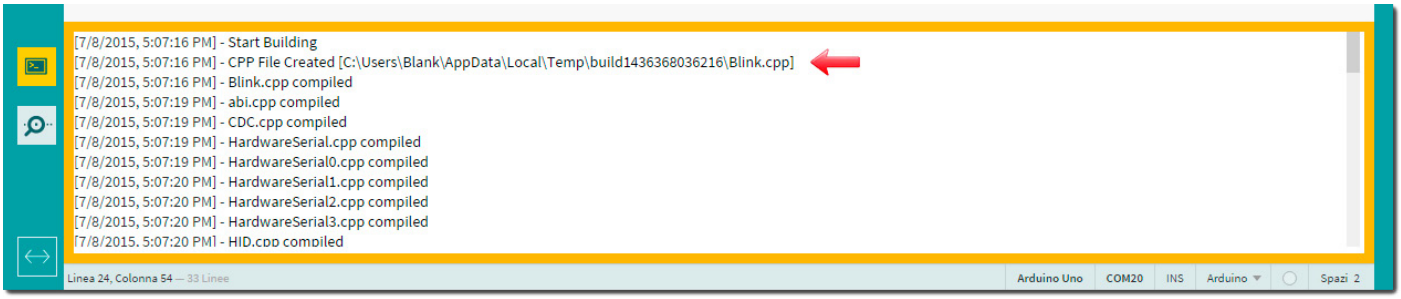
ALTRI LINGUAGGI DI PROGRAMMAZIONE

L'esempio che vedremo di seguito crea una piccola pagina statica in HTML che riprende parte del seguente articolo. Questo esempio non è utilizzato per insegnare a programmare in HTML ma per scoprire i comandi disponibili in Arduino Studio. Iniziamo col creare due file nuovi da "**File/Nuovo File**" e nominandoli rispettivamente:

1. index.html
2. main.css

In "**index**" settare come linguaggio di programmazione "**HTML**" (comando in fondo alla finestra) e inserire il seguente codice di esempio:

Invece in **"main"** settare come linguaggio **"CSS"** e inserire il seguente codice nell'editor:




```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Arduino Studio: getting started</title>
    <meta name="description" content="Una guida interattiva per muovere i primi passi su Brackets.">
    <link rel="stylesheet" href="main.css">
  </head>
  <body>

    <a href="http://it.emcelettronica.com/">
    <img src="http://it.emcelettronica.com/files/img/logo14.png" alt="Elettronica Open Source">
    </a>

    <h2>Arduino Studio: getting started</h2>

    <p>
      Arduino Studio è un nuovo ambiente di sviluppo open source per il linguaggio di programmazione
di Arduino.
      Invece di una architettura monolitica e un modello di sviluppo centralizzato, Arduino Studio sfrutta
      <b>Adobe Brackets; un editor  scritto in JavaScript, HTML e CSS</b>
      che si concentra sulla progettazione di applicazioni web. Brackets è dotato di caratteristiche uniche
come
      modifica rapida, anteprima live su browser e altre particolarità introvabili in altri ambienti di svilup-
po.
    </p>

  </body>
</html>
```

```
html {
   background-color: #e6e9e9;
   background-image: -webkit-linear-gradient(270deg,rgb(230,233,233) 0%,rgb(216,221,221) 100%);
   background-image: linear-gradient(270deg,rgb(230,233,233) 0%,rgb(216,221,221) 100%);
   -webkit-font-smoothing: antialiased;
}

body {
   margin: 0 auto;
   padding: 1em 2em 4em;
   max-width: 800px;

   font-size: 16px;
   line-height: 1.5em;
   color: #545454;
   background-color: #ffffff;
   box-shadow: 0 0 2px rgba(0, 0, 0, 0.06);
}

h2 {
   color: #0000ff;
   margin-top: 1.3em;
}

b, strong {
   font-weight: 600;
}

img {
   -webkit-animation: colorize 2s cubic-bezier(0, 0, .78, .36) 1;
   animation: colorize 2s cubic-bezier(0, 0, .78, .36) 1;
   background: black;
   border: 5px solid rgba(0, 0, 0, 0.12);
   border-radius: 4px;
   display: block;
   margin: auto;
   max-width: 100%;
}
```

Brackets dispone di molte funzioni per sviluppare un programma ma dato che non potrò soffermarmi su tutte, esporrò solo quelle più utilizzate. Una **particolarità di questo editor è la modifica rapida** senza spostarsi da un documento all'altro, infatti quando si lavora al codice HTML, è possibile digitare "**Cmd/Ctrl + E**" per aprire un'area di modifica rapida che mostra il CSS corrispondente. Una volta effettuata una modifica al codice CSS basta premere "**ESC**" **per ritornare sull'HTML** chiudendo l'area di modifica, oppure è possibile lasciarla aperta continuando a lavorare sull'HTML. Proviamo con un esempio; posizionare il cursore su **<h2>** di "**<h2>Arduino Studio: getting started</h2>**" e premere "**Ctrl + E**", apparirà la finestra di modi-

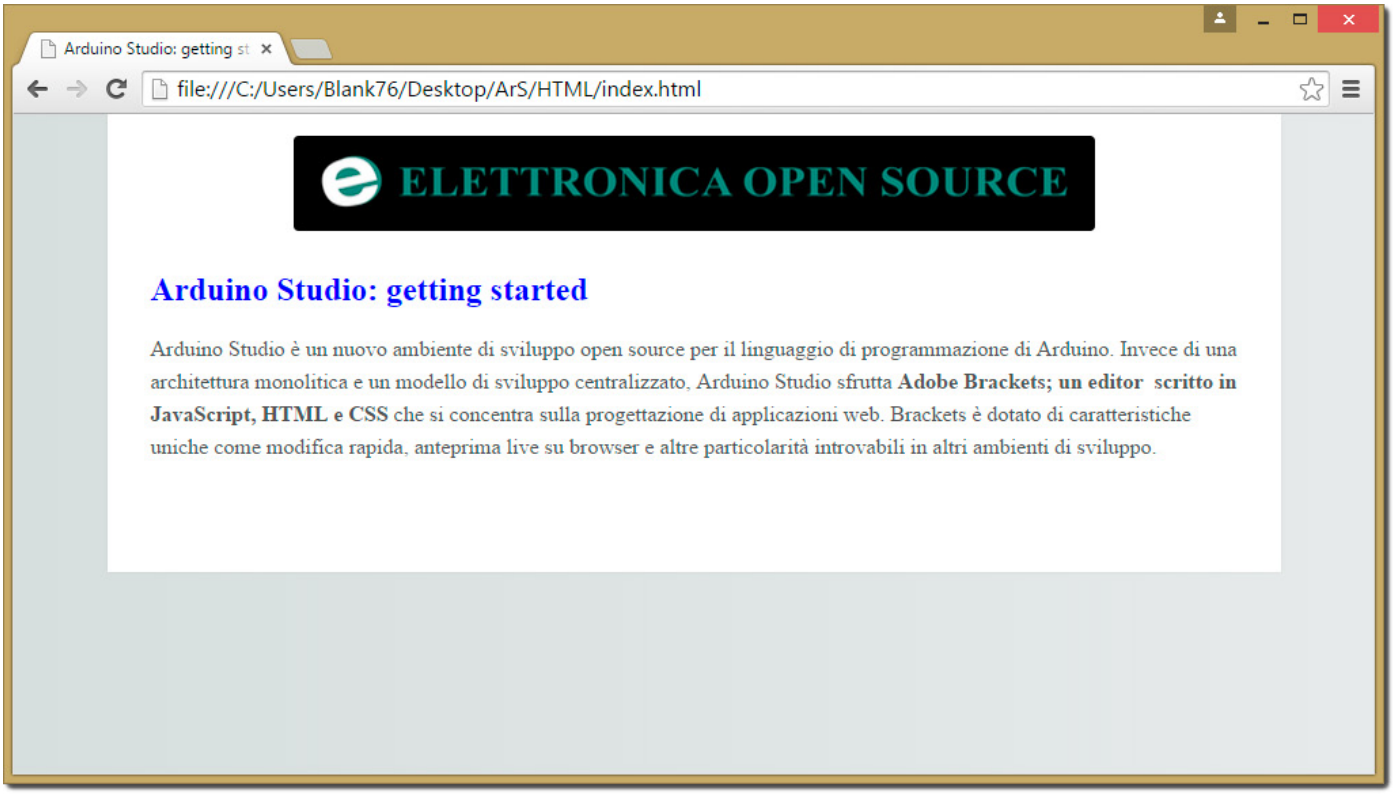
fica rapida come questa:

Sulla destra si vedrà la **lista delle regole CSS corrispondenti al tag**. Per scorrere le regole basta premere "**Alt +Up/Down**" per trovare quella che si intende modificare. Un altro aiuto viene fornito per la modifica del colore, posizionando il cursore sul codice apparirà una piccola finestra con l'anteprima del colore settato

La più importante, invece, funzione di Brackets è "**L'anteprima LIVE**", ossia vedere tutte le modifiche apportate al codice **in tempo reale** sul browser (per adesso funziona solo con Chrome) **senza dover aggiornare continuamente la pagina**. Brackets aprirà una connessione col browser e gli manderà tutte le modifiche del CSS non appena digitate! Esistono già dei strumenti browser-based che fanno qualcosa di simile ma con Brackets non sarà più necessario copiare e incollare il codice CSS definitivo nell'editor. **Il codice gira nel browser ma risiede già anche nell'editor**! Per provare aprire l'anteprima live premendo i tasti "**Cmd/Ctrl + Alt + P**" e modificare il codice CSS come descritto sopra; noterete che le modifiche avvengono in tempo reale. Brackets rende semplice vedere come le proprie modifiche all'HTML e al CSS hanno effetto sulla pagina. Quando il cursore è su una regola CSS, nel browser verranno messi in risalto tutti gli elementi che vengono influenzati da quella regola. Similmente, quando si modifica un file HTML, Brackets metterà in risalto nel browser gli elementi HTML corrispondenti. Oggi, **Brackets supporta solo l'Anteprima Live per il CSS**. Comunque, nella versione corrente, i cambiamenti ai file HTML e JavaScript sono automaticamente ricaricati quando si salva. Al momento il team di sviluppo sta lavorando al supporto Live anche per HTML e JavaScript.

CONCLUSIONE

Personalmente ritengo che questo editor sia fantastico e, a parte la sua prematura versione, ho riscontrato solo punti di forza. Prima di tutto essendo multi ambiente rende possibile creare più progetti in contemporanea, come ad esempio realizzare una interfaccia web (in Html) per gestire un Arduino connesso alla rete (Wiring/Arduino). Inoltre, è non da poco, ho riscontrato una maggior velocità nel caricare i file e nel programmare i micro rispetto ad altri ambienti di sviluppo. Con questo nuovo editor spero di trovare, a versione definitiva, l'integrazione di un tool di debug per le board come la M0 Pro; dopo di ché lo riterrò promosso a pieni voti.



L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-studio-getting-started

Inclinometro con Arduino e accelerometro a 3 assi

di slovati

INTRODUZIONE

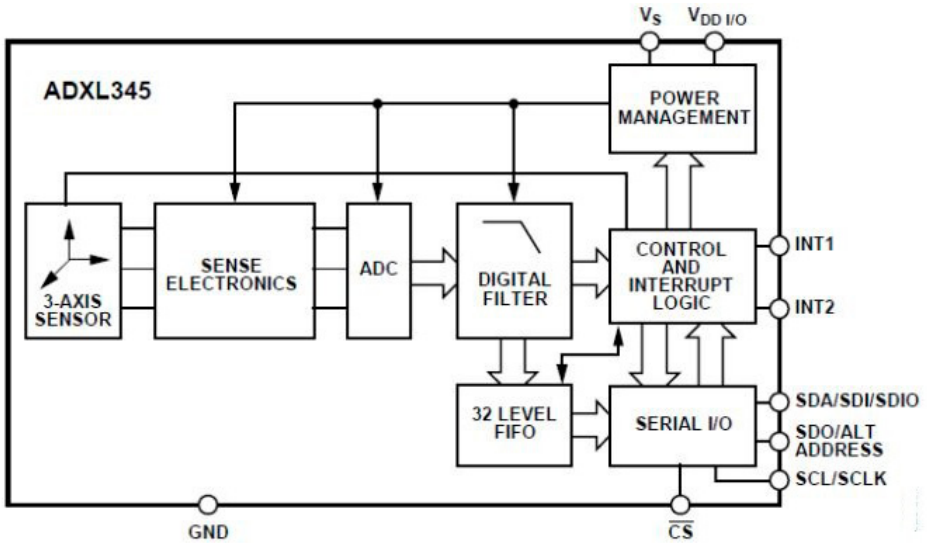
Dal punto di vista **hardware**, l'inclinometro è composto fondamentalmente da tre elementi:

1. **accelerometro**: si tratta del sensore principale dell'applicazione, un dispositivo in grado di fornire il valore di accelerazione sui tre assi X, Y, e Z. Per questa applicazione abbiamo utilizzato l'accelerometro **ADXL345** prodotto da Analog Devices, un accelerometro digitale molto preciso e sensibile, facilmente reperibile già installato su un modulino pronto per essere interfacciato con un microcontrollore. Il modulo accelerometro è completamente configurabile via software tramite una comoda interfaccia **I²C**;
2. **display LCD**: in questo caso la scelta è ricaduta su un display LCD monocromatico a basso assorbimento, con le stesse caratteristiche di quello originariamente montato sui cellulari **Nokia 5110/3110**. Trattasi anche in questo caso di un display facilmente reperibile, con dimensione dello schermo pari a 1,5 pollici, e risoluzione di 84 x 48 punti. Indipendentemente da quale sia il loro produttore, questi display presentano tutti la caratteristica di utilizzare il controllore **PCD8544** di Philips, che andrà opportunamente programmato durante la fase di inizializzazione dell'applicazione. Il display verrà utilizzato per monitorare in tempo reale i valori correnti di pitch e roll, ma potrebbe essere utilizzato per fornire altre informazioni, quali ad esempio una bitmap che rappresenta il valore corrente di inclinazione su uno dei due assi. Il modulo LCD, dotato anche di apposito circuito di retroilluminazione a led blu oppure bianchi (a seconda del modello), si interfaccerà con il microcontrollore tramite porta **SPI**;
3. **microcontrollore**: come già anticipato in precedenza, ci serviremo della consolidata piattaforma Arduino, sia per la sua larghissima diffusione che per la semplicità (relativa in questo caso) di programmazione. In questo caso particolare, un vantaggio enorme deriva dalla disponibilità di librerie già pronte all'uso per quanto riguarda sia la comunicazione su bus I²C che su SPI. Potremo quindi concentrarci sull'applicazione vera e propria, senza dover riscrivere codice per implementare le interfacce hardware di comunicazione.

Dal punto di vista **software**, l'applicazione sarà invece composta da un unico sketch in grado di svolgere tutte le funzioni richieste dall'applicazione, le quali possono essere così sintetizzate:

- **inizializzazione** dell'accelerometro **ADXL345** tramite opportuna configurazione dei registri interni del componente;
- **inizializzazione** del display **LCD**, tramite opportuna configurazione dei registri del controllore PCD8544;
- **ciclo continuo** con:
  - acquisizione dei valori grezzi di accelerazione sui tre assi;

- conversione dei valori grezzi di accelerazione in componenti di accelerazione di gravità (numeri espressi in “g”);
- calcolo degli angoli di inclinazione sugli assi X e Y (pitch e roll) tramite opportune trasformazioni trigonometriche;
- visualizzazione degli angoli di inclinazione sia su display LCD che su interfaccia seriale di debug.



- **calibrazione** (eseguita solo su richiesta dell’utente durante la fase di start-up): vengono in questo caso calcolati gli offset di accelerazione sui tre assi X, Y, e Z da inserire negli opportuni registri dell’accelerometro in modo tale da fornire, in condizioni di piano livellato, i valori di accelerazione attesi (Acc_x=0, Acc_y=0, Acc_z=1g).

Per ragioni di semplicità, si è preferito non implementare delle librerie specifiche per l’ADXL345 e per il display LCD. Lo sketch conterrà quindi tutte e sole le funzioni necessarie per implementare i requisiti dell’applicazione. Vedremo più in dettaglio queste funzioni successivamente, quando verrà esaminato il codice sorgente.

ADXL345

L’ADXL345 (il cui schema a blocchi è riportato nell’immagine seguente) è un accelerometro a 3 assi compatto e sottile, con assorbimento ultra-ridotto, elevata **risoluzione** (**fino a 13 bit**), e possibilità di misurare accelerazioni fino a **±16 g**.

I valori di accelerazione sono forniti in uscita su parole a 16-bit in complemento a 2 (due registri a 8-bit per ciascun asse). Il componente dispone di interfacciamento digitale con un microcontrollore esterno sia tramite I^2C (il tipo di interfaccia che utilizzeremo nella nostra applicazione), che SPI.

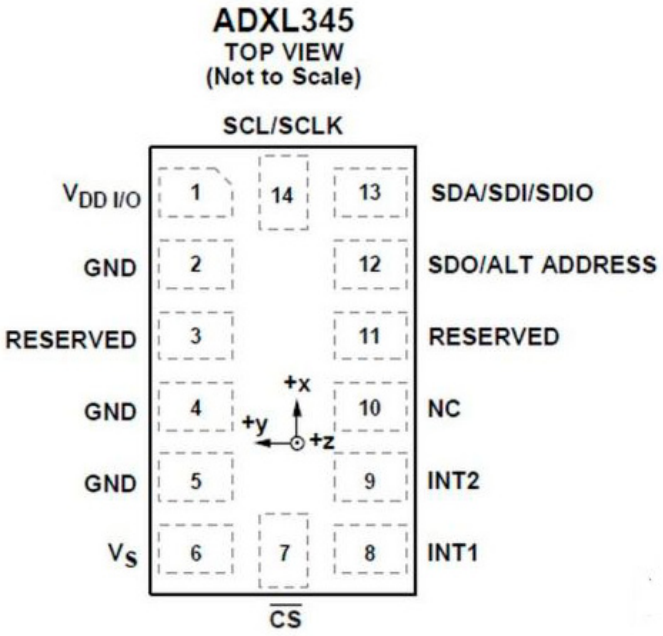
La caratteristica rimarchevole di questo componente è la capacità di misurare le **componenti di accelerazione sui 3 assi** non solo in modo **statico**, ma anche **dinamico**. La prima caratteristica, misurando le componenti dell’accelerazione di gravità sui tre assi, permette di realizzare la funzionalità di **tilt**, particolarmente utile sui dispositivi mobile (smartphone e tablet) per determinare l’orientamento dello schermo ed eseguire, se abilitata, la rotazione dello stesso. Non solo, le componenti statiche dell’accelerazione di gravità consentono, tramitre l’applicazione di opportune trasformazioni trigonometriche, di derivare gli angoli di inclinazione rispetto agli assi X e Y, noti anche con i termini (utilizzati in ambito aeronautico e nella navigazione) di **pitch** e **roll**. L’elevata risoluzione del componente consente, come vedremo a livello pratico, di misurare angoli anche inferiori a 1,0°.

Le componenti di accelerazione dinamica per-

mettono invece di determinare l'entità del moto al quale è soggetto l'accelerometro (si noti come il componente sia in grado di misurare accelerazioni di notevole entità, fino a 16g positivi o negativi). Questa caratteristica può essere pertanto utilizzata per rilevare le condizioni di caduta libera ("***free-fall***"), utile ad esempio per disattivare o mettere in sicurezza un dispositivo elettronico in caso di caduta accidentale, e di urto violento ("***shock***"), utilizzabile ad esempio per attivare un dispositivo di sicurezza quale l'airbag (attenzione: non è detto che questo componente sia utilizzato anche in ambito automotive, il principio di funzionamento è comunque lo stesso).

Un'altra funzionalità offerta dal modulo è la capacità di rilevare le condizioni di "**tap singolo**" e "**doppio tap**", tramite la configurazione di opportuni registri nei quali vengono inseriti i valori di soglia desiderati. Le condizioni di tap e *free-fall* possono poi essere agganciate a **due interrupt distinti** (INT1 e INT2) in modo tale da consentire una rapida e tempestiva gestione degli eventi stessi.

L'accelerometro dispone inoltre di un'area interna di memoria (sistema brevettato da Analog Devices) nella quale possono essere immagazzinati i valori misurati secondo una logica FIFO a 32 livelli. Questa funzionalità permette di alleggerire il carico di lavoro sul processore interfacciato con il modulo stesso, riducendo anche l'assorbimento di corrente. Nella seguente immagine è visibile il **pinout** dell'accelerometro:



Dal punto di vista prettamente elettronico, il sensore è realizzato tramite una struttura **MEMS** in silicio, appoggiata sopra un wafer anch'esso di silicio. La struttura poggia sul wafer tramite delle micro molle, che oppongono una certa resistenza alle forze determinate dalle accelerazioni. Le deflessioni della struttura vengono misurate tramite dei condensatori differenziali, composti da una lastra solidale con la struttura in movimento, e da una lastra in posizione fissa. Le deflessioni della microstruttura MEMS dovute alle accelerazioni provocano una variazione di capacità elettrica nei condensatori, la cui ampiezza è proporzionale all'accelerazione stessa.

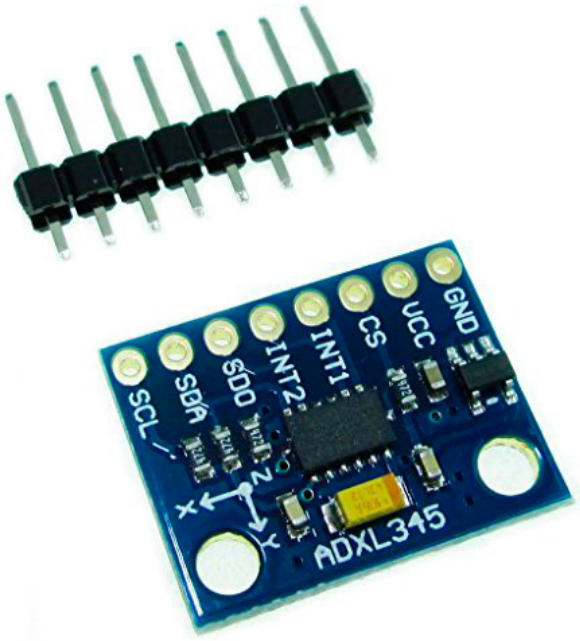
CARATTERISTICHE ELETTRICHE

Di seguito riportiamo le principali caratteristiche elettriche dell'accelerometro ADXL345:

- assorbimento ultra-ridotto: fino a 23 µA durante la misura, e 0,1 µA in modo standby;
- risoluzione selezionabile via software. La risoluzione massima è pari a 13-bit (range di misura: ±16 g), con un fattore di scala costante pari a 4 mg/LSB su tutte le scale;
- memoria interna di tipo FIFO per minimizzare il carico sul processore host;
- rilevamento tap singolo e doppio;
- rilevamento caduta libera (free-fall);
- alimentazione compresa tra 2,0 V e 3,6 V;
- interfacce di comunicazione digitali I^2C e SPI;
- due pin di interrupt configurabili;
- range di temperatura compreso tra −40°C

e +85°C;
- resistenza agli urti fino a 10000 g;
- dimensioni estremamente compatte: package LGA da 3 mm × 5 mm × 1 mm

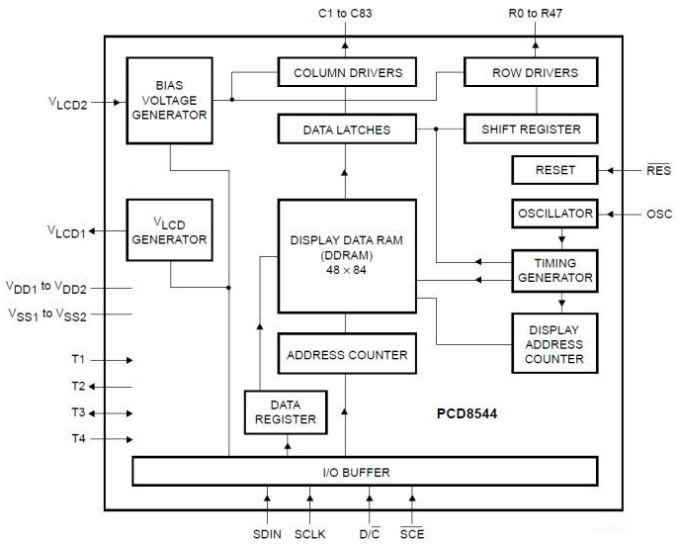
Per le applicazioni pratiche con Arduino o altri microcontrollori analoghi, l'accelerometro ADXL345 è disponibile già installato su un **apposito modulino**, facilmente interfacciabile con il processore host. Nel nostro caso abbiamo utilizzato il modulo noto anche con la sigla "**GY-291**" (vedi immagine seguente), facilmente reperibile presso numerosi rivenditori online a costi molto ridotti (qualche euro). Spesse volte il modulo viene venduto in kit, ed occorre eseguire manualmente la saldatura del connettore a 8 pin sul PCB del modulo stesso. L'operazione risulta comunque alquanto semplice, basta utilizzare una breadboard come supporto durante la saldatura. Vedremo più avanti come collegare elettricamente il modulo GY-291 ad Arduino.

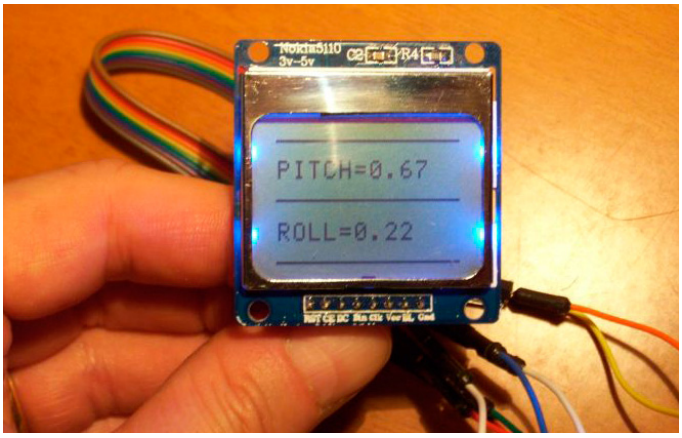


LCD 5110

Il display è realizzato in tecnologia LCD, con una struttura a matrice con risoluzione di **48 x 84 pixel**. La retroilluminazione è realizzata tramite LED blu oppure bianchi, mentre l'interfacciamento con il sistema host avviene tramite interfaccia SPI. Il controller/driver utilizzato da questi display è il ben noto **PCD8544** di Philips, realizzato in tecnologia **CMOS** e caratterizzato da un assorbimento molto ridotto. Tutte le funzioni necessarie per pilotare il display sono incluse nel chip stesso, compresa la generazione del livello di tensione per comandare l'accensione e il contrasto del display LCD. Il display, nato inizialmente per equipaggiare i cellulari Nokia 3310 e 5110, viene normalmente venduto già installato sul PCB, con backlight composta da quattro LED bianchi o blu, e con il relativo connettore a 8 pin saldato e pronto per l'utilizzo. Il numero di pixel disponibili consente di utilizzare il display sia per la visualizzazione di **testo** (con ciascun carattere mappato su un'opportuna matrice di punti), che di **simboli grafici**. Sul web sono anche disponibili dei tool in grado di convertire un'immagine bitmap in un array di byte da passare direttamente al display 5110 per la relativa visualizzazione.

Nelle due immagini seguenti sono riportati, rispettivamente, lo **schema a blocchi** relativo al controller PCD8544, e un'immagine del display in funzione, con visualizzazione dei valori correnti di pitch e roll.



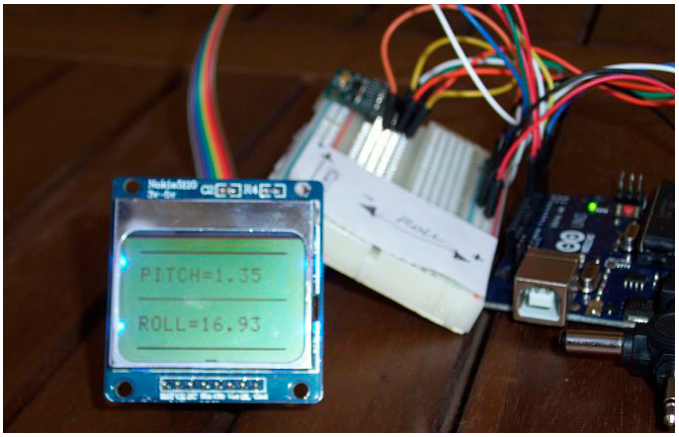


La **programmazione** del controller installato sul display è relativamente **semplice**, come vedremo più in dettaglio affrontando l'analisi del software. Informazioni dettagliate su questo componente, come del resto per l'accelerometro ADXL345, sono contenute nei rispettivi datasheet allegati all'articolo.

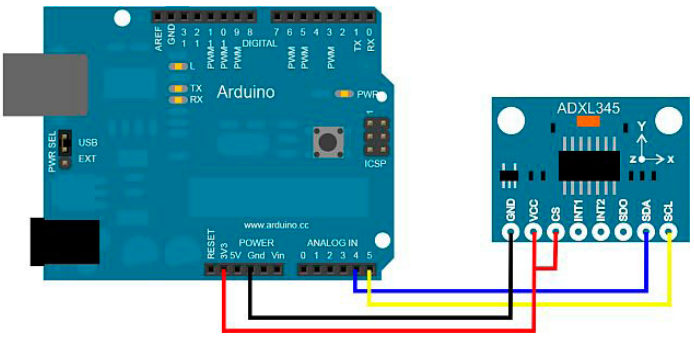
Il controller PCD8544 richiede un'**alimentazione** stabilizzata compresa tra **2,7 e 3,3 V**. Ciò vale non soltanto per i pin di alimentazione in senso stretto (Vcc, Ground, e Backlight), ma anche per tutti i segnali di ingresso al display utilizzati per la sua programmazione e controllo. In generale è pertanto necessario applicare delle opportune resistenze di limitazione sui segnali di controllo del display, oppure utilizzare un circuito che esegua una traslazione dei livelli di tensione da 5V a 3,3V. Per la nostra applicazione, tuttavia, è stato utilizzato un modulo LCD in grado di supportare sia l'alimentazione a 3,3V che quella a 5V (questa indicazione è anche riportata sullo stampato, come visibile nell'immagine precedente), per cui non è stato necessario adottare alcuna delle tecniche precedentemente menzionate.

COLLEGAMENTI CON ARDUINO

Vediamo ora in dettaglio i **collegamenti elettrici** da effettuare tra la board Arduino (per comodità faremo riferimento alla versione Uno Rev 3) e i moduli **accelerometro** ADXL345 e **display LCD** Nokia 5110. Per comodità tutto il circuito è stato assemblato su una breadboard, utilizzando ove necessario dei collegamenti con cavi DuPont. Nulla vieta, ovviamente, di approntare un circuito filato o meglio un PCB sul quale collocare in modo stabile e definitivo i componenti discreti:



I **collegamenti con il modulo ADXL345** sono riassunti dal seguente schema Fritzing:



In sostanza vengono utilizzati i seguenti pin:

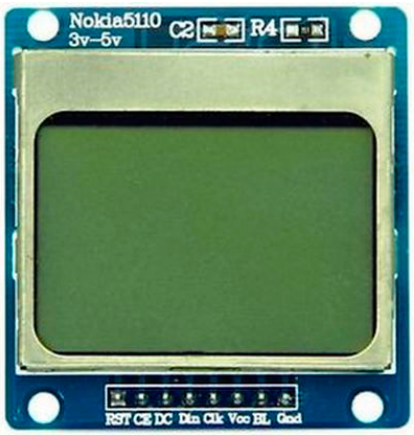
- **Vcc** e **GND**, collegati, rispettivamente, ai pin 3,3V e Gnd di Arduino Uno;
- **CS** (chip select), collegato al pin 3,3V di Arduino Uno;
- **SDA** e **SCL**, collegati, rispettivamente, ai pin Analog In 4 e Analog In 5 di Arduino Uno. Questo rappresenta il collegamento standard utilizzato dalla libreria "**Wire**" di Arduino, cioè quella in carico di gestire la comunicazione sul bus I2C (i segnali SDA e SCL rappresentano, rispettivamente, i

pin dati e clock utilizzati dal bus).

I pin INT1 e INT2 non vengono utilizzati e possono pertanto essere lasciati scollegati. Il pin SDO, anche se non utilizzato, potrebbe essere tirato a massa per sicurezza (su alcuni modi può infatti generare dei disturbi sulla misura). Con il tipo di modulo utilizzato, questa operazione non è stata comunque necessaria.

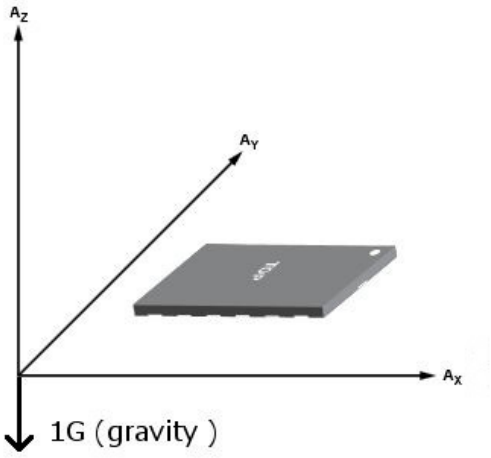
Con riferimento al **display** LCD Nokia 5110 utilizzato (vedi immagine seguente), i **collegamenti** da eseguire verso la board Arduino Uno sono i seguenti:

Pin display LCD	Pin Arduino
RST (reset)	Digital 6
CE (chip enable)	Digital 7
DC (data/command)	Digital 5
Din (data input)	Digital 4
Clk (clock)	Digital 3
Vcc	3,3 V oppure 5 V
BL (backlight)	3,3 V oppure 5 V
Gnd	Gnd



CALCOLO DI PITCH E ROLL

Come abbiamo visto in precedenza, l'accelerometro ADXL345 fornisce una misura statica dell'accelerazione di gravità lungo i tre assi x, y, e z. Si noti come la componente di accelerazione lungo l'asse z sia positivia se rivolta verso il basso, e negativa se rivolta verso l'alto.



Gli angoli di inclinazione del vettore accelerazione di gravità rispetto agli assi x e y corrispondono al **pitch** e al **roll**. L'associazione dipende ovviamente da come è disposto il sensore sulla board: a seconda dei casi il pitch potrebbe pertanto essere associato all'asse x oppure all'asse y (e viceversa il roll all'asse y oppure all'asse x). Assumendo di trovarsi nel primo caso, le formule che permettono di calcolare i due angoli sono le seguenti:

$$Pitch = arctan\left(\frac{x}{\sqrt{z^2+y^2}}\right)$$

$$Roll = arctan\left(\frac{y}{\sqrt{z^2+x^2}}\right)$$

Si tratta perciò di un calcolo che può essere agevolmente eseguito dal microcontrollore Arduino Uno, ricorrendo alla funzione matematica **atan2**. Poichè questa restituisce un angolo in radianti, sarà inoltre necessario convertire il valore finale in gradi.

Le grandezze x, y, e z che compaiono nelle due formule sono i valori di accelerazione forniti dall'accelerometro, convertiti dal software applicativo in "g": in pratica occorre moltiplicare il valore grezzo letto dal componente per l'LSB corrispondente alla scala utilizzata (l'LSB è espresso

in millesimi di g). Vedremo questo dettaglio analizzando il codice.

LO SKETCH

Lo sketch Arduino, disponibile in forma completa tra gli allegati, è strutturato in cinque parti principali:

1. dichiarazioni **costanti**: in questa sezione vengono definite tutte le costanti necessarie per configurare e utilizzare correttamente sia l'accelerometro ADXL345 che il controller del display. Per ogni ulteriore approfondimento è sufficiente consultare i relativi datasheet;
2. dichiarazione **variabili globali**: vengono definite tutte le variabili di appoggio per gestire i due componenti. In particolare, è definita la tabella dei caratteri ASCII utilizzata per visualizzare agevolmente stringhe di testo sul display LCD (i valori di pitch e roll, calcolati come double, vengono convertiti in sequenze di carattere prima della visualizzazione);
3. funzione **setup** dello sketch: esegue l'inizializzazione dell'accelerometro e del display;
4. funzione **loop** dello sketch: vengono ciclicamente acquisiti i valori grezzi di accelerazione, successivamente convertiti in g, e vengono calcolati gli angoli di pitch e roll. Questi valori vengono poi visualizzati sia sul display LCD che sull'interfaccia seriale di debug di Arduino (monitor seriale). Tra un ciclo di acquisizione e il successivo viene applicato un ritardo, in millisecondi, definito attraverso la costante DELAY_MSEC (attualmente è stato programmato un ritardo di 1 secondo);
5. **funzioni specifiche** per la gestione dell'accelerometro e del display.

Per quanto riguarda **l'inizializzazione dell'ADXL345**, occorre spendere un paio di parole. Anzitutto è stato introdotto un **controllo iniziale** (una novità rispetto ad applicazioni analoghe che possono essere reperite sul web) per verificare la bontà del collegamento sul bus I2C con il componente. Il datasheet, infatti, afferma che all'indirizzo 0x00 è presente il registro di sola lettura contenente **l'identificativo** prescelto dai progettisti dell'ADXL345, vale a dire l'identificativo **E5h**. Nella funzione setup dello sketch viene pertanto compiuta una lettura a questo indirizzo, ed eseguito il controllo di identificativo (se questo risulta essere errato, viene inviata un'opportuna segnalazione di errore sul monitor seriale). Questa funzionalità è utile anche in fase di setup del progetto, al fine di verificare il corretto cablaggio tra le parti. Inoltre, il registro DATA_FORMAT (indirizzo 0x31) non viene programmato dall'applicazione, utilizzando quindi il valore impostato come default al reset. Ciò equivale a selezionare un **range** di misura pari a **±2g**, più che sufficiente per i nostri scopi. Sempre dal datasheet sappiamo che con questo range il valore di accelerazione per ogni asse è rappresentato su 10 bit. Ciò significa che una misura pari a 511 (letta sui due byte associati a ogni asse) corrisponderà a +2g, mentre una misura pari a -512 corrisponderà a -2g. **L'LSB** corrispondente sarà perciò pari a (2 / 511) g, definito nel codice tramite la costante ADXL345_LSB.

LA CALIBRAZIONE

Una funzionalità particolare, e per certi versi innovativa, presente in questo progetto è rappresentata dalla **calibrazione automatica** degli offset di accelerazione. L'accelerometro ADXL345

dispone infatti internamente di tre registri offset a 8-bit (OFSX, OFSY, e OFSZ), con **LSB fisso** (indipendente dalla scala di misura selezionata) pari a 15,6 mg. L'accelerometro, dopo aver eseguito ogni misura, somma al valore misurato (in complemento a 2) il contenuto del corrispondente registro di offset. Questa somma è quella che poi viene effettivamente copiata nei due byte acquisiti dal programma.

La procedura di calibrazione prevede che l'accelerometro sia collocato in posizione **neutra** (ad esempio appoggiato a un tavolo perfettamente in piano): in tal caso le letture teoriche dovrebbero restituire Ax = Ay = 0, e Az = 1g (coincidente con l'accelerazione di gravità). In realtà ciò non è sempre possibile, vuoi per tolleranze dovute al cablaggio, vuoi per la sensibilità estrema del componente (l'LSB nel nostro caso corrsiponde a circa 3,9 millesimi di g!). Viene successivamente eseguita una misura delle accelerazioni sui tre assi, e calcolati i valori da scrivere nei tre registri offset affinchè sia soddisfatta la condizione precedente (Ax = Ay = 0, e Az = 1g). Non solo, i valori vengono anche copiati in memoria **Eeprom** (protetti da un'apposita "firma"), in modo tale da essere riutilizzati ad ogni accensione del dispositivo.

Per entrare nella procedura di calibrazione occorre accendere la board Arduino con il **pin digitale 2** collegato a **ground** (nelle condizioni normali esso rimarrà non collegato). Tale pin risulta infatti configurato a livello software come pin di ingresso con resistenza di pull-up abilitata (non è quindi necessario collegare alcuna resistenza esterna, utilizzeremo quella messa a disposizione da Arduino).

CONCLUSIONI

Abbiamo visto con questo progetto come sia possibile realizzare con pochi componenti modulari un inclinometro ad elevata risoluzione e sensibilità, semplice, compatto, e con assorbimenti ridotti (tutto il sistema può essere alimentato tramite il collegamento USB al PC). Abbiamo già elencato diverse applicazioni, ma sappiamo che altre ancora possono scaturire dalla fantasia o dalle specifiche necessità dei lettori.

Il progetto si presta inoltre a potenziali espansioni: si potrebbe ad esempio stabilire un valore di soglia per gli angoli di pitch o roll, superato il quale il sistema deve attivare una condizione di allarme (ad esempio triggerare un'uscita collegata a un relè). Si potrebbero anche memorizzare in Eeprom i valori minimo e massimo dei due angoli, in modo tale da poterli esaminare successivamente. Insomma, si tratta di un progetto già di per sè completo, ma che si presta a innumerevoli espansioni o migliorie.

ALLEGATI

ADXL345

PCD8544

Sketch Arduino

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/inclinometro-con-arduino-e-accelerometro-a-3-assi

Emulare l'Apple II con Arduino UNO

di slovati

INTRODUZIONE

A **Steve Wozniak**, co-fondatore insieme a **Steve Jobs** della Apple Computer Inc., vanno senza alcun dubbio i principali meriti nella progettazione del computer Apple II. Secondo Wozniak stesso (soprannominato familiarmente "Woz"), il computer ideale doveva essere "piccolo, affidabile, semplice da usare, ed economico". Sono stati proprio questi i fattori chiave che hanno guidato la progettazione e lo sviluppo di questo microcomputer, considerato da molti un vero e proprio capolavoro di efficienza e riduzione dei costi.

Il suo predecessore, l'Apple I, era stato progettato sempre da Wozniak due anni prima, nel 1975. Si trattava più che altro di un prototipo, venduto inizialmente tramite "passaparola" in una ristretta cerchia di amici e conoscenti, e successivamente distribuito presso qualche rivenditore selezionato nello stato della California. L'Apple II aveva invece tutte le carte in regola per candidarsi a una vera e propria distribuzione di massa, con volumi di produzione e vendita mai raggiunti prima di allora. Oltre ad una attenta, e per certi versi geniale, fase di progettazione, Apple cominciò anche a introdurre la de-localizzazione della produzione, sempre nell'ottica globale di riduzione dei costi. L'Apple II utilizzava una manciata di componenti, ed era probabilmente il primo microcomputer ad integrare su una stessa board tutte le unità funzionali: CPU, memoria, alimentazione, e interfacce di input/output. Le prime schede furono prodotte in California e in Texas, ma successivamente, con il crescere dei volumi di vendita, la produzione fu spostata a Singapore e in Irlanda.

Il cuore dell'Apple II era rappresentato dal microprocessore **6502**, prodotto da **MOS Technology**, con frequenza di funzionamento pari a 1,023 MHz. Il microcomputer era tutto racchiuso all'interno di un case molto compatto realizzato in plastica (progettato da Steve Jobs stesso), come visibile nell'immagine seguente. Il sistema utilizzava un interprete BASIC residente su memoria ROM, era dotato di un'uscita grafica in grado di operare sia in modalità testo che grafica (anche a colori), memoria RAM espandibile fino a 48 Kb, ed era equipaggiato con una tastiera ASCII. Come memoria di massa utilizzava un normale registratore a cassette, e non mancavano accessori per il gaming, rappresentati dai tipici "game paddle" in voga a quei tempi.

Le **specifiche tecniche** del microcomputer possono essere così riassunte:

- **CPU**: microprocessore MOS 6502 (1 MHz);
- **Display Video**: mappato in memoria, con diverse modalità operative, tutte selezionabili via software:

  - modalità testo: 24 linee da 40 caratteri ciascuna, con caratteri solo maiuscoli;
  - modalità grafica a colori di base: 40 pixel in orizzontale per 48 pixel in verticale, 15 colori;
  - modalità grafica ad alta risoluzione: 280 pixel in orizzontale per 192 pixel in verticale, colori nero, bianco, violetto, e verde (richiede almeno 12 Kb di memoria RAM);
  - è possibile selezionare entrambe le modalità grafiche, includendo 4 linee di testo nella parte bassa del display.
- Memoria **RAM**: 4 Kb, espandibile fino a 48 Kb. L'hardware è stato progettato sin dall'inizio per supportare sia i moduli di memoria a 4 Kb, che i moduli di memoria dinamica a 16 Kb;
- Memoria **ROM**: 8 Kb, espandibili fino a 12 Kb;
- Interprete **BASIC** esteso, residente in ROM, con comandi specifici per la gestione della grafica;
- **Monitor** esteso residente in memoria ROM;
- Interfaccia di I/O per **registratore a cassette**, con data rate pari a 1500 bps;
- Scheda madre con **8 slot** di espansione. Sicuramente un'intuizione geniale da parte di Wozniak, in quanto permetteva di utilizzare schede di espansione o periferiche realizzate anche da terze parti;
- Connettore di I/O per **game controller** Apple;
- Porta per **tastiera ASCII**;
- **Altoparlante**;
- **Uscita video composito**, in grado di pilotare direttamente un monitor oppure un comune televisore (tramite l'aggiunta di modulatore RF opzionale).

IL MOS 6502

Qualche nota merita il processore MOS **6502**, la CPU scelta da Apple come cuore dell'Apple II. Si trattava di un processore a 8 bit (con bus indirizzi a 16 bit) progettato nel 1975 dall'azienda MOS Technology, una startup fondata da alcuni ingegneri fuorisciti da Motorola, gli stessi che contribuirono alla realizzazione del processore **6800**. Prodotto inizialmente su licenza da Rockwell e da Synertek, il 6502 era all'epoca il **processore a 8-bit più economico disponibile sul mercato**. Il suo prezzo era infatti pari a circa 25 US$, almeno sei volte inferiore rispetto al prezzo praticato dalla concorrenza per processori analoghi (quali ad esempio lo Z80). I volumi di produzione conseguenti alla sua affermazione sul mercato indussero altri produttori, tra i quali Zilog, a ridurre drasticamente i prezzi di vendita dei propri componenti, evitando così di perdere significative quote di mercato. **MOS Technology** fu in seguito acquisita da **Commodore International**, che ne continuò la produzione, estendendo le licenze anche ad altri produttori. Il successo e la longevità del processore 6502 sono proseguiti sino ai giorni nostri, grazie alla versione realizzata in tecnologia **CMOS**.

Il prezzo del MOS 6502 era così basso, rispetto a prodotti simili della concorrenza, che inizialmente molti pensarono si trattasse di un prodotto scadente, una sorta di truffa ("scam" in inglese). Sappiamo che le cose andarono diversamente, e Apple vide giusto quando scelse questo componente. Si trattava di un processore con un numero ridotto di istruzioni (56 in totale), e un numero ristretto di registri (registro accumulatore a 8-bit A, due registri indice a 8-bit X e Y, program counter, stack pointer, e registro di stato). La semplicità del suo set di istruzioni, unita alla molteplicità delle modalità di indirizzamento disponibili, permisero di semplificare significativamente lo sviluppo del software, decretando il successo di questo processore.

L'EMULATORE APPLE II

Veniamo ora al progetto vero e proprio, oggetto dell'articolo, vale a dire l'**emulatore hardware di Apple II** realizzato con una normalissima scheda **Arduino Uno Rev 3**. Su Arduino non abbiamo bisogno di aggiungere nulla: penso che ogni lettore sappia di cosa stiamo parlando, e comunque su EOS è disponibile una vasta selezione di articoli che trattano questa piattaforma di sviluppo open source.

L'idea e la realizzazione pratica di questo emulatore sono opera di Damian Peckett, cui va il merito di aver portato a termine con successo un progetto tanto ambizioso quanto molto utile sul piano propedeutico, in quanto implementa alcune tecniche difficilmente riscontrabili nei numerosi progetti per Arduino reperibili sul web.

Il **primo passo** da compiere per realizzare un **emulatore** Apple II consiste nella realizzazione di un emulatore per il processore **6502**. Stando ai commenti dell'autore, questa fase è stata relativamente semplice ed immediata. Lo sviluppo è stato condotto utilizzando il linguaggio C standard, ponendo un'attenzione particolare all'efficienza, soprattutto nell'utilizzo della memoria e della CPU. Il set di istruzioni del processore 6502 è, come detto precedentemente, relativamente semplice: ogni codice operativo ha lunghezza di 8 bit, esistono 56 differenti codici operativi, ed esistono 13 differenti modalità di indirizzamento. La maggiorparte dei codici operativi sono utilizzabili con la maggiorparte delle modalità di indirizzamento, per cui l'emulazione risulta notevolmente semplificata, producendo un codice poco complesso e facilmente leggibile. Questo aspetto, caratteristico del processore 6502, viene detto "**quasi ortogonalità**". Con il termine "**ortogonalità**" si intende infatti la proprietà di un set di istruzioni in base alla quale tutte le istruzioni del set possono utilizzare tutte le modalità di indirizzamento, e quindi istruzioni e modi di indirizzamento variano in modo indipendente l'uno dall'altra.

La decodifica delle istruzioni è stata implementata tramite l'utilizzo dell'istruzione C *switch*, in quanto ciò consente al compilatore di ottimizzare il più possibile il codice, generando una tabella di jump molto efficiente. Si sarebbe potuta seguire anche la strada di utilizzare un array di puntatori a funzioni; questa scelta è stata tuttavia abbandonata in quanto comportava un overhead di chiamate e un utilizzo meno efficiente della memoria.

Il **firmware** originale dell'Apple II occupava **12 Kb** di memoria, ed era memorizzato su 6 ROM

on-board da 2 Kb ciascuna. Lo spazio di indirizzamento era compreso tra D000h e FFFFh. Inizialmente, solo quattro dei sei socket disponibili furono effettivamente utilizzati (per un totale di 8 Kb impiegati, rispetto ai 12 Kb disponibili). Nel dettaglio, il contenuto della memoria ROM era organizzato dal punto di vista funzionale nel seguente modo:

- **F8000h - FFFFh**: *System Monitor* (routine per la gestione dell'hardware). Il System Monitor è in sostanza una sorta di shell comandi molto semplificata. Esso consente infatti di monitorare e modificare il contenuto della memoria, eseguire il trace e debug di un programma applicativo, ed eseguire le istruzioni contenute a un certo indirizzo di memoria. E' inoltre corredato da un elevato numero di routine per la gestione dell'hardware, quali ad esempio: inizializzazione della memoria, lettura dei caratteri dalla tastiera, visualizzazione dei caratteri sullo schermo, salvataggio e caricamento dei programmi applicativi. Durante la fase di avvio, l'hardware dell'Apple II carica il vettore di reset, memorizzato all'indirizzo FFFCh; questo vettore punta proprio all'inizio (entry point) del programma System Monitor;
- **F689h - F7FCh**: *Interprete Sweet-16*. Si tratta di un'altra "chicca" nata dalla mente geniale di Steve Wozniak. Durante la scrittura dell'interprete BASIC, Woz dovette infatti affrontare il problema di come gestire i puntatori a 16 bit e la relativa aritmetica su una macchina a 8 bit. La sua soluzione fu tanto semplice quanto geniale: implementò a livello software una CPU "virtuale", composta da 16 registri interni a 16 bit (da cui il nome, Sweet16). Questi registri, chiamati R0-R15, sono fisicamente mappati sui primi 32 byte della memoria principale dell'Apple II e ognuno svolge una funzione ben definita (accumulatore, program counter, registro di stato, registro flag, ecc.). L'utente può accedere a Sweet16 tramite un'apposita chiamata a subroutine (cioè eseguendo la call a un indirizzo fisico ben definito), e passando i parametri che verranno poi interpretati ed eseguiti da Sweet16. Tra i comandi disponibili, ne esiste inoltre uno che permette di ritornare alla modalità tradizionale 6502, ripristinando il valore dei registri. Tutta la "macchina virtuale" Sweet16 era implementata in soli 300 byte di codice!;
- **F500h - F63ch**: *Mini Assembler*. Si tratta di un programma nato per agevolare l'immissione del codice macchina direttamente nella memoria del computer;
- **E000h - F424h**: *Integer Basic*. E' l'interprete per il linguaggio BASIC, scritto interamente da Steve Wozniak. Inizialmente avrebbe dovuto supportare l'aritmetica in virgola mobile, ma poi, a causa della mancanza di tempo, venne rilasciata una versione che gestiva solo numeri interi con segno a 16 bit.

L'USCITA VIDEO

Come abbiamo visto nella scheda tecnica, il microcomputer Apple II era equipaggiato nella versione base con 4 Kb di memoria RAM dinamica (DRAM). Di questa, circa 1 Kb era riservato alla memoria video, lasciando quindi 3 Kb disponibili come memoria di uso generale. In realtà, 768 byte erano comunque riservati per contenere le variabili del System Monitor, lo stack del processore, e il buffer di ingresso della tastiera. La

mappa completa della memoria relativa ai primi 4 Kb è pertanto la seguente:

- **80000h - 800FFh**: pagina zero della memoria, utilizzata per contenere le variabili del System Monitor;
- **80100h - 802FFh**: stack processore e buffer tastiera;
- **80300h - 803FFh**: spazio disponibile;
- **80400h - 807FFh**: memoria video;
- **80800h - 80FFFh**: spazio disponibile.

L'Apple II utilizzava un circuito per la generazione dei segnali video particolarmente efficiente, e anche in questo caso furono adottate delle tecniche innovative e inusuali per quei tempi.

Una soluzione particolarmente elegante escogitata da Woz riguarda il **rinfresco della memoria dinamica**, necessario per gestire correttamente le memorie DRAM installate sulla macchina. La soluzione consisteva nell'eseguire il rinfresco della memoria "automaticamente", incorporandolo nelle operazioni eseguite dal circuito di generazione del segnale video (la memoria video, come detto precedentemente, era mappata in memoria). Il rinfresco della memoria avveniva quindi "gratis", ed era totalmente trasparente all'utente non richiedendo alcun ciclo extended, missing, oppure delayed. Questa caratteristica dell'Apple II è nota anche con il termine di "**hidden refresh**".

Un'altra particolarità rigurda la modalità con cui vengono memorizzati i frame video in memoria. Anzichè adottare un criterio di memorizzazione sequenziale (ad esempio, prima riga del video, seguita dalla seconda riga, poi dalla terza, ecc.), Wozniak optò per uno **schema interlacciato 8:1** (per la modalità testo) e **16:1** (per la modalità grafica). Sembra che tra i motivi di questa scelta vi fosse la semplificazione della gestione del segnale video **NTSC**, utilizzato in tutto il Nord America, e una semplificazione notevole del circuito di generazione del segnale video. In pratica, nella memoria video gli indirizzi di memoria successivi a quelli che contengono i caratteri della prima linea sono utilizzati per memorizzare il contenuto della nona riga (non la seconda). Le locazioni di memoria successive conterrano poi i caratteri della riga 17 (non la terza), e così via. Nel caso di modalità testo (24 linee da 80 caratteri ciascuna), si ottiene la seguente mappa di memoria (la 1^ riga è seguita, in memoria, dalla 9^, e poi dalla 17^, e così via):

riga	indirizzo memoria video
0	0400h - 0427h
1	0480h - 04A7h
2	0500h - 0527h
3	0580h - 05A7
4	0600h - 0627h
5	0680h - 06A7h
6	0700h - 0727h
7	0780h - 07A7h
8	0428h - 044Fh
9	04A8h - 04CFh
10	0528h - 054Fh
11	05A8h - 05CFh
12	0628h . 064Fh
13	06A8h - 06CFh
14	0728h - 074Fh
15	07A8h - 07CFh
16	0450h - 0477h
17	04D8h - 04F7h
18	0558h - 0577h
19	05D8h - 05F7h
20	0658h - 0677h
21	06D8h - 06F7h
22	0758h - 0777h
23	07D8h - 07F7h

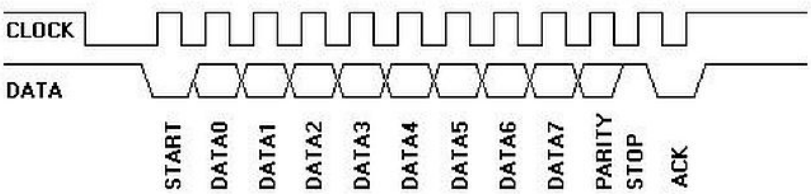
Nella versione di Apple II emulata, l'autore ha deciso di riutilizzare un suo precedente progetto, nel quale aveva realizzato un'interfaccia

video per Arduino Uno basata sull'utilizzo del processore secondario presente sulla board stessa (trattasi nella fattispecie dell'atmega16u2 flatpack utilizzato per implementare il bridge USB-seriale). Il vantaggio di questa soluzione è duplice: da un lato si può disporre di un'**interfaccia video VGA compatibile**, necessaria per produrre i necessari output dell'elaborazione, dall'altro lato si può utilizzare sempre e comunque una e una sola board Arduino Uno (il processore primario si occuperà del sw applicativo, mentre quello secondario dell'interfaccia video). E' importante che questa board sia almeno una Rev 3, in quanto nelle versioni precedenti era presente il convertitore FTDI in luogo dell'attuale Atmega16U2. Poichè, ovviamente, non si è scelto di emulare il circuito di generazione del segnale video originale, non aveva più senso mantenere in vita il meccansimo dell'**interleaving** visto precedentemente: l'emulatore decodifica infatti gli indirizzi di memoria video e li converte semplicemente in locazioni di memoria sequenziali. La **memorizzazione del frame buffer** viene inoltre compiuta dal microcontrollore secondario, liberando perciò su quello principale una quantità di memoria compresa tra 512 e 1024 byte.

L'Apple II originale permetteva di **visualizzare il testo** in tre formati distinti: **normale** (carattere bianco su sfondo nero), **inverse video** (opposto al precedente), oppure **flashing** (lampeggio del carattere). Questa informazione era contenuta nei bit più significativi di ogni carattere, memorizzato all'interno della memoria video. Per motivi legati alle tempistiche temporali, non è stato possibile implementare nel generatore video AVR VGA la gestione dell'inverse video (sono tuttavia emulate sia la modalità normale che quella flashing).

TASTIERA

La tastiera, storicamente, non è stata soggetta allo stesso grado di standardizzazione richiesto da altri componenti del microcomputer. L'Apple II stesso adottava un protocollo custom per la gestione della tastiera, basato su un layout di tipo QWERTY leggermente modificato. Per la versione emulata, l'autore ha scelto di utilizzare un vecchio dispositivo PS/2, particolarmente semplice da interfacciare con una board Arduino. Lo **standard PS/2** utilizza infatti una logica **TTL** a **5V**, e un protocollo seriale di tipo **sincrono**. Le uscite PS/2 sono di tipo open collector, e pertanto richiederebbero l'utilizzo di una resistenza di pull-up. Sulla board Arduino Uno l'operazione è agevolata, in quanto sono già disponibili diversi pin di I/O dotati di resistenze di pull-up interne.



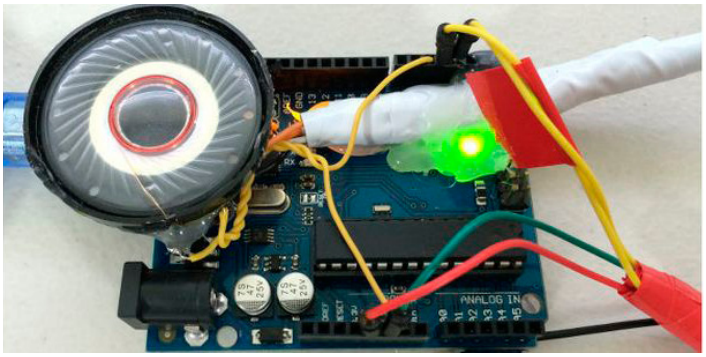
Come visibile nell'immagine precedente, lo standard PS/2 prevede una comunicazione basata su **word di 11 bit** (8 bit dati, 1 bit di start, 1 bit di stop, e 1 it di parità). Il bit di parità serve per controllare l'integrità dei dati, i quali vengono trasmessi sempre a partire dal bit meno significativo (DATA0). L'autore, per semplificare il progetto, ha deciso di non eseguire il controllo sul bit di parità. Per acquisire i dati inviati dalla tastiera, ha inoltre deciso di utilizzare un meccansimo basato sugli interrupt, in quanto ben supportato dalla piattaforma Arduino e più efficiente di un semplice polling.

MEMORIA DI MASSA

L'interfaccia originale dell'Apple II verso il registratore a cassette era semplice, veloce, con una velocità di trasferimento pari a oltre 180 byte al secondo (corrispondenti a circa 1500 baud). La temporizzazione era gestita via software, utilizzando come segnale di riferimento il clock del sistema. In pratica veniva utilizzata una modulazione di tipo FSK: quando il software richiedeva la memorizzazione di un programma, venivano attivate opportune routine del System Monitor in grado di generare (tramite dei semplici bistabili) un'onda quadra oppure rettangolare, a seconda che il bit valesse 0 oppure 1. Il bit 0 era definito come un ciclo completo di un'onda quadra con frequenza pari a 2 kHz (durata 500 µs), mentre il bit 1 da un ciclo completo di un'onda quadra con frequenza di 1 kHz (durata 1 ms). L'uscita del bistabile era poi collegata al jack "cassette data out" del computer. Per quanto riguarda invece l'ingresso ("cassette data in"), abilitato quando veniva richiesto il caricamento di un programma, esso era collegato direttamente a un circuito basato sull'utilizzo di amplificatori operazionali (741) per discriminare il valore di soglia.

L'autore è riuscito a implementare su Arduino queste funzionalità di gestione del registratore a cassette, sia la codifica che la decodifica, utilizzando un algoritmo molto simile a quello originale concepito da Steve Wozniak. Questo codice potrebbe potenzialmente essere utilizzato anche in altri progetti; ad esempio, il registratore a cassette potrebbe essere sostituito da un comune telefono cellulare sul quale è possibile memorizzare i dati del programma. In modo del tutto analogo è stata riprodotta l'uscita altoparlante (speaker).

L'immagine seguente mostra l'hardware completo dell'emulatore di Apple II con Arduino.



ANALISI DELLE PERFORMANCE

Parlando di emulazione, diventa importante stabilire come e quanto le prestazioni del sistema emulato si avvicinino a quelle del sistema reale. L'autore del progetto, eseguendo vari tipi di test con algoritmi e dati significativi, ha stabilito che l'Apple II emulato con Arduino è circa 8 volte più lento rispetto alla versione originale (MOS 6502 con frequenza pari a 1 MHz). Il microcontrollore Atmega328p che equipaggia la board Arduino Uno dispone inoltre di soli 2 Kb di memoria RAM, rispetto ai 4 Kb (minimi) del computer originale. Ciò non rappresenta tuttavia un grosso problema, in quanto la memoria video è stata spostata sul microcontrollore secondario, e rimangono perciò circa 1,5 Kb di memoria per contenere l'applicativo scritto in BASIC e i relativi dati.

Come **demo** per mettere alla prova il sistema emulato, l'autore ha utilizzato un programma per generare gli **insiemi di Mandelbrot**, compatibile con il sistema di sviluppo disponibile, vale a dire il BASIC con artimetica per interi con segno. L'algoritmo, noto come "escape time", utilizza un calcolo ripetitivo per determinare il numero di iterazioni richieste prima che l'equazione cominci a divergere. L'esecuzione dell'algoritmo ha richiesto qualche minuto di elaborazione sull'hardware emulato con Arduino, ma il risultato è soddisfacente, tenendo conto che è stato ottenuto con calcoli matematici con interi a

16 bit su una normalissima board Arduino Uno.
Un video relativo all'esecuzione di questo programma demo è disponibile a questo indirizzo.

CONCLUSIONI

Abbiamo visto in questo articolo come una board ecomomica e versatile quale Arduino Uno possa essere utilizzata anche per applicazioni inconsuete e tutto sommato poco convenzionali, vale a dire l'emulazione hardware di un computer che all'epoca aveva prestazioni di tutto rispetto. Tanto di cappello all'autore di questo progetto, che con caparbia e notevoli capacità personali è riuscito nell'ardua impresa.
Questo progetto può essere anche un valido punto di partenza per chi voglia cimentarsi in imprese simili, emulando magari altre CPU o microcomputer. Consiglio quindi di dare un'occhiata al sito dell'autore, dove possono essere trovate tutte le risorse necessarie per realizzare questo progetto (incluso il codice completo), e altro ancora.

ALLEGATI

Arduino AppleII Master

Video VGA-Apple

Test 6502

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/emulare-lapple-ii-con-arduino-uno

Un vibrometro real-time con Arduino per il settore industriale

di Giuseppe Silano

INTRODUZIONE

Il vibrometro (Figura 1), come anticipato, è un noto strumento utilizzato in ambito industriale, e non solo, per monitorare lo stato di funzionamento di una macchina, in particolare delle sue componenti meccaniche. Ad esempio, se all'interno di una catena di produzione una pompa o un motore si distaccano dal proprio alloggiamento essi inizieranno a vibrare con una frequenza superiore rispetto a quella nominale di lavoro. **Tale variazione sarà misurata attraverso un opportuno sistema HMI (*Human Machine Interface*, Interfaccia Uomo Macchina), come un sistema SCADA (*System Control and Data Acquisition*, Sistema per il Controllo e l'Acquisizione dei Dati)**, che farà scattare un allarme richiedendo l'intervento di un operatore e, a seconda della gravità del caso, potrà arrivare addirittura al distacco dell'alimentazione bloccando l'intero processo di lavorazione.

Figura 1: Vibrometro commerciale per misure sul campo.

Il vibrometro, come quello riportato in Figura 1, può essere utilizzato, avvalendosi di un nastro riflettente o di una ruota con contatto ottico, come tachimetro dunque per misurare la velocità istantanea di un motore o, più in generale, di un qualunque dispositivo. I principi fisici che ne regolano il funzionamento sono diversi, ad esempio, si utilizza una struttura del tipo massa-molla-smorzatore, tipica degli accelerometri (Figura 2), o ancora cristalli piezoelettrici (Figura 3), noti per la loro polivalenza: possono fungere da speaker, quindi produrre suoni (vibrazioni che spostano le molecole d'aria), se ai loro capi si applica una differenza di potenziale oppure da trasduttori se soggetti a vibrazioni, in questo caso ai loro capi leggeremo una differenza di potenziale (ecco spiegata la polivalenza).

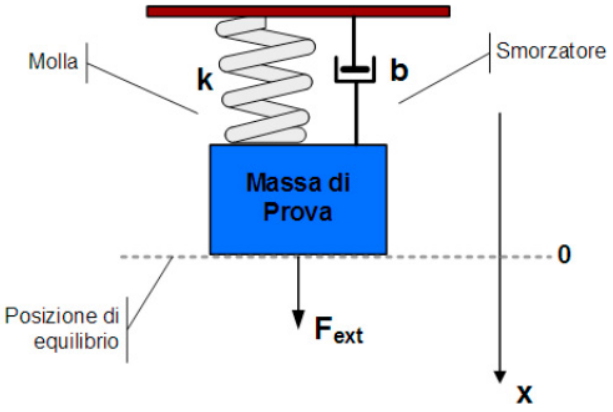


Figura 2: Sistema Massa-Molla-Smorzatore di un Accelerometro.

Figura 3: Schema di funzionamento di un cristallo piezoelettrico.

Alcuni dispositivi inglobano al loro interno dei filtri numerici regolabili che sopprimo gli eventuali rumori di fondo consentendo la misura di vibrazioni ad una specifica frequenza; in questo caso si parla di strumenti da banco.

L'IDEA ALLA BASE DEL PROGETTO

Chiarito a linee generali cosa sia un vibrometro, le branche di utilizzo dello strumento, ed accennati i principi fisici che ne regolano il funzionamento, passiamo ad illustrare l'idea alla base del progetto, tanto semplice quanto ingegnosa. Per l'acquisizione delle vibrazioni faremo uso di un cristallo piezo elettrico, semplicemente chiamato piezo, che sottoposto ad una pressione (la nostra vibrazione) genera una differenza di potenziale acquisita dall'ADC (*Analog Digital Converter*, Convertitore Analogico Digitale) del microcontrollore. La presenza di una tensione non nulla sui pin di input di Arduino incrementerà lo stato di un contatore, il cui valore verrà **mostrato in real-time sul display LCD**. È possibile parlare di azioni in real-time perché le vibrazioni hanno un periodo di generazione (intervallo temporale tra una vibrazione e la successiva) che è superiore, di almeno un ordine di grandezza, rispetto al tempo di l'elaborazione (dell'ordine dei millisecondi, ms). Ricordiamo, infatti, che l'ipotesi di sistema real-time sussiste se il dispositivo di controllo, tra un dato ed il successivo, permane in uno stato di *idle* (attesa) per un tempo non nullo. In sintesi, si può parlare di sistema real-time se i tempi di elaborazione sono inferiori rispetto a quelli con cui giungono i dati. Per il progetto in questione ci focalizzeremo sull'acquisizione di vibrazioni (ricevute sotto forma di colpi di una penna sfera) contenute all'interno di uno specifico range frequenziale (fisseremo delle soglie all'interno del software), non prendendo, dunque, in considerazione l'intero intervallo delle frequenze possibili. La scelta è da imputare alla scarsa risoluzione del convertitore ed al tempo di esecuzione dell'applicazione, oltre che al piezo non consono per essere utilizzato come trasduttore per frequenze elevate (vedi dettagli riportati in Tabella 1). Infatti, se aumentassimo la frequenza delle vibrazioni, conseguentemente riducessimo il periodo, perderemmo l'ipotesi di sistema real-time, che invece è la chiave di lettura dell'articolo. Completiamo la descrizione del progetto fornendo la definizione di trasduttore, ottima per tutti coloro che non masticano troppo di queste cose. **Un trasduttore** è un dispositivo che converte la variazione di grandezza fisica (le vibrazioni) in variazione di grandezza elettriche (la differenza di potenziale). Due gli intervalli di funzionamento:

- **Campo di misura (input range)** è l'intervallo di valori del misurando (grandezza che si intende misurare) entro il quale il trasduttore funziona secondo le specifiche. Il suo limite superiore è la **portata**.
- **Campo di sicurezza** del misurando: intervallo di valori del misurando al di fuori del quale il trasduttore resta danneggiato permanentemente. I suoi valori estremi sono detti di **overload** (sovraccarico) o **over-range** (fuori scala).

Queste informazioni, contenute all'interno dei datasheet, sono molto importanti perché ci consentono prima di tutto di non danneggiare il dispositivo (fornendo una tensione di alimentazione più alta di quella di lavoro), ed in secondo luogo, ma non per questo meno importante, di capirne la portata, il sovraccarico ed il fuori scala. Ad esempio, il nostro piezo non è in grado di misurare frequenze dell'ordine dei GHz, ma solo frequenze nell'intervallo da 200 Hz – 5 kHz.

Il funzionamento dell'intero sistema è mostrato nel video disponibile al link.

IL POTENZIOMETRO

Il **potenziometro** o **trimmer** è un dispositivo equivalente ad un partitore di tensione resistivo variabile (cioè a due resistori collegati in serie, aventi la somma dei due valori di resistenza costante, ma di cui può variare il valore relativo).

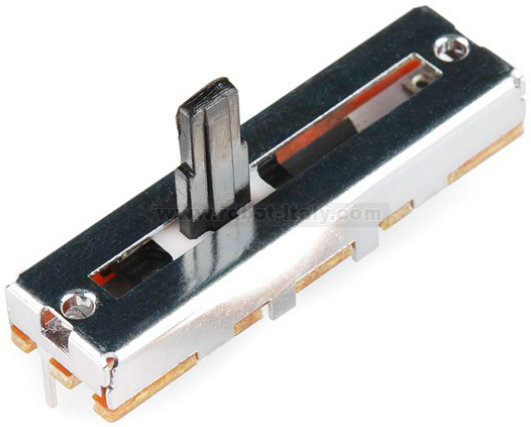


Figura 4: Potenziometro lineare.

Il potenziometro lineare (Figura 4 e 5) è costituito da un cilindro isolante su cui è fittamente avvolto un filo metallico con resistività opportuna, le due estremità sono connesse a due morsetti. Longitudinalmente al cilindro e da un'estremità all'altra, scorre un cursore recante un contatto strisciante sul filo, a sua volta collegato ad un morsetto.

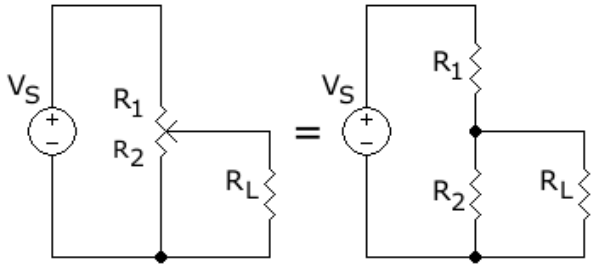


Figura 5: Simbolo e circuito equivalente di un potenziometro.

In origine i potenziometri erano utilizzati per misurare con precisione la tensione elettrica per confronto con una sorgente di riferimento. Il nome significa, infatti, letteralmente misuratore di potenziale (elettrico). Successivamente il termine si è esteso ad indicare l'impiego del dispositivo in applicazioni del tutto diverse dalla pura misura. Ancora oggi vengono utilizzati in alcune configurazioni a ponte, simili a quella di **Wheatstone** (Figura 6).

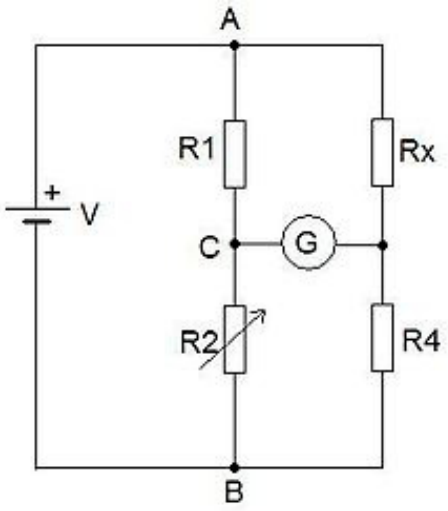


Figura 6: Ponte di Wheatstone.

Un utilizzo classico è nella regolazione di volume negli amplificatori audio. Ai due estremi viene applicato il segnale audio in entrata e dal cursore viene prelevato il segnale attenuato. Nel caso della regolazione del volume si preferisce usare potenziometri in cui la variazione del rapporto rispetto alla posizione non è lineare ma logaritmica, per compensare la sensibilità (logaritmica) tipica dell'orecchio.

Esistono **potenziometri rotativi** (Figura 7) di tipo multigiro (simile a quello utilizzato all'interno del nostro progetto), in cui l'albero deve compiere più di un giro, solitamente 5, 10 o 20, per andare da un estremo all'altro della corsa. Questa soluzione permette di variare con maggiore precisione la posizione. In questi dispositivi il cursore è traslato linearmente da una vite solidale all'albero.

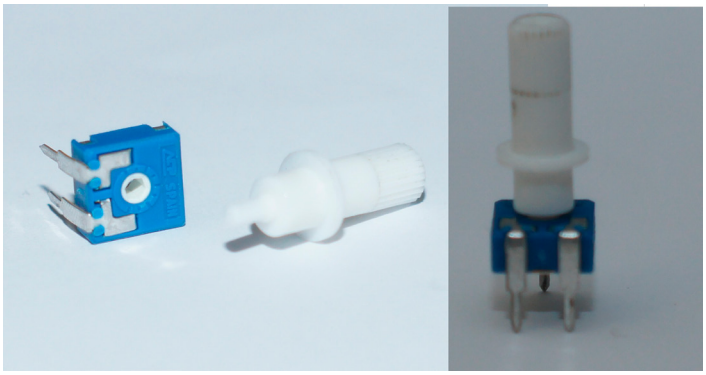


Figura 7: Potenziometro a giro singolo passivo.

L'unica anomalia che può verificarsi in un potenziometro nell'arco della sua vita operativa, è costituita dalla rumorosità, termine adottato per indicare la perdita di linearità del materiale resistivo costituente la pista su cui scorre il contatto strisciante. Nel caso si tratti di un'apparecchiatura audio, l'anomalia è percepibile durante l'azionamento del potenziometro, come vero e proprio rumore sovrapposto al segnale trattato; in apparecchiature di altro tipo, l'anomalia è percepibile come instabilità del segnale controllato, l'esempio più evidente è nel comando di posizione di un segnale sullo schermo di un oscilloscopio, nei casi peggiori risulta impossibile posizionare il segnale in un punto preciso dello schermo. All'interno del nostro progetto utilizzeremo il potenziometro per regolare il contrasto del display LCD.

IL DISPLAY LCD

Lo schermo a cristalli liquidi, in sigla LCD (Figura 8) dalla corrispondente espressione inglese *liquid crystal display*, è una tipologia di display a schermo piatto utilizzata nei più svariati ambiti, con dimensioni dello schermo che variano da poche decine di millimetri a oltre 100 pollici.

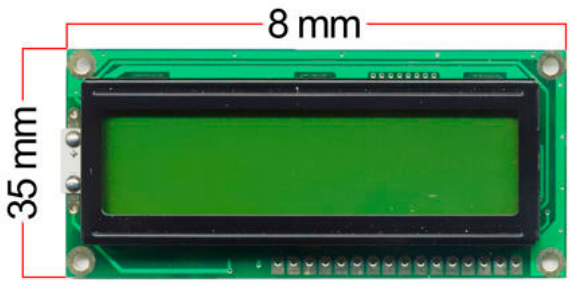


Figura 8: Esempio tipico di display LCD.

Da circa trent'anni, gli LCD sono utilizzati anche in ambito video, inizialmente nei computer portatili, in seguito anche nei monitor e nei televisori riuscendo, all'inizio del secolo, insieme allo schermo al plasma, a mandare in pensione il quasi centenario display CRT (lo schermo a tubo catodico).

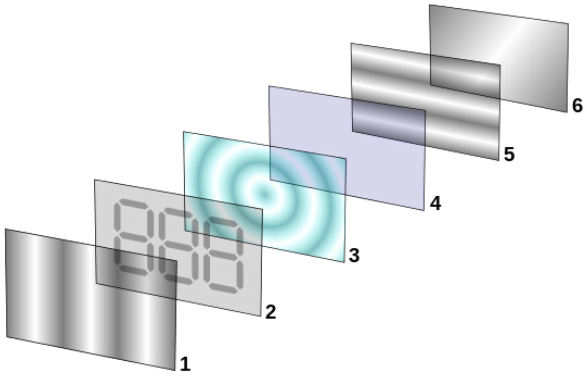


Figura 9: Esempio tipico di display LCD.

