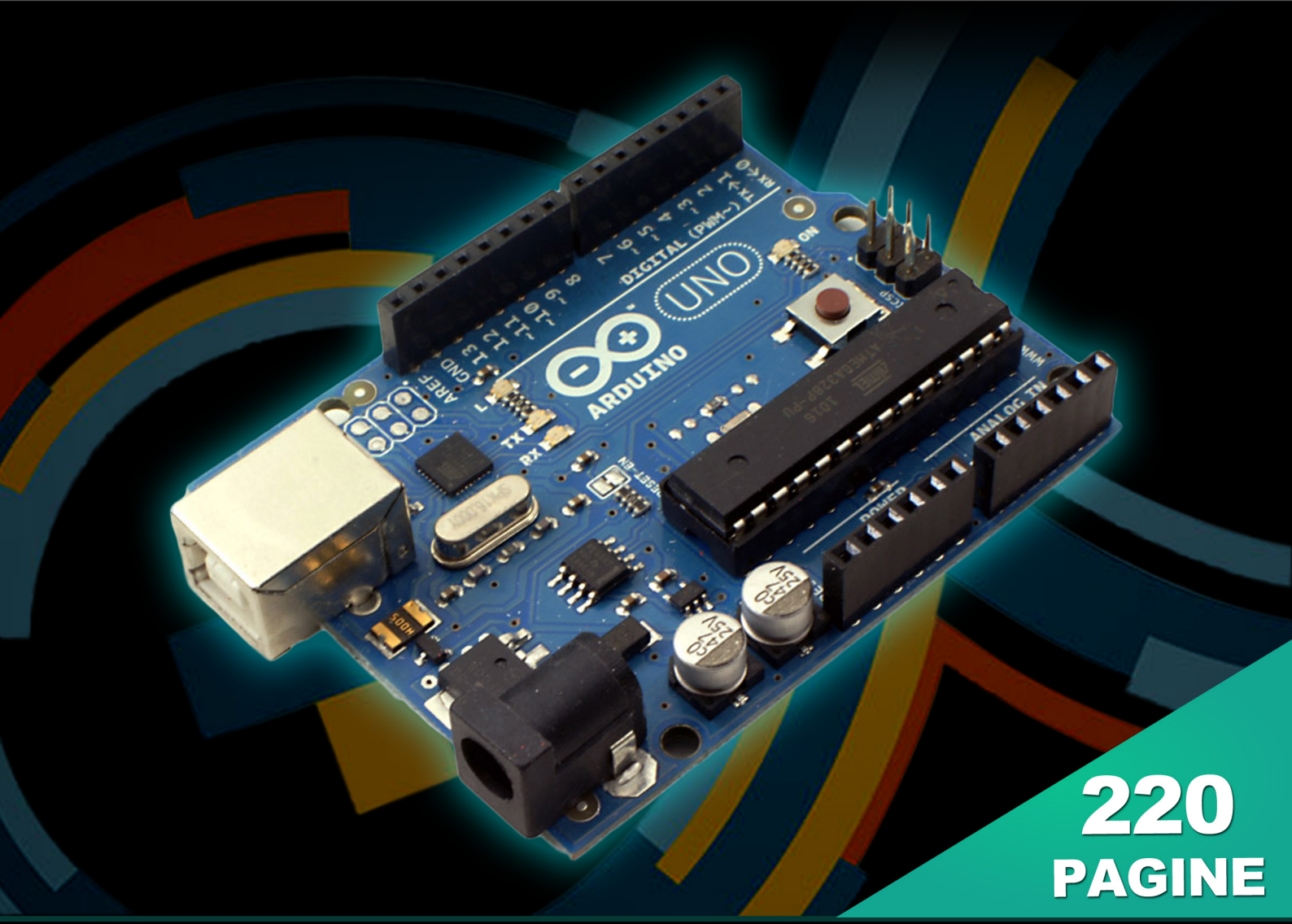
OS Book @8

SPECIALE ARDUINO

TUTORIAL & PROGETTI FAI DA TE



220 PAGINE

Elettronica Open Source

TUTORIAL

3 Just another "Getting Started with Arduino"

13 Come scrivere una libreria per Arduino

17 Sviluppo Software per Arduino con Atmel Studio

26 Programmare Arduino UNO con Atmel Studio

34 Un encoder per Arduino

43 SMAs: gestiamo questi materiali "intelligenti" con Arduino

51 Realizzazione di un rilevatore SONAR con Arduino

64 Più sprint alla tua auto con Arduino e Processing

72 La regolazione di temperatura con Arduino

80 MEMS, dalla teoria alla pratica: usiamo un Accelerometro con Arduino

87 Temperature & pressure monitoring con Arduino

95 Pilotare motori passo-passo con Arduino

98 Controllare I/O multipli con pochi pin di Arduino

PROGETTI

102 Analizzatore MIDI con Arduino

112 Progetto serra domotica open source

121 Inverter ad onda sinusoidale Open Source con scheda Infineon Arduino motor shield

125 Camera slider DIY low cost con Arduino [Progetto completo]

135 Arduino Micro e BMP180 Bosch per realizzare una Weather Station

150 Progetto Giardiniere: Gestire una serra domestica con Arduino

156 Un telecomando TV per comandare un robot cingolato

172 G.F.A.rduino (Generatore di Funzioni Arbitrarie con Arduino)

178 Arduino abbassa il volume degli spot TV! [Fai-Da-Te]

185 Stazione Meteo Online con Arduino

195 Crepuscolare fai-da-te con Arduino

198 Realizziamo una Smart Sveglia Bluetooth con Arduino

214 Costruiamo un Voice Shield per far parlare Arduino [Progetto Completo Open Source]

Speciale Arduino TUTORIAL

Just another "Getting Started with Arduino"

di Luigi Francesco Cerfeda

Come già saprete, il web pullula di pagine dedicate ad Arduino. Facendo una breve ricerca, infatti, si trovano un'infinità di **tutorial step-by-step** e **getting started** che preparano il novizio maker all'utilizzo della scheda. Esistono anche moltissimi articoli che parlano della storia, ormai diventata **leggenda**, di Arduino e della filosofia che c'è dietro la scelta di realizzare un progetto open source. Anche su EOS, ovviamente, si è parlato a fondo di questi argomenti. In particolare Piero Boccadoro si sta occupando, in modo impeccabile ed esaustiva, della scheda Arduino DUE nella sua serie di articoli denominati "**Arduino DUE Tutorial**".

Fatte queste considerazioni, mi sono chiesto quale contributo avrei potuto dare alla community nello scrivere l'ennesimo articolo che riguardasse la mitica scheda ***made in Italy***. Così ho deciso di descrivere semplicemente quali sono state le mie impressioni, le difficoltà che ho affrontato e come le ho superate nel mio primo approccio con la scheda, che in realtà coincide anche con la mia prima esperienza con l'**elettronica pratica**. E qui entra in gioco il nostro **Sheldon**, il tizio, raffigurato nell'immagine di apertura, che guarda con sospetto e un po' di

timore la gente entusiasta che si appresta ad entrare nell'Arduino Park.

Per chi non lo sapesse, Sheldon Lee Cooper, star della sit-com *Big Bang Theory*, è un geniale scienziato e fisico teorico, completamente avverso all'ingegneria pratica, come a qualsiasi altra attività umana che implichi uno sforzo fisico, a parte quella di premere ripetutamente i pulsanti di un joystick o giocare alla Wii.

Come ogni altro tutorial, anche il mio avrà come destinatari i principianti dell'ingegneria pratica, ma la cosa nella quale si distinguerà da altri è che **è stato scritto da un principiante dell'ingegneria pratica** e non da un professionista e/o smanettone navigato. Userò quindi un approccio "***newbie2newbie***", se mi passate il termine. Newbie, ahimè, non per scelta, come nel caso di Sheldon, ma per un motivo che credo sia abbastanza comune a tutti gli studenti di ingegneria in Italia, e cioè, la cronica e patologica **mancanza di laboratori** nelle università.

Prima di iniziare a descrivere la mia esperienza con la scheda, credo sia giusto dedicare due righe alla **storia di Arduino**, cercando di fare il punto della situazione e delinearne '*lo stato dell'arte*' dopo più di 8 anni dalla sua nascita.

Per affrontare questo argomento si è ormai creato uno standard, il quale prevede l'articolazione del discorso sulla base di una pseudo-intervista con domande-risposte pensate ad hoc per spiegare, in un modo più user-friendly ed accattivante, i vari aspetti del progetto Arduino e dello scenario *hardware open source*. Anch'io farò lo stesso.

Le domande di rito sono: *Cos'è Arduino? A cosa serve? Chi lo può utilizzare? Perché ha avuto un così grande successo? Si può guadagnare con l'Open Source?*

Bene, andiamo con ordine:

- ***Cos'è Arduino?***

Cominciamo dal principio. Arduino da Dadone, o Arduino da Pombia, conosciuto come Arduino d'Ivrea, fu re d'Italia dal 1004 al 1014..... No, scherzo, non da così indietro !

Il nome Arduino, del resto, con buona pace del vecchio monarca, è diventato famoso in tutto il mondo solo mille anni dopo, grazie all'intuizione di un team di tecnici ed ingegneri capitanati da **Massimo Banzi**, professore all'**Interaction Design Institute**, un istituto di formazione post-laurea con sede a **Ivrea**, luogo fondamentale per l'informatica italiana dove già la **Olivetti** aveva un tempo il suo nucleo.

Dal sito ufficiale:

> **"Arduino è una piattaforma di prototipazione elettronica open-source che si basa su hardware e software flessibili e facili da usare. È stata creata per artisti, designer, hobbisti e chiunque sia interessato a create oggetti o ambienti interattivi. telecomunicazioni e le industrie mediche di utilizzo finale.**

Arduino, dunque, è un **framework open source** che permette la prototipazione rapida e l'apprendimento veloce dei principi fondamentali dell'elettronica e della programmazione. È composto da una piattaforma hardware alla quale viene affiancato un ambiente di sviluppo integrato (IDE) multipiattaforma (per Linux, Apple Macintosh e Windows), scritto in Java e derivato dall'IDE creato per il linguaggio di programmazione **Processing** e per il progetto **Wiring**.

- ***A cosa serve? Chi lo può utilizzare?***

Arduino è progettato per risultare volutamente **semplice**, essendo destinato ad introdurre alla programmazione ed all'elettronica artisti, designer e costruttori di gadget fai-da-te. Il progetto, infatti, prese avvio nel 2005 con lo scopo di rendere disponibile agli studenti dell'Interaction Design Institute un dispositivo per il controllo dei loro progetti che fosse più economico rispetto ai sistemi di **prototipazione** allora disponibili.
Andando più a fondo nello studio dell'elettronica di Arduino, si può notare che si tratta di una tecnologia abbastanza semplice, o meglio si tratta di un progetto che mette insieme una serie di tecnologie standard, già ampiamente usate e testate da community di ingegneri o semplici makers di tutto il mondo. Non si tratta dunque di hi-tech, o di elettronica di ultima generazione. In un' intervista, lo stesso Banzi afferma, addirittura, che l'elettronica di Arduino sia facilmente replicabile da un diplomato ITIS. Proprio per questa sua apparente semplicità, il progetto ha subìto nel tempo numerose critiche; sempre nella stessa intervista Banzi ne cita una per tutte: "Arduino è uno strumento per donne, pittori e artisti" (che poi dove starebbe la critica non l'ho capito, vabbè).

- ***Ma allora, perché ha avuto un così grande successo?***

Come già detto in precedenza, Arduino è un progetto completamente open source, quindi chiunque lo desidera può legalmente e gratuitamente scaricare lo schema elettrico e l'elenco dei componenti elettronici necessari e auto-costruirsi la scheda nella versione originale o derivarne una **versione modificata** e, addirittura, rivenderla. Per chi è abituato al sistema **closed source**, dove tutto è brevettato e "guai a chi copia (o anche semplicemente ci mette mani!)", questo sistema può sembrare una pazzia. E in effetti, come disse lo stesso Banzi, "c'è una sottile linea di confine tra l'open source e la stupidità".
In realtà si è visto che, sulla scia del software open source, anche l'hardware open source funziona.
Una cosa fondamentale da capire per tentare di motivare questo successo è che la forza di Arduino non è la scheda, ma è la sua **Comunità**. Banzi e il suo team, fondamentalmente, hanno capito che quando la gente ha accesso ai sorgenti dei progetti, suggerisce modifiche. E non solo.
Gli stessi utenti/clienti, a differenza che in passato, sanno già quello che vogliono e dunque risulta più efficiente farlo progettare a loro. Per avere successo in futuro, i produttori di hardware, infatti, dovranno cambiare radicalmente mentalità. Il loro lavoro non è più soltanto quello di avere idee, ma è altrettanto importante cercare e trovare **innovazioni dagli utenti**, i quali, in sostanza, costituiscono anche il loro **servizio tecnico**, a disposizione 24 ore al giorno, 7 giorni alla settimana, senza alcuna spesa, sfruttando al massimo le potenzialità di internet e del Web 2.0. Questo sistema inoltre garantisce molta più **pubblicità (gratuita)** di quanta ne avrebbe potuto ottenere un pezzo di hardware chiuso e proprietario.
L'unico elemento di **proprietà intellettuale** che il gruppo si è riservato è stato il nome, che è diventato il suo **marchio di fabbrica**, in modo che il brand sia ampiamente riconoscibile e che non venga danneggiato da copie di scarsa qualità.
È tutto questo che permette ad un progetto nato in Italia, da un azienda con relativamente poche risorse, di poter essere competitivo a livello globale, e, addirittura, capace di indirizzare

il mercato dell'elettronica a basso costo verso la creazione di prodotti open source **Arduino compatibili**, sia dal lato hardware che da quello software.

- ***Ma, in sostanza, è possibile guadagnare in un mondo di hardware open source?***

Riflettendo sulle considerazioni fatte prima sul perchè Arduino ha avuto questo grande successo, si può facilmente rispondere anche a questa domanda.

Proprio grazie alla comunità che che si crea intorno ad un progetto open source, in pratica, l'azienda produttrice può usufruire di un notevole **taglio ai costi di ricerca e sviluppo**. Si crea, così, un **circolo virtuoso** in cui i clienti fanno risparmiare l'azienda produttrice che a sua volta può investire nella qualità del proprio prodotto e della documentazione offerta all'utente, che, in fin dei conti, vede abbondantemente ripagato il suo sforzo iniziale.

Insomma sembra proprio sia un sistema **più equo, più democratico e, soprattutto, più sostenibile** rispetto al vecchio sistema closed.

Nel caso di Arduino, poi, si è avuto questo **grande successo**, e parliamo di un successo veramente enorme (basta vedere il numero di risultati se si fa una ricerca su google), perchè, come disse Emanuele in questo articolo, è un progetto molto ***cool***, ormai un must-have per ogni nerd o smanettone che si rispetti. Per inciso, Arduino deriva dal nome germanico *Hardwin* che, composto dalle radici *hard* ("forte", "valoroso") e *win* ("amico"), significa "amico valoroso" ... insomma, GENIALE !

Ad ogni modo, riprendendo ancora le parole di Banzi, Arduino è un grande esempio di come "combinando gli elementi del **design**, di cui l'Italia è maestra, con la tecnologia e creando il **branding** giusto, si riesce a vincere nel mondo" e che "non ci vuole il permesso di nessuno per rendere le cose eccezionali."

OK, BASTA CHIACCHIERE, PASSIAMO A QUALCOSA DI PIÙ PRATICO, APPUNTO

Come scrissi nel commento che poi mi ha fatto vincere la scheda, Arduino DUE mi servirebbe per controllare un **motor driver** che a sua volta pilota i motori di un **robot-rover 2WD** in grado di muoversi in modalità CW, CCW, short-brake, e stop. Tale robot-rover, poi, dovrebbe essere usato come applicazione per una Brain Computer Interface, e, nello specifico, funzionare come una sorta di **pre-prototipo di una carrozzina elettrica per disabili** controllata grazie all'elaborazione e traduzione del segnale **EEG** (il tutto usando hardware e software open source). Un progetto ambizioso, certo, ma l'entusiasmo non manca.

Non tutti quelli che, come me, sono alle prime armi, però, hanno la fortuna di avere già ben in mente cosa realizzare con la propria scheda e, paradossalmente, la scelta del progetto iniziale potrebbe essere già un **ostacolo** difficile da superare. In effetti, come detto prima, esistono un'infinità di *getting started* e per il novizio maker questa abbondanza può essere a volte più **frustrante** che utile. Di solito, chi è alle prime armi non possiede né attrezzatura né componentistica elettronica e deve cominciare ad allestire il proprio laboratorio da zero. Si va quindi alla ricerca di un progetto che sia economico, ben documentato, facile da realizzare ma che allo stesso tempo dia delle **soddisfazioni** nella realizzazione ed eventualmente nel suo utilizzo.

Cercare un progetto che soddisfi queste caratteristiche richiede tanto tempo e una buona dose di **pazienza**, cosa che non tutti possiedono, soprattutto se si è all'inizio e si commette l'errore di voler anticipare i tempi. C'è poi da considerare che una soluzione perfetta non esiste e ogni volta che arriva a casa il materiale che si è ordinato si ha sempre qualche **dubbio** sulla bontà della scelta di un componente invece di un altro.

In questo articolo cercherò di indirizzarvi su cosa comprare e in che modo procedere per realizzare un progetto economico che dia già una discreta soddisfazione, **partendo dalle basi** dell'elettronica e dell'informatica, in pratica dagli esempi già presenti nelle librerie di Arduino.

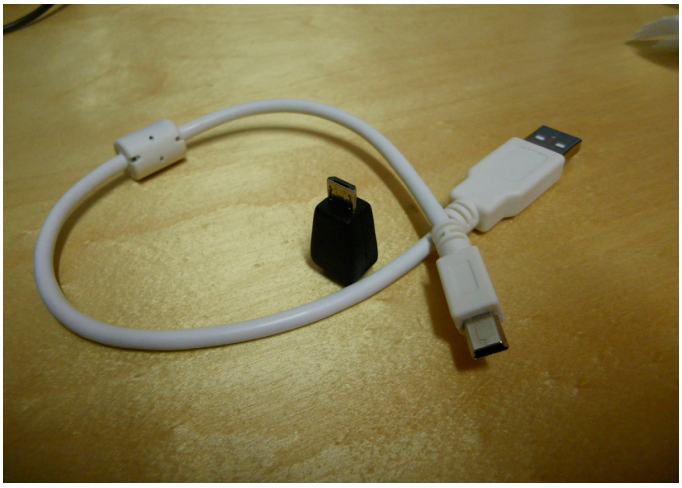
Nello specifico, il progetto che propongo di realizzare è un **dispositivo interattivo** che permette di accendere e spegnere un LED in base al rilevamento della presenza di un ostacolo entro una certa distanza limite. La scelta di proporvi questo tipo di progetto deriva dalle considerazioni fatte precedentemente e dal fatto che un sistema del genere sarà implementato nel mio progetto riguardo il robot-rover prototipo della **carrozzina per disabili**. Come facilmente comprensibile, infatti, è indispensabile che un dispositivo mobile che trasporta una persona disabile abbia autonomamente la capacità di individuare la presenza di ostacoli e **fermarsi** o effettuare prontamente una manovra in modo da **evitare lo scontro**.

Cosa serve per realizzare il progetto?

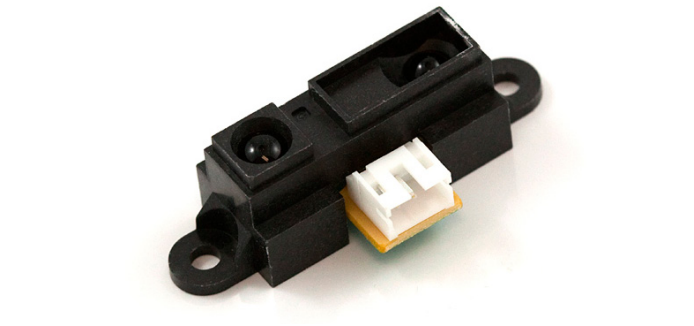
Solo pochi componenti:

- il vostro Arduino, nel mio caso un **Arduino DUE**, gentilmente offerto da **Elettronica Open Source**;

- un cavo USB con connettore micro-USB di tipo B, per il collegamento del proprio computer con Arduino attraverso la **Programming port**, la porta di default per il caricamento degli **sketch**; nel mio caso ho usato un connettore mini-USB con adattatore da mini a micro;



- **un sensore di prossimità**; nel mio caso lo Sharp GP2YOA21YK Infra Red Proximity Sensor (economico, facilmente reperibile e di cui esiste una ampia documentazione). Il sensore ha un connettore Japanese Solderless Terminal (JST). Si consiglia, quindi, l'acquisto del relativo pigtail, il quale viene fornito completamente assemblato e si collega direttamente a molti sensori Sharp.

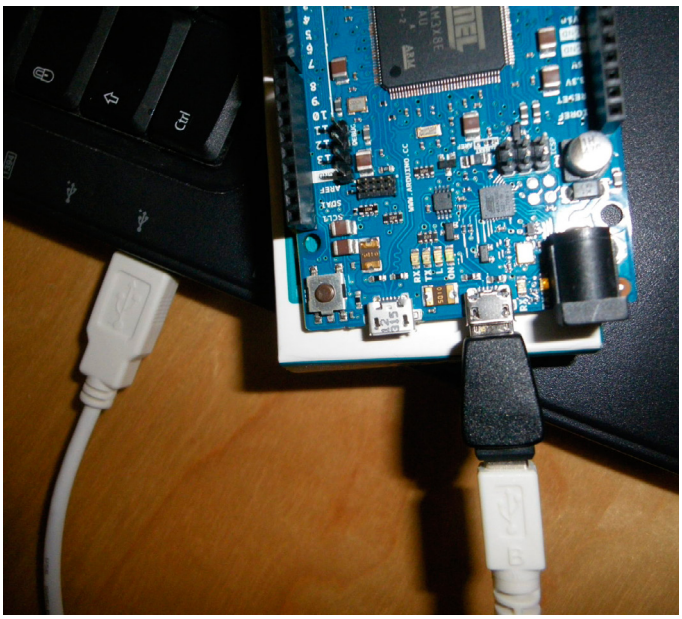


- un computer su cui far girare il software Arduino IDE 1.5 BETA e per alimentare la scheda.