L'LCD (Figura 9) è basato sulle proprietà ottiche di particolari sostanze denominate **cristalli liquidi**. Tale liquido è intrappolato fra due superfici vetrose provviste di numerosissimi contatti con i quali poter applicare un campo elettrico al liquido contenuto. Ogni contatto elettrico comanda una piccola porzione del pannello identificabile come un **pixel** (o subpixel per gli schermi a colori), pur non essendo questi ultimi fisicamente separati da quelli adiacenti come avviene invece in uno schermo al plasma. Sulle

facce esterne dei pannelli vetrosi sono poi posti due filtri polarizzatori disposti su assi perpendicolari tra loro. I cristalli liquidi ruotano di 90° la polarizzazione della luce che arriva da uno dei polarizzatori, permettendole di passare attraverso l'altro. In questo modo solo parte della luce arriverà all'occhio, creando delle sezioni scure (i caratteri sul display). Una delle caratteristiche principali dei pannelli a cristalli liquidi (fatta salva la retroilluminazione) è il basso consumo di potenza elettrica, che li rende di per sé particolarmente indicati per applicazioni in apparecchiature alimentate da batterie elettriche, le cosiddette applicazioni low-power. Nell'ambito del nostro progetto, il display LCD è utilizzato per visualizzare le informazioni raccolte dal microcontrollore (il numero di vibrazioni).

IL CRISTALLO PIEZOELETTRICO

Compresi quali siano gli strumenti che ci consentono di regolare il contrasto dello schermo (il potenziometro, sarà più chiaro nei paragrafi successivi il suo utilizzo) e come visualizzare le informazioni raccolte (possibile attraverso il display LCD), concentriamoci sull'emissione del suono. Per farlo ci serviremo della **piezoelettricità** (Figura 3), la proprietà di alcuni materiali cristallini di polarizzarsi generando una differenza di potenziale quando sono soggetti ad una deformazione meccanica (**effetto piezoelettrico diretto**), ed al tempo stesso di deformarsi in maniera elastica quando sono attraversati da corrente (**effetto piezoelettrico inverso** o **effetto Lippmann**).

I cristalli piezoelettrici trovano applicazione in ambito musicale, dove si utilizzano i cosiddetti **pick-up piezoelettrici**: dispositivi in grado di rilevare le variazioni di pressione esercitate da una corda in vibrazione di uno strumento musicale,

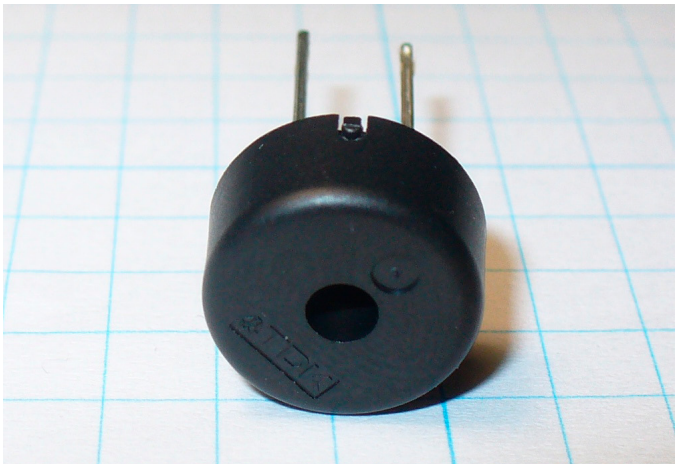


Figura 10: Piezo elettrico utilizzato per la realizzazione del progetto.

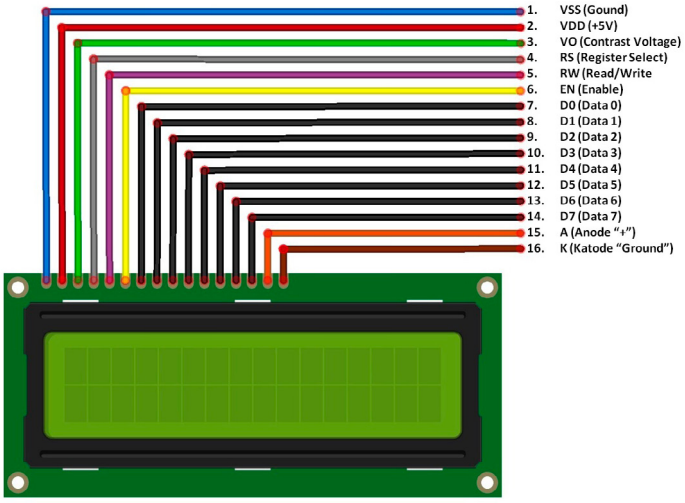


Figura 11: Schema collegamenti del display LCD.

ad esempio una chitarra, generando un segnale elettrico che poi viene amplificato. Nell'ambito del progetto, il cristallo piezoelettrico (Figura 10) è stato utilizzato per trasdurre (come anticipato nell'introduzione) le vibrazioni generate da una normale penna a sfera (il video mostra l'intera sequenza di acquisizione, elaborazione, e visualizzazione dei dati raccolti).

I COMPONENTI E LO SCHEMA CIRCUITALE

Capiti i principi che sono alla base del progetto, affrontiamo nel dettaglio gli aspetti realizzativi, prima fra tutti i componenti utilizzati allo scopo (Tabella 1).

Elenco componenti:	
R_1	1 MΩ 1/4 W 5%
R_2	220 Ω 1/4 W 5%
1 Potenziometro	1 - 10 kΩ
1 Display LCD	2 x 16 punti con cursore; alimentazione 5 V
1 Piezo	75 db 3 V 10 cm; 200 Hz – 5 kHz
Scheda	Arduino UNO

Tabella 1: Elenco componenti

Il display LCD è utilizzato per visualizzare i caratteri alfanumerici ha 16 colonne e 2 righe, per un totale di 32 caratteri. Ogni pin sul display è deputato ad una data funzione, in Figura 11 è riportato lo schema dei collegamenti.

La resistenza R_2 è utilizzata per alimentare il display LCD, un estremo è collegato alla massa della breadboard, l'altro al pin V_{SS} del display (pin 2). Prima di andare avanti nella descrizione dei restanti collegamenti, importante è prestare particolare attenzione al collegamento dei riferimenti (la massa) soprattutto quando si acquisiscono segnali con l'ADC del microcontrollore. Il mancato collegamento porterà a letture sbagliate, dunque errate azioni di controllo/misura.

A questo punto è conveniente capire come sono stati realizzati i collegamenti verso il display (vedi Figura 12), in particolare:

- il pin register select (RS) controlla la posizione dei caratteri sullo schermo;
- il pin read/write (R/W) pone lo schermo in modalità lettura o scrittura (in questo progetto il display è stato posto solo in modalità scrittura ponendo il pin a massa);
- il pin di abilitazione (E) comunica all'LCD che riceverà un comando;
- i pin dei dati (D0-D7) sono utilizzati per inviare i dati dei caratteri allo schermo, nel caso specifico ne utilizzeremo solo 4 (D4-D7);
- i due pin esterni dell'LCD (VSS e LCD-) sono collegati alla massa della breadboard.

Per quanto concerne l'alimentazione dell'LCD (V_{CC}), essa è collegata alla 5V della breadboard, collegata a sua volta alla scheda Arduino come visualizzata in Figura 13. Al contrario, il pin LED+ è collegato all'alimentazione attraverso una resistenza da 220 ohm (R_2), fondamentale per consentire l'accensione dei LED per la retroilluminazione del display.

Per quanto riguarda il potenziometro, esso è utilizzato per variare il livello di contrasto del display (come anticipato nel paragrafo precedente). I suoi tre pin, partendo da destra verso sinistra, sono collegati rispettivamente all'alimentazione (5V forniti dal microcontrollore alla breadboard), al pin V0 del display (vedi Figura 12), ed al riferimento (comune per microcontrollore, display e piezo).

Infine, il piezo è collegato ad un pin analogico di Arduino (A_0) ed alla massa utilizzando la resistenza R_1 (valori di resistenza più bassi rendono il piezo

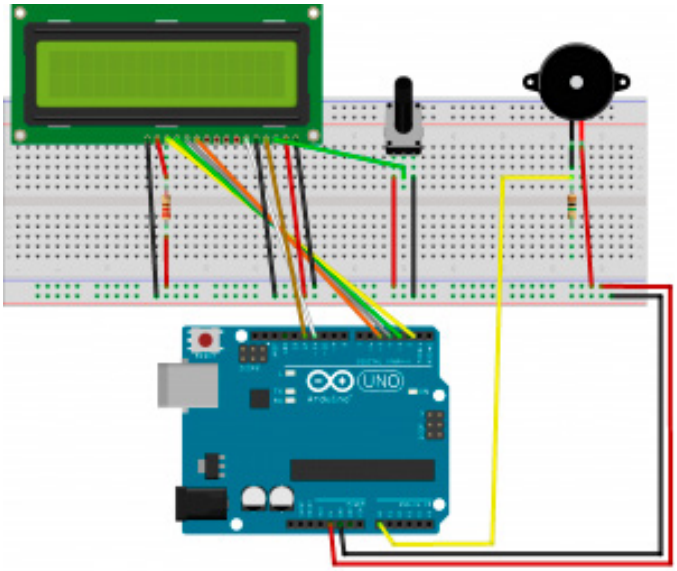


Figura 12: Schema collegamenti sulla breadboard.

poco sensibile alle vibrazioni), anche questo visibile nello schema dei collegamenti riportato in Figura 12.

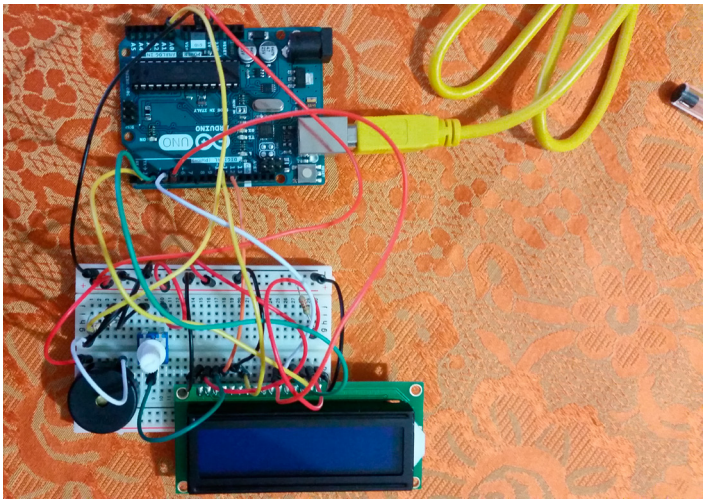


Figura 13: sistema finale.

Il sistema finale, alimentato con il computer via cavo USB (*Universal Serial Bus*, Bus Seriale Universale), è riportato in Figura 13. Prima di passare alla fase successiva, in cui verrà descritto il comparto software, il consiglio è di stare attenti durante il montaggio del piezo. In commercio sono presenti piezo forniti di collegamenti di colore rosso e nero, che rispettivamente indicano l'alimentazione e la massa. È fondamentale al fine di evitare corto circuiti, conseguentemente la rottura del cristallo, rispettare la polarità del dispositivo.

IL SOFTWARE DI CONTROLLO

In questa sezione andremo a descrivere il codice implementato per Arduino (in allegato all'articolo), entreremo nel dettaglio spiegandone ogni sua parte, con particolare attenzione alla gestione del display LCD attraverso l'ausilio della libreria LiquidCrystal. La libreria assume una rilevanza particolare per tutti coloro che si intendono molto di elettronica o di controlli, ma poco di informatica, o per tutti coloro che puntano a ridurre il tempo che intercorre tra l'idea di base ed il progetto finito, evitando di perdere tempo con fastidiose e noiose implementazioni (gestire l'invio di bit al display LCD può risultare un'azione abbastanza tediosa, e che facilmente porta in errore). Prima di proseguire nell'analisi del codice è conveniente averne una visione di insieme, per tale ragione è stato realizzato il diagramma di flusso riportato in Figura 14.

Come sapranno i tanti che hanno già utilizzato questo magnifico device (Arduino), ma che ribadiamo per completezza, nella prima parte del codice (Listato 1) vengono definite le variabili globali, quelle che hanno visibilità all'interno di tutto il programma, e che possono essere gestite da funzioni esterne. In poche parole, si dichiarano in questa sezione le variabili di cui vogliamo conservarne lo stato, ed eventualmente agire sullo stesso, operazioni fondamentale se non si vuole perderne il contenuto alla prossima sequenza di istruzioni cicliche (quelle contenute nel **void loop()**). All'interno di questo sezione del codice vengono incluse anche eventuali librerie, i cui metodi saranno invocati successivamente (nel nostro caso andiamo ad includere la libreria LiquidCrystal).

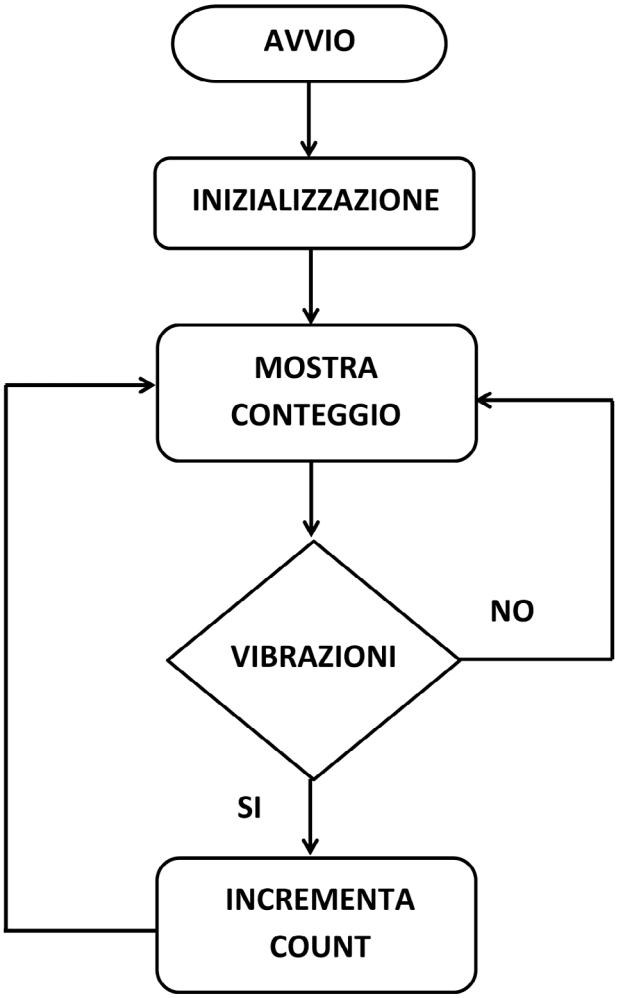


Figura 14: Diagramma di flusso del software di controllo.

```
%Listato 1
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
#define piezoPIN A0
float sensorValue;
int count = 0;
const int quietKnock=10;
const int loudKnock=100;
```

L'istruzione **#include<NomeLibreria.h>** ci consente di includere i metodi della libreria all'interno del codice Arduino, e di creare un oggetto descritto dalla stessa libreria (nel software lo abbiamo indicato con "lcd") al cui costruttore vengono forniti come parametri formali i pin di Arduino collegati al display. Nel caso in cui il lettore non volesse o potesse rispettare lo schema dei collegamenti presente in Figura 13, si consiglia di leggere attentamente la descrizione della libreria al link, in modo da evitare malfunzionamenti o danneggiamenti dello stesso display (per maggiori informazioni si veda il riquadro di approfondimento presente a fine articolo). I **#define** facilitano la stesura del codice, associando ad ogni pin un nome. Difatti, durante la stesura (del codice) risulta più conveniente scrivere il nome associato al piezo, piuttosto che ricordare il numero del pin a cui esso è collegato. Inoltre, migliora la leggibilità del codice, consentendo ad altri ma anche a noi stessi di poter capirne il suo comportamento a distanza di tempo. Se non facessimo uso del **#define** dovremmo continuamente leggere lo schema elettrico (Figura 12), con conseguenti maggiori probabilità di ricadere in errore. La variabile **count** e **sensorValue** contengono rispettivamente il numero di vibrazioni misurate dal nostro sistema, per questo motivo è inizialmente settata a zero, e la variazione di tensione letta ai capi del piezo. Le variabili **quietKnock** e **loudKnock** consentono di impostare le soglie rispettivamente per i livelli minimo e massimo delle vibrazioni. Settata le variabili globali si passa a definire quali saranno i pin di uscita, d'ingresso, e tutte quelle azioni che devono essere compiute prima dell'avvio del programma (Listato 2).

```
%Listato 2
void setup() {
lcd.begin(16, 2);
lcd.print("Inizializzazione...");
lcd.setCursor(0,1);
lcd.print("Vibrometro");
pinMode(piezoPIN, INPUT);
lcd.clear();
}
```

Le istruzioni **lcd.begin()**, **lcd.print()** e **lcd.setCursor()**, consentono rispettivamente di indicare quanto è largo lo schermo (nel nostro caso è composto da 16 colonne e 2 righe), di scrivere sullo schermo LCD, e di muovere il cursore sul display spostandolo dalla prima alla seconda riga. Al contrario, l'istruzione **lcd.clear()** consente di cancellare dallo schermo quanto scritto (i caratteri) con la funzione **lcd.print()**, riportando il cursore alla posizione (0,0). Ricordiamo, per chi non avesse alcuna base di linguaggi di programmazione, che il metodo per scrivere sul display è .print() mentre l'oggetto sul quale viene invocato il metodo è "lcd", per questo motivo scriviamo lcd.print(). L'istruzione **pinMode(PIN, MODE)** consente di definire quale sarà il pin utilizzato nella funzione void loop(), e soprattutto se si tratta di un pin di uscita oppure di ingresso. Nel caso d'interesse il pin A_0 è utilizzato per l'acquisizione del segnale analogico (la differenza di potenziale) dal piezo (listato 3).

```
%Listato 3
void loop() {
lcd.setCursor(0,0);
lcd.print("Vibrometro");
int knockValue=analogRead(A0);
delay(5);
if(knockValue < loudKnock && knockValue >
quietKnock){
count++;
}
lcd.setCursor(0,1);
lcd.print("Numero: ");
lcd.print(count);
delay(10);
}
```

La sezione **void loop()** (Listato 3) racchiude la porzione di software che viene eseguita ciclicamente, quella deputata alla misura del valore di tensione ai capi del piezo, all'aggiornamento del numero di vibrazioni conteggiate sul display, e così via. Le istruzioni iniziali (le prime due per essere precisi) consentono di settare il cursore, e scrivere "Vibrometro" sulla prima riga del display. L'istruzione **analogRead(PIN, VALUE)**, consente di misurare la variazione di differenza di potenziale, derivante dal conseguente colpo sul piezo (la penna a sfera che colpisce il piezo nel video). Il **delay(5)**, che segue l'istruzione **analogRead()**, dà il tempo all'ADC di fare il suo lavoro, ovvero acquisire la forma d'onda. Prima di proseguire oltre è bene introdurre alcuni concetti riguardo il campionamento, e in particolare sulla quantizzazione. **Arduino ha a sua disposizione 6 pin analogici, ognuno di essi è collegato all'ADC che ha una risoluzione di 10 bit** (cioè riconosce 2^{10} = 1024 intervalli di tensione differenti). Il convertitore ADC è settato di default per acquisire valori tra 0 e 5V. Questo vuol dire che il range 0-5V sarà diviso in 1024 intervalli, dunque il risultato della funzione **analogRead()** sarà un valore numerico compreso tra 0 e 1023 (estremi inclusi). Per ottenere la rispettiva unità fisica di riferimento (Volt, Ampere, etc.) basterà moltiplicare il valore numerico ottenuto per la massima tensione (in generale la quantità che si intende misurare) possibile, per poi dividere il risultato per il numero massimo di intervalli (1024).

*PhisicalValue=(SensorValue*MaximumValue) / NumberOfInterval*

Nel caso in cui volessimo acquisire un segnale tra 0 e 3.3V, gran parte dei livelli di quantizzazione sarebbero inutili. Per tale motivo è presente il pin 21 detto **AREF** col quale, per mezzo di un'apposita funzione, si può fissare il valore di riferimento (il valore massimo) per l'ADC. Par-

ticolare attenzione, prima di chiudere questa parentesi, deve essere posta al segnale dato in ingresso al convertitore: non si deve mai superare i 5V, ne fornire una corrente superiore a 0.5A. L'istruzione successiva **if(…)** verifica se vi sono state vibrazioni "corrette", ovvero se la vibrazione è compresea all'interno dell'intervallo fissato (maggiore del colpo debole e minore del corpo forte, rispettivamente **quietKnock** e **loudKnock**), e nel caso incrementa lo stato del contatore (**count++**). L'ultima istruzione lcd. print() si preoccupa di aggiornare il valore del count sul display, per poi riprendere nuovamente l'esecuzione del loop. Prima di concludere è bene sottolineare che la formula sopra indicata è una delle basi teoriche di misure elettroniche. Si rimanda il lettore più curioso ad approfondire tali concetti, utilizzando come parole chiave per le sue ricerche: **passo di quantizzazione**.

CONCLUSIONI

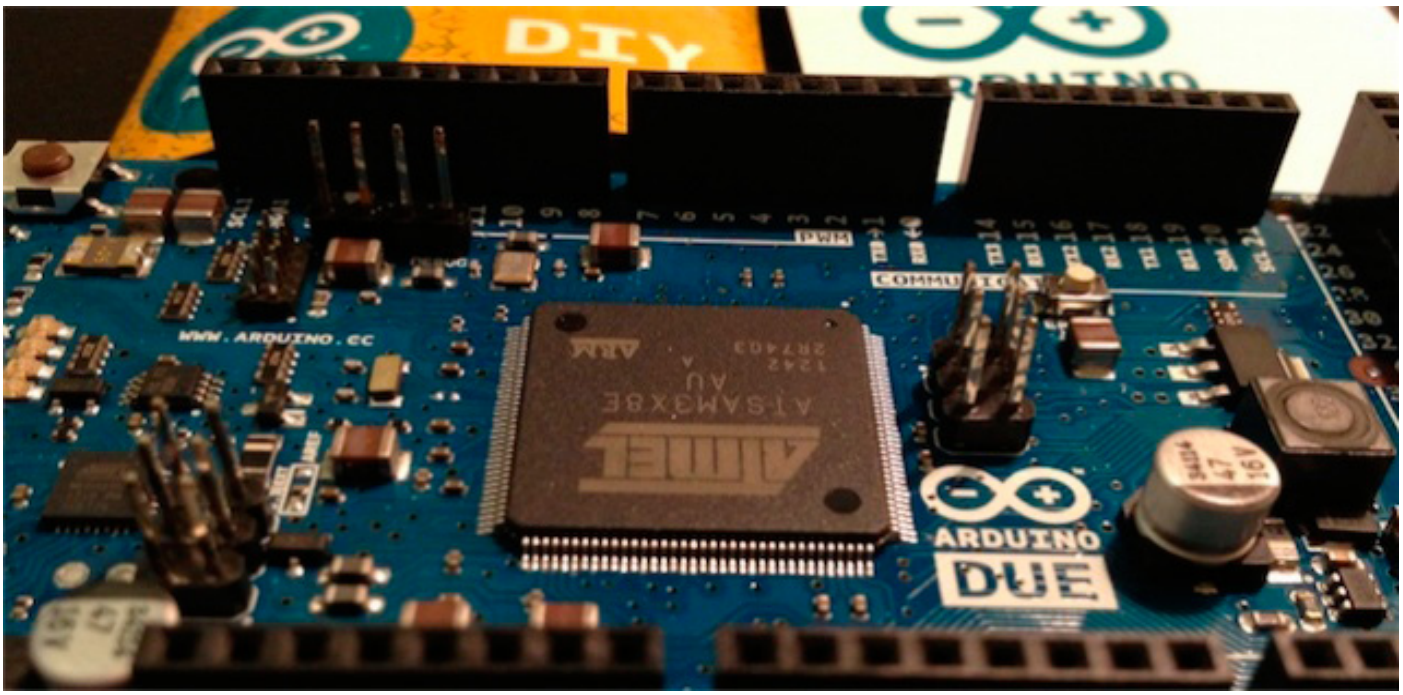
La realizzazione di un vibrometro con Arduino, progetto di relativa semplicità, ha consentito di affrontare concetti chiave: utilizzo di una libreria, campionamento, quantizzazione, acquisizione di segnali analogici, etc. Inoltre, l'implementazione del software ha consentito di introdurre un cardine nella realizzazione degli attuali sistemi di controllo: l'ipotesi di **sistema real-time**. Difatti, oggi, la quasi totalità dei sistemi, da quelli di videosorveglianza, a quelli su più noti droni (molto di voga in questo momento storico), si basano su questo concetto e senza il quale non sarebbero possibili molte delle cose utilizzate nella normale vita quotidiana.

ALLEGATI

Vibrometro_Arduino.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/un-vibrometro-real-time-con-arduino-per-il-settore-industriale

Arduino DUE



Arduino DUE Tutorial: presentazione e confronto con Arduino UNO

di Piero Boccadoro

Siamo abituati a pensare che un'era sia un tempo lungo, quasi infinito. Tuttavia non è proprio così; in un mondo che gira veloce come il nostro un'era può durare veramente poco e allora c'è bisogno di trovare un'altra definizione. Possiamo pensare ad un'era come un'epoca caratterizzata da tratti comuni e modi di fare simili tra loro. Bene l'era di Arduino è iniziata quasi dal niente ma grazie ad una scheda elettronica noi abbiamo completamente cambiato il nostro modo di vivere, di pensare ma anche di fare Open Source. Ecco, questa potenza oggi è ancora di più al nostro servizio.

La scheda Arduino DUE è la nuova arrivata della famiglia Arduino e della prima che utilizza un processore ARM a 32 bit, l'Atmel SAM3X8E AMR Cortex-M3, del quale approfondiremo qualche aspetto più avanti. Questo nuovo processore migliora tutte le funzionalità delle quali Arduino già disponeva e ne aggiunge di nuove che saranno davvero interessanti.

La nuova scheda offre **54 diversi pin di input/**

output (che già possiamo confrontare con i sei analogici e 13 digitali della versione UNO!) dei quali **12** possono essere utilizzati come **output PWM** con risoluzione selezionabile, **12** saranno per **input analogici** con **risoluzione a 12 bit**, ci sono **due uscite di tipo DAC e 4 UART** (porte seriali hardware), un oscillatore ad **84 MHz, due connessioni USB** (delle quali, come vedremo, una sarà dedicata per la programmazione), un header **ICSP** ed uno **JTAG** ed anche un **pulsante di reset** (che comunque era già presente anche nella versione Uno).

Sottolineiamo subito, in partenza, una delle più grandi differenze tra questa versione e la precedente: **questa volta la massima tensione che potrà essere tollerata ai pin sarà di 3.3V ed anche sul sito ufficiale di Arduino gli utenti vengono messi in guardia immediatamente: fornire i 5 V in ingresso sui pin causerà la rottura della scheda**. Tenete a mente, dunque, questa grandissima differenza, o limitazione se preferite.

Dicevamo degli ingressi USB, la scheda ne ha due: il primo, micro-USB AB, è un connettore nativo che agisce come host USB, il che significa che si può connettere una qualunque periferica esterna alla scheda, come un mouse piuttosto che una tastiera e perché no anche uno smartphone, che sarà supportato ampiamente come una fotocamera o un lettore mp3, visto che si tratta di un'uscita USB 2.0 a 480 Mbps. L'altra delle due, invece, viene utilizzata per effettuare debug.

La versione DUE segue un pinout così fatto: SDA e SCL sono vicini al pin AREF, IOREF, il quale permette l'utilizzo di uno shield connesso, e consente allo stesso la compatibilità con la scheda che funziona a 3.3 V, nonchè ai modelli basati su AVR che lavorano a 5 V. Infine c'è un pin non connesso che può essere utilizzato per usi futuri (previsti dall'utente).

Tra le caratteristiche vincenti del progetto c'è sempre stata anche la connotazione economica, ovvero la capacità della scheda di portare tante potenzialità in casa delle persone con costi davvero contenuti. La scelta, anche questa volta, è stata quella di inseguire e perseguire questa filosofia e pertanto Arduino DUE avrà un prezzo, tutto sommato, abbastanza contenuto sebbene poco meno che raddoppiato, che si attesterà su 39 €, escluse le spese di spedizione. La scheda è stata disponibile per breve tempo ed è subito esaurita. Stiamo, insomma, tutti aspettando il nuovo approvvigionamento che, tuttavia, non dovrebbe tardare ad arrivare.

Inoltre, il processore ARM offre un basso consumo energetico, permettendo quindi, un'ottima autonomia.

NUOVO CORE, NUOVE CHANCE

Sì, è proprio così: la vera novità di questa scheda è nel microprocessore che non ha soltanto 32 bit, che sarebbero semplicemente 4 volte quelli di Arduino uno (per maggiori informazioni, date uno sguardo alla pagina int type) ma una serie di altre caratteristiche che la rendono davvero molto accattivante. Alcune le abbiamo viste, altre le rivediamo meglio e più approfonditamente qui.

Il clock della CPU è ad 84 MHz e la dotazione di memoria risulta molto interessante dato che l'ammontare di RAM a bordo è pari a 96 kB (64 + 32) mentre quello di memoria flash è pari a 512 kB.

La scelta più intelligente e coraggiosa fatta da Banzi & Co. è stata quella di aprirsi alla famiglia dei processori a 32 bit per garantire alla scheda la possibilità di eseguire applicazioni real time

sviluppate specificatamente per creare piattaforme ad alte prestazioni, mantenendo bassissimi, o contenuti, i costi. Il processore permette di effettuare calcoli con performance davvero interessanti, garantendo anche tante nuove features.

Perché scegliere il Cortex-M3? Beh, questo è un microcontrollore, introdotto nel 2004, che è stato recentemente aggiornato con nuove tecnologie ed opzioni di configurabilità ed è il vero e proprio mainstream tra i processori ARM.

Uno degli aspetti più interessanti di questo "micro" è rappresentato dal **bassissimo consumo di potenza** in regime dinamico (scusate l'apparente contraddizione ma non c'è modo migliore per descriverlo, in realtà!), e, senza rimanere nel vago, stiamo parlando di **12.5 DMIPS/mW**.

Il processore esegue il set di istruzioni brevettato Thumb®-2 che permette un alto livello di prestazioni ottimizzato anche in relazione alla lunghezza del codice; alcune delle sue caratteristiche principali includono **hardware division, single cycle multiply e bit-field manipulation**.

La grande configurabilità del sistema permette fino a **240 operazioni di interrupt** per il sistema con un meccanismo **di priorità individuale selezionabile e riprogrammabile**. Il tutto gestito attraverso un clock di sistema integrato.

Anche dal punto di vista della connettività il sistema risulta piuttosto ricco e ben dotato. **La combinazione tra le caratteristiche e le performance rende i dispositivi basati su Cortex-M3 molto efficienti** specie nella gestione di canali di I/O multipli e protocolli standard come l'USB OTG (che non è altro che l'acronimo di On-The-Go).

Vediamo di entrare nel vivo della caratterizzazione del microcontrollori con le specifiche dello stesso:

IO VENGO DAGLI 8 BIT...

Sulle difficoltà e le caratteristiche, ma anche le prospettive, che il passaggio da un'architettura ad 8 bit, o magari a 16, verso quella a 32 abbiamo già letto, non troppo tempo fa, su queste pagine, qualcosa. Vediamo, tuttavia, di analizzare la questione più nello specifico.

I processor ARM Cortex-M offre diversi vantaggi, come stavamo dicendo, tra cui la maggiore densità del codice rispetto a quanto accade nelle architetture ad 8 o a 16 bit. Questo propone vantaggi significativi soprattutto in termini di riduzione dei requisiti di memoria e migliore utilizzo della memoria Flash a bordo.

Per quanto concerne la lunghezza delle istruzioni, e concezione comune che i microcontrollori ad 8 bit utilizzino istruzioni ad 8 bit mentre i microcontrollori basati su processori ARM Cortex-M utilizzino istruzioni a 32 bit. In realtà, per esempio, per quanto riguarda i PIC18 ed i PIC16, la dimensione delle istruzioni sono, rispettivamente, a 16 ed a 14 bit. Nel caso di architetture 8051, invece, sebbene alcune istruzioni siano lunghe solo un 1 byte, molte altre risultano lunghe 2 o 3 byte. Stesso discorso si può fare per le architetture a 16 bit, alcune delle quali hanno istruzioni che possono anche essere lunghe 6 byte, se non di più.

Abbiamo accennato prima, utilizzando la tabella, alla tecnologia Thumb-2. Questa è in uso sui processori Cortex-M e ciò permette loro una grande densità di codice. Questa tecnologia permette ai processori di operare con istruzioni del tipo 16-bit Thumb, ovviamente adattate ed estese per supportare le istruzioni a 32 bit che sono certamente più "potenti". In molti casi un compilatore C utilizzerà la versione delle istruzioni a 16 bit fin tanto che l'operazione può essere eseguita in maniera più efficiente rispetto

Features di un ARM Cortex-M3	
ISA Support	Thumb® / Thumb-2
Pipeline	3-stadi
Performance Efficiency	2.17 CoreMark/MHz - 1.25 DMIPS/MHz
Memory Protection	MPU ad “8 region” con “sub” e “background region”
Interrupts	Non-Maskable Interrupt (NMI) + da 1 a 240 interrupts fisicali
Interrupt Priority Levels	da 8 a 256 livelli di priorità
Wake-up Interrupt Controller	Fino a 240 Wake-up Interrupts
Sleep Modes	“Integrated WFI and WFE Instructions and Sleep On Exit capability. Sleep & Deep Sleep Signals.” “Optional Retention Mode with ARM Power Management Kit”
Bit Manipulation	“Integrated Instructions & Bit Banding”
Enhanced Instructions	“Hardware Divide” (2-12 cicli), “Single-Cycle” (32x32) “Multiply, Saturated Math Support”.
Debug	“Optional JTAG & Serial-Wire DebugPorts”. Fino ad 8 Breakpoints e 4 Watchpoints.
Trace	“Optional Instruction Trace (ETM), Data Trace (DWT) e Instrumentation Trace (ITM)”

alla stessa espressione scritta, però, nella sua analoga versione a 32.

E, visto che stiamo parlando di efficienza delle istruzioni, non possiamo delineare un quadro completo se non consideriamo anche il fatto che le istruzioni su questi processori sono certamente molto più potenti, come dicevamo prima. Ci sono molti casi in cui una singola istruzione Thumb è del tutto equivalente ad una istruzione su microcontrollori da 8 o 16 bit. Questo vuol dire che i dispositivi con Cortex-M hanno sostanzialmente codice più piccolo, e snello, che permette però di realizzare le stesse operazioni. Tutto questo può essere ben spiegato con la tabella che segue.

Altra cosa che è importante notare è che **questo tipo di processori supportano il trasferimento dei dati su architetture da 8 e 16 bit**, garantendo comunque un ottimo utilizzo della memoria. Questo vuol dire che i programmatori potranno continuare ad utilizzare gli stessi tipi di dati che hanno utilizzato sulle precedenti architetture.

Riguardo, poi, ai vantaggi dal punto di vista dell’efficienza energetica, e quindi del consumo, la richiesta di diminuzione degli stessi non fa che aumentare, dato l’incremento delle potenzialità che oggi si richiedono e la sofisticazione dei sensori analogici. Questo ha portato la necessità di **routine di pre-process** che possano snellire il codice finale ma anche diminuire il tempo di attività della macchina. La maggior parte dei dispositivi ad 8 bit non offre la possibilità di venire incontro a questa esigenza senza aumentare la frequenza di clock, e quindi il consumo di potenza. Pertanto gli sviluppatori embedded hanno, di fatto, puntato a nuove tecnologie. I dispositivi a 16 bit sono stati i primi ad essere utilizzati in questa direzione; tuttavia le prestazioni raggiunte erano, globalmente, insufficienti per effetto dell’aumento del duty-cycle. La frequenza di

Esempi a 8-bit		Esempi a 16-bit	Su ARM Cortex-M
MOV A, XL ; 2 bytes			
MOV B, YL ; 3 bytes	MUL AB; 1 byte		
MUL AB; 1 byte	ADD A, R1; 1 byte		
MOV R0, A; 1 byte	MOV R1, A; 1 byte		
MOV R1, B; 3 bytes	MOV A, B ; 2 bytes		
MOV A, XL ; 2 bytes	ADDC A, R2 ; 1 bytes		
	MOV R2, A; 1 byte	MOV R4,&0130h	
MOV B, YH ; 3 bytes	MOV A, XH ; 2 bytes	MOV R5,&0138h	MULS r0,r1,r0
MUL AB; 1 byte	MOV B, YH ; 3 bytes	MOV SumLo,R6	
ADD A, R1; 1 byte	MUL AB; 1 byte	MOV SumHi,R7	
MOV R1, A; 1 byte	ADD A, R2; 1 byte		
MOV A, B ; 2 bytes	MOV R2, A; 1 byte		
ADDC A, #0 ; 2 bytes	MOV A, B ; 2 bytes		
MOV R2, A; 1 byte	ADDC A, #0 ; 2 bytes		
MOV A, XH ; 2 bytes	MOV R3, A; 1 byte		
MOV B, YL ; 3 bytes			

clock richiesta ad un microcontrollori a 16 bit rispetto a quella di uno a 32, a parità di istruzione, risulta più alta ed è per questo che l'architettura a 32 bit ottimizza anche il consumo energetico.
Per quanto riguarda lo sviluppo, **i software per microcontrollori basati su processori ARM Cortex possono essere molto più semplici di quelli prodotti per microcontrollori ad 8 bit**; questo non è soltanto vero perché il processore Cortex può essere interamente scritto in C ma anche perché vengono utilizzate ed implementate una serie di features orientate al debug che

sono disponibili da più fonti, anche grazie al fatto che esistono diversi kit di sviluppo basati su quelle MCU.

QUALCHE DETTAGLI TECNICO

A guardar bene la scheda c'è davvero di che rimanere ammirati; oltre le caratteristiche riguardo il suo "cuore pulsante", ci sono anche le nuove caratteristiche che vi, proponiamo qui di seguito in questa facile e comoda tabella. Se avete presente com'era fatta la vecchia scheda vi renderete subito conto del fatto che c'è di che essere soddisfatti.

Microcontrollore	AT91SAM3X8E
Tensione di lavoro	3.3V
Tensioni di ingresso raccomandate	7-12V
Limiti massimi delle tensioni di ingresso	6-20V
Pin di I/O digitale	54 (di cui 12 per output PWM)
Pin di ingresso analogici	12
Pin di uscite analogiche	2 (DAC)
Massima corrente di uscita DC sulle linee	130 mA
Corrente DC per pin a 3.3V	800 mA
Corrente DC per pin a 5V	800 mA
Memoria Flash	512 KB disponibili
SRAM	96 KB (due banchi: 64KB e 32KB)
Frequenza di clock	84 MHz

Passiamo, adesso, ad una più dettagliata **analisi di tutti i pin** di cui la scheda dotata e grazie ai quali essa può essere configurata, potenziata e con la quale si può interagire. Tanto per iniziare, e qui lo vediamo molto più diffusamente, Arduino due è dotata di 54 differenti pin che ciascuno di questi può essere configurato per effettuare I/O. **Le librerie** che abbiamo già conosciuto ed utilizzato per la versione uno, ovvero **pinMode(), digitalWrite() e digitalRead() torneranno utili** anche in questa versione della scheda. Certo, alcune sono state riviste ma la sostanza è rimasta invariata. **Tutti i pin**, come abbiamo avuto modo di spiegare la volta scorsa e com'è giusto ricordare ancora, **lavorano con una tensione massima di 3.3 V**. In ingresso la scheda non sopporta più i 5V della versione precedente ed i produttori avvertono che tale valore di tensione, posto in ingresso, potrebbe portare la rottura della scheda.

Attraverso ciascun pin può scorrere **una corrente compresa tra 3 e 15 mA**, mentre **nella modalità di sink, questo valore risulta compreso tra 6 e 9, a seconda del pin**. Esiste anche un **resistore di pull-up** interno, che di default risulta disabilitato, del valore di **100 kOhm**.

Oltre a questo, esistono anche pin specializzati differenziati, per l'appunto, per la funzione e passiamo subito alla loro descrizione.

- **Serial 0**: pin 0 (RX) e pin 1 (TX);
- **Serial 1**: pin 19 (RX) e pin 18 (TX);
- **Serial 2**: pin 17 (RX) e pin 16 (TX);
- **Serial 3**: pin 15 (RX) e pin 14 (TX).

Questi, come indicato tra parentesi, possono essere utilizzati per effettuare trasmissioni di dati seriali TTL con un livello 3.3 V ed, a coppie, svolgono funzioni di ricevitore e trasmettitore.

Come da tabella informativa, la scheda anche **12 pin riservati alle funzioni di PWM**, in particolare quelli numerati **dal 2 al 13**. Gli **output**

che possono essere **forniti sono PWM ad 8 bit** e per utilizzarli possiamo fare ricorso alla **funzione analogWrite()**, che ben conosciamo. La risoluzione di questi otto tutto può essere cambiata mediante un'altra funzione nota a chi ha già programmato sulla scheda che è **analogWriteResolution()**.

Le comunicazioni seriali possono essere gestiti attraverso **SPI Header** (che su altre schede prendono il nome di **ICSP**); questi pin supportano le comunicazioni attraverso la **libreria SPI**, anche questa una vecchia conoscenza degli utenti Arduino. I pin vengono portati "all'esterno" su di un pin-header a 6 contatti che è fisicamente compatibile con la versione uno ma anche con Leonardo e Mega2560. Questo header può essere utilizzato solo per comunicare con altri dispositivi SPI e non per effettuare la programmazione del SAM3X.

Per quanto riguarda **CANTX** e **CANRX** non c'è molto da dire se non che si tratta dei pin di supporto per le protocollo di comunicazione di tipo **CAN** che, tuttavia, **non sono ancora supportata dalle API di Arduino**. Una predisposizione futura?

Anche in questa versione esiste il **LED 13** ("L"); utilizzato nella versione uno come test per verificare il funzionamento, anche in questa può essere utilizzato dall'utente. È anche possibile effettuare il **"dimming" del LED**, dato che il pin 13 è anche un output PWM.

Le comunicazioni **TWI** sono supportate e gli utenti potranno utilizzare **TW1** (**pin 20 e 21**) e **TW2** con l'ausilio della **Wire library**.

Per quanto riguarda gli **ingressi analogici**, ovvero i pin numerati da **A0 ad A11**, su ciascuno di questi ha una **risoluzione da 12 bit**, il che significa che ci sono **4096 valori possibili in ingresso**. Tanti? Pochi? Dipende, ovviamente!

Di **default la risoluzione** delle letture viene impostata **a 10 bit** per garantire la compatibilità con altre schede Arduino. Con la funzione **analogReadResolution()**, però, a questo si può certamente porre rimedio.

AREF è un pin connesso alla riferimento analogico del SAM3X grazie ad un resistore. Questo pin può essere utilizzato tramite la funzione **analogReference()**.

DAC1 e **DAC2**, invece, permettono di avere accesso ai due output analogici veri e propri che possono essere utilizzati per creare, per esempio, un'uscita audio grazie alla libreria apposita.

Come in altre schede, molto utile risulta la funzione di Reset che non manca grazie all'apposito pulsante.

LE COMUNICAZIONI

Vediamo, adesso, qualcosa di un po' più approfondito sulle comunicazioni. Arduino DUE permette la comunicazione sia con il computer sia con altri dispositivi, così come abbiamo già detto. Il SAM3X è dotato di una UART hardware ed altre tre per le comunicazioni seriali TTL a 3.3 V.

La porta di programmazione, che, come abbiamo detto, è una delle due USB che la scheda ha, è connessa ad un ATMega16U2, e fornisce una porta virtuale COM per la connessione con il computer. Il dispositivo viene riconosciuto dal sistema operativo Windows grazie ad un file ***.inf** mentre su OSX e su Linux essa verrà riconosciuta automaticamente (il solito Windows!).

Il 16U2 è connesso alla UART del SAM3X; RX0 e TX0 permettono la comunicazione seriale-USB per programmare la scheda attraverso questo microcontrollori. Il software include, come vedremo più avanti, il serial monitor, che ben conosciamo, e che permette l'invio e la ricezione di semplici dati di testo. **Durante l'u-**

tilizzo di questa "periferica" il LED RX e TX lampeggerà.

La porta USB nativa è connessa al microcontrollore SAM3X per permettere la comunicazione seriali CDC tramite USB. Questo permette di emulare un mouse o una tastiera USB. Questa seconda porta, tuttavia, permette anche la connessione fisica di mouse, tastiere o similari, per i quali componenti esiste una libreria specifica.

Uno degli aspetti fondamentali della scheda è la **protezione da sovracorrenti. Arduino due è dotato di un fusibile resettabile che protegge la porta USB del computer da cortocircuiti o sovracorrenti**; anche se la maggior parte dei computer ha il proprio sistema di protezione interna da problematiche di questo tipo, il fusibile rappresenta un ulteriore livello di protezione. Era già presente sulla scheda uno e viene riproposto in questo caso come misura utile. **Il valore di corrente al di sopra del quale non si può andare e 500 mA** applicati sulla porta; dovesse arrivare proprio questo valore, la connessione verrà interrotta immediatamente per evitare sovraccarichi.

ARDUINO: LE DUE VERSIONI A CONFRONTO

Che arrivasse questo momento, che proponessi un "testa a testa" era quasi scontato e probabilmente anche gli autori delle schede avranno pensato a come fronteggiare la necessità di argomentare l'introduzione di una nuova scheda con prestazioni che valessero la pena di comprarne un'altra. Ed effettivamente è così!

Le specifiche tecniche della nuova versione di Arduino le abbiamo già riportate la volta scorsa ma, questa volta, vi proponiamo quelle della versione uno sottolineando che cosa è diverso rispetto alla sua evoluzione:

Microcontroller	ATmega328 **contro un SAM3X attuale**
Operating Voltage	5V **contro i 3.3 attuali**
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (di cui 6 PWM) **contro i 54 (e 12) attuali**
Analog Input Pins	6 **contro i 12 attuali**
DC Current per I/O Pin	40 mA **contro i 130 attuali**
DC Current for 3.3V Pin	50 mA **contro gli 800 attuali**
Flash Memory	32 KB (ATmega328) dei quali 2 KB usati dal bootloader
SRAM	2 KB (ATmega328) **contro i 96 attuali**
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz **contro gli 84 attuali**

Insomma, insieme al numero di bit con i quali possiamo avere a che fare, che è passato da 8 a 32, praticamente ogni caratteristica della scheda è stata potenziata di un fattore quattro, rendendola semplicemente più performante in tutto. **Unica limitazione sembrerebbe essere proprio quel valore massimo di tensione di lavoro che è passato da 5 a 3.3 V**.

TIRANDO LE SOMME

Insomma, da quanto abbiamo visto fino a questo momento sembra che ci siano davvero tante buone notizie che attendono tutti coloro che decideranno di acquistare la scheda di cui ab-

biamo parlato. Tante sorprese, molte notizie di grande rilievo, tante buone idee che sono state messe in pratica e concorrono tutte insieme a dar da fare ai progettisti, agli ingegneri, agli hobbisti ed agli studenti di oggi e di domani.
Il suo aspetto grafico, il suo prezzo assolutamente competitivo e tutto quello di cui abbiamo parlato fino a questo momento la rendono la scheda che avete sempre desiderato! Unico neo, punto dolente o nodo da sciogliere, se volete, è la disponibilità: siamo tutti in attesa ed attendiamo i rifornimenti.
La prossima volta entreremo più nel cuore della scheda per vedere che cosa c'è, fare un confronto con la vecchia e prepararci a metterci le mani sopra. Rimanete con noi allora.

Alla prossima.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-due-tutorial-presentazione-e-confronto-con-arduino-uno

Arduino DUE Tutorial: dentro la scheda che ha cambiato il mondo

di Piero Boccadoro

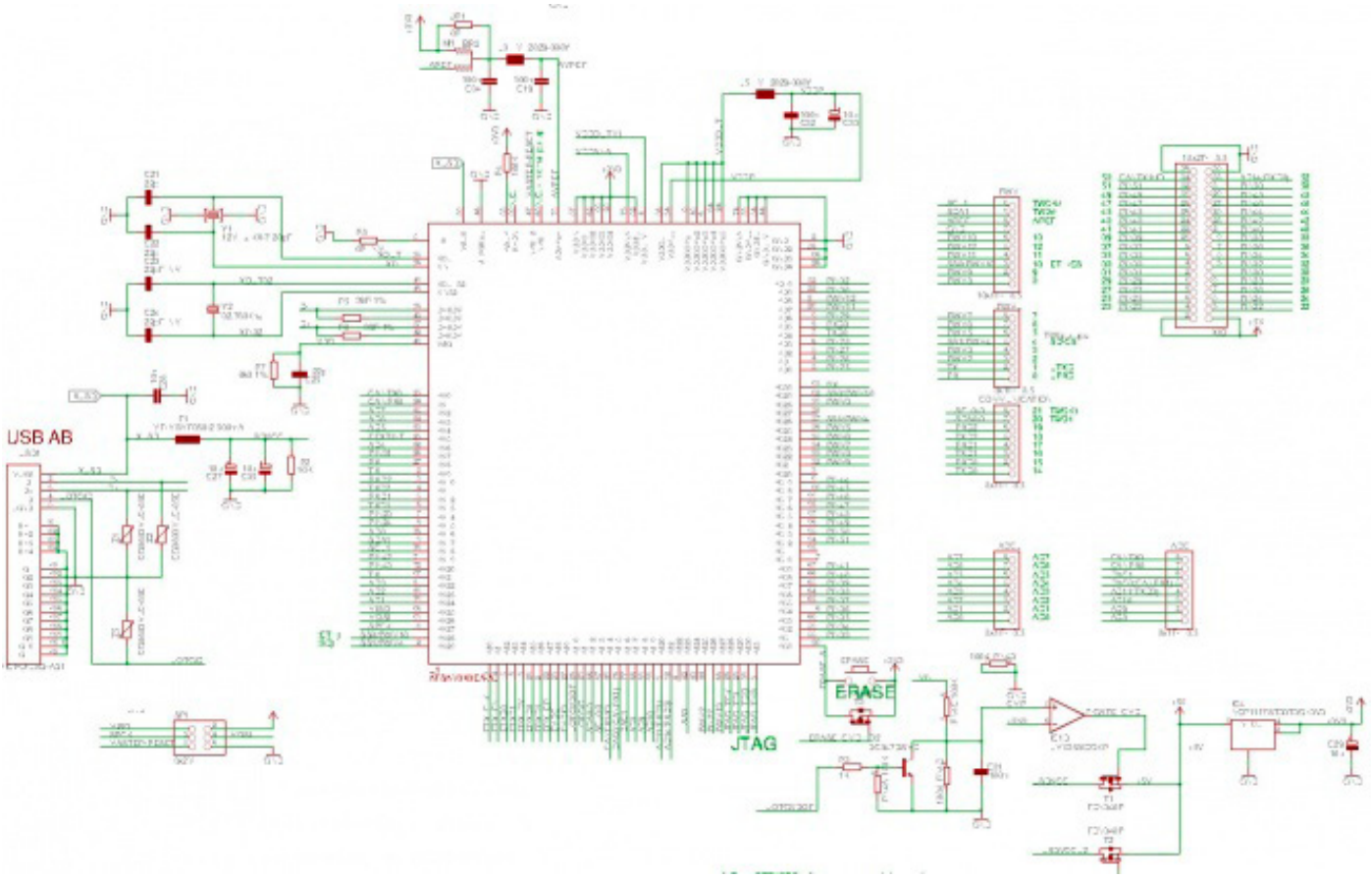
Bene, partiamo subito con l'analisi e vediamolo insieme nel dettaglio...

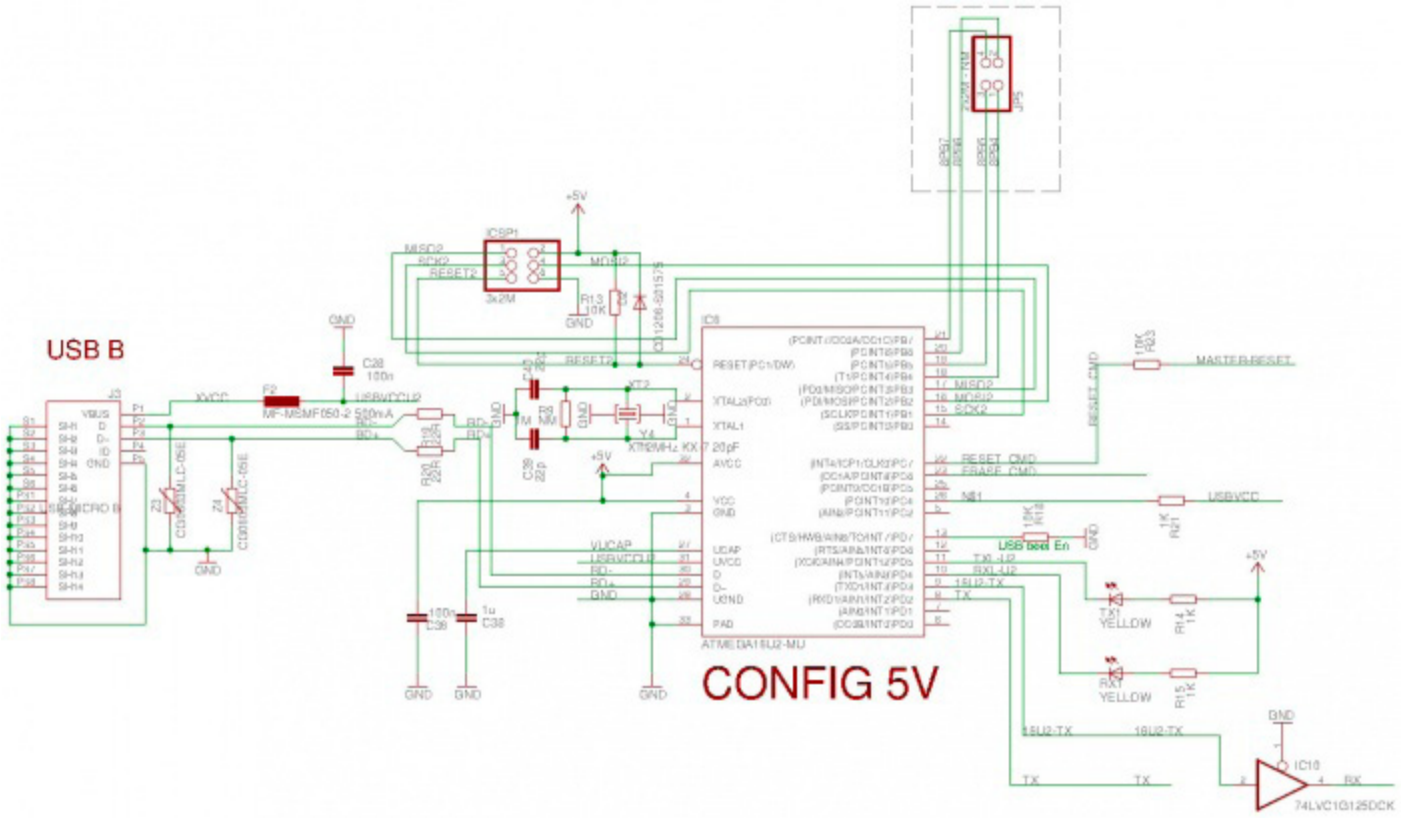
...LO SCHEMA ELETTRICO

Era d'obbligo fare un passaggio, anche piuttosto approfondito, su questo per chiarirci le idee. Quindi, ora che è venuto il momento: entriamo nello specifico della scheda e facciamo un'analisi della sua struttura. Per farlo, utilizzeremo qualche immagine dello schematico, tratta dalla documentazione ufficiale.

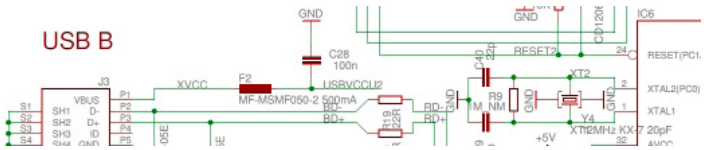
Nella prima di questa sezione vi proponiamo i principali collegamenti al SAM3x, della sezione di alimentazione, all'accesso al pin di riferimento, ai vari connettori e all'interfacciamento tramite porta USB AB.

Nella seconda immagine, invece, quello che vi proponiamo è ciò che c'è appena a valle della seconda porta, proprio per differenziare le due e capire, topologicamente, come sono state pensate. Come si vede, qui i collegamenti sono realizzati con il 16U2, utile per tutto quello di cui vi abbiamo parlato in precedenza.

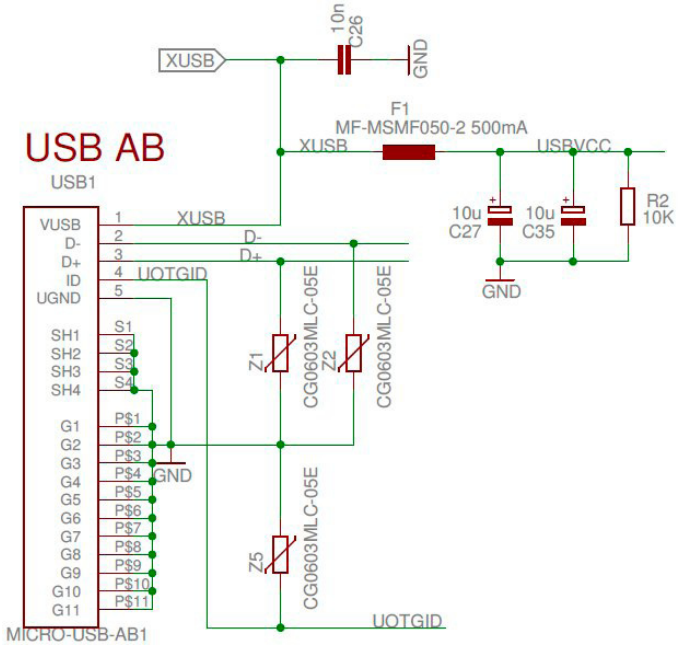




Proporre queste due immagini da sole è certamente assolutamente insoddisfacente ed ecco perchè ora vedremo più nel dettaglio alcune sezioni di nostro interesse. Cominciamo da questa immagine per vedere un dettaglio del collegamento con la porta USB (tipo B):

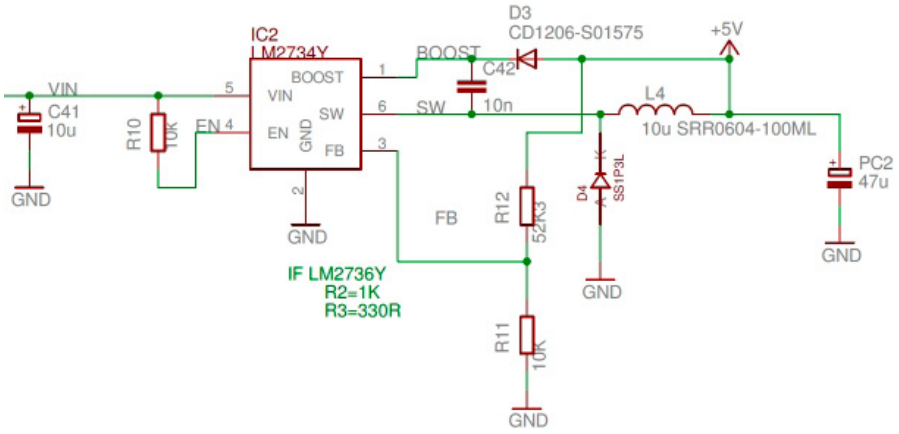


da qui risulta chiaro quale sia il collegamento tra l'ingresso USB tipo B verso la scheda. XVCC, proveniente dal pin P1, attraversa un MF-MSMF050-2 Multifuse SMD (una protezione) da non più di mezzo Ampere. Altrettanto avviene per l'ingresso USB tipo AB che vediamo qui di seguito:



I componenti **CG0603MLC-053** sono degli **ESD-suppressor**, ovvero dispositivi in grado di effettuare lo scarico di eventuali tensioni statiche parassite. In questo modo si provvede all'isolamento della scheda da eventuali sovratensioni causate da malfunzionamenti. Questi dispositivi sono stati collegati alle linee di segnale perchè sono proprio queste che vanno protette da

eventuali componenti di questo tipo sia per preservare l'integrità della scheda sia per tutelare il segnale da componenti di rumore.
Nella figura che segue

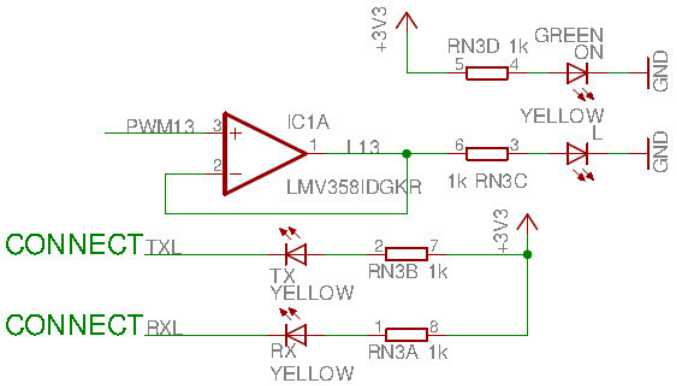


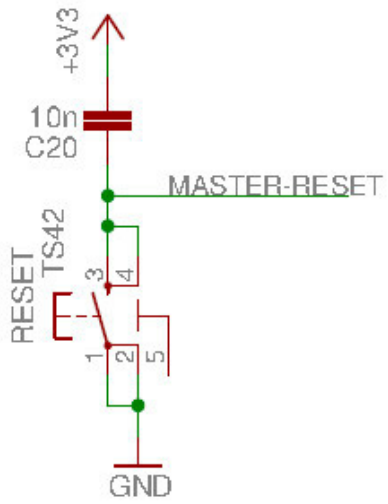
vi mostriamo la sezione di alimentazione del circuito. La tensione di ingresso a 5V diventa input dell'integrato LM2734Y. Si tratta di un DC-DC regulator della Texas Instruments, in particolare è uno Step-Down. Il suo scopo, come il nome suggerisce, è quello di abbassare il livello di tensione in ingresso fino ad ottenere una Vin "accettabile" per la scheda. Come vi avevamo accennato, infatti, **la scheda** assolutamente **non tollera tensioni pari a 5V** ma **il massimo che può sopportare** come tensione operativa **è 3.3V**. Ecco, quindi, come fa' ad abbassarla.
Questo integrato è molto utile per diverse ragioni che si possono intuire dando uno sguardo più approfondito alle sue caratteristiche:

- package **SOT-23-6** (che implica facilità di implementazione);
- range di tensione in ingresso: **da 3.0V a 20V** (che è certamente più del necessario in questo caso);
- range di tensione in uscita: **da 0.8V a 18V** (ed anche qui ci siamo abbondantemente);
- corrente di uscita: **1A** (che soddisfa certamente ogni condizione di carico di lavoro);
- frequenza di switch: **550kHz**;
- corrente di shutdown: **30nA**.

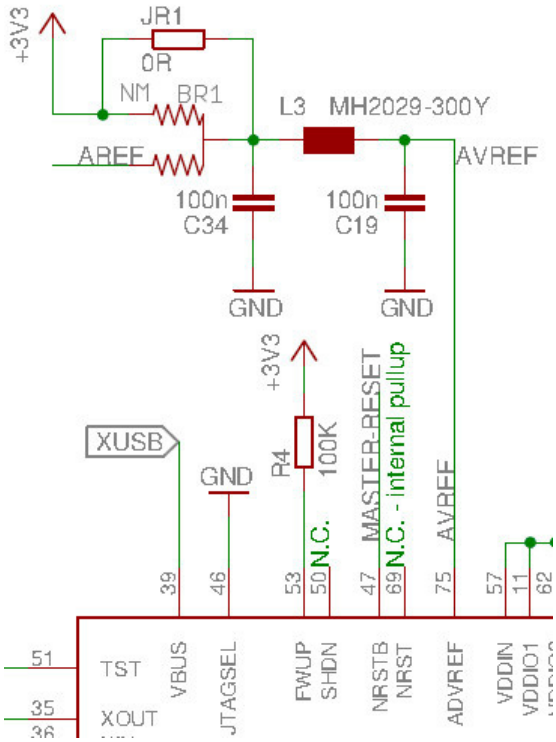
Queste ed altre features (che non è particolarmente utile o significativo riportare in questo caso), rendono l'integrato ideale per una serie di applicazioni tra cui: regolazione dei carichi, alimentazione di dispositivi a batteria, modem DSL, computer notebook e, ovviamente, molti altri ancora.
Altre due sezioni da commentare sono le seguenti: la prima è quella nella quale si controllano i LED presenti sulla scheda. Uno di questi, come vedremo tra un attimo, è pilotato utilizzando un amplificatore operazionale LMV358 e semplici resistori.
L'amplificatore operazione in questione lo troviamo, qui, utilizzato in configurazione buffer per realizzare un semplice inseguitore di tensione. Tuttavia questo integrato presenta diversi vantaggi e features interessanti. Esso fa' parte di una famiglia composta da LMV321, 324 e 324S che sono amplificatori singoli, doppi e quadrupli tutti a bassa tensione e che godono di swing rail-to-rail. Per questi dispositivi esistono, integrate funzioni di **power-saving shutdown** che permette la riduzione della corrente in uscita fino a 5 µA.
Si tratta di soluzioni di facile integrazione e di grande praticità quando si ha bisogno di operazionali versatili in quanto general purpose e con valori di slew rate fino a 1-V/µs e tensioni operative che possono variare da 5 a 30 V.
Immediatamente al di sotto troviamo i connettori TXL ed RXL con annessi LED di "controllo".
La seconda, invece, è quella grazie alla quale si può effettuare il RESET della scheda; questa utilissima funzione viene riproposta, tramite ap-

posito pulsante, anche in questa versione.





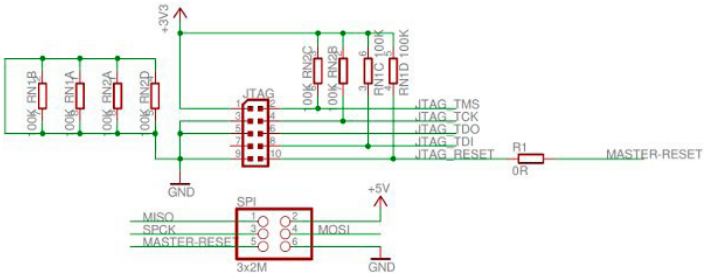
E parlando di cose che abbiamo già visto nella vecchia versione, ecco il pin AREF, riproposto su questa scheda con questo collegamento al microcontrollore.



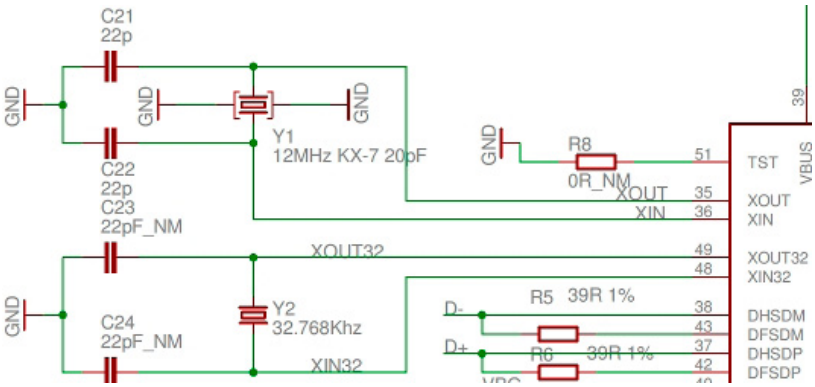
In questa figura si vede molto chiaramente una cosa interessante: un **filtro a "pi-greco"**. Si tratta di una semplice architettura passiva costituita da un induttore e due condensatori (che nello schema sono C34, L3 e C19) che permette, al dispositivo completo di rispondere ai dettami della normativa in materia di compatibilità elettromagnetica. In particolare, con questa architettura si ovvia al problema delle emissioni condotte. Il filtro è un **passa-basso** collegato come ultimo stadio di interfaccia tra dispositivo e tensione di alimentazione. In questo modo, le componenti che ciascun dispositivo elettronico tenderebbe ad emettere, vengono attenuate. Le frequenze cui è più frequentemente riferito lo scopo dell'applicazione di questi filtri sono 150 kHz - 30 MHz.

In questa tipologia di filtri è possibile anche ottenere l'attenuazione del solo modo comune; questo è un fatto di grande importanza visto che la corrente di alimentazione, di solito, presenta componenti di solo modo differenziale (pertanto non viene attenuata).

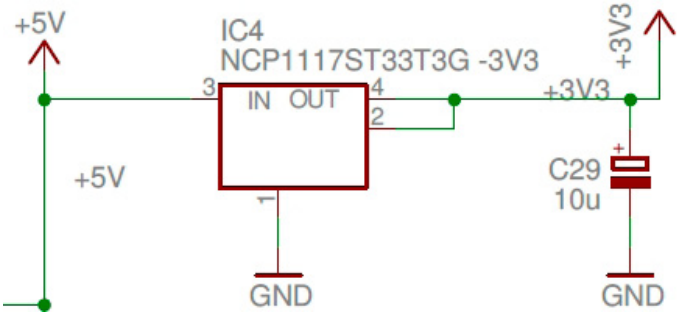
Per quanto riguarda le interfacce, invece, qui di seguito riportiamo la sezione dello schematico in cui si evidenziano le interconnessioni per gestire le interfacce JTAG ed SPI verso i connettori.



Andiamo, ora, a vedere la sezione dei quarzi connessi al SAM3X.



Altra porzione dello schematico che ci interessa è quella dell'**LDO**, come mostrato di seguito:

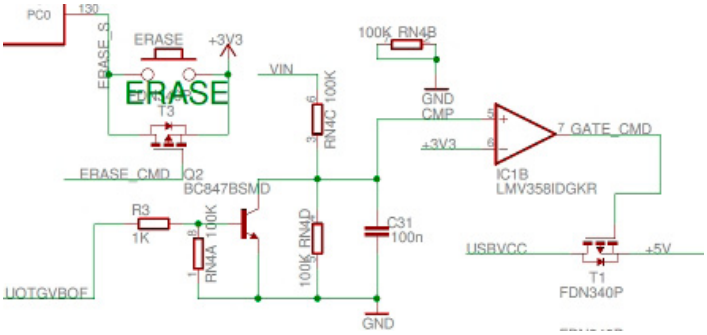


LDO, come noto, è una sigla che sta' per **Low-DropOut regulator**, ovvero regolatore a basso dropout. Si tratta di un regolatore lineare di tensione che può operare con una tensione differenziale in-out molto bassa. Il vantaggio nell'utilizzo di queste soluzioni è costituito dal fatto che si possono ottenere tensioni di lavoro molto più basse, valori d'efficienza più alti e minore dispersione del calore.

Questo regolatore, in particolare, annovera tra le sue caratteristiche:

- Tensione di ingresso (max): **20 V**;
- Tensione di uscita: **3.3 V**;
- Tensione di dropout (max): **1.1 V @ 100 mA**;
- Corrente di uscita: **1 A**;
- Regolazione carico: **4.3 mV**;
- Tipo di uscita: **fissa**;
- Temperatura di lavoro massima: **125 °C**;
- Stile di montaggio: **SMD/SMT**;
- Package: **SOT-223-4**;
- Corrente di bias di ingresso (max): **6 mA**;

Passiamo ora alla condizione di **ERASE**, gestita come nello schema che segue:



Il pulsante evidenziato è a presidio del comando di **"erase"** cui è dedicata la porta 130 (PC0).

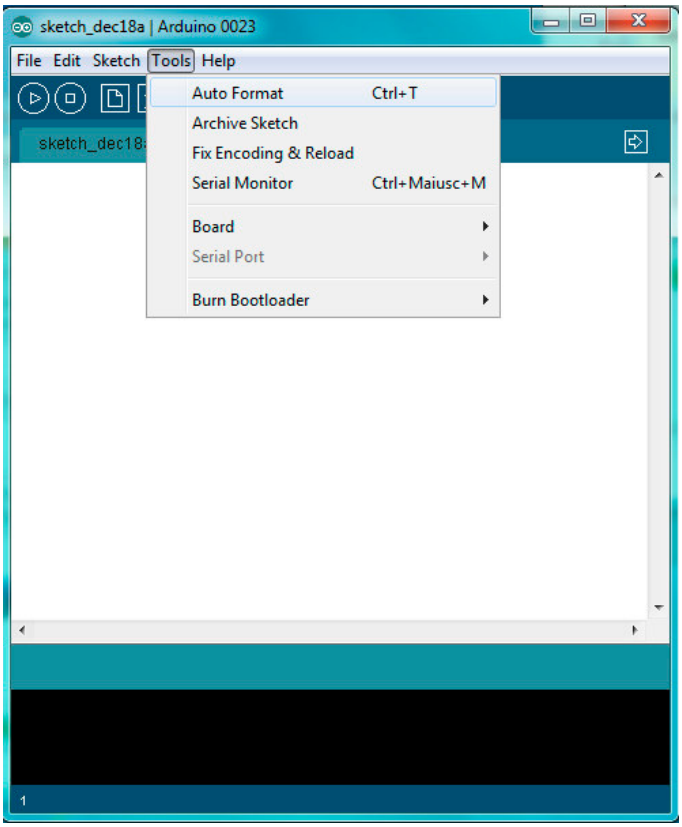
Qui vale la pena di fare un appunto funzionale al segnale UOTGVBOE. Qui è possibile portare al valore 0 l'alimentazione dell'USB (porta) quando si trova in modaltà HOST (ovvero quando viene alimentata dall'esterno). VBUSPO sta' per **VBus Polarity Off** e prevede che "0" implichi che il segnale di uscita sia quello della modalità di default (attivo alto) mentre "1" corrisponda alla modalità invertita, ovvero attivo basso.

A questo punto, per completezza, vi segnaliamo i riferimenti dove potrete trovare la documentazione del progetto: gli EAGLE files sono disponibili qui: arduino-Due-reference-design.zip, lo schematico è scaricabile a questo indirizzo arduino-Due-schematic.pdf mentre il pin mapping è riportato qui: SAM3X Pin Mapping page

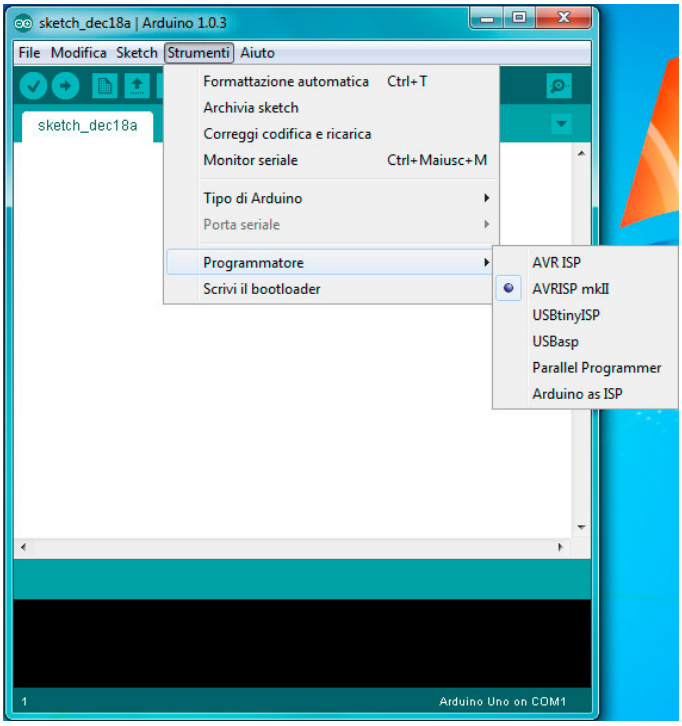
L'INTERFACE DEVELOPMENT ENVIRONEMENT

Sotto questo punto di vista le cose non sono, in realtà, cambiate in maniera altrettanto notevo-

le; l'IDE, nella sua versione 0023, aveva questo aspetto:



mentre l'attuale versione, la 1.0.3, ha quest'altro:



e, al di là di **qualche accorgimento grafico** la verità è che non è cambiato molto.

La funzionalità è sempre quella, con i soliti limiti e le caratteristiche che abbiamo tutti imparato a conoscere. Il piccolo restyling ha portato lo spostamento di qualche pulsante e l'utilizzo di una paletta cromatica leggermente diversa. Anche lo splash screen è stato modificato ma queste sono tutte innovazioni piuttosto marginali.

Questo è davvero un peccato perchè sarebbe molto utile che l'IDE di questo tool dimostrasse di potersi completare con strumenti di debug più sofisticati che permettessero al programmatore di capire come e perchè certe problematiche insorgono. Io stesso, in alcuni esperimenti fatti, ho notato che il compilatore pasticcia un po' quando deve gestire variabili globali e nulla pare poter dissipare i miei dubbi sul fatto che la struttura "bloccata" costituita da setup() e loop() non causi errori di interpretazione e compilazione e, pertanto, di esecuzione. Recentemente uno dei nostri utenti ha fatto un esperimento interessante ragionando su come poter utilizzare Eclipse per superare proprio queste difficoltà. Per chi se lo fosse perso, ecco che cosa ha fatto.

Una delle novità più interessanti riguarda, però, la **localizzazione**: come tutti i progetti Open Source, anche questo prevede un grosso e sostanziale apporto della comunità che, in questo caso particolare, si sta occupando di provvedere alla localizzazione della GUI per la programmazione nelle varie lingue. Da non molto tempo è disponibile una versione localizzata ad opera di Sara Gallo e Simone Majocchi. Qualcuno dice "when ther's a will there's a way"; personalmente ritengo che se tutto questo è riferito a ciò che succede nell'alveo dell'Open Source, allora è vero.

Dal punto di vista della programmazione, come avevamo già accennato in precedenza e riprendiamo qui, **sarà** certamente **necessario rivedere alcune delle librerie** poiché la programma-

zione su un microcontrollore a 32-bit non può essere fatta nello stesso identico modo rispetto come la si faceva sugli 8.

Non solo, nella nuova versione, ovviamente, **sono supportate una serie di schede che prima non potevano essere utilizzate**. D'altronde tutto questo è abbastanza fisiologico, visto che nel tempo il supporto deve essere garantito per ogni singola versione rilasciata.