Come dispositivo di output useremo direttamente il **LED** connesso al **pin digitale 13** della scheda. Il suo funzionamento è molto semplice: quando il pin è HIGH il LED è accesso, quando è LOW il LED è spento. È anche possibile effettuare il "dimming" del LED, dato che il pin 13 è anche un output PWM. Un semplice sketch sull'utilizzo del LED 13 è Blink, incluso tra gli esempi dell'Arduino IDE.

Se per voi è proprio la primissima volta con Arduino, sappiate che, per poter caricare gli sketch all'interno del microcontrollore, bisogna eseguire prima queste semplici istruzioni:

- Collegate Arduino al computer attraverso il cavo USB.



- Se si usa Linux, come nel mio caso, non è necessario installare nessun **driver**. In caso si usassero altri sistemi operativi, seguite le istruzioni presenti sul sito ufficiale di Arduino. Connettendo il cavo USB al computer si accenderà il led ON della scheda ed il led L lampeggerà.
- Effettuate il download dell'IDE per Arduino per il vostro sistema operativo. Ricordo che, in caso si possieda Arduino DUE, si deve necessariamente installare la versione 1.5 (ancora in fase **Beta**). Se si usa Linux, una volta scaricato il pacchetto lo si deve estrarre in una directory qualsiasi e fare doppio click sul file *arduino* per aprire l'IDE.
- Una volta avviato il programma selezionate: **Strumenti > Porta seriale** e scegliete la porta seriale.
- Successivamente selezionate: **Strumenti > Board > Arduino Due (Programming Port)**
- Ora siete pronti per caricare gli sketch su Arduino DUE.

Proviamo subito lo sketch d'esempio **Blink**, per il quale, come già detto, non è necessario nessun altro dispositivo o circuiteria esterna oltre che al nostro Arduino:

- Da **File > Esempi > 0.1Basics** selezioniamo Blink. Si aprirà una schermata contenente il seguente codice

```
/*
  Blink
  Turns on an 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.
```

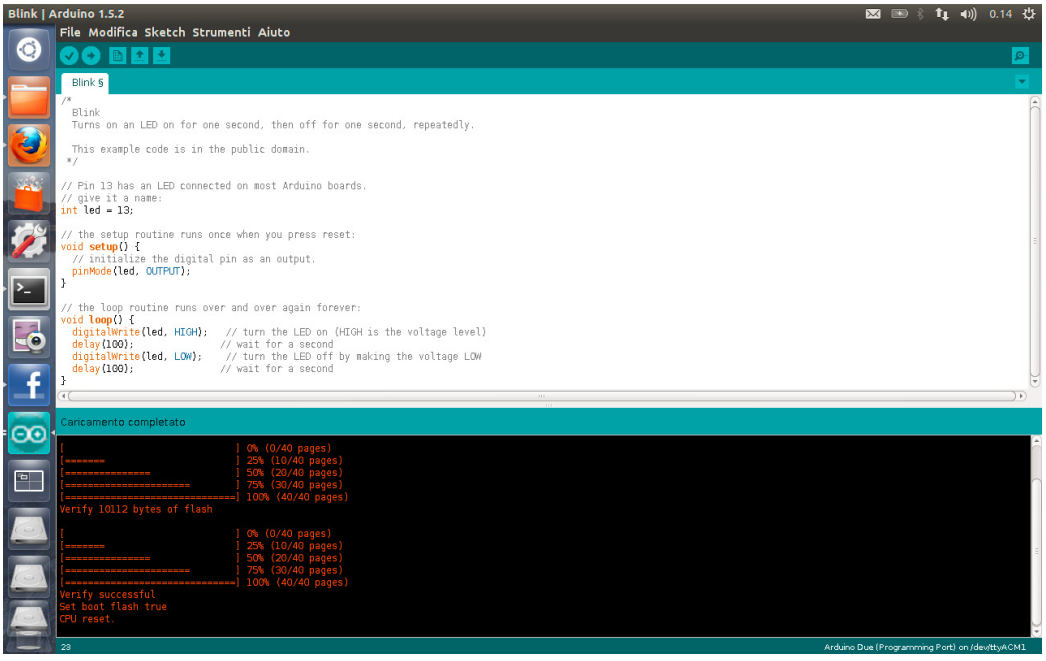
```
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

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

}
```

- Verifichiamo il codice cliccando sul tasto in alto a sinistra "**Verifica**" (ovviamente questa operazione è superflua in questo caso!).
- In caso non ci siano errori andiamo a caricare il codice premendo sul tasto "**Carica**".
- Attendiamo qualche secondo e dovrebbe presentarsi una schermata come quella riportata di seguito. Guardando la scheda, invece, potremo vedere il LED 13 lampeggiare.
- **Congratulazioni**, Avete caricato correttamente il vostro primo sketch!!

Come abbiamo detto prima, però, non ci limiteremo a far lampeggiare un LED ma cercheremo di fargli cambiare stato in risposta all'elaborazione di informazioni che provengono da un dispositivo di input esterno, nel nostro caso da un sensore di prossimità.

Guardando il datasheet del sensore Sharp GP2Y0A21YK possiamo notare che ha un'**uscita analogica** e va, quindi, interfacciato ad uno dei 12 analog inputs presenti sulla scheda. Da notare che nonostante Arduino DUE abbia un ADC a 12 bit (quindi con la capacità di avere 4096 differenti valori), per default la risoluzione è impostata a 10 bit in modo da garantire la compatibilità con le altre schede Arduino che lavorano, appunto, a 10 bit. E' comunque possibile cambiare la risoluzione dell'ADC con il comando analogReadResolution().

Prima di collegare fisicamente il dispositivo cerchiamo di ragionare sul codice da utilizzare per la lettura di un input analogico. Anche in questo caso ci vengono incontro gli esempi presenti di default nell'IDE. In particolare, useremo lo sketch ReadAnalogVoltage, anch'esso selezionabile da **File > Esempi > 0.1Basics**.



Il codice in questo caso è:

```
/*
  ReadAnalogVoltage
  Reads an analog input on pin 0, converts it to voltage, and prints the result to the serial monitor.
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.

 This example code is in the public domain.
 */
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}
// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
```

Ora che sappiamo come far “brillare” un LED e come leggere un segnale analogico, possiamo unire i due sketch per ottenere quello che ci serve per il nostro progetto. Ed ecco qua:

```
/*

Semplice Sketch sull'utilizzo interattivo di un dispositvo di input (GP2YOA21YK Infra Red Proximity Sensor)   con uno di output (LED sul Pin 13 di Arduino), adattando il codice degli Sketch di esempio ReadAnalogVoltage e Blink e aggiungendo la struttura di controllo “if else”

*/

int sensorPin = 5; //analog pin 5

// 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 serial communication at 9600 bits per second:
Serial.begin(9600);

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

// the loop routine runs over and over again forever:
```

```
void loop() {
// read the input on analog pin 5:
int sensorValue = analogRead(sensorPin);

// Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 3.3V):
// Si noti che su Arduino Due la tensione massima che i pin I /O sono in grado di tollerare //è di 3.3V e non 5V. Il valore considerato per la conversione è quindi 3.3V e non 5V, //come da default nello sketch d'esempio ReadAnalogVoltage

float voltage = sensorValue * (3.3 / 1023.0);
Serial.println(voltage);
//just to slow down the output - remove if trying to catch an object passing by
delay(100);        // delay in between reads for stability

if ( voltage < 1 )     // se l'ostacolo si trova ad una distanza tale per cui  il segnale in uscita //dal sensore è minore di 1 V ( distanza superiore a circa 30 cm) il LED si //spegne,altrimenti si accende
{digitalWrite(led, LOW);   // turn the LED off by making the voltage LOW
}else
{
digitalWrite(led, HIGH);    // turn the LED on (HIGH is the voltage level)
}
}
```

Anche in questo caso lo sketch è molto semplice.

Nella sezione globale definiamo le variabili che intendiamo usare e assegniamo dei nomi ai piedini che useremo per l'ingresso e per l'uscita. Il codice di inizializzazione (setup) configura i piedini per le loro rispettive funzioni ed inizializza la comunicazione seriale, in modo da essere in grado di inviare i dati al PC. Il codice eseguito nel ciclo principale (loop) legge dall'ingresso analogico e copia il dato letto nella variabile che abbiamo allocato. Poi fa un semplice test per verificare se il valore letto è al di sotto di un valore soglia, nel qual caso spegne il LED.

Come si può notare la soglia è impostata in base al valore di potenziale letto e non in base alla effettiva distanza dell'oggetto. Sarebbe opportuno quindi fare una **conversione** da V in cm in base alla **curva caratteristica** del sensore. Guardando il datasheet si nota che, purtroppo, tale curva **non è lineare**. Per una conversione soddisfacente sarebbe, quindi, opportuno effettuare delle operazioni matematiche per linearizzare la curva ed ottenere una funzione di conversione da aggiungere nello sketch, oppure affidarsi a librerie scritte ad hoc per il sensore che si sta utilizzando. Purtroppo questa libreria funziona solo per la vecchia versione di Arduino e non per Arduino DUE. Nel caso disponiate di un Arduino precedente al DUE, potrete utilizzare la libreria seguendo le istruzioni presenti nella pagina già indicata, dove c'è anche una breve spiegazione di come funziona il sensore.

OSSERVAZIONE: una nota assolutamente importante è che, a differenza di altre schede Arduino, **la scheda Arduino DUE funziona a 3.3V**! La tensione massima che i pin I /O sono in grado di tollerare è di 3.3V. Fornire tensioni più elevate, come 5V a un pin I /O potrebbe dan-

neggiare la scheda.

Per poter essere sicuri che la tensione di uscita del sensore sia inferiore a 3.3V diamo uno sguardo al datasheet.

MALE, MOLTO MALE!

Dalla tabella *Absolute Maximum Ratings* si legge che l'*Output terminal Voltage* va da -0,3V a Vcc+0,3V, e considerando che Vcc, cioè l' *Operating Supply Voltage*, è 5V, in linea teorica l'output del sensore può arrivare ben al di sopra di quel valore massimo, con il rischio concreto dei "friggere" la scheda.

Guardando meglio il datasheet, tuttavia, possiamo notare dalla figura 4 che il massimo della curva "*Analog Output Voltage vs. Distance to Reflective Object*" è intorno ai 3 V. Quindi, usando un po' di **accortezza** si potrebbe comunque collegare il sensore stando ben attenti a lasciare il campo libero da oggetti per una distanza di almeno 20 cm. In questo modo l'output arriverà al massimo a circa 1,5 V, ben al di sotto dei fatidici 3,3 V. Nonostante questo accorgimento sia ragionevolmente sufficiente per lavorare in tutta sicurezza, senza la preoccupazione di bruciare qualcosa, vi esorto a **NON** collegare il sensore ad Arduino DUE, a meno che non abbiate abbastanza esperienza e/o vi piaccia rischiare.

Per ovviare a questo inconveniente della compatibilità dei pin a 3,3V, sono aperti numerosi topic sul forum ufficiale di Arduino come anche su vari forum di elettronica in giro su internet. Le soluzioni proposte sono tante ma sembra che non si sia ancora arrivati ad una soluzione definitiva o ottimale e che abbia riscosso più successo delle altre.

Di comune accordo con Emanuele, si è deciso di non proporre una soluzione e di lasciare il quesito aperto alla Community proprio a partire da questo articolo, in modo che se ne possa discutere nei commenti.

Dunque, a voi la parola!

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/just-another-getting-started-arduino

Come scrivere una libreria per Arduino

di slovati

Come esempio, creeremo una libreria molto semplice. Del resto, quello che ci interessa maggiormente ora è apprendere i passi da seguire per la creazione di una nuova libreria; questi passi potranno poi essere applicati anche al caso di librerie per applicazioni maggiormente complesse.

Quali sono i **principali vantaggi** che derivano dall'utilizzo di una libreria? I vantaggi offerti da una libreria possono essere sintetizzati nel modo seguente:

- semplifica l'utilizzo e l'organizzazione del codice. Infatti, una libreria ben scritta e testata, mette a disposizione dell'utilizzare delle funzioni pronte per l'uso: l'utente non deve preoccuparsi di come una particolare funzione è stata implementata, è sufficiente sapere come deve essere utilizzata
- migliora la leggibilità del codice: il codice dell'applicazione si snellisce ed è più semplice da comprendere
- decentralizza la logica: lo sketch può infatti focalizzarsi su uno specifico processing, trascurando (o meglio "nascondendo") gli aspetti implementativi gestiti all'interno della libreria. Si usa spesso dire che una libreria deve specificare cosa, quando, e dove, ma non come
- permette di mantenere aggiornato il codice: se una stessa libreria viene utilizzata da più utenti, quando viene rilasciato un aggiornamento (in seguito ad esempio a del bug-fixing, oppure a seguito dell'aggiunta di nuove funzionalità), la stessa libreria può essere importata dagli utenti, che potranno così disporre immediatamente della nuova versione (con minime o addirittura senza alcuna modifica al proprio codice)

Supponiamo ora di voler scrivere una libreria che permetta di controllare lo stato di un led collegato a una uscita di Arduino. Possiamo pensare alle seguenti funzioni da includere nella nostra libreria:

- initialize: è la funzione di inizializzazione, quindi verrà chiamata una sola volta allo startup del sistema. Riceve come parametro il numero del segnale (pin) della scheda Arduino che si intende utilizzare per comandare il led. Il codice di questa procedura provvederà automaticamente a configurare questo pin come pin di uscita. Non è previsto alcun valore di ritorno dalla funzione. Il suo prototipo sarà perciò di questo tipo:

```
void initialize (byte pinLed);
```

- on: forza l'accensione del led. Non è richiesto alcun parametro e non esiste valore di ritorno. Il suo prototipo sarà perciò il seguente:

```
void on(void);
```

- off: forza lo spegnimento del led. Non è richiesto alcun parametro e non esiste valore di ritorno. Il suo prototipo sarà perciò il seguente:

```
void off(void);
```

- blink: causa il lampeggio del led. Come parametro viene passato il periodo desiderato per il lampeggio. Se ad esempio si vuole ottenere un lampeggio alla frequenza di 20 Hz, occorrerà passerà come parametro il valore 50 (l'unità di misura sono i millisecondi). Per questo parametro è stata prevista una variabile di tipo *unsigned short* in modo tale da poter gestire periodi anche superiori a 255 millisecondi. Non è previsto alcun valore di ritorno, per cui il prototipo di questa funzione è il seguente:

```
void blink (unsigned short periodMs);
```

Possiamo a questo punto vedere l'implementazione completa di queste funzioni; il passo successivo sarà quello di "calarle" all'interno di una libreria:

```
void initialize (byte pinLed)
{
  pinGlob = pinLed;
  pinMode(pinGlob, OUTPUT);
}
```

```
void on(void)
{
  digitalWrite(pinGlob, HIGH);
}
```

```
void off(void)
{
  digitalWrite(pinGlob, LOW);
}
```

```
void blink (unsigned short periodMs)
{
  on();
  delay(periodMs/2);
  off();
  delay(periodMs/2);
}
```

Abbiamo a questo punto visto una possibile implementazione per le funzioni che vogliamo includere nella nostra libreria. Prima di procedere, è però utile spendere qualche parola sulla **convenzione adottata nel mondo Arduino per scegliere i nomi delle funzioni e delle variabili**. Questa convenzione, che seguiremo nella scrittura della nostra libreria, prevede che i nomi della libreria siano in UppercaseUppercase (MaiuscoloMaiuscolo), mentre quelli delle funzioni in lowercaseUppercase (minuscoloMaiuscolo). Ciò significa che la nostra libreria dovrà avere un nome tutto maiuscolo (scegliamo ad esempio il nome **LEDLIB**), mentre i nomi delle funzioni e delle variabili saranno tutti in minuscolo, con maiuscolo solo la prima lettera di ogni parola (eccetto la prima parola, che è tutta in minuscolo). Nel mondo Arduino, le librerie corrispondono a delle classi (si segue la stessa sintassi del C++), cioè a dei particolari tipi di strutture. Ogni classe si suddivide poi in due file:

- un file header (estensione .h), contenente la dichiarazione della classe, oltre alla definizione di eventuali tipi e costanti. Nel nostro caso, questo file si chiamerà LEDLIB.h
- un file con il codice sorgente (estensione .cpp), contenente l'implementazione della classe, cioè il codice relativo a tutte le sue funzioni (esportate o locali). Le funzioni esportate da una classe corrispondono a

quelle esportate dalla libreria, e vengono anche chiamate *metodi*. Nel nostro caso, il file sorgente della libreria si chiamerà LEDLIB.cpp

Il listato relativo al file LEDLIB.h sarà perciò il seguente:

```
#ifndef LEDLIB_H
#define LEDLIB_H

#include “Arduino.h”

class LEDLIB
{
  private:
    byte pinGlob;  // pin utilizzato per pilotare il
LED

  public:
    void initialize (byte pinLed);
    void on(void);
    void off(void);
    void blink (unsigned short periodMs);
};

#endif
```

E’ importante notare come tutte le funzioni della libreria siano “public”, quindi visibili all’esterno, e quindi richiamabili da un’applicazione. E’ invece dichiarata come “private” la variabile pinGlob in quanto questa è locale alla classe e non deve essere visibile al suo esterno. Questo è invece il codice relativo all’implementazione della classe (LIBLED.cpp):

Siamo ora a buon punto, dobbiamo ancora vedere come “installare” la libreria (non preoccupatevi, è un’operazione molto semplice), e scrivere un piccolo esempio di utilizzo della stessa. I passi da seguire sono i seguenti:

1. create un nuova cartella “LIBLED” nella sottocartella “libraries” del vostro ambiente di sviluppo per Arduino

```
/*
  LEDLIB.cpp - Libreria di esempio per gestire
  l’accensione, lo spegnimento, e il lampeggio
  di un LED.
*/

#include “LEDLIB.h”  // dichiarazione della classe

/* funzione di inizializzazione */
void LEDLIB::initialize (byte pinLed)
{
  pinGlob = pinLed;
  pinMode(pinGlob, OUTPUT);
}

/* funzione di accensione del led */
void LEDLIB::on(void)
{
  digitalWrite(pinGlob, HIGH);
}

/* funzione di spegnimento del led */
void LEDLIB::off(void)
{
  digitalWrite(pinGlob, LOW);
}

/* funzione di lampeggio del led */
void LEDLIB::blink (unsigned short periodMs)
{
  on();
  delay(periodMs/2);
  off();
  delay(periodMs/2);
}
```

2. copiate i file LIBLED.h e LIBLED.cpp nella nuova cartella LEDLIB
3. lanciate l’ambiente di sviluppo Arduino e selezionate l’opzione Sketch->Import Library dal menu principale: magia! Se avete eseguito correttamente i passi precedenti, sarà visibile nell’elenco la nuova libreria LEDLIB. Non selezionate comunque l’opzione di importazione della libreria, questa

serve solo per includere l'header della libreria stessa nello sketch corrente

Il codice dell'esempio è il seguente:

```
#include <LEDLIB.h>

LEDLIB led;

void setup()
{
  led.initialize(13);
}

void loop()
{
  led.blink(1000);
}
```

La creazione di almeno un file di esempio è molto importante per due motivi:

1. permette di testare la libreria: questa infatti non viene compilata fino al momento in cui è inclusa da un'applicazione
2. la "filosofia" di Arduino prevede di allegare sempre, insieme alla libreria, almeno un file di esempio. Se provate infatti a guardare nella cartella "libraries" della vostra installazione Arduino, potrete trovare la sottocartella "examples" in corrispondenza di ogni libreria presente (ad esempio per la libreria EEPROM)

Se avete provato a caricare l'esempio di cui sopra, avrete potuto notare come l'ambiente di sviluppo di Arduino non abbia riconosciuto nessuno dei nuovi nomi introdotti (la classe e i suoi metodi). Ciò è normale, bisogna infatti "istruire" Arduino su quali sono questi nuovi simboli, in modo tale che l'IDE possa applicare ad essi l'appropriata colorazione. Per fare questo è sufficiente creare un file chiamato "keywords.txt" che andrà copiato nella stessa cartella in cui sono presenti i file della libreria. Nel nostro caso, il contenuto del file sarà il seguente:

```
#######################################
# Syntax Coloring Map For LEDLIB
#######################################

LEDLIB	KEYWORD1

initialize	KEYWORD2
on	KEYWORD2
off	KEYWORD2
blink	KEYWORD2
```

Vengono in pratica elencati i nuovi simboli e il tipo di colorazione da applicare agli stessi (per separare il nome del simbolo e il tipo di keyword, occorre utilizzare un singolo carattere TAB). Questo è l'ultimo passo necessario alla creazione di una libreria. Potete a questo punto distribuire a chi interessato le librerie da voi stessi create: sarà sufficiente, per praticità, comprimere tutti i file della cartella relativa alla libreria e rilasciare il file compresso.

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/come-scrivere-libreria-arduino

Sviluppo Software per Arduino con Atmel Studio

di slovati

Nonostante sia bello ottenere immediatamente dei risultati significativi con i numerosi sketch di esempio già disponibili per Arduino, quando si comincia a sviluppare un'applicazione completa su una board Arduino con processore ARM ci si deve inevitabilmente confrontare con i seguenti problemi:

- la compilazione del programma eseguita nell'ambiente Arduino richiede troppo tempo;
- il caricamento del programma eseguito dall'ambiente di sviluppo Arduino è troppo lento;
- non è disponibile un debugger hardware.

Per porre rimedio a questi inconvenienti, l'autore ha individuato un sistema alternativo che ora descriveremo.

Uno dei principali vantaggi derivanti dall'utilizzo di un microcontrollore a 32 bit è rappresentato da un accesso più rapido alle risorse hardware, soprattutto ai pin di I/O. Ne consegue che un buon sistema per testare le capacità di elaborazione di un microcontrollore è quello di scrivere un programma di esempio (uno sketch Arduino) che genera un'onda quadra con la minor ampiezza possibile, e misura l'ampiezza dell'impulso del segnale in uscita. Nel corso dell'articolo descriveremo come scrivere un simile sketch di test, utilizzando l'ambiente di sviluppo Atmel Studio 6.2 [1].

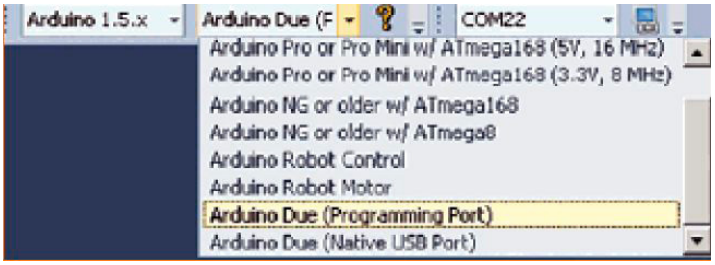
COMPATIBILITÀ CON ARDUINO

La procedura qui descritta assume che abbiate installato il sistema di sviluppo Atmel Studio 6.2 (o una versione successiva) per Windows. Per rendere l'ambiente di sviluppo compatibile con Arduino, occorre inoltre installare l'add-in Visual Micro [2], che è in sostanza un IDE Arduino per Microsoft Visual Studio e per Atmel Studio. Visual Micro è direttamente disponibile in Atmel Studio dopo la sua installazione.

Il debugger USB per Arduino disponibile all'indirizzo [2] non è liberamente scaricabile, ma non è tuttavia necessario in quanto il debugging attraverso l'ICE Atmel è sicuramente più conveniente. Quest'ultimo strumento è disponibile presso numerosi rivenditori on-line a prezzi che si aggirano intorno ai 100 dollari (si veda la presentazione apparsa nel numero Elektor di ottobre 2014 [3]). Vale sicuramente la pena affrontare questo investimento se si utilizzano i microcontrollori Atmel in modo non occasionale.

Grazie a questo debugger è possibile impostare dei veri breakpoint senza dover ricompilare il programma, osservare le variabili nella finestra 'Watch', ed esaminare o modificare il contenuto della memoria. Inoltre, è anche possibile ispezionare i numerosi registri di I/O, e modificarne il contenuto tramite un semplice click del mouse.

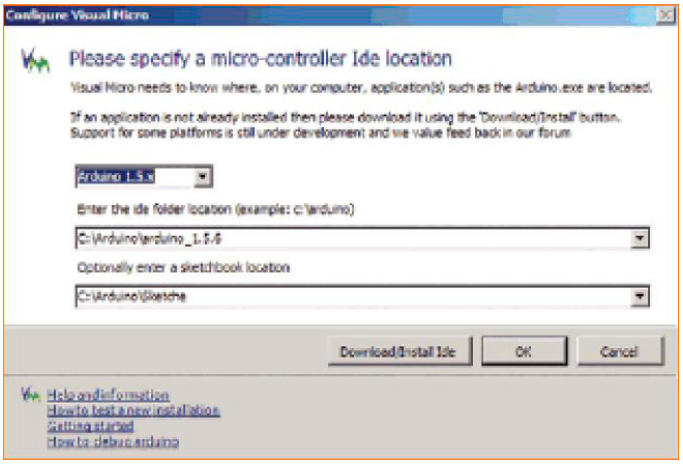
Dopo aver installato Visual Micro, all'interno di Atmel Studio sarà disponibile una nuova toolbar. Con essa sarà possibile selezionare la versione corrente di Arduino (1.5.x), il tipo di scheda (Arduino Due), e l'interfaccia per la programmazione (porta di programmazione), come indicato in **Figura 1**.

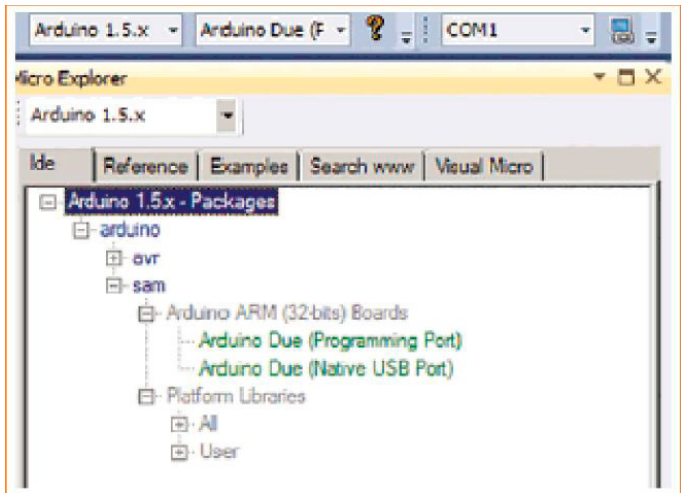


Successivamente occorrerà configurare opportunamente l'interfaccia seriale virtuale, che potrete trovare nella finestra di Gestione dei Dispositivi quando Arduino Due è collegata al PC tramite un cavo USB. Il tasto relativo al monitor seriale si trova, esattamente come nell'IDE Arduino, alla destra di queste impostazioni. Se le impostazioni non compaiono automaticamente, ad esempio quando il software di Arduino non si trova nella cartella di default, occorre selezionare la voce "Configuration Manager". Si aprirà la finestra visibile in **Figura 2**. Qui occorrerà inserire manualmente la cartella di destinazione, in quanto non è disponibile alcuna dialog di selezione della stessa. Premendo il punto di domanda sulla barra menu di Figura 1, si aprirà la finestra Micro Explorer (**Figura 3**), che corrisponde alla finestra Solution Explorer di Atmel Studio. Qui, i riferimenti a Arduino sono rappresentati sotto forma di collegamenti sotto il tab Reference. Cliccando su una di queste voci, si viene rediretti al sito corrispondente. Il tab Examples contiene una lista di esempi (incluse librerie di esempio) di Arduino, raggruppate per argomento.

SKETCH IN ATMEL STUDIO

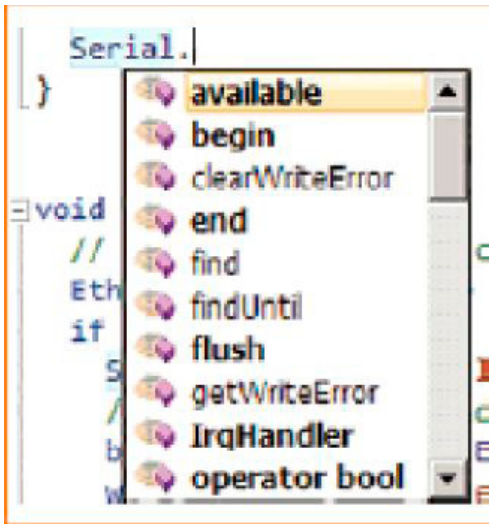
Uno dei principali vantaggi offerti dall'IDE di



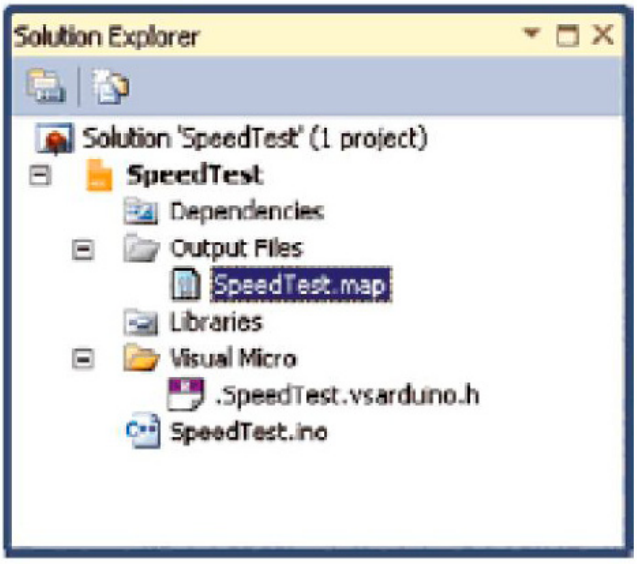


Atmel Studio è la funzione di autocompletamento del codice, che è ora disponibile anche per gli sketch Arduino. Per utilizzarla occorre abilitare Visual Assist X tramite il menu VAssistX->Enable/Disable. Attivando la funzione di autocompletamento, vengono elencate tutte le possibili opzioni di completamento ogni volta che si inserisce un carattere nella finestra Code. Se ad esempio si digita "S" all'inizio di una nuova riga dello sketch, i termini Serial, Server, SPI, e così via, vengono proposti per il completamento. E' poi possibile selezionare direttamente il termine appropriato, senza correre rischi di introdurre errori di sintassi. Non solo, dopo aver inserito un punto, per esempio dopo il termine "Serial", viene proposta una lista con tutti i possibili attributi

e metodi relativi a quella classe (**Figura 4**).



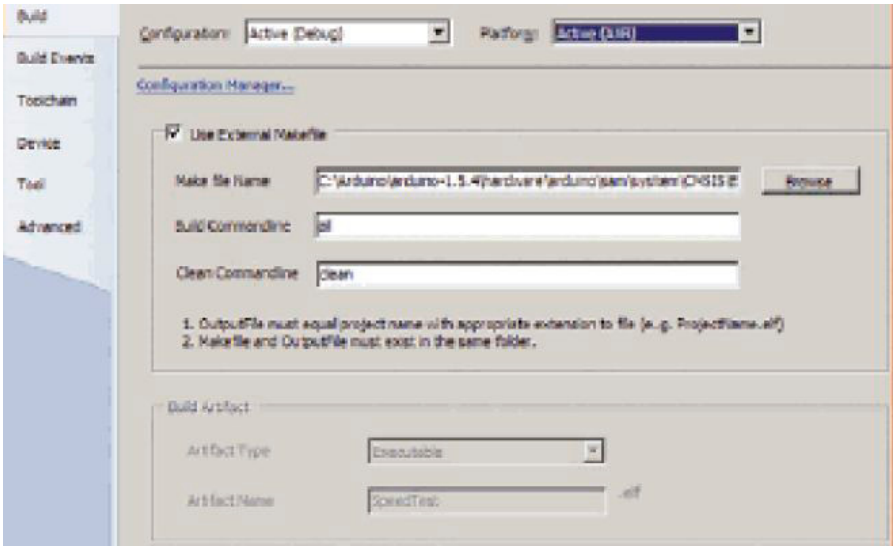
A questo punto, seguendo la consueta procedura, è possibile creare oppure aprire uno sketch nell'IDE Arduino e poi importarlo in Atmel Studio. Tuttavia, esiste anche la possibilità di creare dei nuovi sketch Arduino direttamente in Atmel Studio. Dopo aver creato e salvato lo sketch SpeedTest (si veda più avanti il listato) seguendo uno di questi due modi, nella finestra Solution Explorer apparirà la struttura evidenziata in **Figura 5**. Ora è possibile aprire e modificare lo sketch in Atmel Studio seguendo la modalità consueta. Da questo punto in poi non è più necessario impiegare l'IDE di Arduino.

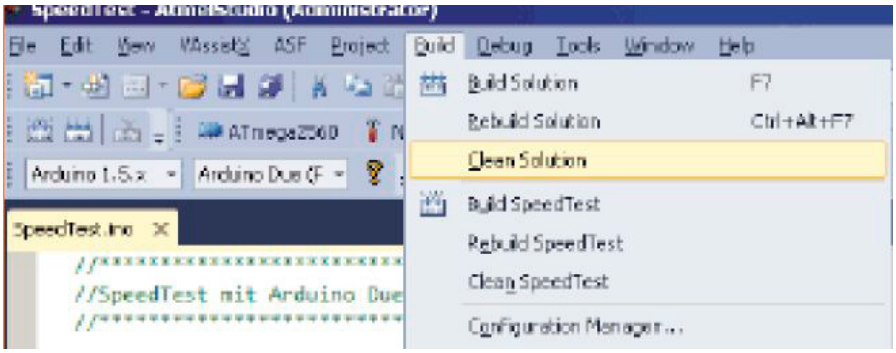


Il fatto che la piattaforma sia mostrata come "Active AVR" nelle proprietà del progetto (click del tasto destro su SpeedTest: Properties->Build) può essere un pò fuorviante, almeno nella versione attuale di Atmel Studio (la 6.2). Purtroppo non esiste alcuna soluzione a questo inconveniente. La piattaforma viene anche evidenziata come "AVR" sotto "Toolchain" e "Device". Tuttavia, ciò non comporta alcuna conseguenza a livello di funzionamento operativo. Selezionanado la board "Arduino Due", il progetto viene compilato esattamente come con la piattaforma SAM per i microcontrollori Atmel ARM. Come visibile in **Figura 6**, la piattaforma AVR è attiva, ma la destinazione è riferita alla toolchain SAM dell'IDE Arduino.

Come consuetudine in Atmel Studio, premendo il tasto F7 si compila il progetto, e premendo il tasto F5 (o in alternativa la freccia di colore verde) si esegue il download del programma sulla scheda tramite il bootloader. E' sempre meglio selezionare "Clean Solution" sul tab Build (**Figura 7**), in quanto è l'unico modo che garantisce che tutti i file siano compilati nello stesso modo dell'IDE Arduino.

Come indicato in precedenza, uno dei principali vantaggi offerti da Atmel Studio, rispetto all'ambiente di sviluppo tradizionale Arduino, è rappresentato dalla riduzione del tempo di compilazione. Questo aspetto permette di accelerare in modo significativo il processo iterativo di compilazione, test, e modifica di un programma. Ciò è dovuto al fatto che Arduino compila sempre tutti i file sorgente, mentre Atmel Studio compila solo i file modificati. Nella maggiorparte dei casi ciò implica che venga compilato soltanto il file .ino, che contiene il codice intero dello sketch. Sul PC dell'autore, la prima compilazione dello sketch SpeedTest ha richiesto 3,7 secondi, mentre la





seconda compilazione (dopo aver eseguito una modifica del codice) ha richiesto soltanto 0,23 secondi.

DEBUGGING CON L'ICE ATMEL

Un vero debugger hardware come l'ICE di Atmel permette di semplificare notevolmente lo sviluppo software: è sufficiente collegare il cavo fornito in dotazione tra l'ICE e la scheda Arduino Due. La board Arduino Due può essere alimentata direttamente dalla porta USB o tramite un adattatore esterno, mentre l'ICE Atmel ha soltanto bisogno di una porta USB.

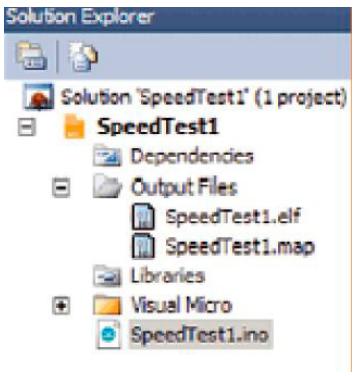
Esistono due modalità distinte di collegamento del cavo di debug. Nel caso della board Arduino Due, la modalità corretta è quella di collegare il cavo da un lato alla porta SAM, e dall'altro lato al connettore JTAG presente sulla board stessa.

La versione attuale di Atmel Studio (6.2) non consente di modificare gli sketch Arduino su una piattaforma ARM Atmel, ma esiste un modo per aggirare questo problema. Per eseguire il debugging con l'ICE Atmel, è sufficiente lanciare una seconda istanza di Atmel Studio; selezionare poi Open->Open Object File For Debugging. I file di compilazione generati da Atmel Studio sono collocati sotto windows 7 nella seguente cartella:

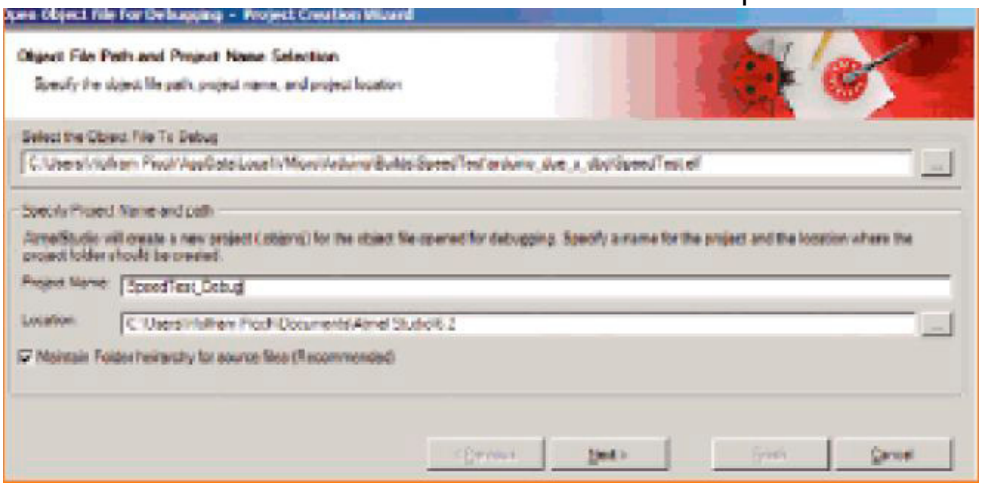
```
C:\Users\XXXX\AppData\Local\
VMicro|Arduino|Builds\SpeedTest\arduino_due_x_
dbg\
```

dove è possibile trovare anche il file SpeedTest.elf. Apriamo ora questo file per eseguire il debug.

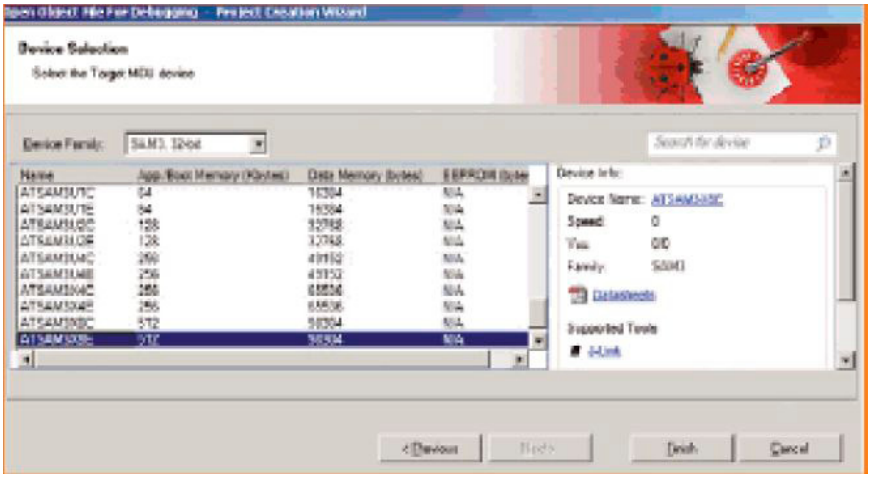
Se lo sketch è stato compilato e salvato con l'IDE Arduino, una copia dello stesso file sarà anche presente nella cartella debug dello sketch. E' possibile verificare ciò nella finestra Solution Explorer dell'istanza di generazione del codice di Atmel Studio, sotto "Outpurt Files" (**Figura 8**).



Utilizzando la funzione di browsing, selezioniamo ora il file di output SpeedTest.elf. Dovrebbe comparire una finestra come indicato in **Figura 9**. E' inoltre possibile specificare un nome e una destinazione a scelta per la cartella debug del progetto. Le impostazioni verranno memorizzate e utilizzate automaticamente quando si aprirà nuovamente il progetto di debug.



Comparirà a questo punto una finestra da cui selezionare la famiglia del dispositivo e il microcontrollore target (**Figura 10**). Nel caso di Arduino Due, occorre selezionare "SAM3, 32 bit" come famiglia del dispositivo, e "ATSAM3X8E" come tipo di microcontrollore. Il file **SpeedTest.cpp** apparirà ora nella finestra Solution Explorer; si tratta semplicemente dello sketch che originariamente aveva l'estensione **.ino**.

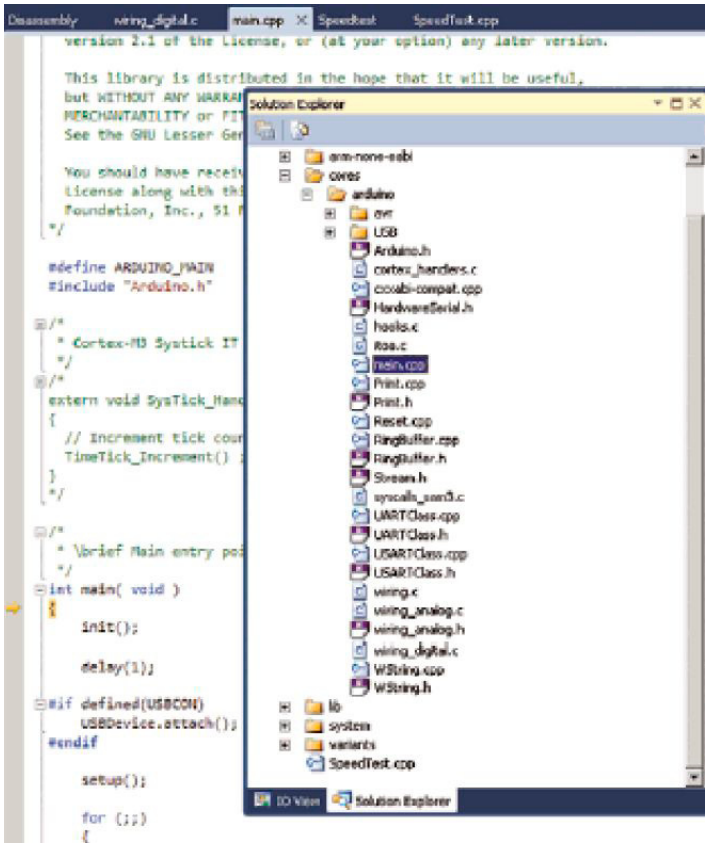


Eseguire ora un click col tasto destro sul nome del progetto "SpeedTest_Debug" in modo tale da aprire la finestra Properties. Quando l'ICE Atmel è collegato al PC tramite USB, il debugger può essere selezionato sotto Tool->Selected debugger/programmer. Qui occorre selezionare l'interfaccia "SWD" anzichè "JTAG".

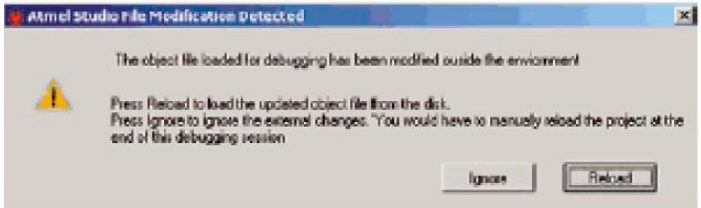
L'ICE Atmel a questo punto è pronto. E' possibile anche utilizzarlo come programmatore. Per aprire la finestra di programmazione, premere Ctrl-Shift-P oppure cliccare sul simbolo del chip con <lightning flash>: le funzioni saranno disponibili dopo aver premuto Apply. Nel tab Interface Settings è possibile impostare il massimo clock rate SWD (2 MHz), mentre nel tab Memory è possibile scaricare il file SpeedTest.elf direttamente nella memoria flash del microcontrollore. Il programma parte automaticamente non appena viene completata la procedura di programmazione.

Tuttavia, è più efficiente utilizzare i comandi Debug->Start Debugging and Break oppure Start Without Debugging per scaricare e debuggare o eseguire, rispettivamente, il programma. Il download dell'intero programma sul microcontrollore richiede con l'ICE Atmel circa 1 secondo, ed è quindi molto rapido.