Anche il menù "programmatore", presente tra gli strumenti, in cui son presenti diverse possibilità, risulta più popoloso del precedente, in modo da dare all'utente maggiore "spazio" rispetto a quello che aveva in precedenza.

Insomma, si tratta del "solito" strumento un po' potenziato per essere molto più utile a tutti gli utenti.

Poco o niente, quindi, **è stato fatto rispetto alla sua più grande pecca, ovvero l'assenza** pressochè totale **di strumenti di debug** più professionali e completi.

L'IDE di Arduino potrebbe diventare un laboratorio di grande pregio se solo alcuni strumenti fossero inseriti. In particolare, come molti utenti avranno avuto modo di sperimentare durante l'uso, non esiste una gestione degli errori, un feedback completo ed esaustivo che descriva l'errore in maniera particolareggiata. Cose che chi conosce ambienti come il Visual Studio è ben abituato ad utilizzare, e talvolta, diciamocelo, anche a dare per scontate.

IN CONCLUSIONE

Vi è venuta un'idea che vorreste davvero riuscire a mettere in pratica? Avete deciso di potenziare il vostro smartphone con un apparato GPS portatile che possa tenere anche traccia della posizione tramite un supporto di memoria?

Volete costruirvi un antifurto intelligente ed automatizzato per la casa e che comunichi grazie ad Android?

Oppure volete collegare il lettore Mp3 alla videocamera per aggiungere degli effetti speciali ai vostri filmati?

Insomma, tutte queste cose e molte altre ancora, compresa qualunque idea che abbiate avuto e che non siete riusciti a mettere in pratica su Arduino UNO per via delle sue comunque limitate potenzialità, possono finalmente diventare realtà grazie alle prestazioni ed alla potenza di calcolo ed alla velocità di risposta della nuova incarnazione del progetto Arduino. Ancora un esempio di come in una dimensione di poco più grande di una carta di credito si possa raggiungere un'enorme potenza di calcolo ed una versatilità di poco inferiori rispetto a quelle di un computer completo.

Arduino DUE si candida a pieno titolo a continuare il lavoro iniziato dal suo predecessore strizzando l'occhio al mondo del fai-da-te ma soprattutto a quello della didattica. Un'eccellenza del Made in Italy come il mondo può soltanto sognare. L'esempio più fulgido di che cosa gli italiani, anche nel campo dell'elettronica, sono capaci di fare e di quanto abbiano da insegnare.

L'unica cosa che c'è da sperare è che il gruppo al lavoro su Arduino decida di iniziare lo sviluppo di un ambiente di programmazione che includa strumenti più adatti al debugging ed all'analisi di ciò che succede durante la compilazione e la

successiva esecuzione del programma, in modo da rendere questa e le altre schede della famiglia controllabili in toto.

Prima di lasciarci, un breve annuncio finale sulla questione della disponibilità. La situazione, quando avete letto lo scorso articolo, era di esaurimento delle scorte disponibili praticamente su tutti i canali di distribuzione. Mentre leggete queste ultime righe dell'articolo, è avvenuto il nuovo rifornimento e pertanto la scheda viene nuovamente distribuita, non ultimo grazie allo Store di Arduino. Il "blackout" nella disponibilità, insomma, è durato solo alcune settimane per cui... cosa aspettate?

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-due-tutorial-dentro-scheda-che-ha-cambiato-mondo

Arduino DUE Tutorial: Atmel SAM3X8E ARM Cortex-M3 CPU

di Piero Boccadoro

Dopo aver analizzato con la dovuta attenzione lo schema elettrico di **Arduino DUE**, ci siamo già fatti un'idea abbastanza chiara di quelle che sono le sue reali potenzialità.

Il **pinout** che abbiamo visto, soprattutto evidenziando alcune porzioni di grande interesse per noi, sottintende il fatto che venga configurato (via hardware) opportunamente in modo tale da essere pronto all'uso.

Oggi vedremo meglio com'è fatto, quali sono le caratteristiche peculiari dell'architettura che andiamo a studiare ma soprattutto quali sono le sue reali possibilità anche rispetto ai diretti concorrenti.

La serie SAM3X/A di Atmel fa parte della famiglia di microcontrollori Flash basati sull'architettura dei processori **RISC** a **32 bit** dell'**ARM Cortex-M3**. La massima frequenza di lavoro è di **84 MHz** ed al suo interno è possibile trovare fino a 512 kB di memoria Flash e sino a 100 kB di memoria SRAM.

Il pinout, di cui abbiamo ricordato in precedenza, non è altro che il modo in cui ci si può interfacciare, in maniera user-friendly, a tutte le specifiche funzioni accessibili tramite pin del micro controllore. È per questo scopo, infatti, che, come abbiamo visto, esistono accessi diretti ad interfacce quali UART, SPI, I2C ed altre ancora, cui poter connettere periferiche.

Ma non c'è solo questo, in realtà, visto che tra le tante funzioni esistono anche convertitori **ADC** e **DAC**.

Altra funzionalità per la quale questo microprocessore risulta molto interessante è la sua possibilità di utilizzare facilmente la libreria **QTouch**, garantendo totale controllo di pulsanti, slider e qualunque altro supporto hardware compatibile.

Tra le applicazioni per cui questo dispositivo è considerato particolarmente utile ci sono quelle di tipo "networking" e pertanto risulta essere molto interessante nell'ambito dell'automazione sia in campo industriale sia, volendo, nel campo domestico.

LE FEATURES PRINCIPALI

Parliamo del **Core**:

- ARM® Cortex®-M3 revisione 2.0 (con frequenza di lavoro fino a 84 MHz);
- Memory Protection Unit (MPU);
- instruction set Thumb®-2;
- 24-bit SysTick Counter;
- controller Nested Vector Interrupt.

Abbiamo già accennato qualcosa sulla **memoria**; ecco che cosa mancava:

- da 256 a 512 Kbytes di memoria Flash embedded a 128-bit wide access;
- da 32 a 100 Kbytes di memoria SRAM embedded;
- 16 Kbytes ROM con bootloader embedded;
- Static Memory Controller (SMC): supporto per SRAM, NOR, NAND. Controller NAND Flash con 4-kbyte RAM buffer ed ECC.

Vediamo, adesso, qualcosa in più sul **sistema**:

- **regolatore di tensione embedded per operazioni single supply**;
- POR, BOD e timer Watchdog;
- oscillatori al quarzo o ceramici: da 3 a 20 MHz principale ed opzionale;
- clock real time a 32.768 kHz;
- oscillatore RC interno ad alta precisione da 8/12 MHz;
- PLL interno dedicato al clock ed uno dedicato all'Host/Device USB 2.0;
- sensore di temperatura;
- fino a 17 canali per periferiche DMA (PDC) e 6 per central DMA con DMA dedicato per Host/Device USB.

Modalità low power:

- sleep e backup mode; consumo ridotto fino a 2.5 µA;
- backup domain: VDDBU pin, RTC, 8 registri di backup a 32-bit;
- ultra low-power RTC.

Le **periferiche**, invece, sono:

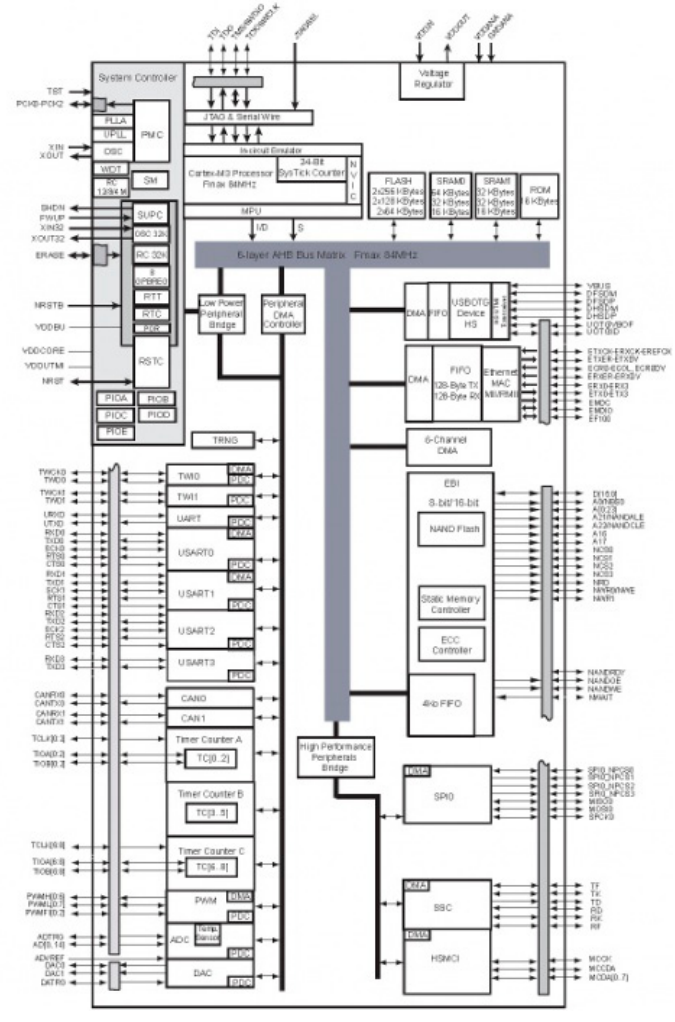
- USB 2.0, Device/Mini Host (480 Mbps, 4-kbyte FIFO);
- fino a 4 USARTs (ISO7816, IrDA®, SPI, supporto Manchester e LIN);
- 2 TWI (compatibili con I2C), fino a 6 SPIs, 1 SSC (I2S), 1 HSMCI (SDIO/SD/MMC) con massimo 2 slots;
- Timer&Counter a 32 bit e 9 canali;
- PWM fino a 8 canali da 16-bit;
- RTT a 32-bit ed RTC con calendario e funzione d'allarme;
- ADC con guadagno programmabile e modo d'ingresso differenziale da 16 canali, 12 bit;
- DAC 2 canali e 12 bit;
- Ethernet MAC 10/100 con DMA dedicato;
- due controller CAN;
- un TRNG (True Random Number Generator).

Le operazioni di **I/O** sulla scheda possono essere gestite con:

- fino a 103 porte di I/O con diverse sensibilità (edge o level) e debouncing, glitch filtering e resistori in serie on-die;
- fino a 6 PIO parallele da 32 bit.

Per quanto riguarda le soluzioni di vendita, i **Packages** sono 4:

- 100-lead LQFP, 14 x 14 mm, pitch 0.5 mm
- 100-ball LFBGA, 9 x 9 mm, pitch 0.8 mm
- 144-lead LQFP, 20 x 20 mm, pitch 0.5 mm
- 144-ball LFBGA, 10 x 10 mm, pitch 0.8 mm



A scopo riepilogativo, ecco, grazie alla prossima figura, un'immagine che spiega nel dettaglio come sia fatto il diagramma a blocchi del microcontrollore in esame.

Dal punto di vista del **Debug** e **Testing**, le features previste sono:

- debug access a tutta la memoria ed ai registri all'interno del sistema compresi quelli del Cortex-M3, all'interno dei quali il core è attivo, in halt oppure in stato di reset;
- debug port Serial Wire (SW-DP) e JTAG Debug Serial Wire port (SWJ-DP);
- unità Flash Patch and Breakpoint (FPB) per la creazione di breakpoint e patch;
- unità Data Watchpoint and Trace (DWT) per l'implementazione di watch points, data tracing e system profiling;
- Instrumentation Trace Macrocell (ITM) per supporto di operazioni come printf;
- supporto per Boundary-scan JTAG IEEE® 1149.1 su tutti i pin digitali.

Diamo, adesso, uno sguardo un po' più approfondito alle periferiche embedded ed iniziamo parlando della **SPI** (**Serial Peripheral Interface**). Diciamo intanto che è garantito il supporto per comunicazioni con dispositivi esterni collegati tramite seriale.

- 4 chip select con supporto per decoder esterni che permette la comunicazione con un massimo di 15 periferiche;
- memorie seriali, come DataFlash ed EEPROM 3-wire;
- periferiche seriali, tra cui ADC, DAC, LCD Controllers, CAN Controllers e sensori;
- co-processori esterni.
- Quest'interfaccia permette anche le comunicazioni master-slave su bus grazie a:
  - chip select a lunghezza programmabile da otto a 16 bit;
  - fase e polarità programmabili per chip select;
  - ritardi di trasferimento programmabili tra comunicazioni consecutive;
  - mode fault selezionabile.

Il trasferimento dei dati è anche piuttosto rapido grazie anche al fatto che la linea di chip select può essere "lasciata alta" (attiva) per velocizzare il trasferimento sullo stesso dispositivo.

Anche il trasferimento dei dati, e quindi la capacità del canale DMA, è stata ottimizzata grazie ai canali dedicati per trasmissione e ricezione.

Quando parliamo di **TWI**, ovvero di **Two Wire Interface**, ci riferiamo ad un'interfaccia che permette le comunicazioni in modalità master, multi-master e slave. Tra le sue caratteristiche c'è anche la garantita compatibilità con l'interfaccia two-wires di Atmel ma anche con I2C e serial memory. Possono essere utilizzati, in maniera selezionabile, uno oppure due o anche 3 byte per gli slave address. Tra le caratteristiche abbiamo anche:

- operazioni read/write sequenziali;
- Bit Rate fino a 400 kbit/s;
- General Call supportata in modalità Slave.

Le **UART**, nel quadro generale delle interfacce di cui si potrà fare uso, rivestono una grande importanza per via delle molteplici caratteristiche previste, tra cui la possibilità di impostare il baud rate ma anche la generazione di parità pari, dispari, mark o space. La generazione di segnali di wake up, le modalità di test Remote Loopback, Local Loopback ed Automatic Echo completano il quadro.

Continuiamo sull'argomento "interfacce" per

parlare della **Controller Serial Synchronous** (**SSC**). Esso permette la comunicazione seriale sincrona in applicazioni audio oppure, in generale, nelle telecomunicazioni. Questo sarà vero anche quando verranno utilizzati CODECs in modalità Master o Slave, I2S, TDM Buses, lettori di carte magnetiche e così via dicendo.
Ricevitore e trasmettitore sono indipendenti ed il clock divider è comune. È possibile configurare il sincronismo di frame e la lunghezza dei dati mentre sia il ricevitore sia il trasmettitore possono essere programmati per iniziare automaticamente la comunicazione oppure per effettuare la rilevazione di eventi differenti all'interno del frame del segnale di sincronismo.
Per quanto concerne il **Timer Counter** (**TC**), abbiamo tre differenti canali a 32 bit. Le sue molteplici funzionalità includono la misura di frequenza, il conteggio di eventi ma anche la misura di intervalli (di tempo), la generazione di impulsi ed altre funzioni quali PWM, Delay Timing.
Ciascun canale può essere configurato direttamente dall'utente e può contenere tre differenti input esterni (clock) e cinque interni e due segnali di I/O multi-purpose.
Quando parliamo di interfaccia per schede multimediali ad alta velocità (**High Speed Multimedia Card Interface**, **HSMCI**), ci riferiamo ad una dotazione per questo processore che garantisce la compatibilità con le MultiMedia Card Versione 4.3, le irrinunciabili (ormai) SD Memory Card (v. 2.0) ma anche CE-ATA (1.1). È garantito il supporto per le modalità ad alta velocità di trasferimento dei dati e sono predisposti due supporti per slot multiplexed.
Il tutto è dotato di protezione contro modifiche ai dati non previste on-the-fly per i registri di configurazione.
E veniamo adesso ad una delle interfacce più utili di questo microprocessore che permette l'equipaggiamento con pieno potenziale e la facilità di interfacciamento di Arduino DUE, ovvero lo **USB On-The-Go** (**UOTGHS**).
Il supporto permette Low/Full/High-Speed (LS/FS/HS) e On-The-Go, 1.5Mb/s, 12Mb/s, 480Mb/s. Tra le features abbiamo:

- 10 Pipes/Endpoints;
- 4K bytes di DRAM Embedded Dual-Port;
- fino ad due banchi memoria per Pipe/Endpoint;
- configurazione flessibile Pipe/Endpoint e gestione di canali DMA per un massimo di sei;
- tansceiver UTMI On-Chip che include Pull-ups/Pull-downs;
- pad OTG On-Chip che include un comparatore analogico.

Altro aspetto fondamentale del SAM3X è l'accoppiata **ADC**-**DAC**, una dotazione straordinariamente utile se si pensa che convertire i dati è un'esigenza davvero primaria.
La risoluzione, per entrambi, è di 12 bit ed il tasso di conversione non supera 1 MHz.
L'**alimentazione** si "estende" **da 2.4 a 3.6 V** ed è possibile selezionare ingressi single ended o differenziali per entrambi. Il guadagno e l'offset, per entrambi e per ciascun canale, sono selezionabili a partire da un valore massimo full scale input fino a zero.
Il convertitore analogico-digitale offre fino a 16 canali d'ingresso analogico indipendenti e per ciascuno è possibile optare per l'attivazione oppure la disattivazione.
Trigger software e/o hardware sono di tipo esterno oppure possono corrispondere ai trigger TIOA (uscite del TC). È garantito il supporto PDC ed esiste la possibilità di effettuare la con-

figurazione del timing per l'ADC.
Il sistema può prevedere il wake-up automatico perché viene rilevato un evento di trigger ed altrettanto si può fare per riportare il sistema nello stato di sleep mode; questo vale anche per il convertitore e, quindi, non solo il sistema (globalmente) ma anche soltanto il convertitore può godere di questa funzionalità.
Per quanto riguarda il convertitore DAC, invece, più specificatamente dobbiamo dire che esiste la possibilità di godere di Trigger multipli per ciascun canale. Il suo utilizzo può anche essere quello di fornire un input ad un comparatore analogico oppure un altro convertitore ADC.
È anche possibile utilizzarlo in modo tale che si abbia ridotto consumo di potenza.
Fino a questo momento risulta evidente che si tratta di un sistema piuttosto smart, molto avanzato che prevede funzionalità anche straordinariamente versatili ed è proprio per questo motivo che utilizzare queste periferiche, con la potenza delle comunicazioni che si possono mettere in piedi, utilizzando i diversi protocolli, è garanzia del fatto che **utilizzare Arduino DUE potrà essere davvero divertente**, non foss'altro che tutta questa potenza prima non esisteva per la scheda.
Mancano all'appello due importanti esponenti della categoria dei controller cui abbiamo la possibilità di accedere: **CAN** ed **Ethernet**.
Per quanto riguarda il primo noi abbiamo piena compatibilità con CAN 2.0 Part A e 2.0 Part B, Bit Rates massimo fino a 1Mbit/s ed otto mailboxes object oriented con le seguenti proprietà:

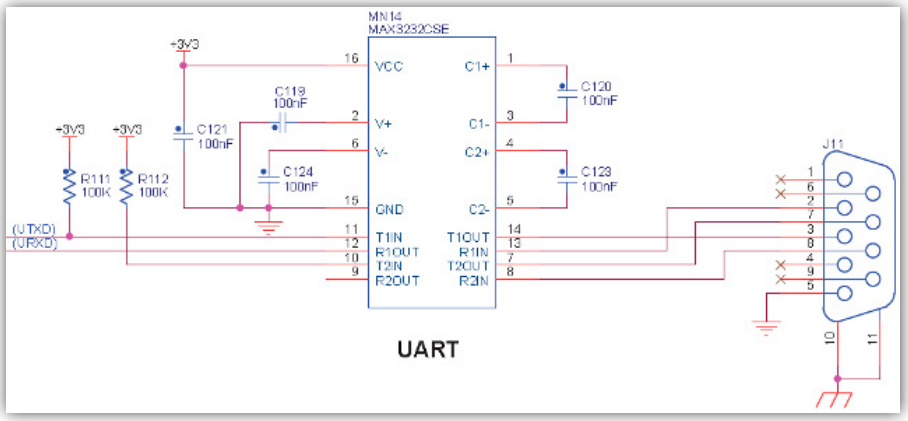
- programmabilità secondo specifiche per ciascun messaggio;
- identificatore indipendente a 29-bit e mask defined per ciascuna mailboxe;
- utilizzo del CAN_SIZE_COUNTER-bit Timestamp su messaggi trasmessi o ricevuti;
- concatenazione hardware dei bitfields ID.

Il timer interno è a 16 bit e può essere utilizzato per Timestamping e sincronizzazione di rete. È anche possibile effettuare la gestione della priorità delle trasmissioni. Sono supportate le modalità autobaud e listening.
Anche in questo caso è possibile prevedere modalità di funzionamento Low Power e di wake-up programmabile comandato dall'attività sul bus o dalla specifica applicazione.
Per quanto riguarda, invece, il controller **Ethernet MAC** (EMAC), abbiamo la più completa compatibilità con uno standard IEEE 802.3 e l'operatività a 10/100 Mbit/s.
Le operazioni che avvengono su questo canale sono di tipo full duplex ma, volendo, possono anche essere half-duplex.
È garantito il supporto per il Promiscuous Mode, in cui tutti i frame ricevuti validi vengono copiati in memoria. Il controllo del flusso half duplex viene garantito forzando le collisioni tra frame in arrivo; quello full duplex, invece, tramite il riconoscimento dai frame di pausa in arrivo.
Viene anche garantito il supporto per il VLAN tagging 802.1Q grazie al riconoscimento delle comunicazioni in arrivo.
Sono sopportati anche Jumbo frames di dimensione fino a 10,240 bytes.
Altra caratteristica interessante è il generatore di numeri casuali (**TRNG**), un dispositivo che ha passato i test della "NIST Special Publication 800-22" nonché quelli "Diehard Random". Ogni 84 cicli di clock, il dispositivo propone un numero casuale rappresentato con una profondità di 32 bit. Per che cosa si può utilizzare? Magari per implementare meccanismi di sicurezza ma potete usarlo per qualsiasi applicazione che lo

richieda.

UART

Lo standard di cui parliamo adesso è piuttosto conosciuto e ci stiamo avvicinando a grandi passi al momento in cui lo utilizzeremo. Si tratta dello Universal Asynchronous Receiver Transmitter, il quale utilizza una UART a 2 pin che può essere sfruttato per effettuare comunicazioni ed offre un ottimo supporto per trasmissioni on-site. Lo standard è ancora molto attuale, sebbene sia nato diverso tempo fa. Inoltre, il suo utilizzo, insieme con periferiche a controller DMA (PDC), permette la trasmissione a pacchetto con tempi ridotti al minimo nell'esecuzione delle istruzioni. I due pin di cui parlavamo prima sono RX e TX che possono essere collegati al connettore DB9 come in figura.



UART

USART

Lo standard Universal Synchronous Asynchronous Receiver Transceiver, meglio noto con l'acronimo USART, permette una comunicazione di tipo full-duplex seriale asincrona sincrona. Il formato dei dati è molto ben personalizzabile utilizzando una serie di parametri come la lunghezza, la parità oppure il numero di stop bit e così via. Questo garantisce il più ampio e largo supporto per la maggior parte degli standard esistenti.

Il ricevitore implementa il controllo di parità per gestire gli errori ma anche il framing error e l'overrun error detection.

Il time-out del ricevitore abilita l'handling per la lunghezza delle variabili ed il timeguard del trasmettitore facilita le comunicazioni con dispositivi slow remote. Le comunicazioni multidrop sono anche supportate attraverso l'handling dei bit di indirizzo sia in ricezione sia in trasmissione.

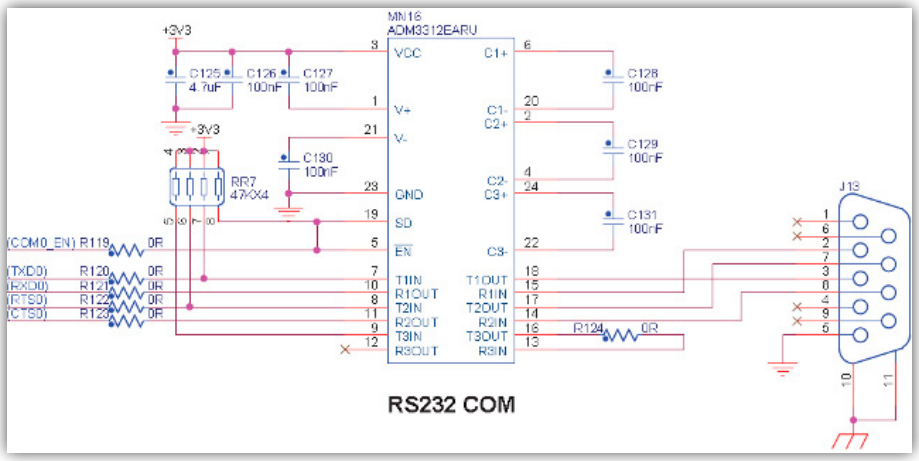
Le principali modalità di test della USART sono, come accennato in precedenza: remote loopback, local loopback e automatic echo.

La USART supporta specifiche modalità operative grazie ad interfacce tipo LIN, RS495 ma anche bus SPI.

La funzionalità di handshaking permette un controllo del flusso out-of-band attraverso gestione automatizzata dei pin RTS e CTS.

RS232

Il SAM3X-EK si può connettere al bus USART0 (comprendendo TXD, RXD, ed i segnali di controllo) tramite connettore DB9 maschio attraverso il transceiver RS232 MN16 in figura.



RS232 COM

Ed a proposito di seriale, basta dare uno sguardo al datasheet del componente per trovare, con relativa facilità, quello che stiamo cercando ovvero il dettaglio, nella Signal Description List, di come lavorare su interfaccia seriale. Vediamolo brevemente in tabella:

I/O Line	Peripheral A	Peripheral B	Extra Function	EK Usage	Device
PA8	URXD	PWMH0	WKUP4	URXD	UART
PA9	UTXD	PWMH3	-	UTXD	UART

TOOLS

Veniamo, ora, alla parte più "applicativa" di questo articolo, quella in cui cerchiamo di capire come usare il microcontrollore.

Emulatore: Atmel SAM-ICE

Si tratta dell'emulatore JTAG per microcontrollori basati su Atmel ARM® core. ICE è stato progettato per SAMA5, SAM3, SAM4, SAM7 and SAM9 con supporto per Thumb®. Supporta una velocità di download non superiore a 720K Bytes/s mentre perJTAG la frequenza (massima) supportata è 12 MHZ.

Evaluation Kit: SAM3X-EK

L'evaluation Kit (progettato per dispositivi SAM3X e SAM3A) è una soluzione piuttosto rapida ed efficace per farvi un'idea delle potenzialità di questa MCU. Se siete a caccia di informazioni ma non vi accontentate di ciò che leggete, si tratta certamente di un'ottima possibilità.

Programmer: Atmel SAM-BA In-system Programmer

SAM-BA® è una soluzione software ISP dedicata alla serie di microcontrollori già elencati che fornisce un set di tool per la programmazione molto versatile ed utile. Questo, così come altre features e debug tool, permetterà un controllo ed una velocità di programmazione notevoli.

Software libraries: Atmel QTouch Library

Ci siamo, nel recente passato, interessati a queste librerie citandole. Si tratta di un set di istruzioni pre-compilato che supporta i metodi di acquisizione QTouch e QMatrix. All'interno delle librerie sono contenuti alcuni progetti di esempio che prevedono, per esempio, l'uso dell'ATtiny88 MCU con il pannello QTOUCH8 oppure anche una scheda basata sull'MCU ATmega324PA con QMATRIX8x8. I progetti sono disponibili sia per compilatore IAR sia per GCC ma gli eseguibili ed i file HEX sono comunque inclusi per una più rapida consultazione.

Software libraries: Atmel Software Framework

Si tratta della "Source code library" per Atmel® megaAVR®, AVR® XMEGA®, AVR UC3 e micro basati su Atmel ARM Cortex-M. Il framework, denominato **Atmel® Software Framework** (**ASF**), non è altro che una libreria software che fornisce una serie di soluzioni software per le MCU Atmel flash, megaAVR, AVR XMEGA, AVR UC3 e per i dispositivi SAM. Tra i vantaggi abbiamo:

- semplificazione dell'uso dei microcontrollori (perchè il livello di astrazione sale);
- ASF è progettato per essere utilizzato in tutte le fasi del progetto: valutazione, prototipazione, progetto (vero e proprio) e produzione;
- è integrato all'interno dell'Atmel Studio

IDE con una GUI oppure è disponibile in versione standalone per compilatori GCC e IAR;
- è scaricabile gratuitamente.

Software tools: Atmel Studio

Atmel® Studio è un ambiente di sviluppo creato al fine di effettuare progettazione e debug in applicazioni embedded su Atmel ARM® ed AVR®.

Software tools: QTouch Studio 4.3.1

L'**Atmel QTouch Studio** è un'applicazione per PC che permette la visualizzazione real-time del data stream di debug delle librerie QTouch proveniente dalla scheda di test o in sviluppo. Alcune soluzioni sono:

- QT600
- SAM3N-EK
- SAM3S-EK
- AVRTS2080A
- AVRTS2080B
- QTouch Xplained

Il software funziona, stando a quanto dichiarato dal produttore, solo su piattaforma Windows.
Per approfondimenti, ulteriori dettagli o altre informazioni potete consultare il sito intenet del prodotto.

CONCLUSIONI

Bene, per oggi direi che possiamo fermarci qui. Ora abbiamo un quadro piuttosto completo della scheda: sappiamo com'è fatta, sappiamo cosa ci possiamo fare ed abbiamo capito più o meno come fare ad interfacciarci e dialogare con essa. La prossima volta vedremo di cominciare qualche esperimento con la scheda per scoprirne realmente le possibilità mettendola sotto test sul campo. Alla prossima.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-due-tutorial-atmel-sam3x8e-arm-cortex-m3-cpu

Arduino DUE & Eclipse: accoppiata vincente

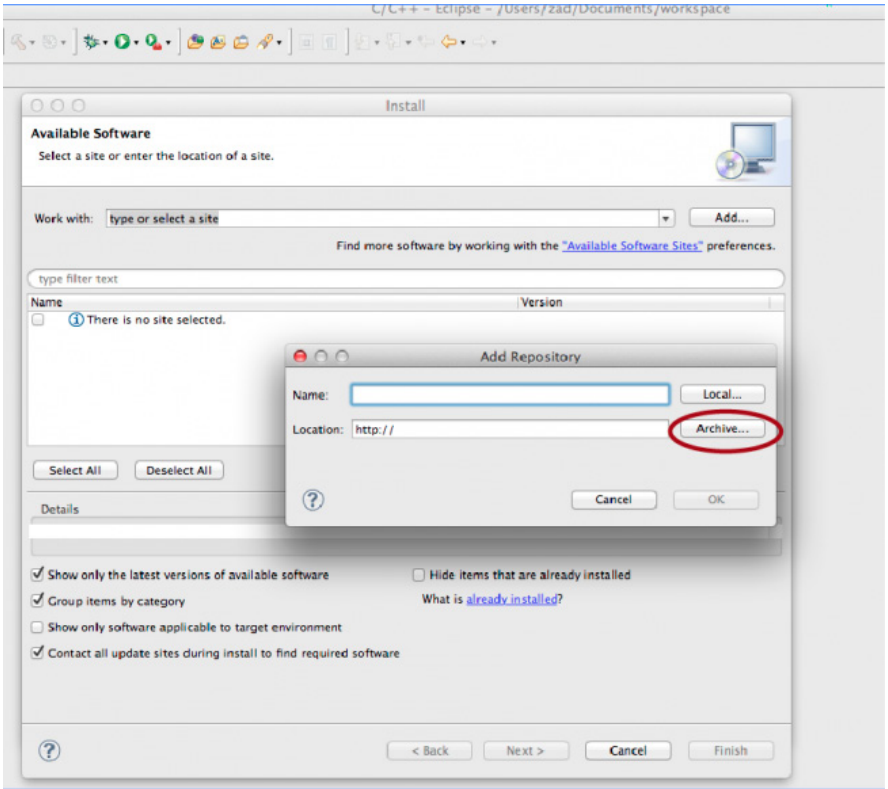
di zad

Tutto quello che si vorrebbe conoscere riguardo l'Arduino DUE puo' essere trovato nell'interessante articolo di Piero Boccadoro "Arduino DUE prestazioni a confronto". Per quanto riguarda il review4U, ho deciso di evitarvi il lampeggio di un led, ma bensi di affrontare un argomento piu' stimolante e utile per progetti corposi quale lo sviluppo su Arduino DUE tramite Eclipse. Questo argomento e' stato affrontato e risolto in questi giorni sul forum dell'arduino con la mia personale partecipazione.
Basta chiacchiere e iniziamo con gli step necessari.

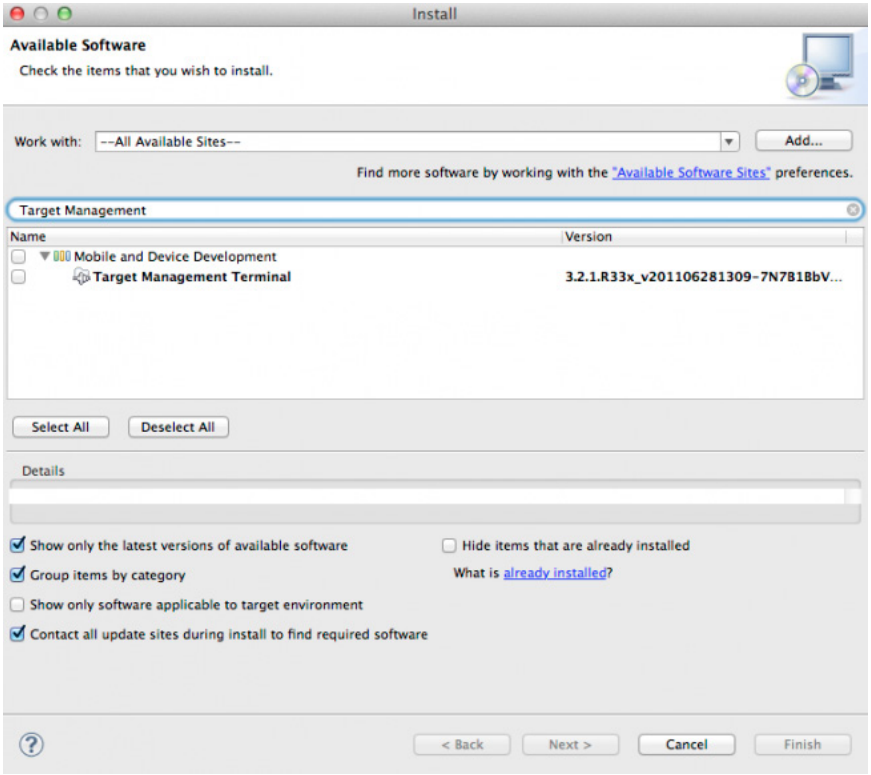
1) DOWNLOAD E INSTALLAZIONE DEI COMPONENTI

Al fine di poter creare un progetto per l'Arduino DUE e' necessario installare i seguenti software nell'ambiente di sviluppo.

- **Arduino DUE IDE** : da cui verranno prese le librerie del progetto. Scompattare il contenuto e appuntarsi il percorso assoluto, ad esempio "C:\arduino-1.5.1r2" o "/Applications/Arduino.app", quindi aggiungere a tale percoso: in Windows: hardware\tools\g++_arm_none_eabi\bin in OSX: Contents/Resources/Java/hardware/tools/g++_arm_none_eabi/bin/ e renderlo disponibile nel path di esecuzione.
- **Eclipse IDE for C/C++ Developers a 32 bits** (la versione a 64 bits sembra non sia compatibile con la RXTX plug-in), installare utilizzando il semplice wizard.
- **GNU ARM Eclipse Plug-in contenente la cross-chain**: Scaricate la versione corrente, che in questo momento è la versione 0.5.4. L'installazione può essere fatta usando il gestore plug-in di eclipse: Eclipse -> Install New Software -> Add ->Archive , indicando lo zip scaricato. Si noti che non utilizzeremo la toolchain fornita dalla plug-in ma quella dell'IDE arduino, l'installazione e' necessaria per avere tutte le opzioni per la cross compilazione in un formato "user friendly".



- **La plug-in RXTX** deve essere installata utilizzando il gestore plug-in nel seguente modo: Scrivendo “RXTX” nel campo Name e il seguente url http://rxtx.qbang.org/eclipse/ nel campo location. Quindi spuntare la versione 2.1.7-r4 e procedere con l’installazione
- **Target Management Terminal Plug-in**. Questa puo’ essere trovata tramite Eclipse->Install New Software... Selezionare in “work with” “all site” e in “Details” scrivere “Target Terminal” dopo qualche secondo vi apparira’ la plugin, selezionare ed installare.



2) PREPARAZIONE DELL’ENVIRONMENT

A questo punto siamo pronti per preparare l’ambiente di compilazione.

CREARE IL PROGETTO ARDUINOLIB2RARY IN ECLIPSE:

Aperto Eclipse, creare un nuovo progetto File->New->C++ Project e specificare come nome del progetto: ArduinoLib2. Disabilitare l’opzione “Use default location” tramite l’apposito checkbox e indicare il percorso del progetto usando il bottone “Browse” SOLO nel caso si volesse usare una directory diversa da quella porposta da eclipse: “c:\Documents\workspace\” come “Project type” specificare “ARM Cross Target Static Library/Empty Project”, mentre per la “Toolchains” selezionare “ARM Windows GCC (Sourcery G++ Lite)” per OSX “ARM OSX GCC (Sourcery G++ Lite)”, quindi terminare premendo “Finish”.

Da qui in avanti i percorsi verranno specificati secondo OSX, essendo l’OS su cui sviluppo.

Aprire le proprieta’ del progetto, click destro sulla cartella nominata ArduinoLib2 nel tab “Project Explorer”.

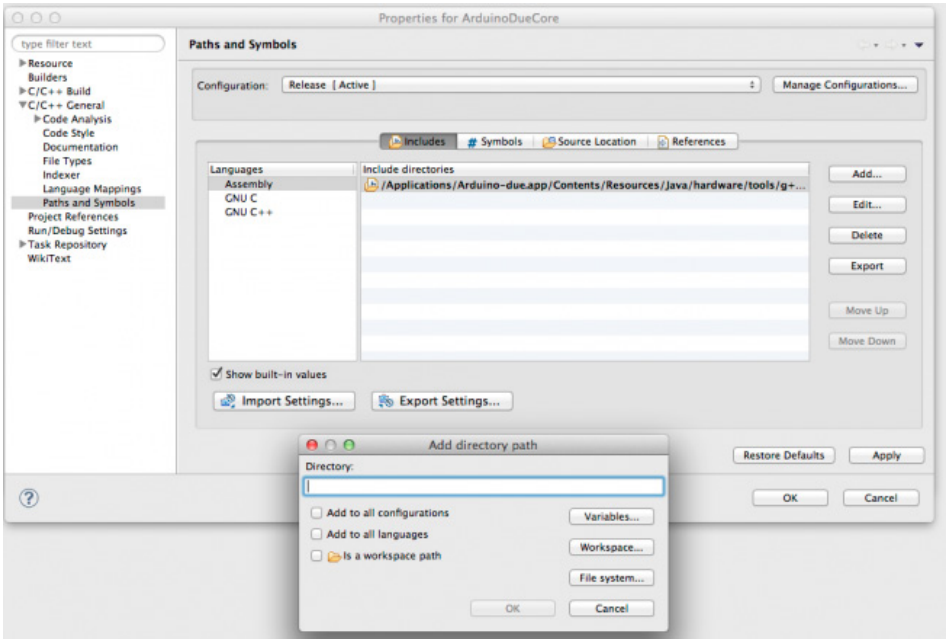
Dobbiamo ora verificare che in “C/C++ General/Paths and Symbols” precisamente nel tab “Includes” e in “Include directories” sia presente per tutti e tre i linguaggi “ Assembli, GNU C e GNU C++, il percorso reso disponibile nel PATH durante l’installazione dell’Arduino IDE. Per essere chiari mettiamo un esempio :

Nel caso di OSX il percorso che dovra’ apparire e’

```
/Applications/Arduino.app/Contents/Resources/
Java/hardware/tools/...
...g++_arm_none_eabi/arm-none-eabi/include/
```

Si noti come la directory finale non sia “bin”, ma “include”. Nel caso tale percorso non sia presente, sara’ possibile includerlo manualmente

tramite il pulsante “add...” e flaggando “add to all languages”.



Aggiungere nello stesso modo anche i seguenti percorsi:

```
/Applications/Arduino.app/Contents/Resources/
Java/hardware/arduino/sam/...
...system/libsam/
/Applications/Arduino.app/Contents/Resources/
Java/hardware/arduino/sam/...
...system/CMSIS/CMSIS/Include/
/Applications/Arduino.app/Contents/Resources/
Java/hardware/arduino/sam/...
...system/CMSIS/Device/ATMEL/
```

Nota: in Windows le directory “Contents\Resources\Java\” non sono presenti, ma la cartella “hardware” si trova direttamente nella root dell’IDE.

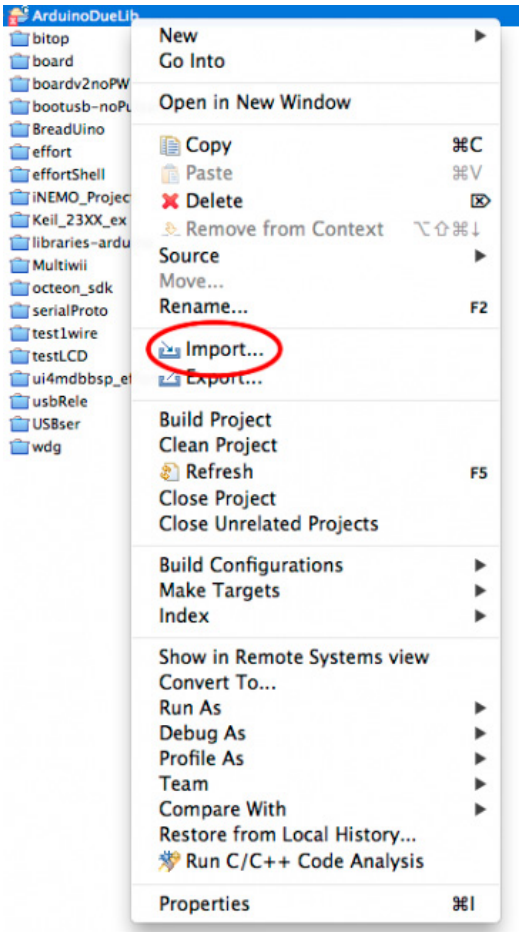
Passando ora a “Symbols” aggiungiamo i seguenti simboli per tutti i linguaggi:

```
printf=iprintf
F_CPU=84000000L
ARDUINO=151
__SAM3X8E__
USB_PID=0x003e
USB_VID=0x2341
USBCON
```

Per inserire un simbolo, ad esempio “ARDUINO=151”, mettere in “Name” ARDUINO ed in “Value” 151. Per i simboli senza valore valorizzare solo il campo “Name”.

Nota: __SAM3X8E__ contiene due underscore per parte.

Adesso che il progetto e’ pronto, non resta che copiare i file di core dell’arduinoDue, per prima cosa, nel “Project Explorer” di Eclipse selezionare la cartella di progetto “ArduinoLib2” e premendo con il destro selezionare “import”.



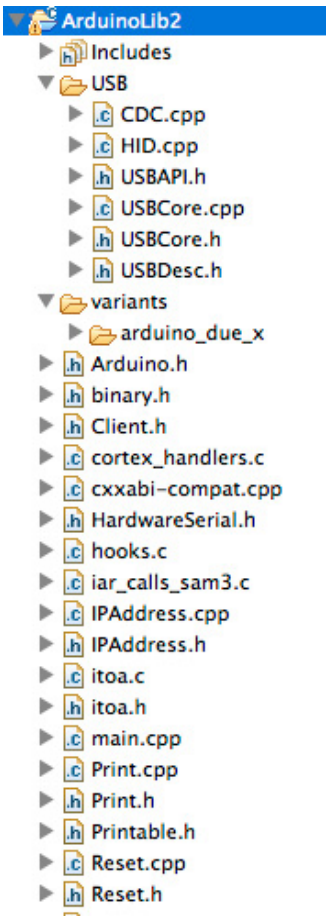
Selezionere quindi General->FileSystem, premere “Browse” e selezionare la cartella arduino presente nel seguente path:

```
/Applications/Arduino.app/Contents/Resources/
Java/hardware/arduino/sam/...
...cores/arduino
```

Non flaggare “Create top Level directory” e premere ok.
Stessa cosa per la directory

```
/Applications/Arduino.app/Contents/Resources/
Java/hardware/arduino/sam/...
...variants/
```

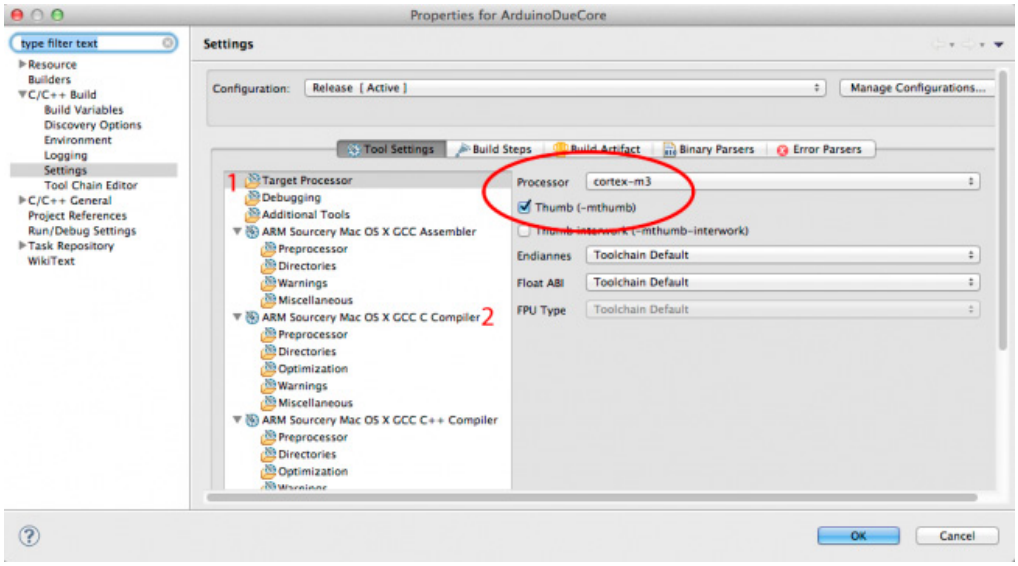
Questa volta flaggando “Create top Level directory”
A questo punto selezionata la directory di progetto, premere F5 ed espandere l’alberatura, il risultato sara’ il seguente:



Nota: Non sono presenti tutti i file per motivi di spazio, l’importante e’ capire l’albertura del progetto, tutte le directory importate sono comunque visibili.

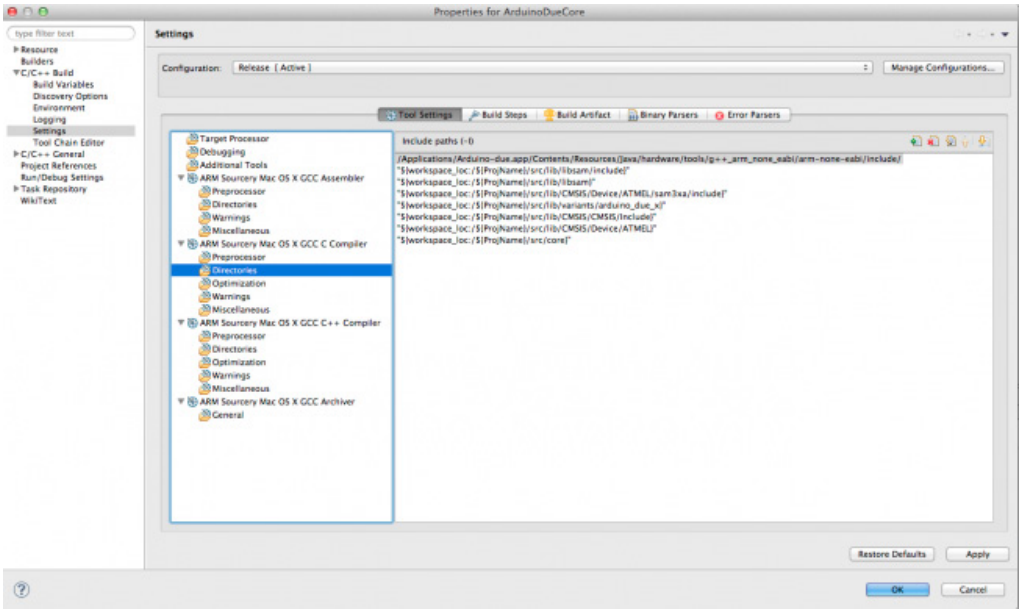
CONFIGURAZIONE DEL PROGETTO PER LA COMPILAZIONE:

E’ necessario configurare le opzioni e ottimizzazioni del cross-compilatore: right-click sul progetto e selezionare “property”, quindi selezionare C/C++ Build/ Settings sulla destra e nel “Tool Settings” tab impostare come “Target Processor “ il “cortex-m3” abilitando le istruzioni “Thumb” tramite il checkbox (1).



Per il “Debugging” selezionare “Default –g” e come “Debug format” gdb.
Per una compilazione senza errori, configurare il Compilatore C passando per i seguenti punti:

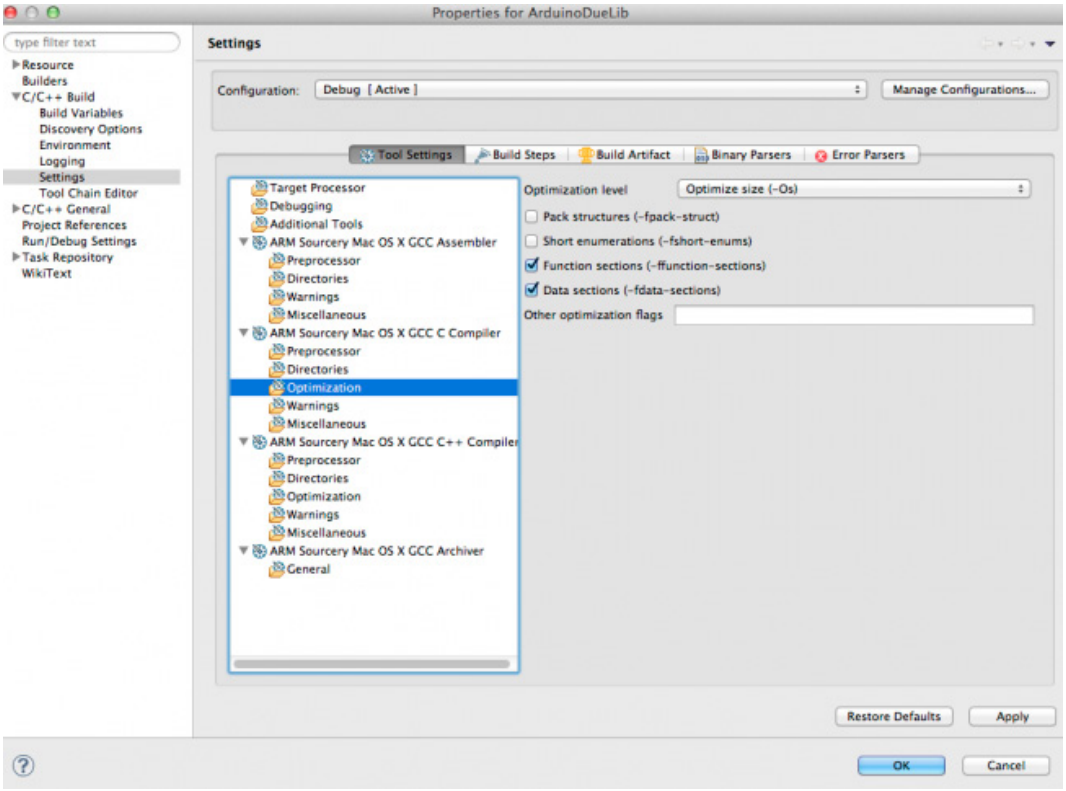
- “**ARM Sourcery Mac OSX GCC C Compiler/Directories”**
  (2), agginungere, selezionando “workspace” come riferimento, i seguenti path:
  “variants\arduino_due_x”
  “USB”
  e la cartella di progetto stessa,



Come “Language standard” selezionare “Compiler defaults”. Non selezionare nessuna checkbox e assicurarsi che in “Other Flags” sia presente:

```
-c --param max-inline-insns-single=500 –nostdlib
```

- **“ARM Sourcery Mac OSX GCC C Compiler/Optimization”**



- **“ARM Sourcery Mac OSX GCC C Compiler/Warnings”**

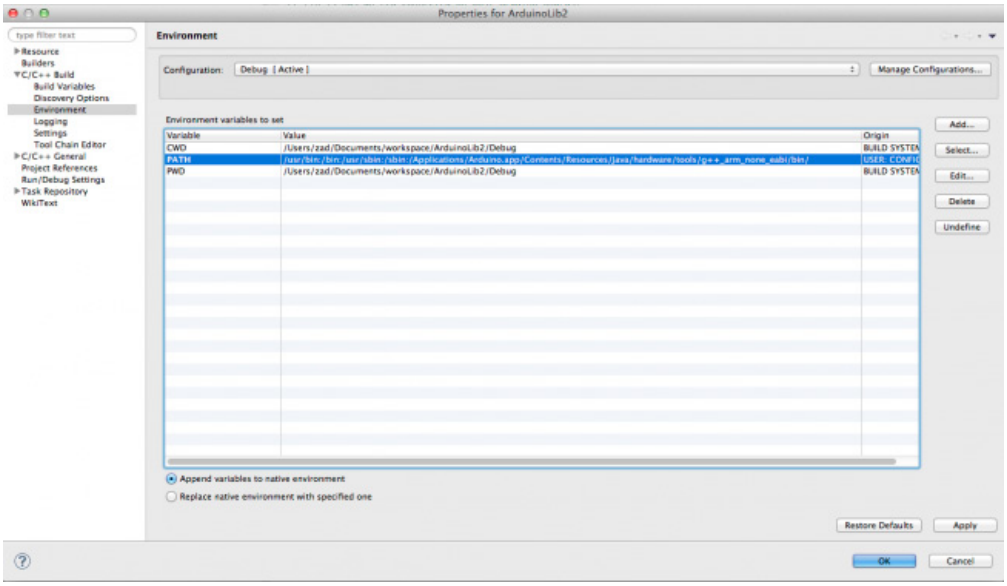
  Selezionare –Wall.

- **“ARM Sourcery Mac OSX GCC C Compiler/Miscellaneous”**

Le stesse configurazioni vanno applicate a **“ARM Sourcery Mac OSX GCC C++ Compiler”**, ad eccezione di “ARM Sourcery Mac OSX GCC C++ Compiler/Miscellaneous” dove bisognera’ selezionare inoltre:

“Do not uses exceptions” e “Do not use RTTI”

Per OSX infine e’ necessario aggiungere in “environment” il percorso della toolchain, vedi riga blu nella figura sottostante, al momento dell’inserimento accertarsi che “Append variables to native environment” sia flaggato.



Finalmente possiamo compilare le librerie di base usando il il tasto build (simbolo del martello), al termine della compilazione avremo sotto Archives la libreria di core.

3) PRIMA APPLICAZIONE

CREARE UN PROGRAMMA PER ARDUINO DUE: BLINK

Le librerie del core verranno da qui in avanti linkate a tutti i nuovi progetti.

Iniziamo dal tipico led, e' vero, avevo esordito dicendo di saltare il blink project, ma quello sara' il template da cui tutti i progetti piu' complessi potranno partire. Bastera' infatti duplicare il progetto blink per potere avere, in un batter d'occhio, un ambiente testato e pronto all'uso.

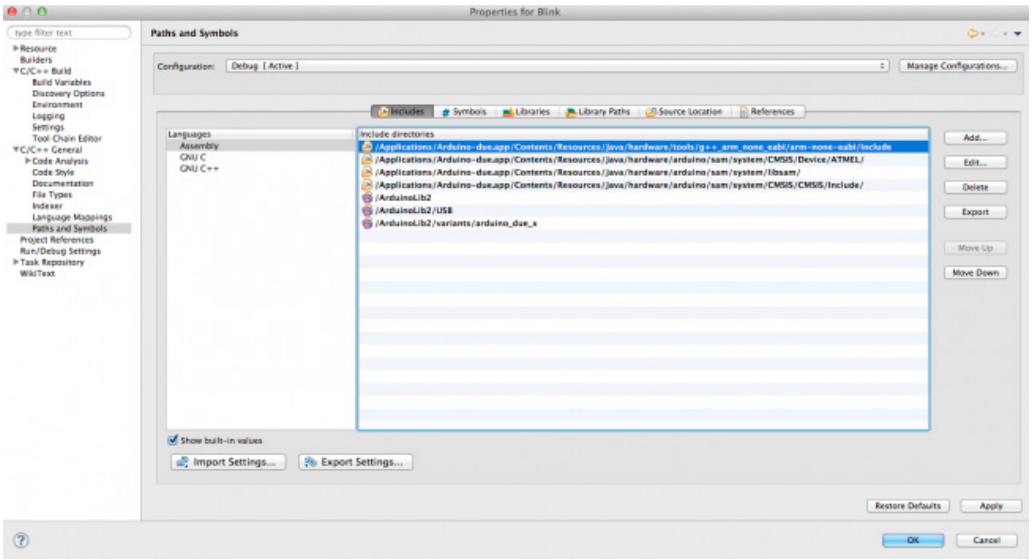
Iniziamo con il creare un nuovo progetto C++, questa volta invece di selezionare "static library" selezioneremo "ARM Cross Target Application" con Toolchain "Sourcery G++ Lite", nominiamo il progetto Blink e premiamo "Finish".

La maggior parte delle configurazioni sono uguali a quelle della libreria sopra creata, vediamole nuovamente tutte evidenziando cosa cambiare: apriamo le proprieta' del progetto:

In "C/C++ General/Paths and Symbols" tutto identico alla configurazione precedente, aggiungere in Path, selezionando "Workspace" anche:

```
/<NomeProjLib>/
/<NomeProjLib>/USB/
/<NomeProjLib>/variants/arduino_due_x
```

Al posto di <NomeProjLib> dovremo inserire il nome di progetto delle librerie compilate precedentemente, nel mio caso ArduinoLib2.



C/C++ Build/ Settings:

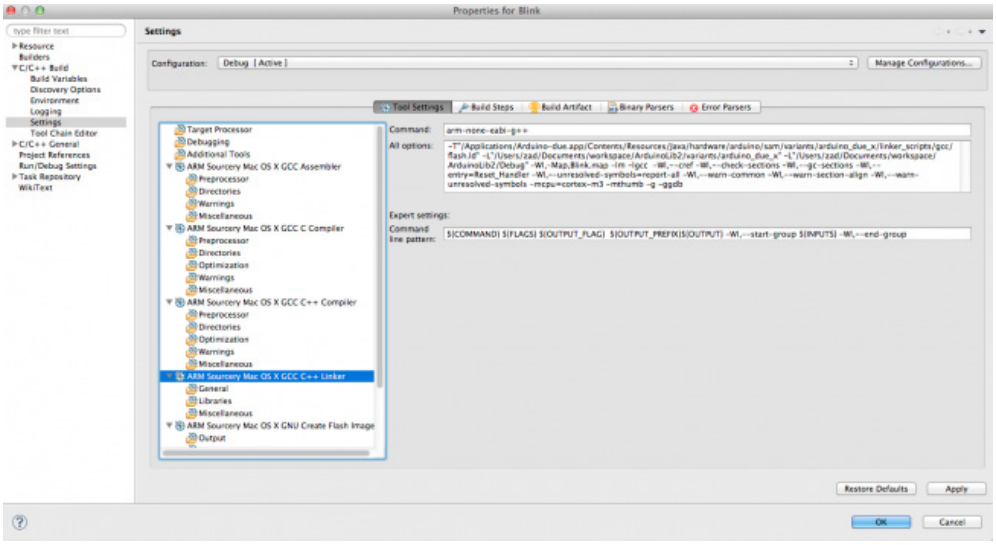
“Target Processor” come per le librerie.

“Debugging” selezionare “Default –g” e come “Debug format” gdb.

Per quanto riguarda la voce “Additional Tools” selezionare tutte e tre le voci: ”Create flash image” , “Create Extended Listings” e “Print Size”

Passiamo alla configurazione del compilatore C:

- **“ARM Sourcery Mac OSX GCC C Compiler/Directories”**, agginungere, selezionando “workspace” soltanto la cartella di progetto.
- **“ARM Sourcery Mac OSX GCC C Compiler/Optimization”**, identico alle librerie.



- **“ARM Sourcery Mac OSX GCC C Compiler/Warnings”**
  Selezionare –Wall.
- **“ARM Sourcery Mac OSX GCC C Compiler/Miscellaneous”**
  Come “Language standard” selezionare “Compiler defaults”. Non selezionare nessuna checkbox e assicurarsi che in “Other Flags” sia presente:

```
-c --param max-inline-insns-single=500 –nostdlib
```

Le stesse configurazioni vanno applicate a **“ARM Sourcery Mac OSX GCC C++ Compiler”**, ad eccezione di **“ARM Sourcery Mac OSX GCC C++ Compiler/Miscellaneous”** dove bisognera’ selezionare:

“Do not uses exceptions” e “Do not use RTTI”

La vera differenza viene nella configurazione del Linker:

Partiamo selezionando l’elemento root del linker **“ARM Sourcery Mac OSX GCC C++ Linker”** e in “Command Line Pattern” copiamo la seguente riga:

```
${COMMAND} ${FLAGS} ${OUTPUT_FLAG}
${OUTPUT_PREFIX}${OUTPUT}
 -Wl,--start-group ${INPUTS} -Wl,--end-group
```

In “General” nel campo “Script File (-T)” mettiamo il percorso che punta al file di configurazione flash.ld.

In ambiente OSX lo troviamo in:

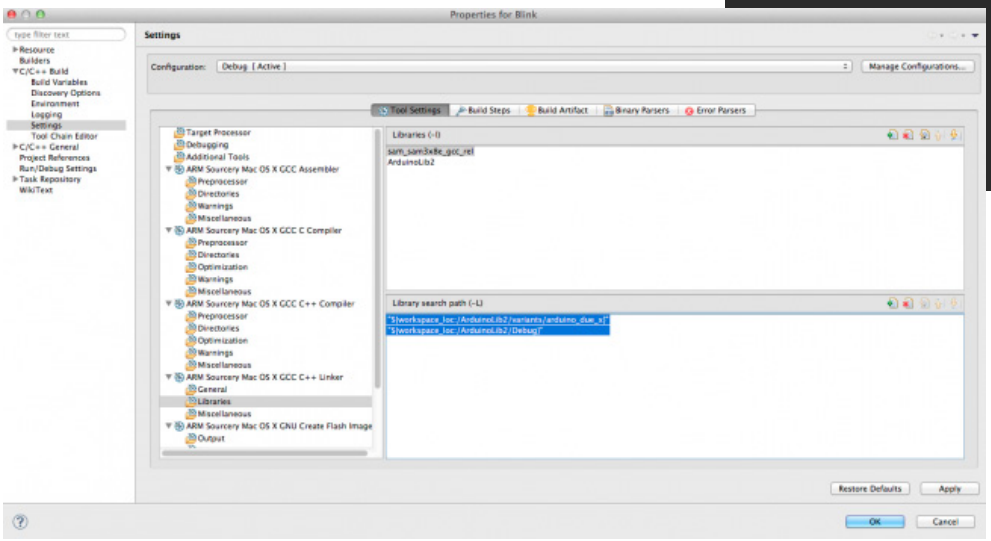
```
/Applications/Arduino.app/Contents/Resources/
Java/hardware/arduino/sam/...

...variants/arduino_due_x/linker_scripts/gcc/flash.
ld
```

Questo file specifica in fase di linking la dimensione e le aree di memoria del processore (flash) e la definizione delle sezioni del programma : .bss .data .text etc.

In “Libraries” specifichiamo “contro” quali librerie il nostro programma deve essere linkato; qui inseriremo per prima cosa la libreria compilata precedentemente e poi una libreria precompilata che viene fornita con ArduinoIDE.

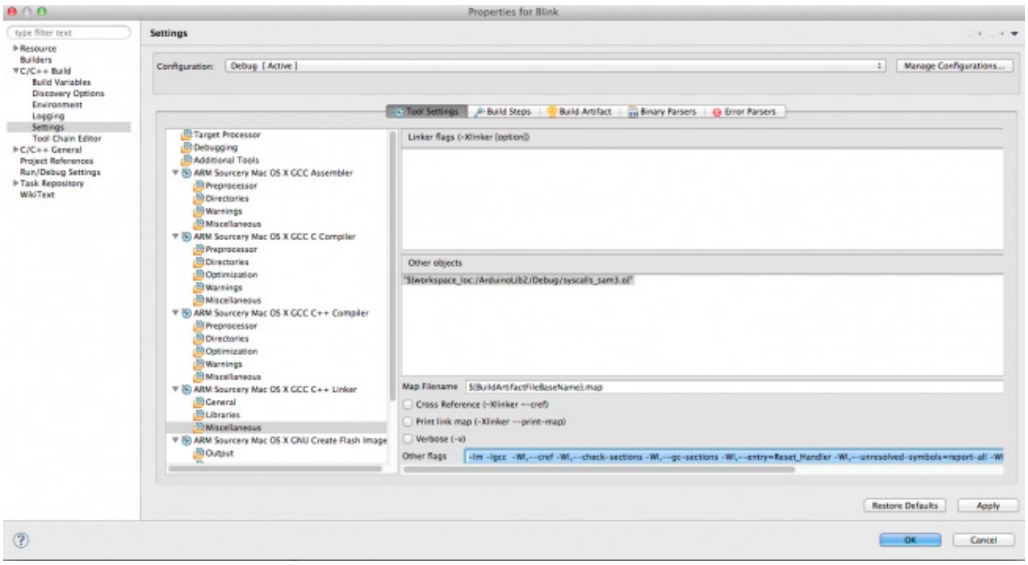
Nell'area "Libraries (-l)" aggiungere: ArduinoLib2 e sam_sam3x8e_gcc_rel.
In "Library Search Path (-L) aggiungere come workspace:

```
"ArduinoLib2/variants/arduino_due_x"
"ArduinoLib2/Debug"
```



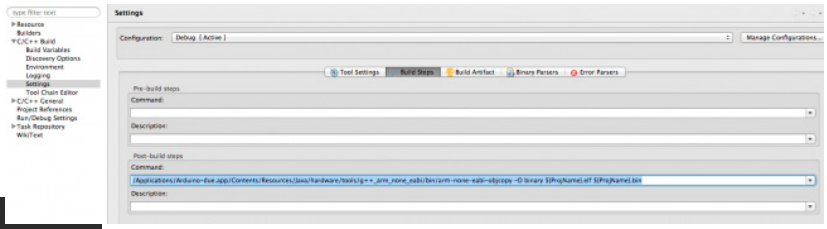
In "Miscellaneous" aggiungere in "Other Objects" il percorso di workspace al file "ArduinoLib2/Debug/syscall_sam2.o" e in "Other Flags" specificare:

```
-lm -lgcc  -Wl,--cref -Wl,--check-sections -Wl,--gc-sections
-Wl,--entry=Reset_Handler -Wl,--unresolved-symbols=report-all
-Wl,--warn-common -Wl,--warn-section-align
-Wl,--warn-unresolved-symbols
```



Passiamo al tab "Build Steps", qui inseriremo l'ultimo passaggio necessario per ottenere il binario da caricare sull'Arduino.
Nella riga command: di "Post-Build Steps" scrivere:

```
/Applications/Arduino.app/Contents/Resources/
                    Java/hardware/tools/...
                    .../g++_arm_none_eabi/bin/
                    arm-none-eabi-objcopy
                    -O binary ${ProjName}.elf
                    ${ProjName}.bin
```



Configurando infine il PATH in C/C++ Build/Environment come fatto per le ArduinoLib2, potremo aggiungere finalmente il file .cpp per la compilazione di "Blink".
Chiudiamo le proprieta' di progetto e creiamo un nuovo file c++, click con il destro sulla cartella di progetto New->source file, nominiamolo Blink.cpp e selezioniamo C++ Source Template.
Non ci resta che aggiungere nel nuovo file il "familiare" codice:

```
#include <Arduino.h>


/*

  Blink

  Turns  a LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.

 */

// Pin 13 has an LED connected on most Arduino boards.

// give it a name:

int led = 13;


// the setup routine runs once when you press reset:

void setup() {
  // initialize the digital pin as an output.
    Serial.begin(115200);
    pinMode(led, OUTPUT);

}
// the loop routine runs over and over again forever:
void loop() {
    Serial.println("Test");
    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
}
```

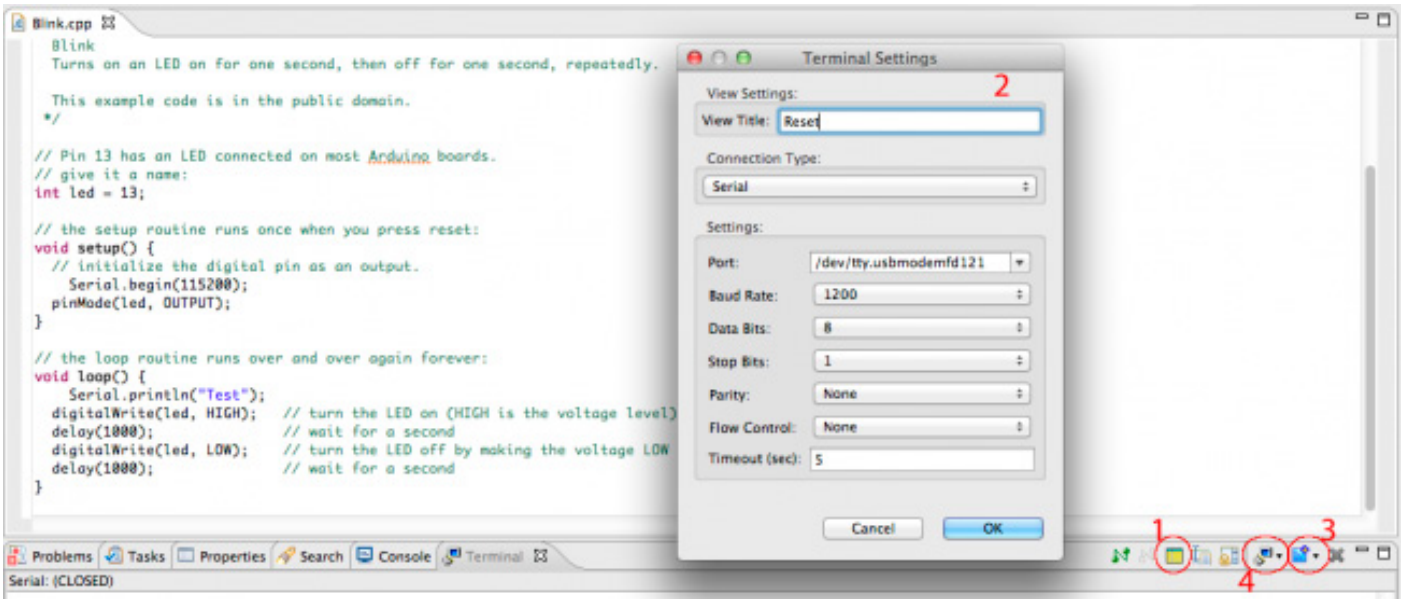
Salviamo e compiliamo! Il binario Blink.bin si trova nella cartella Debug del progetto.

PROCEDURA DI ERASE E UPLOAD DEL "COMPILATO" SUL TARGET

Arduino DUE utilizza il programma precompilato chiamato bossac.exe o semplicemente bossac (Linux e OSX) per effettuare l'upload verso l'hardware.

Fino ad ora sono riuscito a programmare l'Arduino DUE usando la "Programming Port", ovvero la porta mini usb vicina al jack dell'alimentazione. Sulla pagina ufficiale viene riportato che per usare la porta in programmazione e' necessario avviare la procedura di erase della flash. Tale procedura, viene scritto, puo' essere avviata semplicemente aprendo e chiudendo una connessione seriale sulla stessa.

Per quanto riguarda la "Native Port" invece servirebbe aprire una connessione con baud rate di



1200bps per attivare il “soft erase” effettuato dal processore.

elle varie prove ho notato che in realta’ sulla “Programming port” non basta aprire e chiudere una connessione ma bisogna aprirla a 1200bps. Un’alternativa alla procedura software e’ quella manuale, si puo’ infatti attivare la procedura di erase anche premendo per piu’ di un secondo il tasto “erase”, quello sottostante alla serigrafia “COMMUNICATION”.

Per procedere dobbiamo a questo punto creare una configurazione Terminal usando la plugin installata precedentemente.

Bisogna far apparire in eclipse il tab “Terminal” cliccando nel menu’ “Windows”->ShowView->Others... e selezionare “Terminal”

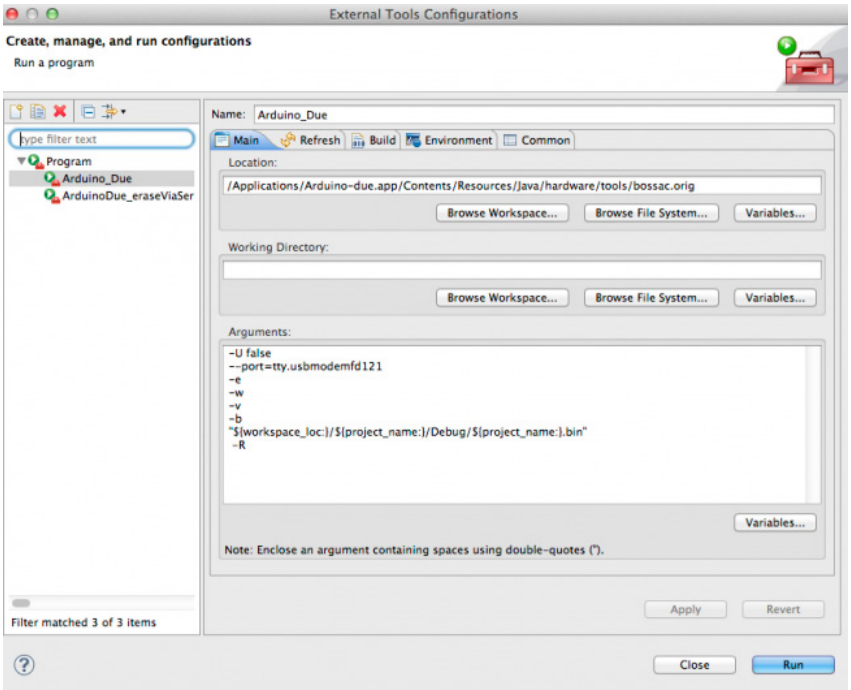
Il tasto (1) permette di creare un nuovo settaggio del terminal, chiamiamolo “Reset” ed impostiamo i valori (2) come in figura. Ovviamente la porta dovra’ corrispondere a quella verso cui e’ collegato l’Arduino DUE.

Il primo tasto verde presente sulla barra per i settaggi serve per collegarsi all’arduino. Il tasto (3) per creare una nuova configurazione e il tasto (4) per passare da una configurazione all’altra.

Bastera’ premere il tasto “connetti” per attivare l’erase dell’Arduino DUE, una volta connessi, apparira’ un pulsante graficamente uguale a quello di connessione, ma di colore rosso, usato per disconnettere la seriale.

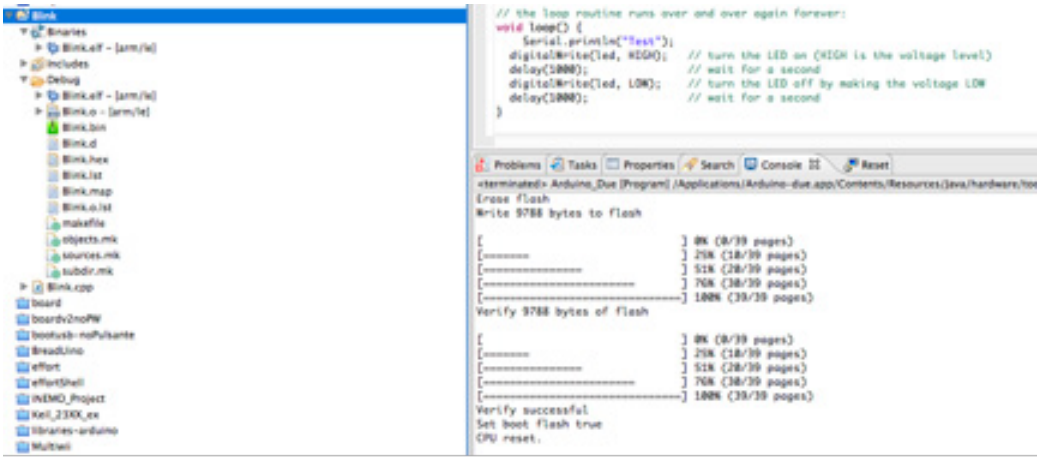
L’ultimo passo necessario e’ la configurazione del bossac per l’upload:

Premere “Run”->”External Tools”->”External Tool Configurations” , selezioniamo “Program” e inseriamo i valori in figura.



"Apply" e poi "close".

Collegato l'Arduino DUE apriamo e chiudiamo un connessione verso l'arduino col Terminal, quindi successivamente premiamo il tasto "Run" (il play con la valigetta marrone" e dopo qualche secondo la nostra board blinkera' con frequenza di 1Hz.



4) USO LIBRERIE AGGIUNTIVE

Per aggiungere una libreria esterna, quale ad esempio "LiquidCrystal", e' un'operazione semplice ed intuitiva.

Selezionando la cartella di progetto, apriamo il menu' contestuale, tasto destro del mouse, e selezioniamo import. Selezionare General->FileSystem e cliccare su browse.

```
/Applications/Arduino.app/Contents/Resources/
Java/Libraries/
```

La libreria "LiquidCrysta" si trova in selezionare la cartella LiquidCrystal, espanderla e flaggare solo LiquidCrystal.cpp LiquidCrystal.h, quindi selezionare "Create top level directory" per mantenere l'alberatura di progetto ordinata e premere ok.

Per includere la libreria, aggiungere nelle proprieta' del sistema-> CC++ Build -> Settings: **"ARM Sourcery Mac OSX GCC C++ Compiler"**, e **"ARM Sourcery Mac OSX GCC C Compiler"** in Directories , Include Paths anche il percorso alla cartella appena importata.