La struttura del programma Arduino appena scaricato può essere visualizzata nella finestra Solution Explorer, dove è selezionato il file main.cpp e il cursore si trova posizionato sulla prima riga della procedura (**Figura 11**).



Come si può osservare il file **SpeedTest.cpp** rappresenta solo una piccola parte dell'intero programma, ed è questo il motivo per cui la compilazione richiede un certo tempo. Il debugger consente di accedere a tutte le funzionalità di Atmel Studio, come ad esempio il single step (F11), l'impostazione dei breakpoint (F9), la definizione di breakpoint condizionali (finestra Breakpoint), e così via. Se si modifica e ricompila il file **SpeedTest.ino** nella prima istanza di Atmel Studio, il messaggio mostrato nella **Figura 12** apparirà nella seconda istanza (l'ambiente di debug). Premendo il tasto "Reload" è possibile chiudere il messaggio di debug e caricare il programma modificato sul microcontrollore. Il cursore è ora nuovamente posizionato sulla prima linea di codice del programma, e il programma può essere eseguito nuovamente.



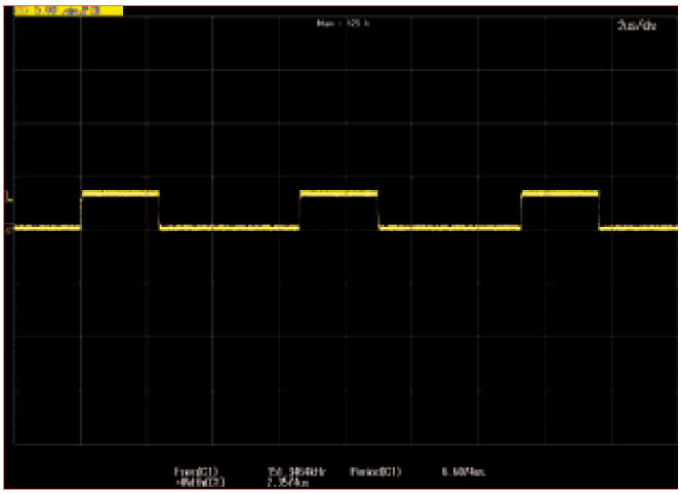
Se avete già lavorato con i microcontrollori AVR a 8-bit in ambiente Atmel Studio, non avrete alcuna difficoltà a lavorare con i microcontrollori ARM a 32-bit, in quanto l'interfaccia utente è praticamente la stessa. Si noti inoltre come la cancellazione e la scrittura della memoria flash con l'ICE Atmel non alterano il contenuto del bootloader , in quanto quest'ultimo risiede nella ROM del microcontrollore e non può pertanto essere cancellato o sovrascritto. Analogamente, il download di un programma via interfaccia USB si comporta nello stesso modo.

SPEEDTEST

Il main loop dello sketch di esempio (si veda il listato **SpeedTest**) comprende soltanto due istruzioni:

```
digitalWrite(TP1,1); //On
digitalWrite(TP1,0); //Off
```

Tramite un oscilloscopio, è possibile visualizzare un'onda quadra generata in corrispondenza di questo pin, con un'ampiezza di 2,35 us e periodo di 6,6 us (**Figura 13**), che corrisponde a una frequenza di circa 150 kHz.



Questa non è molto elevata, e potrebbe sembrare inutile utilizzare un microcontrollore ARM con clock di 84 MHz, quando un comune AVR a 8-bit potrebbe fare lo stesso o meglio.

Il motivo di queste performance piuttosto scarse è che l'istruzione Arduino **digitalWrite** si traduce in una moltitudine di chiamate a funzioni C, come può essere verificato eseguendo il programma in modalità single-step con il debugger (F11). Per ottenere delle frequenze più elevate, occorre aggirare questo inconvenieten. Fortunatamente, il linguaggio utilizzato da Arduino non è altro che una particolare forma di linguaggio C, utilizzato con un compilatore GNU configurato opportunamente per Arduino. Ciò significa che un pin configurato come uscita può essere indirizzato direttamente settando o resettando un bit nel registro PIO. L'istruzione:

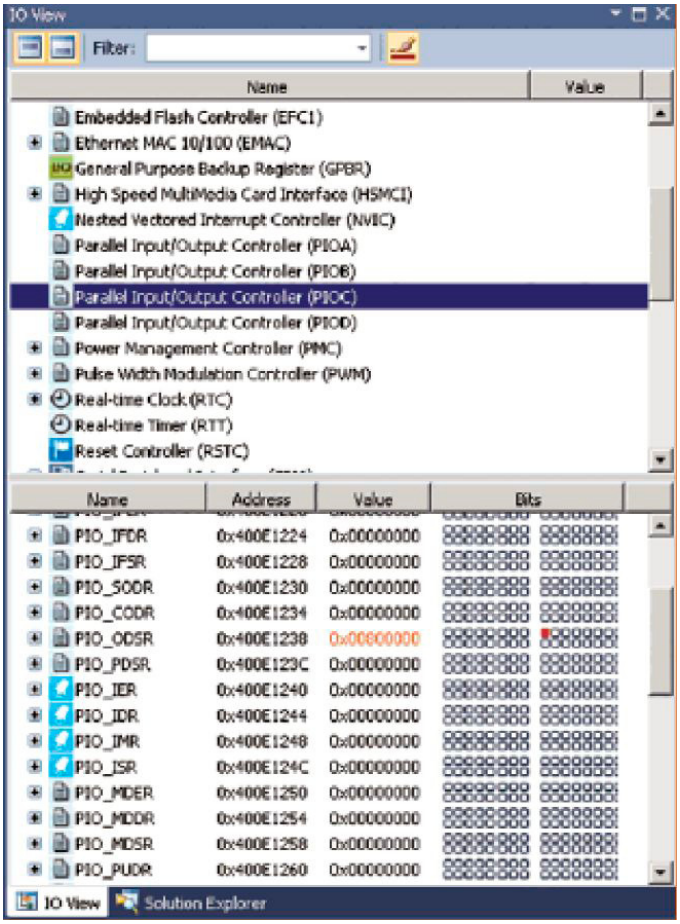
```
const int TP1 = 7; //Test pin
```

seleziona il pin 7 di Arduino come pin di test. In base alla mappatura dei pin utilizzata da Arduino, questo corrisponde al pin PC23 del microcontrollore, o al bit 23 del registro parallel input/output (PIO). Con il programma in stop, è possibile andare nella finestra I/O View del debugger (**Figura 14**) e cliccare il bit 23 del registro PIO_ODSR per impostare alto il pin. Per resettare il pin, è sufficiente cliccare sullo stesso bit del registro PIO_CODR. Nel main loop è possibile utilizzare l'istruzione

```
PIOC->PIO_SODR = 1<<23;
```

per settare il pin.

Per eseguire i test, e nello stesso tempo per verificare se la compilazione condizionale si comporta correttamente, l'autore ha generato una nuova versione dello sketch (si veda il listato **SpeedTest 2.0**). Se l'espressione Direct è stata definita, il segmento di codice successivo a



#ifdef Direct verrà compilato; in caso contrario, verrà compilata la versione precedente con l'istruzione lenta digitalWrite. Ovviamente, questa modifica funzionaerà anche nell'ambiente di sviluppo normale di Arduino.

Listato 1. Speed-Test.

```
//*******************************
//Speed-Test with Arduino Due
//*******************************
```

```
const int TP1 = 7; //Testpin
```

```
void setup()
{
 /* add setup code here */
 // set the digital pin as output:
 pinMode(TP1, OUTPUT);
}
```

```
void loop() {
 // put your main code here, to run repeatedly:
```

```
digitalWrite(TP1,1); //On
 digitalWrite(TP1,0); //Aus
}
```

Listato 2. Speed-Test 2.0.

```
//********************************
//Speed-Test 2.0 with Arduino Due
//********************************
#include "arduno.h"
//
//
const int LED1 = 13;
 int LED2 = 12;
 int LED3 = 11;
 int TP1 = 7; //Testpin
```

```
#define Direct
```

```
void setup()
{
 /* add setup code here */
 // set the digital pin as output:
 pinMode(LED1, OUTPUT);
 pinMode(LED2, OUTPUT);
 pinMode(LED3, OUTPUT);
 pinMode(TP1, OUTPUT);
 // set output low
 digitalWrite(LED1,0);
 digitalWrite(LED2,0);
 digitalWrite(LED3,0);
 digitalWrite(TP1,0);
}
```

```
void loop() {
 // put your main code here, to run repeatedly:
#ifdef Direct
//x= state of PIO_SODR with bit 23 = 1
int x = PIOC->PIO_SODR | 1<<23;
```

```
while(1){ //Only for testpurposes, don't exit this
loop
 PIOC->PIO_SODR = x;
 PIOC->PIO_CODR = x;
}
```

```
#else
 digitalWrite(TP1,1);
 digitalWrite(TP1,0);
```

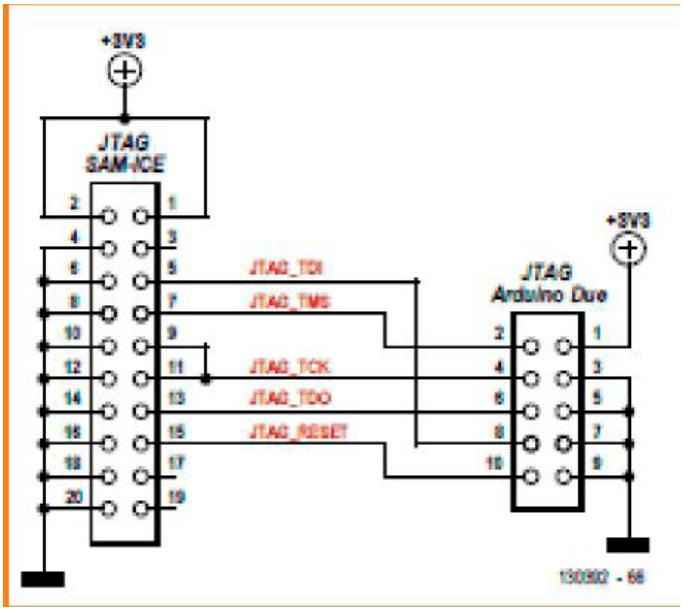
#endif

}

Il risultato di questa ottimizzazione del codice, come visibile in **Figura 15**, è impressionante: la forma d'onda ha ora un'ampiezza di soli 23,2 ns, e un periodo di 83,3 ns. Ciò equivale a una frequenza di 12 MHz - circa 80 volte maggiore rispetto alla versione precedente. Possiamo quindi concludere che l'utilizzo di un microcontrollore ARM non è stata una cattiva idea.

ADATTATORE PER SAM ICE

Questo adattatore è necessario per utilizzare il debugger/programmatore SAM ICE con Arduino Due. In pratica viene realizzato un collegamento tra il connettore JTAG a 10 pin della board Arduino Due con il connettore a 20 pin presente sul dispositivo SAM ICE.



SOMMARIO

L'utilizzo di Atmel Studio accelera notevolmente lo sviluppo di uno sketch per Arduino, grazie a tempi di compilazione molto ridotti (solo il codice effettivamente modificato viene ricompilato). Atmel Studio, insieme ai tool che lo accompagnano (come il debugger e programmatore Atmel ICE), sono molto utili quando si lavora con i microcontrollori Atmel, in quanto offrono delle funzionalità di debugging professionali e velocizzano i tempi di download del programma sulla memoria del microcontrollore.

Inoltre, per lo sviluppo di progetti ARM, è possibile utilizzare il SAM ICE di Atmel al posto dell'Atmel ICE. Con il programmatore e debugger hardware SAM ICE, la velocità di programmazione del microcontrollore può essere velocizzata fino a quattro volte, in quanto esso può operare alla frequenza di 8 Mhz anzichè 2 MHz. Per sfruttare questa opzione, occorre utilizzare l'adattatore descritto in precedenza. Se siete degli sviluppatori software di professione, l'investimento si ripagherà velocemente, anche se il dispositivo può essere utilizzato soltanto con la famiglia di microcontrollori AT91xx.

LINK

[1] Atmel Studio 6.2:
www.atmel.com/microsite/atmel_studio6
[2] Arduino IDE for Visual Studio:
www.visualmicro.com
[3] Atmel ICE:
www.atmel.com/tools/atatmel-ice.aspx
Review: www.elektor-magazine.com/140275
[4] SAM ICE:
www.atmel.com/tools/atmelsam-ice.aspx

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/sviluppo-software-per-arduino-con-atmel-studio

Programmare Arduino UNO con Atmel Studio

di Ernesto Sorrentino

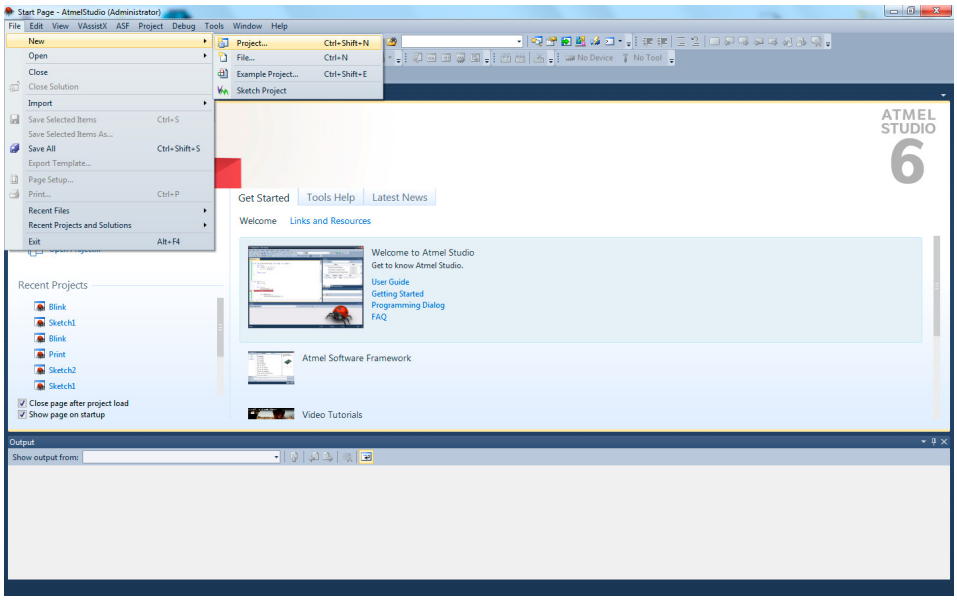
Atmel Studio 6 è una piattaforma di sviluppo integrato (**IDP**) per lo sviluppo e il debug di microcontroller (**MCU**) AVR di Atmel. Questo software fornisce un ambiente unico e facile da utilizzare per scrivere, costruire ed eseguire il debug delle applicazioni scritte in **C / C ++** o codice **Assembly**. Al contrario dell'IDE di Arduino, basato sul linguaggio **Wiring** (un progetto di Hernando Barragàn), che facilita il programmatore offrendo un modo semplice per accedere alle periferiche di input/output della piattaforma hardware, in Atmel Studio si è costretti a programmare i dispositivi a livello di registri, il che implica una buona conoscenza dell'architettura del microcontrollore che si vuole utilizzare; in pratica bisogna scrivere il programma nella stessa modalità in cui Wiring svolge la compilazione di uno sketch a nostra insaputa. Questo metodo può risultare macchinoso e poco interpretabile ma si ha il vantaggio di avere il pieno controllo del micro sfruttando tutta la sua potenzialità. In alternativa, per chi ritiene difficile questo tipo di programmazione ma vorrebbe utilizzare ugualmente questo tool di sviluppo per usufruire delle funzioni come il debugger, il sistema di autocompletamento, l'integrazione di plugin di terze parti e altro ancora, potrà configurare la piattaforma per integrare le librerie dell'IDE di Arduino e programmare in "**Wiring**".

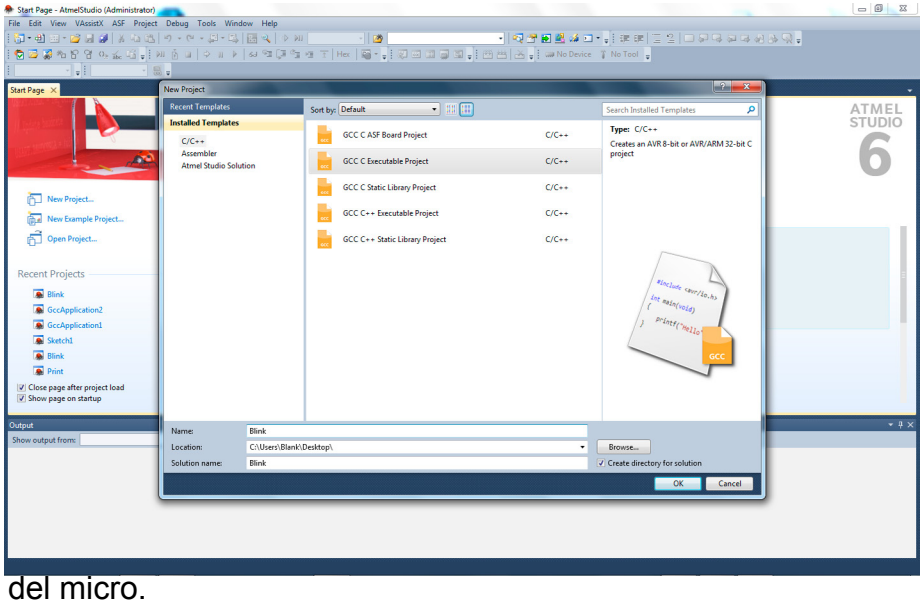
Atmel Studio è disponibile sul sito del produttore, con **licenza freeware** (versione attuale 6.2). Quest'ambiente di sviluppo non è leggero come quello di Arduino, l'installazione occupa oltre 1Gb di spazio su disco; perché si porta dietro un **framework** per lo sviluppo di applicazioni alquanto corposo, oltre alla **toolchain di AVR-GCC** e vari **datasheet**.

PROGRAMMAZIONE IN ASSEMBLY

Nell'esempio seguente vedremo come far lampeggiare il led posto sulla board di Arduino UNO con uno sketch scritto in Assembly.

Il linguaggio Assembly, detto anche assemblativo, è il più vicino a quella"macchina" vera e proprio. Per motivi hardware non esiste un unico linguaggio Assembly ma comunque molti meccanismi sono analoghi o del tutto identici tra loro. Spesso il passaggio, tra marchi diversi, si limita all'apprendimento di nuovi codici mnemonici, nuove modalità d'indirizzamento e altre varie peculiarità dovute alla architettura hardware





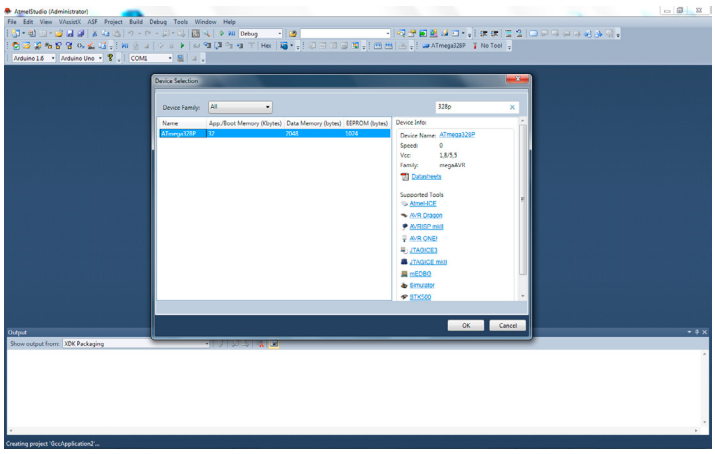
del micro.

Avviata l'applicazione, creiamo un nuovo progetto andando per "**file/nuovo progetto**".

Nella videata successiva bisogna scegliere il tipo di progetto da realizzare, impostare la destinazione di salvataggio dei file e il nome da assegnare. Il progetto da scegliere è "**GCC C executable project**" e come nome ho scelto "**Blink**"; confermiamo con "**OK**".

Di seguito bisogna selezionare il modello di microcontrollore in utilizzo, per la board Arduino UNO è l'**Atmega328P**. Da notare, sulla destra della finestra, il supporto tecnico ai Tool e il datasheet del micro in esame; ottimi strumenti per aiutare l'utente nello sviluppo del progetto.

Ora possiamo scrivere il nostro programma, lo



script che useremo è il seguente:

```
* Name: Blink.c
* Created: 02/04/2015
* Author: Sorrentino Ernesto
*/

#define F_CPU 16000000L

#include <util/delay.h>
#include <avr/io.h>

int main(void){
 DDRB=0x20;

 while(1){
  PORTB=0x20;
  _delay_ms(500);
  PORTB=0;
  _delay_ms(500);
  }
 }
```

La struttura base di un script assembly è composto principalmente da tre parti:

1) configurazione del micro e importazioni librerie:

```
#define F_CPU 16000000L
#include <util/delay.h>;
#include <avr/io.h>;
```

#define F_CPU 16000000L definisce il clock del micro di 16Mhz;

#include <util/delay.h> include nello script la libreria per la funzione delay;

#include <avr/io.h> include la libreria per la gestione delle periferiche In/Out.

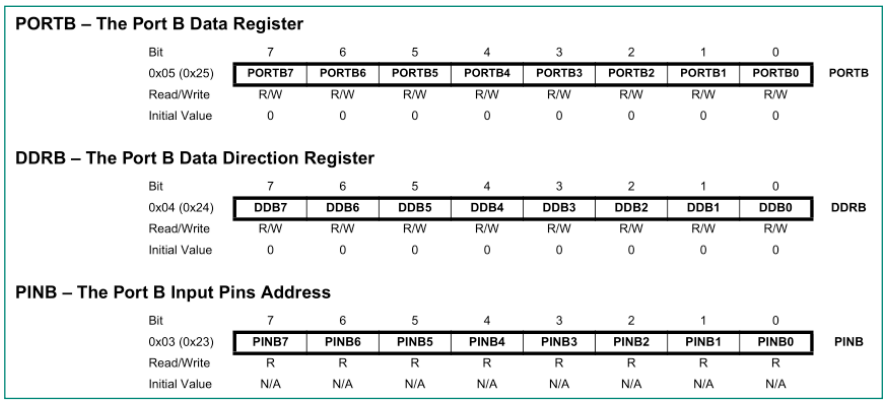
2) Settaggio dei registri di configurazione dispositivo; quello che nell'IDE di Arduino corrisponde al "**void setup()**".