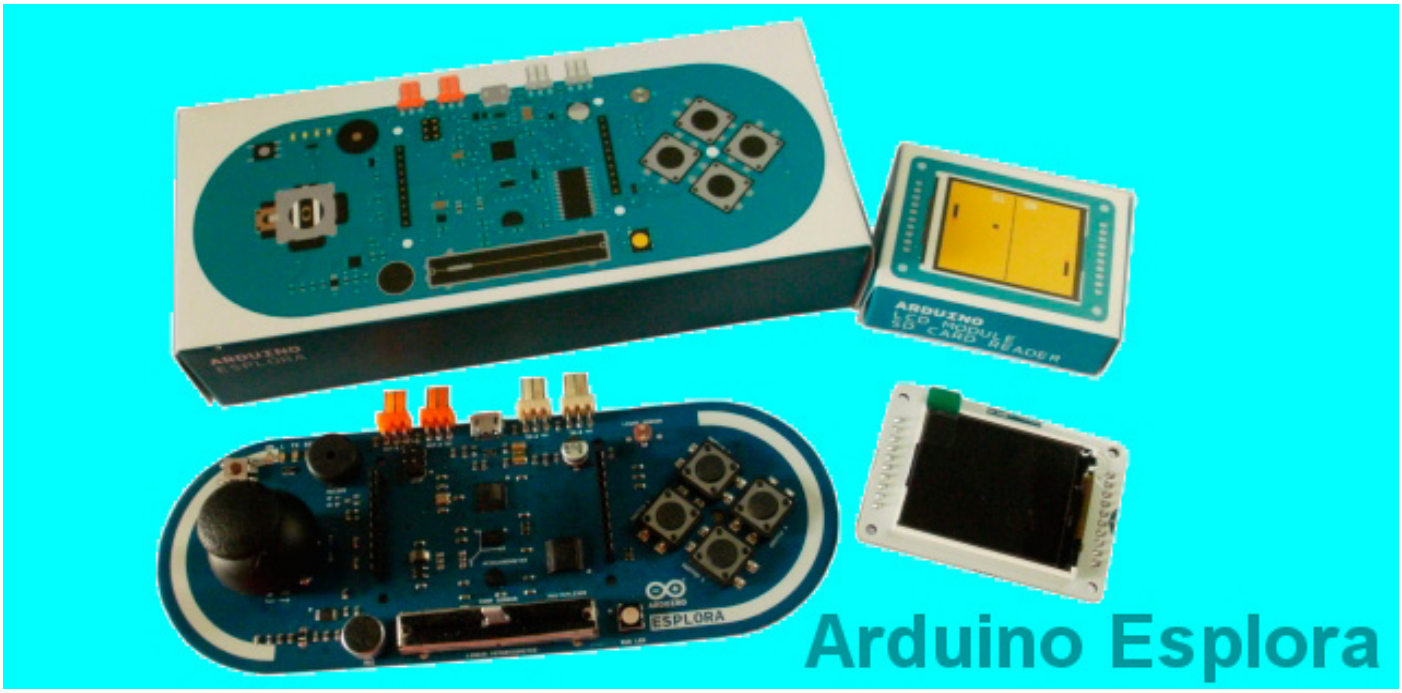
Salvata la configurazione, si potra includere <LiquidCrystal.h> all'interno del file .cpp che deve utilizzarla.

CONCLUSIONI

L'IDE compreso con l'Arduinio Uno e DUE e' sicuramente un valido compagno per tutti gli sviluppatori, nasconde le complicazioni della cross-compilazione per lasciare il programmatore concentrato sull'implementazione del progetto. Cio' nonostante i piu' esigenti potrebbero desiderare maggiori funzionalita' e magari un ambiente piu' famigliare, Eclipse puo' essere una valida alternativa pur richiedendo uno sforzo di configurazione iniziale maggiore.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/affrontiamo-questo-articolo-configurazione-di-eclipse-cross-compilazione-con-larduinodue

Arduino ESPLORA



Scopriamo la nuova scheda Arduino ESPLORA

di adrirobot

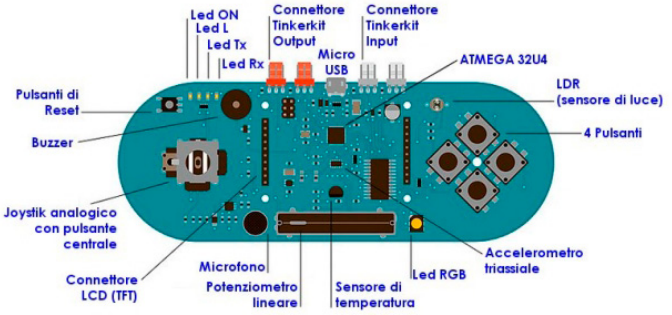
La nuova **Arduino ESPLORA** è una scheda a microcontrollore, le sue dimensioni sono di circa 163x60 mm; nella parte alta, a lato della porta USB, sono presenti dei connettori compatibili Tinkerkit che sporgono leggermente dal profilo della scheda. Per fissarla ad una superficie sono presenti quattro fori.

La scheda ESPLORA si differenzia da tutte le schede Arduino precedenti poiché su di essa sono già montati, di serie, diversi sensori. Essa è, inoltre, dotata di una porta di comunicazione USB, che può essere riconosciuta dal computer come un mouse o una tastiera, oltre ad una porta seriale COM / virtuale (CDC).

In particolare sulla scheda ESPLORA sono presenti i seguenti dispositivi:

- Un joystick analogico a due assi (X e Y) con pulsante centrale.
- 4 pulsanti disposti a rombo - Un potenziometro lineare a cursore.
- Un microfono per rilevare il rumore ambientale.
- Un sensore di luce per la misurazione dell'intensità luminosa.

- Un sensore per la misurazione della temperatura ambiente.
- Un accelerometro triassiale (X, Y e Z).
- Un buzzer per produrre suoni.
- Un LED luminoso tipo RGB con elementi Rosso Verde e Blu.
- 2 Ingressi per collegare i moduli sensore della serie Tinkerkit.
- 2 uscite per collegare i moduli attuatori della serie Tinkerkit.
- Un connettore per display tipo TFT a colori opzionale, dotato di uno slot per scheda SD, o altri dispositivi che utilizzano il protocollo SPI.



Analisi del circuito

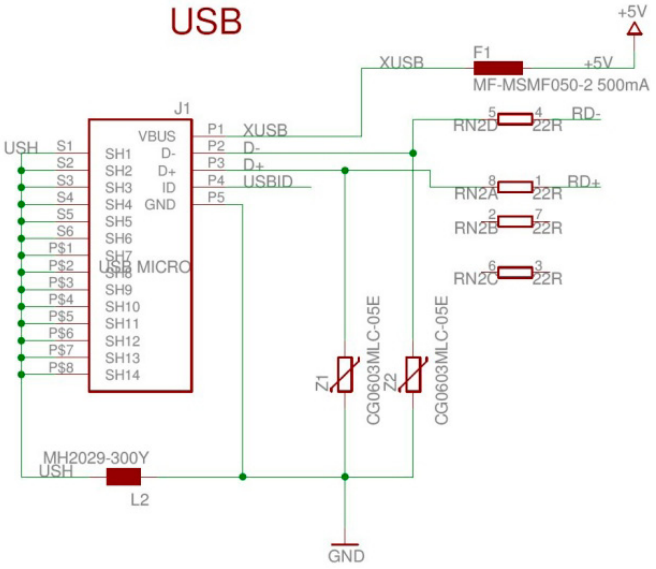
Dal sito Arduino è possibile scaricare lo schema della scheda, che presenta le seguenti caratteristiche:

- Microcontroller **ATMEGA32U4**;
- Tensione di funzionamento 5V;
- Flash Memory 32 KB di cui 4 KB utilizzati dal bootloader;
- SRAM 2.5 KB EEPROM 1 KB;
- Velocità di clock 16 MHz .

Schema elettrico della scheda Arduino Esplora

Alimentazione

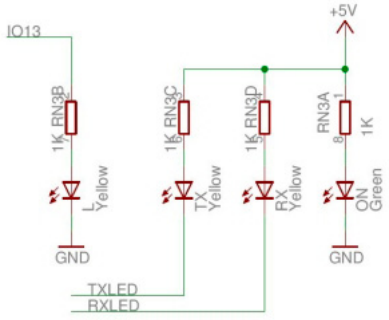
L'alimentazione della scheda è derivata dalla presa USB, e sebbene la maggior parte dei computer fornisca una protezione interna, questa è protetta da un fusibile resettabile che, nel caso di un assorbimento superiore ai 500 mA, interromperà automaticamente la connessione finché non sarà rimosso il corto o il sovraccarico.



Zona processore

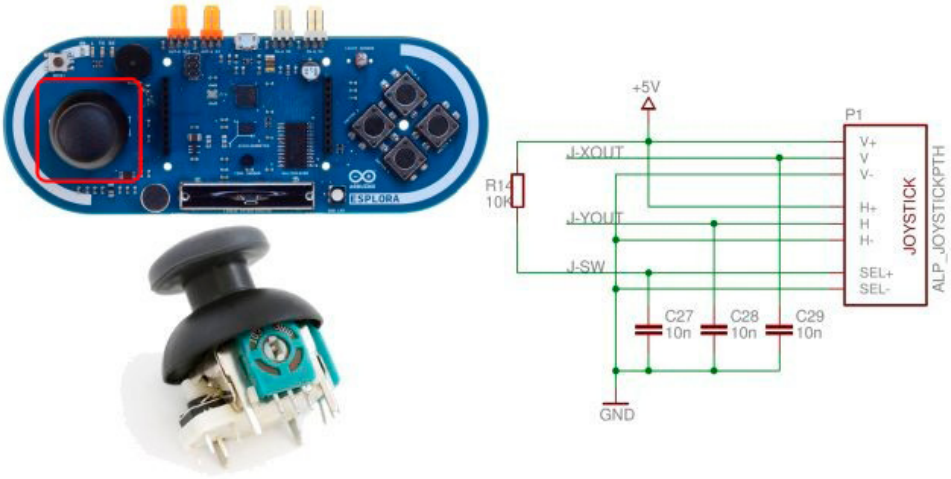
Come detto in precedenza, la scheda utilizza lo stesso processore della Arduino Leonardo, un microcontrollore AVR tipo **ATMEGA32U4**, che opera a una frequenza di 16 MHz: è connesso ad una porta USB ed è in grado di agire come un dispositivo client USB, come un mouse o una tastiera. Nel circuito è presente un pulsante di reset, con cui è possibile riavviare la scheda.

Ci sono quattro LED di stato: **ON [verde]** indica se la scheda è alimentata. **L [giallo]** collegata direttamente al microcontrollore, accessibile attraverso il pin 13. **RX e TX [giallo]** che indicano che i dati sono trasmessi o ricevuti nel corso della comunicazione USB.



Per la comunicazione la scheda ESPLORA dispone già di una serie di servizi per la connessione con un computer, un altro Arduino, o con altri microcontrollori.

L'ATMEGA32U4 appare come una porta COM virtuale per software sul computer. Il chip funziona anche come dispositivo USB secondo lo standard 2.0. L'integrato supporta, inoltre, la comunicazione SPI a cui si può accedere tramite la libreria SPI.



All'interno della memoria è già presente il bootloader che permette di caricare un nuovo codice senza l'uso di un programmatore hardware esterno. Esso comunica utilizzando il protocollo AVR109.

È anche possibile bypassare il bootloader e programmare il microcontrollore attraverso l'**ICSP** (In Circuit Serial Programming) presente sulla scheda.

Joystick analogico con pulsante centrale.

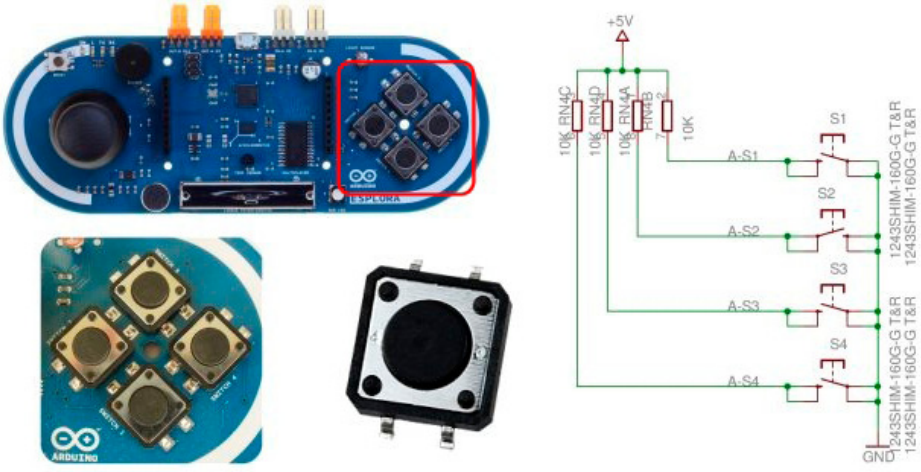
Il joystick analogico è posto alla sinistra della scheda ed è formato da due potenziometri connessi a croce, premendo il joystick si attiva poi un pulsante.

I tre segnali: J-XOUT, J-YOUT, J-SW sono collegati all'integrato IC3 tipo **74HC4067D** che è un multiplexer/demultiplexer a 16 canali. Questo permette di utilizzare il numero totale di sensori disponibili. Ciò significa che un singolo ingresso analogico del microcontrollore è condiviso tra tutti i canali d'ingresso (eccetto l'accelerometro a tre assi).

Quattro ulteriori pin del microcontrollore permettono di scegliere il canale da leggere.

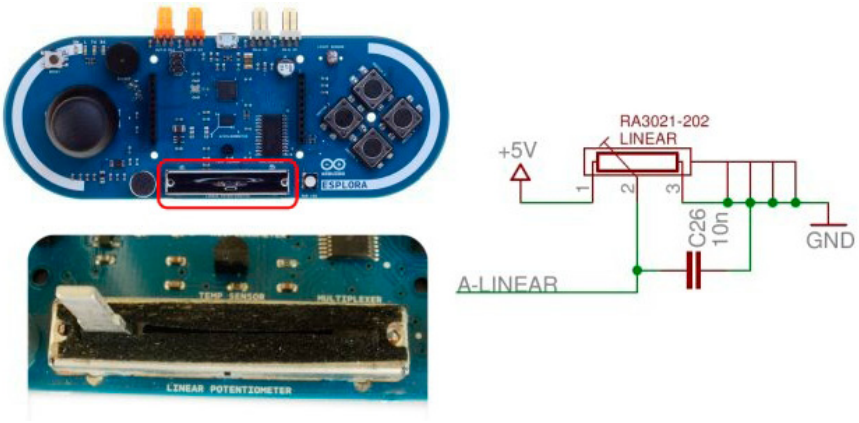
4 pulsanti disposti a rombo.

Sul lato destro della scheda, sono presenti quattro pulsanti disposti a rombo: ognuno ha la propria resistenza di pull up da 10 kohm per mantenere l'uscita a livello alto quando il pulsante non è premuto; come per il Joystick, le 4 uscite A-S1, A-S2, A-S3, A-S4 sono connesse all'integrato IC3.



Un potenziometro lineare a cursore

Nella parte bassa della scheda, è presente un potenziometro lineare da 1 kohm, collegato come partitore resistivo per variare il livello di tensione in uscita sul pin A-LINEAR connesso sempre all'integrato IC3.



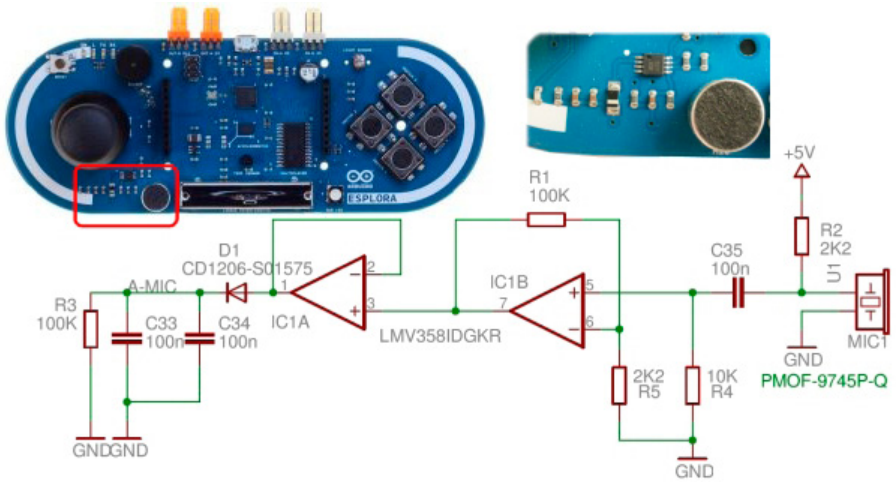
Il valore letto sarà un numero con una risoluzione pari a 10 bit, ed è possibile leggerlo tramite il comando di libreria **Esplora.readSlider()**.

Un microfono per rilevare il rumore ambientale.

La capsula microfonica tipo **PMOF-9745P-Q** è un microfono omnidirezionale a condensatore Electret, ha una frequenza di funzionamento compresa tra i 20 e i 16000 Hz, il massimo valore del livello di pressione pari a 120 dB S.P.L., un rapporto segnale/rumore di 60 dB.

Nel circuito, il segnale in uscita, è amplificato tramite due amplificatori operazionali contenuti all'interno di un integrato tipo **LMV358** del tipo rail-to rail.

Anche in questo caso l'uscita A-MIC è collegata all'integrato IC3, il valore può essere letto tramite il comando di libreria **Esplora.readMicrophone()**.



Sensore di luce per la misurazione dell'intensità luminosa

Per la misurazione dell'intensità luminosa, la scheda ESPLORA ha una fotoresistenza, anche se nello schema e sullo stampato figura un fototransistor.

Le fotoresistenze sono delle particolari resistenze il cui valore dipende dall'intensità e dal colore della luce che le colpisce; in genere sono dei sottili film di solfato di cadmio su un supporto rigido, chiusi in involucri protettivi trasparenti.

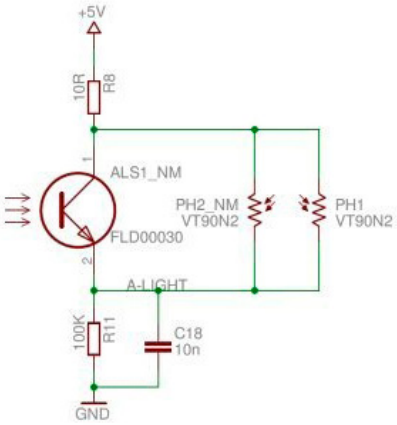
Il modello montato presenta a 10 lux valori di resistenza compresi tra 12 kohm e 36 kohm, e un valore massimo di resistenza al buio di 500 kohm

L'uscita A-LIGHT è collegata all'integrato IC3, il valore può essere letto tramite il comando di libreria **Esplora.readLightSensor()**, che legge l'intensità della luce che colpisce il sensore come un numero a 10 bit. Ciò significa che mapperà tensioni di ingresso comprese tra 0 e 5 volt in valori interi compresi tra 0 e 1023. Questo produce una risoluzione tra le letture di: 5 volt / 1024 unità, 0,0049 volt (4,9 mV) per unità.

Sensore di temperatura per la misurazione della temperatura ambiente.

Per la misurazione della temperatura, la scheda ESPLORA utilizza un integrato tipo **TMP36** che è un sensore di temperatura di precisione alimentabile a bassa tensione.

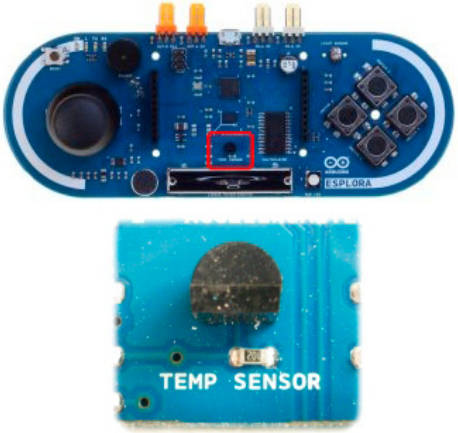
Il sensore fornisce una tensione di uscita che è linearmente proporzionale, la temperatura è in gradi Celsius. Il sensore non

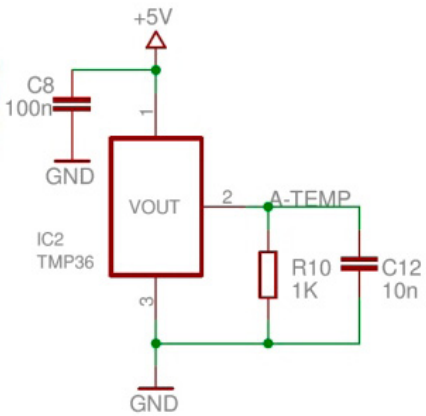


conda dell'argomento, il valore della temperatura può essere in Celsius o in Fahrenheit.

richiede nessuna regolazione esterna per fornire precisioni tipiche di ± 1 ° C a +25 ° C e ± 2 ° C nel range compreso tra i -40 ° C e i +125 ° C. L'uscita lineare e la calibrazione di precisione semplificano l'interfacciamento al circuito di controllo della temperatura e ADC. Il sensore è utilizzabile con singola alimentazione con valori compresi tra un minimo di 2,7 V e un massimo di 5,5 V. La corrente assorbita dal sensore è minore di 50 µA, questo permette un basso autoriscaldamento tipicamente inferiore a 0,1 ° C in aria calma.

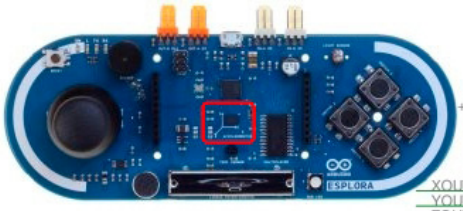
Il TMP36 fornisce un uscita di 750 mV a 25 ° C, e opera a 125 ° C con soli 2,7 V di alimentazione. L'uscita A_TEMP è collegata all'integrato IC3, il valore può essere letto tramite il comando di libreria **Esplora.readTemperature(scale)** a seconda dell'argomento, il valore della temperatura può essere in Celsius o in Fahrenheit.

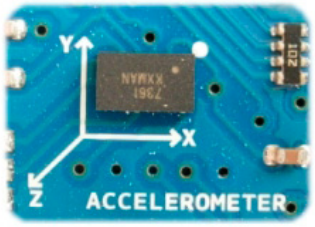


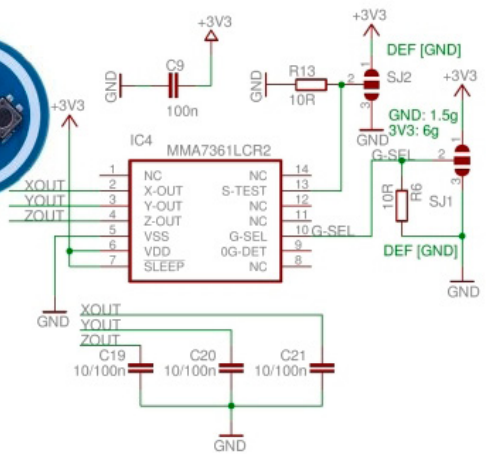


Accelerometro triassiale (X, Y e Z).

L'accelerometro triassiale utilizzato sulla scheda ESPLORA, è il modello a basso costo tipo **MMA7361 LCR2** prodotto dalla Freescale, misura solamente 3x5x1 mm, ha un consumo di 400uA e opera con una tensione compresa tra i 2.2 e i 3.6V. Presenta una sensibilità di 800 mV / g @1.5g: agendo sul pin G-SEL è possibile scegliere la sensibilità per 1.5 o 6g, agendo sul pin S-TEST si può impostare l'auto test. I tre pin di uscita XOUT, YOUT, ZOUT sono connessi all'integrato IC3 e possono essere letti tramite il comando di libreria **Esplora.readAccelerometer(axis)**





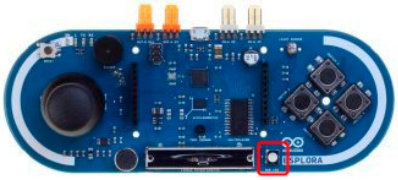


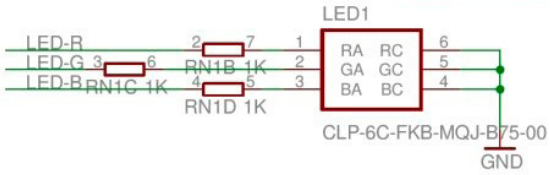
singolarmente, impostando il parametro asse che può essere **X_AXIS, Y-AXIS, o Z_AXIS**.

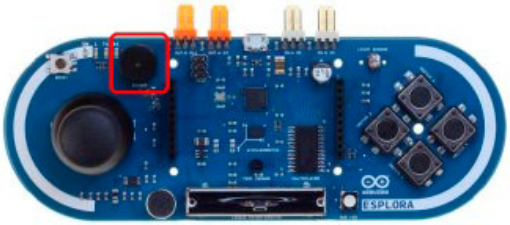
Buzzer per produrre suoni.

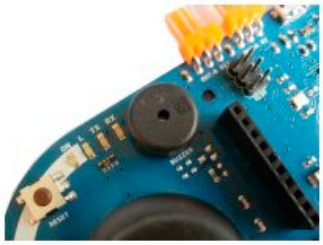
Per la produzione dei suoni, è presente un buzzer connesso alla porta PD7 del processore. Per non sovraccaricare la porta è

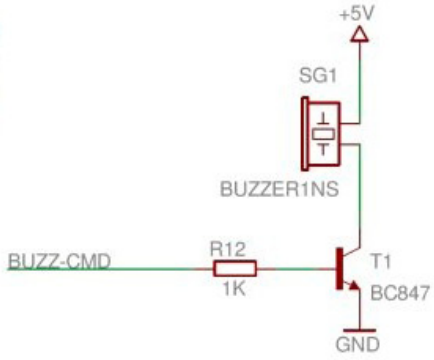
presente un transistor tipo **NPN BC847**. Per la sua gestione sono presenti due comandi della libreria **ESPLORA.tone()** e **ESPLORA.noTone()**. Nel primo caso, il comando genera un'onda quadra di frequenza specificata, può essere specificata la durata in millisecondi, altrimenti l'onda continua finché non è richiamato il comando **ESPLORA.noTone()**.











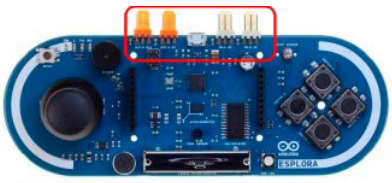
Ingressi e uscite Tinkerkit

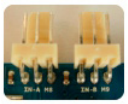
La scheda ESPLORA ha quattro connettori compatibili con il sistema **Tinkerkit**, due ingressi di colore bianco e due di uscita di colore arancio. Questi hanno tre terminali, due per l'alimentazione (+5V e GND) e uno per l'ingresso o l'uscita del segnale. A questi connettori possono essere collegati i moduli della serie Tinkerkit, tra cui moduli sensore come: accelerometri, pulsanti, giroscopi, sensori di tilt, di tocco, magnetici oppure moduli attuatori come led alta potenza, relè, relè allo stato solido, servomotori.

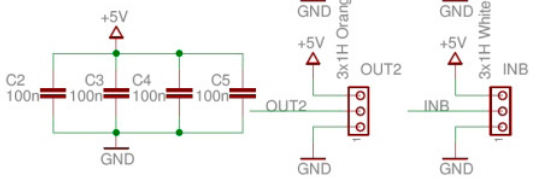
Led RGB con elementi Rosso Verde e Blu per la miscelazione del colore.

Sulla scheda ESPLORA, è presente un led RGB tipo **CLP-6C-FKP**, il led si presenta come un piccolo rettangolo delle dimensioni di 6x5mm.

Al suo interno sono presenti tre led: Rosso (619-624nm), Verde (520-540nm), Blu (460-480nm) che possiedono un'intensità luminosa di 560-1120 mcd per il led rosso, 1120-2240mcd per il led verde e tra i 280-560 mcd per il led blu. L'angolo di vista è di 120° e per ogni led è prevista una resistenza di limitazione da 1kohm. Per il pilotaggio dei led, sono presenti diversi comandi della libreria come **Esplora.writeRGB(red, green, blue)** che permettono di miscelare i vari colori, oppure **Esplora.writeRed(value)**, **Esplora.writeGreen(value)**, **Esplora.writeBlue(value)** per comandare l'emissione dei singoli colori a diverse intensità.





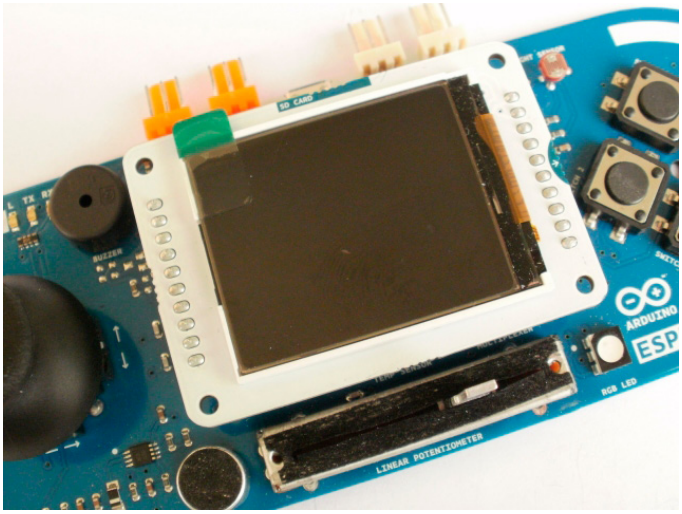


Connettore per display tipo TFT a colori opzionale.

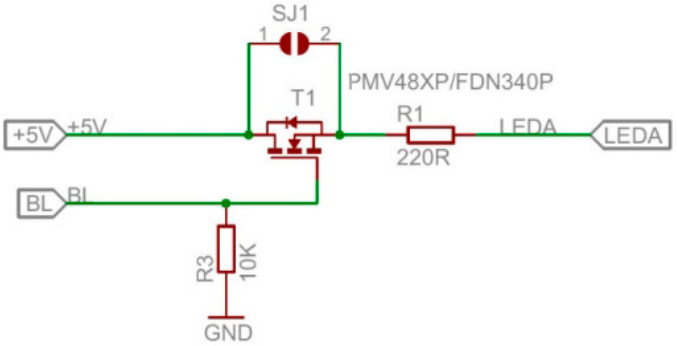
La scheda ESPLORA può essere dotata, di un display tipo **TFT** (Thin Film Transistor), in italiano transistor a pellicola sottile: è una tecnologia

applicata ai display piatti a cristalli liquidi (LCD) o OLED (Organic Light Emitting Diode ovvero diodo organico ad emissione di luce) che vengono in questo modo, identificati come display a matrice attiva. Il display TFT utilizzato è di tipo retroilluminato e misura 1,77" di diagonale, con risoluzione di 160 x 128 pixel, basato sul chip **ST7735** che è un controller / driver a chip singolo per 262K colori.

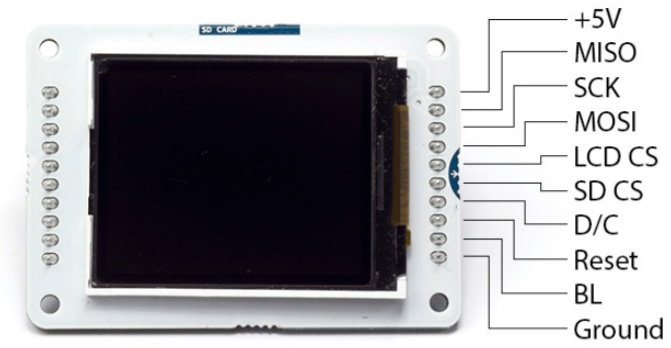
LINK Schema elettrico del display



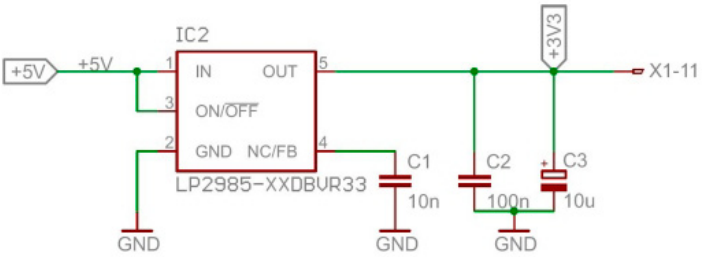
Questo chip, è in grado di collegarsi direttamente a un microprocessore esterno tramite un'interfaccia SPI. I dati possono essere memorizzati all'interno della memoria RAM on-chip. La retroilluminazione del display è dimmerabile tramite il pilotaggio **PWM** del mosfet T1.



Il modulo misura 40x44mm circa, nella parte posteriore è presente uno slot per schede **micro-SD** che, tra le altre cose, permette di memorizzare le immagini bitmap da visualizzare sullo schermo.



Lo schema di alimentazione del modulo, è molto semplice: l'alimentazione a +5V è ridotta al valore di 3,3V con un integrato regolatore **LP2985** del tipo a basso rumore e basso valore di dropout che eroga sino a 150 mA.

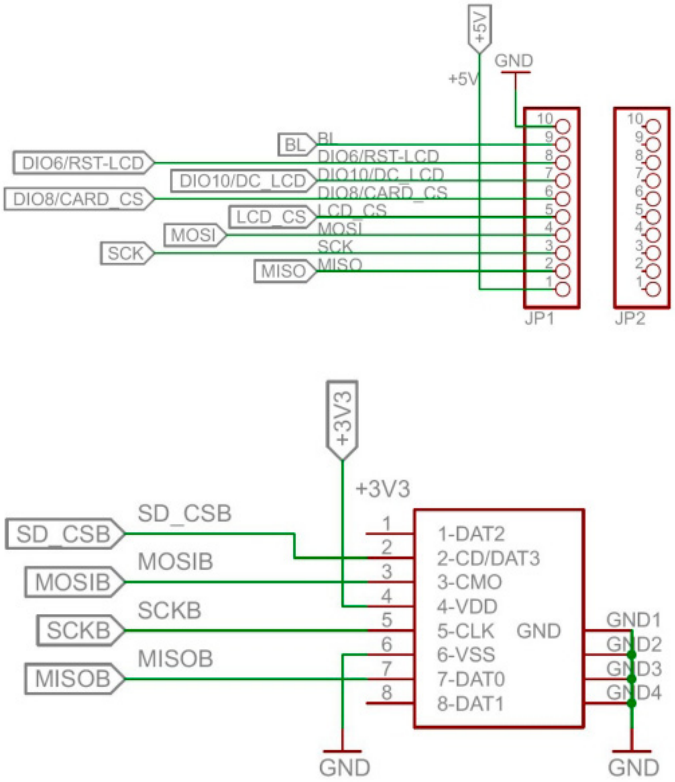


Per la comunicazione, il display utilizza un protocollo SPI (Serial Peripheral Interface). In questo caso, la scheda ESPLORA è il dispositivo master mentre il display è il dispositivo slave. Abbiamo le tre linee di controllo:

- **MISO** (Master In Slave Out) - La linea Slave per l'invio dei dati al master.
- **MOSI** (Master Out Slave In) - La linea principale per l'invio dei dati alle periferiche.
- **SCK** (Serial Clock) - Gli impulsi di clock che sincronizzano la trasmissione di dati generati dal master.

Lo slot micro-SD riceve l'alimentazione a +3,3V tramite il regolatore presente sulla scheda. L'accesso al contenuto della scheda è possibile tramite la libreria SD Library, questa supporta file systems **FAT16** o **FAT32** per schede standard

SD oppure SDHC.



Per la gestione del display, è disponibile un'apposita libreria che è inclusa in quelle disponibili per l'IDE 1.0.5 e successive. Il comando **ESPLORATFT** permette alla scheda ESPLORA di comunicare con lo schermo LCD TFT e semplifica il processo di disegno di forme, linee, immagini e testo sullo schermo.

CONCLUSIONE E PROGETTI FUTURI

È così terminata l'analisi della scheda Arduino Esplora, che sembra avere le carte in regola per diventare un'interfaccia di comando wireless.

Il progetto prevede di dotare la scheda di un modulo **BlueSMiRF-42 RN** che è un modem wireless Bluetooth che può sostituire un collegamento di tipo seriale mediante cavo.

In questo modo sarà possibile comandare un piccolo robot, e ricevere dei dati di telemetria da visualizzare sul display.

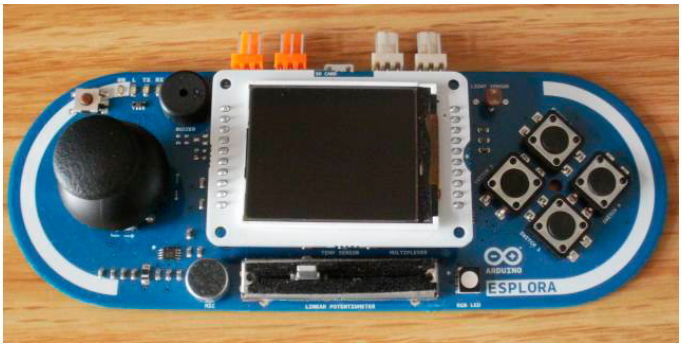
I risultati e le prove saranno oggetto di un prossimo articolo.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/scopriamo-nuova-scheda-arduino-esplora

Programmiamo la scheda Arduino ESPLORA

di adrirobot

La Arduino ESPLORA, la cui presentazione è riportata nell'articolo Scopriamo la nuova scheda Arduino Esplora, a prima vista sembra un controller di videogame, in realtà è un dispositivo che ha un piccolo computer chiamato microcontrollore e una serie di ingressi e uscite.
Per gli ingressi: sono disponibili un joystick, quattro pulsanti, un sensore di luce, un potenziometro a cursore, un microfono, un sensore di temperatura, e un accelerometro.
Per le uscite c'è un "cicalino" e un led RGB.
Con la ESPLORA è possibile scrivere un software che, utilizzando le informazioni acquisite dagli ingressi, gestisce le uscite della scheda o controlla il computer, proprio come si potrebbe fare con un mouse o una tastiera.



Scaricamento del Software Arduino.

Se siete già possessori di schede Arduino e mantenete costantemente aggiornato il software di gestione, probabilmente potrete saltare questo passo. Nel caso invece, ne foste sprovvisti dovrete scaricare l'ultima versione dell' IDE Arduino dalla pagina di download.

http://arduino.cc/en/Main/Software

Attualmente è disponibile la versione 1.0.5. Al termine del download, salvate il file scaricato in qualsiasi directory. Assicuratevi di conservare la struttura delle cartelle.

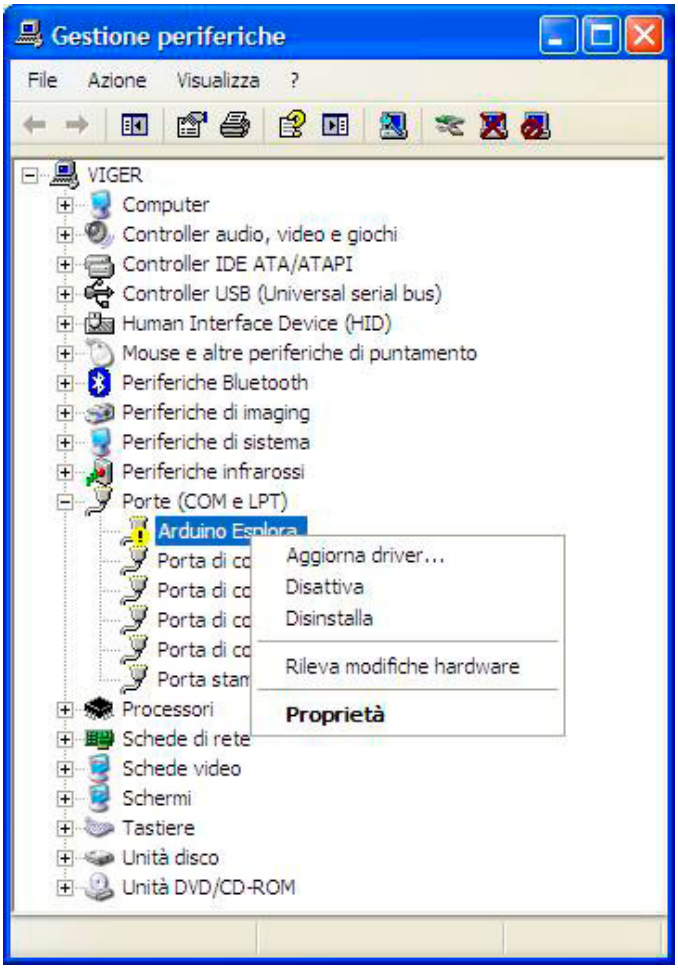
Collegamento della scheda al PC.

La prima operazione da compiere è quella di connettere la scheda al PC: si utilizzerà il cavo USB presente all'interno della confezione, il cavo dispone da un lato di una tipo "A" da un lato e una di tipo "Micro-B" dall'altro. La presa è simile a quella presente su alcuni telefoni cellulari o lettori musicali portatili che utilizzano questo tipo di cavo per il trasferimento dati da/verso il PC. Colleghiamo quindi la scheda Arduino al computer; l'avvenuto collegamento è segnalato dall'accensione del LED di alimentazione verde (etichetta ON) mentre il LED giallo contrassegnato con "L" dopo una prima accensione, dopo circa 8 secondi inizierà a lampeggiare.

Installazione dei driver.

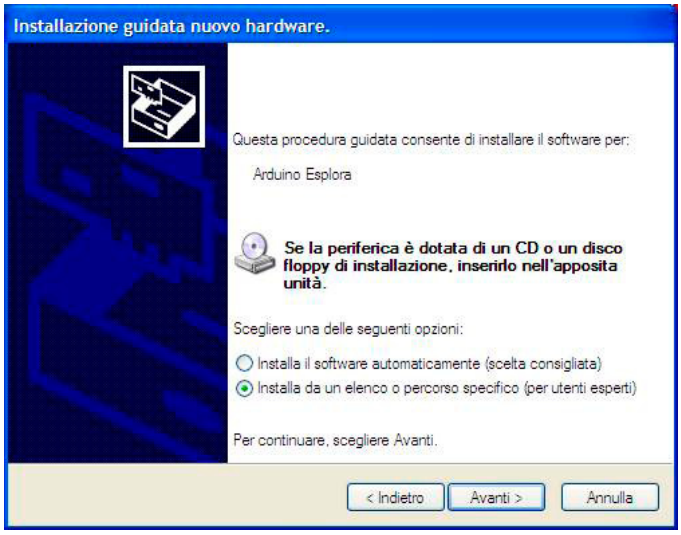
Se è la prima volta che si collega la scheda, sarà necessario l'istallazione dei driver, la procedura è diversa a seconda del sistema operativo. In questa guida vedremo quella per il sistema operativo per Windows XP valida anche per Windows 7, con piccole differenze nelle finestre di dialogo.
Collegate la vostra scheda e attendete sino a che Windows avvia il processo di installazione del driver. Se il programma di installazione non viene avviato automaticamente, individuate la

Gestione periferiche di Windows (Start> Pannello di controllo> Hardware) e trovate il riferimento ad Arduino ESPLORA. Con il tasto destro premuto, scegliete Aggiorna driver.

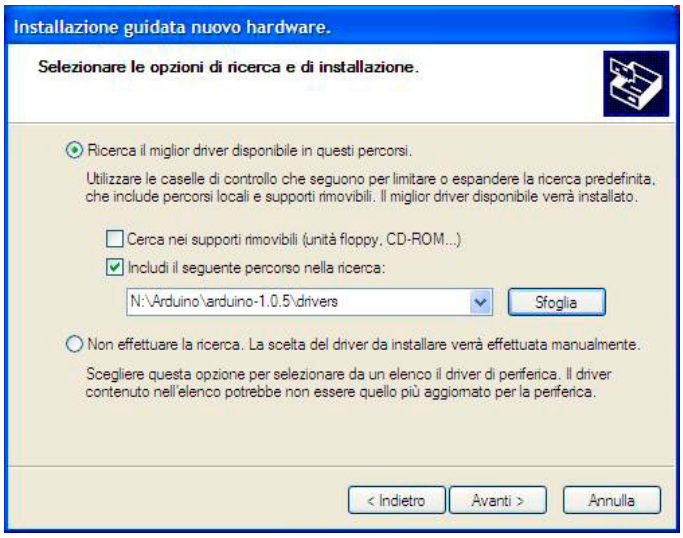


Nella schermata successiva, sarà proposto di consentire a Windows Update di ricercare il software, selezionare “No, non ora” e fare click su Avanti.

scegliere “Installa da un elenco o percorso specifico” e fare clic su avanti

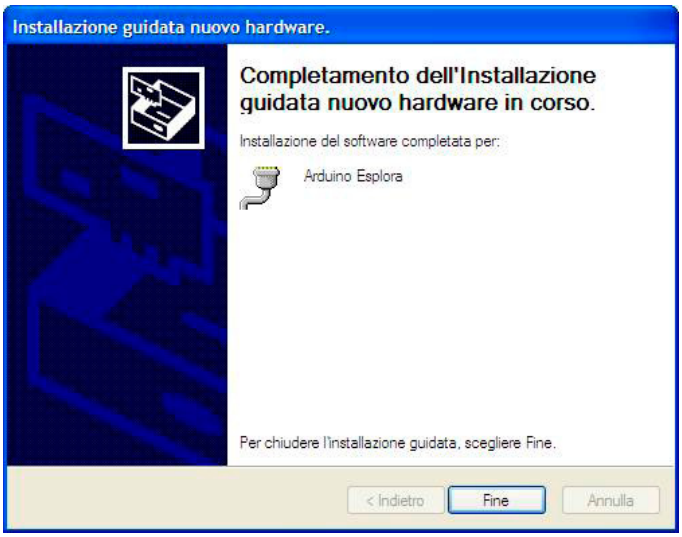


Nella prossima schermata si indicherà la posizione in cui è presente la cartella con il software Arduino che avete appena scaricato. Selezionare la certella driver, quindi fare clic su Avanti.



Si riceverà una notifica che la scheda non ha superato il test di Windows Logo. Fare clic sul pulsante Continua.

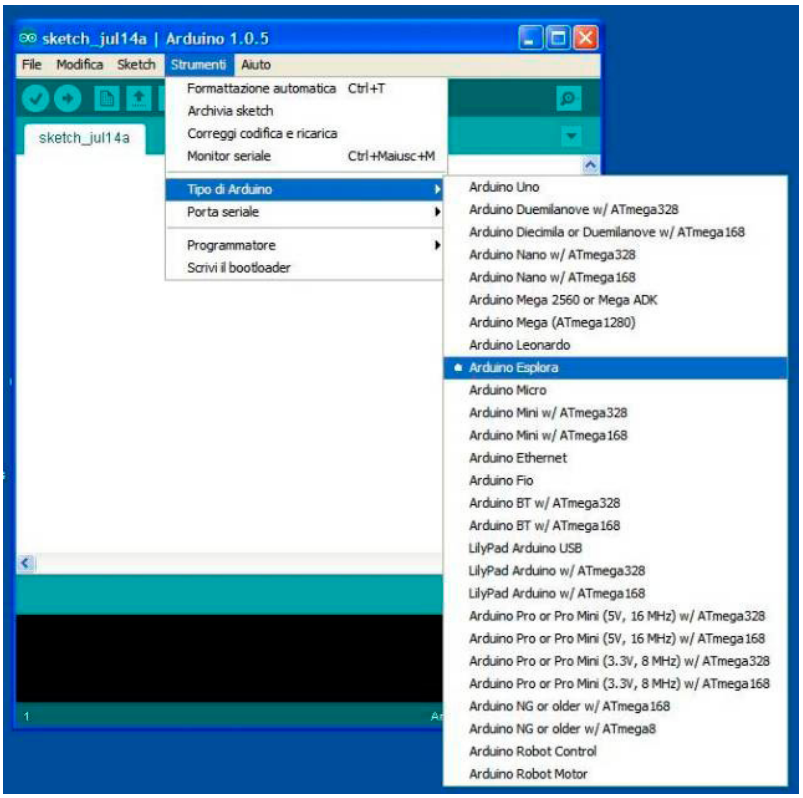
Dopo pochi istanti, una finestra segnalerà che la procedura guidata ha terminato l'installazione del software per Arduino ESPLORA. Premere il pulsante Fine.



Configuriamo l'IDE per utilizzare la Arduino ESPLORA.

Dal momento che l'IDE Arduino è utilizzato per molte schede Arduino diverse, è necessario impostare l'IDE con la scheda ESPLORA.

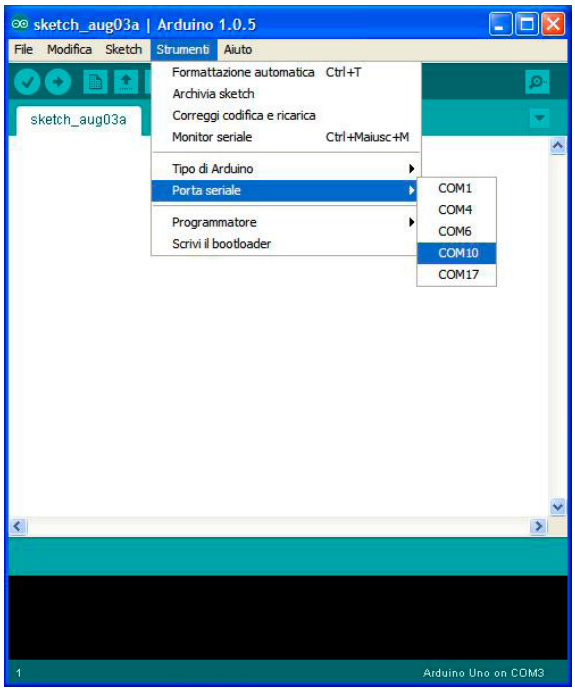
Per selezionare la scheda aprire il menu Strumenti> Tipo di Arduino e scegliere **Arduino ESPLORA**.



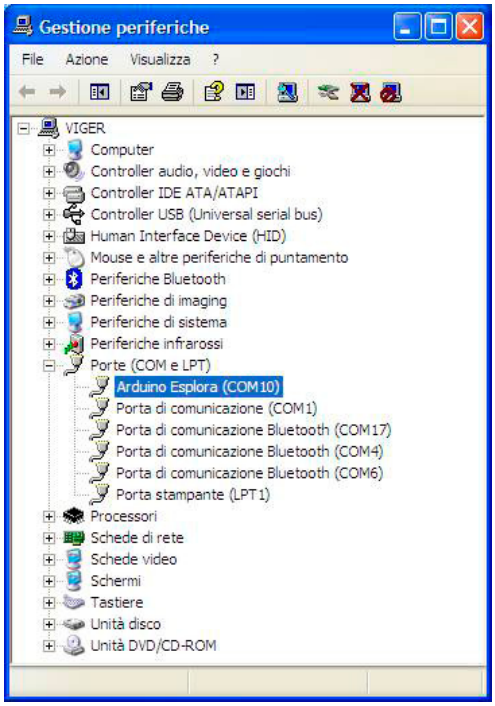
Selezionare la porta USB.

Per programmare la scheda, l'IDE ha bisogno di sapere la porta USB a cui è collegata. Nel menu Strumenti>Porta seriale sono elencate le porte disponibili.

Tra quelle disponibili selezionare quella a cui è collegata la scheda



Il numero della porta è visibile da: Gestione periferiche di Windows (Start> Pannello di controllo> Hardware) e trovate il riferimento a Arduino ESPLORA.



Altro metodo è quello di scollegare ESPLORA e riaprire il menu, la voce che scompare dovrebbe essere la scheda ESPLORA. Collegare nuovamente la scheda e selezionare quella porta seriale.

Le librerie per la gestione della scheda ESPLORA.

Con l'uscita della scheda ESPLORA sono state inserire nell'IDE (dalla versione 1.0.3 in poi) le librerie per la sua gestione, tra cui:

- Esplora : con una serie di funzioni per l'interfacciamento con i sensori e gli attuatori montati sulla scheda.
- TFT: per la gestione del display 160X128 di cui può essere dotata la scheda.
- SD: per la gestione della scheda SD che può essere istallata sul display TFT

Sul sito sono disponibili degli esempi per meglio comprenderne il loro uso.

La libreria Esplora.

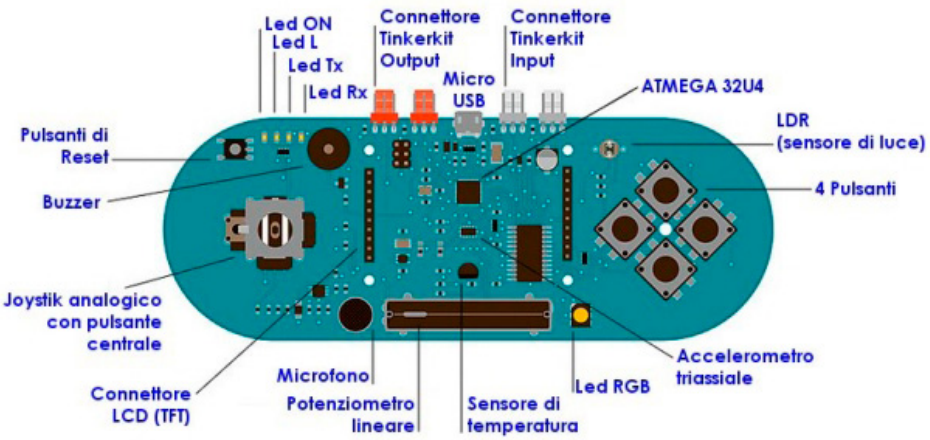
La libreria offre un facile accesso ai dati dai sensori di bordo e fornisce la possibilità di cambiare lo stato delle uscite. Le funzioni della libreria sono accessibili tramite la classe ESPLORA. (http://arduino.cc/en/Reference/EsploraLibrary)

Ricapitolando, i sensori che sono disponibili sulla scheda:

- Joystick analogico a due assi.
- Pulsante centrale del joystick
- 4 pulsanti
- Un microfono
- Un sensore di luce
- Un sensore di temperatura
- Un zccelerometro a 3 assi
- 2 connettori di ingresso Tinkerkit

Abbiamo poi i seguenti attuatori:

- Un led RGB (Rosso-verde-blu)
- Un piezo buzzer
- 2 connettori di uscita Tinkerkit



Per la gestione delle funzioni disponibili all'interno della libreria troviamo:

- Esplora.readSlider()
- Esplora.readLightSensor()
- Esplora.readTemperature()
- Esplora.readMicrophone()
- Esplora.readJoystickSwitch()
- Esplora.readJoystickButton()
- Esplora.readAccelerometer()
- Esplora.readButton()
- Esplora.readJoystickX()
- Esplora.readJoystickY()
- Esplora.writeRGB()
- Esplora.writeRed()
- Esplora.writeGreen()
- Esplora.writeBlue()
- Esplora.readRed()
- Esplora.readGreen()
- Esplora.readBlue()
- Esplora.tone()
- Esplora.noTone()

Analizziamo le varie funzioni: per ognuna è presente la descrizione, la sintassi del comando, gli eventuali parametri da inserire con il loro range e l'eventuale dato che viene fornito dalla funzione.

Esplora.readSlider()
La funzione legge il valore del potenziometro lineare come un numero a 10 bit. Ciò significa che mapperà la tensione d'ingresso compresa tra 0 e 5 volt in valori interi compresi tra 0 e 1023. Questo produce una risoluzione tra le letture di: 5 volt / 1024 unità oppure 0,0049 volt (4,9 mV) per unità.

Sintassi: *Esplora.readSlider()*

Dato fornito int: La posizione del potenziometro lineare, mappato a un valore compreso tra 0 e 1023.

Esplora.readLightSensor()
La funzione legge l'intensità della luce che colpisce il sensore (rappresentato da una fotocellula) come un numero a 10 bit. Ciò significa che mapperà tensioni d'ingresso analogamente al comando Esplora.readSlider()

Sintassi: *Esplora.readLightSensor()*

Dato fornito: int : La quantità di luce che colpisce il sensore, mappato a un valore tra 0 e 1023.

Esplora.readTemperature()
Legge la temperatura dell'ambiente fornito dal sensore e restituisce una lettura sia nella scala Celsius che Fahrenheit, a seconda del parametro passato.

Sintassi: *Esplora.readTemperature(scale)*

Parametri: scale: la scala in uscita, argomenti validi sono: DEGREES_C e DEGREES_F
Dato in uscita: int: La lettura della temperatura in gradi Celsius o Fahrenheit. In Celsius l'intervallo è compreso tra -40 ° C a 150 ° C, la gamma Fahrenheit tra -40 ° F a 302 ° F.

Esplora.readMicrophone()
Legge l'ampiezza fornita dal microfono, restituendo un valore tra 0 e 1023.

Sintassi: *Esplora.readMicrophone()*

Dato fornito int: Il valore dell'ampiezza mappato in una gamma da 0 a 1023.

Esplora.readJoystickSwitch()
Legge lo stato del tasto del joystick e restituisce 0 o 1023. Se si preferisce qualcosa di più coerente con la funzione readButton (), si consiglia di utilizzare la funzione readJoystickButton (). Tale funzione si comporta nello stesso modo ma restituisce LOW quando si preme il tasto del joystick, e HIGH quando non è premuto.

Sintassi: *Esplora.readJoystickSwitch()*

Valore fornito 0 quando il tasto è premuto, 1023 quando il tasto non è premuto (stato normale).

Esplora.readAccelerometer()
Legge i valori forniti dall'accelerometro. Ciascuno dei tre assi è accessibile indipendentemente.

Sintassi: *Esplora.readAccelerometer(axis)*

Parametri: axis : char, determina quale asse

deve essere letto.

- X_AXIS per leggere il valore dell'asse X.
- Y_AXIS per leggere il valore dell'asse Y.
- Z_AXIS per leggere il valore dell'asse Z.

Dato fornito int: il valore delle letture sull'asse prescelto. L'accelerometro restituisce zero quando è perpendicolare alla direzione della gravità. Sono forniti valori positivi o negativi quando è mosso in una delle due direzioni dell'asse.

Esplora.readButton()
Legge lo stato di un pulsante e restituisce se è HIGH o LOW

Sintassi: *Esplora.readButton(button)*

Parametri: button: il pulsante associato che si vuole leggere.
Argomento validi sono:

- SWITCH_1 oppure SWITCH_DOWN
- SWITCH_2 oppure SWITCH_LEFT
- SWITCH_3 oppure SWITCH_UP
- SWITCH_4 oppure SWITCH_RIGHT
- JOYSTICK_DOWN = JOYSTICK_BASE
- JOYSTICK_LEFT = JOYSTICK_BASE+1
- JOYSTICK_UP = JOYSTICK_BASE+2
- JOYSTICK_RIGHT = JOYSTICK_BASE+3

Dato fornito: LOW quando premuto, HIGH quando non premuto (situazione normale).

Esplora.readJoystickY()
Utilizzata per leggere la posizione dell'asse Y del joystick. Quando il joystick è al centro, restituisce zero. Valori positivi indicano che il joystick è spostato in alto, valori negativi quando questo è spostato verso il basso.

Sintassi: *Esplora.readJoystickY()*

Dato fornito:
int: 0 quando il joystick è in mezzo l'asse y; valori tra 1 e 512 quando il joystick è spostato in alto; valori tra -1 e -512 quando il joystick viene spostato verso il basso.

Esplora.readJoystickX()
Utilizzata per leggere la posizione dell'asse X del joystick. Quando il joystick è al centro, restituisce zero. Valori positivi indicano che il joystick è spostato a destra, valori negativi quando è spostato a sinistra.

Sintassi: *Esplora.readJoystickX()*

Dato fornito:
int: 0 quando il joystick è nel mezzo dell'asse x; valori tra 1 e 512 quando il joystick viene spostato verso destra; -1 e -512 quando il joystick viene spostato a sinistra.

Esplora.writeRGB()
Utilizzata per scrivere i valori che rappresentano la luminosità del LED RGB. Mescolando valori diversi, è possibile creare diverse combinazioni di colori.

Sintassi: *Esplora.writeRGB(red, green, blue)*

Parametri:

- red: int, valore per modificare la luminosità del led rosso con un range tra 0 e 255
- green: int, valore per modificare la luminosità del led verde con un range tra 0 e 255
- blue: int, valore per modificare la luminosità del led blu con un range tra 0 e 255

Esplora.writeRed();Esplora.writeGreen();Esplora.writeBlue();
Utilizzata per variare la luminosità dei singoli led contenuti nel led RGB

Sintassi:
- *Esplora.writeRed(value)*
- *Esplora.writeGreen(value)*
- *Esplora.writeBlue(value)*

Parametri:
value: int, valore per modificare la luminosità del led rosso (oppure verde o blu) con un range tra 0 e 255

Esplora.readRed(),readGreen(), readBlue()
Utilizzata per leggere il valore di luminosità di uno dei tre LED RGB.

Sintassi:
- *Esplora.readRed()*
- *Esplora.readGreen()*
- *Esplora.readBlue()*

Dato fornito:
int : valore della luminosità del led rosso (oppure verde o blu) con un range tra 0 e 255

Esplora.tone()
Genera un'onda quadra di frequenza specificata emessa attraverso il buzzer di bordo di ESPLORA. Può essere specificato la sua durata espressa in millisecondi, altrimenti l'onda continua finché non si effettua una chiamata a Esplora.noTone (). Può essere generato un solo tono alla volta. Se il segnale è già in riproduzione, ne viene variata la frequenza. L'uso della funzione Esplora.tone () può interferire con la dissolvenza il LED rosso.
Sintassi: *Esplora.tone(frequency, duration)*

Parametri:
- frequency: la frequenza del tono in hertz - unsigned int
- duration: la durata del tono in millisecondi (opzionale) - unsigned long

Esplora.noTone()
La funzione interrompe la generazione di un'onda quadra generato tramite la funzione Esplora.tone (). Non ha effetto se non è generato nessun tono.

Sintassi: *Esplora.noTone()*

Libreria TFT
Questa libreria permette alla scheda Arduino Esplora di comunicare con lo schermo LCD TFT.
http://arduino.cc/en/Reference/TFTLibrary

Semplifica il processo di disegno di forme, linee, immagini e testo sullo schermo. La biblioteca estende le librerie Adafruit GFX, e Adafruit ST7735 su cui si basa. Adafruit è la società che ha creato il primo modello di scheda con display TFT, da cui è stata creato il modello ora commercializzato da Arduino.
La biblioteca GFX è responsabile delle routine di disegno, mentre la libreria ST7735 è specifica per lo schermo dell'Arduino TFT. Le aggiunte

specifiche sono state progettate per funzionare in modo simile a delle API (Application Programming Interface).

Per impostazione predefinita, lo schermo è orientato in modo da essere più largo che alto. La parte superiore dello schermo è la stessa parte del testo 'SD CARD'. In quest'orientamento, lo schermo è di 160 pixel di larghezza e 128 pixel di altezza.

Quando ci si riferisce alle coordinate sullo schermo, occorre immaginare una griglia. Ogni quadrato della griglia è un pixel. È possibile identificare la posizione dei pixel con coordinate specifiche. Un punto in alto a sinistra avrebbe il valore 0,0. Se questo punto dovesse spostarsi nella parte superiore destra dello schermo, le sue coordinate sarebbero 0, 159, nell'angolo in basso a sinistra, le coordinate sarebbero 127,0, e in basso a destra 127.159.

E' possibile utilizzare lo schermo in orientamento verticale (chiamato anche "portrait"), richiamando il comando EsploraTFT.setRotation (0), a questo punto le coordinate x ed y degli assi cambiano di conseguenza, così come cambiano i dati forniti da screen.width () o screen.height ().

Come abbiamo visto, sulla scheda è presente uno slot per schede SD, che può essere utilizzata attraverso la libreria SD.

La biblioteca TFT si basa sulla libreria SPI per la comunicazione con lo schermo e la scheda SD e deve essere inclusa in tutti gli sketches.

Utilizzo della libreria

Dato che su Arduino ESPLORA la configurazione dei pin di collegamento con il display è fissa, per la gestione sarà sufficiente la creazione di un solo oggetto ESPLORA. Nella parte iniziale dello sketch dovranno essere presenti le seguenti linee #include:

- #include <Esplora.h>
- #include <TFT.h> // Arduino LCD library
- #include <SPI.h>

Nella cartella della libreria sono presenti alcuni esempi per meglio comprendere l'utilizzo dei comandi.

- *Esplora TFT Bitmap Logo*: Legge un file di immagine da una scheda micro-SD e lo disegna in posizioni casuali.
- *Esplora TFT Color Picker*: Utilizzando il joystick e il dispositivo di scorrimento, è possibile modificare il colore dello schermo TFT
- *Esplora TFT Etch a Sketch*: Un'implementazione ESPLORA del classico disegno a mano libera.
- *Esplora TFT Graph*: Disegna sul display il Grafico dei valori dal sensore di luce.
- *Esplora TFT Horizon*: Disegna una linea di orizzonte artificiale basato sul tilt dall'accelerometro
- *Esplora TFT Pong*: Un'implementazione del classico gioco
- *Esplora TFT Temperature*: visualizza sullo schermo la temperatura rilevata dal sensore a bordo

Ecco i comandi presenti nella libreria:

- EsploraTFT.begin()
- EsploraTFT.background()
- EsploraTFT.stroke()
- EsploraTFT.noStroke()
- EsploraTFT.fill()
- EsploraTFT.noFill()
- EsploraTFTtext()
- EsploraTFT.setTextSize()
- EsploraTFT.begin()
- EsploraTFT.point()

- EsploraTFT.line()
- EsploraTFT.rect()
- EsploraTFT.width()
- EsploraTFT.height()
- EsploraTFT.circle()
- PImage
- EsploraTFT.image()
- EsploraTFT.loadImage()
- PImage.height()
- PImage.width()
- PImage.isValid

EsploraTFT.begin()

La funzione deve essere chiamata per inizializzare lo schermo prima di disegnare o scrivere qualsiasi cosa.

Sintassi: *EsploraTFT.begin();*

EsploraTFT.background ()

La funzione cancella tutto il contenuto dello schermo con il colore indicato.

Può essere utilizzato all'interno di un loop () per cancellare lo schermo.

Lo schermo ha la capacità di mostrare il colore a 16 bit. Il rosso e il blu hanno 5 bit di risoluzione ciascuno (32 livelli di rosso e blu), il verde, dispone di 6-bit di risoluzione (64 livelli diversi).

Per coerenza con altre applicazioni, la libreria permette di inserire colori con valori di 8 bit per i canali rosso, verde e blu (0-255) e bilancia i colori in modo appropriato.

Sintassi: *screen.background(red, green, blue);*

Parametri:

- red : int 0-255
- green : int 0-255
- blue : int 0-255

Esempio:

```
#include <Esplora.h>  //Libreria Arduino Esplora
#include <SPI.h>      //Libreria SPI
#include <TFT.h>      //Libreria Arduino LCD TFT
Void setup() {
  // inizializza lo scermo
  EsploraTFT.begin();
  // Imposta lo sfondo Bianco
  EsploraTFT.background(255, 255, 255);
  delay (1000);
  // Imposta lo sfondo nero
  EsploraTFT.background(0, 0, 0);
  delay (1000);
  // Imposta lo sfondo rosso
  EsploraTFT.background(255, 0, 0);
  delay (1000);
  // Imposta lo sfondo giallo
  EsploraTFT.background(255, 255, 0);
  delay (1000);
  // Imposta lo sfondo verde
  EsploraTFT.background(0, 255, 0);
  delay (1000);
  // Imposta lo sfondo ciano
  EsploraTFT.background(0, 255, 255);
  delay (1000);
  // Imposta lo sfondo blu
  EsploraTFT.background(0, 0, 255);
  delay (1000);
  // Imposta lo sfondo magenta
  EsploraTFT.background(255, 0, 255);
  delay (1000);
}
void loop() {
}
```

EsploraTFT.stroke()

Chiamato prima di disegnare un oggetto sullo schermo, imposta il colore delle linee e dei bordi intorno alle forme.

stroke () si aspetta valori a 8 bit per ognuno dei canali rosso, verde e blu, ma lo schermo non visualizza con questa fedeltà. Vedere comando EsploraTFT.background ()

Sintassi: *EsploraTFT.stroke(red, green, blue);*

Parametri:

- red : int 0-255
- green : int 0-255
- blue : int 0-255

EsploraTFT.noStroke

Dopo aver eseguito questo comando, eventuali forme disegnate sullo schermo non avranno un colore del tratto di contorno.

Sintassi: *EsploraTFT noStroke();*

EsploraTFT.fill()

Chiamato prima di disegnare un oggetto sullo schermo, imposta il colore di riempimento di forme e testo.
fill () si aspetta valori a 8 bit per ognuno dei canali rosso, verde e blu, ma lo schermo non visualizza con questa fedeltà. Vedere comando EsploraTFT.background ().

Sintassi: *EsploraTF.fill(red, green, blue);*

Parametri:

- red : int 0-255
- green : int 0-255
- blue : int 0-255

EsploraTFT.noFill

Dopo aver inseriro questo comando, eventuali forme disegnate sullo schermo non saranno riempite.

Sintassi: *EsploraTFT noFill();*

EsploraTFT.text

Scrive il testo sullo schermo nelle coordinate indicate.

Sintassi: *EsploraTFT.text(text, xPos, yPos);*

Parametri:

- text : array di caratteri, il testo da scrivere sullo schermo
- xPos : int, la posizione su l'asse x in cui si desidera iniziare a scrivere il testo sullo schermo
- yPos : int, la posizione su l'asse y in cui si desidera iniziare a scrivere il testo sullo schermo

Esempio

```
#include <Esplora.h>  //Libreria Arduino Esplora
#include <SPI.h>      //Libreria SPI
#include <TFT.h>      //Libreria Arduino LCD TFT

void setup() {
  // inizializza lo scermo
  EsploraTFT.begin();
  // Imposta lo sfondo nero
  EsploraTFT.background(0, 0, 0);
  //Imposta il colore del testo a bianco
  EsploraTFT.stroke(255,255,255);
  //Scrive il testo sullo schermo in alto a sinistra
  EsploraTFT.text("Testo di prova!", 0, 0);
}
void loop() {
}
```

EsploraTFT.setTextSize

Consente di impostare la dimensione del testo che segue. La dimensione predefinita è "1". Ogni variazione di dimensione aumenta il testo di 10 pixel di altezza. Cioè, taglia 1 = 10 pixel, dimensione 2 = 20 pixel, e così via.

Sintassi: *EsploraTFT.setTextSize(size);*

Parametri:
size : int 1-5

Esempio

```
#include <Esplora.h>  //Libreria Arduino Esplora
#include <SPI.h>      //Libreria SPI
#include <TFT.h>      //Libreria Arduino LCD TFT

void setup() {
  // inizializza lo scermo
  EsploraTFT.begin();
  // Imposta lo sfondo nero
  EsploraTFT.setRotation(0);
  EsploraTFT.background(0, 0, 0);
  //Imposta il colore del testo a bianco
  EsploraTFT.stroke(255,255,255);
  //Altezza di default
  EsploraTFT.setTextSize(1);
  EsploraTFT.text("H=1", 0, 0);
  //Incremento dell'altezza
  EsploraTFT.setTextSize(2);
  EsploraTFT.text("H=2", 0, 10);
  //Incremento dell'altezza
  EsploraTFT.setTextSize(3);
  EsploraTFT.text("H=3", 0, 30);
  //Incremento dell'altezza
  EsploraTFT.setTextSize(4);
  EsploraTFT.text("H=4", 0, 60);
  //Incremento dell'altezza
  EsploraTFT.setTextSize(5);
  EsploraTFT.text("H=5", 0, 100);
}
void loop() {
}
```

EsploraTFT.point()

Disegna un punto nelle coordinate indicate. Il primo parametro è il valore orizzontale, il secondo è il valore verticale per il punto.

Sintassi: *EsploraTFT point(xPos, yPos);*

Parametri:

- xPos : int, la posizione orizzontale del punto.
- yPos : int, la posizione verticale del punto.

EsploraTFT.line()

Disegna una linea tra due punti. Utilizzare EsploraTFT stroke () per cambiare il colore prima di tracciare la linea.

Sintassi: *EsploraTFT.line (xStart, yStart, xEnd, yEnd);*

Parametri

1. xStart: int, la posizione orizzontale in cui la linea inizia
2. yStart: int, la posizione verticale in cui la linea inizia
3. xEnd: int, la posizione orizzontale in cui la linea si conclude
4. yEnd: int, la posizione verticale in cui la linea si conclude

EsploraTFT.rect()

Disegna un rettangolo sullo schermo TFT. EsploraTFT.rect() accetta 4 argomenti, i primi due sono l'angolo superiore sinistro della forma, gli ultimi due sono la larghezza e l'altezza della forma.

Sintassi: *EsploraTFT.rect(xStart, yStart, width, height);*

Parametri

- xStart: int, la posizione orizzontale in cui iniziare il rettangolo
- Ystart: int, la posizione verticale in cui iniziare il rettangolo
- width: int, la larghezza del rettangolo
- height: int, l'altezza del rettangolo

EsploraTFT.width()

Riporta la larghezza dello schermo TFT in pixel.

Sintassi: *EsploraTFT.width();*

Dato fornito:
int : la larghezza dello schermo in pixel

EsploraTFT.height()
Riporta l'altezza dello schermo TFT in pixel.

Sintassi: *EsploraTFT.height();*

Dato fornito:
int : l'altezza dello schermo in pixel

EsploraTFT.circle()
Disegna un cerchio sullo schermo. Il cerchio è disegnato rispetto al suo punto centrale, il che significa che il diametro totale sarà sempre un numero dispari.

Sintassi: *EsploraTFT.circle(xPos, yPos, radius);*

Parametri:
- xPos: int, la posizione del centro del cerchio sull'asse x
- yPos: int, la posizione del centro del cerchio sull'asse y
- radius: int, il raggio del cerchio

PImage
E' la classe di base che trasferisce un'immagine bitmap caricata dalla scheda SD sullo schermo. È necessario includere la libreria SD per utilizzare PImage.

Sintassi: PImage image;

Parametri
- Image: il nome dell'oggetto PImage. Ci si riferirà a questo nome successivamente per elaborare le immagini sullo schermo.

EsploraTFT.image
Disegna un'immagine dalla scheda SD allo schermo in una posizione specificata.

Sintassi: *EsploraTFT.image(image, xPos, yPos);*

Parametri:
- image: l'istanza denominata di PImage.
- xPos: int, posizione sul l'asse x per iniziare a disegnare
- yPos: int, posizione sulla asse y per iniziare a disegnare

EsploraTFT.loadImage()
Carica un file immagine dalla scheda SD in un'istanza denominata da PImage.

Sintassi: *EsploraTFT.loadImage(name);*

Parametri:
- Nome: array di caratteri, il nome dell'immagine dalla scheda SD che si desidera leggere

PImage.height
La funzione fornisce l'altezza dell'oggetto PImage.

Sintassi: *EsploraTFT image.height();*

Dato fornito:
int: altezza dell'immagine in pixel

EsploraTFT.width
La funzione fornisce la larghezza dell'oggetto PImage.

Sintassi: *EsploraTFT.width();*

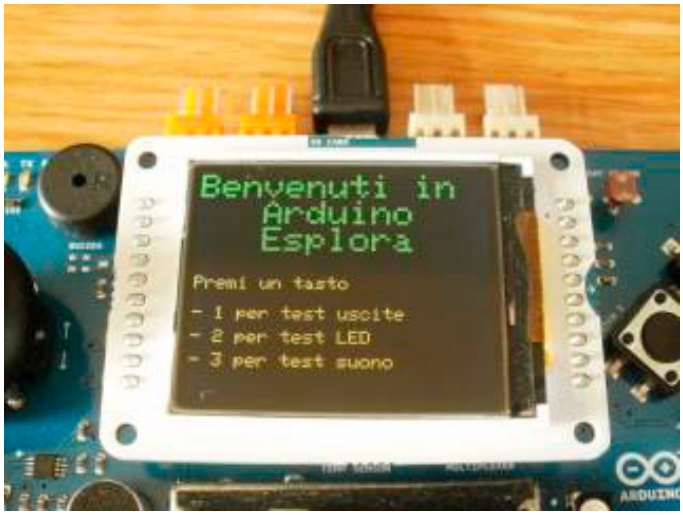
Dato fornito:
int : larghezza dell'immagine in pixel

PImage.isValid
La funzione controlla se il file bitmap caricato con riferimenti PImage è valido.

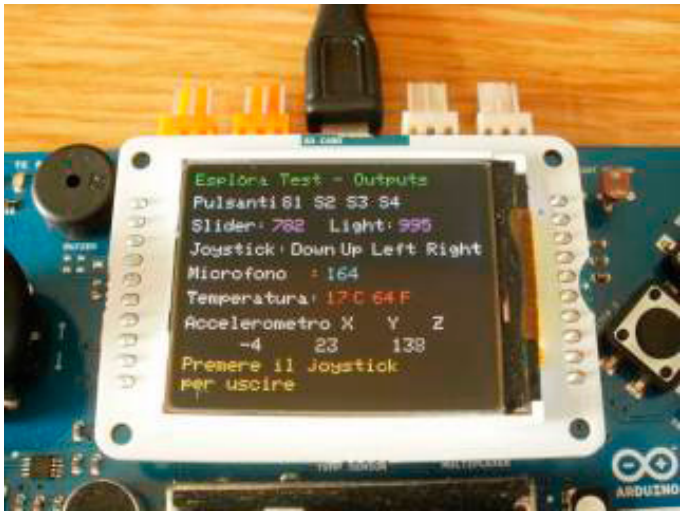
Sintassi: *image.isValid();*

Dato fornito:
Booleano

Programma di test per la scheda Arduino Esplora dotata di display TFT
Il programma proposto permette di testare tutti i sensori e gli attuatori presenti sulla scheda.
Il programma si trova all'interno del file allegato arduino_esplora_test.zip.
Le istruzioni e i valori letti sono mostrati sul display TFT che deve essere obbligatoriamente installato.
Il programma originale è presente a questo link.
Questa versione è solamente un adattamento/traduzione delle istruzioni per utilizzarlo con le librerie originali Arduino. Anche i testi presenti sul display sono stati tradotti.
Appena caricato il programma sarà mostrata una pagina di presentazione da cui scegliere il tipo di prova da effettuare.
Per fare questo occorrerà premere il relativo tasto tra quelli presenti alla destra del display.
Con il tasto 1 si testano i vari sensori, con il tasto 2 si effettua il test del led RGB, mentre il tasto 3 permette il test del buzzer.