Devono essere inseriti sotto la funzione "**main()**"

```
int main(void){
 DDRB=0x20;
```

Le porte del micro sono gestite tramite tre registri: DDRx, PORTx e PINx (dove "x" corrisponde al gruppo di port "B", "C" , "D", ecc…)

DDR: Registro per configurare il pin del micro come ingresso o come uscita (1 = output, 0 = input);
PORT: Registro per settare lo stato del pin impostato come uscita (1 = alto, 0 = basso);
PIN: Registro per la lettura di stato della porta.



PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

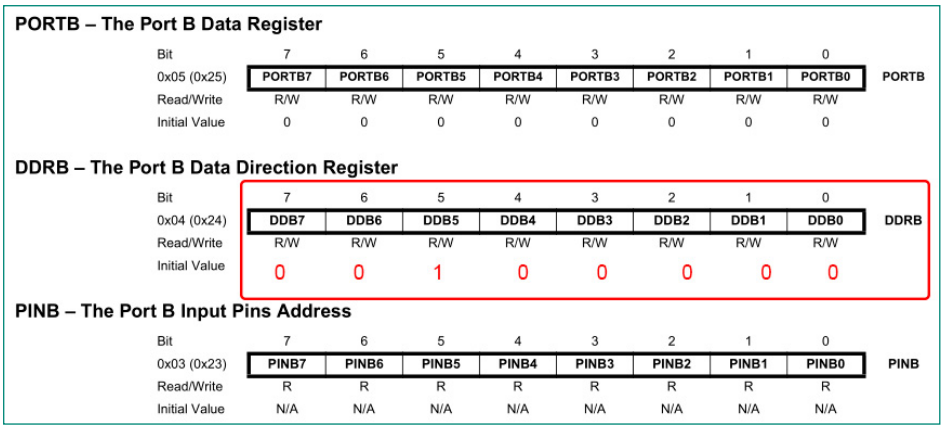
DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

In questo caso impostiamo la **PORTB 5** come uscita e le altre come ingresso settando il registro **DDRB** col valore **0x20** (in binario corrisponde a **00100000**)



PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

3) Ciclo del programma, da scrivere sotto la funzione "**while(1)**" (1 = condizione sempre vera). Questa corrisponde alla funzione "**loop()**" dell'IDE di Arduino.

```
while(1){
 PORTB=0x20;
 _delay_ms(500);
 PORTB=0;
 _delay_ms(500);
 }
}
```

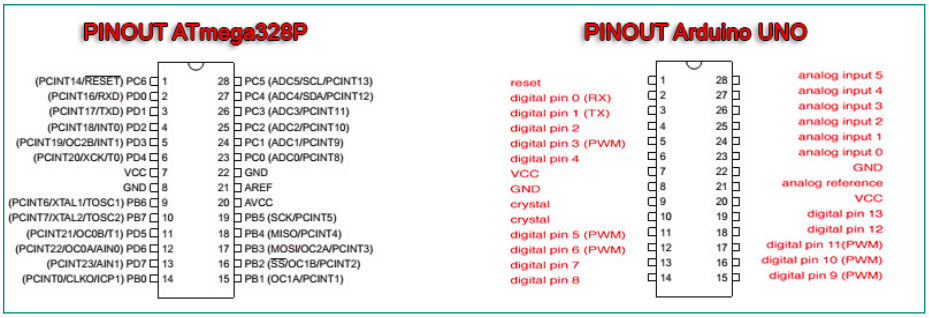
Come visto in precedenza il registro **PORTB** è utilizzato per assegnare lo stato di uscita della porta B. Inserendo il valore **0x20** (**binario corrisponde a 00100000**) si porta al livello alto solo la porta **B5** (**corrispondente al pin 19 del micro e al D13 di Arduino**) accendendo così il led.

L'istruzione "**_delay_ms (500)**" effettua una "chiamata" alla libreria "**util/delay.h**" che a sua volta genera un ritardo di mezzo secondo. Poi sarà spento il led portando al livello basso tutte le porte B con il comando "**PORTB = 0**"; di nuovo un'attesa di mezzo secondo e ricomincia il ciclo.
Una volta scritto il programma bisogna compilare, o meglio assemblare, le istruzioni col comando "**BUILD SOLUTION**" dal menu "**Bulid\Bulid Solution**" o premendo il pulsante "**F7**" della tastiera.
Dopo la compilazione, il tool produrrà una serie di file nella cartella "**Debug**" del vostro progetto, tra questi troviamo:

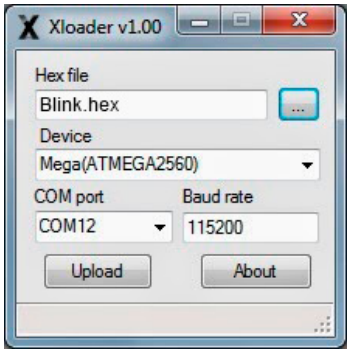
- **Blink.eep**: un file EEPROM Data, contiene informazioni da scrivere su EEPROM in formato hex e nel nostro caso è sostanzialmente vuoto;
- **Blink.elf**: Executable and Linkable Format

(Formato eseguibile e collegabile), contiene informazioni per il debug;

- **Blink.hex**: è il nostro file di "codice" in formato hex che utilizziamo per "flashare" il controller;
- **Blink.lss**: file disassembly del nostro programma intercalato con le linee C del progetto;
- **Blink.map**: mappa relativa alla posizione e grandezza delle sezioni di codice e dati del programma.



Per caricare il programma su Arduino possiamo utilizzare **Xloader**, **una applicazione gratuita**.



Il settaggio è semplice, basta selezionare il file .hex dalla cartella "Debug", il tipo di dispositivo, la porta di comunicazione e cliccare su "Upload" e vedere lampeggiare il led sulla board. **In alternativa è possibile caricare il programma tramite Atmel Studio utilizzando "avrdude"**, impostandolo come strumento esterno. Per questa configurazione è necessario che l'IDE di Arduino sia installata sul PC in uso.

- Aprire il menu di settaggio da "**Tools/ External Tools**";
- Mettere la spunta su "**Use Output window**";
- In "**Title**" inserire il nome da assegnare all'applicazione, tipo "**AVRISP mkII**";
- In Command inserire il percorso completo ad avrdude.exe. Nel mio caso la stringa è
- C:\ProgramFiles\arduino\hardware\tools\avr\bin\avrdude.exe;
- In Arguments inserire gli argomenti di avrdude -C"C:\Program Files\arduino\hardware\tools\avr\etc\avrdude.conf" -patmega328p -carduino -P\\.\COM20 -b115200 -Uflash:w:"$(ProjectDir)Debug\$(ItemFileName).hex":i

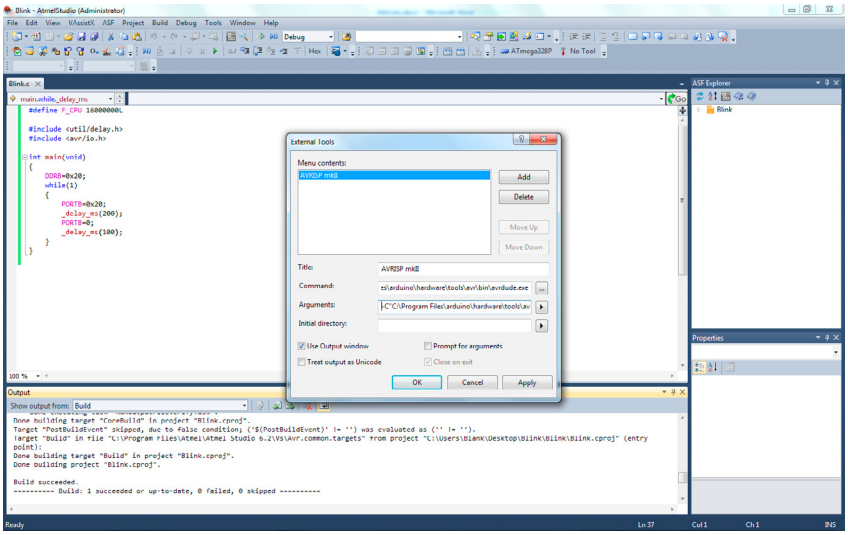
Nota: *Il parametro "-P\\.\COM20" deve riflettere la porta COM utilizzata dall'Arduino e per tanto potrebbe cambiare. Se il percorso alle directory contiene spazi (come in "Program Files (x86)"), potrebbe essere necessario inserire l'intero percorso tra virgolette; ad esempio : -C "C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf".*

Per dispositivi diversi dalla board UNO apportare le seguenti modifiche:

- Arduino Pro Mini: patmega328p - carduino - P.COM20 -b57600
- Arduino Duemilanove: patmega328p - carduino - P.COM20 -b57600
- Arduino Mega2560: patmega2560 - cwiring - P.COM3 - b115200

Riferimenti:

- **p**: Nome del microcontrollore;
- **c**: Nome della board;
- **b**: Baudrate di programmazione.



Ultimato il settaggio confermare col pulsante "**OK**" per salvare. Ora dal menù "**Tools**" sarà presente l'applicativo appena creato, cliccare per caricare il programma su Arduino.

PROGRAMMAZIONE IN C CON LE LIBRERIE DI ARDUINO

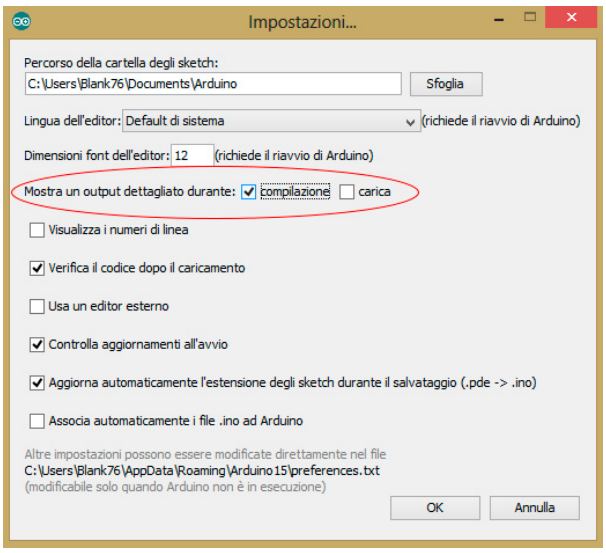
Con un'opportuna configurazione è possibile integrare all'ambiente di sviluppo Atmel le librerie di Arduino e programmare in modo "**Wiring**". Per ottenere questa modalità è necessario importare le librerie di Arduino in Atmel Studio con i seguenti passi:

- Creare una directory, nominandola "**ArduinoCore**" in C:\Users\xxx\Documents\Atmel Studio
- Copiare, in questa cartella, tutti file presenti in "C:\Program Files (x86)\Arduino\hardware\arduino\cores\arduino"

La libreria "**cores**" è un insieme di file necessarie alla creazione di uno sketch. In essa sono contenuti il file **main.cpp** (che include la funzione setup(), loop() e init() per inizializzare il micro) e tutte le librerie di wiring come la classe base **print.cpp** per le funzioni print() e il **pins_arduino.c** per le informazioni hardware dei pin della board in esame.

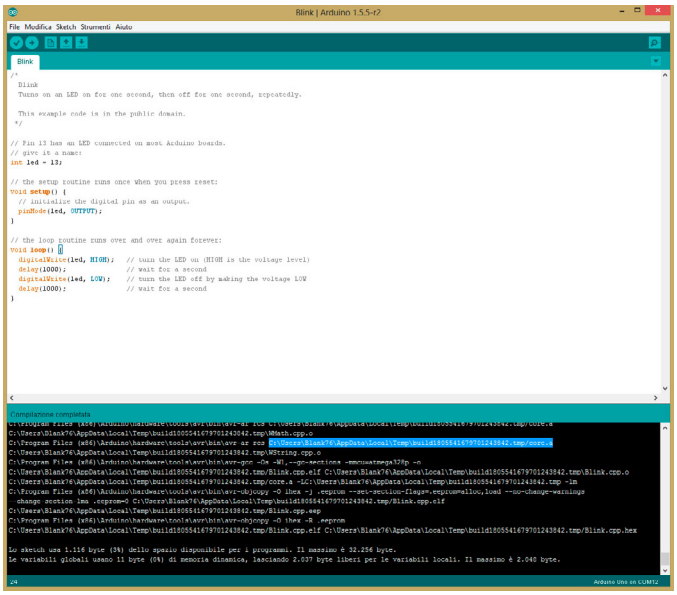
Ultimo file da aggiungere alla directory è il **linker core** nominato "**core.a**". "**Core.a**" è un archivio generato dall'IDE di Arduino, in fase di compilazione, che riunisce tutte le librerie per il funzionamento e la programmazione del micro. Ogni qualvolta che si ricompila lo sketch viene rigenerato ugualmente, il valore di quest'archivio cambia solo se si modificano alcuni parametri, come la frequenza di clock o la tipologia di microcontrollore. Per tanto si può definire che "**core.a**" **è associabile ad un solo tipo di board**.

- Aprire l'IDE di Arduino e caricare un qualunque esempio; **anche il "blink" va bene**;
- In "**File/Impostazioni**" mettere la spunta su "**compilazione**" per output dettagliato, chiudere con "OK";



- Compilare lo sketch e nella finestra di output localizzare il percorso del file "**core.a**";
- Copiare il file nella cartella "**ArduinoCore**" e rinominarlo in "**libcore.a**", **ricordare che questo file è utilizzabile solo per la**

board Arduino UNO.



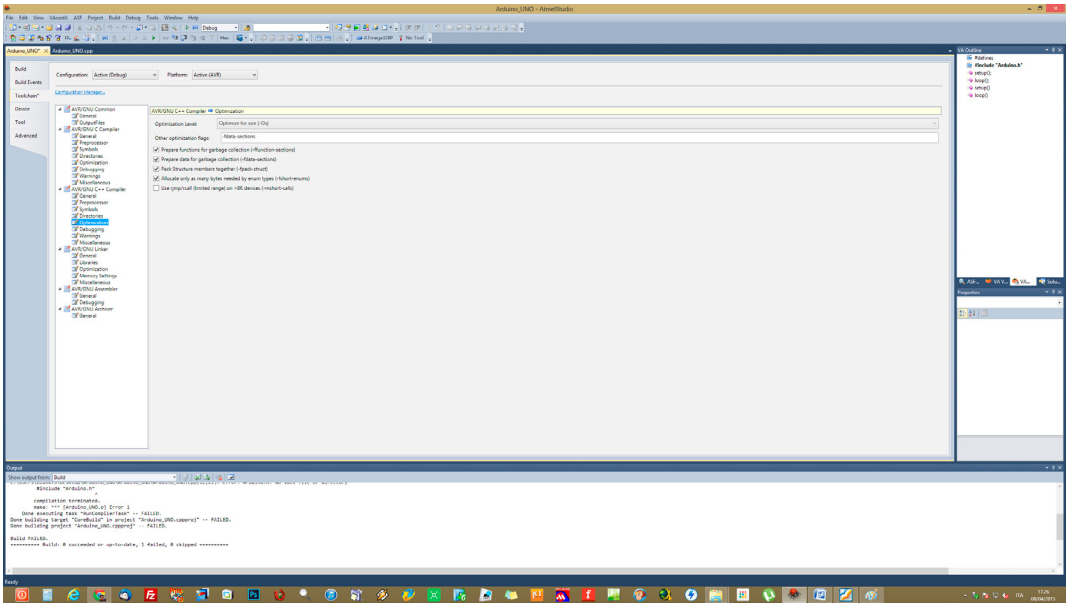
Ora resta la preparazione dell'ambiente di sviluppo Atmel per ospitare i progetti Arduino.

- Avviare Atmel Studio;
- Creare un nuovo progetto "**AVRGCC C++ Executable Project**". In "**Name**" inserire il nome del progetto (Arduino_Uno) e in "**location**" la destinazione di salvataggio;
- Al passaggio successivo bisogna selezionare il microcontrollore in uso, quello della board UNO è l'**Atmega328P**, e confermare con "**OK**";
- Premere "**ALT+F7**" per aprire le proprietà e selezionare il tab "**Toolchain**";
- In "**AVR/GNU C++ Compiler**" selezionare "**Directories**";
- In "**Include Paths (-I)**" inserire il percorso a "ArduinoCore" e togliere la spunta a "**relative Path**";
- Selezionare "**Optimization**" e in "Optimization Level" scegliere la voce "**Optimize for size (-Os)**. "-Os" ottimizza per la minore occupazione di memoria. Il compilatore non si cura a ottimizzare le prestazioni di esecuzione del codice generato. E' l'impostazione predefinita dell'IDE di Arduino;
- Inserire in "**Other optimization flags**" **-fdata-sections**;
- Mettere la spunta su "**Prepare functions for grabage collections (-ffunction-sections)**"

  Le due opzioni precedenti lavorano associate all'opzione "**–gc-sections**" del linker. In pratica, in fase di compilazione, ogni funzione viene posta in sezioni separate di memoria interna. In fase di linking le sezioni (dati o funzioni) che non sono mai referenziate vengono scaricate dalla memoria.
- In "**AVR/GNU Linker**" selezionare "**Libraries**";
- aggiungere in "**Libraries(-wl, -l)**" il nome associato al linker core di Arduino, in questo caso è "**libcore**";
- In "**Library search path (-wl, -L)**" inserire il percorso a "ArduinoCore" e togliere la spunta a "**relative Path**";
- In "**Miscellaneous**" inserire "-Wl,-lcore" in "**Other Linker Flags**".
- Salvare il tutto e ritornare alla finestra "**Arduino_UNO.ccp**";

Sostituire tutto il contenuto con i seguenti comandi:

```
#define F_CPU 16000000L
#define ARDUINO 160

#include "Arduino.h"

void setup();
void loop();

void setup() {

}

void loop() {

}
```



Premere **"F7"** per compilare lo sketch e **"Tools/AVRISP mkII"** per programmare la board. Ora il led dovrebbe lampeggiare in modo differente dalla programmazione precedente.

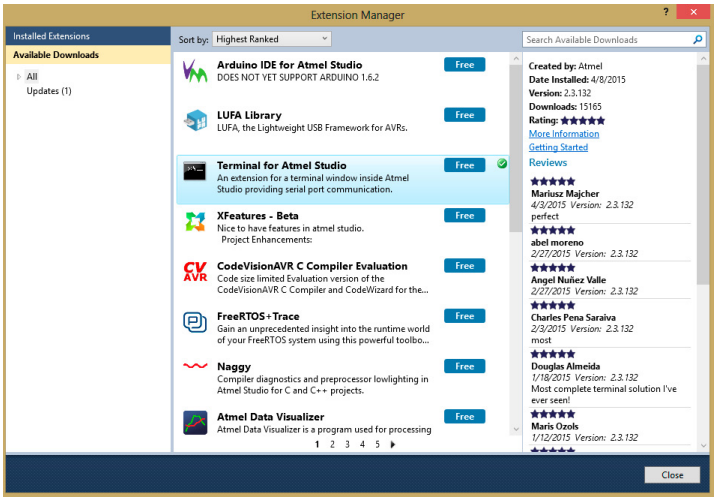
Questa è la configurazione di default per scrive un programma, dove:
#define F_CPU 16000000L specifica la frequenza di clock del micro (16Mhz);
#define ARDUINO 160 specifica l'uso della libreria riferita all'IDE di Arduino versione 1.6;
#include "Arduino.h" include la libreria Arduino;
void setup() richiamo della funzione setup();
void loop() richiamo della funzione loop();

le righe successive sono da strutturare come nell'IDE di Arduino per creare gli sketch.
Proviamo ora a scrivere il programma del blink:

```
#define F_CPU 16000000L
#define ARDUINO 160

#include "Arduino.h"

void setup();
void loop();

void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(100);
  digitalWrite(13, LOW);
  delay(500);
}
```

EXTENSION MANAGER

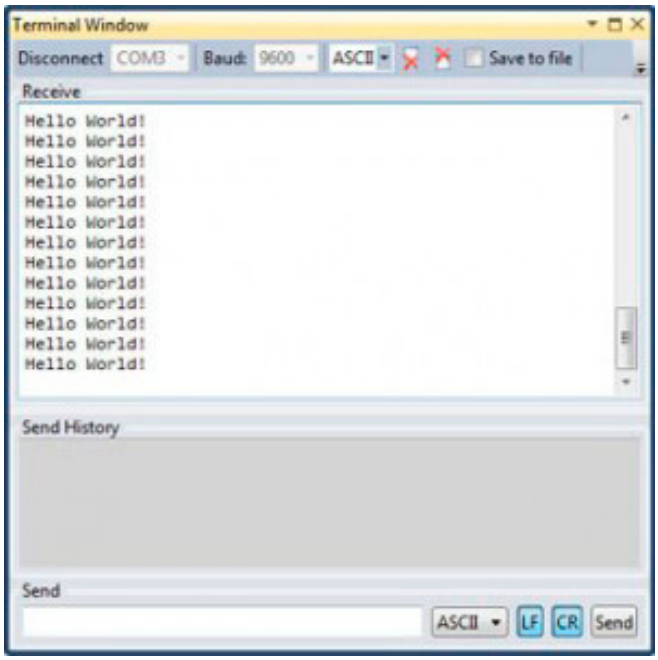
Una funzione particolare dell'Atmel Studio è l'integrazione dei **Tools di terze parti**. Uno in particolare è il **"Terminal for Atmel Studio"**; una sorte di "monitor seriale" come quella di Arduino. Per aggiungere questa applicazione selezionare **"Tools/Extension Manager"** e cliccare su **"Terminal for Atmel Studio"**.



Terminata l'installazione riavviare Atmel Studio e caricare il file "**Arduino_Uno.cpp**" salvato in precedenza. Ora modificheremo lo sketch aggiungendo il comando "**Serial**".

- In **"setup()"** aggiungere la riga:
  Serial.begin(9600);
- e in **"loop()"** la riga:
  Serial.println("Hello World!");

Premere **"F7"** per compilare lo sketch e "**Tools/AVRISP mkII**" per programmare la board.
Per avviare il monitor selezionare "**View/Terminal Window**"; settare la porta di comunicazione, il baudrate e cliccare su "**connect**". Il risultato sarà il seguente:



CONCLUSIONE

Non basta un articolo per descrivere tutte le potenzialità di questo magnifico ambiente di sviluppo. C'è molto da approfondire sulla funzione debug "I/O View" che monitorizza lo stato del processore, memoria e tutte le interfacce di comunicazione del microcontrollore; e molto altro ancora, che probabilmente vedremo in 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/programmare-arduino-uno-con-atmel-studio

Un encoder per Arduino

di Giovanni Carrera

Dopo aver illustrato i risultati di alcune prove di laboratorio, si passano in rassegna le varie tecniche hardware/software per l'eliminazione dei disturbi creati dal rimbalzo dei contatti e si studia lo schema di un efficace circuito antirimbalzo. Come possibile esempio di impostazione numerica, è presentato un programma con Arduino Uno che fa uso di due interrupt. **Le applicazioni di questo sensore sono molteplici, esempi sono alimentatori con controllo digitale, generatori di segnali, radio, orologi, etc**. Mi ricordo che già una ventina d'anni fa usavo un ottimo generatore di funzioni HP che aveva un encoder per impostare la frequenza. Molti apparati Hi-fi, radio e TV, oscilloscopi, sono ormai dotati di encoder per selezionare la frequenza o per variare il volume o altri parametri.