Test 1 - Sensori
Quando si premono i pulsanti o si sposta il joystick, la parola appropriata passa dal bianco al verde. Muovendo il potenziometro lineare (slider) i valori varieranno partendo da 1023 a sinistra a 0 sulla destra.
Viene visualizzato il valore del suono rilevato dal microfono, così come il valore di sensore di luce.
Il valore della temperatura è fornito sia in gradi Celsius che Fahrenheit. Infine, sono visualizzati i valori forniti dell'accelerometro (X, Y e Z).
Quando si vuole uscire, spingere verso il basso il joystick (non spostarlo, ma spingerlo con decisione verso il basso) per tornare al menu.

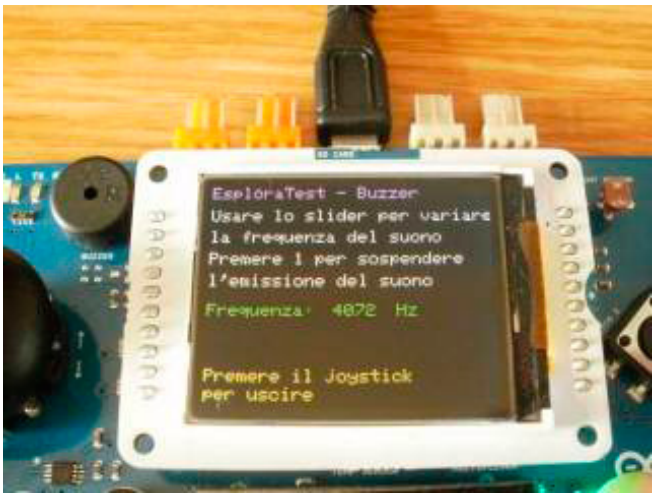


Test 2 - Led RGB.
Se dal menu principale si preme il tasto 2 si pas-

sa al programma di Test del LED RGB.
I valori del rosso e verde possono essere variati da 0 a 255 spostando il joystick verso l'alto, il basso, sinistra e destra. Mentre per regolare i valori del blu si dovrà variare la posizione del cursore del potenziometro lineare. Per uscire, premere di nuovo verso il basso il joystick.

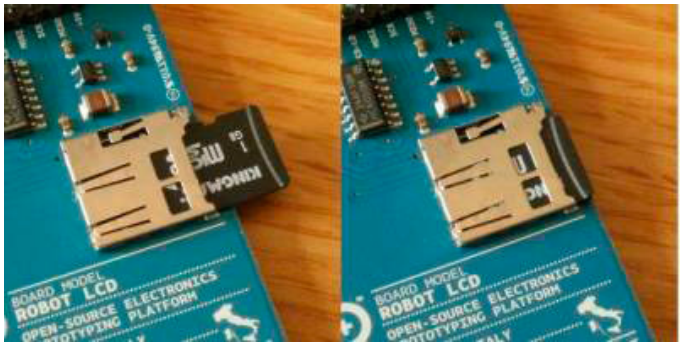
Test 3 - Buzzer

Infine, per testare il buzzer selezionare il tasto 3 del menu principale. Si dovrà spostare il cursore del potenziometro lineare per regolare la frequenza. Partendo da frequenze basse con il cursore tutto a sinistra a frequenze più alte spostando il cursore verso destra.
Se non si sposta il cursore il suono si arresterà dopo 5 secondi, per arrestarlo immediatamente premere il tasto 1. Premere di nuovo il joystick verso il basso per accedere al menu.

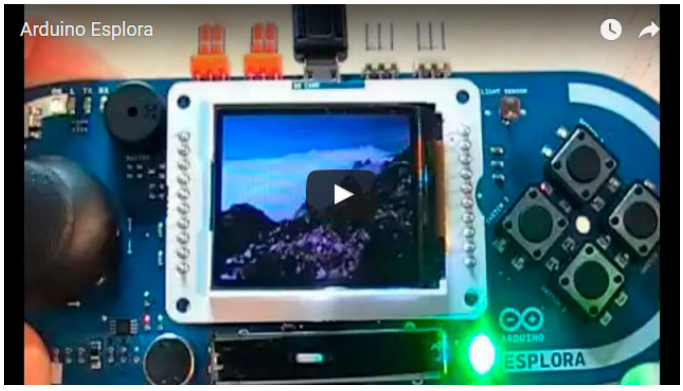


Programma di test della memoria SD

Un altro programma presente negli allegati (arduino_esplora_test.zip) permette di testare la possibilità di lettura di file immagine bitmapped da mostrare poi sul display TFF.
Per utilizzare il programma occorrerà avere una memoria mini-SD con una capacità da 1 MB, su cui memorizzare i file .bmp allegati.
Si inserirà quindi la memoria nell'apposito alloggiamento presente sul display, nella parte superiore, lato presa USB.
Una volta caricato il programma saranno mostrati ciclicamente i file sul display, l'accensione del led RGB in verde confermerà il riconoscimento corretto del file, l'accensione del led di colore rosso segnalerà un'anomalia.



Filmato illustrativo



https://www.youtube.com/watch?v=ll1cRHcti24

CONCLUSIONI

Fatta questa panoramica, ora potrete cominciare ad utilizzare la scheda per poterne saggiare

le potenzialità. Per il futuro, nei prossimi articoli sarà presentato un piccolo robot che sarà comandato in remoto tramite questa interfaccia.

DOCUMENTAZIONE COMPLETA

La documentazione è disponibile, a questo link: arduino_esplora_test.zip.

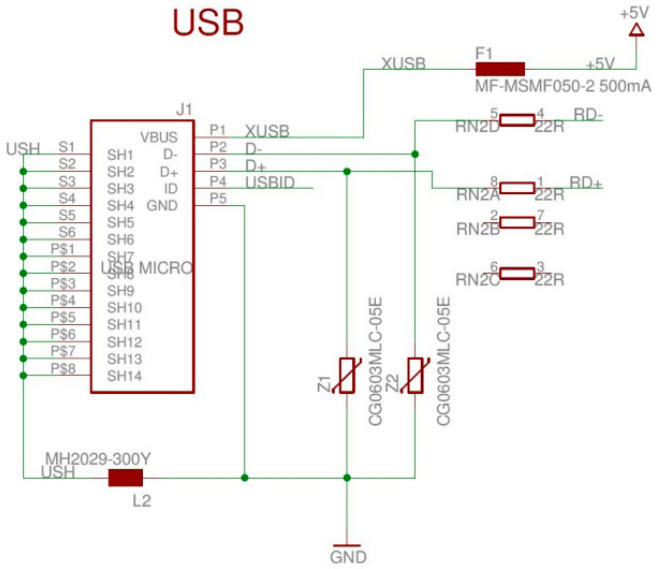
Sono presenti due cartelle, in una sono contenuti i due programmi di test descritti nell'articolo e mostrati nel filmato: *Esplora_test_TFT.ino, EsploraTest_new.ino*, nella seconda cartella, sono presenti i dieci files immagine da salvare nella memoria SD.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/programmiamo-scheda-arduino-esplora

Rendiamo autonoma la scheda Arduino ESPLORA

di adrirobot

La scheda Arduino Esplora, per il suo funzionamento deve essere alimentata tramite la porta USB.



Come visibile nello schema, la linea XUSB collegata al pin **P1** della presa USB, è protetta dal fusibile resettabile F1 che, nel caso di un assorbimento superiore ai 500 mA, interromperà automaticamente la connessione finché non sarà rimosso il corto o il sovraccarico.

Il terminale GND è invece connesso al terminale **P5** della presa USB.

Contrariamente agli altri modelli di schede Arduino, non è previsto un connettore polarizzato connesso ad un circuito regolatore che possa fornire la tensione stabilizzata a +5V per il funzionamento della scheda.

In questo articolo vedremo invece come poter alimentare la scheda con una batteria da 3,7V di tipo **Li-Ion** connessa ad un circuito convertitore DC/DC che fornirà la tensione di 5V.

Altre informazioni sulla scheda possono essere trovate nell'articolo: Scopriamo la nuova scheda Arduino Esplora.

DESCRIZIONE

Per realizzare l'alimentazione a batteria della scheda **Arduino Esplora**, utilizzeremo questi componenti:

- Una batteria ricaricabile di tipo Li-Ion
- Un modulo caricabatteria
- Un modulo convertitore DC/DC del tipo Step Up
- Un deviatore a slitta unipolare
- Un fusibile resettabile da 500 mA
- Una mini breadboard
- Cavetti assortiti maschio/maschio per cablaggi su breadboard

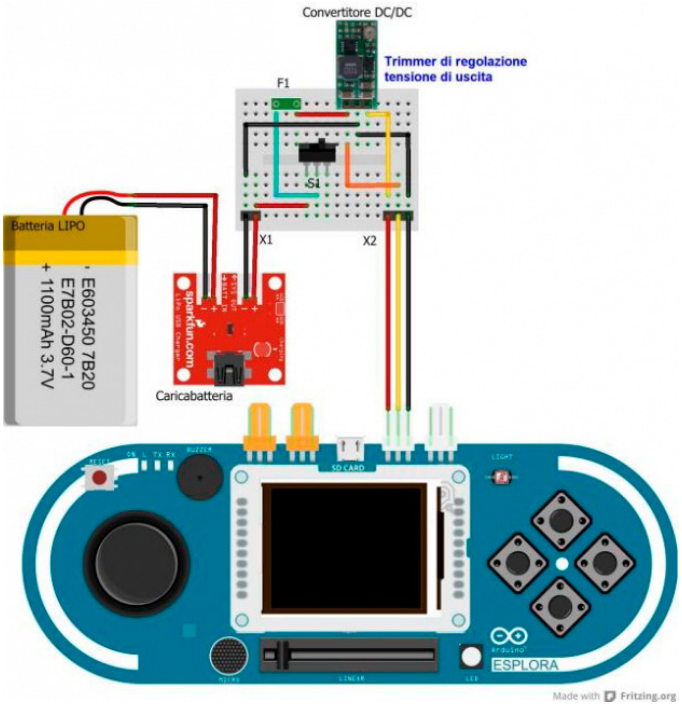
Il circuito è stato realizzato su breadboard, poiché la versione definitiva sarà poi integrata da una parte TX/RX basata su un modulo bluetooth che renderanno la scheda **Arduino Esplora** una stazione di comando completamente autonoma e completamente wireless.

SCHEMA DEL CIRCUITO

Lo schema del circuito, realizzato con il programma Fritzing è riportato sotto.

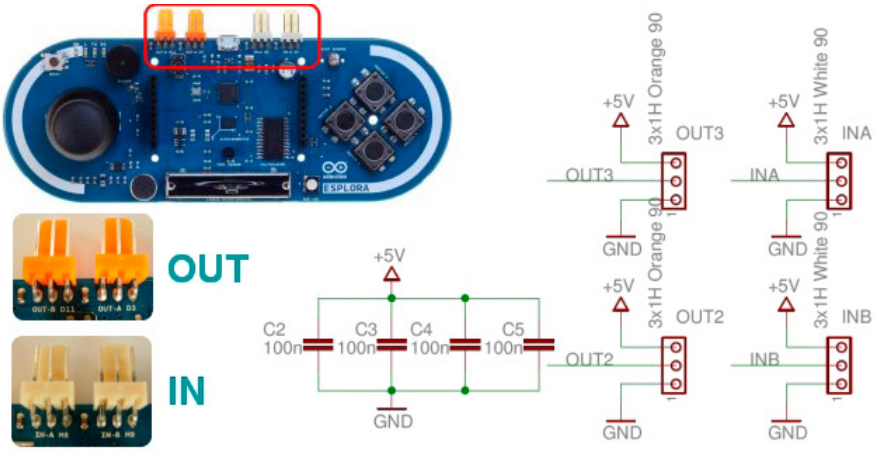
La batteria è connessa al caricabatteria e da qui la tensione tramite l'interruttore e il fusibile resettabile arriva al convertitore DC/DC che eleva il valore della tensione da 3,7V a 5V.

Per alimentare la scheda utilizzeremo uno dei connettori **Tinkerkit**, l'ingresso INA, i cui pin 5V



e GND sono connessi al resto del circuito.

La scheda ESPLORA ha quattro connettori compatibili con il sistema Tinkerkit: due per gli ingressi di colore bianco e due di uscita di colore arancio.



Utilizziamo anche il pin d'ingresso che in questo caso monitora il valore della tensione della batteria.

Passiamo adesso all'analisi in dettaglio dei componenti principali.

LA BATTERIA

La batteria ricaricabile scelta è un accumulatore agli ioni di litio (a volte abbreviato in Li-Ion).

Per maggiori informazioni si può fare riferimento all'articolo Batterie al Litio: scegliamo le migliori. Per questa applicazione viene utilizzata una batteria Li-Ion con una tensione di 3,7 e una corrente di 184 ma/h prodotta dalla ENIX Energies.

Il caricabatterie

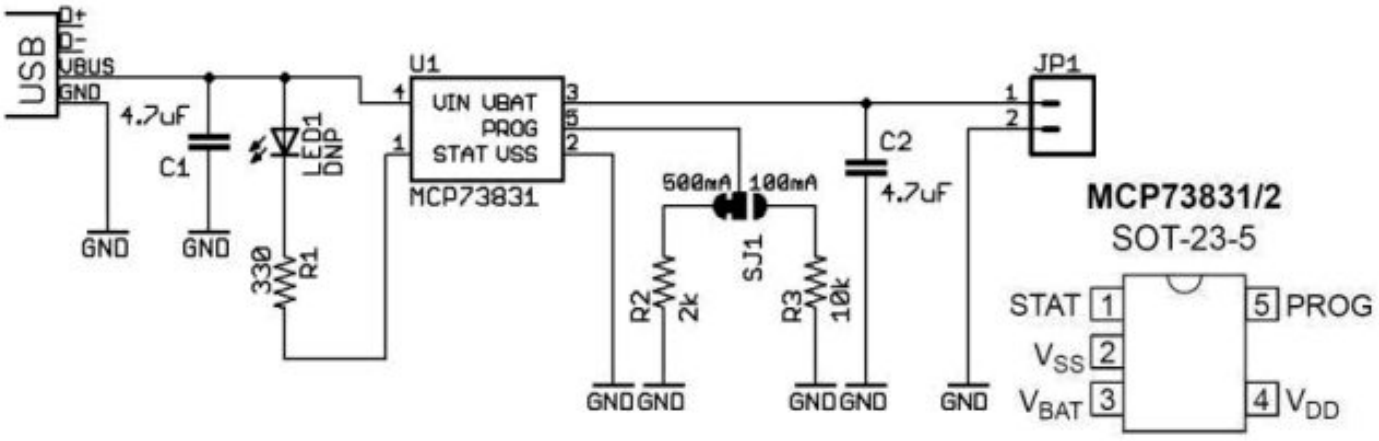
Per la carica della batteria Li-Ion, si è scelto un piccolo circuito prodotto dalla SparkFun codice PRT-10161.

Questo permette, utilizzando un alimentatore da 5V, oppure direttamente la presa USB del PC, di caricare la batteria con una corrente selezionabile tra 100 e 500 mA.

Con la stessa schedina è altresì possibile caricare batterie single-cell **LiPo** o **Li-Polymer**.

La scheda incorpora un circuito di carica basato sull'integrato prodotto dalla Microchip tipo MCP73831/2.

Il dispositivo è contenuto in un piccolo package SOT-23 a 5 pin e per il suo funzionamento sono necessari un basso numero di componenti esterni: è stato creato appositamente per quelle applicazioni portatili che necessitano di ricarica



da una porta USB.

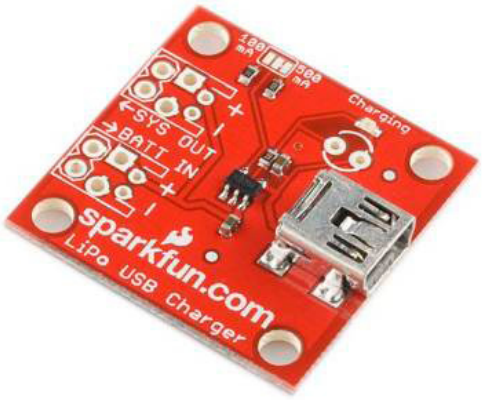
L'integrato utilizza un algoritmo di carica con corrente e tensione costante con precondizionamento e fine della carica selezionabile.

La tensione costante è fissata a quattro valori disponibili: 4.20V, 4.35V, 4.40V e 4.50V.

La corrente costante è impostata tramite il valore di una resistenza esterna.

L'integrato limita la corrente di carica in base alla sua temperatura e alle condizioni ambientali. Questa regolazione termica consente di ottimizzare il tempo del ciclo di carica pur mantenendo l'affidabilità del dispositivo.

Sono disponibili diverse opzioni, per impostarle si può fare riferimento al data sheet.



La scheda proposta dalla **SparkFun** ha un LED di stato, ed un jumper a saldare per selezionare il valore della corrente di carica a scelta tra 500mA o 100 mA.

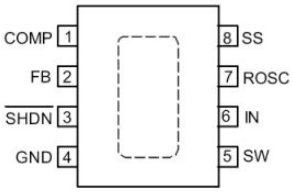
Dispone poi, della presa USB per l'alimentazione e di due connettori; una per collegare la batteria "**BAT IN**" e una di un'uscita "**SYS OUT**", che consente di collegare il circuito di carica direttamente al progetto in modo che non è necessario scollegare il caricabatterie ogni volta che si desidera utilizzarlo.

CONVERTITORE DC/DC

Per elevare la tensione dal valore di 3,7V fornita dalla batteria a quello di 5V necessario ad alimentare la scheda **Arduino Esplora**, la scelta è andata ad un piccolo convertitore DC/DC prodotto dalla Pololu con codice 791.

Ha dimensioni particolarmente ridotte (10,7 x 22,4 x 5,8 mm), ed è in grado di convertire una tensione continua compresa tra 1,5 e 16 volt in una tensione di uscita da 2,5 a 9,5 volt.

La massima corrente disponibile è di circa 1600 mA.



La scheda si basa su un integrato SC4501 prodotto dalla Semtech, un regolatore switching di

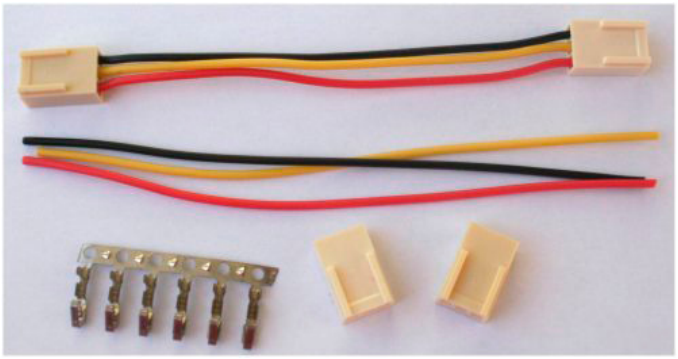
tipo step-up current-mode ad alta frequenza di commutazione con un transistor di potenza integrato da 2A. La sua frequenza di commutazione elevata (programmabile fino a 2 MHz) consente l'utilizzo di piccoli componenti passivi esterni a montaggio superficiale. Dispone di un Soft-start programmabile che elimina l'alta corrente di spunto in fase di avviamento.
La tensione di uscita può essere regolata ruotando il piccolo trimmer presente sulla scheda.

FUSIBILE AUTORIPRISTINANTE

Per la protezione del circuito si è utilizzato un **fusibile autoripristinante** chiamato PTC (polyfuse o polyswitch): si tratta di termisistori non-lineari che nel caso del loro surriscaldamento (dovuto ad un elevato assorbimento) diventano non conduttori, ritornando ad esserlo dopo la rimozione del problema e consentendo al circuito di funzionare di nuovo senza aprire il telaio o sostituire nulla.

ASSEMBLAGGIO

Come già detto, attualmente il circuito è stato assemblato su una basetta bread-board, ma in futuro è prevista la costruzione di un apposito circuito stampato.
Per il cablaggio occorre riferirsi allo schema riportato precedentemente nell'articolo.
Per il collegamento con la scheda occorre realizzare un apposito cavo utilizzando due connettori a tre pin femmina passo 2,54 mm tipo Molex 22-1-3037 (codice RS 679-5375P).

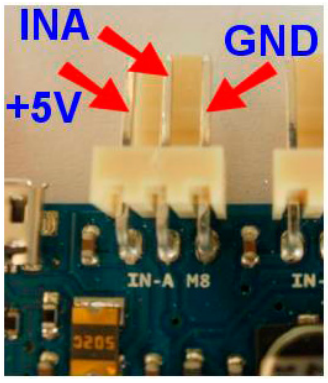


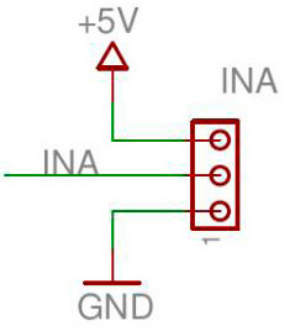
TARATURA E COLLAUDO

L'unica regolazione da compiere è quella della tensione di uscita del modulo switching, per cui con la batteria collegata e l'interruttore su ON, si collegherà un multimetro digitale all'uscita e si regolerà tramite un piccolo cacciavite il trimmer presente sul modulo elevatore sino a leggere il valore di 5V.
Si potrà a questo punto, considerare conclusa la taratura del circuito.

PROGRAMMA DI TEST

Per verificare il funzionamento del circuito, è presente un piccolo programma che mostrerà sul display TTF il valore fornito dalla batteria.
Nel programma, la sezione inerente la lettura del pin Analogico è derivata da un programma scritto da Mike Barela che potrete trovare a questo link.
L'operazione preliminare sarà quella di eseguire un collegamento, in modo da portare la tensione tramite un cavo al pin centrale del connettore **IN-A**.





Per caricare il programma, che è presente in allegato, utilizzeremo la scheda **Arduino Esplora** collegata normalmente tramite il solo cavo USB alla porta del PC.
Se non avete ancora configurato la scheda potrete leggere l'articolo Programmiamo la scheda Arduino Esplora.

```
// Include le librerie necessarie
#include <Esplora.h>
#include <TFT.h>      // Libreria per LCD ArduinoLCD library
#include <SPI.h>

//Definizione variabile di servizio
int BatteryValue = 0;
char tensione1 [50];
String BatteryValue1;
char batteryPrintout[5];
char valuePrintout[5];
float fBattVoltage = 0;
int iBattVoltage = 0;

/Programma derivato dal sito di Mike Barela per la lettura delle porte analogiche
// http://21stdigitalhome.blogspot.it/2013/03/reading-arduino-esplora-tinkerkit-inputs.html

const byte CH_TINKERKIT_INA = 8;   // Add values missing from Esplora.
const byte CH_TINKERKIT_INB = 9;
const byte INPUT_A          = 0;
const byte INPUT_B          = 1;
unsigned int readTinkerkitInput(byte whichInput) {  // return 0-1023 from Tinkerkit Input A or B
return readChannel(whichInput+CH_TINKERKIT_INA);
}   //   as defined above
unsigned int readChannel(byte channel) {       // as Esplora.readChannel is a private function

digitalWrite(A0, (channel & 1) ? HIGH : LOW); //  we declare our own as a hack
digitalWrite(A1, (channel & 2) ? HIGH : LOW); //
digitalWrite(A2, (channel & 4) ? HIGH : LOW); // digitalWrite sets address lines for the input
digitalWrite(A3, (channel & 8) ? HIGH : LOW);
return analogRead(A4);                    // analogRead gets value from MUX chip
}
void setup() {
   Serial.begin(9600);
  // Inizializzazione del display grafico
  EsploraTFT.begin();

  // cancella lo schermo con uno sfondo nero
  EsploraTFT.background(0,0,0);
  // Imposta il colore del testo a viola
  EsploraTFT.stroke(255,0,255);
  // Imposta altezza testo a 2
  EsploraTFT.setTextSize(2);
  // Scritturadel testo nella parte superiore sinistra dello schermo
  // Questo testo rimarra' statico
  EsploraTFT.text("Tensione: \n ",0,0);

  // Imposta altezza testo per il loop a 5
  EsploraTFT.setTextSize(5);
}

void loop() {
  // lettura valore della tensione e inserimeto valore in
  // una variabile
  unsigned int BatteryValue= readTinkerkitInput(INPUT_A);
```

```
  // Calcolo del valore della tensione della batteria
  float fBattVoltage = ((BatteryValue*5.0)/1023)*1000.0;

  iBattVoltage = (int)fBattVoltage; // Conversione del valore float in integer
  char battVoltageStr[50]; //Asseganzione a char
  sprintf(battVoltageStr,"%d",iBattVoltage); // Copia intero char come stringa
  // Utilizzando gli argomenti indicizzati viene messo ogni numero al posto giusto
  // nota il punto decimale e' inserito manualmente.
  sprintf(tensione1,"%c.%c%cV",battVoltageStr[0],battVoltageStr[1],battVoltageStr[2]);

// Conversione valore letto da porta analogica in stringa
  BatteryValue1= String (BatteryValue);
  BatteryValue1.toCharArray (valuePrintout, 6);

// imposta il colore del testo a rosso
  EsploraTFT.stroke(255,0,0);
  // stampa su LCD valore letto e di quello in volt
  EsploraTFT.text(tensione1, 0, 80);
  // imposta il colore del testo a verde
  EsploraTFT.stroke(0,255,0);
  // stampa su LCD valore letto dall' ADC
  EsploraTFT.text(valuePrintout, 0, 30);

 //Pausa di un secondo per la prossima lettura
  delay(1000);

// cancella il testo per la prossima lettura
  EsploraTFT.stroke(0,0,0);
  EsploraTFT.text(tensione1, 0, 80);
  EsploraTFT.text(valuePrintout, 0, 30);
}
```

FILMATO ILLUSTRATIVO

https://www.youtube.com/watch?v=SGUgKfM2XAU

CONCLUSIONI

Nella prossima puntata vedremo come collegare un modulo bluetooth alla scheda e con questo comanderemo un dispositivo connesso ad una scheda **Arduino UNO**.

Per cui, non perdetevi la prossima puntata.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/rendiamo-autonoma-scheda-arduino-esplora

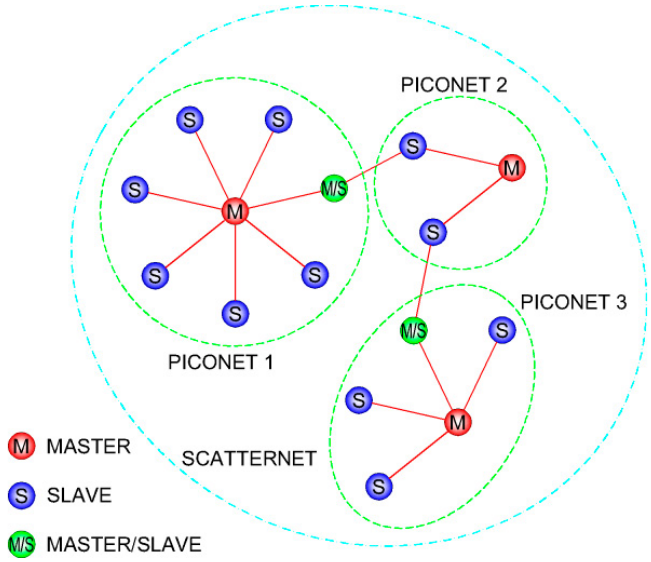
Dotiamo l'Arduino Esplora dell'interfaccia Bluetooth

di adrirobot

La scheda Arduino Esplora non dispone di interfacce di comando remoto, vedremo come dotarla di un modulo bluetooth.

BREVE DESCRIZIONE DELLA TECNOLOGIA BLUETOOTH

Il Bluetooth è una delle tecnologie che ha sostituito la necessità di collegamento via cavo, permettendo lo scambio di dati su brevi distanze.
Molti sono i dispositivi che ne sono attualmente dotati, a partire dai telefoni cellulari, alle stampanti e molti altri ancora.
La trasmissione utilizza la porzione inferiore dello spettro radio EM con frequenza di 2,45 GHz.
Il sistema può connettere un'unità master con un massimo di altre sette unità, denominate slave, formando un **PAN** (**P**ersonal **A**rea **N**etwork) chiamato **Piconet**.
Otto di questi gruppi possono essere collegati insieme, sempre per mezzo della tecnologia Bluetooth, per formare un network ancora più grande chiamato Scatternet.



In teoria, non vi è un limite al numero di unità che possono essere collegate in questo modo via Bluetooth.
Ogni dispositivo Bluetooth è dotato di un indirizzo univoco a livello mondiale, assegnato dal costruttore.
Equivalente del **MAC address** nelle reti ethernet, è rappresentato da **48 bit**, il che permette di avere fino a $2{,}84\times10^{14}$ dispositivi.
Ad ogni dispositivo può anche essere assegnato un nome che può essere una stringa di caratteri unicode, definito dall'utente che non è univoco, ma che facilita il riconoscimento del dispositivo.

NATURA MASTER-SLAVE DELLA COMUNICAZIONE

Come abbiamo visto, il collegamento bluetooth prevede un'unità **Master** e un'unità **Slave**, che devono seguire alcune regole:

- Chi inizia una conversazione funge da Master
- La sequenza dei canali utilizzati è derivata dall'indirizzo MAC del Master
- Il master invia, durante uno slot, una richiesta ad uno slave, che gli risponde nello slot successivo (sul prossimo canale della sequenza pseudocasuale)
- Possibilità di avere fino a 7 slave attivi
- Master e slave devono mantenere il clock allineato

PAGING E INQUIRY

Un dispositivo che intende iniziare l'invio di dati,

deve inviare dei pacchetti di richiesta riportanti l'indirizzo del destinatario.
Se il destinatario è in ascolto, può sincronizzarsi al master e accettare la comunicazione. L'invio di richieste si chiama "**paging**", la disponibilità del destinatario a ricevere richieste di connessione di chiama "**page scan**". La connessione richiede la conoscenza dell'indirizzo MAC del destinatario.
In alcuni casi per scoprire i possibili destinatari, una stazione può inviare pacchetti di "**Inquiry**": chi riceve tale pacchetto, risponde indicando il proprio indirizzo MAC, il proprio clock ed un intero indicante la tipologia del dispositivo (Class Of Device).
Per ricevere richieste di paging e di inquiry, mittente e destinatario devono utilizzare lo stesso canale radio.
Per ulteriori informazioni vedere l'articolo **Bluetooth come funziona**

COMPONENTI DEL PROGETTO

Dopo la teoria sulla comunicazione bluetooth, passiamo ora ad analizzare i principali componenti che compongono il progetto:

- Scheda Arduino Esplora
- Due Moduli Bluetooth BlueSMiRF Silver
- Arduino UNO

Per i componenti del circuito di alimentazione, fate riferimento all'articolo **Rendiamo autonoma la scheda Arduino Esplora**

SCHEDA ARDUINO ESPLORA

Per una descrizione accurata della scheda si può fare riferimento al precedente articolo **Scopriamo la nuova scheda Arduino Esplora**
Mentre per la sua programmazione poteve leggere l'articolo **Programmiamo la scheda Arduino Esplora**

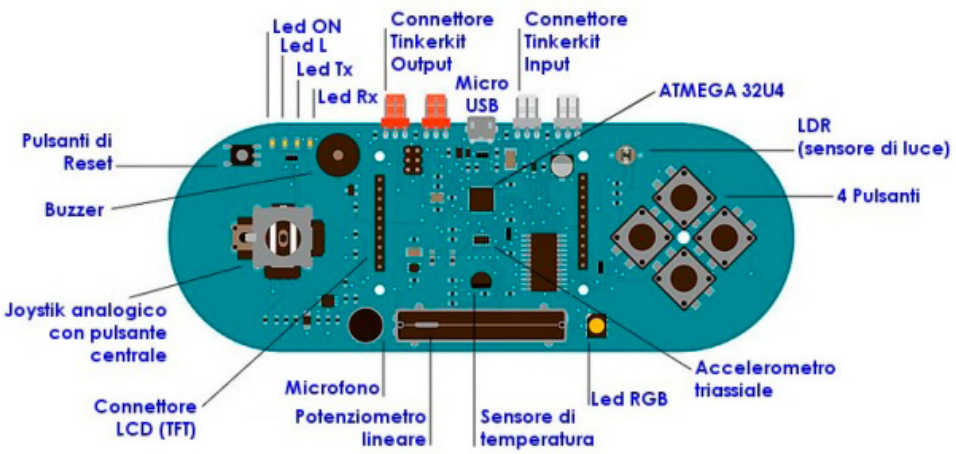
Le principali caratteristiche della scheda sono:

- Processore ATMEGA32U4 con bus a 8 bit prodotto dalla Atmel con architettura di tipo RISC, Velocità di clock 16 MHz, memoria Flash da 32 KB di cui 4 KB utilizzati dal bootloader, memoria EEPROM da 1KB, 20 porte Digital pin I/O, 12 Canali di ingresso analogici, 7 canali PWM, tensione di funzionamento 5V, 1 porta USB Full speed
- Un joystick analogico a due assi (X e Y) con pulsante centrale
- 4 pulsanti disposti a rombo
- Un potenziometro lineare a cursore
- Un microfono per rilevare il rumore ambientale
- Un sensore di luce per la misurazione dell'intensità luminosa
- Un sensore per la misurazione della temperatura ambiente
- Un accelerometro triassiale (X, Y e Z)
- Un buzzer per produrre suoni
- Un led luminoso a LED tipo RGB con elementi Rosso, Verde, Blu
- 2 Ingressi per collegare i moduli sensore della serie Tinkerkit
- 2 uscite per collegare i moduli attuatori della serie Tinkerkit
- Un connettore per l'inserimento del display TFT a colori, dotato di uno slot per scheda SD

ARDUINO UNO

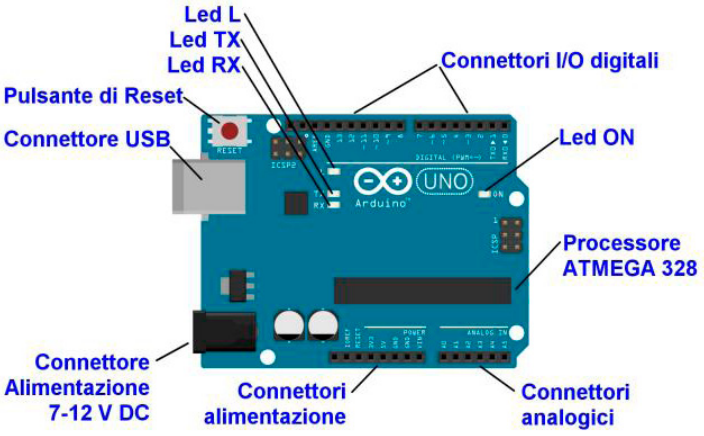
La **scheda Arduino UNO** è basata sul processore **ATMEGA328** con bus a 8 bit prodotto dalla Atmel con architettura di tipo RISC, le sue principali caratteristiche sono:

- Memoria flash da 32KB ISP con possibilità di lettura/scrittura



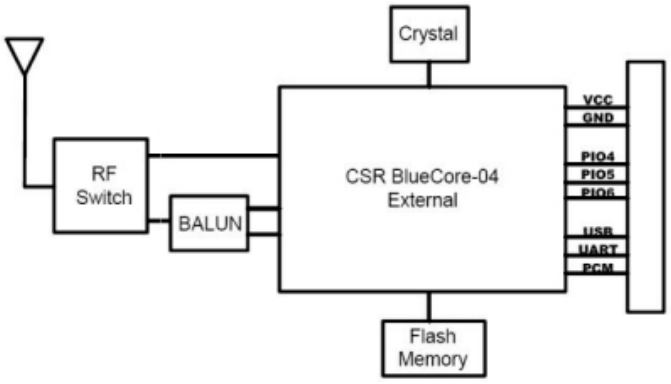
- Memoria EEPROM da 1KB
- Memoria SRAM da 2KB.
- 23 porte general purpose di I/O
- 32 registri di lavoro general purpose
- Tre flessibili timer / contatori con modalità di confronto
- Possibilità d'interrupts esterni e interni
- Una seriale USART programmabile
- Un'interfaccia seriale a 2 fili compatibile I2C
- Una porta SPI seriale
- 6 convertitori A/D a 10-bit
- Timer watchdog programmabile con oscillatore interno
- 5 modalità di risparmio energetico selezionabili via software
- Tensione di funzionamento compresa tra 1,8-5,5 volt.

La scheda presenta dei pin strip, dove sono riportate l'alimentazione e le porte di I/O.

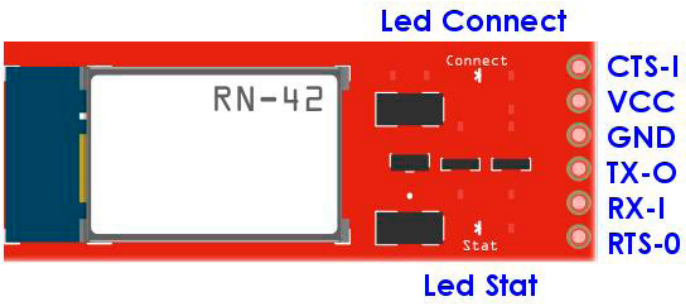


MODULOBLUETOOTH BLUESMIRF SILVER

Il modulo **BlueSMiRF Silver** è un modem wireless Bluetooth prodotto dalla **SparkFun**, basato sull'integrato **RN-42** prodotto dalla **ROVING Networks** acquisita recentemente dalla **Microchip**.



Schema a blocchi del modulo RN-42



Alcune specifiche tecniche:

- Approvazione FCC Classe 2 Bluetooth Modem Radio
- Piccole dimensioni - 51.5x15.8x5.6mm
- Collegamento molto robusto sia in integrità che come distanza di trasmissione (18m)
- Hardy frequency hopping scheme - opera in ambienti RF come il Wi-Fi, 802.11g, e Zigbee
- Connessione crittografata
- Frequenza: 2.4 ~ 2,524 GHz

- Tensione di funzionamento: 3.3V-6V
- Comunicazione seriale: 2400-115200 bps
- Temperatura di funzionamento: -40 ~ +70 C
- Antenna integrata

Questo modem funziona come una porta seriale (RX/TX), potendo trasmettere un flusso seriale da 2400 a 115200 bps.

L'unità può essere alimentata con una tensione i cui valori possono essere compresi tra i 3,3 V fino a 6V, in quanto è dotato di un regolatore di tensione tipo **MIC5205** che fornisce in uscita i 3.3V per il funzionamento dell'integrato.

Tutti i pin del segnale sono 3V-6V tolleranti, in quanto sulla scheda sono già presenti i circuiti per adattare i livelli di tensione formati da MOSFET (N-Channel) di tipo SMD modello **BSS138**.

Il modulo presenta una pin strip a 6 pin in cui sono presenti i segnali: **RX**, **TX**, **CTS**, **RTS**, **VCC** e **GND**.

Sono inoltre, presenti due led indicatori: il led di colore rosso (**Stat**) segnala che il modulo è alimentato, ma non connesso; il led di colore verde (**Connect**) segnala che è stata stabilita una connessione.

COLLEGAMENTI STAZIONE DI CONTROLLO

Nello schema seguente (**vedere allegato** realizzato con il **programma Fritzing**), sono indicati i collegamenti da compiere. Questi comportano una modifica di quelli previsti nel precedente articolo **Rendiamo autonoma la scheda Arduino Esplora**.

Per cui occorre montare, oltre ai componenti descritti nel precedente articolo, il modulo bluetooth.

Ecco l'elenco completo dei componenti necessari:

- Scheda Arduino Esplora
- Display LCD TFT
- Una Mini Breadboard 17x10
- Una batteria di tipo Li-Ion
- Un modulo caricabatteria SparkFun codice PRT-10161
- Un modulo convertitore DC/DC del tipo Step Up dalla Pololu con codice 791
- Modulo Bluetooth BlueSMiRF Silver SparkFun codice WRL-10269
- Un interruttore
- Un fusibile resettabile da 500 mA
- Due cavi realizzati utilizzando dei connettori a tre pin femmina passo 2,54 mm tipo Molex 22-1-3037 (codice RS 679-5375P)
- Cavi colorati per breadboard

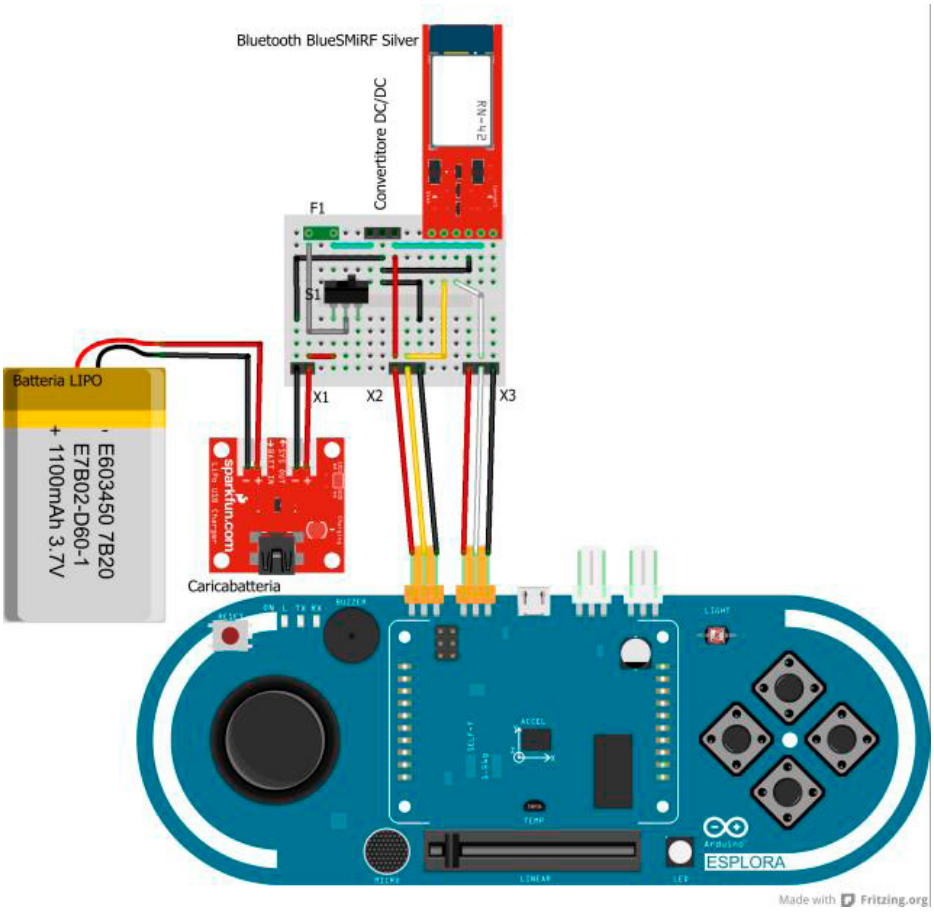
Il collegamento viene effettuato utilizzando i due connettori presenti sulla scheda collegati ai pin digitali D3 e D11.

COLLEGAMENTI STAZIONE RICEVENTE

Nello schema seguente (vedere allegato), sono indicati i collegamenti da effettuare per il test della stazione ricevente.

Ecco l'elenco completo dei componenti necessari:

- Scheda Arduino UNO
- Una Mini Breadboard 17x10
- Modulo Bluetooth BlueSMiRF Silver SparkFun codice WRL-10269
- un led
- Cavi colorati per breadboard



Schema di collegamento

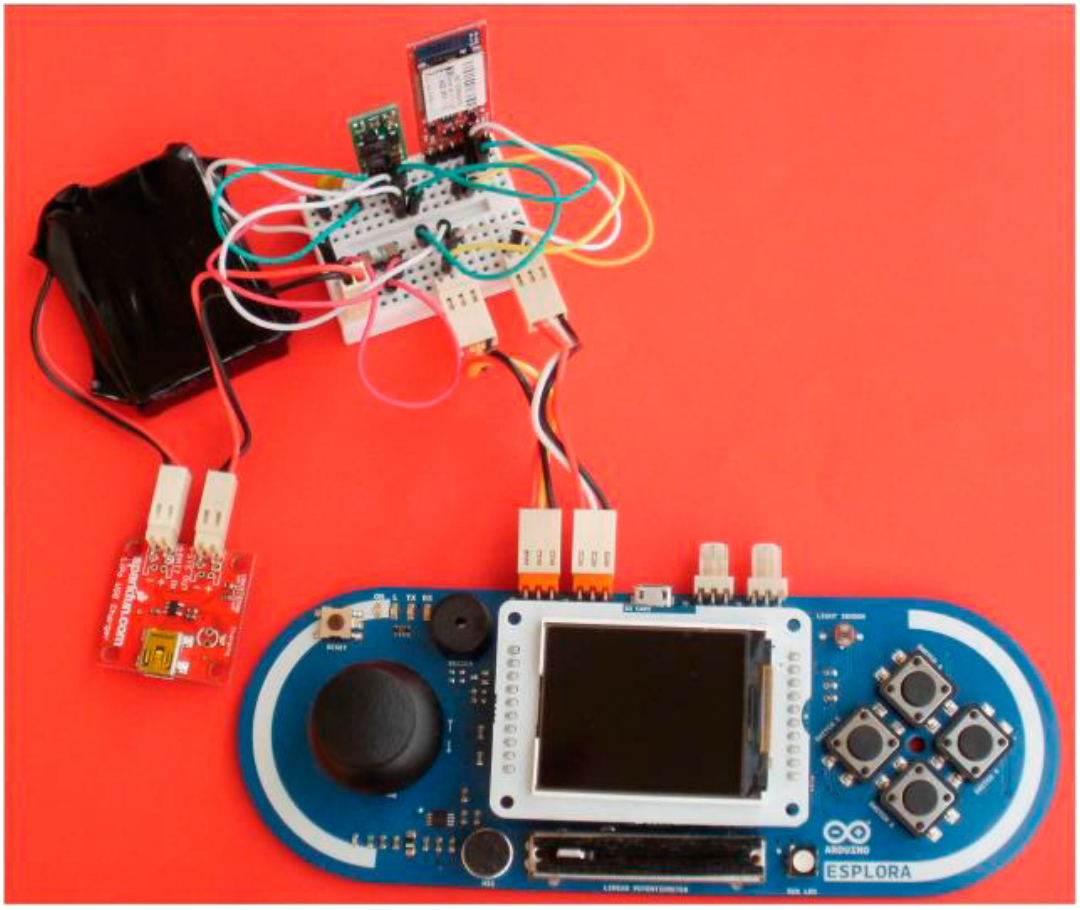
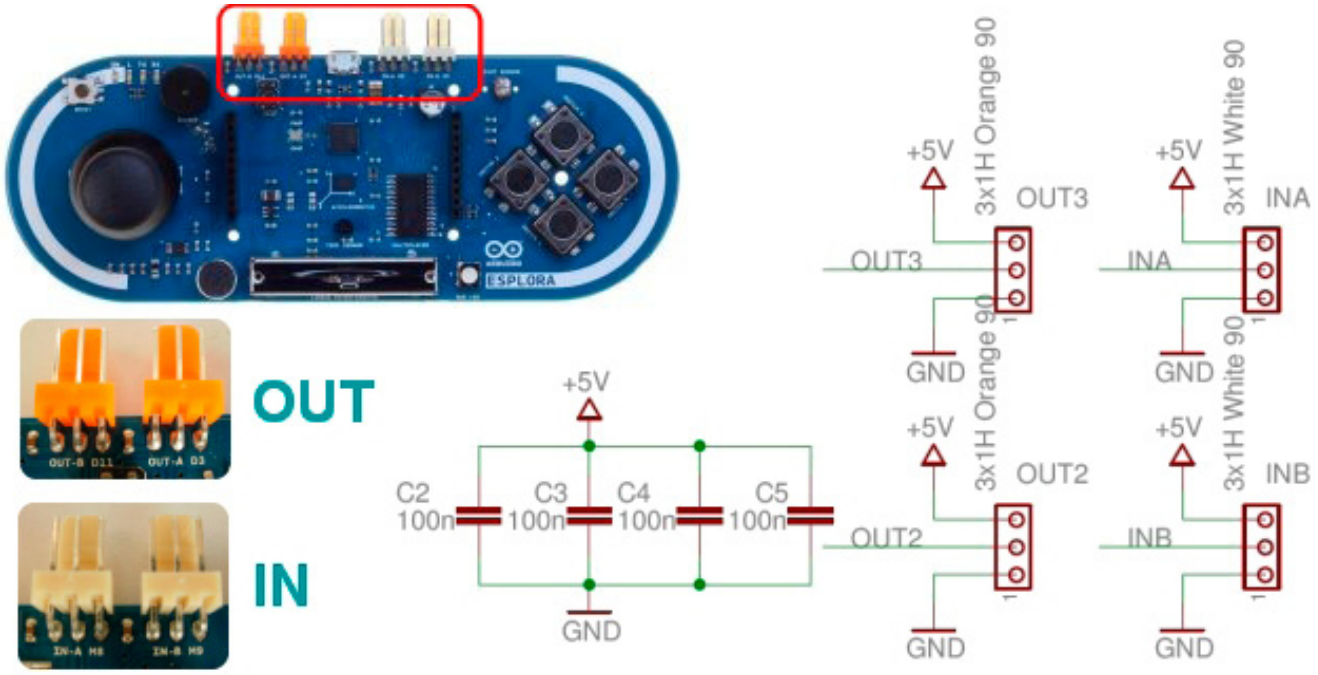
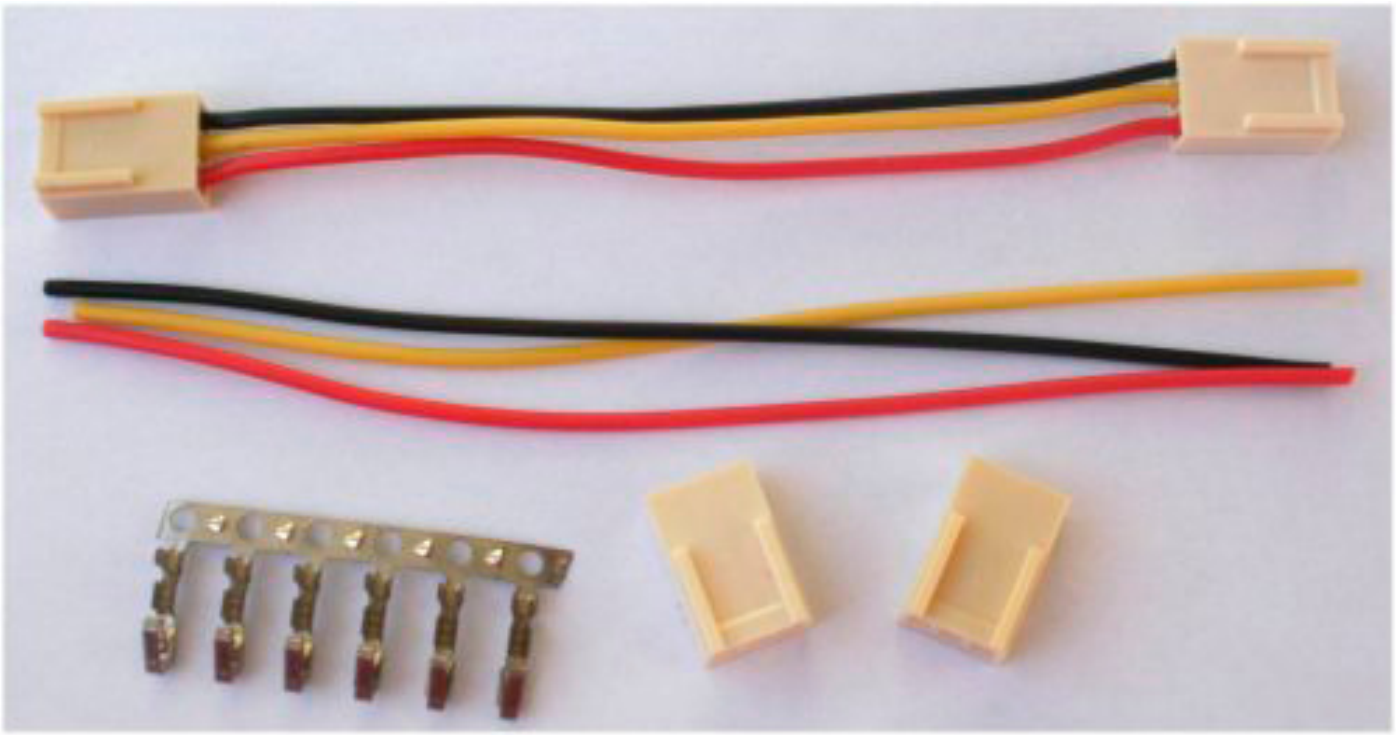


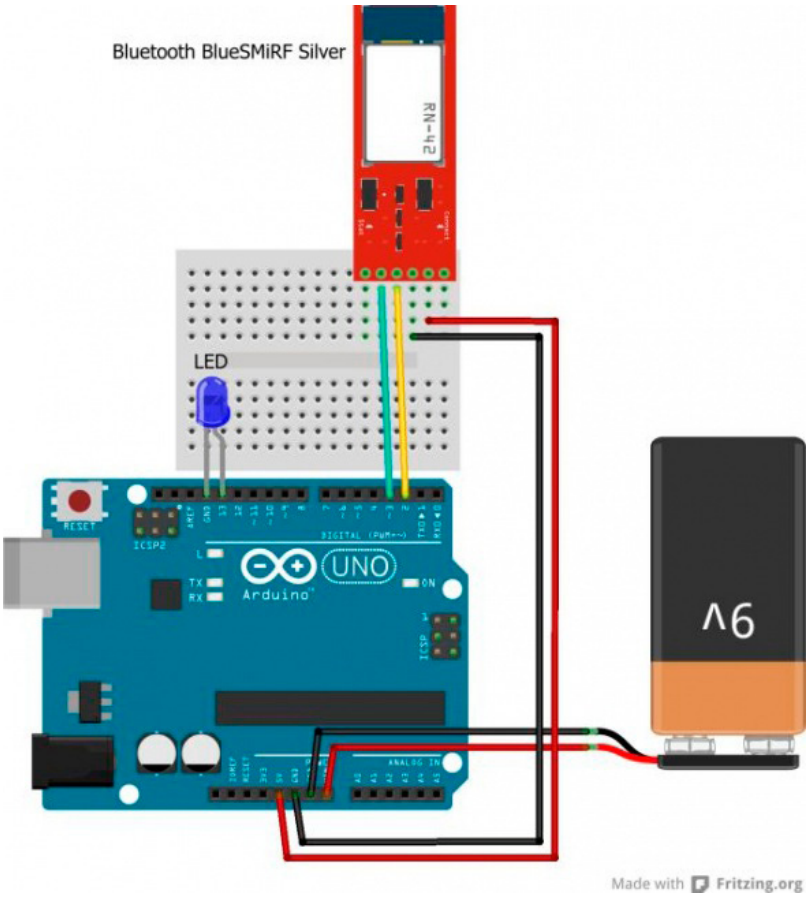
Foto del collegamento realizzato



Connettori di IN/OUT della scheda Esplora



Cavo di collegamento con scheda Arduino Esplora



Schema di collegamento

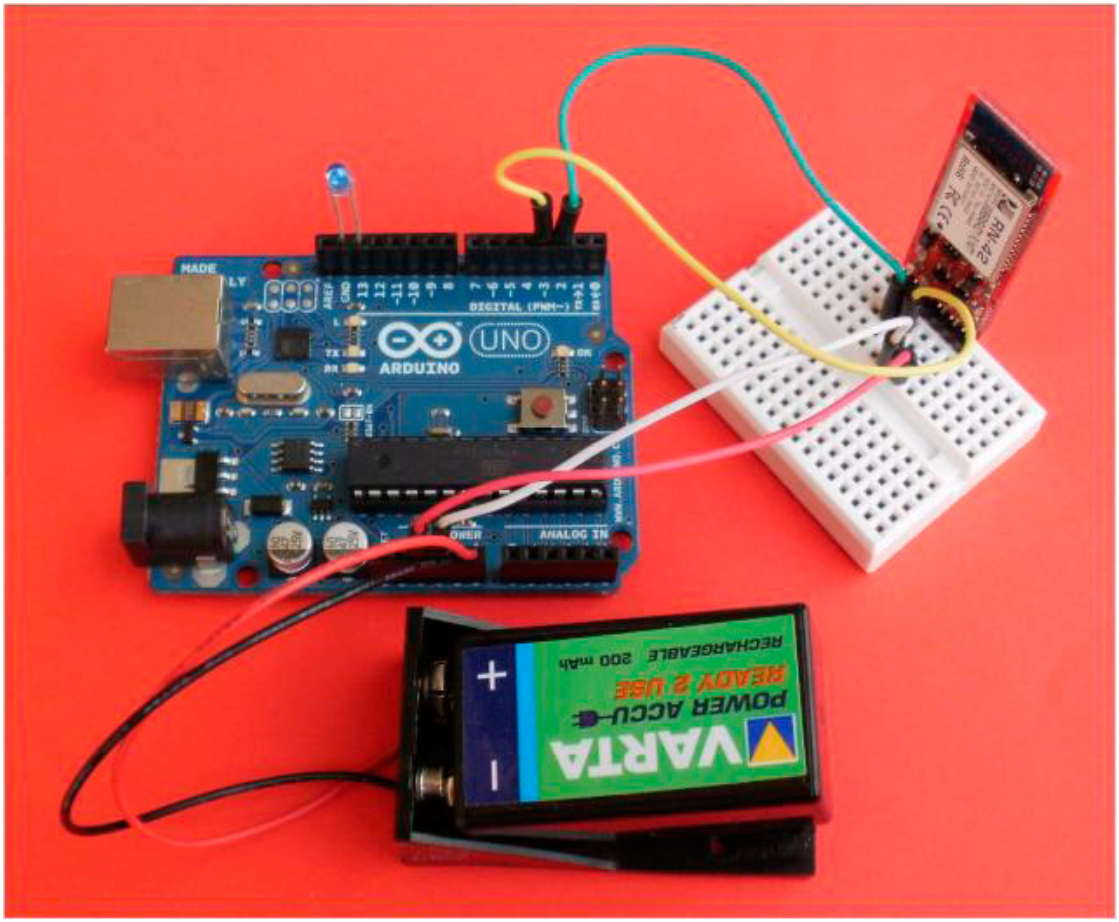


Foto del collegamento realizzato

PROGRAMMA MASTER

Passiamo ora ad analizzare il programma da inserire sulla Scheda **Arduino Esplora** che eseguirà le seguenti funzioni:

- Caricamento delle librerie necessarie: Software Serial, SPI, Esplora TFT
- Definizione dei pin collegati alla porta seriale (nel nostro caso i pin 11 e 3)
- Impostazione del pin di collegamento al led presente sulla scheda (il pin 13)

Nel dettaglio nel ciclo di **setup**, abbiamo l'impostazione del modulo bluetooth come **Master**.

Per fare questo, dopo aver impostato la velocità di trasmissione della seriale a **115200 bbs** (che è quella di default del modulo), viene inviato il comando per entrare in **Command Mode** rappresentato da **print ("$$$")** a cui deve seguire una breve pausa.

Si invia il comando **println("SM,1")**, che imposta il modulo come **Master**. A questo punto indichiamo a quale modulo deve connettersi mediante il comando **println("C,<address>)** dove **<address>** deve essere sostituito dal codice MAC del vostro modulo slave, questo numero si trova scritto sull'esterno del modulo, ed è formato da 12 cifre.

Dopo una nuova pausa, trasmettiamo il comando **println("---")**, che permette l'uscita dal **Command Mode**.

Sotto le righe di codice corrispondenti:

```
mySerial.begin(115200);
 mySerial.print("$$$");
 delay(100);
 mySerial.println("SM,1");
 delay(100);
 //inserire il codice MAC del modulo Slave
 mySerial.println("C,000666464321");
 delay(100);
 mySerial.println("---");
```

Segue, una parte di programma per l'impostazione del **display TFT**.

Si passa quindi al programma vero e proprio presente nel **ciclo LOOP**.

Questo legge ciclicamente i tasti impostati (**2 e 4**) e a seconda di quello premuto, accede o spegne, il led presente sulla scheda (connesso al pin I/O 13), inviando poi tramite la seriale (utilizzando il pin I/O 3) il carattere "**a**" o "**s**", in contemporanea invia un Feedback al **display TTF**.

Segue il **listato completo** che è presente negli allegati.

PROGRAMMA SLAVE

Il programma da inserire sulla **Scheda ARDUINO UNO** esegue le seguenti funzioni:

- Caricamento della libreria necessaria: Software Serial
- Definizione dei pin collegati alla porta seriale (nel nostro caso i pin 2 e 3)
- Impostazione del pin di collegamento al led presente sulla scheda (il pin 13)

```
/*
Programma di test per interfaccia SERIALE
 Sezione MASTER - Arduino esplora
 * RX è il pin digitale 11 (da collegarsi al TX del modulo BlueSMiRF)
 * TX è il pin digitale 3 (da collegarsi al RX del modulo BlueSMiRF)
 Questo codice di esempio è di dominio pubblico.
 */
#include <SoftwareSerial.h>
#include <SPI.h>
#include <Esplora.h>
#include <TFT.h>
SoftwareSerial mySerial(11, 3); // RX, TX
int led = 13; // il Pin 13 è collegato al led L sulla scheda esplora
void setup()
{
  //Imposta il modulo BlueSMiRF SILVER come MASTER
  //e setta il numera MAC del ricevente slave
  mySerial.begin(115200);
  mySerial.print("$$$");
  delay(100);
  mySerial.println("SM,1");
  delay(100);
  //inserire il codice MAC del modulo Slave
  mySerial.println("C,000666464321");
  delay(100);
  mySerial.println("---");
  //------------------
  pinMode(led, OUTPUT);  // initialize il pin digitale come output
  EsploraTFT.begin();
```

```
  EsploraTFT.background(0,0,0); //sfondo nero
  EsploraTFT.setTextSize(2);
  EsploraTFT.stroke(0,255,0); //colore del testo verde
  EsploraTFT.text("   Test uso",0,0);
  EsploraTFT.text("    Modulo",0,16);
  EsploraTFT.text("  Bluetooth",0,32);
  EsploraTFT.setTextSize(1);
  EsploraTFT.stroke(255,255,255); //colore del testo bianco
  EsploraTFT.text("  2-Accende  -  4 Spegne",0,60);
  delay(500);
}
void loop() // LOOP di lettura tasti e invio carattere
{
  int button_2 = Esplora.readButton(SWITCH_2);
  int button_4 = Esplora.readButton(SWITCH_4);
  if (button_2==LOW) {
    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
    mySerial.print ("a");    //Accende led
    cancellazione ();
    EsploraTFT.stroke(255,0,0); //colore del testo rosso
    EsploraTFT.text("LED ACCESO",30,80);
    EsploraTFT.stroke(255,0,0);
    EsploraTFT.fill(255,0,0);
    EsploraTFT.circle(105,83,10);
  }
  if (button_4==LOW){
    digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
    mySerial.print ("s");   //Spegne led
    cancellazione ();
```

```
    EsploraTFT.stroke(255,255,255); //colore del testo bianco
    EsploraTFT.text("LED SPENTO",30,80);
    EsploraTFT.stroke(255,0,0); // Colore rosso
    EsploraTFT.circle(105,83,10); // Disegno cerchio
  }
  delay (100);
}
void cancellazione ()
{
  EsploraTFT.stroke(0,0,0);
  EsploraTFT.fill(0,0,0);
  EsploraTFT.rect(30,80, 80, 8); //Cancellazione valore
}
```

Nel ciclo di setup abbiamo: prima di tutto l'impostazione del modulo bluetooth come **SLAVE**, la procedura è analoga a quella precedentemente descritta, l'unica variazione è che il comando in questo caso sarà:

```
println("SM,0");
```

che imposta appunto il modulo come **Slave**.

Si passa quindi al programma vero e proprio presente nel **ciclo LOOP**: questo legge ciclicamente la porta seriale connessa al modulo bluetooth e quando riceve un carattere, lo analizza; se il carattere è **"a" accende il led**, mentre se il carattere è **"s"** lo **spegne**.

Segue il **listato completo** che è presente negli allegati.

COLLAUDO DEL DISPOSITIVO

Una volta caricati i relativi programmi nelle due schede, si potrà passare al collaudo del nostro collegamento wireless.

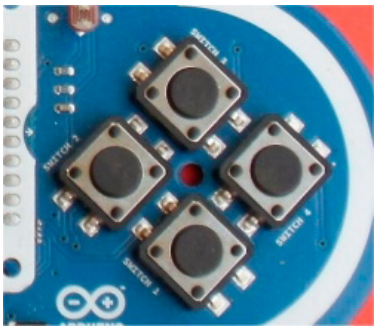
In questo caso, è necessario alimentare la scheda Arduino UNO (modulo slave) prima della scheda Arduino Esplora (modulo Master) con un anticipo di circa 5 secondi.

Se tutto funzionerà, dopo un lampeggio dei led rossi (Stat), si avrà l'accensione dei led verdi (Connect) che segnalerà l'avvenuta connessione dei due moduli.

A questo punto, premendo il tasto "**SWITCH 2**", si dovrà accendere il led presente sulla scheda Arduino UNO, mentre il tasto "**SWITCH 4**" lo farà spegnere.

```
/*Programma di test per interfaccia SERIALE
Riceve dalla seriale hardware, invia alla seriale del software.
Riceve daL software seriale , e invia al sesiale hardware.
Il circuito:
* TX è il pin digitale 2 (da collegarsi al RX del modulo BlueSMiRF)
* RX è il pin digitale 3 (da collegarsi al TX del modulo BlueSMiRF)
Questo codice di esempio è di dominio pubblico.
*/
#include <SoftwareSerial.h>
SoftwareSerial mySerial(3, 2); // RX, TX
const int ledPin =  13;      // the number of the LED pin
void setup()
{
 //Imposta il modulo BlueSMiRF SILVER come SLAVE
 mySerial.begin(115200);//imposta la porta di comunicazione con il modulo BlueSMiRF
 mySerial.print("$$$");
 delay(100);
 mySerial.println("SM,0");
 delay(100);
 mySerial.println("---");
 delay (1000);
 //------------------
 Serial.begin(115200); //imposta la porta di comunicazione seriale
 pinMode(ledPin, OUTPUT);// initialize the LED pin as an output:
}
void loop()
{
 while (mySerial.available() < 1) {
 }                    // Attesa sino a quando riceve un carattere
```

```
  char val = mySerial.read(); // Legge il carattere dal modulo Bluetooth
  Serial.println (val); //per test
  switch(val)       // Eseguie i comandi in base al carattere ricevuto
  {
  case 'a':// Se il carattere ricevuto è 'a' accende il led
    // turn LED on:
    digitalWrite(ledPin, HIGH);
    break;
  case 's'://Se il carattere ricevuto è 's' spegen il led
    // turn LED off:
    digitalWrite(ledPin, LOW);
    break;
   }
}
```



Tasti utilizzati dal programma

Le istruzione d'uso e lo stato del led sono riportate sullo schermo TTF della scheda Arduino esplora.

Display Arduino Esplora

FILMATO ILLUSTRATIVO

https://www.youtube.com/watch?v=sao3LBseguc

CONCLUSIONI

In quest'articolo, abbiamo visto com'è possibile creare un collegamento wireless tra la scheda Arduino Esplora e un'altra scheda Arduino.

Dalla prossima puntata vedremo come costru-

ire un piccolo robot che potremo comandare in modo wireless.

Nulla vieta comunque, di utilizzare questo sistema per ricevere da sensori esterni collegati alla scheda Arduino UNO , mostrando per esempio le informazioni sul display della Arduino Esplora.

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/dotiamo-larduino-esplora-dellinterfaccia-bluetooth

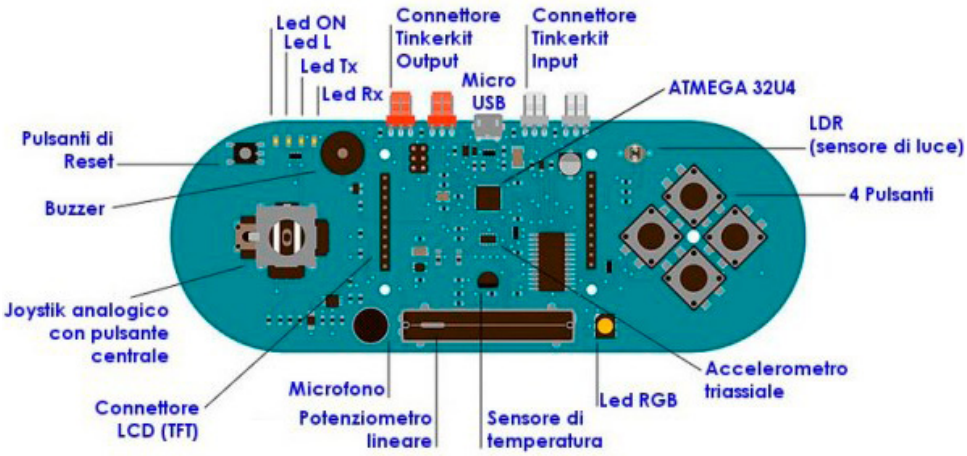
Gestione di un dispositivo Pan & Tilt con la scheda Arduino ESPLORA

di adrirobot

SCHEDA ARDUINO ESPLORA

In questo articolo spiegheremo come collegare dei servomotori alla **scheda Arduino Esplora**, della quale si è parlato più diffusamente nell'articolo **Scopriamo la nuova scheda Arduino Esplora**, in cui ne trovate una descrizione completa. La scheda Arduino Esplora ha le seguenti caratteristiche:

- Processore **ATMEGA32U4** con bus a 8 bit prodotto dalla Atmel con architettura di tipo RISC, Velocità di clock 16 MHz, memoria Flash da 32 kB di cui 4 kB utilizzati dal bootloader, memoria EEPROM da 1 kB, 20 porte Digital pin I / O, 12 Canali di ingresso analogici, 7 canali PWM, tensione di funzionamento 5V, 1 porta USB Full speed
- Un joystick analogico a due assi (X e Y) con pulsante centrale
- 4 pulsanti disposti a rombo
- Un potenziometro lineare a cursore
- Un microfono per rilevare il rumore ambientale.
- Un sensore di luce per la misurazione dell'intensità luminosa
- Un sensore per la misurazione della temperatura ambiente
- Un accelerometro triassiale (X, Y e Z)
- Un buzzer
- Un LED luminoso a LED tipo RGB con elementi Rosso Verde e Blu.
- 2 Ingressi per collegare i moduli sensore della serie Tinkerkit.
- 2 uscite per collegare i moduli attuatori della serie Tinkerkit.
- Un connettore per l'inserimento del display TFT a colori, dotato di uno slot per scheda SD

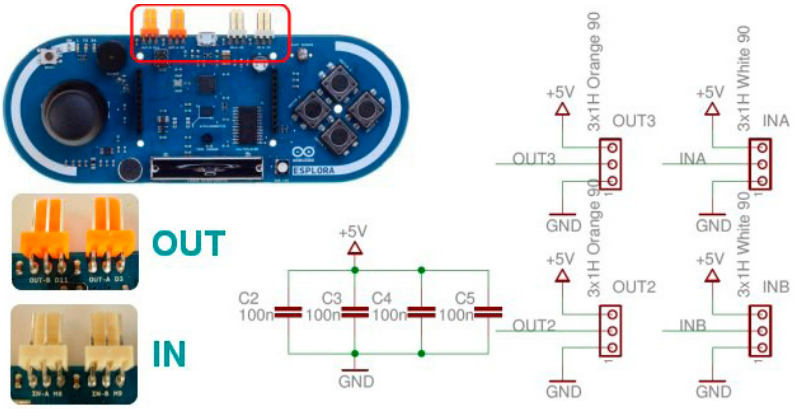


INGRESSI E USCITE TINKERKIT

Come abbiamo visto, la scheda Arduino Esplora ha quattro connettori compatibili con il sistema Tinkerkit, due ingressi di colore bianco e due di uscita di colore arancio.

Questi hanno tre terminali, due per l'alimentazione (+5V e GND) e uno per l'ingresso o l'uscita del segnale.

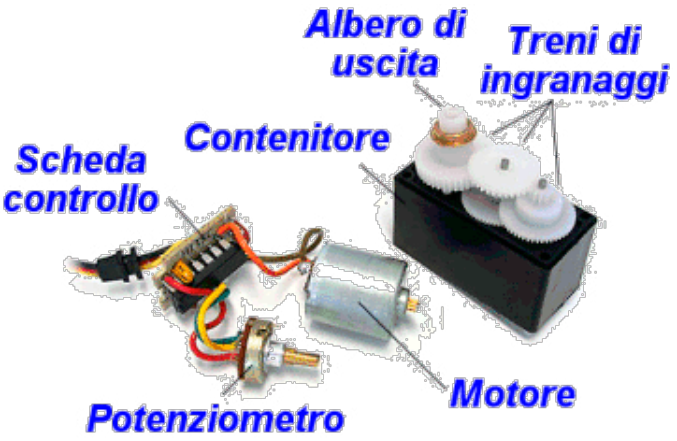
Per il nostro progetto utilizzeremo i due connettori di uscita colorati in Arancione, connessi alle porte D3 e D11.



QUALCHE CENNO INERENTE I SERVOMOTORI.

Il servomotore è un piccolo dispositivo che incorpora un motore DC, un treno di ingranaggi, un potenziometro, un circuito integrato e un albero di uscita.

Dei tre fili che sporgono dal corpo motore, uno è per l'alimentazione, uno è per la massa e uno è una linea di ingresso di controllo. L'albero del servo può essere posto a determinate posizioni angolari inviando un segnale codificato. Finché esiste il segnale codificato sulla linea di ingresso, il servo manterrà la posizione angolare dell'albero; se il segnale codificato cambia, si modifica la posizione angolare dell'albero.



Il circuito di controllo ha il compito di generare gli impulsi per controllare il servo.

Possono essere generati impulsi compresi tra 0.25 ms e 2.75 ms che coprono la richiesta della maggioranza dei tipi di servo per rotazioni oltre i 180 gradi.

Generalmente con un impulso di durata pari a 1.5 ms, il perno del servomotore si posiziona esattamente al centro del suo intervallo di rotazione; da questo punto il perno può ruotare fino a -90 gradi (senso antiorario) se l'impulso fornito ha una durata inferiore a 1.5 ms e fino +90 gradi (senso orario) se l'impulso fornito ha durata superiore a 1.5 ms. Il rapporto esatto tra la rotazione del perno e la larghezza dell'impulso fornito può variare tra i vari modelli di servo.

Se questi valori sono ripetuti con un intervallo non superiore a 20 ms, la posizione raggiunta sarà mantenuta.

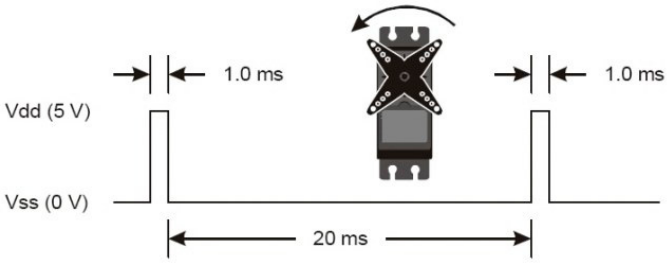


Diagramma temporizzazione per rotazione antioraria.

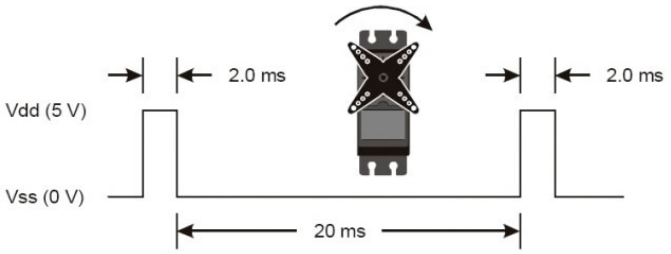


Diagramma temporizzazione per rotazione oraria.

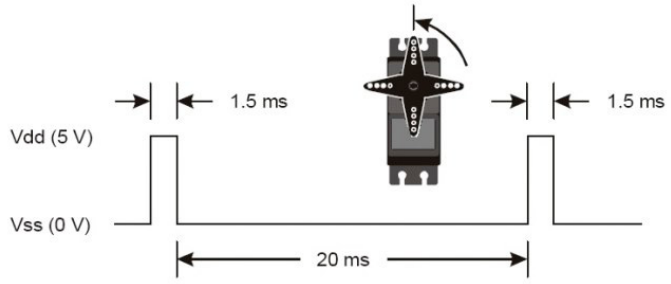
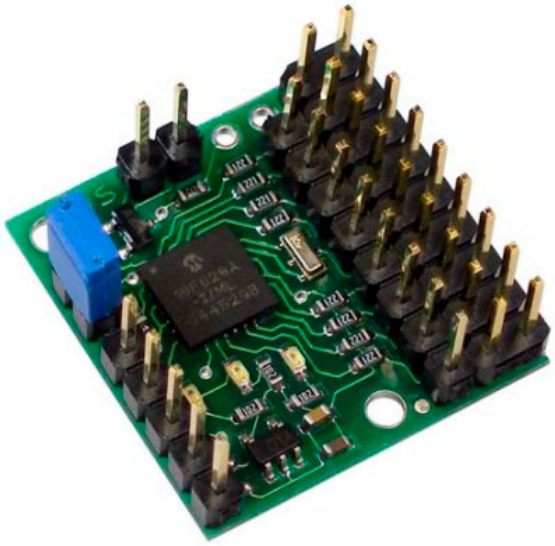


Diagramma temporizzazione per posizionamento al centro.

MICRO SERIAL SERVO CONTROLLER

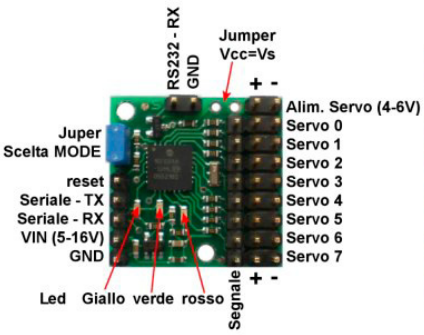
La prima scheda di controllo servo che analizziamo è denominata **Micro Servo Serial Controller** ed è prodotta dalla **Pololu**: si presenta come uno stampato di piccole dimensioni di poco più di due centimetri di lato.

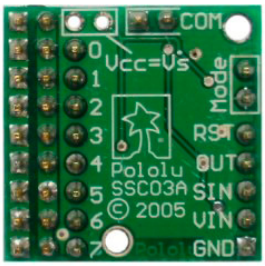


Su di essa trovano posto, oltre al processore a 8 bit con memoria flash tipo **PIC16F628A** della **Microchip**, il risuonatore per il clock, un regolatore di tensione, tre LED di stato e vari connettori con funzione di alimentazione di interfaccia e per il collegamento dei servomotori. La scheda può controllare fino a 8 servocomandi con trasmissione tramite una linea seriale.