GLI ENCODER ROTATIVI

Gli encoder sono dei dispositivi digitali in grado di fornire informazioni sulla posizione o sulla velocità angolare o lineare. Data la loro natura digitale e la grande precisione, essi trovano larga applicazione nelle macchine a controllo numerico e in tutti i casi in cui occorra misurare spostamenti con grande precisione.

Gli encoder possono essere di due tipi:

- Encoder assoluti.
- Encoder incrementali.

Gli encoder assoluti danno in uscita un numero binario corrispondente alla posizione assoluta dell'asse (nel caso di encoder angolari) o dell'asta per quelli lineari.

La parte più importante dell'encoder angolare ottico è un disco realizzato con materiali a bassa dilatazione termica ed elevata precisione meccanica. In genere è realizzato in metallo o in vetro mediante tecniche di fotoincisione. Esso è suddiviso in un numero di corone che dipende dalla risoluzione angolare desiderata, in figura 1a è mostrato il disco di un encoder assoluto a 10 bit. Si divide il disco in tante zone concentriche quanti sono i bit di cui è composto il numero, alternando le zone opache a quelle trasparenti secondo un particolare codice binario, chiamato codice Gray. Questo particolare codice, passando da un numero a quello attiguo, permette la variazione di un solo bit alla volta, evitando possibili errori nella digitalizzazione della posizione. Con il codice binario naturale si ha un cambiamento di tutti i bit quando il prossimo numero è una potenza di due, così come avviene nel nostro codice decimale in corrispondenza di una potenza di 10.

Un pettine di fotocellule, disposte in senso radiale, permette di leggere il codice e inviarlo in uscita. Essi sono molto complessi perché richiedono un numero di fotocellule pari al numero di bit in uscita e una lavorazione meccanica a basse tolleranze, quindi è molto costoso.

Nella maggior parte dei casi si preferisce invece l'encoder incrementale, più semplice perché ha una o due fotocellule al massimo. Questo tipo di trasduttore non ha un riferimento assoluto ma relativo, per cui non può fornire da solo l'informazione di una posizione, ma richiede un'unità di conteggio (conta impulsi) in grado di registrare gli incrementi di posizione dati dall'encoder. Per una certa posizione, che costituisce il riferimento iniziale, si azzera il conta impulsi, quindi

si contano gli incrementi dati dal movimento della rotella dell'encoder.

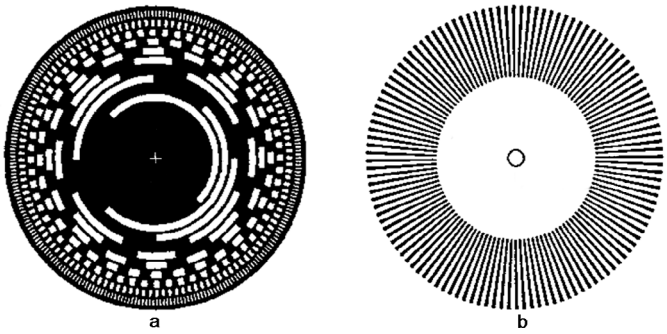


Figura 1: Disco di un encoder assoluto a 10 bit (a) , Disco di un encoder incrementale (b)

Questa è suddivisa in tanti settori chiaro/scuro quanti sono gli incrementi/giro richiesti, come mostrato dalla figura 1b. Per esempio, con 360 finestrelle si ottiene un impulso per grado. Una sola fotocellula va bene solo per moti unidirezionali, per avere anche il senso occorrono due fotocellule (encoder a due uscite) disposte in modo che gli impulsi siano sfasati di circa 90° tra loro, come mostra la figura 2. Osservando che, in corrispondenza del fronte di salita di A, B=0 se il verso è orario mentre B=1, se il verso è antiorario, si può determinare il verso del movimento con l'impiego di un semplice circuito basato su porte logiche e flip-flop. Il contatore deve essere del tipo avanti/indietro per conteggiare gli incrementi o i decrementi dalla posizione di riferimento. Questa logica può essere facilmente realizzata da un microcontrollore.

Gli encoder incrementali sono usati anche per rilevare velocità angolari con elevata precisione. Esistono anche encoder lineari, con caratteristiche del tutto simili a quanto visto per quelli angolari. Molte stampanti hanno un encoder lineare, realizzato con una sottile pellicola e una fotocellula, per rilevare la posizione della testina.

Gli encoder incrementali possono essere ottici, magnetici, elettromeccanici, capacitivi.

Questo tipo di trasduttore trova vaste applicazioni in tutti i dispositivi elettromeccanici controllati da microprocessori quale elemento di retroazione dell'asservimento di posizione. Per la loro maggiore semplicità costruttiva sono più usati gli encoder angolari rispetto a quelli lineari.

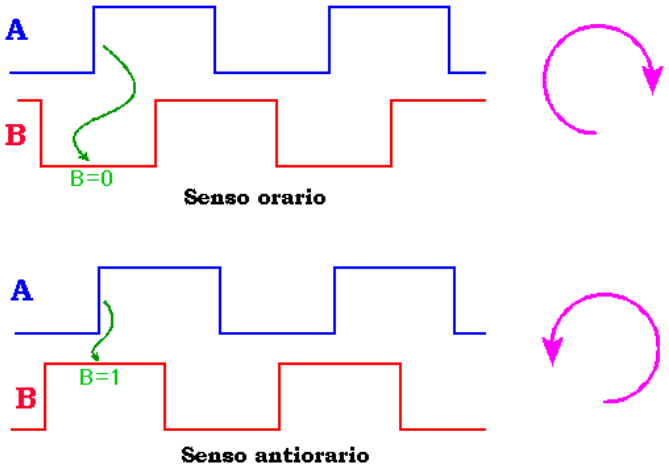


Figura 2: Impulsi per senso orario e antiorario

Gli encoder ottici, non avendo contatti striscianti, hanno una coppia minima e sono particolarmente robusti e relativamente semplici. Nei vecchi mouse a pallina, la rotazione della sferetta gommosa era trasferita, nelle componenti verticali e orizzontali, a due encoder incrementali ottici. Quelli con la rotellina avevano un terzo encoder collegato a essa, come hanno anche i moderni mouse a sensore ottico.

Oggi essi trovano applicazioni anche nelle apparecchiature elettroniche digitali, per impostazioni rapide o per sostituire i classici potenziometri analogici. Sono sul mercato a prezzi molto convenienti encoder meccanici, dove il disco è metallico e le fotocellule sono sostituite da contatti striscianti. Naturalmente hanno un limitato numero di finestrelle, tipicamente 12. La figura 3 mostra un tipico esemplare di encoder incre-

mentale, anche munito di pulsante assiale. Da un lato ci sono tre pin: quello centrale è la massa comune e gli altri sono i segnali A e B dell'encoder. Sul lato opposto ci sono altri due pin che sono collegati al pulsante che si chiude quando si preme la manopola.

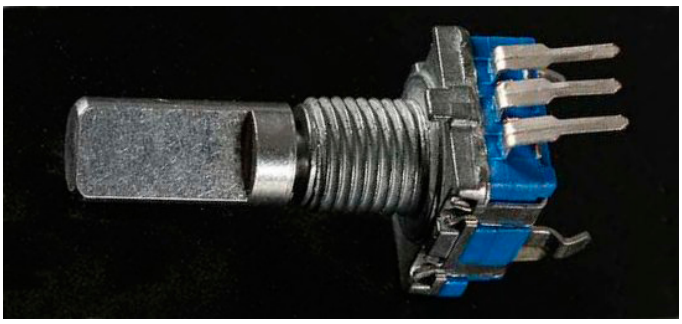


Figura 3: Encoder meccanico incrementale

LE PRIME PROVE

Dopo avere alimentato a 5V e inseriti due resistori di pull-up da 10k sui due segnali dell'encoder, sono state fatte numerose misure con un oscilloscopio digitale USB collegato ai due canali. I risultati sono mostrati nelle figure 4 e 5.

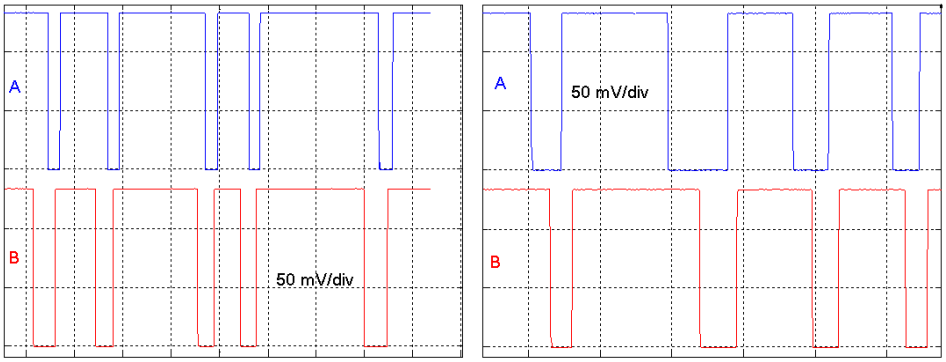


Figura 4: prime prove e segnali di test

In figura 4a si vede che sul fronte di salita di A, il segnale B è alto, mentre in senso orario B è basso. Gli impulsi non sono di eguale durata perché la velocità con cui ruotavo a mano il perno non era costante e forse anche perché la larghezza della parte conduttrice delle finestrelle è minore della parte isolante. Non si è voluto sacrificare l'encoder per una verifica diretta.

A questo punto è stato abbastanza facile scrivere un programma su Arduino per riconoscere il verso e incrementare o decrementare una variabile. Ma il funzionamento è stato molto incerto: gli incrementi erano irregolari e spesso non concordi.

Aumentando la frequenza di campionamento dell'oscilloscopio, si sono potuti notare i disturbi causati dallo strisciamento delle spazzole, presenti anche nella figura 4, ma non visibili data la bassa frequenza di campionamento. **Nella figura 5 si vedono chiaramente questi disturbi, per cui il programma non riusciva a funzionare correttamente**. Negli encoder ottici questi inconvenienti sono praticamente assenti.

TECHNICHE DI DEBOUNCE PER ELIMINARE I DISTURBI

I contatti striscianti, come abbiamo visto nelle prove pratiche, portano a rumore con conseguenti errori provocati dai falsi impulsi creati dal rimbalzo (*bounce*) dei contatti. Per evitare questo notevole disturbo si usano tecniche di *debounce* sia a livello hardware, con circuiti RC o con appositi circuiti integrati come il MC14490, sia intervenendo sul programma con ritardi.

La cosa migliore è quella di ridurre il disturbo già a livello del segnale, lo schema di figura 6 mostra una semplice soluzione circuitale di tipo RC. Inserendo un filtraggio RC sulle uscite dell'encoder, i fronti di salita e di discesa sono notevolmente meno ripidi, quindi occorre utilizzare uno *schmitt trigger* in uscita, per ottenere fronti ripidi e puliti.

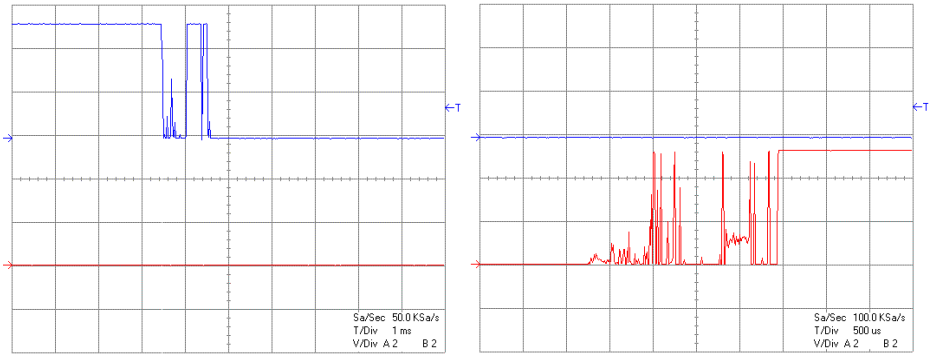


Figura 5: prime prove con visualizzazione dei disturbi dovuti allo strisciamento delle spazzole

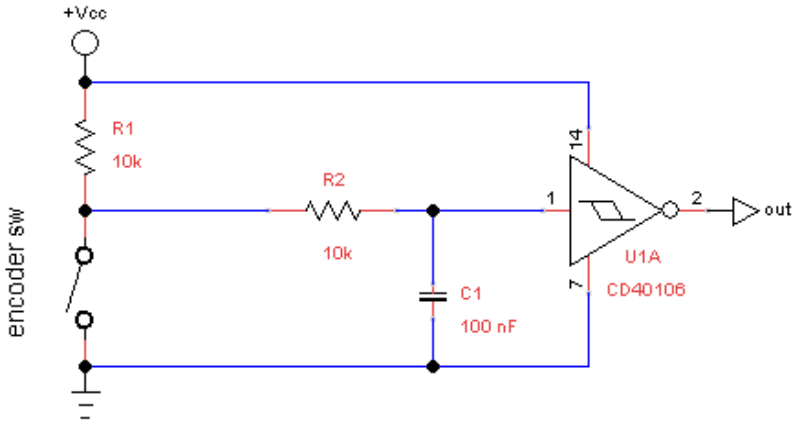


Figura 6: soluzione circuitale debounce di tipo RC

Come risulta dalla figura 6, gli impulsi di rimbalzo sono abbastanza sottili, hanno una durata molto inferiore al millisecondo. Questo ci permette di calcolare un filtraggio opportuno. Come si vede dallo schema, in fase di carica del condensatore C (contatto aperto) $R=R_1+R_2$= 20 kΩ, mentre in scarica R= 10 kΩ. Usando per C un condensatore da 0,1 uF si ha una costante di tempo di carica tc = 2*10^4×10^-7 = 2 ms mentre quella di scarica è ts = 1 ms. Sapendo che i valori delle soglie V_t del trigger CD40106, alimentato a V_{cc}= *5V*, sono di circa V_{td} = *1,4V* per la discesa e V_{ts}= *3,6V* per la salita, si possono calcolare i ritardi:

$$v(t) = Vcc\left(1 - e^{-t/\tau}\right) \rightarrow \quad t_s = \tau_c \cdot \ln\left(\frac{Vcc}{Vcc - V_{ts}}\right) \approx 2{,}5\ [ms] \qquad \text{ritardo in salita}$$

$$v(t) = Vcc \cdot e^{-t/\tau} \rightarrow \quad t_d = \tau_s \cdot \ln\left(\frac{Vcc}{V_{td}}\right) \approx 1{,}3\ [ms] \qquad \text{ritardo in discesa}$$

Queste formule sono quelle di carica e scarica di un condensatore con un resistore in serie.

Per un maggior effetto di filtraggio è meglio lavorare sui fronti di salita, ossia sulla carica del condensatore. Naturalmente, mantenendo l'inversione della porta logica, il software dovrà generare l'interrupt sui fronti di discesa, se usiamo due porte invertenti in cascata si dovrà operare sul fronte di salita.

Questi valori sono adatti per gli encoder elettromeccanici mossi a mano, come nel nostro caso. Per applicazioni più veloci è meglio ridurre la capacità. Con 2,5 ms di ritardo, aumentando la velocità, diminuisce il periodo dei segnali. Se il semiperiodo è uguale o superiore ad esso, il segnale non ritorna a zero per cui l'encoder non è leggibile. Per calcolare la velocità limite, considerando 12 posizioni, si ottiene il periodo corrispondente ad un giro: T = 24*2,5 = 60 ms. La velocità è pari a 1000/60 = 16,7 giri/secondo, molto più alta di quella manuale. Va comunque detto che questo dispositivo non è adatto per movimenti continui e veloci, per cui sono più indicati gli encoder ottici o magnetici.

Come vedremo, la soluzione di inserire questo circuito è quasi d'obbligo nel caso si lavori con interrupt. Con questo accorgimento, il programma ha funzionato perfettamente.

IL CIRCUITO ANTI-RIMBALZO DELL'ENCODER

La soluzione adottata è stata quella di realizzare una piccola scheda da montare direttamente sull'encoder con tre circuiti anti-rimbalzo, due per l'encoder e uno per il pulsante incorpora-

to. Sfruttando tutte le porte dell'integrato U1, le uscite non sono invertite, quindi occorre che gli interrupt dell'encoder siano sul fronte di salita e non di discesa, questo per avere ritardi maggiori.

La figura 7 mostra lo schema completo della scheda di interfaccia. Al posto del CD40106, possono essere usati anche un 74HC14 o un 74C14.

Molti più rimbalzi sono stati rilevati sul pulsante rispetto ai segnali A e B, se si rilevasse un funzionamento incerto causato da questi disturbi, basterebbe aumentare la capacità di C4 a 220nF. Comunque non sono stati rilevati questi problemi sul prototipo.

Elenco componenti:

R1-R6	Resistore 10 kΩ 1/4W
C1-C4	Condensatore 0,1 MF 25V ceramico
Encoder	Vedere testo
U1	CMOS Hex schmitt trigger CD40106, 74HC14
J1	Pettine 5 poli passo 0,1"

In figura 8 si vede la realizzazione del prototipo su scheda millefori. Per avere una bassa altezza dei componenti si è usato un 74AC14 con case smd. **La soluzione ottimale sarebbe quella di usare una millefori a doppia faccia o realizzare un piccolo circuito stampato con componenti smd**. In figura è stata rimossa la manopola per vedere meglio il *layout* dei componenti.

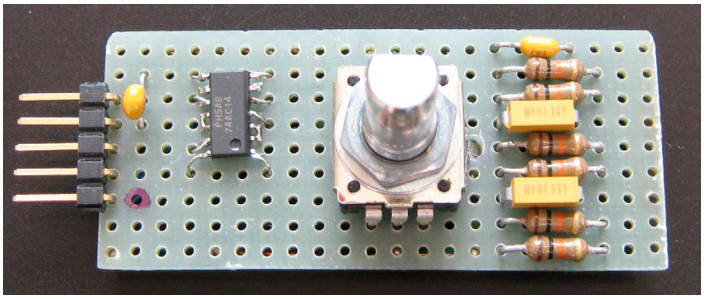


Figura 8: prototipo su scheda millefori

ALGORITMI DI ELABORAZIONE DEGLI ENCODER INCREMENTALI

Se il numero di finestre è elevato, solitamente si lavora solo su un fronte degli impulsi di uno dei segnali, per quanto detto in precedenza, il fronte di salita del segnale A. Come evidenziato in figura 2, a ogni fronte di salita di A si va a leggere il valore che ha il segnale B, se esso è basso si incrementa un contatore in quanto si identifica un senso orario, se esso è alto, si decrementa il contatore. L'evento può essere rilevato via interrupt, soluzione migliore, oppure continuando a leggere il segnale A.

Utilizzando l'interrupt risulta difficile effettuare un filtraggio del segnale perché non si può usare la funzione delay(), che è più facile inserire in un programma che esegue un continuo polling sul segnale, come fanno molti programmi che si trovano in rete. Essi aspettano il cambiamento di stato e quindi effettuano un filtraggio digitale inserendo degli opportuni ritardi. Questa tecnica fa perdere molto tempo alla CPU che non può fare altre operazioni. L'interrupt è più vantaggioso in quanto lascia libera la CPU per altri impieghi e questo è un motivo molto valido per giustificare la scelta, per cui si è utilizzato il circuito anti-rimbalzo visto precedentemente. L'uso dell'interrupt permette anche di lavorare su encoder più veloci come quelli ottici, montati sull'asse di un motore.

Volendo raddoppiare la risoluzione si lavora sia sul fronte di salita, sia su quello di discesa di un segnale (interrupt sul cambiamento di stato, modo CHANGE) oppure sui fronti di discesa (o salita) di entrambi i segnali, ma si è preferito lavorare con una risoluzione inferiore e utilizzare il pulsante incorporato per avere la possibilità di incrementi maggiori. Per questo motivo si è dotato di circuiti anti-rimbalzo anche questo pul-

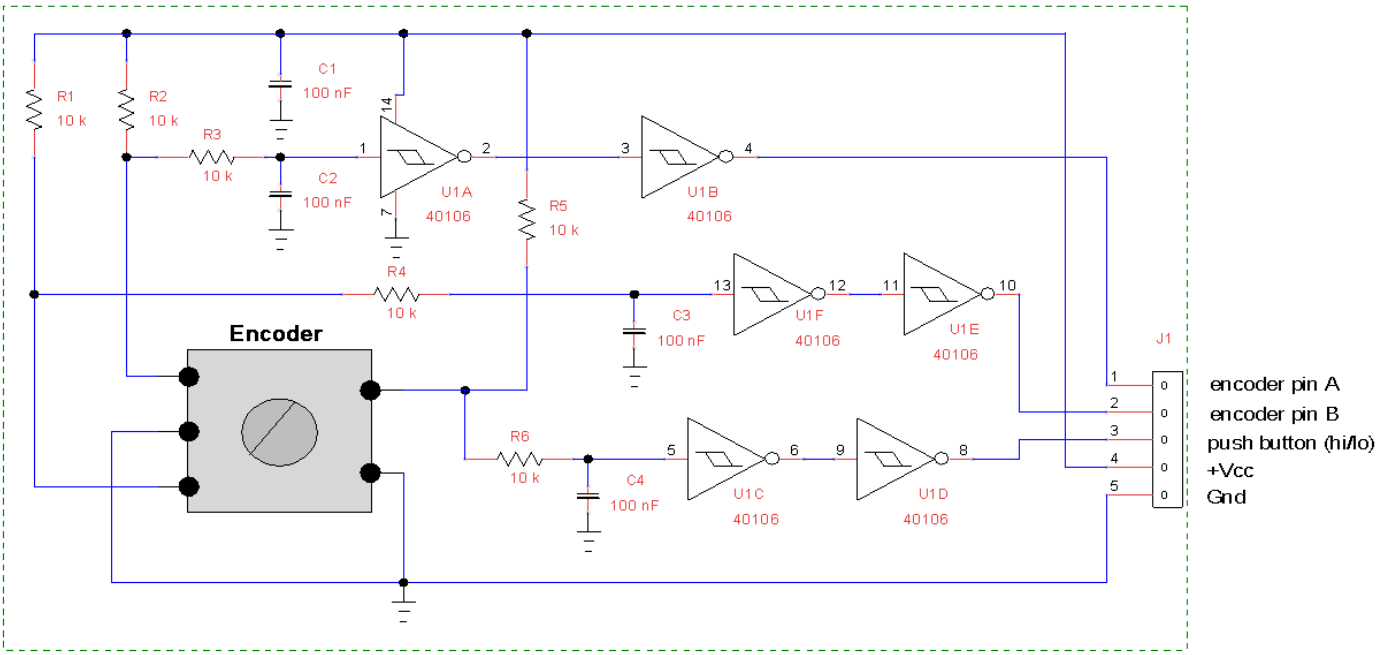


Figura 7: scheda di interfaccia

sante, in modo da generare un interrupt anche per questa funzione.

PROGRAMMA ARDUINO

Per provare il programma si è usato un Arduino Uno collegato con la scheda encoder con cinque fili, secondo la tabella seguente:

Encoder pin:	A	B	pulsante	+Vcc	Gnd
Arduino pin	2	4	3	5V	Gnd

Le uscite del programma sono visibili sul terminale seriale della IDE di Arduino.

Come già descritto, il programma utilizza i due soli bit disponibili per l'interrupt di Arduino Uno, per il fronte di salita del segnale A e per il pulsante incorporato nell'encoder. Ad esempio: premendo la manopola l'incremento passa da uno a dieci, premendo nuovamente passa a 100, poi a 1000, quindi ritorna a 1 con un ulteriore schiacciamento del pulsante. Con poche modifiche del programma si potrebbe iniziare ad impostare le cifre più significative, come si fa quando si scrive un numero. Il programma presentato è solo un esempio, è possibile usare il programma per impostare l'ora e la data su un orologio RTC o per numerose altre applicazioni. Queste operazioni risultano più agevoli con un display LCD dove si fanno lampeggiare le cifre interessate oppure usando appositi led.

Il programma usa due semplici ISR (Interrupt Service Routine) che si è preferito renderle più leggere possibili, come bisogna sempre fare. La gestione del contatore è fatta dal programma principale, che deve controllare, per quanto riguarda l'encoder, solo se il flag delta è diverso da zero.

La routine *encint()* in corrispondenza del fronte di salita di A legge il segnale B e pone delta=1 se B=0 (verso orario), oppure delta = -1 se B = 1. La variabile delta funge anche da *flag* di interrupt, se non avviene niente, questa rimane zero. Essa è azzerata dal programma loop, dopo aver elaborato l'incremento.

Se, per qualsiasi motivo, girando la manopola in senso orario il contatore decrementa invece di incrementare il conteggio, basta invertire i segni di *delta*.

```
/* test program encoder.ino
  read a rotary encoder with push button and debounce circuit
  uses interrpts on A and button
  write the counts to serial monitor
  Giovanni Carrera, rev. 16/10/2015
*/

#define encoder_A  2
#define encoder_B  4
#define encoder_pb  3

volatile signed int count = 5000; // initialize encoder counter
volatile signed int delta = 0; // increment
volatile unsigned int multi = 0; // multiplier index
const int multiplier[4] = {1,10,100,1000}; //increment multiplier

void setup() {
  pinMode(encoder_A, INPUT);
  pinMode(encoder_B, INPUT);
  pinMode(encoder_pb, INPUT);
  attachInterrupt(digitalPinToInterrupt(encoder_A), encint, RISING); // interrupt on  _- A
  attachInterrupt(digitalPinToInterrupt(encoder_pb), buttint, RISING); // interrupt on button pressing
  Serial.begin (9600);
  Serial.println("Encoder test");
}

void loop(){
  if (delta !=0){ //an encoder interrupt occurred
```

```
  count= count+delta*multiplier[multi];// increment or decrement counter
  delta= 0;// resets the flag
  if (count > 10000 || count < 0) count=0;// reset on overrange
  Serial.println(count);
  }
}

void encint() {
  /* encoder interrupt, see the status of B signal
   in order to decide the sense
   */
  if (digitalRead(encoder_B) == 0) delta=1;
  else delta= -1;
}
void buttint() {
  // button interrupt, change multiplier
  multi++;
  if (multi==4) multi=0;// back to top
}
```

La routine *buttint()* è richiamata ogni volta che è premuto il pulsante, essa incrementa l'indice multi della tavola multiplier dei moltiplicatori dell'incremento. Se l'indice supera il numero di elementi della tavola, esso è azzerato. Una logica opposta va applicata nel caso si voglia iniziare dalle cifre più significative.

Il programma principale (loop), appena trova delta diversa da zero incrementa la variabile *count* con l'incremento delta*multiplier[multi], quindi, azzera *delta*, come interrupt flag.

La variabile *count* può essere inizializzata, nel programma è posta a 5000, e il suo valore è pure controllato e azzerato nel caso siano superati i limiti imposti.

Questo sketch vuole essere solo un esempio di utilizzazione, ovviamente il programma principale può essere completato dalla parte che uti-

lizzerà il numero impostato.
Per rendere il programma leggibile a tutti, i commenti sono stati scritti in inglese.

Bibliografia

- Dispense di “Automazione navale” ,Capitolo 7, Carrera.
- “Encoders Signal Conditioning Technical Note”, Bourns, 2010

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-encoder-per-arduino

SMAs: gestiamo questi materiali "intelligenti" con Arduino

di Vincenzo Motta

LA STORIA

Gli SMAs sono delle leghe formate dall'unione di Nichel+Titanio, vengono solitamente chiamati "***Nitinol***" , "*Ni*" per *Nickel*, "*ti*" per *Titanium* e "*nol*" da *Naval Ordnance Laboratory*. La capacità di *memoria di forma* degli SMAs fu scoperta nel 1961 da *William J. Buehler*, un ricercatore del *Naval Ordnance Laboratory in White Oak, Maryland*. La scoperta avvenne per caso. Durante una riunione di laboratorio, fu presentata una striscia di Nitinol che aveva subito diverse variazioni della sua forma, uno dei presenti, il *Dr. David S. Muzzey*, riscaldandola con la sua pipa si accorse che la striscia di Nitinol ritornò alla sua forma originale. Da quel momento fino ai giorni nostri, gli SMAs hanno avuto un notevole successo e un largo spettro di applicazioni in ambito scientifico e non solo.

STRUTTURA CRISTALLINA

La maggior parte dei solidi ha un'unica struttura cristallina, ma il NiTi ne ha due! Il passaggio dall'una all'altra costituisce una vera e propria transizione di fase allo stato solido, analoga in molti aspetti alla transizione solido-liquido. Come in quest'ultimo caso, infatti, il passaggio tra due stati della materia è indotto dalla variazione di temperatura, dalla forza applicata, o da una combinazione dei due fattori.

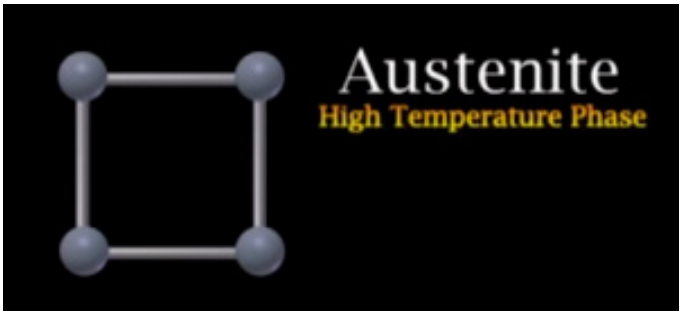
La caratteristica distintiva di una transizione di fase è il brusco cambiamento di una o più proprietà fisiche. Le proprietà macroscopiche di un materiale infatti dipendono fortemente dalla struttura cristallina, pertanto una sua seppur lieve modificazione può portare a materiali anche molto diversi tra loro.

Un esempio familiare è quello del Carbonio: un cristallo di Carbonio con la struttura tetraedrica (*ogni atomo è al centro di un tetraedro ai cui vertici siedono altri atomi*) costituisce il *diamante*. Ma il Carbonio può ugualmente presentarsi nella forma cristallina della *grafite*, in cui gli atomi si dispongono su strati debolmente legati tra loro. Grafite e diamante hanno la stessa composizione chimica essendo fatti di soli atomi di carbonio. La loro struttura cristallina è però profondamente diversa.

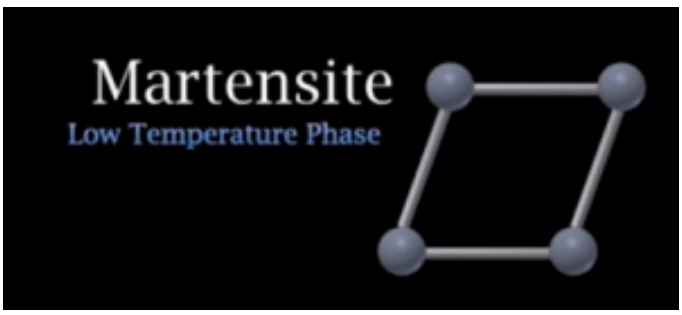
Ogni struttura è stabile in un determinato intervallo di pressione e temperatura. Tuttavia, in genere una volta che il cristallo si è formato con una specifica forma cristallina (per esempio, il diamante si forma solo a pressioni molto elevate) è molto difficile fargli cambiare forma modificando semplicemente pressione e temperatura. Nel caso del Nitinol, invece, si può indurre il passaggio tra le due strutture variando la temperatura in un intervallo facilmente realizzabile in laboratorio. Le forme cristalline del Nitinol sono due:

- **Austenite**: è la fase stabile ad alta temperatura. È caratterizzata da un reticolo a simmetria cubica a corpo centrato (BCC), in cui gli atomi cioè occupano i vertici di due reticoli cubici compenetrati. Il mate-

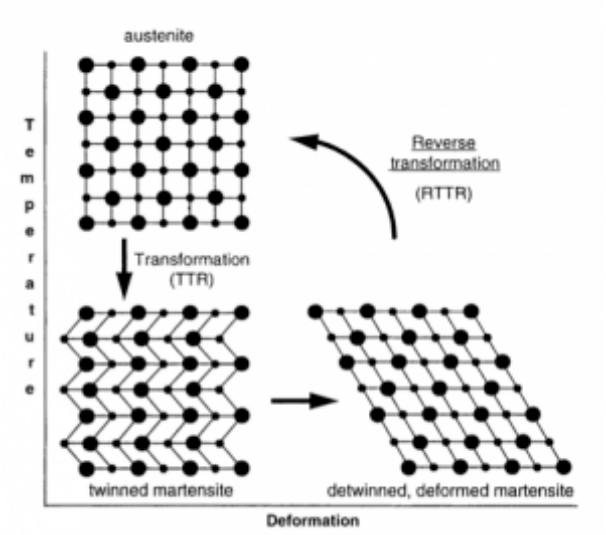
riale in questa fase è duro e difficilmente deformabile.



- **Martensite**: è la fase stabile a bassa temperatura. Ha una struttura cristallina monoclina distorta, molto meno simmetrica del tipo BCC. È caratterizzata da grande flessibilità e dalla capacità di essere facilmente deformata senza che tuttavia tale deformazione sia permanente, infatti, questa fase si forma e si accomoda in forma *twinned*, ovvero speculare rispetto ad un piano ideale tra due celle, non creando difetti irreversibili nel reticolo cristallino. Questa struttura a twins è pertanto facilmente deformabile, aprendosi nella forma *detwinned* come il mantice di una fisarmonica e, non avendo creato difetti di slittamento dei piani reticolari, è reversibile. Nella fase martensitica, perciò, il materiale sottoposto a uno sforzo meccanico è in grado di sopportare un alto grado di deformazione senza tuttavia rompere i legami chimici.



Poiché la trasformazione tra Austenite e Martensite è **reversibile**, alzando la temperatura il materiale esegue la trasformazione cristallina inversa e riprende la forma regolare e rigida della Austenite, indipendentemente dalla deformazione eventualmente subita nella fase Martensite. Si manifesta cioè l'**effetto di memoria di forma SME (Shape Memory Effect)**.



SUPERELASTICITÀ

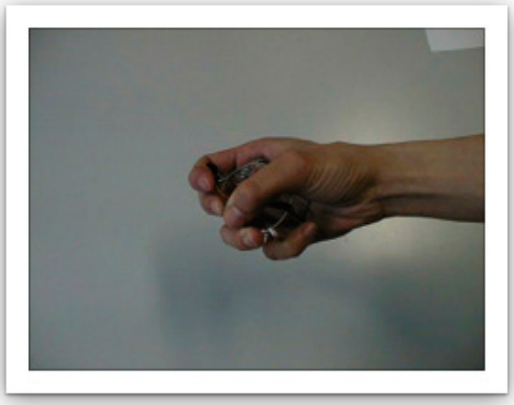
Oltre all'importante proprietà di "memoria" dei fili di Nitinol, un' altra importante caratteristica che li ha resi importanti dal punto di vista commerciale è sicuramente la **superelasticità**.

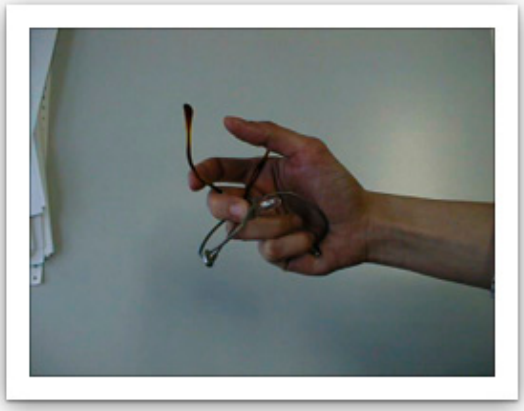
Tale proprietà si sviluppa quando la lega viene deformata, applicando un'appropriata forza, sopra la sua temperatura di trasformazione. Si genera in tal modo una Martensite indotta da sforzo (SIM) che si trova a temperatura maggiore del suo campo di esistenza: non appena lo sforzo viene rimosso essa si riconverte in Austenite indeformata. Questo fenomeno conferisce al materiale un'ottima elasticità.

Il fenomeno della superelasticità non è altro che un **effetto di memoria meccanica** del materiale: esso, sotto l'azione di uno stato di sollecitazione, assume una configurazione deformata, ben oltre il limite elastico, che può essere ripristinata togliendo lo stato di sollecitazione.

La superelasticità viene sfruttata per realizzazione di particolari montature per occhiali, come

quelli mostrati qui di seguito:





in questo modo e' possibile realizzare montature per occhiali con eccezionali caratteristiche di resistenza alla deformazione e che, nel contempo, risultano particolarmente confortevoli da indossare.

COME SETTARE LA MEMORIA DI FORMA IN UN FILO DI NITINOL?

Non solo i " metalli intelligenti" possono ricordare la forma originaria, ma addirittura possono essere addestrati a "memorizzarne" una nuova. Per settare la forma che più preferiamo, dobbiamo tenere il materiale nella fase Austenite ad alte temperature. Possiamo creare uno "stampo" utilizzando delle viti per tenere fermo il filo su una superficie, ad una data forma. Il passo successivo è quello di riscaldare il tutto all'interno di un forno o con una fiamma continua. Ad alte temperature, il filo entra nella fase di Austenite, memorizzando la forma che abbiamo deciso di imprimergli. Quando lo raffreddiamo, immergendolo in acqua fredda, il filo entra nella fase Martensite ma esso lo fa senza cambiare la forma, poichè è ancora vincolato al nostro "stampo".
Una volta rimosso il filo di Nitinol dallo stampo, possiamo deformarlo come preferiamo. Questo provoca un cambiamento di forma permanente. Se successivamente riscaldiamo il filo, esso passa dalla fase Martensite alla Austenite, ritornando indietro, alla forma originale. Così facendo, osserviamo la sua capacità di memoria. Una volta fissata la forma, possiamo ripetere per quante volte vogliamo gli step di deformazione e successivo riscaldamento del filo .

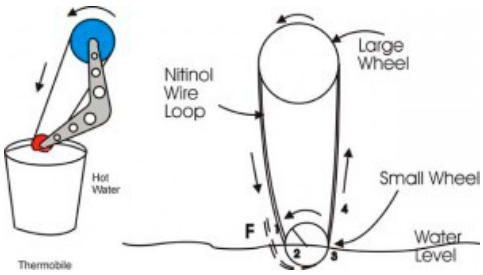
MOTORI TERMICI CON I FILI DI NITINOL: IL THERMOBILE

Il Thermobile è un modello di motore termico che usa un di filo di Nitinol per generare potenza. Il filo di Nitinol è posto su due ruote libere rotanti. Una di queste due ruote viene tenuta a diretto contatto con l'acqua calda (lato caldo) mentre l'altra è a diretto contatto con l'aria fredda (lato freddo). Il filo di nitinol, prima di essere montato sulla struttura, ha subito un trattamento di impressione della forma. Per ogni giro, ogni volta che il filo passa all' interno del recipiente con acqua calda superando la sua temperatura di transizione, si "ricorda" della sua forma origi-

nale e tenta di raddrizzarsi.

Se osserviamo la figura seguente, possiamo dire che nella posizione 1 il filo è “freddo”, quando passa dalla posizione 1 alla 2, è piegato attorno alla rotella di ottone ed entra nell’acqua calda. Quando il filo passa dalla posizione 2 alla 3 viene superata la temperatura di transizione e questo tenta di raddrizzarsi. Dalla posizione 3 alla 4 il filo si è raddrizzato e ha permesso alla ruota di ruotare Quando passa dalla posizione 4 alla 1, attraversando la ruota più grande, il filo, ha il tempo sufficiente per raffreddarsi scendendo al di sotto della sua temperatura di transizione, pronto per un altro giro.

Modello reale del Thermobile



Riassumendo, la differenza di temperatura provoca la contrazione del filo nel lato della ruota posta in acqua calda. Il filo si rilassa quando si trova a contatto con l’aria fredda. Ciò provoca una forza meccanica che garantisce la rotazione continua delle ruote. Lo stesso risultato può essere ottenuto utilizzando, al posto del recipiente con acqua calda, un raggio di sole focalizzato da una lente in un punto del filo di Nitinol.

In alcuni casi è necessario fornire un avviamento alla ruota applicando noi stessi una forza esterna.

Nel 1982 Innovative Technologies International (ITI) costruì un motore che usava fili da 0.558 mm di diametro. Il motore fu testato usando acqua calda a 55°C e aria a 25°C. Il motore riuscì a raggiungere un avelocità di 270 rpm (giri per minuto) e continuò a funzionare per un anno e mezzo.

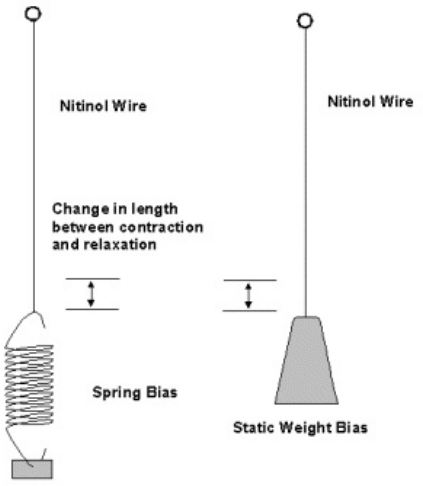
ESPERIMENTO 1

Il filo di Nitinol genera una forza per riacquisire la forma di 22.000 psi (**pound per square inch** o *libbre per pollice quadrato*). Considereremo fili da 0.006 pollici di diametro e da 0.015 pollici di diametro. Il primo ha una forza di contrazione maggiore, rispetto al secondo e può contrarsi fino all’ 8%/10% della sua lunghezza.

Contrazione e rilassamento dipendono esclusivamente dalla temperatura del filo di Nitinol. Per riscaldare e raffreddare il filo può essere utilizzato qualsiasi metodo. Un modo solitamente usato per riscaldare il filo di Nitinol è quello di farlo attraversare da **corrente elettrica**. Il filo di Nitinol rappresenta una resistenza di circa **1.25 ohm** per pollice per il filo da 0.006 pollici di diametro. La resistenza del filo alla corrente elettrica genera calore sufficiente (***riscaldamento ohmico*** *detto anche* ***“effetto Joule”***) da portarlo alla sua temperatura di transizione in un tempo che è funzione della resistenza e della corrente.

Potremmo applicare al filo una controforza nella direzione opposta alla sua contrazione. La controforza distende il filo alla sua forma originale durante la fase di bassa temperatura. Questa controforza è chiamata **forza di bias**.

Se il filo di Nitinol è portato alla sua temperatura di transizione senza forza di bias esso si contrarrà, ma, quando si raffredda, esso non ritornerà alla sua lunghezza originale proprio perchè non ha un contro peso che gli consenta di distendersi. Di conseguenza, senza una forza di bias, quando il filo viene ricaldato nuovamente non avrà luogo nessuna ulteriore contrazione. La figura di sopra mostra due metodi per applicare una forza di bias: nel primo caso una molla, nel secondo caso un peso statico.



La velocità e la forza di contrazione del filo dipendono da quanto alta e da quanto velocemente viene incrementata la temperatura del filo. Ad esempio, 400mA di corrente elettrica attraverso il filo di nitinol di 0.006 pollici di diametro produrrà un sollevamento massimo di 11 grammi raggiungendo la completa contrazione in 1 secondo. Il tempo di reazione può essere più veloce, dell'ordine del millisecondo.

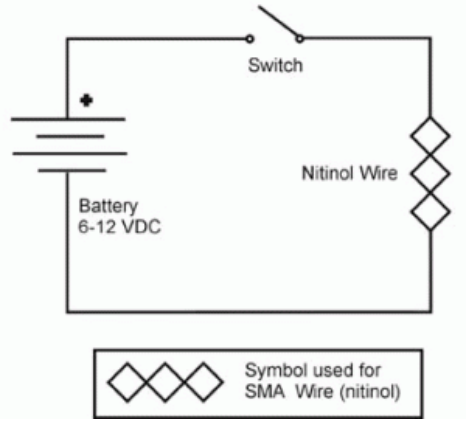
Per raggiungere queste alte correnti sono usati impulsi di breve durata. Nel fare questo bisogna considerare la massa e la velocità del materiale da spostare.