Nonostante le sue minuscole dimensioni, è ricca di funzioni: può controllare la velocità e la posizione di ognuno degli 8 servi indipendentemente, la velocità della porta seriale è rilevata automaticamente nel range da 1200 a 38400 Baud ed ha 3 LED di stato.

I collegamenti del Servo Controller sono visibili nella foto di seguito, la maggior parte dei pin sono identificati sul retro della scheda del Servo Controller. Tutte le piazzole connesse a massa hanno forma quadra.





Specifiche Tecniche:

Dimensioni stampato	~ 23x23 mm
Numero di porte per servo	8
Range larghezza impulso	0.25-2.75 ms
Risoluzione	0.5 µs (~0.05 °)
Tensione di alimentazione	5-16V
Tensione I/O	0 e 5 V
Velocità seriale	1200 - 38400 (autodetect)
Consumo	5 mA (valore medio)

ALIMENTAZIONE

I servo sono alimentati con una tensione compresa tra i 4.8 ed i 6 V. Questa alimentazione può essere fornita tramite il connettore posto in alto a destra.

L'alimentatore deve essere in grado di garantire la corrente richiesta dai servo che, nel caso siano tutti collegati e vengano mossi simultaneamente, può essere molto elevata (vicina a 10 A). Il processore necessita di una sua alimentazione separata che può essere compresa tra i 5 e i 16 V, fornita tramite il connettore in basso a sinistra (PIN segnati con VIN e GND).

SEGNALE DI CONTROLLO

Il Servo controller è controllato tramite una logica seriale TTL (0-5 V) fornita al PIN “logic-level serial input”.

Esistono poi i pin “reset” e “logic-level serial input” che nella maggioranza delle applicazioni, possono essere lasciati sconnessi.

OPZIONI DI INTERFACCIA

Si può comunicare con il Servo controller con due diversi protocolli di comunicazione.

La scelta avviene utilizzando un cavallotto che è letto all’atto dell’accensione della scheda. Per cambiare il protocollo occorre resettare la scheda.

Modo Pololu: è la modalità di funzionamento predefinita con cavallotto non inserito. In questa maniera, il Servo Controller può essere connesso con altre apparecchiature seriali.

Questo modo permette anche l’accesso a tutte le caratteristiche speciali come settaggio velocità, range e settaggio posizione neutra.

Modo Mini SSC II: Questa modalità operativa è scelta inserendo il cavallotto sull’apposito spinotto. In questo modo il Servo Controller risponde al protocollo usato dal controllore **Mini SSC II** Servo realizzato dalla **Scott Edwards Elettronics**.

Questo protocollo è più semplice e permette solamente di specificare il numero del servo e la sua posizione.

MICRO MAESTRO

La seconda scheda che analizziamo e che utilizzeremo per le prove, è la **Micro Maestro** prodotta sempre dalla **Pololu**,

questa scheda fa parte di una serie di altri 4 modelli che differiscono principalmente per il numero di servomotori che possono comandare che va da 6 a 24.

Sulla scheda troviamo un processore **PIC18F14K50-I/SS** a 8 bit, 48MHz, memoria 16 kB, memoria dati da 256 bytes, con implementati un’interfaccia USB V2.0, un risuonatore per il clock, un regolatore di tensione, tre LED di stato e vari connettori con funzione di alimentazione di interfaccia e per il collegamento dei servomotori.

Supporta tre metodi di controllo: USB per una connessione diretta con il computer, seriale TTL per l’utilizzo con sistemi embedded, e internal scripting per applicazioni integrate, senza bisogno di un controllore esterno.

I canali possono essere configurati come uscite servo per utilizzarli con i normali servo RC da modellismo o con gli ESC (Electronic Speed Control), come uscite digitali, oppure ancora

come ingressi analogici.
Con il modulo è possibile il controllo integrato di velocità ed accelerazione su ciascun canale così da ottenere movimentazioni morbide e senza scatti.
Ogni scheda può essere collegata in cascata con altri controllers Pololu su una singola linea seriale. Il firmware della scheda è aggiornabile.
Il modulo Micro Maestro ha tre LED indicatori di tre colori diversi: il LED verde indica lo stato del dispositivo USB, il LED rosso error/user di solito indica un errore, il LED giallo indica lo stato di controllo.

Specifiche Tecniche :

Dimensioni stampato	~ 21mm×30mm
Numero di porte per servo	6
Range larghezza impulso	64 e 3280 µs
Risoluzione	0,25 µs (~0.025 °)
Tensione di alimentazione	5-16V
Tensione I/O	0 e 5 V
Velocità seriale	300 - 115200 bps
Consumo	30 mA (valore medio)

ALIMENTAZIONE

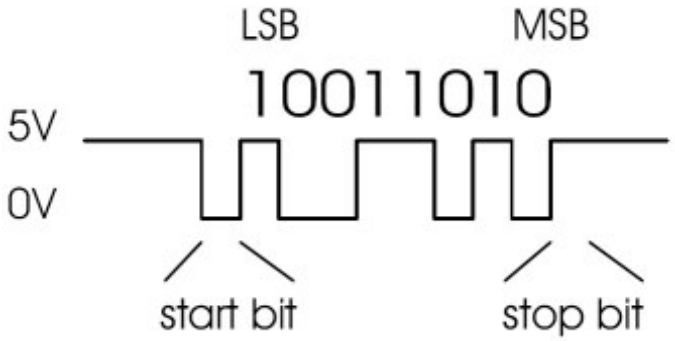
L'alimentazione del processore può essere fornita tramite la presa USB o da un alimentatore esterno che abbia un valore compreso tra i 5 e i 16V collegato agli ingressi VIN e GND. I servo sono alimentati da un connettore presente in alto a destra nella scheda.
L'alimentazione ai servi è fornita direttamente, senza passare attraverso un regolatore, in modo che le uniche restrizioni siano la potenza dell'alimentatore e che sia in grado di fornire la corrente necessaria; approssimativamente l'assorbimento di corrente di un servo di media grandezza è di 1 A. È presente un'uscita 5V che consente di alimentare dispositivi esterni, la corrente è però limitata a 50mA.

SEGNALE DI CONTROLL0

Il Servo controller è controllato tramite una logica seriale **TTL (0-5 V)** fornita al PIN **TX/RX**. Esiste poi il pin **RST** che può anche essere lasciato sconnesso.

LA COMUNICAZIONE SERIALE

I comandi seriali inviati ai moduli controller devono essere in blocchi di 8 bit, senza parità e con un bit di stop, oppure abbreviato **8N1**.



Il livello logico deve essere non-invertito, considerando che "0" è inviato come 0V, mentre "1" è inviato come 5V.

COMANDI DELLA SCHEDA MICRO SERIAL SERVO CONTROLLER - MODO POLOLU

In questa modalità sono possibili molti comandi del movimento del servomotore.
Baud Rate: la serie di velocità di trasmissione disponibili in questa maniera è approssimativamente tra 2000 e 40000 baud.
Protocollo: per comunicare con il Servo Con-

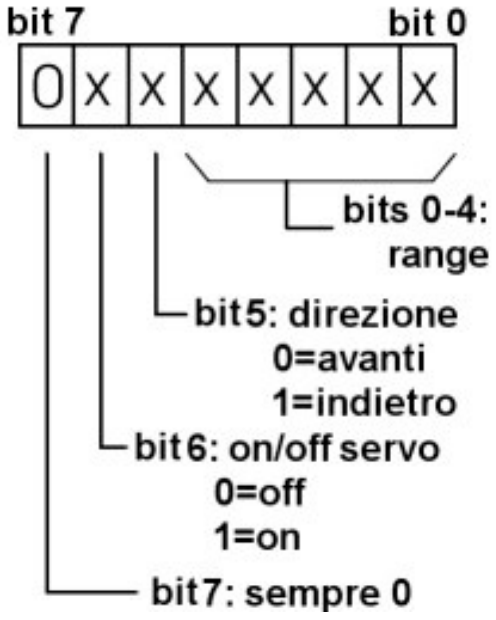
troller occorre inviare una sequenza di 5 o 6 byte.
Il primo byte è di sincronizzazione e deve essere sempre 0x80.
Il secondo byte identifica l'apparecchiatura Pololu nel nostro caso 0x01.
Il terzo byte è uno di sei valori per i diversi comandi (vedere sotto).
Il quarto byte è il numero del servo che si vuole comandare.
Il 5 o 6 byte sono i parametri per il comando impartito.
I valori dei byte da 2 a 6 sono compresi tra 0-0x7F (0-127).

Comando 1: Impostazione velocità (1 byte di dati)
Questo comando permette di variare la velocità con cui si muove il servomotore.
Di default la velocità è posta a 0, in questo modo il servo si sposterà immediatamente alla posizione fissata.
Se il valore di velocità è diverso da 0, il servo si sposterà progressivamente dalla vecchia alla nuova posizione. Con una velocità di 1, gli impulsi cambiano di 50 µs per secondo; la massima velocità è di 6.35 ms quando la velocità è posta a 127.

Start byte = 0x80	Numero ID = 0x01	Comando	Dato 1	Dato 2

Comando 0: Impostazione parametri (1 byte di dati)

- Il byte 7 è sempre 0;
- Il byte 6 specifica se un servo è attivo oppure no; 1 attiva il servo, 0 (default) lo disattiva;
- Il byte 5 stabilisce la direzione di rotazione; 0 (default) avanti, 1 indietro;
- I byte 0-4 stabiliscono il range del movimento.



Comando 2: Impostazione posizione, 7-bit (1 byte di dati)
Quando questo comando è inviato, il valore del dato è moltiplicato per il range impostato per il servomotore corrispondente e corretto per il settaggio neutrale.
Questo comando può essere utile nell'andare velocemente ad una posizione in quanto si utilizzano solamente 5 byte per impostare la posizione. Assegnando una posizione, automaticamente il servo sarà attivato.

Comando 3: Impostazione posizione, 8-bit (2 byte di dati)
Questo comando è simile a quello a 7-bit eccetto per il fatto che devono essere inviati due byte.

Comando 4:Impostazione Posizione, Assoluto (2 byte di dati)
Permette il controllo diretto della posizione del servo. Range Neutrale e impostazione direzione non hanno effetto in questo comando. La serie di valori validi è compresa tra 500 e 5500. Assegnando una posizione automaticamente il servo sarà attivato.

Comando 5: Impostazione Neutrale (2 byte di dati)

L'impostazione neutrale è applicabile solamente a comandi a 7 e 8 bit. Il valore neutrale imposta il valore medio del range e corrisponde per il bit 7 a 63.5 e per il bit 8 a 127.5. La posizione neutrale è una posizione assoluta simile al comando 4, e impostando la posizione neutrale, il servo si sposterà a quella posizione. Il valore predefinito è 3000. Può essere utile cambiare la posizione neutrale per calibrare un sistema in modo da correggere tolleranze dei meccanismi meccanici connessi ai servomotori.

COMANDI DELLA SCHEDA MICRO MAESTRO - MODO POLOLU

I comandi per la gestione di servomotori disponibili per il **Micro Maestro** in modo Pololu sono:

- **Set Target** - Impostazione della posizione del servo;
- **Set Speed**- Imposta la velocità con cui il servo raggiunge la posizione assegnata;
- **Set Acceleration** - Questo comando imposta l'accelerazione del servomotore;
- **Get Position** - Questo comando consente di ricevere il valore di posizione di un servomotore;
- **Get Moving State** - Questo comando è utilizzato per determinare se le uscite servo hanno raggiunto la loro posizione finale o stanno ancora cambiando;
- **Get Errors** - Utilizzato per esaminare gli errori che il Maestro ha rilevato;
- **Go Home** - Questo comando porta tutti i servi alle loro posizioni iniziali.

Esaminiamo in dettaglio alcuni comandi:

Set Target

Protocollo : 0xAA, numero del dispositivo, 0x04, numero servo, target low bit, target high bit.

Si utilizza questo comando per ruotare il servo di un determinato angolo.

Nel nostro caso il target rappresenta la larghezza dell'impulso da trasmettere in unità di quarti di microsecondi.

Un valore di 0 indica alla scheda di interrompere l'invio di impulsi al servo. Ad esempio: se si vuole inviare sul canale 2 un impulso di 1500 ms (1500 × 4 = 6000 = 01011101110000 in binario), è possibile inviare la seguente sequenza di byte:

- in binario: 10000100, 00000010, 01110000, 00101110
- in esadecimale: 0x84, 0x02, 0x70, 0x2E
- in decimali: 132, 2, 112, 46

Set Speed

Protocollo: 0xAA, numero del dispositivo, 0x07, numero servo, accelerazione bit bassi, accelerazione bit alti.

Questo comando limita la velocità con cui cambia il valore di uscita di un canale servo. Il limite di velocità è espresso in unità di (0,25 ms) / (10 ms). Una velocità di 0 (valore di default) rende la velocità illimitata, in modo che l'impostazione della destinazione influenzi immediatamente la posizione. Si noti che la velocità reale alla quale si muove il servo è limitata dalle caratteristiche del servo stesso, la tensione di alimentazione e carichi meccanici.

Set Acceleration

Protocollo: 0xAA, numero de dispositivo, 0x09, numero servo, accelerazione bit bassi, accelerazione bit alti.

Questo comando imposta l'accelerazione del servomotore. Il limite di accelerazione è un valore compreso tra 0 a 255 in unità di (0,25 ms) / (10 ms) / (80 ms). Il valore 0 corrisponde a nessun limite di accelerazione. Il limite di accelerazione determina la velocità che impiega il servo

ad arrivare alla velocità massima; il tempo impiegato in decelerazione provoca un movimento relativamente fluido da un punto ad un altro. Con l'impostazione di minima accelerazione 1, l'uscita del servo impiega circa 3 secondi per passare agevolmente da un valore di 1 ms ad un valore di 2 ms.

Get Position

Protocollo: 0xAA, numero di dispositivo, 0x10, numero del canale.

Questo comando consente al dispositivo di comunicare con la scheda di controllo per ottenere il valore di posizione di un servo. La posizione è inviata come risposta a due byte immediatamente dopo che il comando viene ricevuto. Il valore rappresenta la larghezza d'impulso che la scheda sta trasmettendo al servo.

Get Moving State

Protocollo: 0xAA, numero di dispositivo, 0x13

Risposta: 0x00 se i servo non si muovono, 0x01 se i servo si stanno muovendo. Questo comando è utilizzato per determinare tramite le uscite se i servo hanno raggiunto la loro posizione o stanno ancora ruotando. Il valore sarà 1 purché vi sia almeno un servo che sia ancora in movimento. L'utilizzo di questo comando è necessario per avviare diversi movimenti servo e attendere che tutti i movimenti per finire prima di passare alla fase successiva del programma.

Get Errors

Protocollo: 0xAA, numero di dispositivo, 0x21

Risposta: bit di errore 0-7, errore di bit 8-15. Si può utilizzare questo comando per esaminare gli errori che la scheda controllo ha rilevato. Sul manuale, alla Sezione 4.b, sono elencati gli errori specifici che possono essere rilevati dal Maestro. Il registro di errore viene inviato come risposta a due byte immediatamente dopo che il comando è stato ricevuto, allora tutti i bit di errore vengono cancellati.

Go Home

Protocollo: 0xAA, numero di dispositivo, 0x22

Questo comando invia a tutti i servi la loro posizioni iniziale, proprio come se si fosse verificato un errore. Per i servi e le uscite impostate su "Ignora", la posizione sarà invariata.

USO DELLA LIBRERIA PER LA GESTIONE DEL MODULO MICRO MAESTRO

Come abbiamo visto, i comandi per la gestione dei servo sono molti, e realizzare un programma di gestione da utilizzare con la scheda Arduino potrebbe essere complesso.

Per questo motivo è molto più comodo utilizzare un'apposita libreria che si occupi di inviare i giusti comandi.

Ricercando in rete si è trovata una libreria già pronta, questa denominata **PMCtrl** realizzata da **Graham Sortino**

Per utilizzare la libreria sarà sufficiente accedere alla pagina e scaricare l'ultima versione disponibile, questa dovrà essere salvata all'interno della cartella Library dell'IDE di Arduino.

FUNZIONI DI ASSEGNAZIONE E INIZIALIZZAZIONE

Per utilizzarla, occorre prima di tutto includerla nel programma con il comando.

```
#include <PMCtrl.h>
```

Quindi occorre definire il nome con cui si chiamerà il modulo, a quali pin sarà connesso per il dialogo seriale e a quale velocità si utilizzerà per la comunicazione; il comando da utilizzare è

```
PMCtrl servoCtrl (11, 3, 9600); //RX, TX, Baud
```

COMANDI DISPONIBILI NELLA LIBRERIA

A questo punto si potranno utilizzare i vari comandi disponibili nella libreria, questi sono:

servoCtrl.setServoSpeed (servoSpeed, channel, 12)

Il comando imposta la velocità servoSpeed con cui il servo si muove, channel è il servo che di cui vogliamo variare la velocità e deviceID per il Micro Maestro è il numero 12.

servoCtrl.setTarget (pos, channel, 12);

Il comando imposta la posizione pos che vogliamo far assumere al servo, channel è il servo che di cui vogliamo variare la posizione e deviceID è sempre il numero 12.

servoCtrl.goHome(12)

Il comando sposta tutti i servi collegati alla loro posizione di riposo

servoCtrl.getPosition(channel,I12)

Il comando restituisce la posizione attuale di un servo

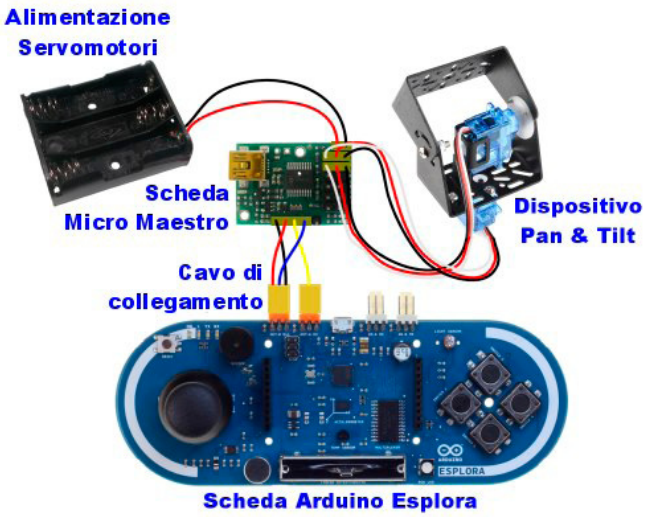
getErrors (channel, 12)

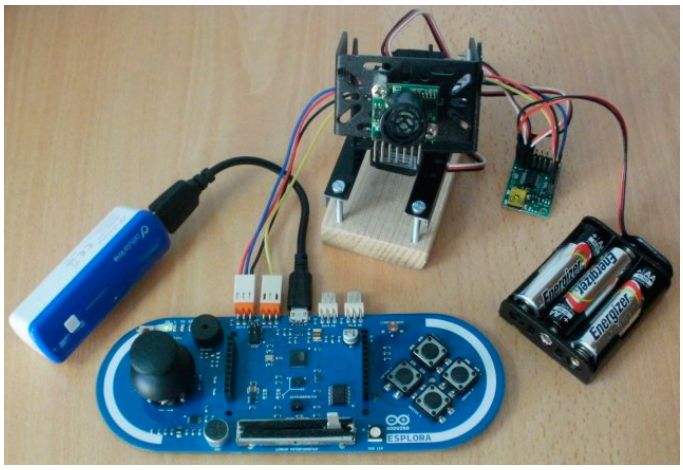
Il comando restituisce le informazioni riguardanti lo stato di errore del servo connesso a canne, per i codici vedere la **relativa pagina**.

CIRCUITO DI PROVA

Per testare il modulo con la scheda Arduino Esplora, si utilizzerà il circuito di prova mostrato di seguito che è molto semplice e prevede i seguenti elementi.

- Scheda Arduino Esplora;
- Scheda Micro Maestro;
- Due servomotori per esempio connessi ad un semplice dispositivo pan & Tilt;
- Cavo di collegamento seriale tra scheda Arduino Esplora e Micro Maestro;
- Fonte di alimentazione per i servomotori, per esempio porta batterie e 3 batterie tipo AA 1,5V;
- Fonte di alimentazione per la scheda Arduino Esplora, per esempio una batteria/caricabatteria che fornisca 5V.



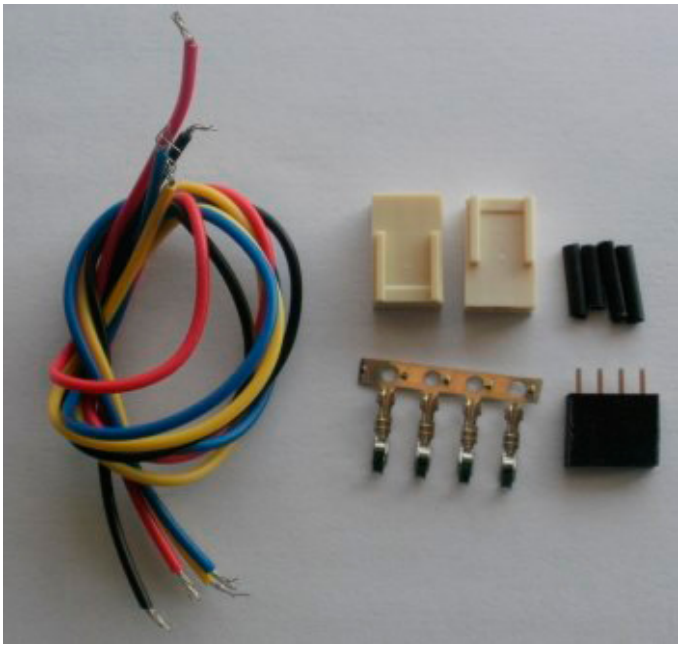


REALIZZAZIONE CAVI DI COLLEGAMENTO TRA SCHEDA ARDUINO ESPLORA E MICRO MAESTRO

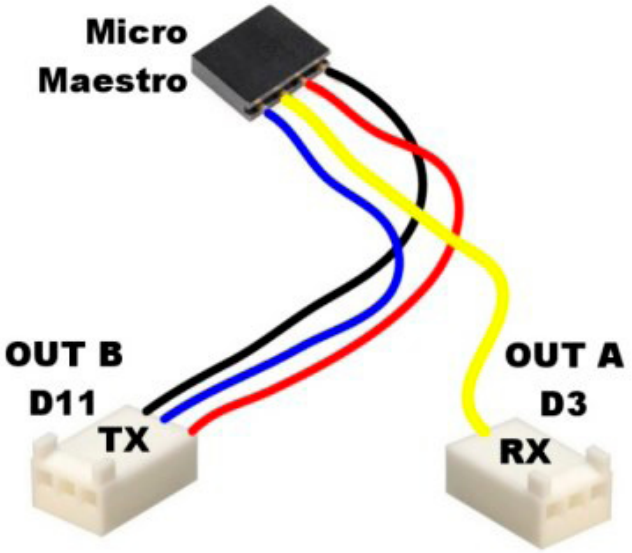
Per realizzare il cavo di collegamento tra scheda Arduino Esplora e Micro Maestro sono necessari:

- N° 2 connettori a tre pin femmina passo 2,54 mm tipo Molex 22-1-3037 (codice **RS 679-5375P**);
- N° 1 connettore pin strip femmina 1X4 passo 2.5 mm;
- N° 4 spezzoni di cavi lunghezza a piacere di colori: nero, rosso, blu, giallo.

- N° 4 spezzoni di guaina termoretraibile.



Nella figura è riportato come devono essere collegati i 4 cavi che sono Positivo e Negativo, alimentazione, TX e RX.

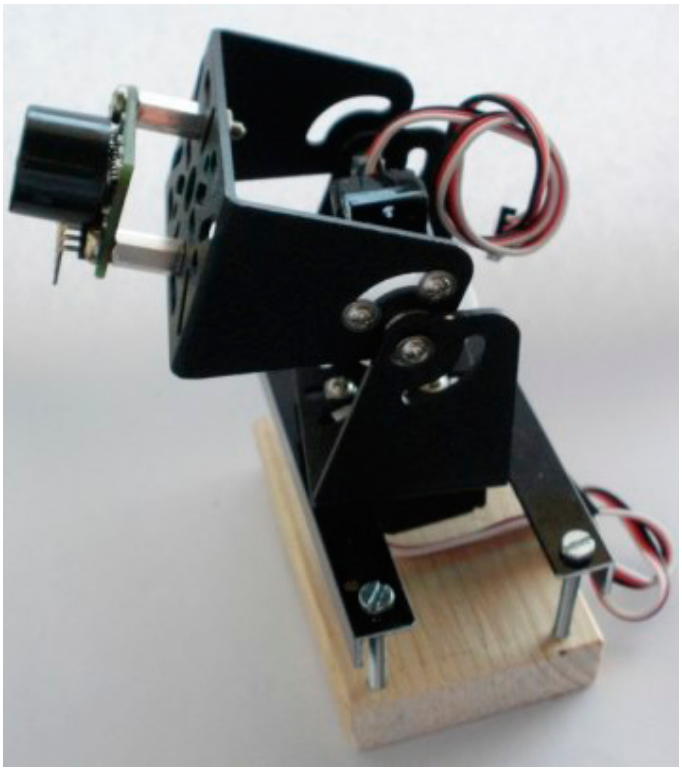


DISPOSITIVO PAN & TILT

Per il test si è utilizzato un piccolo dispositivo Pan & Tilt, questo è venduto dalla **SparkFun codice ROB-10335**.

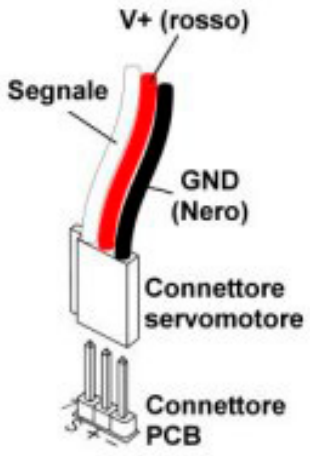
Nel kit sono presenti i componenti meccanici cioè due staffe e tutto l'hardware necessario per il fissaggio, i servomotori devono essere presi a parte, quelli consigliati sono dei servo compatibili con il modello Hitec HS-55 codice **Sparkfun ROB-09065**

Per le istruzioni di montaggio si potrà fare riferimento alla **seguente pagina**.



COLLEGAMENTO DEI SERVOMOTORI

Quando si collegano i servomotori occorre prestare attenzione poiché un errato collegamento può danneggiarli. Assicurarsi che il filo del segnale (di solito bianco o giallo) sia posto verso l'interno della scheda mentre il cavo nero verso l'orlo esterno.



PROGRAMMA DI TEST

Il programma di test (**Arduino_esplora_servo**) è molto semplice, è utilizzato il Joystick presente sulla scheda Arduino Esplora per comandare un dispositivo Pan & Tilt dotato di due servomotori.

Il programma utilizza le seguenti librerie:

Gestione Arduino Esplora

#include <Esplora.h>

Gestione della Micro Maestro

include <PMCtrl.h>

che a sua volta utilizza la libreria gestion seriale

#include <SoftwareSerial.h>

Il programma è sufficientemente commentato e non dovrebbe presentare problemi di comprensione.

Per quanto riguarda il comando map, questo è utilizzato per adattare i valori letti dal Joystick ai valori e nei limiti necessari alla gestione dei servomotori, si potrà far riferimento alla guida presente sul sito Arduino.

FILMATO ILLUSTRATIVO

Nel filmato, potrete vedere i vari componenti utilizzati e l'esecuzione del programma.

https://www.youtube.com/watch?v=ZDMl8B7b7Nk

CONCLUSIONI

Questo è solo un esempio di quello che è possibile fare con la **Micro Mastro** in unione con la **Arduino Esplora**.

In questo caso, la gestione era effettuata tramite cavo ma, come abbiamo visto in un precedente articolo, "**Dotiamo l'Arduino Esplora dell'interfaccia Bluetooth**", è possibile interfacciare la scheda con un altro dispositivo dotato anch'esso di un modulo Bluetooth.

In questo caso occorrerà utilizzare magari un **Arduino Micro** che funga da interfaccia tra il modulo Bluetooth e la scheda **Micro Maestro**.

L'unico limite ai progetti realizzabili è dato dai sei servomotori che possono essere pilotati, limite che può essere superato utilizzando una delle altre schede della serie Maestro che, nel modello "maggiore", arriva a ben 24 unità.

```
//Gestione Arduino Esplora
#include <Esplora.h>:
//Gestione Micro Mestro
#include <PMCtrl.h>
#include <SoftwareSerial.h>
//Definizione pin gestione Micro Maestro
 #define rxPin 11
 #define txPin 3
 //Impostazione Parametri di collegamento con Micro Mestro
 PMCtrl servoCtrl (rxPin, txPin, 9600); //RX, TX, Baud
 //Definizione delle varibili utilizzate
 //Assegnazione canali servomotori
 const int servo0 = 0;
 const int servo1 = 1;
 //Definizione delle posizioni minime e massime
 // per i due assi
 const int MAX_W = 2224;
 const int MIN_W = 598;
 const int MAX_H = 1808;
 const int MIN_H = 495;
 //Definizione delle variabili della posizione
 //assunta dal Joystick
 int xPos = 0;
 int yPos = 0;
 void setup ()
 {
 //Imposta la velocità dei due servomotori
 servoCtrl.setServoSpeed (100, servo0, 12);
 servoCtrl.setServoSpeed (100, servo1, 12);
 //Sposta i servomotori nella posizione di riposo
servoCtrl.goHome(12);
 }
 void loop ()
 {
 //Assegna a XValue e YValue i valori letti dal Joystick
 int xValue = Esplora.readJoystickX();
 int yValue = Esplora.readJoystickY();
 //Utilizzo del comando map http://arduino.cc/en/reference/map
 //per adattare i valori letti nel range dei servomotori
 xPos = map( xValue, 512, -512, MIN_W,MAX_W);
 yPos = map( yValue, 512, -512, MIN_H,MAX_H);
 //Invia alla scheda MicroMestro le posizioni dei due servomotori
 servoCtrl.setTarget (xPos, servo0, 12);
 servoCtrl.setTarget (yPos, servo1, 12);
 delay (50); //Pausa tra gli aggiornamenti dei valori letti
 }
```

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/gestione-di-un-dispositivo-pan-tilt-con-la-scheda-arduino-esplora

Arduino M0 Pro

Arduino M0 Pro: presentazione e specifiche tecniche

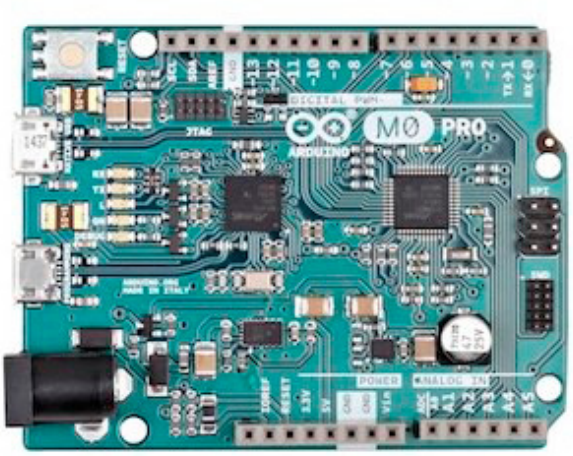
di Sara Ercolani

PRESENTAZIONE:

Microcontrollore - ATSAMD21G18, 48pin LQFP
Tensione di funzionamento - 3.3V
Pin Digitali I/O - 14, con 12 PWM e UART
Pin di ingresso analogico - 6, canali ADC 12 bit
Pin di output analogico - 1, DAC 10 bit
Corrente DC per Pin I/O - 7 mA
Memoria Flash - 256 KB
SRAM - 32 KB
EEPROM - fino a 16KB
Frequenza di clock - 48 MHz

ALIMENTAZIONE

L'Arduino M0 Pro può essere alimentato attraverso la connessione micro USB o con un alimentatore esterno. La fonte di alimentazione viene selezionata automaticamente.
L'alimentazione esterna (non USB) può essere fornita da un alimentatore AC-DC o da una



batteria. L'adattatore può essere collegato alla scheda inserendo lo spinotto nel jack di alimentazione della scheda stessa. I fili elettrici della batteria possono essere inseriti nei pin GND e Vin del connettore di alimentazione.

Il kit rileverà automaticamente quali fonti di alimentazione sono disponibili e sceglierà quale utilizzare secondo le seguenti priorità:

1. Alimentazione esterna
2. Embedded debugger USB
3. Target USB

L'alimentazione esterna è necessaria quando i 500mA tramite il connettore USB non sono sufficienti per alimentare un dispositivo USB collegato in un'applicazione USB host.

I pin di alimentazione sono i seguenti:

- **VIN**: la tensione di ingresso alla scheda Arduino quando sta utilizzando una sorgente di alimentazione esterna (rispetto ai 5 volt dal collegamento USB o un'altra fonte di alimentazione regolata). È possibile fornire tensione attraverso questo pin, o, se l'alimentazione avviene tramite il jack, accedervi attraverso questo pin.
- **5V**: l'alimentatore regolato utilizzato per caricare il microcontrollore e altri componenti sulla scheda. Questo può venire sia da VIN attraverso un regolatore sulla board, o essere fornita da USB o da un altro 5V regolato.
- **3V3**: tensione di 3.3 volt generati dal regolatore sulla scheda. La massima corrente di assorbimento è di 50 mA.

Power input	Requisiti di tensione	Requisiti di corrente	Tipo di connettore
Alimentazione esterna	5V ± 2% (± 100mV) per il funzionamento host USB. Da 4.3V a 5.5V, se non è richiesto il funzionamento host USB	La corrente minima consigliata è di 1A per essere in grado di fornire abbastanza corrente ai dispositivi USB collegati e alla board stessa. La massima raccomandata è di 2A a causa della protezione di input	PWR
Debugger integrato USB	da 4.4V a 5.25V (secondo spec. USB)	500mA (secondo spec. USB)	DEBUG USB
Target USB	da 4.4V a 5.25V (secondo spec. USB)	500mA (secondo spec. USB)	Target USB

- **GND**: GND
- **IOREF**: la tensione alla quale i pin I/O della scheda sono in funzione (ad esempio VCC per la scheda). Questa è 3.3V sulla M0 Pro.

MEMORIA

L'ATSAMD21G18 ha 256 KB (con 4 KB usati per il bootloader). Il bootloader è programmato di fabbrica da Atmel, viene memorizzato in una ROM dedicata ed è protetto con il fusibile NVM. Sono inoltre disponibili 32 KB di SRAM e fino a 16 KB in emulazione di EEPROM (che può essere letta e scritta con la libreria EEPROM).

INPUT ED OUTPUT

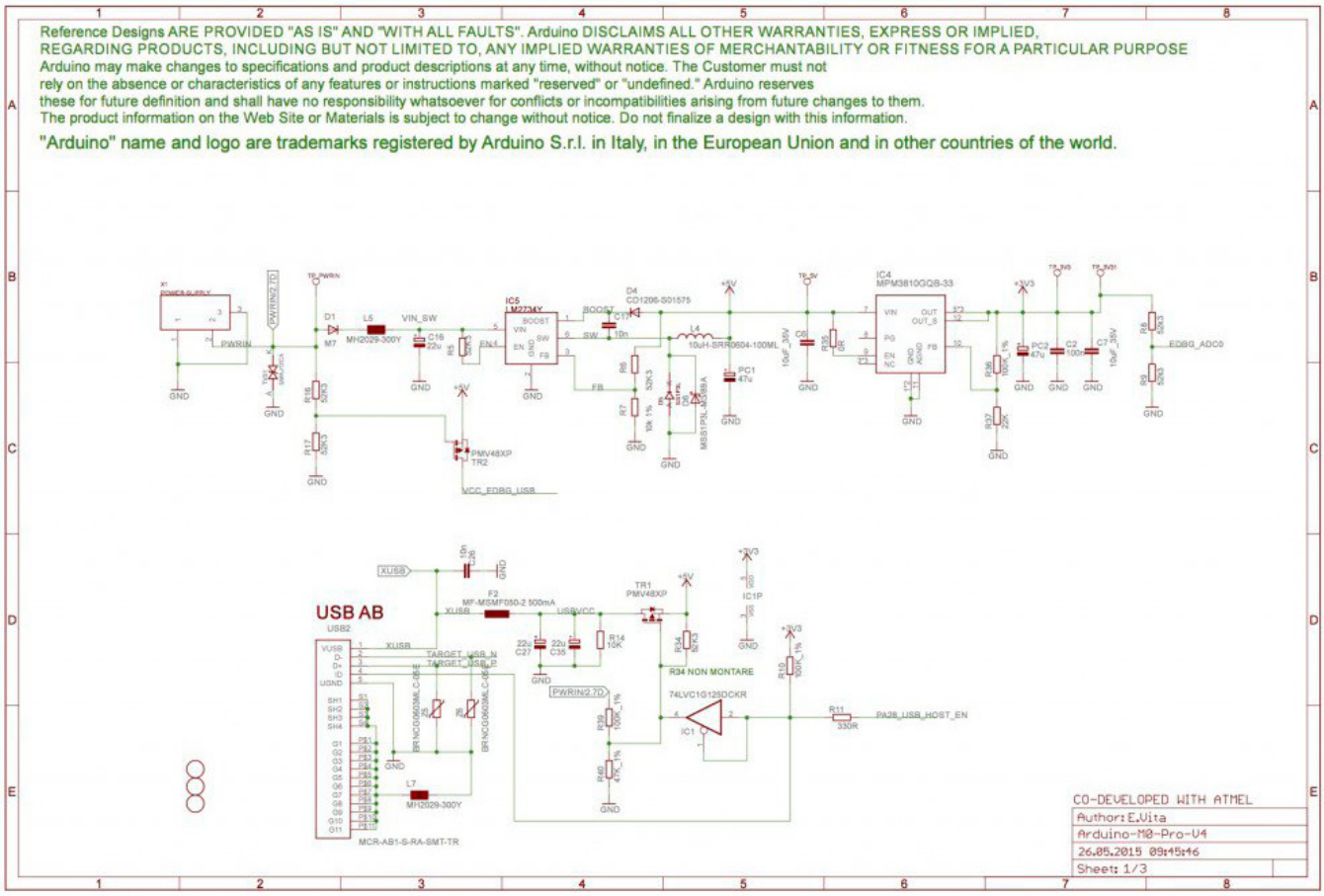
Ciascuno dei 20 pin I/O digitali dello M0 Pro può essere utilizzato come ingresso o uscita, utilizzando pinMode (), digitalWrite () e le funzioni digitalRead (). Queste operano a 3,3 volt con 7mA di corrente DC massima per pin I/O e con una resistenza di pull-up interna (scollegata di default) di 20-50 kOhms. Inoltre, alcuni pin hanno funzioni specializzate:

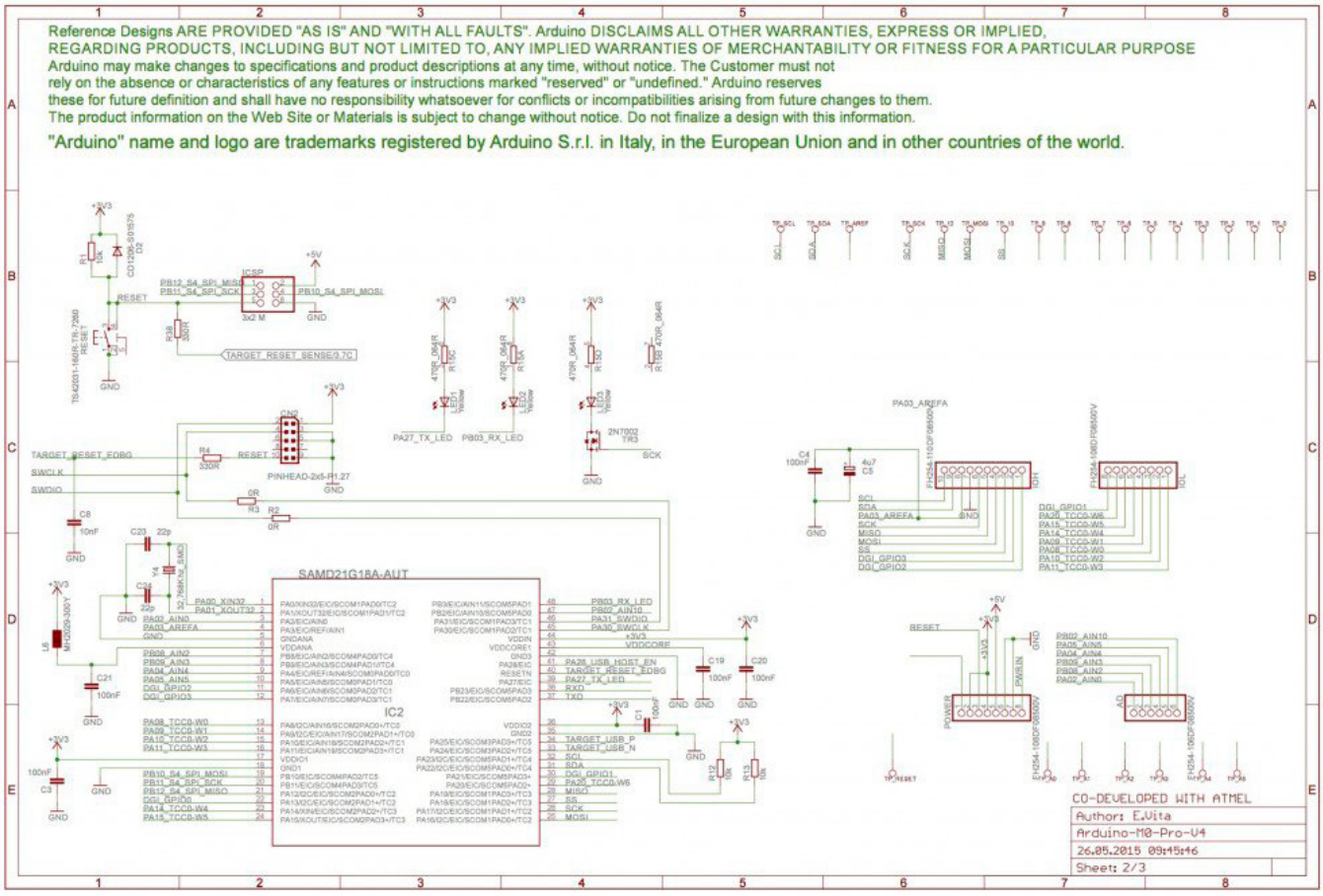
- **Serial**: 0 (RX) e 1 (TX). Utilizzati per ricevere (RX) e trasmettere (TX) i dati seriali TTL utilizzando la capacità dell'hardware ATSAMD21G18. Si noti che sullo M0 Pro, la Serial Class si riferisce alla comunicazione USB (CDC); per i TTL serial sui pin 0 e 1, è necessario utilizzare la Serial1 class.
- **TWI**: 2 (SDA) e 3 (SCL). Supporta la comunicazione TWI utilizzando la libreria Wire.
- **PWM**: i pin 2-13 forniscono un output PWM a 8-bit con la funzione analogWrite (). La risoluzione del PWM può essere cambiata con la funzione analogWriteResolution ().Nota 1: I pin 4 e 10 non possono essere utilizzati contemporaneamente come PWM.Nota 2: I pin 5 e 12 non possono essere utilizzati contemporaneamente come PWM.
- **SPI**: sull'header ICSP. Questi pin supportano la comunicazione SPI utilizzando la libreria SPI. Si noti che i pin SPI non sono collegati ai pin digitali di I/O come sulla UNO, ma sono disponibili solo sul connettore ICSP. Questo significa che se si dispone di uno shield che utilizza SPI, ma non si dispone di un connettore a 6 pin ICSP che si collega ai 6 pin nell'header della M0 Pro ICSP, lo shield non funzionerà.
- **LED**: 13. Vi è un LED incorporato collegato al pin digitale 13. Quando il pin ha un valore alto, il LED è acceso, quando il pin è basso, è spento.
- **Ingressi analogici**: A0-A5. La M0 Pro dispone di 6 ingressi analogici, etichettati A0-A5. I pin A0-A5 appaiono nelle stesse posizioni della UNO; ogni ingresso analogico fornisce 12 bit di risoluzione (cioè 4096 valori differenti). Per default la misura degli ingressi analogici riferita a GND è di 3,3 volt, anche se è possibile cambiare l'estremità superiore utilizzando il pin AREF e la funzione analogReference ().
- **DAC**: il pin A0 fornisce vere uscite analogiche con 10 bit di risoluzione (1023 livelli) con la funzione analogWrite (). Questo pin può essere usato per creare un output audio utilizzando la libreria audio.

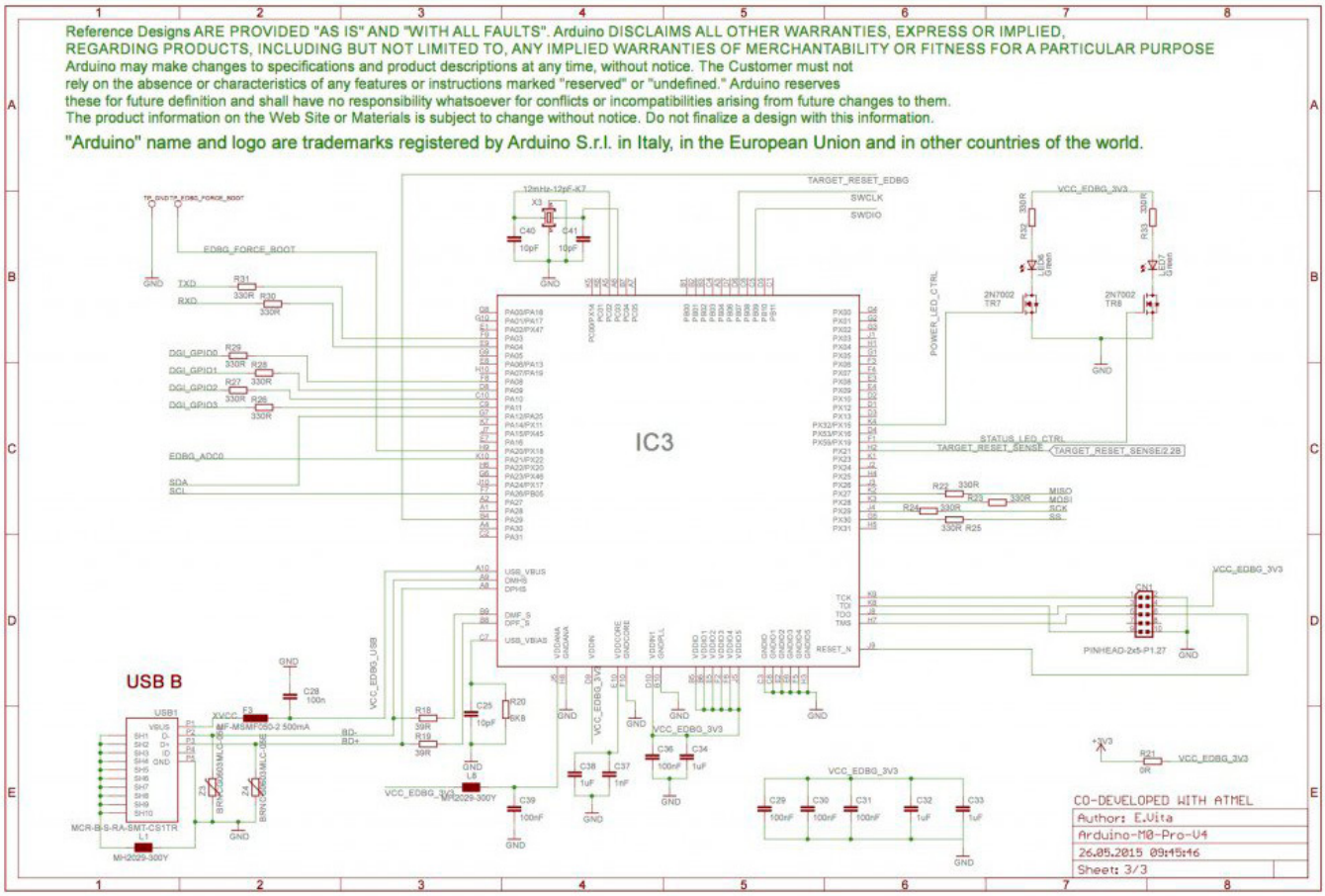
Ci sono infine altri due pin sulla scheda:

AREF: tensione di riferimento per gli ingressi analogici. Utilizzato con analogReference ().

Reset: portare questa linea LOW per resettare il microcontrollore. Questo è in genere utilizzato per aggiungere un pulsante di reset, quando gli shield che vengono utilizzati bloccano quello già presente sulla scheda.


Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED,
REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE
Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not
rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves
these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.
The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.
"Arduino" name and logo are trademarks registered by Arduino S.r.l. in Italy, in the European Union and in other countries of the world.
USB AB
R34 NON MONTARE
CO-DEVELOPED WITH ATMEL
Author: E.Vita
Arduino-M0-Pro-V4
26.05.2015 09:45:46
Sheet: 1/3


Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED,
REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE
Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not
rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves
these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.
The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.
"Arduino" name and logo are trademarks registered by Arduino S.r.l. in Italy, in the European Union and in other countries of the world.
SAMD21G18A-AUT
CO-DEVELOPED WITH ATMEL
Author: E.Vita
Arduino-M0-Pro-V4
26.05.2015 09:45:46
Sheet: 2/3



COMUNICAZIONE

L'Arduino M0 Pro possiede una serie di periferiche per la comunicazione con il computer, con un altro Arduino o altri microcontrollori e con diversi dispositivi come telefoni, tablet, macchine fotografiche e così via. Il SAMD21 possiede una UART hardware e tre USARTs hardware per la comunicazione seriale TTL (3.3V).

La Programming Port è collegata all'EDBG, che fornisce una virtual COM port al software su un computer collegato (per riconoscere il dispositivo, le macchine Windows avranno bisogno di un file .inf, ma le macchine OSX e Linux riconosceranno la scheda automaticamente come COM port). L'EDBG è anche collegato all'UART hardware del SAMD21. La seriale sui pin RX0 e TX0 fornisce una comunicazione USB per la programmazione della scheda attraverso il microcontrollore ATSAMD21G18. Il software Arduino include un Serial Monitor che permette l'invio di dati testuali semplici da e per la board. I LED RX e TX sulla scheda lampeggeranno quando i dati vengono trasmessi attraverso il chip ATSAMD21G18 e l'USB (ma non per la comunicazione seriale sui pin 0 e 1).

La Native Port è collegata al SAMD21. Permette la comunicazione seriale (CDC) tramite USB. Ciò fornisce un collegamento seriale al Serial Monitor o altre applicazioni sul computer. Consente inoltre alla board di emulare un mouse o una tastiera USB a un computer collegato.

La Native USB Port può anche fungere da host USB per le periferiche collegate, come mouse, tastiere e smartphone.

Il SAMD21 supporta anche la comunicazione TWI e SPI. Il software Arduino include una libreria Wire per semplificare l'uso del bus TWI. Per la comunicazione SPI, è necessario utilizzare la libreria SPI.

PROGRAMMAZIONE

L'Arduino M0 Pro può essere programmato con il software di Arduino (download).

Se si utilizza Linux-OS è necessario seguire la guida di Arduino IDE su Linux-OS.

Il caricamento degli sketches sul SAMD21 è diverso da come funziona con i microcontrollori AVR che si trovano in altre schede Arduino: la memoria flash deve essere cancellata prima che vengano riprogrammati. L'upload è gestito dalla ROM sul SAMD21, che viene eseguito solo quando la memoria flash del chip è vuota.

Entrambe le porte USB possono essere utilizzate per programmare la scheda, anche se si raccomanda l'uso della Programming Port, a causa del modo in cui viene gestita la cancellazione del chip:

Programming Port: per utilizzare questa porta, selezionare "Arduino M0 Pro (Programming Port)" come board nell'Arduino IDE. Collegare la porta di programmazione della M0 Pro (quella più vicino al jack DC) al computer. La porta di programmazione utilizza l'EDBG come USB-to-serial collegato al primo UART del SAMD21 (RX0 e TX0). L'EDBG ha due pin collegati ai pin di ripristino e di cancellazione del SAMD21. L'apertura e la chiusura della Programming Port connessa a 1200bps innesca una procedura di "hard erase" del chip SAMD21, attivando l'erase e il reset del pin sulla SAMD21 prima di comunicare con l'UART. Questa è la porta consigliata per la programmazione della M0 Pro. E' più affidabile rispetto al "soft erase" che si verifica sulla Native Port, e dovrebbe funzionare anche se l'MCU principale è andato in crash.

Native Port: per utilizzare questa porta, selezionare "Arduino M0 Pro (Native USB Port)" come board nell'Arduino IDE. La Native USB Port è collegata direttamente al SAMD21. Collegare la Native USB Port dello M0 Pro (quella più vicino al pulsante di reset) al computer. L'apertura e la chiusura della Native Port a 1200bps attiva una procedura di "soft erase": la memoria flash viene cancellata e la scheda viene riavviata con il bootloader. Se l'MCU è andato in crash per qualche motivo, è probabile che la procedura di soft erase non funzionerà come invece avviene normalmente. **L'apertura e la chiusura della porta nativa ad una diversa baudrate non resetterà il SAMD21**.

PROTEZIONE DI SOVRACORRENTE USB

La M0 Pro ha un polyfuse ripristinabile che protegge le porte USB del computer da corti e sovracorrenti. Sebbene la maggior parte dei computer forniscono loro la protezione internamente, il fusibile fornisce un ulteriore livello di protezione. Se vengono applicati alla porta USB più di 500 mA, il fusibile interromperà automaticamente la connessione fino a quando il corto o il sovraccarico non verrà rimosso.

CARATTERISTICHE FISICHE

La lunghezza e la larghezza massima del PCB M0 Pro sono rispettivamente 6.9 e 5.3 centimetri, con il connettore USB e l'alimentatore jack esclusi. Sono presenti quattro fori per le viti per consentire alla board di essere collegata a una superficie o un case. Si noti che la distanza tra i pin digitali 7 e 8 è di 160 mil (0.16"), non un multiplo della classica spaziatura 100 mil degli altri pin.

FILE SORGENTI DEL PROGETTO HARDWARE

Reference Design Arduino M0 Pro

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-zero-pro-presentazione-e-specifiche-tecniche

Arduino M0 Pro: getting started

di Ernesto Sorrentino

La scheda di sviluppo Arduino M0 Pro è dotata di un microcontrollore Atmel **ATSAMD21G18** basato su un core **ARM Cortex M0+** a 32 bit. Il D21G18A è provvisto di una memoria **Flash di 256 Kb** (di cui 4 occupati dal bootloader), **32 Kb di SRAM** e opera a una **frequenza di 48Mhz**. Questo core offre all'Arduino M0 Pro una elevata flessibilità, aumentando il ventaglio di progetti possibili da realizzare: dai dispositivi indossabili, Internet of Things, automazione high tech, fino alla robotica, e tanto altro ancora. Una caratteristica fondamentale, che rende unico questo dispositivo nella famiglia Arduino, è il debugger incorporato a bordo della scheda (EDBG) che elimina la necessità di hardware aggiuntivi. L' EDBG (**Embedded Debugger**) offre uno streaming di dati tra il PC e la MCU; oltre alla programmazione e il debug, attraverso Atmel Studio, supporta inoltre una porta **COM virtuale** per programmare il dispositivo nella funzionalità bootloader di Arduino.

NOTA: Al contrario della Arduino Uno, che ha una tensione di funzionamento di 5V, questo microcontrollore supporta solo i 3.3V; Una tensione operativa che si adegua ai nuovi standard richiesti dalle periferiche di ultima generazione, come l'ESP8266.

La M0 dispone di **due porte USB**; la prima detta "**Native**" è connessa direttamente al MCU, come per la board Arduino Micro, utilizzata per la programmazione tramite l'IDE e per essere collegata facilmente alle periferiche esterne quali: telefoni, tablet, macchine fotografiche e così via, consentendo così di realizzare un numero variegato di applicazioni in cui la scheda possa comunicare con i dispositivi più disparati. La seconda porta USB è chiamata "**Programming**", connessa direttamente al controller EDBG, utilizzata per la programmazione e per il debug del SAMD21G18A.

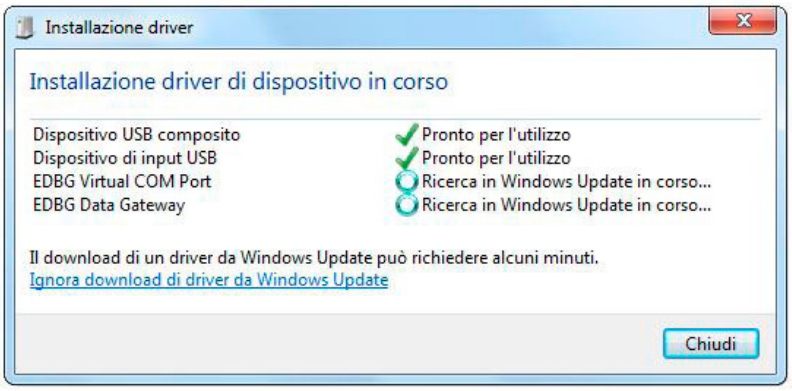
CARATTERISTICHE

Microcontroller	ATSAMD21G18, 48pins LQFP
Tensione di funzionamento	3.3V
Pins Digital I / O	14, con 12 PWM e UART
Pins di ingresso analogico	6, 12-bit ADC channels
Analog pin di uscita	1, 10-bit DAC
Corrente Max per Pin O/I	7 mA
Memoria Flash	256 KB
SRAM	32 KB
EEPROM	fino a 16KB per emulazione
Frequenza di clock	48 MHz

PROGRAMMING E DEBUG CON L'IDE DI ARDUINO

Per programmare l'Arduino M0 Pro è necessario l'utilizzo dell'IDE **aggiornata alla versione 1.7 o maggiore**; le prove eseguite per questo articolo sono state realizzate con la versione 1.7.3. La M0 Pro è stata rilasciata dall'azienda Arduino srl, pertanto tutti i driver e software sono sul portale arduino.org. Durante l'installazione del nuovo software si noterà l'aggiunta del pacchetto "**Atmel USB Driver**", una novità rispetto alle vecchie versioni, per il supporto e la gestione del driver Jungo.

Terminata l'installazione collegare l'Arduino M0 Pro al PC tramite la porta USB native e attendere il riconoscimento del dispositivo da parte del sistema operativo.



Avviare l'IDE per scrivere lo sketch di prova, come esempio ho utilizzato il classico progetto per far lampeggiare il led sulla board, come di seguito:

```
/*
  Blink Led
  Maggio 2015

  Sorrentino Ernesto
 */

#define  LED_PIN 13                // Definisce la porta DGT 13 come LED_PIN
#define  LED_ON  HIGH              // Definisce il livello logico ALTO come LED_ON
#define  LED_OFF  LOW              // Definisce il livello logico BASSO come LED_OFF

void setup() {
  pinMode(LED_PIN, OUTPUT);        // Imposta la porta DGT 13 come uscita
}
```

```
void loop() {
  digitalWrite(LED_PIN, LED_ON);       // Setta la porta DGT 13 a livello logico alto (led acceso)
  delay(250);                          // Attesa di 250ms
  digitalWrite(LED_PIN, LED_OFF);      // Setta la porta DGT 13 a livello logico basso (led spento)
  delay(250);                          // Attesa di 250ms
}
```

La prima parte del codice è composta dalle dichiarazioni “#define” per settare di default lo stato del led e il numero del pin dove è collegato il led al micro

```
#define  LED_PIN 13
#define  LED_ON  HIGH
#define  LED_OFF  LOW
```

La seconda parte setta la configurazione hardware, in particolar modo imposta la porta digitale 13, dove è collegato il led della board, come uscita

```
pinMode(LED_PIN, OUTPUT);
```

La terza parte, corrispondente al ciclo del programma, ci sono le istruzioni per settare il livello logico sulla porta DGT 13 e la routine di ritardo impostato a 250ms.

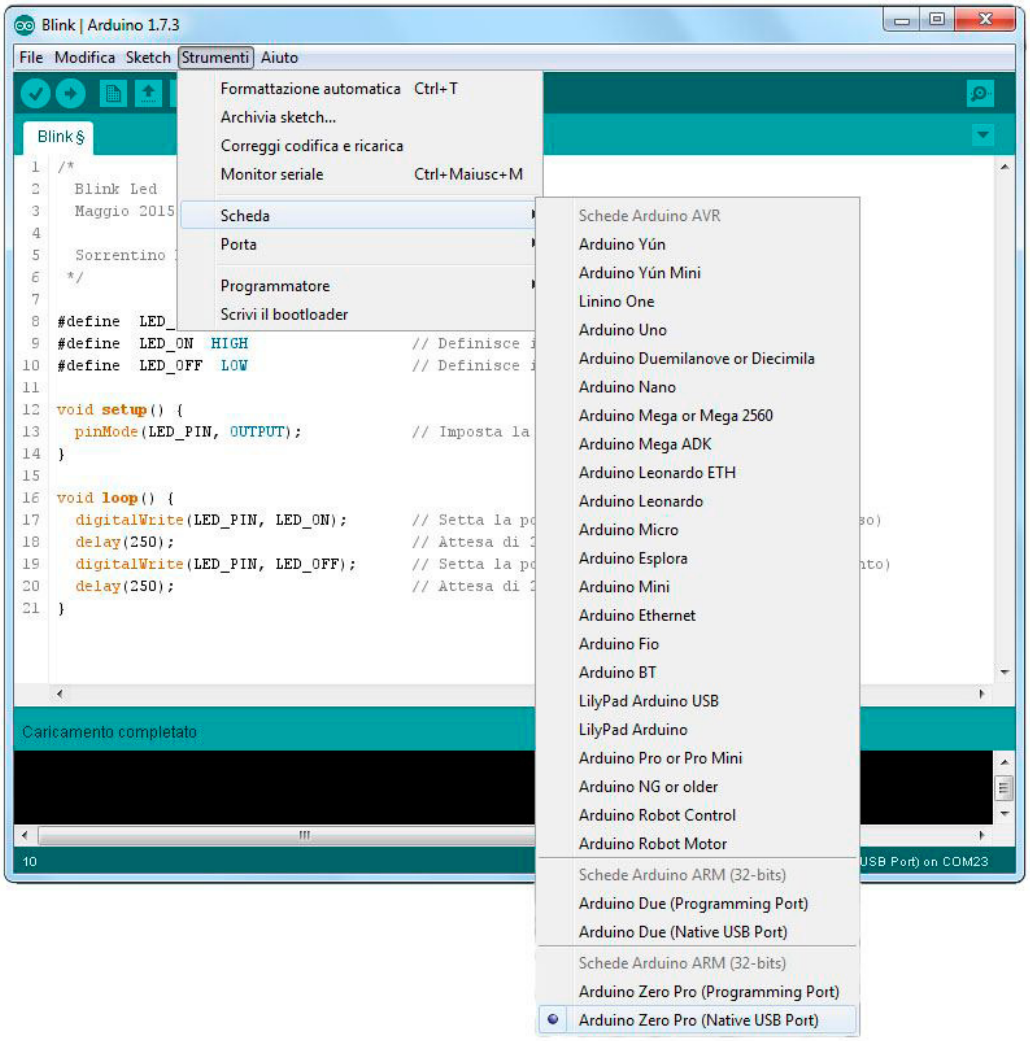
```
digitalWrite(LED_PIN, LED_ON);
delay(250);
digitalWrite(LED_PIN, LED_OFF);
delay(250);
```

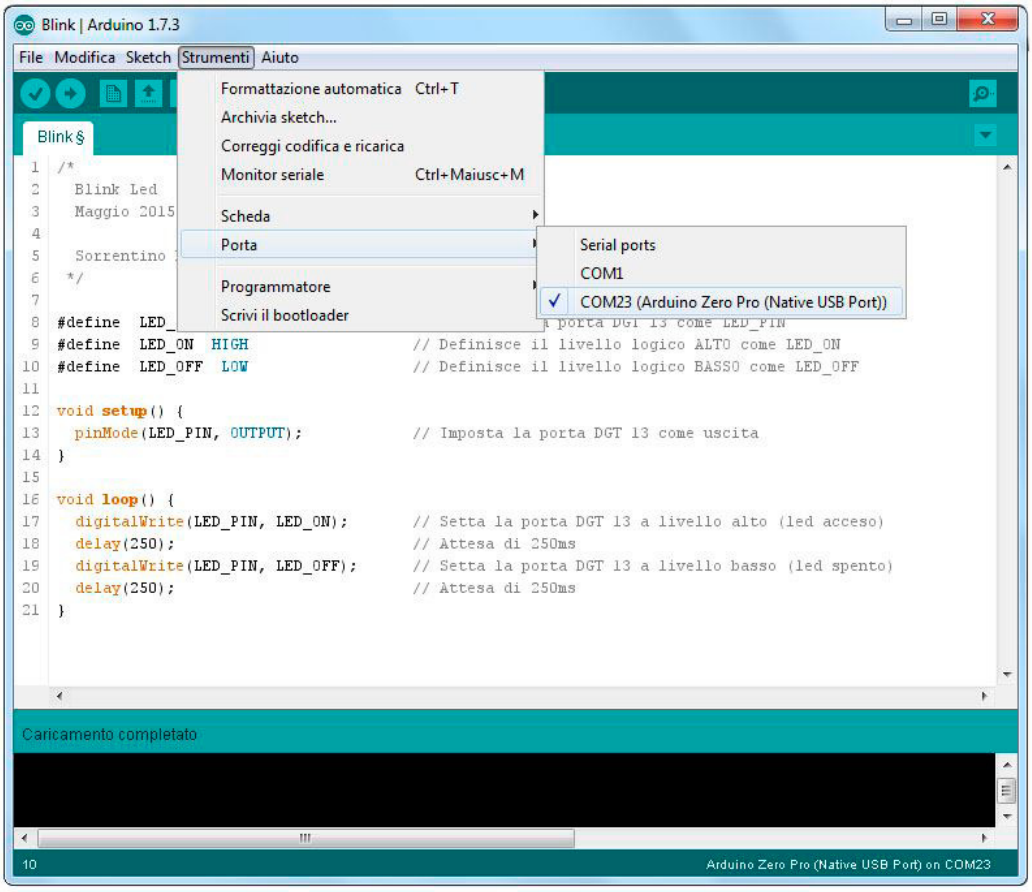
Prima di compilare lo sketch è necessario settare il software per la board in utilizzo; dal menù “**Strumenti/Scheda**” selezionare “**Arduino Zero Pro (Native USB Port)**”

Impostare la porta di comunicazione dal menù “**Strumenti/Porta**”, selezionare la COM riconosciuta dal sistema operativo

Terminato il settaggio compilare lo sketch e programmare la scheda con la combinazione dei tasti “**Ctrl + U**” o con l’apposito pulsante in alto a sinistra della finestra. Al termine della programmazione, **se tutto è andato a buon fine**, noterete il led lampeggiare; al contrario se il led non lampeggia bisogna effettuare un “**debug**” per risalire al problema. Per l’IDE in uso l’unico debug disponibile è la funzione “**Serial.print()**”, cioè inviare dall’Arduino, tramite la porta seriale, un comando testuale per avvisare durante l’esecuzione del programma il raggiungimento della riga che si vuol tenere sotto controllo. **Per la M0 il comando di “stampa” si differenzia in base alla porta USB in utilizzo**:

Serial5.print();	Per la porta "USB Programming"
SerialUSB.print();	Per la porta "USB Native"


Blink | Arduino 1.7.3
File Modifica Sketch Strumenti Aiuto
Formattazione automatica Ctrl+T
Archivia sketch...
Correggi codifica e ricarica
Monitor seriale Ctrl+Maiusc+M
Scheda
Porta
Programmatore
Scrivi il bootloader
Schede Arduino AVR
Arduino Yún
Arduino Yún Mini
Linino One
Arduino Uno
Arduino Duemilanove or Diecimila
Arduino Nano
Arduino Mega or Mega 2560
Arduino Mega ADK
Arduino Leonardo ETH
Arduino Leonardo
Arduino Micro
Arduino Esplora
Arduino Mini
Arduino Ethernet
Arduino Fio
Arduino BT
LilyPad Arduino USB
LilyPad Arduino
Arduino Pro or Pro Mini
Arduino NG or older
Arduino Robot Control
Arduino Robot Motor
Schede Arduino ARM (32-bits)
Arduino Due (Programming Port)
Arduino Due (Native USB Port)
Schede Arduino ARM (32-bits)
Arduino Zero Pro (Programming Port)
Arduino Zero Pro (Native USB Port)
Caricamento completato

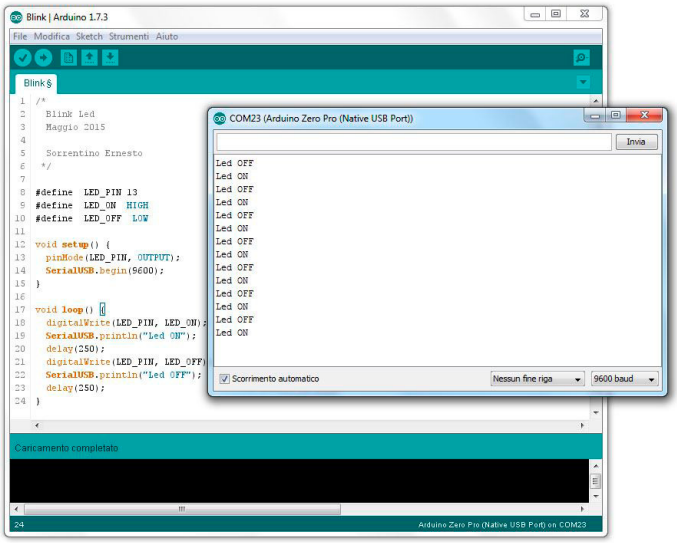

Blink | Arduino 1.7.3
File Modifica Sketch Strumenti Aiuto
Formattazione automatica Ctrl+T
Archivia sketch...
Correggi codifica e ricarica
Monitor seriale Ctrl+Maiusc+M
Scheda
Porta
Programmatore
Scrivi il bootloader
Serial ports
COM1
COM23 (Arduino Zero Pro (Native USB Port))
Caricamento completato
Arduino Zero Pro (Native USB Port) on COM23

Dato che la scheda è connesa al PC tramite porta USB Native, useremo quest'ultimo comando per modificare lo sketch come segue:

```
/*
  Blink Led
  Maggio 2015

  Sorrentino Ernesto
 */

#define  LED_PIN 13               // Definisce la porta DGT 13 come LED_PIN
#define  LED_ON  HIGH             // Definisce il livello logico ALTO come LED_ON
#define  LED_OFF  LOW             // Definisce il livello logico BASSO come LED_OFF

void setup() {
  pinMode(LED_PIN, OUTPUT);       // Imposta la porta DGT 13 come uscita
  SerialUSB.begin(9600);          // Inizializza la porta seriale
}

void loop() {
  digitalWrite(LED_PIN, LED_ON);  // Setta la porta DGT 13 a livello alto (led acceso)
  SerialUSB.println("Led ON");    // Messaggio inviato da Arduino
  delay(250);                     // Attesa di 250ms
  digitalWrite(LED_PIN, LED_OFF); // Setta la porta DGT 13 a livello basso (led spento)
  SerialUSB.println("Led OFF");   // Messaggio inviato da Arduino
  delay(250);                     // Attesa di 250ms
}
```

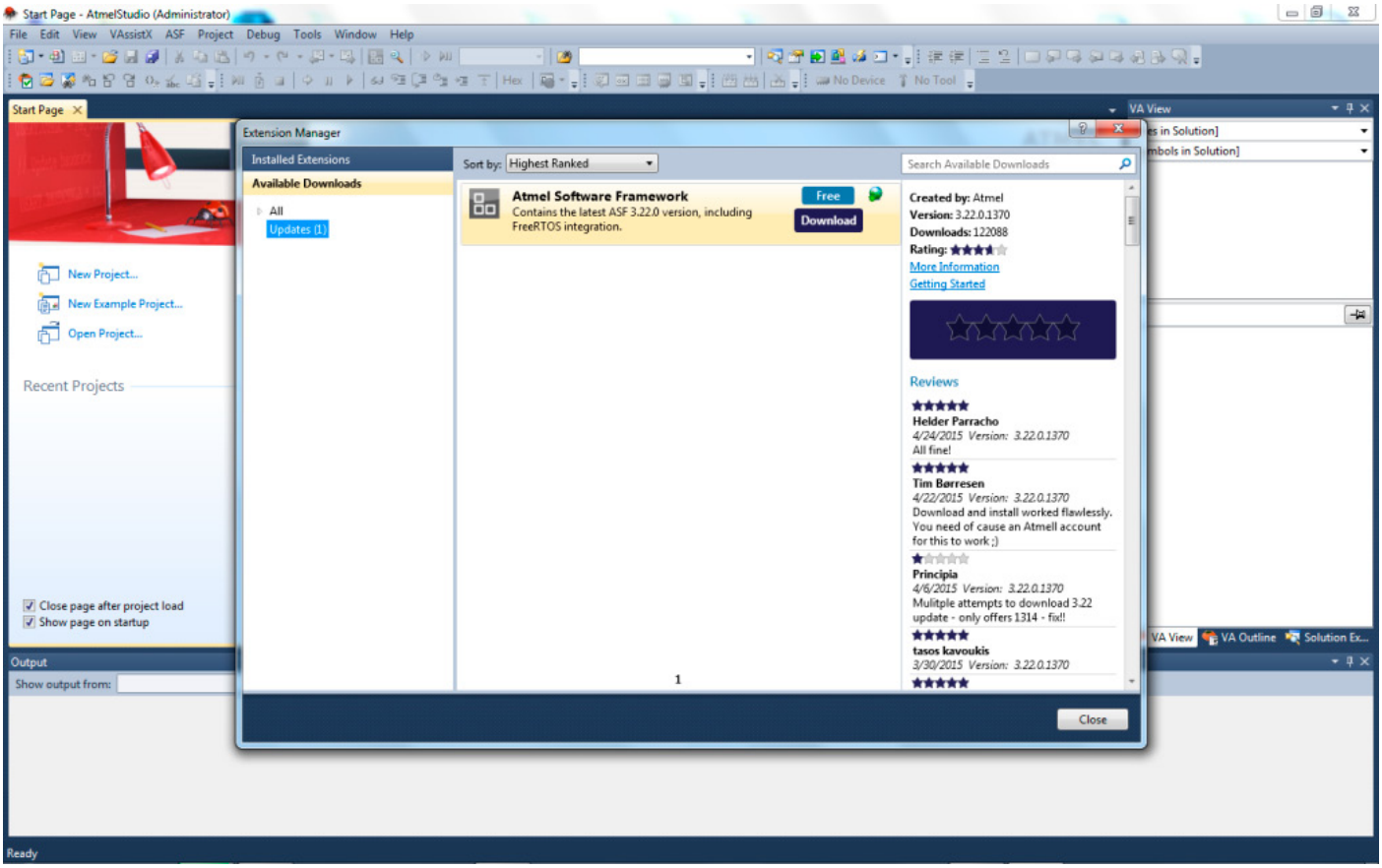
Le istruzioni SerialUSB.println() inserite nel ciclo del programma consentiranno di avvisare l'utente il passaggio avvenuto da una riga all'altra, in corrispondenza del cambio stato sulla porta DGT 13.

Compilare lo sketch e programmare la scheda con la combinazione dei tasti "**Ctrl + U**"; aprire la comunicazione con la serial port avviando il tool "**Monitor seriale**" dal menù "**Strumenti**", ottenendo una nuova finestra dove visualizzare un "debug" elementare.



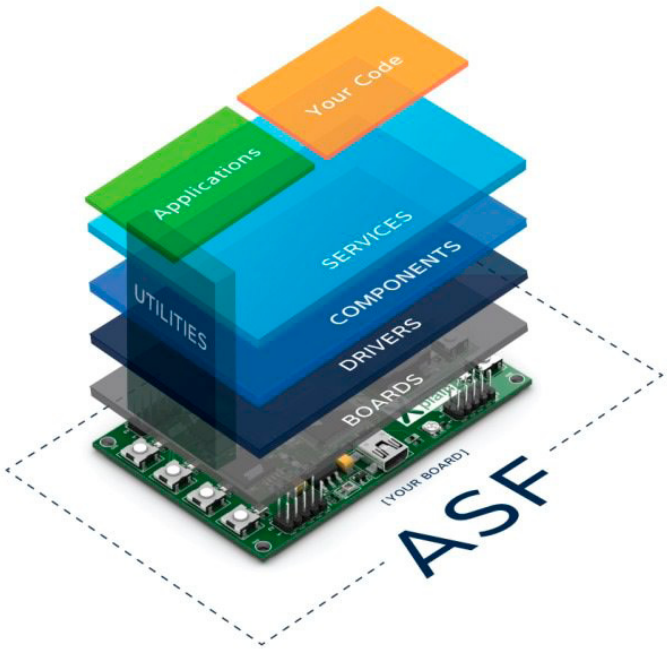
PROGRAMMING E DEBUG CON ATMEL STUDIO

Le prove effettuate e descritte nei seguenti passi sono state realizzate con **Atmel Studio 6.2** e con l'ASF (**l'Atmel Software framework**) aggiornato alla **versione 3.23**; quest'ultima è molto importante che sia aggiornata dato che contiene tutte le librerie per utilizzare al meglio l'integrato SAMD21G18A.



L'aggiornamento può essere scaricato attraverso "**Extension Manager**" dal menù "**Tools**" o dal sito ATMEL.

ASF è una raccolta di software per lo sviluppo di programmi sui microcontrollori di Atmel ed è integrata nella IDE di ATMEL STUDIO. **Il framework è organizzato in strati di livelli** come di seguito:



Per la programmazione e il debug utilizzeremo la porta **USB Programming dell'Arduino M0 Pro**, tale porta è connessa al controller EDGB che simula una porta COM virtuale.

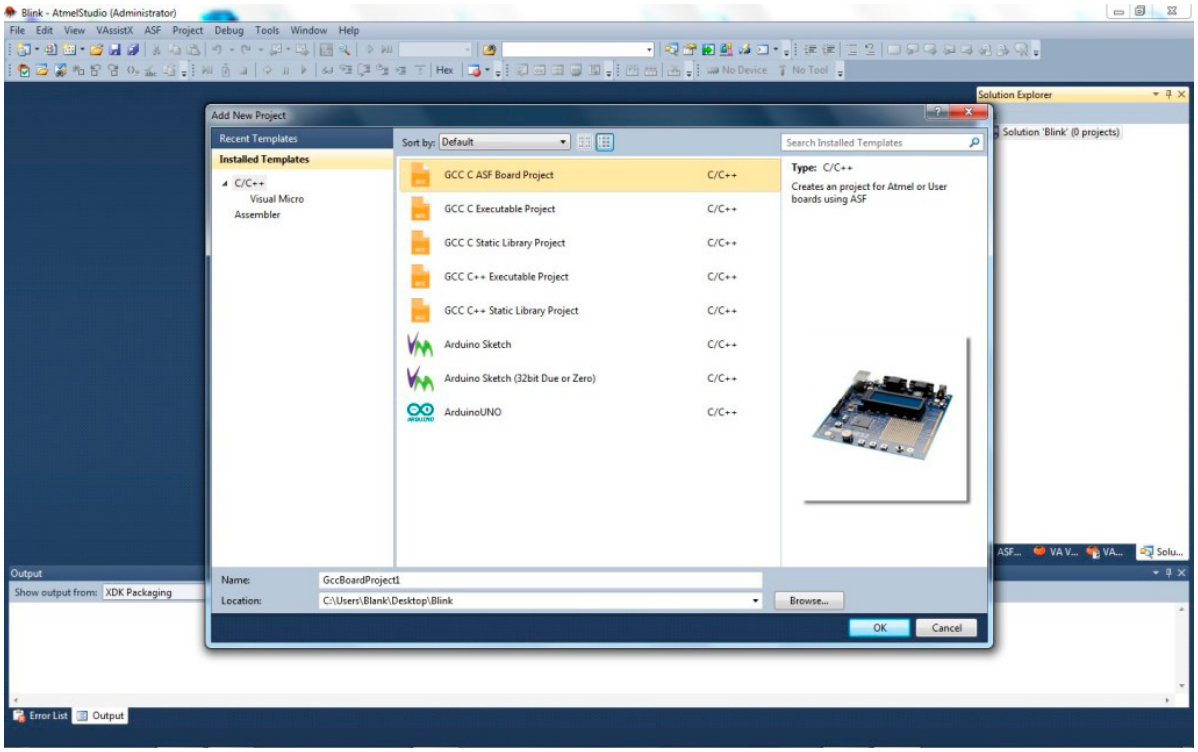
Dopo aver avviato l'ambiente di sviluppo Atmel Studio creare un nuovo progetto da "**File/New/Project...**" .

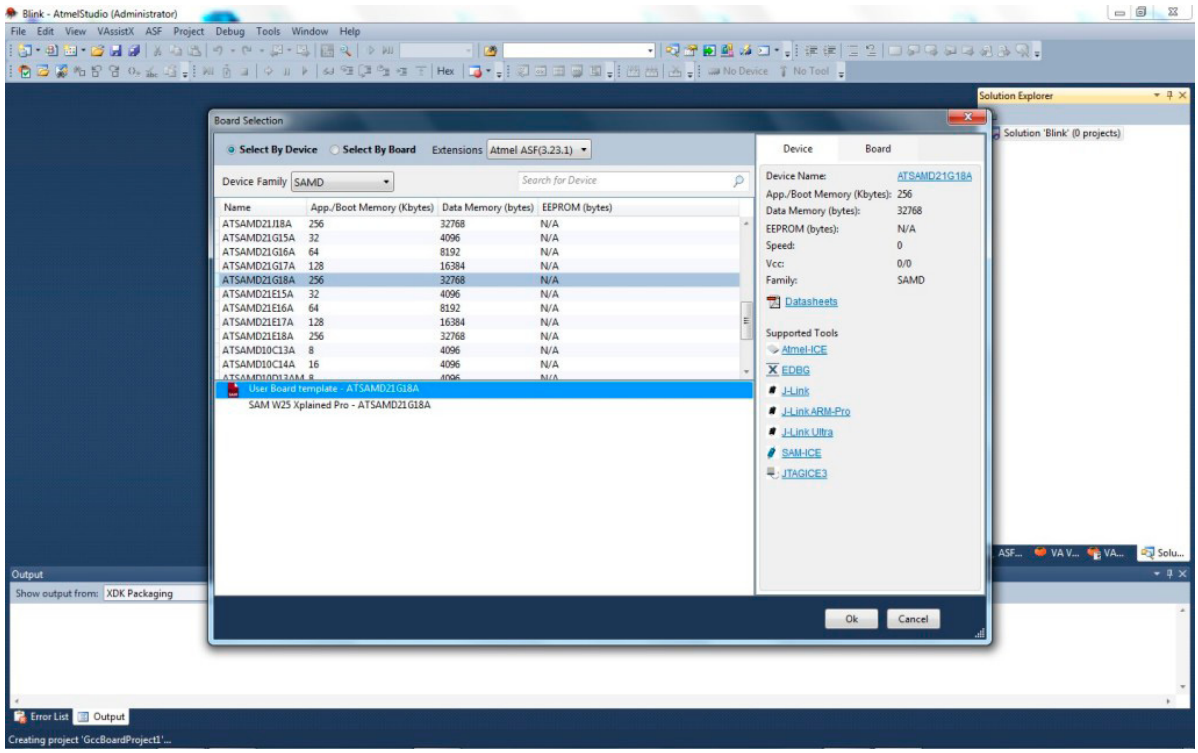
Il progetto da creare sarà basato sul framework perciò selezioneremo "**GCC C ASF Board Project**", completare impostando la directory di salvataggio ed il nome.

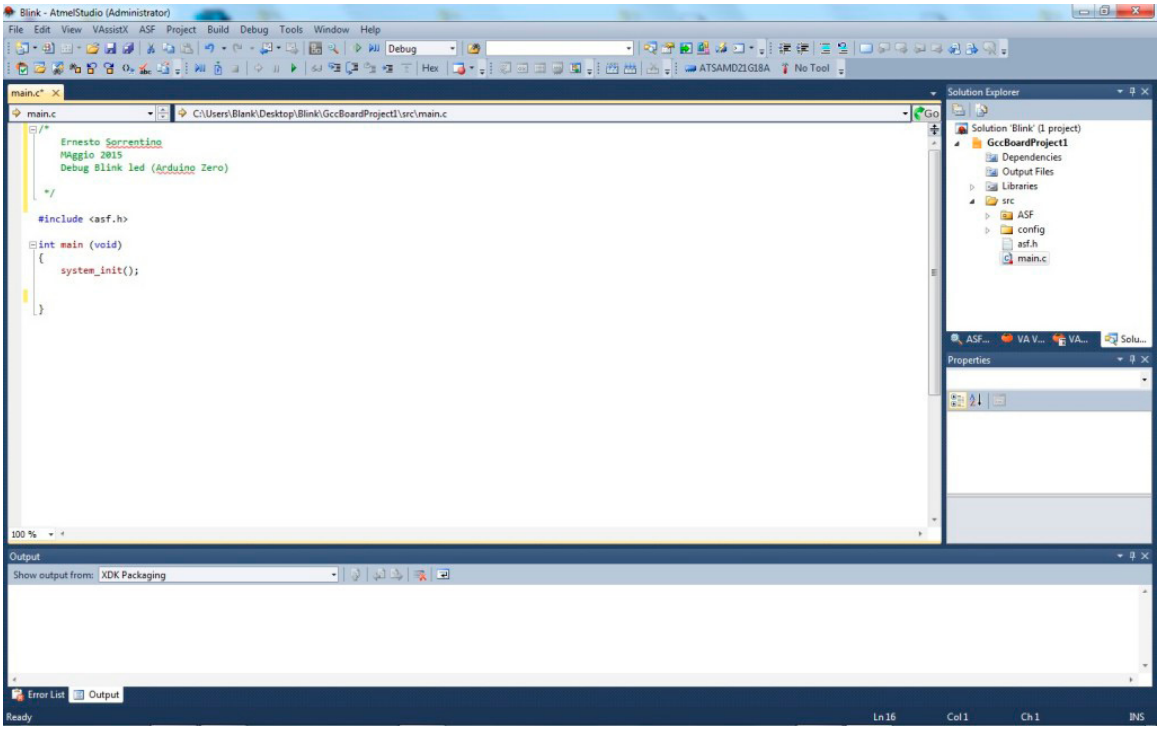
Nella finestra successiva selezionare il micro da programmare, che **per Arduino M0 Pro è il SAMD21G18A**, mentre come dispositivo selezionare "**User Board template - ATSAMD21G18A**", confermare con OK.

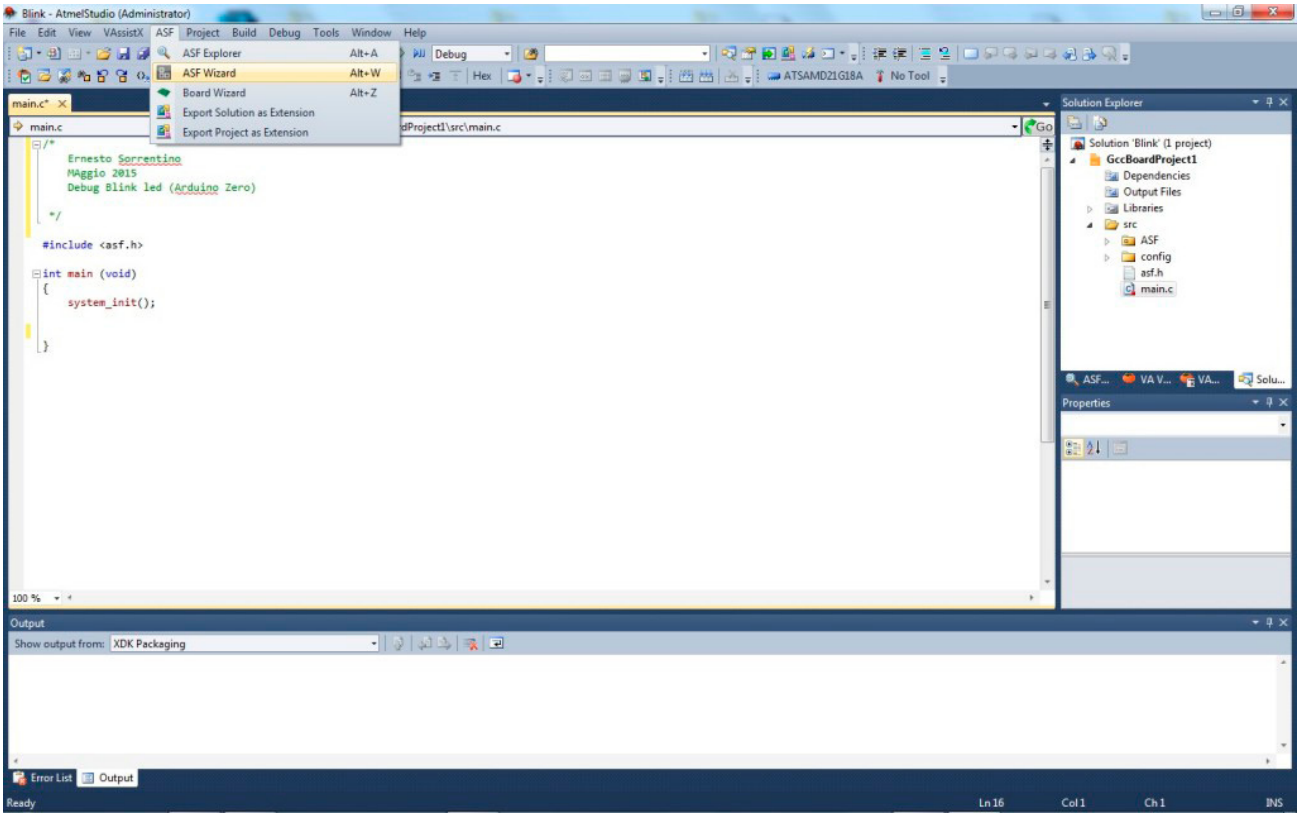
Da "**Solution Explorer**" espandere la directory "**src**" ed aprire il file "**main.c**" con un doppio click; in questo file verrà scritto il codice sorgente ossia lo sketch per far lampeggiare il led. I

BOARDS	rappresenta la board (pin, pulsanti, led, ecc...)
DRIVERS	sono le funzionalità di basso livello per interfacciarsi con le periferiche (DMA, SPI, DAC, ecc...)
COMPONENTS	driver per la funzionalità ai componenti esterni (Display, sensori, ecc...)
SERVICE	funzione che offre un più alto livello, di DRIVERS, per interfacciarsi con le periferiche
APPLICATION	esempi già pronti
UTILITIES	funzioni utilizzate da tutti gli altri strati
YOUR CODE	il nostro programma



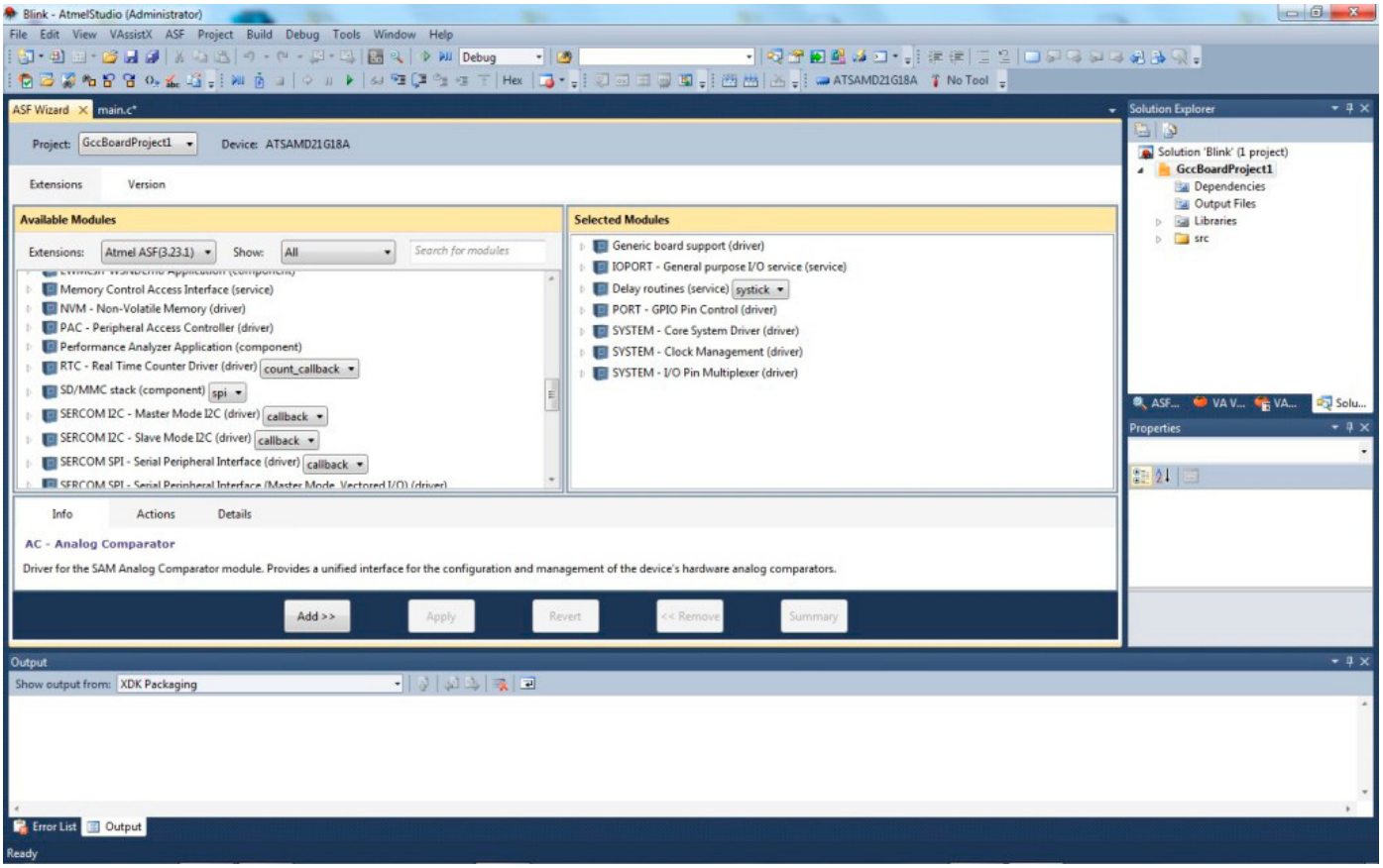






due comandi di default “**#include asf.h” e “system_init()**” **non devono mai essere eliminati**, sono necessari per la compilazione dello sketch in riferimento al tipo di microcontrollore in utilizzo.

Per semplificare la progettazione di un programma l’ASF viene in aiuto con dei moduli aggiuntivi. Questi moduli, come descritto in precedenza, sono parti di codici già scritti e compilati da Atmel per le funzioni più utilizzate, come le **routine di ritardo, la gestione delle periferiche I/O ed altro**. I moduli possono essere aggiunti con l’applicativo “**ASF Wizard**” sotto il menu “**ASF**”.



Ai due già presenti **“Generic board support” e “SYSTEM - Core System Driver”** aggiungere anche :

System - Clock Management (driver);

System – IO Pin Multiplexer (driver);

IOPORT – General Purpose I/O service (service);

PORT – GPIO Pin Control (service);

Delay Routines (service).

Dopo la conferma dei moduli col pulsante “**apply**” ritornare al file “**main.c**” e compilare il progetto col pulsante “**F7**”. Questa procedura aggiungerà al file “**asf.h**” i codici necessari per gestire le funzioni selezionate dal “**ASF Wlazrd**”.

Di seguito lo sketch, da scrivere in ”**main.c**”, per far lampeggiare il led sulla board M0 Pro.

```
#include <asf.h>



#define LED_PIN PIN_PA17

#define LED_ON true

#define LED_OFF false


int main (void)

{

  void portConfig(void){

    struct port_config config_port_pin;

    port_get_config_defaults(&config_port_pin);

    config_port_pin.direction = PORT_PIN_DIR_
OUTPUT;

    port_pin_set_config(LED_PIN, &config_port_
pin);
```

```
 }


 system_init();
 delay_init();
 portConfig();


 while(true)
 {
   delay_ms(250);
   port_pin_set_output_level(LED_PIN, LED_
ON);
   delay_ms(250);
   port_pin_set_output_level(LED_PIN, LED_
OFF);
 }
}
```

In un articolo precedente, "Programmare Arduino UNO con Atmel Studio", è stato spiegato come deve essere strutturato uno sketch con Atmel Studio; potete farvi riferimento per maggior dettagli.
La prima parte del codice è composta dalle dichiarazioni "**#define**" per settare di default lo stato del led

```
#define LED_ON true
#define LED_OFF false
```

e il numero del pin dove è collegato il led al micro

```
#define LED_PIN PIN_PA17
```

NOTA: per l'IDE di Arduino il led è collegato alla porta Digitale n°13 ma in realtà il collegamento "fisico" è al pin 26 del micro che corrisponde alla PORT_A17, fare riferimento allo schema elettrico e al datasheet di Atmel per maggior dettagli.

Successivamente, sotto il comando "**main**", troviamo la funzione "**portConfig**" che racchiude i comandi per settare le porte del micro.

```
struct port_config config_port_pin;
```

crea un modulo di configurazione PORT che può essere compilato per settare ogni singolo pin della porta.

```
port_get_config_defaults(&config_port_pin);
```

Inizializzare il modulo di configurazione dei pin con i valori di default.

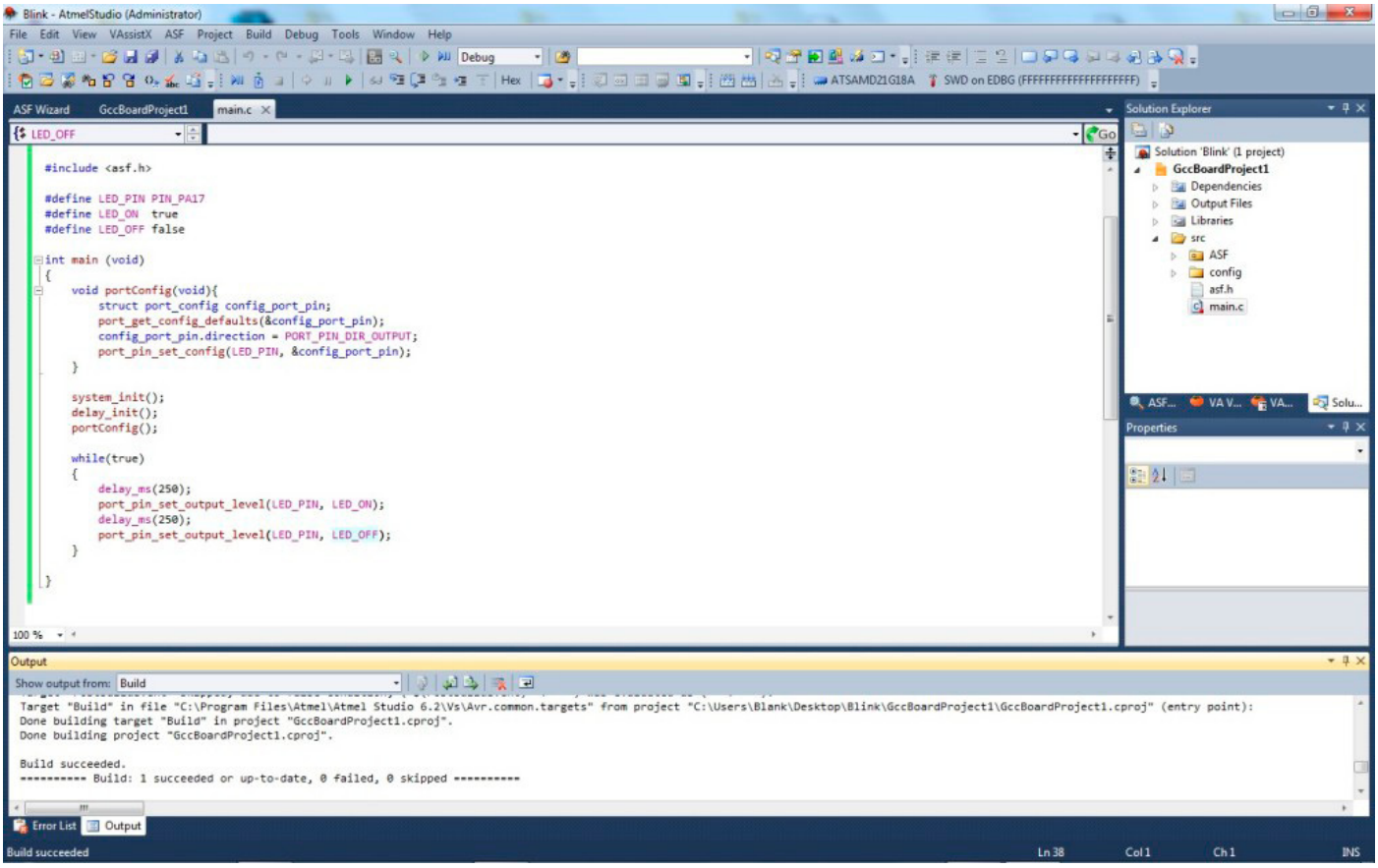
```
config_port_pin.direction = PORT_PIN_DIR_
OUTPUT;
```

Regola il modulo di configurazione per chiedere un pin di uscita.

```
port_pin_set_config(LED_PIN, &config_port_
pin);
```

Abilita la PORT_A17 del micro come uscita
Sempre sotto il comando "**main**" risiedono anche le chiamate all'inizializzazione del sistema, alla routine dei ritardi e allo stesso modulo di configurazione porte che abbiamo appena visto.
Infine nel comando "**while**" troviamo il codice vero e proprio per far lampeggiare il led.

```
delay_ms(250);
```



routine per creare un ritardo di 250ms

```
port_pin_set_output_level(LED_PIN, LED_
ON);
```

porta al livello logico alto la PORT_A17 del micro

dopo un altro ritardo di 250ms ritroviamo lo stesso comando per portare al livello logico basso la PORT_A17 del micro.

Nota: sul portale di Atmel è possibile consultare la guida completa del ASF con tutti i comandi alle API.

Termina la scrittura del codice premere il tasto "**F7**" per compilare lo sketch. Ora non rimane che programmare la board ma prima bisogna settare il programmatore ed il debugger. Selezionare la proprietà del progetto, tramite l'icona a martello, per impostare il debbuger/Programmatore;

selezionare come debbuger/programmer l'**EDGB** di Arduino M0 Pro;

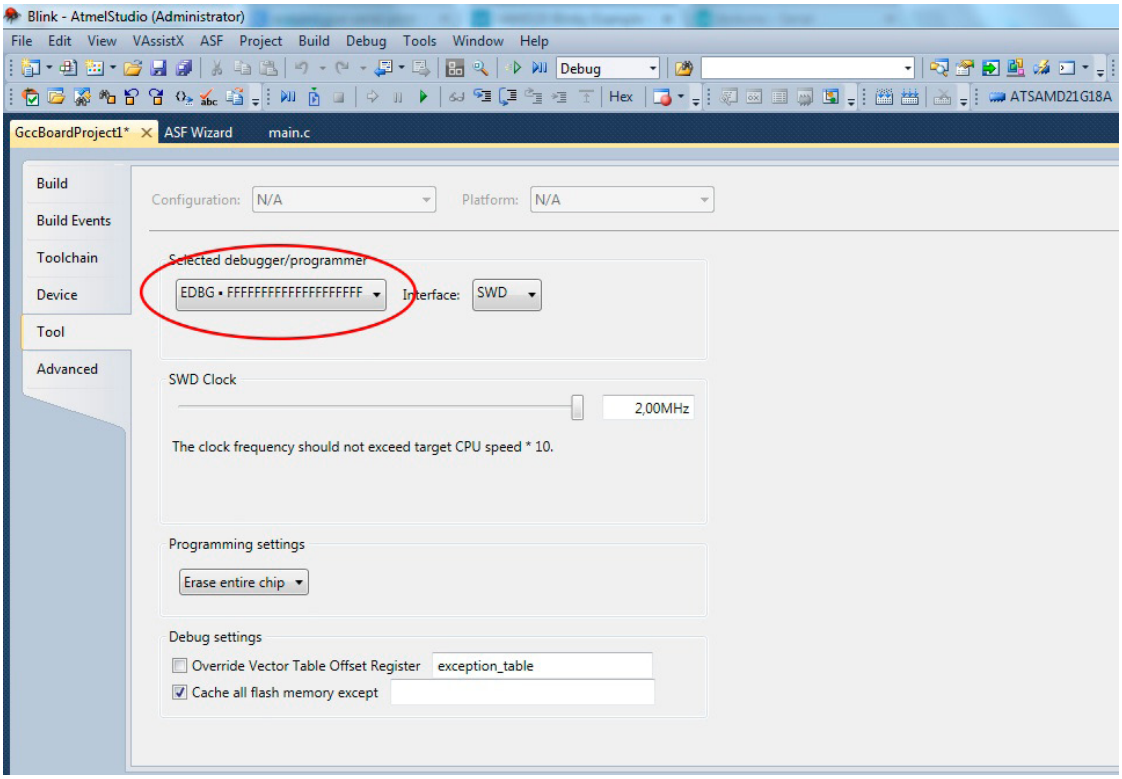
Dal menu "**Tools**" selezionare "**Device Programming**" per scegliere il tipo di dispositivo da programmare, anche in questo caso impostare l'**EDBG**;

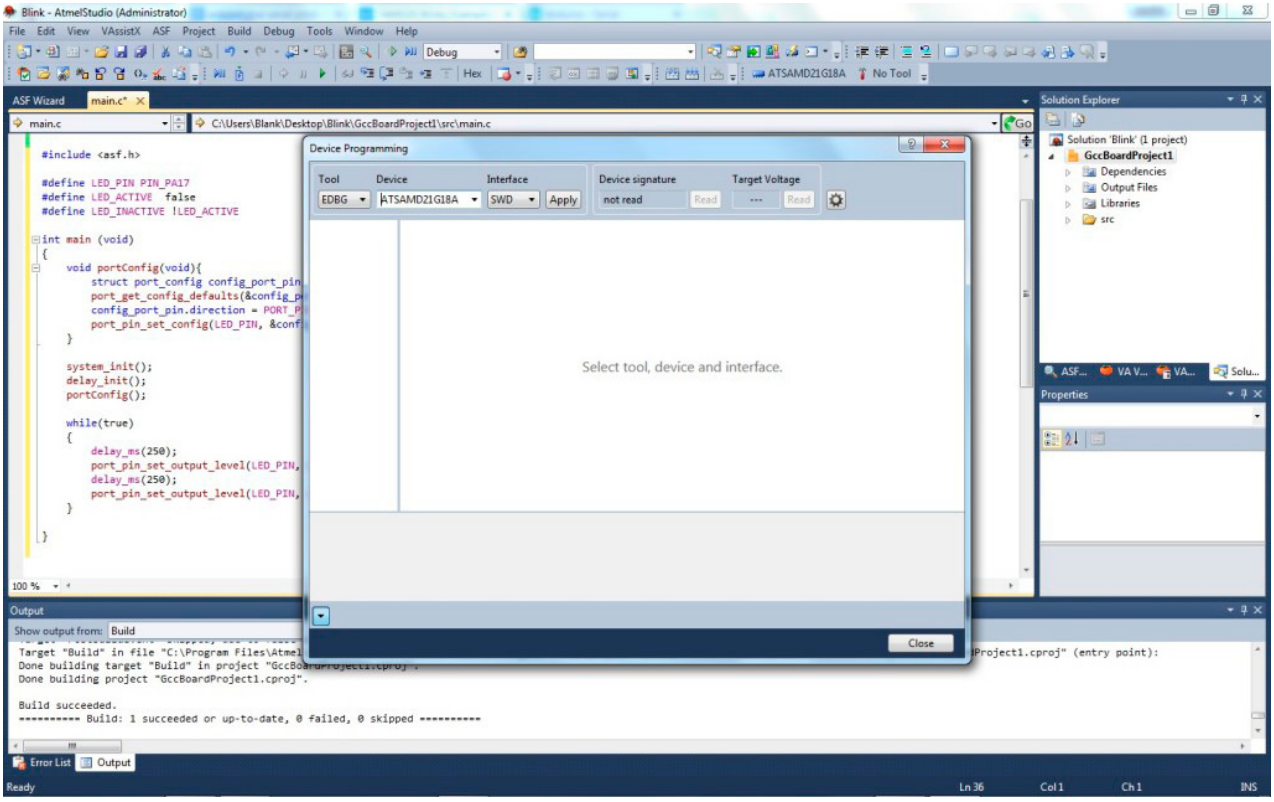
Confermare con "**Apply**" e leggere di dati del dispositivo col pulsante "**Read**";

dal menu **"Fuses" impostare nel registro "NVMCTRL_BOOTPROT" il valore 0x07** e confermare col pulsante "**Program**";

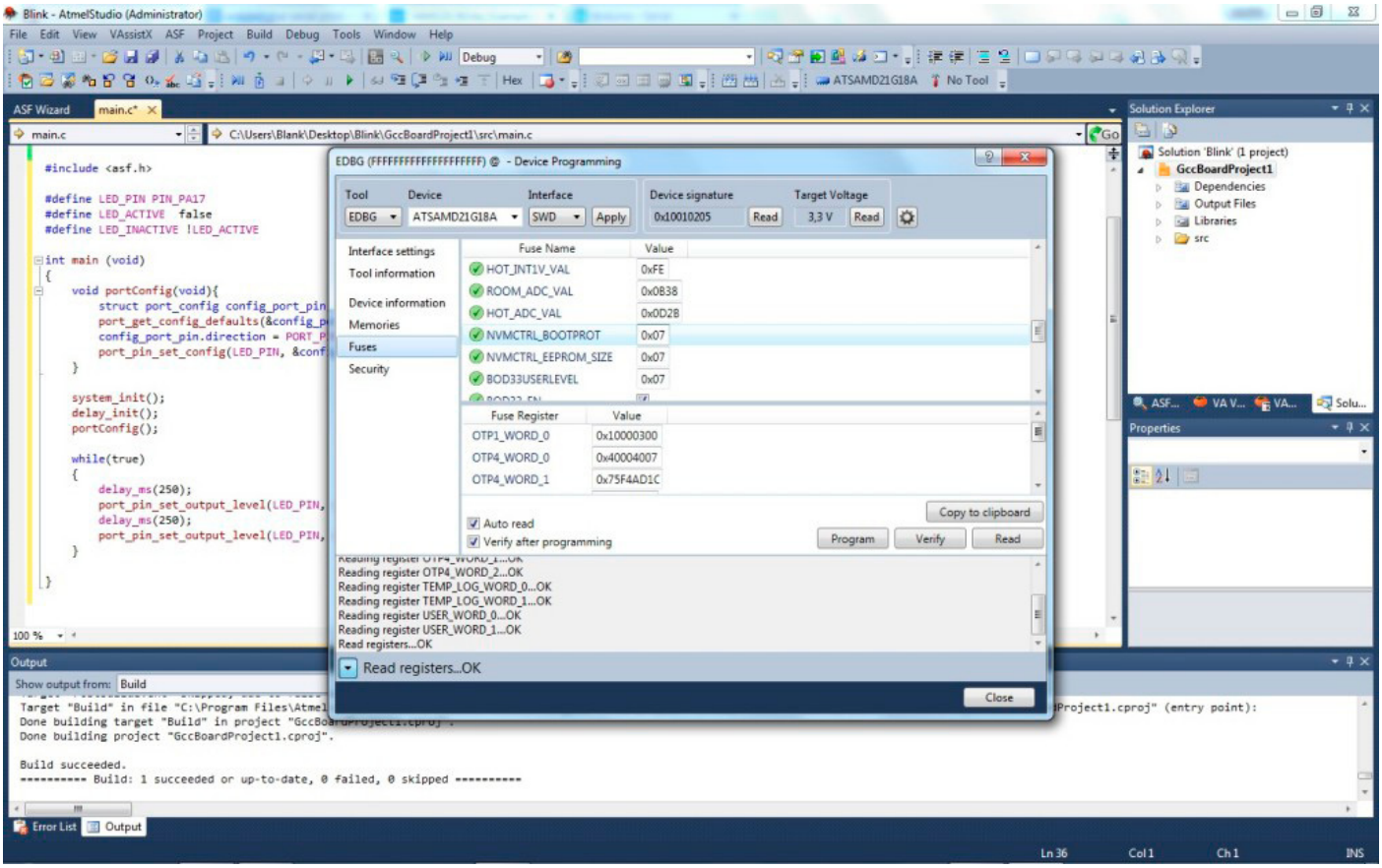
Terminati questi passaggi chiudere la finestra del "**Device Programming**" e ritornare al file "**main.c**".

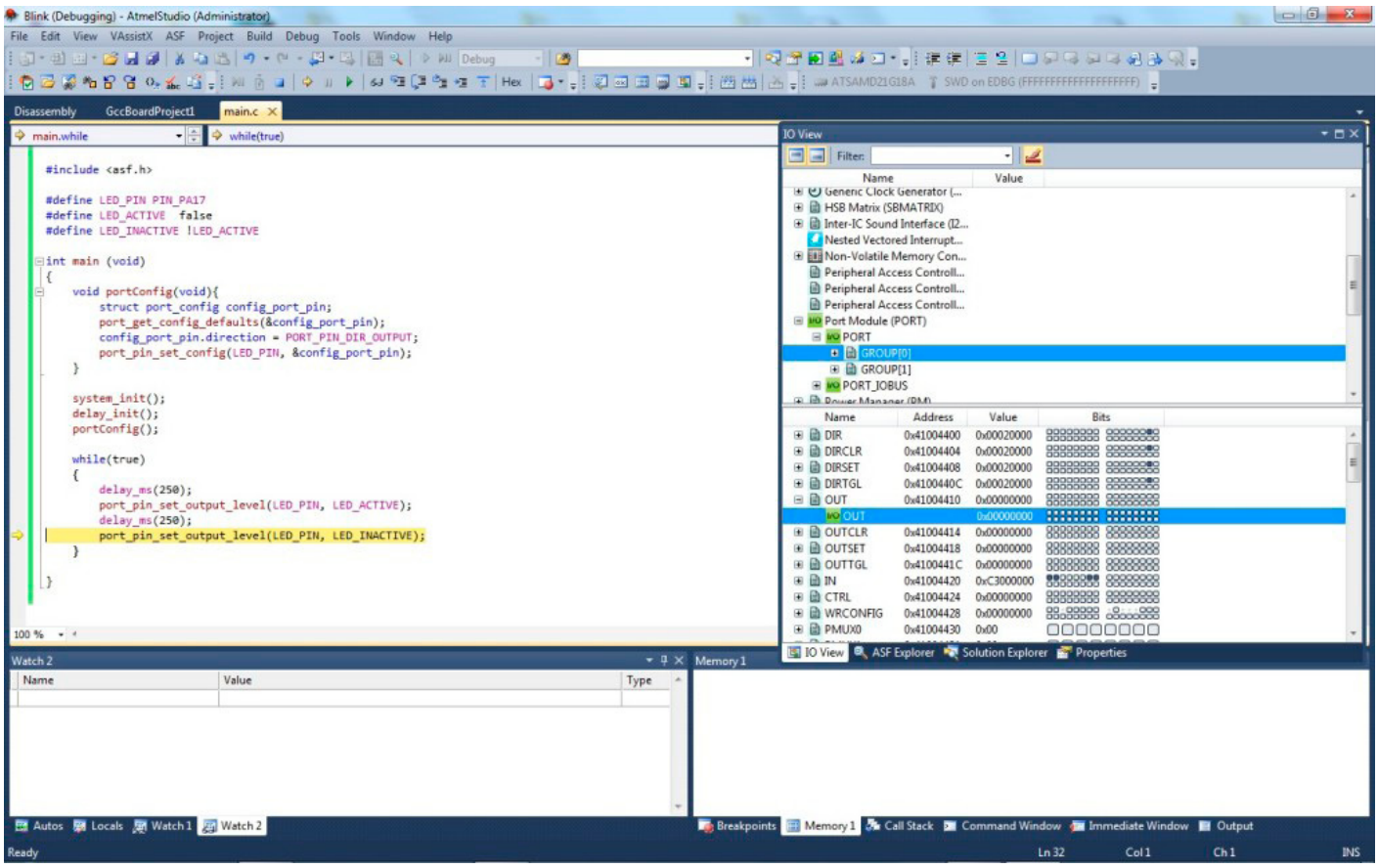
Ora si è pronti per **programmare l'Arduino col tasto "F5" della tastiera oppure avviare il debug col tasto "Alt+F5"**; quest'ultimo comando avvia il **"Debugging and Break"** che permette la lettura del programma in real time passo per passo. Avviata la funzione si noterà, nella finestra "**main.c**", che il programma si ferma





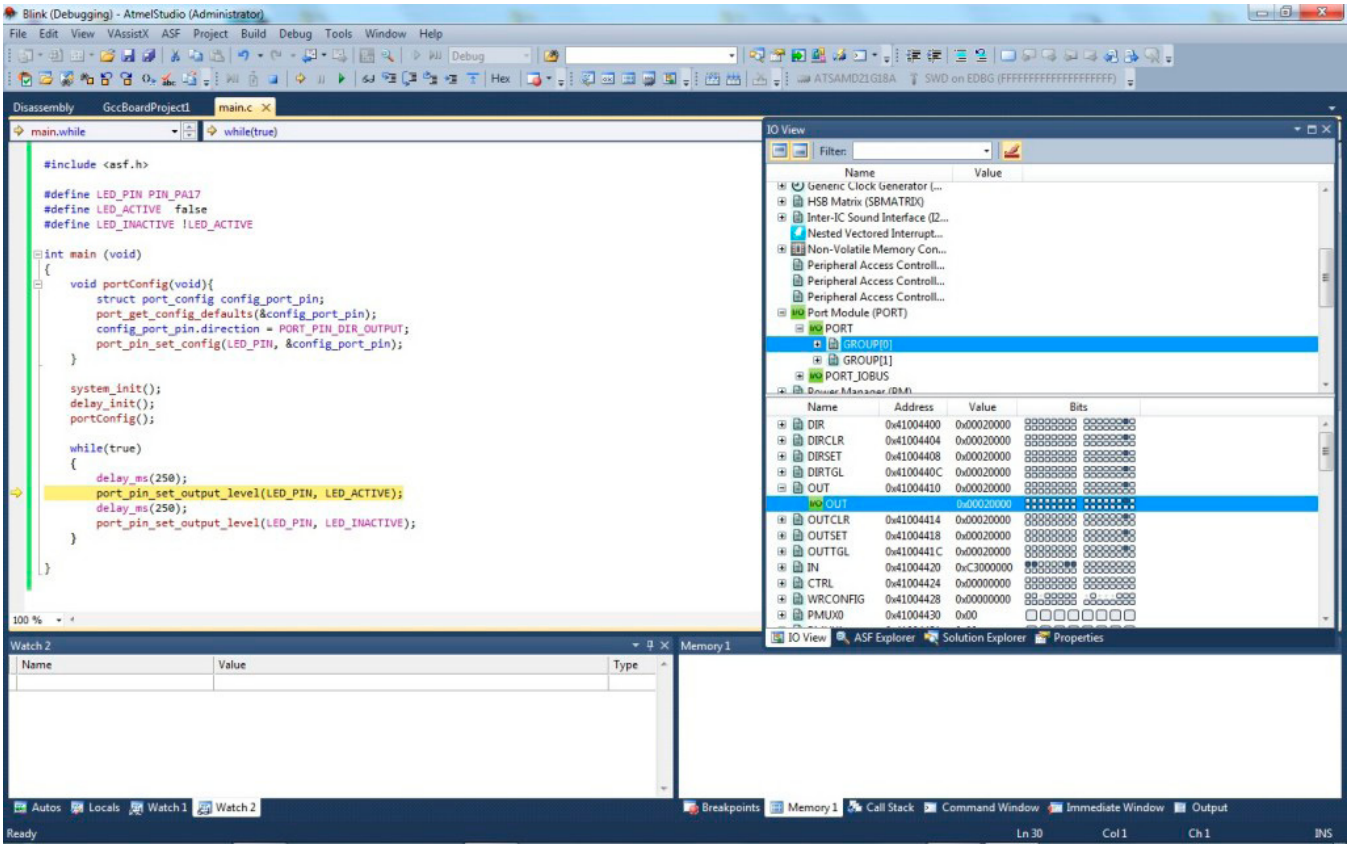
(break) alla prima istruzione dopo il "**main**" con un puntatore giallo, questo permetterà di passare manualmente alle **istruzioni successive utilizzando il tasto "F10" oppure passando all'istruzione precedente utilizzando il tasto "F11"**.






```
#include <asf.h>

#define LED_PIN PIN_PA17
#define LED_ACTIVE  false
#define LED_INACTIVE !LED_ACTIVE

int main (void)
{
    void portConfig(void){
        struct port_config config_port_pin;
        port_get_config_defaults(&config_port_pin);
        config_port_pin.direction = PORT_PIN_DIR_OUTPUT;
        port_pin_set_config(LED_PIN, &config_port_pin);
    }

    system_init();
    delay_init();
    portConfig();

    while(true)
    {
        delay_ms(250);
        port_pin_set_output_level(LED_PIN, LED_ACTIVE);
        delay_ms(250);
        port_pin_set_output_level(LED_PIN, LED_INACTIVE);
    }
}
```

Dalla finestra **“IO View”** è possibile visualizzare l’andamento dei registri del micro, come **“OUT” del “I/O Port Module (PORT)”** che visualizza lo status dei pin del micro ad ogni passaggio del break point sull’istruzione **“port_pin_set_output_level( )”**.



Come si può notare, dall'immagine precedente, il **bit17** del registro **"OUT"** si illumina ad ogni passaggio del break point sul comando che accende il led. Con altrettanto semplicità si possono controllare gli altri registri come quello del **"Power Manager"** o quello dei **"Timer"**; invece nella finestra **"Processor View"** si può controllare l'andamento del **"Program Counter"** o lo **"Status Register"**, tutte funzioni ottime per diagnosticare un malfunzionamento del proprio progetto.

Grazie alla nuova Arduino M0 Pro ora la programmazione per le famose board made in Italy diventa Professionale.

BOOTLOADER

Potrebbe verificarsi, durante le prove di programmazione e debug, la cancellazione accidentale del Bootloader nel micro; nei passi successivi verrà illustrato come riprogrammare il firmware di Arduino utilizzando Atmel Studio.

Prima di tutto bisogna recuperare il file necessario per la M0 che fortunatamente si trova in una cartella d'installazione dell'IDE di Arduino, con precisione all'indirizzo:

C:\Program Files\Arduino\hardware\arduino\samd\bootloaders\zero

Il file è nominato **"Bootloader_zero_pro_150504.hex"**, dove **"_150504"** si riferisce alla data di uscita, pertanto potrebbe cambiare valore in base alla versione dell'IDE. La procedura di programmazione è la seguente:

1. Avviare Atmel Studio;
2. Collegare l'Arduino M0 al PC tramite la porta USB Programming;
3. Selezionare **"Tools -> Device Programming"**;
4. Settare **"EDBG" + "ATSAMD21G18A" + "SWD"**;
5. Confermare col pulsante **"Apply"**;

6. In **“Memories”**, selezionare “Erase Chip”e confermare col pulsante **“Erase now”**;
7. Spostarsi in **“Fuses”**;
8. Impostare il registro **“NVMCTRL_BOOTPROT”** a **0x07**
9. Confermare cliccando sul pulsante “Programming”;
10. Ritornare in “Memories”;
11. Nella sezione “Flash (256KB)” scegliere il file (**Bootloader_zero_pro_150504.hex**);
12. Riscrivere il bootloader premendo il pulsante **“Program”**.

CONCLUSIONI

Esaminate le due tecniche di debug è facile intuire che con Atmel Studio si svolge un lavoro più preciso e professionale. Semplice nell’utilizzo e non necessita scrivere ulteriori comandi nello sketch, come avviene invece per l’IDE di Arduino. Le potenzialità di questo tool sono tante e sorprendenti, come la visualizzazione della “Memory Base Flash” e il “Disassembly” del programma. Non basta un articolo per illustrare tutte le potenzialità di questo debug ma vi invito a provare e sperimentare, anche con semplici sketch, per farvi rendere conto di quanto sia facile utilizzare e apprezzare il fantastico connubio formatosi tra l’Arduino M0 Pro e Atmel Studio.

L’autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell’Articolo.
Di seguito il link per accedere direttamente all’articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-m0-pro-getting-started

Arduino M0 Pro: debug con GDB e OpenOCD

di Ernesto Sorrentino

In un articolo precedente si è descritto come utilizzare Atmel Studio per effettuare il debug sulla M0 Pro; un ambiente di sviluppo con una buona interfaccia grafica per aiutare l'utente a svolgere il lavoro in modo intuitivo e veloce. Al contrario, **GDB è un tool a linee di comando da utilizzare tramite il prompt di DOS, capace di effettuare il debug anche su uno sketch realizzato con l'IDE di Arduino**.

Il **GDB** (**Gnu Source-Level Debugger**) è un software open source, gira su molte piattaforme (sistemi Unix-like e Microsoft Windows) ed è capace di analizzare numerosi linguaggi di programmazione, tra cui Ada, C, C++ e Fortran. Con il GDB è possibile avviare quattro tipi di operazione:

1. Avviare un programma, specificando tutte le componenti che influiscono sul suo comportamento.
2. Far sì che il programma utilizzato si interrompa rispettando le condizioni impostate.
3. Esaminare i processi coinvolti nell'interruzione del programma.
4. Modificare gli elementi nel programma utilizzato, così da poter visionare i risultati delle eventuali correzioni a un dato bug.

In accoppiata al GDB verrà utilizzato anche **OpenOCD** (**Open On-Chip Debugger**), software che mira a fornire il debug in-system programming; ossia un tool che interfaccia l'hardware di debug, in questo caso il controller EDBG, al PC. OpenOCD consente anche di caricare il codice di un progetto nella memoria flash del microcontrollore ma questo lo si vedrà in un'altra occasione, questo articolo si concentrerà per lo più sul GDB.

Anche se **entrambi i software sono implementati nell'Ide di Arduino**, in tal proposito si consiglia l'installazione sul PC della versione 1.7.0 o superiore (reperibile sul sito Arduino.org), per OpenOCD bisogna scaricare una variante, realizzata da Freddie Chopin, dedicata all'utilizzo con Windows. Scompattare l'archivio e copiare l'intera cartella all'indirizzo:

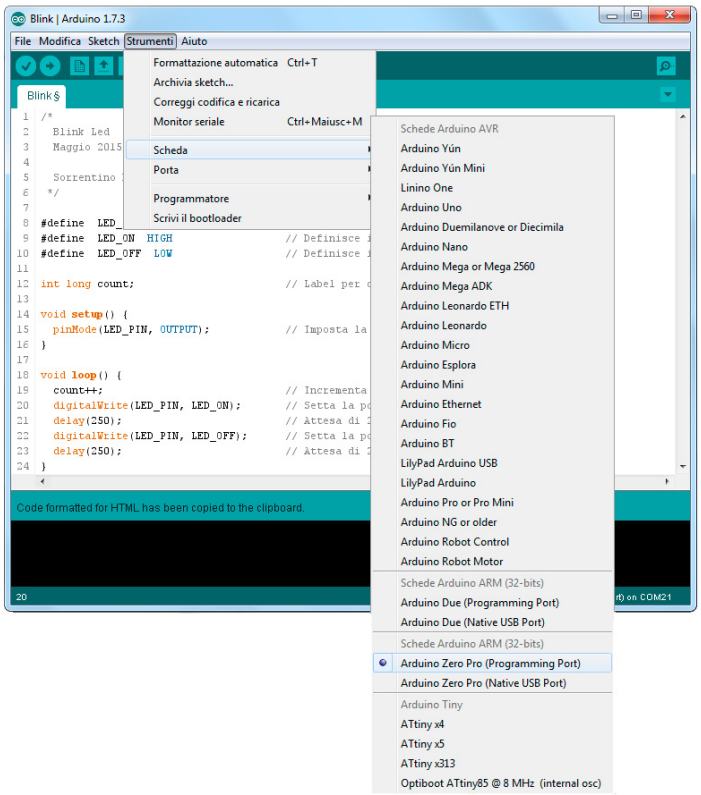
```
C:\Program Files\Arduino\hardware\tools\
```

PROGRAMMAZIONE

Connettere la M0 Pro al PC tramite la porta "usb programming", quella più vicina al connettore di alimentazione, che a sua volta è connessa direttamente al controller EDBG che gestisce il microcontrollore SAMD21G18A sia per la programmazione che per il debug.

Attendere che il sistema operativo riconosca la board, avviare l'IDE di Arduino e settare il tipo

di scheda e porta di comunicazione dal menù "Strumenti".

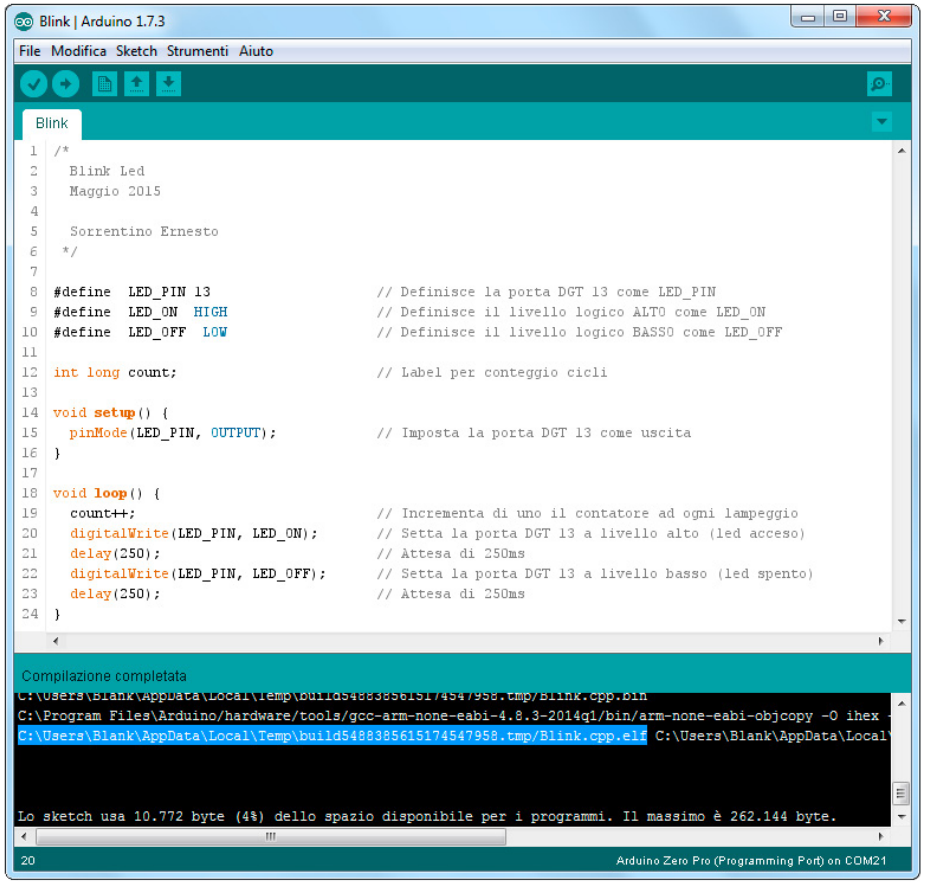


Come sketch di prova utilizzeremo le seguenti istruzioni per fare lampeggiare il led sulla board:

```
/*
  Blink Led
  Maggio 2015

  Sorrentino Ernesto
 */

#define  LED_PIN 13                 // Definisce la porta DGT 13 come LED_PIN
#define  LED_ON  HIGH               // Definisce il livello logico ALTO come LED_ON
#define  LED_OFF  LOW               // Definisce il livello logico BASSO come LED_OFF

int long count;                     // Label per conteggio cicli

void setup() {
  pinMode(LED_PIN, OUTPUT);         // Imposta la porta DGT 13 come uscita
}

void loop() {
  count++;                          // Incrementa di uno il contatore ad ogni lampeggio
  digitalWrite(LED_PIN, LED_ON);    // Setta la porta DGT 13 a livello alto (led acceso)
  delay(250);                       // Attesa di 250ms
  digitalWrite(LED_PIN, LED_OFF);   // Setta la porta DGT 13 a livello basso (led spento)
  delay(250);                       // Attesa di 250ms
}
```

Le istruzioni sono simili a quelle del programma **"blink"** tra gli esempi di Arduino ma con in aggiunta la variabile **"count"** per memorizzare il numero dei lampeggi del led. Salvare il progetto col nome **"Blink"** in un percorso a vostra scelta. Compilare lo sketch e programmare la scheda premendo i tasti **"Ctrl + U"**. Dall'output dettagliato individuare l'indirizzo di destinazione del file **".elf"** appena creato; **file (Executable and Linkable Format) contenente informazioni per il debug relativo allo sketch**.

Copiare **"Blink.cpp.elf"** nella root di **"c:\"**; questo passaggio non è obbligatorio ma facilita il comando per caricarlo nel GDB.



NOTA: Il file ".elf" è associato allo sketch in uso, per tanto ogni qualvolta che si modifica il programma si genera un nuovo file".elf", che dovrà essere sostituito al precedente.

AVVIO OPENOCD E GDB

Per avviare e gestire il programma bisogna utilizzare il prompt di DOS, reperibile all'indirizzo:

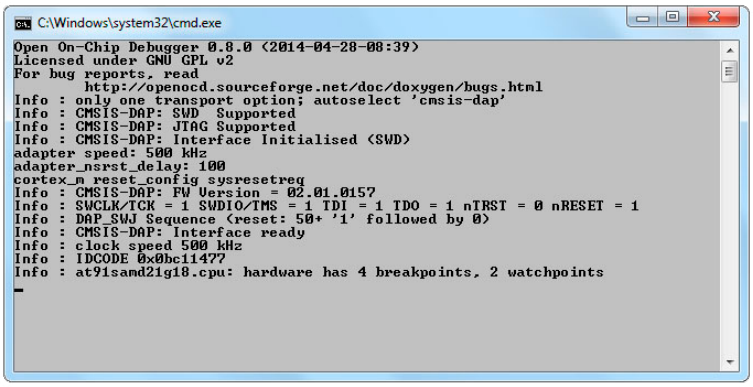
```
c:\Windows\system32\cmd.exe
```

Dalla directory "User" spostarsi nella cartella d'installazione dell'IDE di Arduino e digitare il comando:

```
/hardware/tools/OpenOCD-0.9.0/bin/openocd -s hardware/tools/OpenOCD-0.9.0/scripts/ -f hardware/arduino/samd/variants/arduino_zero/openocd_scripts/arduino_zero.cfg
```

```
/*
  Blink Led
  Maggio 2015

  Sorrentino Ernesto
 */

#define  LED_PIN 13                 // Definisce la porta DGT 13 come LED_PIN
#define  LED_ON  HIGH               // Definisce il livello logico ALTO come LED_ON
#define  LED_OFF  LOW               // Definisce il livello logico BASSO come LED_OFF

int long count;                     // Label per conteggio cicli

void setup() {
  pinMode(LED_PIN, OUTPUT);         // Imposta la porta DGT 13 come uscita
}

void loop() {
  count++;                          // Incrementa di uno il contatore a ogni lampeggio
  digitalWrite(LED_PIN, LED_ON);    // Setta la porta DGT 13 a livello alto (led acceso)
  delay(250);                       // Attesa di 250ms
  digitalWrite(LED_PIN, LED_OFF);   // Setta la porta DGT 13 a livello basso (led spento)
  delay(250);                       // Attesa di 250ms
}
```

L'argomento **"-s"** definisce la cartella con i file di configurazione di OpenOCD mentre **"-f"** carica il file di configurazione della Mo Pro. Inviati i comandi si stabilirà una connessione, tramite il servizio telnet, **tra le porte 4444 (server telnet) e 3333 (localhost)**. Ottenendo il seguente risultato:



Ora che la connessione è attiva non resta che avviare il programma GDB in una nuova finestra del prompt di DOS, e digitare:

```
\hardware\tools\gcc-arm-none-eabi-4.8.3-2014q1\bin\arm-none-eabi-gdb -d "percorso del file Blink.ino"
```

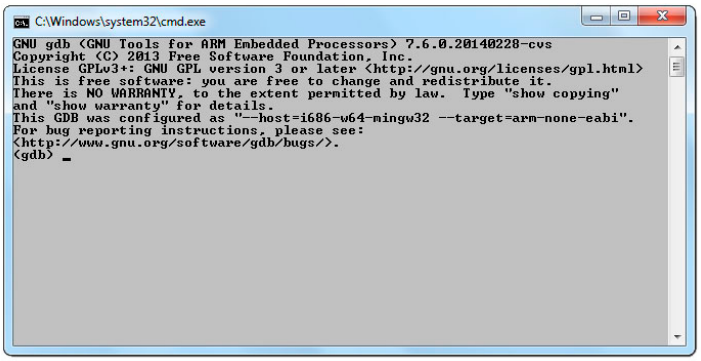
L'argomento **"-d"** definisce la cartella dove si trova lo sketch originale, questo servirà per poter caricare sul video, in un secondo momento, le istruzioni da esaminare. Avviato il programma si dovrebbe vedere la linea di comando "(GDB)_".

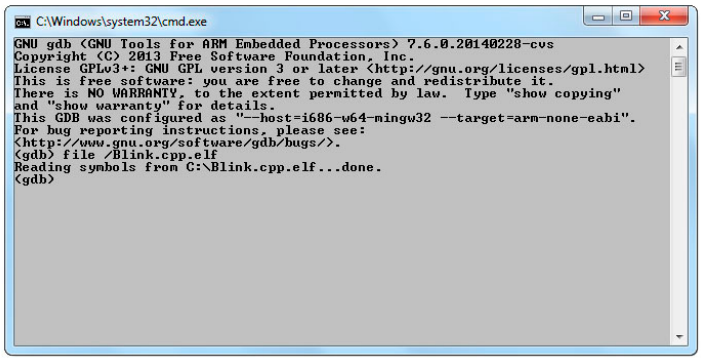
DEBUG

Partiamo col caricare il file **".elf"**, copiato in precedenza nella root di **"C:\"**, con l'istruzione:

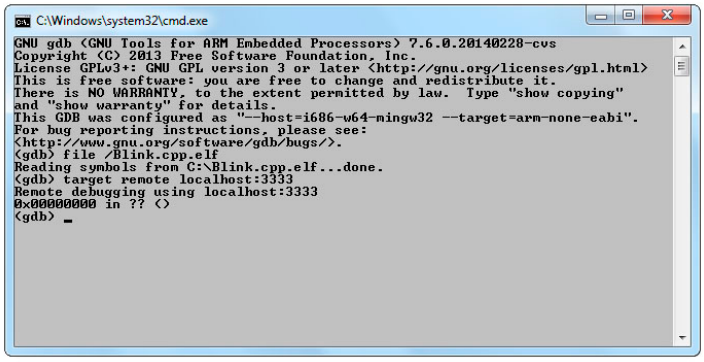
```
file /Blink.cpp.elf
```

Successivamente bisogna indicare la porta host






```
target remote localhost:3333
```



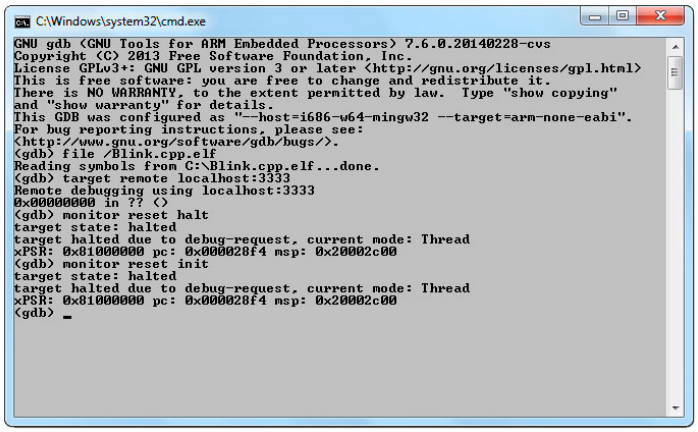
per poter dialogare con l'EDBG tramite OpenOCD, digitando:

Ora si è pronti per partire col debug. Il primo comando che invieremo è quello di fermare l'esecuzione dello sketch sulla board con:

```
monitor reset halt
```

Dopo l'inoltro del comando si noterà che anche il led sulla scheda smetterà di lampeggiare. Il programma si è fermato ma in un punto qualsiasi dello sketch, per far in modo che il ciclo ricominci dalla prima istruzione bisogna riavviare anche la CPU, col comando:

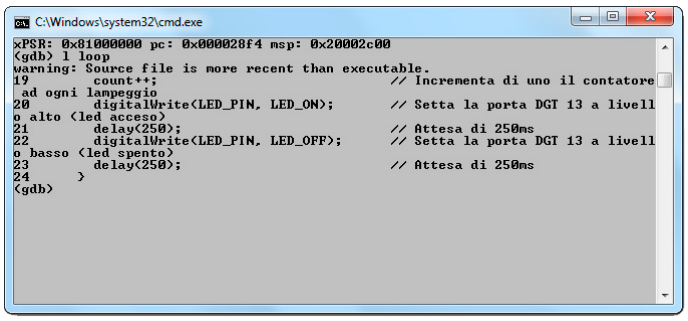
```
monitor reset init
```



Ora proviamo a inserire due breakpoint ma per essere sicuri su quale riga di comando inserirli carichiamo le istruzioni sul video digitando:

```
l loop
```

questa funzione permette di visualizzare l'intero modulo "loop"

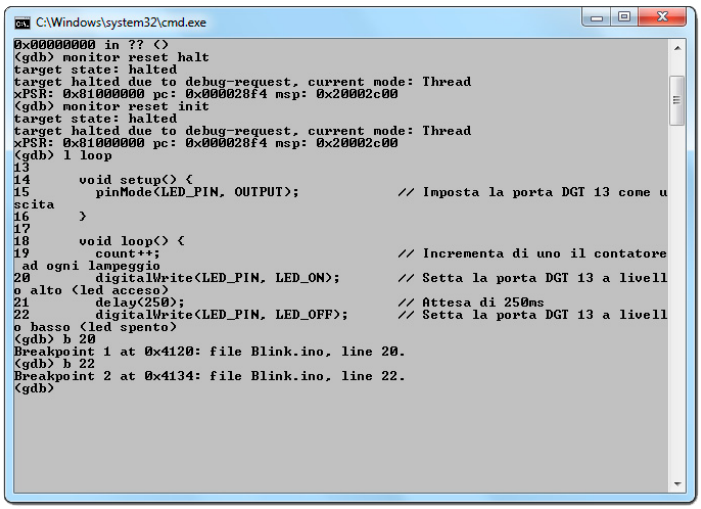


I breakpoint devono essere inseriti digitando la lettera **"b" e il numero della riga** dove inserirlo. Come esempio aggiungeremo due bp; uno alla riga 20 corrispondente al led ON e uno alla riga 22 per il led OFF

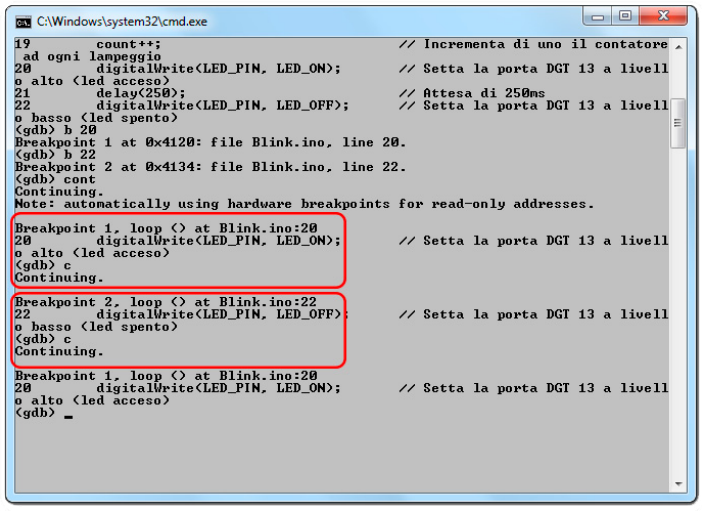
```
b 20
b 22
```

Inseriti tutti i bp non resta che far ripartire lo sketch, col comando:

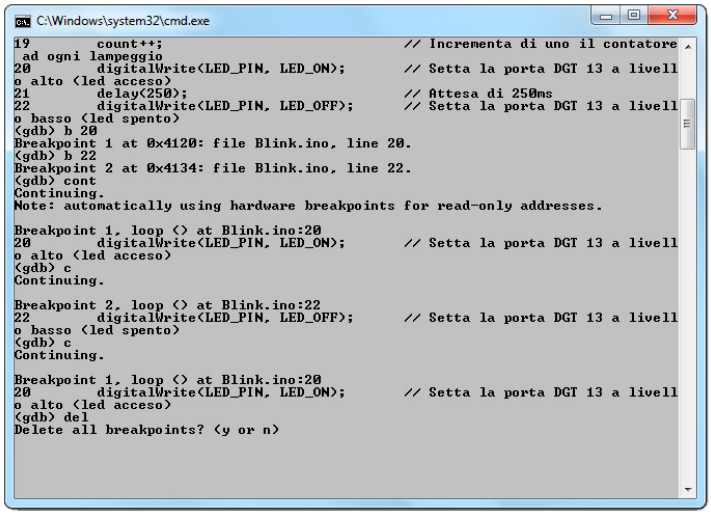
```
cont
```



Arduino riprende l'esecuzione del programma e si ferma alla riga n°20 dato che incontra un bp, cioè il n°1 (led ON). Digitando, invece, la lettera **"c"** lo sketch prosegue fino al successivo bp (riga 22, led OFF). Ripetendo quest'ultimo comando è possibile esaminare il programma passo per passo; **ottimo per individuare i bug di uno sketch**. Come si può notare anche il led sulla board cambia di stato, in corrispondenza all'istruzione precedente al bp in cui si è fermato il programma.



Un'altra esempio di prova che eseguiremo sarà quello di leggere e modificare il contenuto della variabile **"count"**. Per fare ciò dovremo prima eliminare tutti i breakpoint con l'invio del comando **"del"**. Un messaggio chiederà conferma per l'eliminazione di tutti i bp; digitare **"y"** per continuare.



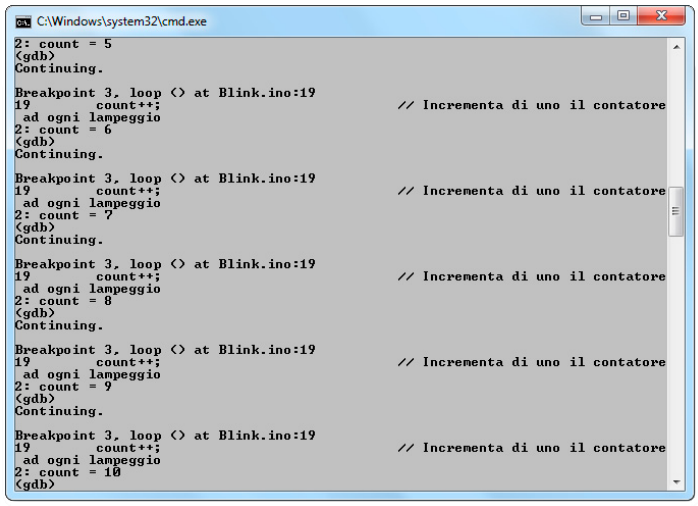
Inserire un nuovo bp alla riga n° 19, corrispondente all'istruzione **"count ++"**, in modo da **eseguire un solo ciclo di "loop" a ogni invio del comando "c"**. Per ultimo aggiungere la funzione display per visualizzare il contenuto della variabile **"count"**. I comandi da digitare per svolgere tutta la procedura sono nel seguente ordine:

```
b 19
 cont
 display /d count
 c
```

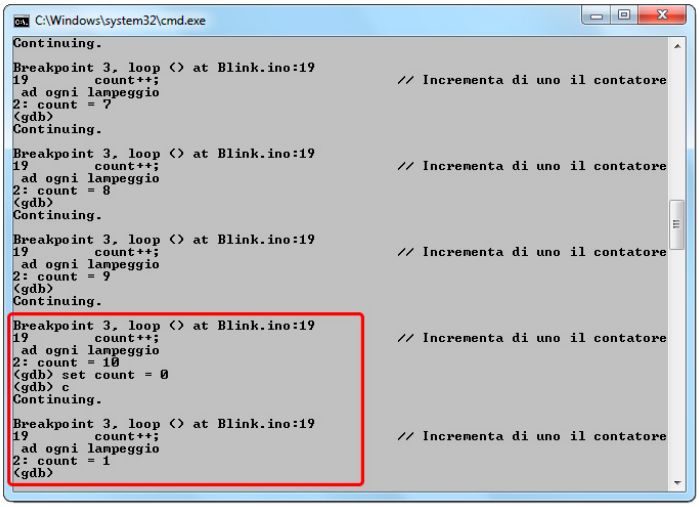
A ogni invio del comando **"c"** si noterà un lampeggio del led di Arduino mentre sul monitor verrà visualizzato, in decimale, il valore contenuto nella variabile **"count"**, ossia il numero dei lampeggi effettuati.

Utilizzando il comando **"set"** è possibile modificare il contenuto della variabile; provare ad azzerare **"count"** digitando:

```
set count = 0
```



Effettuare un altro ciclo di **"loop"** col comando **"c"** per verificare il valore contenuto in **"count"**.



TOOL OCD/GDB_CONFIGURATOR

Per semplificare l'avvio dei programmi **OpenOCD e GDB** ho realizzato un tool, con Processing e PGUI, che assembla **due file tipo Batch per connettere e avviare le applicazioni in automatico**. Questo tool è in download a fine articolo.

L'utilizzo è semplice basta caricare, tramite dei pulsanti, i file indicati precedentemente nell'articolo poi salvare il Batch in un percorso a vostra scelta. Per avviare la connessione fare doppio click sul file appena creato. La realizzazione del Batch OpenOCD avviene tramite l'uso dei seguenti pulsanti:

Pulsante OpenOCD: Caricare il file "OpenOCD.exe" dall'indirizzo:

C:\Program Files\Arduino\hardware\tools\openocd-0.9.0\bin-x64\

Pulsante OpenOCD cfg: Caricare il file di configurazione "openocd-usb.cfg" dall'indirizzo:

C:\Program Files\Arduino\hardware\tools\openocd-0.9.0\scripts\target\

Pulsante Arduino cfg: Caricare il file di configurazione della M0Pro "arduino_zero.cfg" dall'indirizzo:

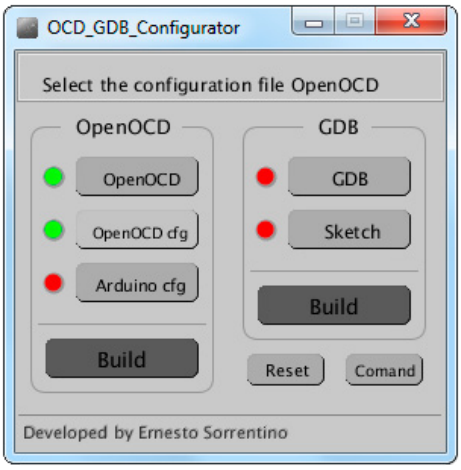
C:\Program Files\Arduino\hardware\arduino\samd\variants\arduino_zero\openocd_scripts\

Si noti che a ogni file caricato si illumina di verde il led posto sulla sinistra del pulsante, questo sta a indicare che l'indirizzo è stato memorizzato nell'applicativo per i successivi utilizzi; rendendo così più veloce la creazione dei nuovi file batch. In caso d'indirizzi errati o se si vuol modificare la loro destinazione è possibile azzerare lo storico col pulsante **"reset"**.

Per creare il secondo Batch, quello del GDB, utilizzare i pulsanti:

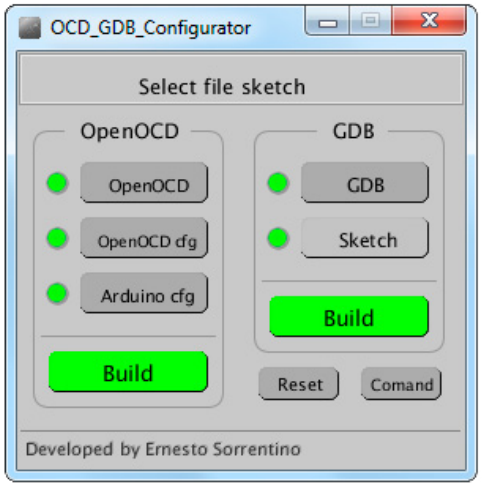
Pulsante GDB: Caricare il file "arm-none-eabi-gdb.exe" dall'indirizzo:

C:\Program Files\Arduino\hardware\tools\gcc-arm-none-eabi-4.8.3-2014q1\bin\



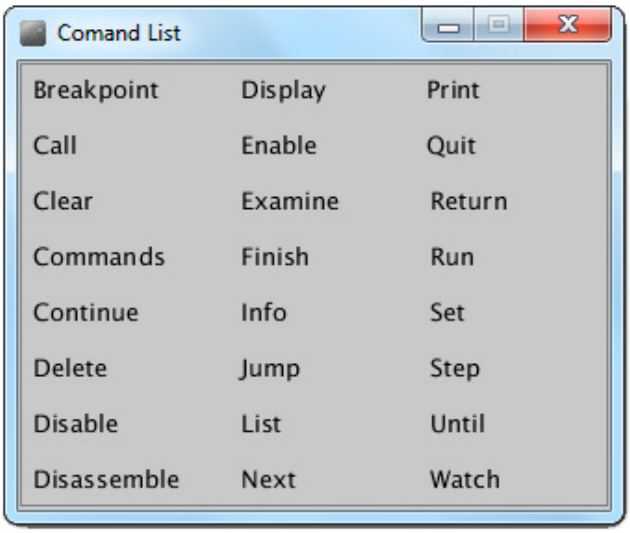
Pulsante Sketch: Indicare la cartella dove è salvato lo sketch **".ino"**

Al termine del caricamento dei file i **pulsanti "Build" si illumineranno di verde**, sarà quindi possibile salvare con nome i Batch nel percorso



desiderato.

Per facilitare l'utilizzo del programma GDB è stata aggiunta una guida ai comandi con relative spiegazioni d'uso, accessibile tramite il pulsante **"comand"**.



CONCLUSIONE

Anche se l'articolo mostra l'utilizzo del GDB solo con linee di comando in realtà, con opportune configurazioni, è possibile integrarlo in molti programmi con una interfaccia grafica, ad esempio con DDD, CodeLite, Eclipse e altri ancora proposti dallo stesso creatore; rendendo più semplice l'uso di questo ottimo tool di debug.

Configurator.part1

Configurator.part2

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/arduino-m0-pro-debug-con-gdb-e-openocd

Progetto di una libreria per LCD 16x2 compatibile con Arduino M0 Pro

di Mario Mottula

L'idea di sviluppare la libreria è nata da una mia esigenza di progettazione, visto che quella messa a disposizione dall'IDE di Arduino non era ancora stata resa compatibile con il controller di Arduino M0 Pro, avendo la necessità di controllare un display di questo tipo ho deciso di crearla da zero.

Componenti utilizzati

- *Board Arduino M0 Pro*
- *LCD Data Vision DV-16275* compatibile con i segnali di dato a 3.3 Volt provenienti dalla M0 Pro (da notare che il display va comunque alimentato a 5 Volt, ma riconosce perfettamente i segnali d'ingresso a 3.3 Volt come visibile a pagina 21 del datasheet scaricabile dal link seguente Datasheet dv-12765.
- *Resistenza 220 Ohm 1/4 Watt* per impostazione contrasto standard LCD tra pin 3 e GND

PREPARAZIONE DI UNA NUOVA LIBRERIA

Una libreria per Arduino è costituita da due file fondamentali che sono un file .h ed un file .cpp, ed un file opzionale che è il keywords.txt.

Il file .h o header file contiene la dichiarazione della classe, tutte le funzioni usate nella stessa e le definizioni delle variabili necessarie, il file .cpp contiene il codice vero e proprio di ogni funzione , mentre il file keywords.txt, che ripeto è opzionale, contiene le parole chiave e permette di visualizzare le chiamate alla libreria di colore diverso ogni qualvolta si invoca una singola funzione nello sketch.

Di fondamentale importanza è che sia la cartella che il file .h ed il file .ccp abbiano lo stesso nome, come vedremo di seguito, quindi dovremo innanzitutto creare una cartella con un nome che descriva intuitivamente per questione di comodità il nome della libreria, che contenga sia il file .h che il file .cpp ed eventualmente anche il keyword.txt **tutti con lo stesso nome**.

Nel mio caso ho nominato libreria "LCD_m0" e di conseguenza, come accennato precedentemente, ho creato una cartella con lo stesso nome e con all'interno i tre file di cui abbiamo discusso prima, ottenendo il risultato seguente (Fig.1):

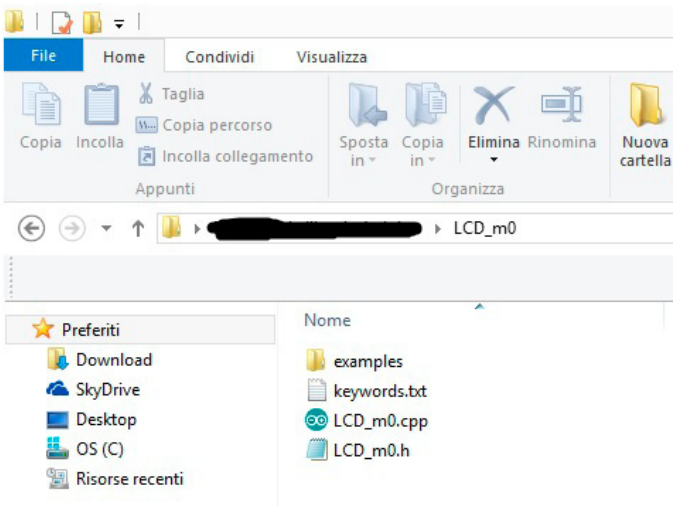


Fig.1

Nella cartella si nota una sottocartella con nome examples, della quale parleremo in seguito e

che sostanzialmente contiene gli esempi della libreria sviluppata che vengono messi a disposizone da chi la progetta, per facilitare la comprensione d'utilizzo della stessa.

FILE LCD_M0.H

Entriamo a questo punto in merito al file LCD_m0.h e vediamo come è strutturato (Fig.2).

```
#ifndef LCD_m0_h
#define LCD_m0_h

#include "Arduino.h"


class LCD_m0
{

public:
        .
        .
        .
        .
private:
        .
        .
        .

};

#endif
```

Fig.2

Come si può vedere ogni file header che definisce una classe, è delimitato dalle direttive ***#ifndef*** (Con al seguito il nome della libreria e l'identificativo **_h**) e ***#endif*** che devono sempre essere presenti.

Va inoltre usato sempre il ***#define*** seguito dal nome della libreria, in questo caso LCD_m0_h, che definisce la costante LCD_m0_h.

Detto questo vediamo cosa bisogna inserire all'interno della classe, che chiameremo sempre con il nome assegnato alla libreria, in questo caso LCD_m0.

La dichiarazione public: In questa zona del programma si inseriranno tutte le funzioni della libreria che ovviamente dovranno essere rese pubbliche dalla classe, per poter essere richiamate in qualunque punto dell'applicazione.

Aprendo ad esempio il file LCD_m0.h da me creato, si può notare come in public vengono definite le funzioni :

- void Inizializza_LCD(int pin_RS,int pin_EN,int pin_D4,int pin_D5,int pin_D6,int pin_D7);
- void Cursore_ON();
- void Cursore_OFF();
- void Scrivi_Car_PC(char c);
- void Scrivi_Car_PS(char c, int rig, int pos);
- void Scrivi_Testo(String stringa, int rig);
- void Pulisci();

N.B:Ho voluto appositamente riportare le funzioni più interessanti, tralasciando quelle ad uso interno della libreria.

La prima funzione ***Inizializza_LCD(int pin_RS,int pin_EN,int pin_D4,int pin_D5,int pin_D6,int pin_D7)***, come si intuisce dallo stesso nome, è quella di inizializzazione ovvero quella che permette di "avviare" il controller del display specificando al momento della chiamata, come vedremo in seguito, i pin di RS di Enable ed i 4 bit di dato.

Subito dopo troviamo le funzioni ***Cursore_ON() e Cursore_OFF()*** che permettono rispettivamente l'accensione e lo spegnimento del cursore al disotto del carattere.

La funzione ***Scrivi_Car_PC(char c)***, permette l'inserimento di un singolo carattere nella posizione corrente del display, ovvero alla prima posizione successiva a quella in cui si è fermata la data ram del display.

La funzione ***Scrivi_Car_PS(char c, int rig, int pos)*** , permette di scrivere un singolo carattere

in una posizione specifica del display, passando alla stessa il numero di riga e la posizione "colonna" alla quale si vuole scrivere.
La funzione ***Scrivi_Testo(String stringa, int rig)***, permette invece di scrivere del testo racchiuso tra virgolette nella riga specificata.
Ed in fine la funzione ***void Pulisci()*** , la quale ha il compito di svuotare la memoria interna del display e quindi di pulire le due righe.
Da notare che tutte queste funzioni sono di tipo void, ovvero non restituiscono alcun valore al programma chiamante, quindi eseguono dei comandi specifici e ma non invieranno nessun risultato verso il programma principale.
Di queste funzioni cercherò di dare illustrazione più approfondita nel seguito, tentando di far comprendere in modo semplice qual è il meccanismo che permette di interfacciare il software della libreria all'hardware del display studiando per sommi capi il datasheet dello stesso.
La dichiarazione private :in questa zona del programma si inseriscono invece tutte le dichiarazioni di variabili che verranno usate nella libreria. Queste potranno essere visibili solo all'interno di quest'ultima e non utilizzabili all'interno del programma principale perchè appunto dichiarate come private.
Nel nostro caso, ne elenco solo alcune, si possono notare le variabili di dato ***D7,D6,D5,D4*** di tipo int, le variabili ***carattere*** e ***c*** di tipo char etc etc… queste come vedremo in seguito sono utilizzate tutte nel file .cpp.

FILE LCD_M0.CPP

Il file .cpp è quello a mio avviso più importante della libreria, ovvero quello che permette di dare vita alle funzioni essenziali che permettono il dialogo tra software ed hardware.
Nel nostro caso questo file è denominato LCD_m0.cpp.

Nella descrizione delle funzioni più importanti di questa libreria, impareremo anche come come è di fondamentale importanza la consultazione del datasheet del componente da interfacciare, in modo da rispettare il più possibile i parametri descritti dalla casa produttrice.
Aprendo tramite un editor di testo il file LCD_mo.cpp ci imbatteremo nella funzione fondamentale della libreria ovvero ***LCD_m0::Inizializza_LCD(int pin_RS, int pin_EN,int pin_D4,int pin_D5, int pin_D6, int pin_D7)***.
Da notare che il pin R/W non è utilizzato ed è posto a GND inquanto nel nostro caso provvederemo solo ed esclusivamente alla scrittura verso il display e mai alla lettura della sua linea dati, anche perchè essendo il display alimentato a 5Vdc, per leggere i dati sul suo Bus dovrei utilizzare un traslatore di livello, inquanto questi avrebbero valore alto di tensione pari a 5Volt non compatibile con quelli della board M0 Pro.
Siccome la board M0 Pro lavora con segnali a livello altro di 3.3 Volt, leggendo dei dati provenienti dal display senza un traslatore di livello finirei per danneggiare microcontrollore ATSAMD21G18 a bordo.

Dovendo invece solo scrivere dei dati verso il display a 3.3 Volt, questi saranno correttamente riconosciuti dal dispositivo.
Tornando alla funzione vi invito a notare come prima di quest'ultima si inserisce la scritta "*LCD_m0::*", che indica l'appartenenza della funzione Inizializza_LCD alla libreria LCD_m0.
Ogniqualvolta quindi si crea un file .cpp è da tenere presente questa caratteristica, ovvero tutte le funzioni scritte nel file dovranno essere prece-

dute dalla sintassi:

Nome*DellaLibreria::NomeDellaFunzione*

Vediamo dunque che questa prende in ingresso 6 variabili intere che rappresentano i PIN che assegneremo rispettivamente al RS all'Enable ed ai bit di dato che vanno da D4 a D7.

La libreria è stata resa più generica possibile e la scelta dei pin del micro dedicati alla comunicazione con il display è praticamente arbitraria compatibilmente con l'hardware in uso.

Nel momento in cui si richiama la funzione di inizializzazione, questa esegue i seguenti passaggi:

1. Causa un ritardo di 250 ms (più che sufficiente per la stabilizzazione delle tensioni di alimentazione)
2. Definisce tutti i pin scelti della scheda come Output con la funzione *pinMode(....., OUTPUT);*
3. Come da datasheet invia i comandi di inizializzazione mediante la funzione SetCommandPin(.....)
4. Prepara il display a lavorare con interfaccia a 4 bit font 5x8 e scrittura da destra a sinistra **(leggere approfondimenti punto 3 per comprendere meglio l'argomento)**.

Approfondimenti del punto 3 :

Come suggerito dai datasheet dei costruttori di display con controller HD44780, per inviare dei dati a quest'ultimi bisogna rispettare una sequenza ben precisa di segnali come visibile nella figura seguente (Fig.3):

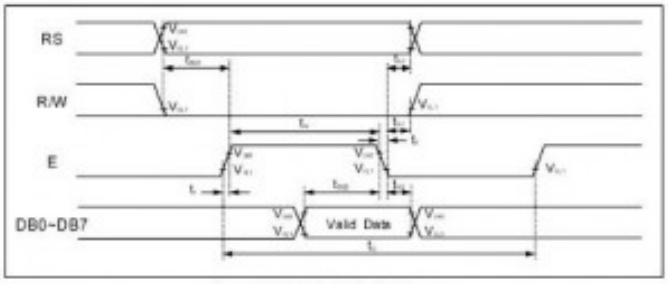


Fig.3 image credits: datasheet

tralasciando il segnale R/W, si nota come innanzitutto il segnale RS deve essere posto basso per specificare che si stanno per inviare dei comandi.

Il controller infatti a seconda del valore del pin di RS, discrimina tra comandi e dati, cioè capisce se voglio impartire un comando come ad esempio pulire lo schermo quando RS=0, oppure leggere un dato dalla sua DDRam con RS=1.

Queste due funzioni devono essere sempre validate da un impulso di Enable che dovrà subire una transizione L-H- L con durata minima di 230 ns , quando sia il bit di RS che i bit di dato assumono un valore stabile sul bus.

Per inviare un comando verso il controller ho utilizzato la funzione ***SetCommandPin()***, la quale manda i valori booleani dei bit da D7, D6, D5 e D4 mediante la funzione ***digitalWrite(_pin_D7, D7)*** seguendo il protocollo di Fig.3.

Di seguito il codice della funzione SetCommandPin:

```
//************************************************************
//*   Funzione di invio Comandi
//************************************************************
void LCD_m0::SetCommandPin(int D7,int D6,int D5, int D4)
{
  digitalWrite(_pin_RS, L);

  digitalWrite(_pin_D7, D7);
  digitalWrite(_pin_D6, D6);
  digitalWrite(_pin_D5, D5);
  digitalWrite(_pin_D4, D4);
  digitalWrite(_pin_EN, H);
  delay(1);
  digitalWrite(_pin_EN, L);
```

Infatti come prima operazione la funzione ab-

bassa il pin RS con il comando *digitalWrite(_pin_RS, L)*, dopo questo invia i comandi sul bus da D7 a D4 e poi per validare il tutto genera un impulso di Enable della durata di 1 msec che è ben più ampio di quello richiesto dal datasheet, quindi assolutamente idoneo.

Le funzioni SetCommandPin e SetDataPin nonostante molto semplici sono di fondamentale importanza per il corretto funzionamento della libreria, infatti vengono invocate più volte e provvedono all'invio ed alla corretta validazione dei comandi sul Bus.

Detto questo e tenendo presente il funzionamento di SetCommandPin, andiamo avanti nella descrizione della funzione di inizializzazione dell'LCD, procedendo con il punto 4 ed analizzando quali sono i dati da inviare in sequenza per fare in modo da poter utilizzare il display con configurazione *4 bit* e font *5x8*.

Sempre guardando i datasheet troviamo che per inizializzare il display con le caratteristiche scritte poc'anzi, i passi sono i seguenti : (Fig.4)

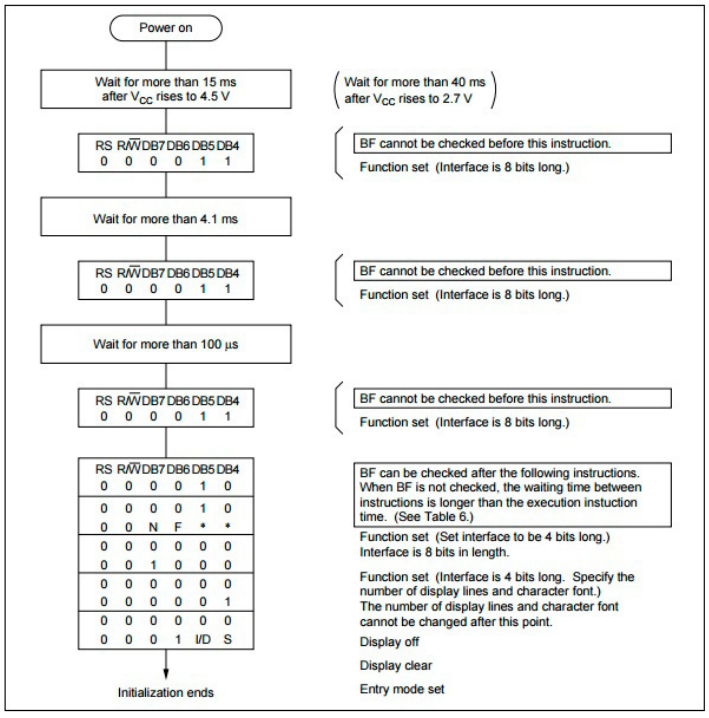


Fig.4 image credits: sparkfun.com

da come si evince dal diagramma di Fig.4, per prima cosa viene richiesto di attendere circa 40ms per la stabilizzazione della tensione di alimentazione (la nostra libreria attende 250 ms) e poi ad intervalli ben precisi di inviare il dato (0,0,1,1) sui pin D7...D4, cosa che noi facciamo con la solita funzione **SetCommandPin(0,0,1,1)** per 3 volte consecutivamente, inserendo dei ritardi abbastanza elevati per non incorrere in problemi di temporizzazione (Fig.5).

```
//ritardo per stabilizzazione Vdc
delay(250);
pinMode(pin_RS, OUTPUT);
pinMode(pin_EN, OUTPUT);
pinMode(pin_D7, OUTPUT);
pinMode(pin_D6, OUTPUT);
pinMode(pin_D5, OUTPUT);
pinMode(pin_D4, OUTPUT);

delay(50);
SetCommandPin(0,0,1,1);
delay(10);
SetCommandPin(0,0,1,1);
delay(10);
SetCommandPin(0,0,1,1);
 delay(50);
//impostazione interfaccia 4 bit
  SetCommandPin(0,0,1,0);
  delay(5);

//impostazione font e righe (2 righe Font 5x8)
```

Fig.5 Inizializzazione

Inviate queste 3 sequenze di dati il passo successivo sarà quello di specificare al controller che abbiamo intenzione di lavorare con interfaccia a **4 bit** anzichè ad **8 bit** e lo facciamo con l'invio del comando:

SetCommandPin(0,0,1,0);

Da questo momento in poi ogni comando da inviare verso il display dovrà essere suddiviso in due pacchetti da 4 bit ciascuno.

Di seguito avremo l'impostazione del font che sarà la standard 5x8 su due righe :

SetCommandPin(0,0,1,0);

SetCommandPin(1,0,0,0);

e successivamente le istruzioni relative a: spegnimento, riaccensione del display, pulizia dello schermo e shift a sinistra dei caratteri (Fig.6):

```
// impostazione Off display

   SetCommandPin(0,0,0,0);
   SetCommandPin(1,0,0,0);
   delay(5);


// impostazione On display cursore OFF

   SetCommandPin(0,0,0,0);
   SetCommandPin(1,1,0,0);
   delay(5);

// pulizia display

   SetCommandPin(0,0,0,0);
   SetCommandPin(0,0,0,0);
   delay(5);

//shift a sinistra cursore (Entry mode da SX a DX)

   SetCommandPin(0,0,0,0);
   SetCommandPin(0,1,1,0);
```

Fig.6 Inizializzazione

per analizzare nello specifico le varie sequenze di 8 bit (4 + 4) inviate al dispositivo, fare riferimento alla tabella seguente (Tab.1.).

HD44780U

		RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Clear display	Code	0	0	0	0	0	0	0	0	0	1	
Return home	Code	0	0	0	0	0	0	0	0	1	*	Note: * Don't care.
Entry mode set	Code	0	0	0	0	0	0	0	1	I/D	S	
Display on/off control	Code	0	0	0	0	0	0	1	D	C	B	
Cursor or display shift	Code	0	0	0	0	0	1	S/C	R/L	*	*	Note: * Don't care.
Function set	Code	0	0	0	0	1	DL	N	F	*	*	
Set CGRAM address	Code	0	0	0	1	A	A	A	A	A	A	
									Higher order bit		Lower order bit	

Dove si indicano con:

- I/D Incremento o Decremento DDRAM ==> I/D =0 ho un decremento della DDRAM e le scritte vengono eseguite alla rovescia, se I/D=1 le scritte vengono visualizzate normalmente
- S Shift caratteri ==> se S=0 Allora scrivo da Destra a Sinistra mentre se S=1 ho il contrario
- S/C Spostamento del cursore
- R/L Spostamento dell'intero contenuto del display senza variare il contenuto della DDRAM
- DL Indica in che modo verranno inviati i dati se in 2 pacchetti da 4 o in uno da 8 bit "nel nostro caso DL=0 , interfaccia a 4 + 4 bit"
- N Numero di righe N = 0 una sola riga N = 1 più di una riga
- F Font Display F = 0 5x8 mentre F = 1 5x10

Fino ad ora abbiamo analizzato l'inizializzazione del display, senza la quale quest'ultimo non sarà in grado di visualizzare alcun carattere, studiando la funzione SetCommandPin che provvede ad inviare i dati a pacchetti da 4 + 4 bit.

Ogni comando inviato verso il controller è costituito dunque dai primi 4 bit più significativi, seguito dai 4 bit meno significativi.

Successivamente vedremo come far scrivere un carattere al display, adoperando due funzioni tra loro uguali e simili alla SetCommandPin, che questa volta ho chiamato SetDataPin1 e SetDataPin2.

I più attenti a questo punto noteranno che, mentre nell'invio di un comando adoperavo per due volte la stessa funzione, adesso invece, per la richiesta di un dato uso due funzioni diverse.

Tutto ciò è dovuto al fatto che oltre alla transizione L-H-L del pin di Enable deve verificarsi una transizione anche sul pin di RS ogni 4 bit inviati.

L'INVIO DEI DATI E RICHIESTA DI SCRITTURA CARATTERE

Analizziamo quindi un'altra funzione importante della libreria che permette di inviare un testo e visualizzarlo sul display, che è ***LCD_m0::Scrivi_Testo(String stringa,int rig)*** , la quale prende in ingresso come parametri una stringa con il testo da scrivere immesso tra virgolette " ", ed il numero della riga sulla quale scriverlo.
La funzione non fa altro che calcolare la lunghezza della stringa e memorizzare questa in una variabile numerica n, dopodichè a seconda che abbia selezionato la prima o la seconda riga, dico al display tramite la solita SetCommadPin che voglio iniziare a scrivere sulla prima o sulla seconda riga, mediante la sequenza di comandi :

SetCommandPin(1,0,0,0) e SetCommandPin(0,0,0,0) "per la riga 1"

SetCommandPin(1,1,0,0) e SetCommandPin(0,0,0,0) "per la riga 2".

Impostato il display, la funzione provvede ad estrapolare bit dopo bite tutti i caratteri presenti nella stringa tramite il ciclo for e provvede a scriverli uno ad uno tramite l'altra importantissima funzione **Scrivi_Car_PC(c)** che analizzeremo tra un attimo.
Come si può vedere dal listato, la funzione prende in ingresso un carattere e mediante il ciclo for, shifta uno ad uno tutti i bit che compongono lo stesso.

Questi verranno moltiplicati mediante la funzione AND con il valore uno booleano e memorizzati uno per volta nel vettore bin[i]. Di conseguenza alla fine del ciclo for, il vettore bin[i] conterrà i vari bit che rappresentano il codice ASCII del carattere inviato.

L'unico problema è che però questo codice ASCII memorizzato in bin[i] è disposto al contrario, quindi mediante un altro ciclo for riallineo correttamente tutti i bit del carattere e li invio uno ad uno al controller del display mediante le funzioni:

SetDataPin1(bin_inv[0],bin_inv[1],bin_inv[2],bin_inv[3]) SetDataPin2(bin_inv[4],bin_inv[5],bin_inv[6],bin_inv[7])

a questo punto vedremo finalmente visualizzato il carattere o i caratteri prescelti.
A seguito dell'analisi fatte sulle principali funzioni della libreria, ritengo non necessaria la spiegazione delle altre funzioni, in quanto una volta comprese quelle analizzate fin'ora le altre risultano di facile comprensione.
Mi soffermerò solo per qualche altro rigo, spiegando per chi ancora non lo sapesse, come fare ad inserire nell'ide la libreria e l'esempio sviluppato.
Per inserire la libreria nell'IDE di Arduino, è conveniente scaricare il file Zip e memorizzarlo nella cartella Documenti =>Arduino=>Libraries.
Fatto ciò basta entrare nell'IDE e nella barra degli strumenti sotto la voce Sketch selezionare Aggiungi Libreria, andando ad aggiungere dunque il file zip che poco fa abbiamo inserito nel percorso precedente.
Detto questo non ci rimane che aprire il file nella cartella Example e testare il file di esempio.
Spero che il lavoro da me eseguito possa essere utile per due aspetti fondamentali, il primo è quello di aver provato a dare delle nozioni sulla creazione di una libreria Arduino partendo dal

software e scendendo fino allo studio dell'hardware del dispositivo da interfacciare, con l'auspicio di poter essere stato di aiuto a tutti coloro che volessero crearne una personalizzata.
Il secondo aspetto riguarda l'aver creato una libreria funzionante non ancora sviluppata per la board Arduino M0 Pro, che potrà essere utile a tutti coloro che come me hanno preferito partire da questo dispositivo per iniziare una nuova e affascinante avventura.

Segue il link per il download dei file della libreria LCD_m0.

Buon Lavoro!

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/progetto-di-una-libreria-per-lcd-16x2-compatibile-con-arduino-m0-pro

Termometro ed Igrometro con Arduino M0 Pro su display LCD 16x2

di Mario Mottula

Qualcuno di voi lettori potrà subito pensare che la board in questione è sicuramente sovradimensionata per il compito che avrà da svolgere in questo progetto, ed in effetti è così viste le potenzialità della M0Pro.
La sostanza dell'articolo però non vuole essere banalmente quella di insegnare a come leggere il valore da un sensore e visualizzarlo su un display, ma quella di far prendere dimestichezza con gli strumenti fondamentali della programmazione di Arduino M0 Pro a chi è alle prime armi con lo studio di un progetto utile e poco complesso.
Arduino M0Pro differisce un pò per alcuni aspetti dalle altre board Arduino, per le quali ad oggi si trova tantissimo materiale online, mentre vista la recente uscita di quest'ultima alcune informazioni sono difficili da reperire quindi in molti si allontanano dallo scoprire un dispositivo che ha davvero tantissime potenzialità.
Per la descrizione del progetto partiremo dall'introdurre i due sensori utilizzati, per poi dare un'occhiata alla conversione A/D ed alle parti salienti del firmware.

1. IL SENSORE DI TEMPERATURA LM35

Questo sensore oltre ad essere molto comune nelle applicazioni a microcontrollore è facilmente interfacciabile e molto preciso nel suo range di funzionamento ideale e reale.
E' costituito da soli 3 pin ed è anche disponibile nel package TO-92 (figura 1), ha un andamento lineare di 10mV/°C , un range di funzionamento che va da -55°C a + 150°C ed un'accuratezza di 0.5 °C a 25°C di temperatura di lavoro.
Nel nostro progetto sfrutteremo la sola dinamica positiva del dispositivo, ovvero il range 0 - 100 °C.

Inoltre come visibile dalla tabella precedente, il dispositivo è perfettamente compatibile con una tensione di alimentazione di 3.3 Volt, che nel nostro caso verrà prelevata direttamente dalla M0Pro.
L'interfaccia tra sensore e MoPro è semplice, si potrebbe pensare di collegare direttamente il pin Vo all'ingresso analogico A0 della board. Io ho scelto la configurazione con filtro RC di modo che, qualora si voglia installare il sensore distante dalla scheda mediante un cavo di prolungamento, non si avranno problemi di accoppiamento di impedenza ed inoltre in presenza di segnali ad alta frequenza vicini al cavo, che potrebbe fungere da antenna, questi verrebbero cortocircuitati a massa dal filtro evitando così interferenze sul segnale utile che come già detto è di qualche decina/centinaia di millivolt (Figura 2.).

		MIN	MAX	UNIT
Supply voltage		–0.2	35	V
Output voltage		–1	6	V
Output current			10	mA
Maximum Junction Temperature, $T_J$max			150	°C
Storage Temperature, T_{stg}	TO-CAN, TO-92 Package	–60	150	°C
	TO-220, SOIC Package	–65	150	

Figura 1: Piedinatura e caratteristiche del sensore LM35

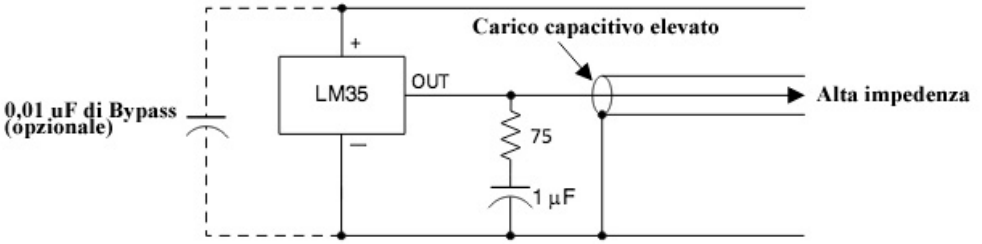


Figura 2: Configurazione filtro RC

Nella conversione A/D dei valori di tensione generati da questo sensore, verrà utilizzata una tensione di riferimento interna all'ATSAMD21G18 pari ad 1 Volt.

Mediante questa scelta ed in base alle caratteristiche del LM35, avremo modo di sfruttare a pieno la dinamica del sensore sui valori positivi delle temperature.

2. IL SENSORE DI UMIDITÀ YL-38 +YL-69

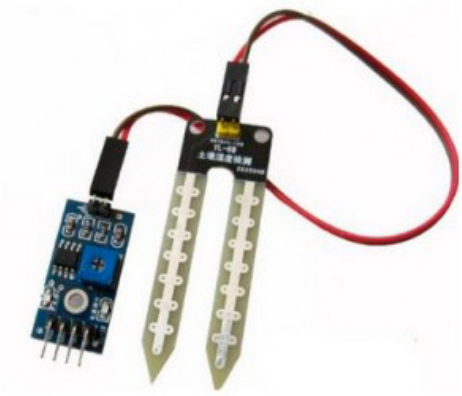


Figura 3: sensore di Umidità YL-38 +YL-69

Questo sensore (Fig.3) è della categoria low cost, ha un principio di funzionamento molto semplice ed è costituito da due uscite, una analogica Ac ed una digitale Out.

Il dispositivo è composto sostanzialmente da una resistenza variabile che cambia il suo valore in base a quanta umidità entra in contatto con la sua superficie della zona conduttiva del YL-69 e da un amplificatore operazionale in configurazione comparatore (Yl-38), che permette di attivare o disattivare opportunamente l'uscita digitale in base al valore di tensione sul pin non invertente INA^+.

Il valore della resistenza generato dal YL-69, provoca appunto una caduta di tensione sul pin invertente dell'opamp LM393 che viene trasferito direttamente in uscita.

Nello stesso tempo questo valore di tensione dato dalla caduta sull'elemento sensibile del YL69, viene comfrontato con quello presente sul pin non invertente INA^- impostato precedentemente dal trimmer R2 da 10K.

Quando la tensione sul pin INA^- sarà maggiore di quella presente su INA^+ l'uscita digitale avrà una commutazione a livello logico altro causando l'accensione del led D2.

Ricapitolando dunque, il sensore è composto

da due uscite Ac, uscita analogica direttamente interconnessa al pin invertente dell'opamp e da Out uscita digitale, comandata dal comparatore formato dalla resistenza variabile del sensore YL-69 e da quella di riferimento settabile mediante il trimmer R2 (Fig.4).
La visualizzazione dello stato dell'uscita digitale è possibile mediante il LED D2, mentre il led D1 segnala la presenza dell'alimentazione.

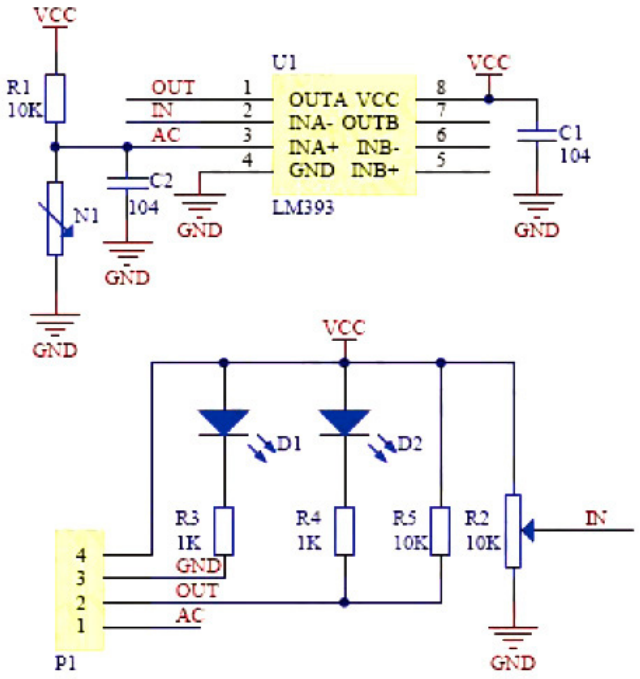


Figura 4: Layout elettrico

La cosa interessante, è quella di capire quali sono i valori di tensione generati in uscita dal sensore YL-69 a fronte di un determinato valore di umidità.
Purtroppo il datasheet presente sul sito della casa produttrice non fornisce questo tipo di indicazione, che ho provato comunque a ricostruire mediante delle prove empiriche.
Ho ipotizzato, vista la linearità dell'elemento sensibile, un andamento per l'appunto lineare nel suo intervallo di funzionamento, di conseguenza ho stabilito due punti fermi di misurazione ovvero:

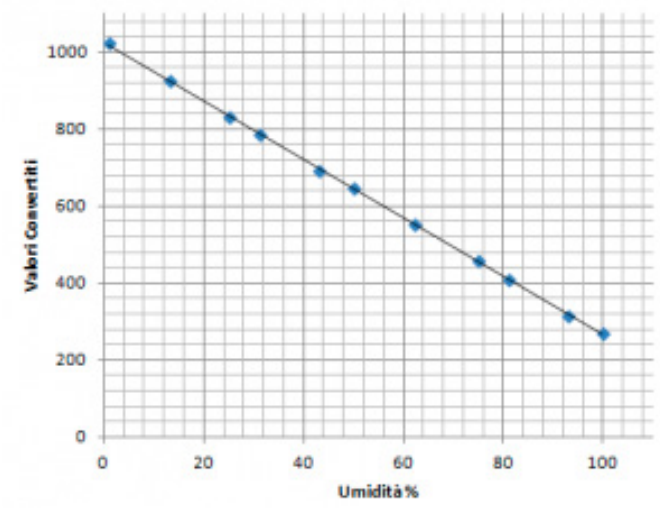


Figura 5: Andamento lineare del sensore di umidità

100 % di umidità ==> sensore completamente bagnato e valore di conversione nell'intorno di 270

1% di umidità==> sensore completamente asciutto e valore di conversione pari a 1023

Da questa semplice ipotesi confermata dall'immersione in un terreno completamente bagnato del sensore ed in un terreno arido, ho estrapolato i dati intermedi ipotizzando un andamento lineare del sensore nell'intervallo 1% - 100% (Fig.5).
Da tenere presente che questi valori sono stati ottenuti inserendo completamente il sensore nel suolo fino alla sua ultima tacca.
I dati subiranno delle variazioni se questa regola non verrà rispettata.

3. LA CONVERSIONE A/D

Per gestire i due ingressi analogici provenienti dai due sensori LM 35 e YI-38+69, ho usato i due canali digitali A0 ed A1 rispettivamente.
Purtroppo per loro natura fisica, questi sensori producono tensioni in uscita con ''scale'' completamente differenti, ovvero l'LM35 ha un ran-

ge di funzionamento che va da:

0 mV ==> 0°C fino a 1V==>100°C

mentre il sensore YL-38+69 ha un range di funzionamento che va da:

1.2Volt ==> circa 90/100 % Umidità

3.2 Volt ==> Asciutto

Si deduce quindi che essendo grandezze diverse avrò la necessità di usare scale di conversione differenti e quindi valori di riferimento differenti.

Tutto questo è stato impostato nel firmware in modo **dinamico**, ovvero ho previsto la variazione della tensione di riferimento in runtime come vedremo più avanti.

Adesso è necessario un breve accenno alle caratteristiche funzionale del convertitore A/D presente nella board M0Pro ed in particolare nall'ATSAMD21G18.

-Il convertitore Analogico Digitale interno:

Il microcontrollore ATSAMD21G18 ha a bordo un convertitore A/D a 12bit (4096 valori di conversione da 0 a Vref), che può essere impostato anche a 8 o 10 bit tramite la funzione analogReadResolution(res), dove res indica in intero che può assumere i valori 8,10 o 12 a seconda della risoluzione scelta.

Ha 6 ingressi analogici A0-A5, ed ognuno di essi accetta tensioni tensioni che vanno da 0 a 3.3Volt, è possibile sfruttare le tensioni di riferimento interne al chip mediante l'uso della funzione analogReference(), oppure impostare quest'ultima tramite il pin esterno AREF.

Mediante la funzione **analogReference(ref)** avremo accesso ai seguenti valori di riferimento:

- AR_DEFAULT
- AR_INTERNAL
- AR_EXTERNAL

quindi con analogReference (**AR_DEFAULT**) imposteremo come Vref un valore pari a VDDana ovvero uguale alla tensione di alimentazione della circuiteria analogica che coincide con la VDD =3.3Volt.

Con analogReference (**AR_INTERNAL**) imposteremo Vref = 1 Volt e con analogReference (**AR_EXTERNAL**) imposteremo noi la tensione di riferimento sul pin AREF con un valore di tensione che va da **0 a VDD-0.6 =2.7 Volt massimi**.

Nel nostro progetto è stata impostata una tensione di riferimento interna, per la conversione dei valori provenienti dal LM35, mentre una tensione di riferimento di default per il sensore di umidità, tutto ciò in maniera dinamica come vedremo nella descrizione del firmware.

4. IL FIRMWARE

Per una comprensione più chiara di questo paragrafo, si consiglia la visualizzazione del firmware allegato all'articolo.

Il firmware del nostro progetto è strutturato in 4 parti, la prima parte riguarda la dichiarazione ed inizializzazione delle variabili, la seconda parte riguarda la rilevazione e visualizzazione della temperatura, la terza parte la rilevazione e visualizzazione dell'umidità e la quarta parte l'inserimento tramite routine di interrupt di un valore di soglia per l'attivazione/disattivazione delle due uscite.

Vediamo di analizzarle un pò più nel dettaglio.

Dichiarazione variabili e costanti principali:

In questo paragrafo discuterò brevemente dell'u-

so delle variabili e delle costanti del firmware più importanti, iniziando dalla dichiarazione delle due costanti conferma ed incremento impostate rispettivamente a valore 7 e 6 che saranno utilizzate per identificare i pin sui quali sono interconnessi pulsanti di avvio routine di **interrupt ed incremento valore di soglia e di conferma del valore inserito.**

In concomitanza a queste due costanti vengono definite le due variabili che contengono lo stato della pressione dei due pulsanti PB1 e PB2, che sono rispettivamente **buttonState6 e buttonState7**.

Quando questi pulsanti saranno premuti le variabili conterranno il valore logico 1 mentre assumeranno il valore logico 0 se rilasciati.

Le variabili **ValTemp,ValHu** contengono il valore intero proveniente dalle rispettive conversioni, nel caso della temperatura, il dato prima di essere memorizzato in questa variabile viene mediato su 10 valori letti e poi memorizzato.

Nel caso dell'umidità invece il valore letto a seguito della conversione, viene prima inserito nella variabile d'appoggio **ValoreConvertito** e poi memorizzato in **ValHu** a seguito dei vari if che ne determinano il corretto valore.

La variabile soglia_EXT, conterrà il valore di soglia impostato da interrupt, le stringhe TVis e HuVis conterranno i valori di temperatura ed umidità da inviare sotto forma di stringa al display.

Di fondamentale importanza sono la variabile maschera e la variabile flag, quest'ultima permette in base al suo valore di far si che si entri nel menu di inserimento della soglia o che si rimanga all'interno del programma principale.

La variabile maschera invece quando il flag è a valore 1 permette di visualizzare un sottomenu per l'impostazione della soglia.

Setup:

Nel setup vengono inizializzate le variabili al loro valore di partenza e definiti i pin del micro con input ed output, ma la cosa importante è che viene impostata la risoluzione del convertitore Analogico Digitale con risoluzione a 10 bit ed attivato l'interrupt sul pin 6 sul fronte di salita rispettivamente mediante le funzioni:

```
analogReadResolution(10);
attachInterrupt(6,attiva_flag, RISING);
```

ed in fine viene inizializzato il display con la funzione :

```
LCD.Inizializza_LCD(12,11,5,4,3,2);
```

della quale abbiamo discusso ampiamente nell'articolo da me scritto precedentemente: **Progetto di una libreria per LCD 16×2 compatibile con Arduino M0 PRO**

Il LOOP del programma:

Il main program o loop, in questo caso contiene due sottoprogrammi a loro volta esplicitati all'interno di due cicli di while.

Nel ciclo di while in cui si deve necessariamente verificare la condizione flag = 0 avremo tutte le funzioni necessarie alla lettura e visualizzazione di temperatura ed umidità ed in più la gestione delle due uscite alle quali sono collegati i led D1 e D2 (pin 10 e pin 13), in base all'impostazione della soglia desiderata.

Nell'altro while in cui si dovrà verificare **flag = 1**, avremo invece la gestione del menu di inserimento della soglia e delle variabili che tengono traccia del valore di quest'ultima.

Verranno quindi gestite le pressioni dei pulsanti

di incremento e conferma del valore scelto da assegnare alla soglia, entro il quale attivare / disattivare le uscite.

Il loop principale parte quindi con il **flag=0**, ed all'accensione verrà visualizzato il valore di temperatura ed umidità secondo i seguenti criteri:

viene impostato il valore di riferimento del convertitore Analogico Digitale come INTERNAL, poi viene confrontato il valore attuale della temperatura con il valore di soglia impostato di default e di conseguenza verrà attivata o disattivata l'uscita corrispondente.

Sulla riga uno del display verrà visualizzata la scritta "*Temp.* " e di seguito il valore della temperatura che verrà calcolato mediante la lettura consecutiva di 10 valori di temperatura sui quali verrà fatta una media (Fig.6).

```
analogReference(AR_INTERNAL);
if(soglia_EXT >= ValTemp/10000){
                          digitalWrite(10, HIGH);   //Accensione Elemento riscaldante
                          digitalWrite(13, LOW);    //Spegnimento Refrigeratore
                          }

if(soglia_EXT < ValTemp/10000){
                          digitalWrite(10, LOW);   //Spegnimento Elemento riscaldante
                          digitalWrite(13, HIGH);  //Accensione Refrigeratore
                          }
somma=0;
LCD.Scrivi_Testo("Temp.",1);

for(i=0;i<10;i++)
                {
                int ValoreConvertito = analogRead(A0);
                delay(5);
                Valori[i]= (ValoreConvertito*1023)/1.03;

                }
//Somma dei 10 valori letti per esecuzione della media
for(i=0; i<9; i++)
                   {
                    somma = somma + Valori[i];

                   }
//Media dei valori
ValTemp= somma/10;
```

Figura 6: Codice sketch

Come si può facilmente vedere dalla figura precedente nel ciclo *for (i=0 ; i<10;i++)*, impongo al convertitore la lettura di 10 valori di temperatura, ogni valore verrà memorizzato nella posizione i-esima dell'array Valori[i].

Al termine della memorizzazione i valori contenuti all'interno dell'array verranno sommati e mediati ed il valore ottenuto memorizzato nella variabile ValTemp.

Successivamente affinchè il valore ottenuto possa essere visualizzato su display, dovrà essere convertito in stringa con la funzione:

TVis=String(ValTemp);

verrà in seguito eseguito un arrotondamento ai 0.5°C:

```
//Arrotondamento ai 0.5°C
if( TVis[0] >='0' && TVis[2]<='5' )
                                    {
                                    TVis[2]='0';
                                    }
if( TVis[2]>'5'&& TVis[2]<='9' )
                                    {

                                    TVis[2]='5';
                                    }
```

si tenga presente che i valori verranno visualizzati sul display al termine della routine di conversione dell'umidità tramite le funzioni :

```
LCD.Scrivi_Car_PS(TVis[0],1,6);
LCD.Scrivi_Car_PS(TVis[1],1,7);
LCD.Scrivi_Car_PS('.',1,8);
LCD.Scrivi_Car_PS(TVis[2],1,9);
LCD.Scrivi_Car_PS(223,1,10);
LCD.Scrivi_Car_PS('C',1,11);
```

Dopo la gestione del LM35 verrà letto il valore del sensore proveniente dal YL38-69 semplicemente cambiando il valore di riferimento del convertitore da INTERNAL a DEFAULT con la funzione *analogReference(AR_DEFAULT)*.
Il valore letto verrà memorizzato nella variabile ValoreConvertito e verrà confrontato con alcuni range preimpostati mediante gli if. A seconda della condizione verificata sarà assegnato un valore alla variabile ValHu il quale dopo la conversione a stringa verrà visualizzato su display con le funzioni:

```
LCD.Scrivi_Car_PS(HuVis[0],2,10);
LCD.Scrivi_Car_PS(HuVis[1],2,11);
LCD.Scrivi_Car_PS('%',2,12);
```

Nella seconda riga del display verrà quindi visualizzata l'umidità nel seguente formato "Umidita: XX%".
Quando verrà fatto scattare un interrupt, con la pressione di PB1, questo porrà il flag=1 ed azzererà la variabile menu.
In questo modo verrà eseguito il sottoprogramma con la condizione while (flag ==1) verificata, che permetterà l'uscita dal programma principale e l'impostazione del valore di soglia.
Infatti alla pressione del pulsante PB1, il display visualizzerà la scritta "Inserisci soglia [ ]", premendo ancora PB1 vedremo incrementare il valore della soglia che partirà da quello preimpostato che è pari a 10.
Quando arriveremo al valore di soglia desiderato per confermarlo basterà tenere premuto PB2 per circa 2 secondi, in questo modo il valore verrà memorizzato e le uscite di refrigeratore ed elemento riscaldante verranno abilitate o disabilitate in funzione del valore impostato.
Alla pressione di PB2 verrà riattivato l'interrupt e verranno ripristinati i valori delle variabili maschera e flag.

5. ROUTINE DI INTERRUPT

Questa routine serve semplicemente a far si che in qualunque momento si possa intervenire sul valore di soglia prescelto.
La funzione di servizio dell'interrupt viene invocata nel momento in cui si preme il tasto PB1 collegato al pin 6, in questo istante il programma principale termina le sue normali operazioni per servire la funzione attiva_flag():

```
void attiva_flag(){
menu =0;
flag=1;
detachInterrupt(6);//Disabilito l'interrupt per evitare che durante la pressione del pulsante, i rimbalzi meccanici generino altri interrupt indesiderati

}
```

All'attivazione dell'interrupt quindi viene posta a zero la variabile menu e messo ad uno il flag, in questo modo verrà abilitato il ciclo while nel programma principale che gestisce la scelta della soglia.
Come si può notare dalle righe di codice della funzione attiva_flag(), ad un certo punto l'interrupt viene disabilitato per essere riabilitato successivamente al termine della pressione del pulsante PB2.
Ho fatto questo per evitare che durante la pressione del pulsante PB1 si potessero scatenare altri interrupt indesiderati dovuti ai rimbalzi meccanici del pulsante.
Il flag verrà riportato a zero dopo la pressione

del pulsante PB2.

6. SCHEMA ELETTRICO

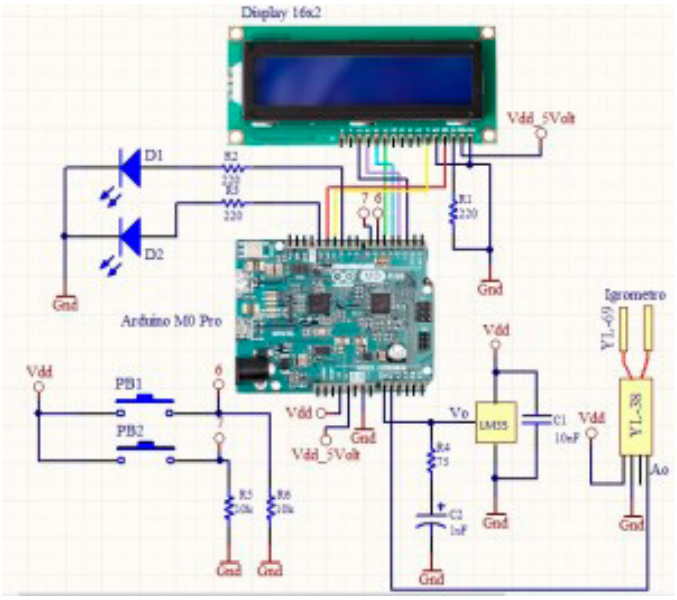


Figura 7: Schema elettrico

Elenco Componenti:

- Arduino Mo Pro
- PB1, PB2 = Pulsanti normalmente aperti
- R1,R2,R3 = Resistenze 220 Ohm 1/4Watt
- R4 = Resistenza 75 ohm 1/4Watt
- R5,R6 = Resistenze 10Kohm 1/4 Watt
- D1,D2 = Diodi Led 5mm
- C1 = Condensatore 1onF
- C2 = Condensatore 1uF Elettrolitico
- LM35 = Sensore di Temperatura
- YL38/69 = Igrometro

7. CONCLUSIONI

Per far si che sulle uscite alle quali sono sollegati D1 e D2, si possa collegare un carico qualunque bisogna predisporre un comunissimo circuito di controllo di un Relè o Triac a seconda della scelta più consona alle esigenze.

Rimango comunque a disposizione per qualunque chiarimento in merito all'articolo.

ALLEGATI

TermoIgrometroMoPro

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/termometro-ed-igrometro-con-arduino-m0-pro-su-display-lcd-16x2

EMG superficiale per sport performance con Arduino MO Pro

di Domenico Canepa

INTRODUZIONE

L'**Elettromiografia Superficiale (SEMG)** è una tecnica che misura l'attività elettrica durante la contrazione dei muscoli. Opera in maniera simile all'elettrocardiogramma che invece misura l'attività del cuore. L'esame è indolore. E' totalmente non invasivo e non causa irritazioni. È assolutamente sicuro per bambini e neonati, donne incinte e persone anziane.

L'analisi elettromiografica rappresenta la manifestazione elettrica della contrazione muscolare. Se sulla superficie della cute vengono applicati due elettrodi bipolari è possibile osservare durante la contrazione muscolare una variazione della distribuzione di potenziale. Registrando questa variazione si ottiene un segnale indicativo dell'attività muscolare in funzione del movimento effettuato. L'utilizzo di elettrodi di superficie semplifica sensibilmente le operazioni di prelievo del segnale e, unitamente alla non-invasività della tecnica, rende possibile l'effettuazione di registrazioni sia in condizioni di contrazione isometrica sia in sforzo dinamico, ad esempio durante l'esecuzione di esercizi o gesti funzionali (1).

CONCETTI BASE

Il muscolo è un organo costituito da tessuto muscolare con proprietà contrattili. Ci sono diversi tipi di muscoli, distinti in base alle proprietà del tipo di tessuto muscolare che lo costituisce, quello che prenderemo in considerazione appartiene alla categoria dei muscoli striati scheletrici. Questo tipo di muscoli è costituito da unità di fibre muscolari di forma allungata, detti sarcomeri, che si contraggono in seguito agli impulsi nervosi inviati dai motoneuroni.

La contrazione muscolare si può dividere in tre fasi principali:

Contrazione;

Rilassamento;

Fase latente.

La contrazione avviene quando uno stimolo elettrico, proveniente dai motoneuroni, arriva ai bottoni sinaptici, i quali liberano il neurotrasmettitore acetilcolina che apre i canali del sodio e depolarizza la membrana plasmatica delle fibre muscolari. Questa prima **depolarizzazione** dà origine ad un potenziale d'azione di 70mV che, propagandosi lungo la fibra, permette la diffusione di ioni calcio e quindi l'unione di actina e miosina con una precisione di 45°.

Nella fase di rilassamento il procedimento è sostanzialmente eseguito al contrario. Ciò che varia sono i neurotrasmettitori coinvolti, specifici nell'inibire il rilascio di **acetilcolina**.

La fase latente è la fase successiva allo stimolo in cui il muscolo deve riadattarsi al potenziale di origine. In questa fase, le fibre muscolari non sono in grado di ricevere un nuovo stimolo se non un nuovo impulso che non sia superiore a

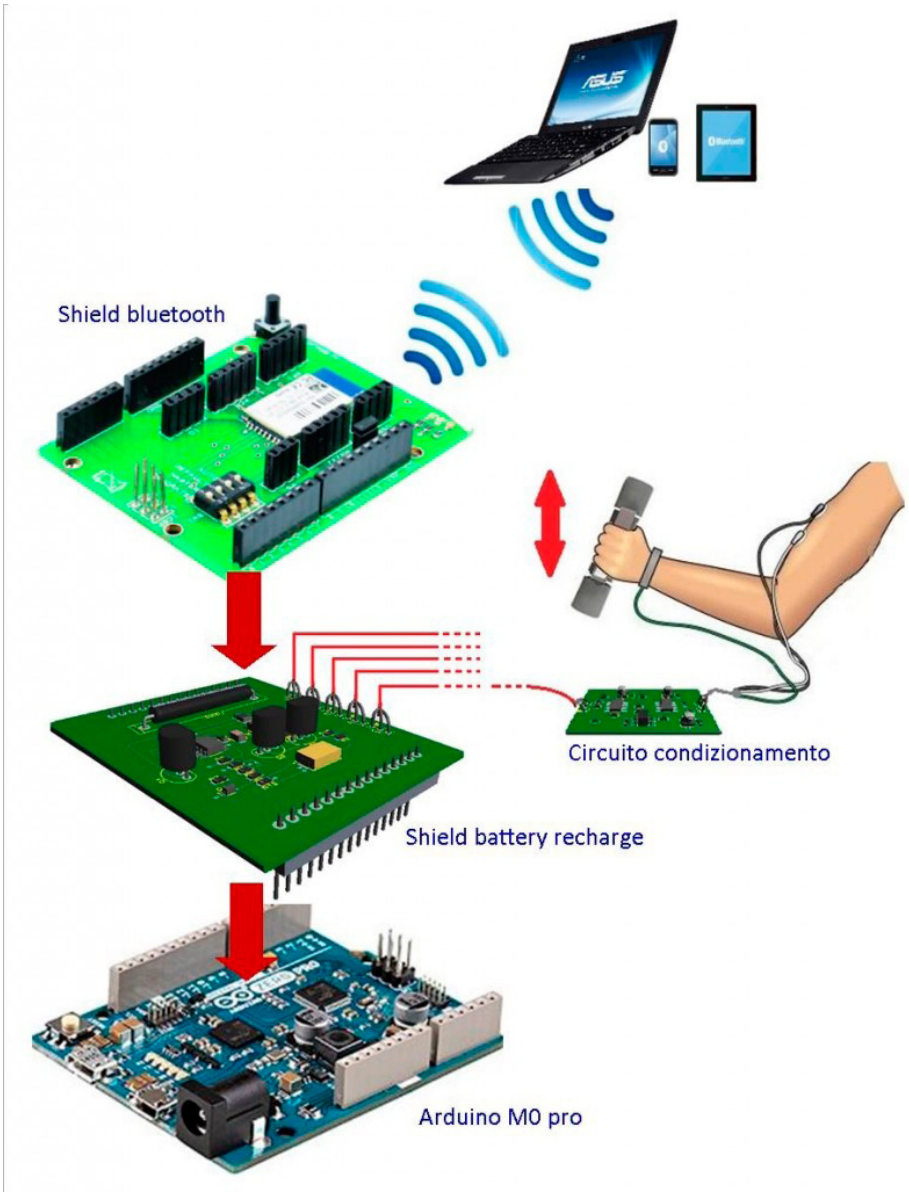


Figura 1: composizione del sistema

quello ricevuto precedentemente. In alcuni casi, il muscolo è sottoposto anche ad un **periodo refrattario** durante il quale le fibre muscolari non sono in grado di rispondere a nessun tipo di stimolo, anche di forte entità (2).

La fase di contrazione può essere monitorata tramite tracciato **EMG** (**elettromiogramma**). L'elettromiografia misura i potenziali d'azione che si generano durante la contrazione volontaria di un muscolo. Questo esame viene effettuato normalmente grazie all'uso di elettrodi superficiali che amplificano e registrano l'impulso nervoso.

Essendo depolarizzazioni della membrana cellulare, un singolo potenziale d'azione corrisponde, nel caso dell'uso di elettrodi interni, all'attività di una singola unità motoria (un'unica fibra muscolare collegata ad una terminazione nervosa) oppure, nel caso di **elettrodi superficiali**, ad un gruppo di unità motorie, come nel nostro caso.

Dalla letteratura, emerge che il segnale mioelettrico per essere comprensibile, deve essere opportunamente elaborato, in particolare filtrato,

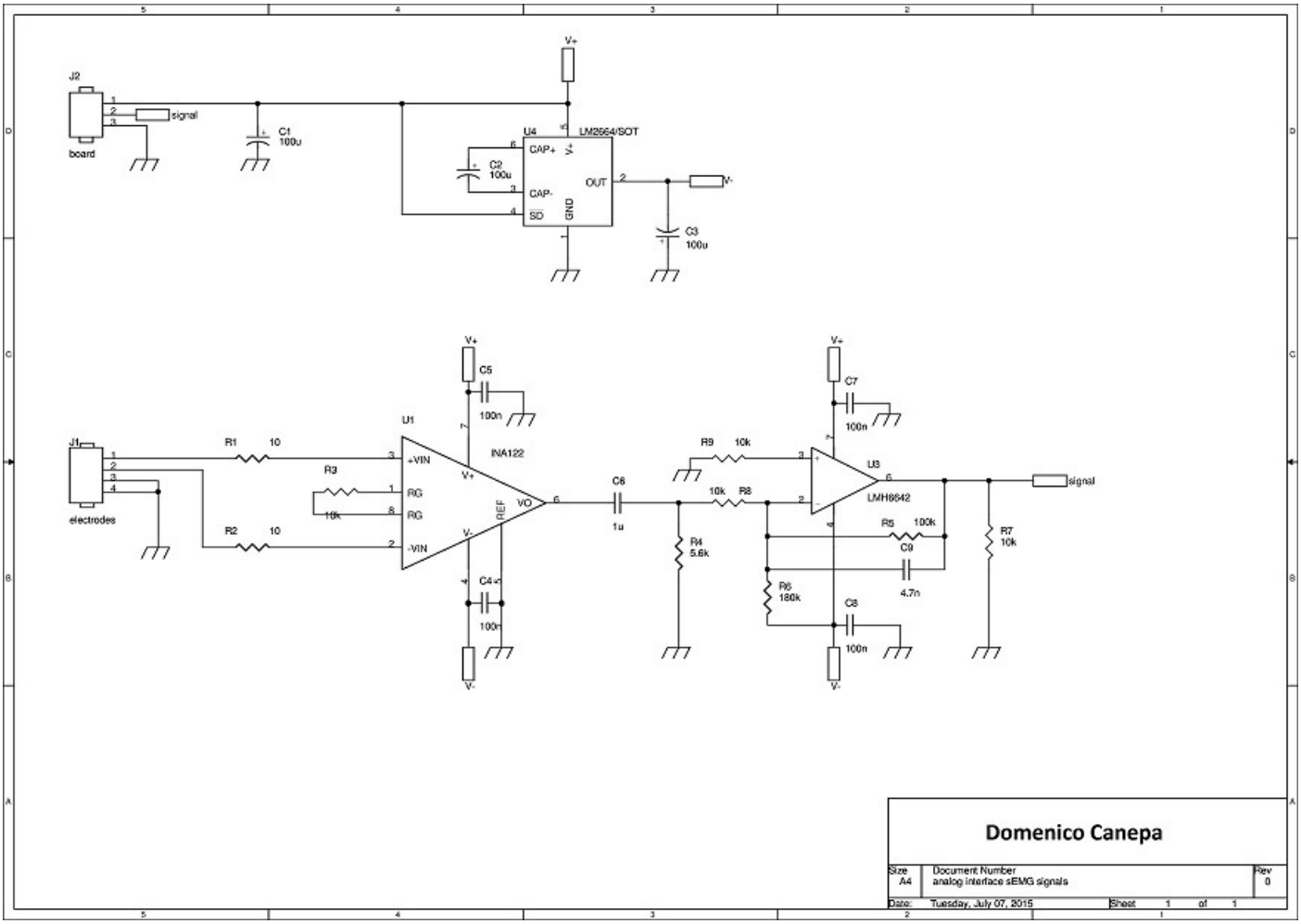


Figura 2: schema elettrico del circuito di condizionamento

raddrizzato e calcolato il valore RMS (valore quadratico medio) nel dominio del tempo; mentre nel dominio della frequenza si deve calcolare lo spettro in frequenza, la frequenza media e la frequenza mediana.

ARCHITETTURA DEL SISTEMA

Il sistema è composto di un' unità di controllo ed elaborazione costituita da Arduino M0 Pro, la quale campiona il segnale amplificato e filtrato da un circuito di condizionamento, effettua delle elaborazioni numeriche per poi inviare i dati attraverso un modulo Bluetooth RN42. È possibile osservare in figura 1 che si ha la possibilità di acquisire 5 coppie di elettrodi (canali), garantendo la massima efficienza, senza perdere prestazioni, grazie al potente microcontrollore ARM 32bit montato su Arduino.

Il circuito di condizionamento è formato da un amplificatore per strumentazione INA122 che amplifica il segnale di 30db, avente un CMRR di 90db; in cascata, un filtro passa banda con frequenze di taglio 20Hz-600Hz elimina la componente continua e frequenze superiori alla banda di interesse.

Il circuito di condizionamento è connesso alla board Arduino mediante uno shield che equipaggia a bordo anche un carica batterie a litio, permettendone l'utilizzo senza la rete elettrica. Lo schema dello shield mostrato in figura3, per-

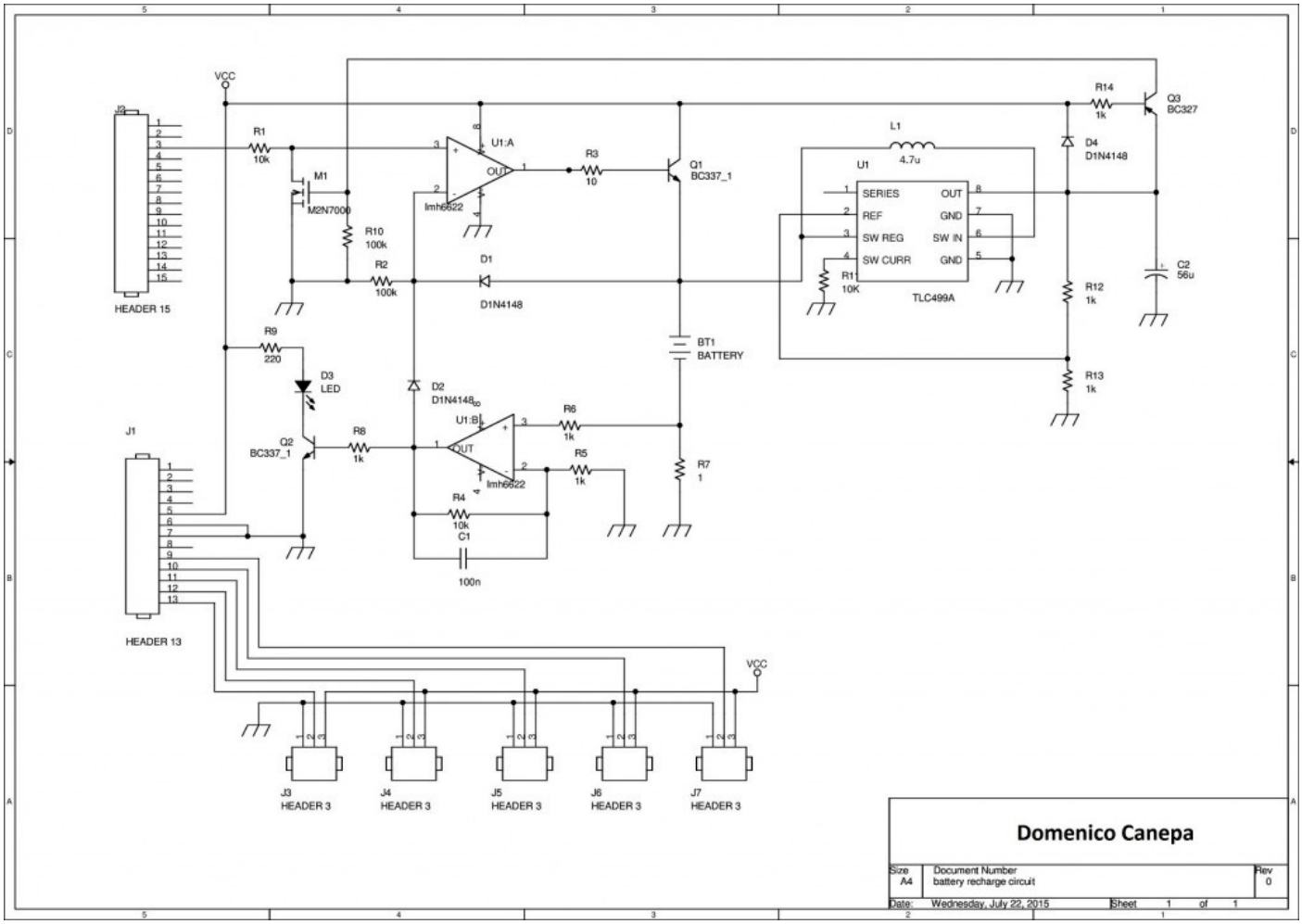


Figura 3: schema elettrico shield carica batteria

mette di alimentare Arduino M0 Pro senza dover connettere altri fili o quant'altro, consentendo di caricare una batteria al litio quando Arduino è connesso al pc tramite porta usb o è connesso al proprio alimentatore. La carica della batteria è affidata a due operazionali, dove il primo, in configurazione voltage follower mantiene la tensione costante, mentre l'altro amplificatore in configurazione di convertitore corrente tensione, fa si che scorra una corrente costante sulla batteria sino al raggiungimento di una tensione di 4.1V, per poi essere mantenuta costante mediante il voltage follower. Per alzare la tensione della batteria a 5V è stato utilizzato il convertitore DC-DC TLC499A

Durante il funzionamento, il sistema dovrà essere alimentato esclusivamente dalla batteria, poiché una connessione con la rete elettrica può provoca disturbi elettromagneti non trascurabili che altererebbero i segnali elettromiografici.

Per risolvere il problema d'isolamento elettrico nella comunicazione dei dati, è stata utilizzata un connessione wireless. In particolare, il sistema è in grado di comunicare con un pc o con uno smartphone che hanno un modulo Bluetooth.

Il modulo Bluetooth utilizzato è il famoso RN42, con il quale è possibile stabilire facilmente una connessione ed è possibile trasmettere dati come una normale comunicazione seriale punto

punto.

ELABORAZIONE NUMERICA SEGNALI

Il compito più importante è affidato ad Arduino che tramite un'elaborazione numerica permette di filtrare e ricavare il valore true RMS del segnale acquisito.

Il filtro digitale realizzato è un IIR (infinite impulse response) del 6° ordine con approssimazione di butterworth frequenze di taglio 20Hz e 500Hz.

Senza entrare molto nel dettaglio, vediamo la struttura del filtro IIR.

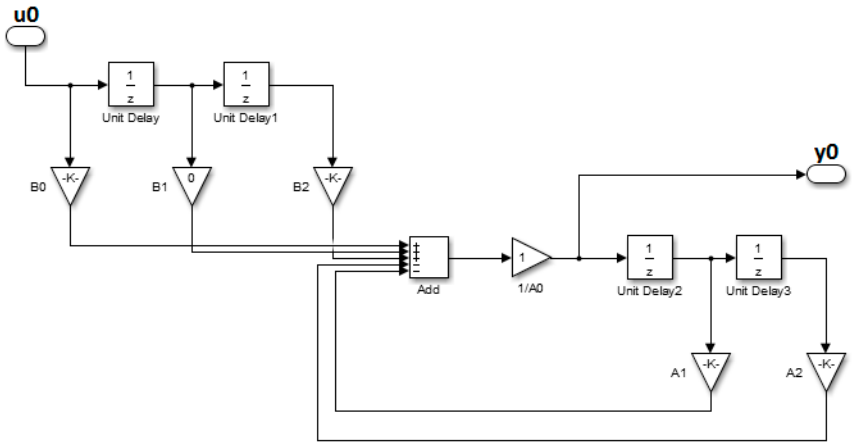


Figura 4: filtro digitale IIR 2° ordine

La figura 4 rappresenta un filtro IIR del secondo ordine che collegato in cascata con altri due filtri identici, costituiscono un filtro del sesto ordine. I coefficienti A0,A1,A2,B0,B1,B2 sono stati calcolati con il software Matlab, garantendo le specifiche in frequenza.

Il passo successivo è stato la conversione del sistema in figura 4 nel codice per Arduino.

```
const float b0=0.0954,b2=-0.0954,a1=-1.729,a2=0.787;

float filter_IIR(void)
{

  u2=u1;

  u1=u0;

  u0=in(i);

  y2=y1;

  y1=y0;

                y0=b0*u0+ b2*u2 -a1*y1 -a2*y2;

  return y0;

}
```

Essendo Arduino M0 pro un micro a 32 bit, permette facilmente di operare con numeri float a 32 bit. Ogni campione IN(i) acquisito dall'ADC con una frequenza di campionamento di 5KHz, viene dato in pasto alla funzione filter_IIR che effettua un filtraggio e ne ritorna il valore.

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i{}^2}$$

A questo punto viene calcolato il valore true RMS con il seguente algoritmo:

Dove N è la finestra dei campioni, nel nostro caso N= 100.

```
float true_RMS(void)
```

```
{

    For( i=0;i<100;i++)

  {

    Somma=(Y[i]^2)/100;

   }

   return somma;

}
```

Dopo essere stato elaborato, il segnale è inviato tramite Bluetooth all'utente finale. Essendo in questo caso il dato in uscita a 32 bit, esso viene spezzato in 4 byte. Alla ricezione, viene ricostruita l'intera parola e vengono visualizzati su un grafico i dati finali.

SOFTWARE

Com'è stato accennato nei capitoli precedenti, non è presente uno shield che fa da interfaccia visiva, per questo motivo è stato studiato un sistema di visualizzazione dati sia tramite pc sia smartphone. Grazie al canale Bluetooth non dobbiamo preoccuparci del protocollo di comunicazione, poiché il modulo RN42 prevede all'incapsulamento dei dati secondo lo stack Bluetooth. Il massimo baud-rate ammesso è di circa 1Mb/s il che permette uno scambio dati agevole, considerando poi che tutta l'elaborazione è interna all'Arduino, i dati da fornire in uscita hanno una baud rate di pochi Kb/s.

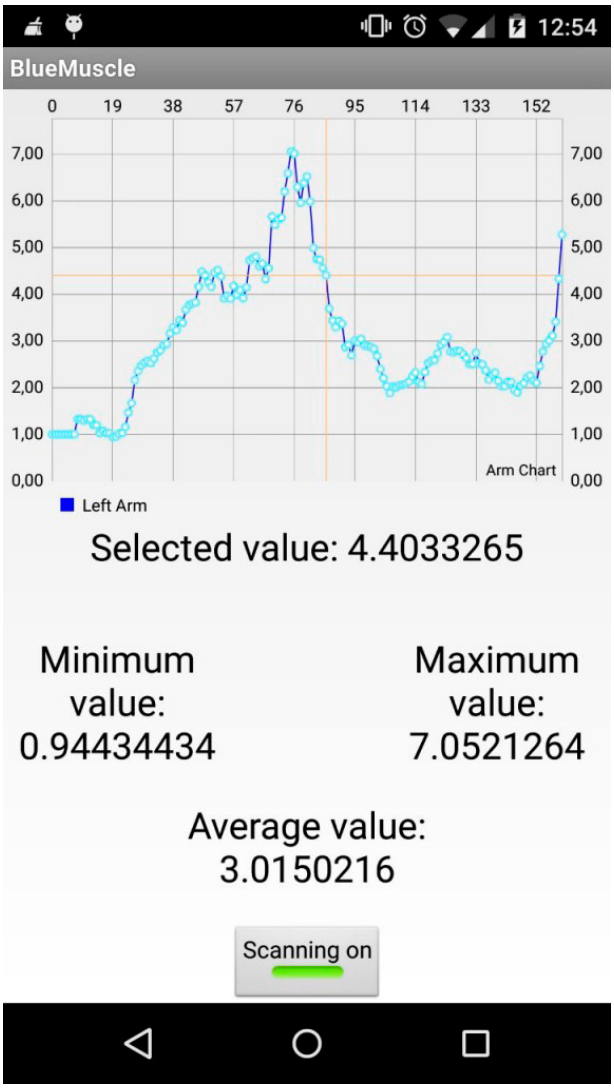


Figura 5: applicazione android

L'applicazione è stata sviluppata per Android con il software Eclipse in java, ricorrendo alle numerose API che il mondo android mette a disposizione. Con l'applicazione per smartphone è possibile visualizzare esclusivamente un canale alla volta, però allo stesso tempo il programma immagazzina anche i dati degli altri canali.
Il software per il pc è stato sviluppato in visual C# con l'ambiente visual studio della Microsoft. Com'è possibile notare dalla figura 6 il sistema è stato progettato per acquisire un massimo di 5

coppie di elettrodi. I sensori (elettrodi) vengono acquisiti sequenzialmente, elaborati e inviati in maniera seriale al pc e agendo sui “tab” presenti nel software è possibile visualizzarne l’andamento nel tempo.

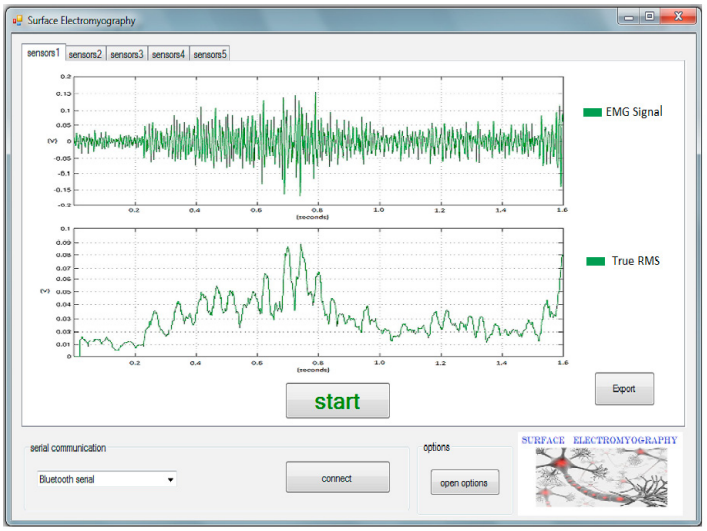


Figura 6: software pc

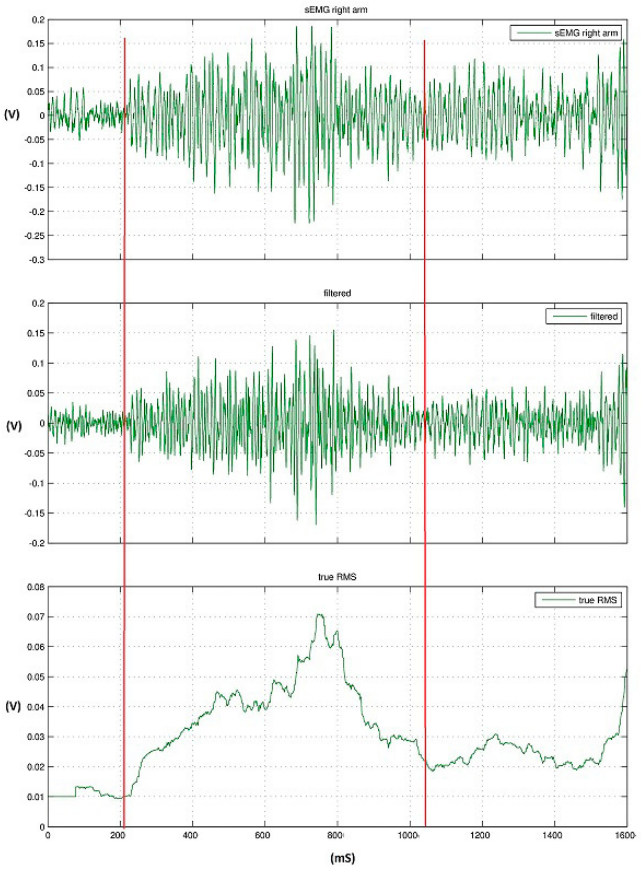


Figura 7: dati braccio destro

RISULTATI

Per verificare il funzionamento dell’intero sistema, sono state fatte due misure elettromiografiche sollevando un peso calibrato da 2Kg, con il braccio destro e con il braccio sinistro. Di seguito sono riportati i risultati ottenuti.

Dai grafici esportati con il software Pc è possibile riscontrare come l’attività muscolare del braccio destro non sia identica a quello sinistro come ampiezza e come valore RMS. Tra le linee rosse verticali, possiamo vedere la contrazione del muscolo quando solleviamo il peso. Per evidenziare meglio questa differenza si potrebbe effettuare una trasformata di Fourier in digitale (FFT) del segnale mioelettrico quantificando il contenuto in frequenza e la densità spettrale di potenza.

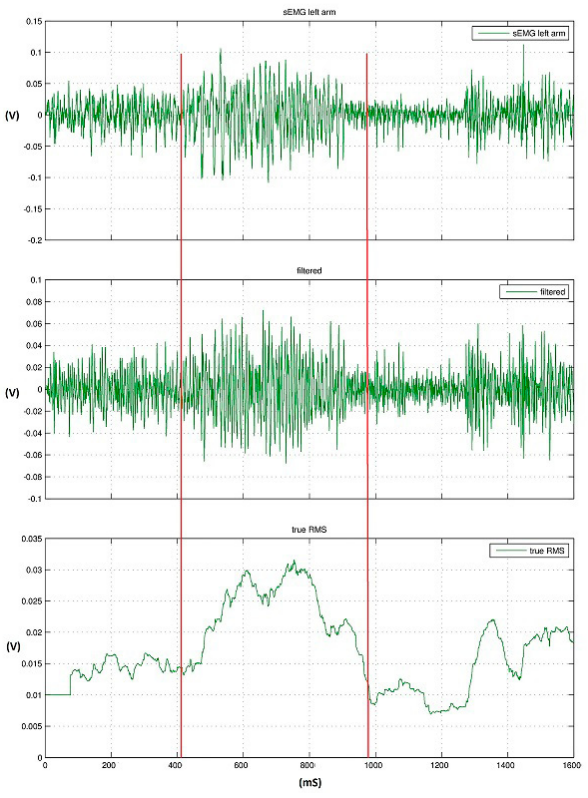


Figura 8: dati braccio sinistro

SVILUPPI FUTURI

Il seguente sistema è da intendersi come un prototipo di test per tutte quelle persone che vogliono conoscere l'attiva muscolare quando svolgono un'attività fisica, senza dover ricorrere a complicati mezzi che costano centinaia di euro. Il prototipo può essere modificato a proprio piacimento secondo quello che si vuole analizzare; è un'idea basilare da cui partire e sviluppare qualcosa di più interessante, utilizzabile in altre circostanze come ad esempio: motion control, protesi artificiali e quant'altro possiamo immaginare. Com'è stato accennato nel capitolo precedente, un altro parametro che andrebbe misurato è la composizione armonica del segnale mioelettrico per quantificare e qualificare al meglio l'attività muscolare.

BIBLIOGRAFIA

1. **kisshealth.** kisshealth laboratorio di biomeccanica posturale. [Online] www.kisshealth.it.
2. **Bentivoglio, M.** Anatomia umana e Istologia . torino : Minerva Medica, 2010.
3. **thought technology ltd.** thoughttechnology. thoughttechnology.com. [Online]

ALLEGATI

AllegatiEMG

L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/emg-superficiale-per-sport-performance

Arduino UNO

- Arduino ai raggi X: cosa fare per renderlo professionale – Prima Parte
- Arduino ai raggi X: rendiamolo professionale – Seconda Parte
- Arduino Studio: getting started
- Inclinometro con Arduino e accelerometro a 3 assi
- Emulare l'Apple II con Arduino UNO
- Un vibrometro real-time con Arduino per il settore industriale

Arduino DUE

- Arduino DUE Tutorial: presentazione e confronto con Arduino UNO
- Arduino DUE Tutorial: dentro la scheda che ha cambiato il mondo
- Arduino DUE Tutorial: Atmel SAM3X8E ARM Cortex-M3 CPU
- Arduino DUE & Eclipse: accoppiata vincente

Arduino ESPLORA

- Scopriamo la nuova scheda Arduino Esplora
- Programmiamo la scheda Arduino Esplora
- Rendiamo autonoma la scheda Arduino Esplora
- Dotiamo l'Arduino Esplora dell'interfaccia Bluetooth
- Gestione di un dispositivo Pan & Tilt con la scheda Arduino Esplora

Arduino M0 Pro

- Arduino M0 Pro: presentazione e specifiche tecniche
- Arduino M0 Pro: getting started
- Arduino M0 Pro: debug con GDB e OpenOCD
- Progetto di una libreria per LCD 16×2 compatibile con Arduino M0 Pro
- Termometro ed Igrometro con Arduino M0 Pro su display LCD 16X2
- EMG superficiale per sport performance con Arduino M0 Pro