Quanto più velocemente vogliamo muovere una data massa, tanto maggiore sarà l'inerzia che dovrà essere superata. Se l'inerzia diventa maggiore di quelle sostenibili dal filo, esso potrà spezzarsi.

RISCALDAMENTO ATTRAVERSO CORRENTE ELETTRICA DIRETTA

Il filo di Nitinol può essere "attivato" (cioè portato alla sua temperatura di transizione) con una bassa tensione di alimentazione in DC (6-12Volt). Possiamo costruire un semplice circuito utilizzando una batteria, un interruttore e un pezzo di filo di Nitinol, come mostrato di seguito:

Quando si porta il filo nel suo "stato di attivazione" utilizzando corrente continua è importante non surriscaldarlo, altrimenti si correrebbe il rischio di apportare dei danni permanenti alle sue proprietà fondamentali. È importante osservare che **una corrente continua non riesce a surriscaldare il filo di Nitinol in maniera uniforme. Il metodo migliore è riscaldare il filo con un segnale PWM**. Usando un segnale PWM possiamo cambiare il duty cycle proporzionalmente alla quantità di contrazione che vogliamo ottenere. Questo tipo di controllo ci permette di "attivare " il filo di Nitinol per lunghi periodi di tempo senza causare danni causati dal sovrariscaldamento della struttura cristallina del Nitinol.

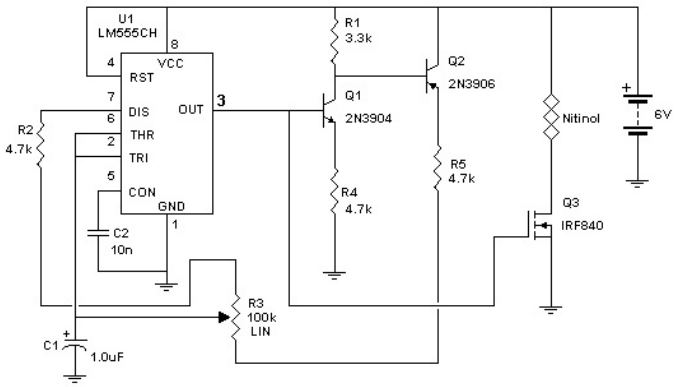


Possiamo realizzare un circuito per generare un segnale PWM.

Il circuito sarà costituito da un timer 555. Utilizzando due transistor Q1 e Q2 e un potenziometro con il timer 555 possiamo creare un uscita

con una relativa frequenza costante con un duty cycle variabile. Ecco mostrato di seguito il circuito:

Per capire come funziona nel dettaglio un timer



555 rimando all'articolo scritto da Marven.

APPLICAZIONI FUTURE

Oggi, le proprietà di memoria di forma hanno reso il Nitinol un materiale ideale in ambiti molto diversi tra loro, dalle missioni spaziali alle decorazioni floreali (farfalle e libellule animate), dagli stent vascolari utilizzati per garantire il flusso sanguigno nelle arterie otturate, agli attuatori (tendini artificiali) per microrobot operanti tramite effetto Joule. L'estrema flessibilità del nitinol trova impiego anche nelle antenne dei cellulari, negli apparecchi ortodontici, nelle montature di occhiali, nelle placche per saldare fratture ossee, come attuatori nelle protesi biomedicali.

Per quanto riguarda il futuro, la ricerca si sta impegnando sull' includere motori, nelle macchine, negli aerei e nei generatori elettrici, che utilizzano l'energia meccanica derivante dalle trasformazioni di forma. È già in corso lo studio per l'applicaizione dei fili di Nitinol nelle carrozzerie delle auto:

> **"proviamo a immaginare che deformando la carrozzeria di un auto dopo un incidente, sia possibile far ritornare la carrozzeria nella sua forma originale semplicemente riscaldando la superficie danneggiata.**

Si pensa che, quanto sopra detto, sarà una tecnologia alla portata di tutti tra circa un decennio. Altre applicazioni in campo automotive sono previste per sistemi di raffreddamento del motore, controlli di lubrificazione e per ridurre il flusso d'aria attraverso il radiatore all'avvio quando il motore è freddo e quindi a ridurre il consumo di carburante e le emissioni.

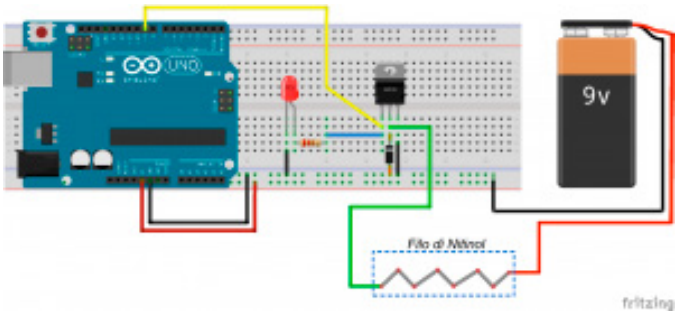
ESPERIMENTO CON ARDUINO

Materiale occorrente:

- Arduino Uno
- Potenziometro 10kΩ
- Tip 120
- breadboard
- jumpers
- connettori a coccodrillo
- batteria 9V
- Led
- Diodo
- filo di Nitinol (acquistabile su KRL)

Di seguito vedremo due diversi esperimenti realizzati con Arduino e semplicemente riproducibili. Questi esperimenti ci permettono di capire meglio il comportamento degli SMAs

SCHEMA 1

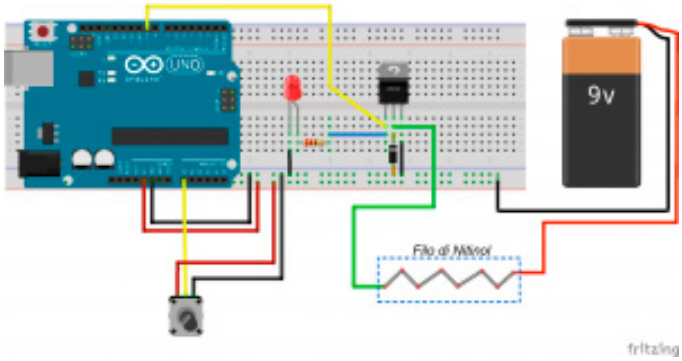


Schema 1

Il codice da caricare è il seguente:

```
int transistorPin = 9;

void setup() {
  pinMode(transistorPin, OUTPUT);
}

void loop() {
  digitalWrite(transistorPin, HIGH);
  delay(8000);
  digitalWrite(transistorPin, LOW);
  delay(8000);
}
```

SCHEMA 2



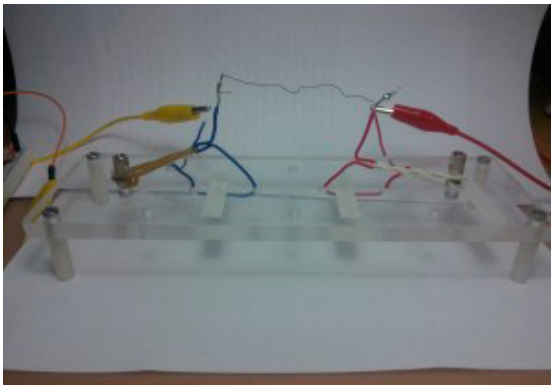
Schema 2

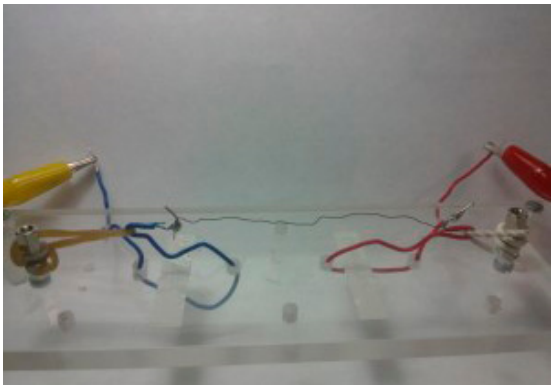
Il codice da caricare è il seguente:

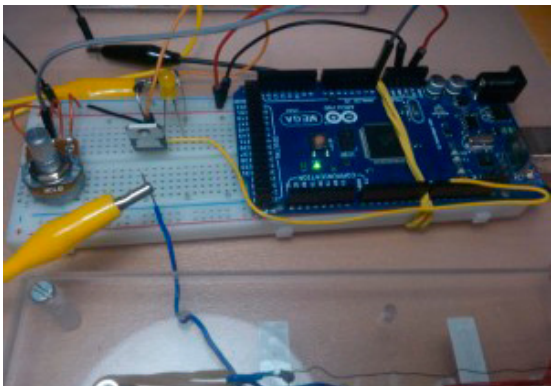
```
int transistorPin = 9;

void setup() {
  pinMode(transistorPin, OUTPUT);
}

void loop() {
  int sensorValue = analogRead(A0);
  int outputValue = map(sensorValue, 0, 1023, 0,
255);
  analogWrite(transistorPin, outputValue);
}
```

Per quanto riguarda il codice riferito allo schema 1, è stato utilizzato l 'esempio del Blink Led di Ardino con un delay di 8 secondi, utile a permettere il raffreddamento del filo di Nitinol permettendone la successiva contrazione. Ogni 8 secondi dal pin 9 di Arduino vengono erogati 5V di tensione che portano in conduzione il TIP120 il quale amplifica la corrente destinata ad attraversare il filo di Nitinol. A causa della ridotta corrente di uscita dal pin 9 (40mA) è stato inserito un transistor NPN modello TIP120. Per portare a buon fine l'esperimento è necessaria un'alimentazione esterna, nel caso specifico è stata utilizzata una batteria da 9V ma il consiglio è quello di utilizzare un alimentatore da laboratorio. Ad ogni contrazione del filo, pilotata dal pin 9, vengono assorbiti circa 800mA. **La corrente assorbita è proporzionale alla resistività intrinseca del filo**, infatti, a seconda delle caratteristiche tecniche del filo di Nitinol, come ad esempio il diametro, la corrente assorbita che permette l' ingresso in fase Austenite sarà diversa.

Per quanto concerne lo Schema 2, è stato aggiunto un potenziometro in modo da regolare il duty cycle del segnale PWM in uscita dal pin 9. Nel codice relativo al secondo schema il pin 9 non si occupa più di erogare una tensione continua di uscita di 5V ma è stato programmato per erogare un segnale PWM. Qui di seguito alcune foto della fase di test in laboratorio:







Circuito reale

Per eseguire i test, basandomi sui precedenti schemi, ho realizzato una particolare struttura che mi consentisse di mantenere teso il filo di Nitinol. La struttura è caratterizzata da una base in plexiglas forata in cui ho inserito due normalissimi fili con anima rigida a cui ho saldato il filo di Nitinol. Per garantire la tensione al filo di Nitinol ho utilizzato due elastici ancorati ai fili con anima rigida.

Per la realizzazione del struttura ho preso ispirazione da un esperimento realizzato da un certo Anton e pubblicato su youtube che è possibile vedere cliccando sul seguente link.
Durante la realizzazione dell'esperimento vi è stato un interessante scambio di informazioni via e-mail con Anton, dove ci si confrontava sui limiti e le possibili applicazioni dei fili di Nitinol. **Questa è un ulteriore prova di come l'open source avvicini appassionati con interessi comuni e ne nasca un grande confronto e una crescita reciproca**.

AVVERTIMENTI E PRECAUZIONI

È doveroso consigliare di prestare la massima attenzione durante l' esperimento. I possibili pericoli possono derivare dalla diminuzione del delay (nel codice dello schema 1) o dalla diminuzione drastica del valore del potenziometro (nello schema 2). Entrambi i casi, infatti, portano il filo di Nitinol a un pesante stress dovuto alla tensione continua che lo attraversa per un tempo prolungato provocandone il surriscaldamento e la conseguente incandescenza. Prestiamo, inoltre, molta attenzione a non toccare il circuito durante il funzionamento a causa delle alte correnti in gioco. Il TIP120 raggiungerà temperature abbastanza elevate, il consiglio è quello di installare un dissipatore per evitare di danneggiare il dispositivo a lungo termine.

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/smas-gestiamo-questi-materiali-intelligenti-con-arduino

Realizzazione di un rilevatore SONAR con Arduino

di Giuseppe Silano

L'IDEA ALLA BASE DEL PROGETTO

L'idea alla base del progetto è tanto semplice quanto ingegnosa: utilizzare un sensore ad ultrasuoni, nello specifico un *HC-SR04*, per rilevare la distanza da un oggetto, con l'ausilio di Arduino, e scansionare l'area circostante utilizzando un motore in continua, attraverso la *Motor Shield di Infineon*. L'elaborazione avviene in ambiente *Matlab Simulink*, lo stesso utilizzato per programmare il microcontrollore, dove vengono raccolte le misure temporali, convertite poi in distanza, e visualizzato un grafico che mostra la sagoma di eventuali oggetti presenti. Ovviamente, l'ambiente *Matlab Simulink* non è l'unico spendibile sia per la programmazione, sia per l'acquisizione e l'elaborazione delle informazioni. Soluzione alternativa è realizzare uno script in *Python*, che acquisisce le misure di distanza (la conversione tempo-distanza viene gestita in Arduino, anche se formalmente l'operazione non è corretta) e visualizza il grafico aggiornandolo di volta in volta con le nuove misure. Si tratta di una valida alternativa, ma che necessita, al contempo, di spendere maggior tempo nella stesura del codice.

Parliamo di operazione non corretta perché è buona norma deputare al microcontrollore il minor numero di operazioni possibili, in modo che l'hardware sia quanto più possibile dedicato alla gestione delle attività essenziali. Eventuali elaborazioni, all'interno dello *sketch* (nome dato al software realizzato per Arduino), potrebbero far "perdere" tempo al microcontrollore in attività svolgibili tranquillamente in fase di elaborazione dati da un processore *general purpose*, dalle frequenze di clock più elevate, dunque riducendo la possibilità di misure falsate o perdita di informazione.

All'interno di questo articolo verranno presentate entrambe le soluzioni, allo scopo di fornire un'informazione quanto più completa possibile, lasciando poi al lettore la facoltà di decidere quale soluzione adottare nel caso voglia realizzare il progetto, valutando con mano le differenze tra le due alternative descritte.

In sintesi, l'utilizzo di Matlab potrebbe essere la soluzione migliore per chi mastica poco i linguaggi di programmazione, mentre scrivere uno *sketch* per Arduino potrebbe essere la soluzione più adeguata per coordinare motore, sensore, acquisizione ed elaborazione.

Criticità alla base del progetto sono:

- **Gestione del tempo minimo** in cui il sensore deve sostare in una specifica posizione, affinché la misura avvenga correttamente e non venga falsata dal ritorno di altri segnali (sarà più chiaro quando si analizzerà il funzionamento del sensore ad ultrasuoni, in particolare l'*HC-SR04*);
- **Calibrazione della velocità** del motore in continua per la rotazione del sensore, capace di scansionare l'area con una angolo che spazia tra gli 0° ed i 180°, ed inversione della rotazione;
- **Dimensionamento dell'alimentazione** necessaria a garantire il funzionamento dell'intero sistema, e sua interconnessione con l'apparato sensoristico e di movi-

mentazione.

RADAR O SONAR: LE DIFFERENZE

Prima di addentrarci e capire nel dettaglio quali sono stati i passi che hanno portato alla realizzazione del rilevatore SONAR con Arduino, è bene definire quali sono le differenze tra un sistema SONAR ed uno RADAR. Indagare sulle differenze è fondamentale per comprendere il perché della scelta fatta, le limitazioni che essa comporta ed i possibili problemi realizzativi.

Il termine RADAR è l'acronimo di *Radio Detection And Ranging*, e denota i sistemi capaci di rilevare e localizzare oggetti a mezzo di onde elettromagnetiche. Schematicamente in un sistema radar un trasmettitore invia nello spazio energia elettromagnetica sotto forma di impulsi a microonde, di frequenza tra le onde radio e la radiazione infrarossa; tale energia viene riflessa da eventuali bersagli, catturata da un'antenna ed elaborata dal ricevitore al fine di determinare la presenza o meno del bersaglio (Detection=rivelazione) e di misurare la distanza dal ricevitore (Ranging=localizzazione in distanza). Di norma trasmettitore e ricevitore sono collocati e condividono lo stesso sistema d'antenna (come nel progetto in esame), si parla di radar monostatico. Al contrario, in applicazioni belliche si separa il trasmittente dal ricevente, per evitare l'individuazione dell'antenna, realizzando il cosiddetto radar bistatico. Giusto per completezza, in questi ambienti la rivelazione è un problema di classificazione binaria, mentre la localizzazione è un problema di stima.

Il SONAR, l'acronimo di *SOund Navigation And Ranging*, essenzialmente si comporta come un sistema radar, ovvero è in grado di determinare la presenza o meno di un bersaglio e di misurarne la distanza. Al contrario del sistema radar, il sonar utilizza onde acustiche e non elettromagnetiche, che meglio si propagano in materiali rigidi come i liquidi, e di meno nell'aria, con un comportamento del tutto duale rispetto alle onde elettromagnetiche. Tali caratteristiche fanno del sonar lo strumento più indicato in ambienti marini, o dove in generale bisogna attraversare dei liquidi, mentre il radar è utile quando è necessario misurare distanze nell'etere. In generale, possiamo affermare che più è rigido il mezzo meglio le onde sonore si propagano, raggiungendo distanze più elevate, al contrario di ciò che accade con quelle elettromagnetiche.

Le onde, siano esse sonore o elettromagnetiche, possono essere inviate come la modulazione di un'onda continua oppure ad impulsi, la seconda soluzione è la più efficiente da un punto di vista energetico quando si intende raggiungere grandi distanze.

Capite le differenze, per sommi capi, verrebbe da pensare: **perché è stato realizzato un sistema SONAR di rivelazione nell'etere con Arduino piuttosto che uno RADAR?** Il motivo è semplicissimo: i costi. Il sensore utilizzato allo scopo (lo analizzeremo a breve) ha un costo decisamente ridotto rispetto ad un'equivalente che lavora alle microonde. Dunque, questa è la ragione che ne hanno decretato la scelta, ritenendolo il più giusto da utilizzare per la realizzazione del progetto.

Ricordiamo, infatti, che l'intento non è realizzare un sistema vendibile ma progettare e, soprattutto, costruire un dispositivo *embedded* capace di chiarire i passi che separano la teoria dalla pratica, capendo quali possono essere i problemi in corsa d'opera. Arduino – secondo l'opinione di chi scrive – è un ottimo strumento didattico ed utile per le aziende, per tutto quello che è il discorso prototipizzazione rapida.

I COMPONENTI

Definita l'idea alla base del progetto, le limitazioni e le motivazioni che hanno portato alla scelta di un sensore ad ultrasuoni, piuttosto che uno a microonde, per la rivelazione di un oggetto nell'etere, si può tranquillamente passare a quelli che saranno i componenti che verranno utilizzati, da qui a poco, per la realizzazione del sistema. Capire l'hardware a disposizione è un passaggio chiave per farsi un'idea degli eventuali problemi che possono sorgere in corsa d'opera e, soprattutto, come evitarli.

Figura 1.3.1: Sistema di controllo, Arduino UNO.

Poche ma essenziali sono le unità utilizzate:

1. **Arduino Uno**: sistema di controllo con il quale viene acquisita la distanza rilevata dal sensore ad ultrasuoni, ed inviato il comando per la movimentazione del motore in continua;
2. **Motor Shield di Infineon**: necessaria per comandare il motore in continua, regolandone velocità e direzione. Il tutto è possibile grazie alla presenza di un H-Bridge (ricevuta grazie al contest);
3. **Un motore in continua da 9 V**: deputato alla rotazione della base su cui è installato il sensore;
4. **Un sensore ad ultrasuoni, l'HC-SR04**: attraverso il quale vengono acquisite le misure di distanza.

Arduino (Figura 1.3.1) è una scheda elettronica di piccole dimensioni con un microcontrollore ed una circuiteria di contorno, utilizzata per la realizzazione di prototipi, per scopi hobbistici e didattici.

La forza di Arduino risiede nell'essere Open Source: sia l'ambiente di sviluppo che il software di corredo, come anche lo stesso schema hardware, sono liberi ed aperti a tutti. Volendo fare un'analogia, si tratta della stessa filosofia utilizzata dalle distribuzioni Linux. Per di più, la presenza di interfaccia di comunicazione USB (necessaria per lo scambio dati con il PC, utilizzabile anche come alimentazione), di un regolatore di tensione (maggiore flessibilità nella scelta del sistema di alimentazione) e pin per l'interfacciamento I/O (possibilità di collegare dispositivi analogici e/o digitali, sia in input che in output), lo rendono molto versatile.

All'hardware viene affiancato un ambiente di sviluppo integrato (IDE) multipiattaforma (Macintosh, Linux e Windows), che richiama la stessa struttura del C e C++ (noti linguaggi di programmazione) e consente, anche a chi di elettronica ne sa ben poco, di cimentarsi nella realizzazione di progetti scrivendo da se gli *sketch*. Altro punto a favore di Arduino, è la possibilità di interagire con i sistemi operativi e con i software ivi installati, ad esempio Adobe Flash.

Tutte queste caratteristiche hanno consentito lo sviluppo, da parte di aziende e di privati, di librerie che facilitano sempre di più i neofiti nella realizzazione di sistemi abbastanza evoluti, riducendo i tempi ed i costi di progetto. Inoltre, la possibilità di collegarlo addirittura ad un LCD

(*Liquid Cristal Display*, Schermo a Cristalli Liquidi) lo rende un dispositivo completamente autonomo. Avviato il software, il sistema lavora in maniera del tutto indipendente acquisendo i dati che si trovano sul campo, elaborandoli ed intraprendendo azioni sulla base del sistema di controllo implementato. Si rimanda il lettore ad ulteriori approfondimenti al riguardo.

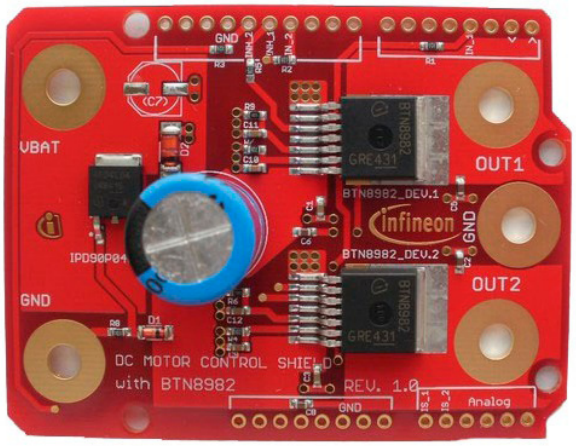


Figura 1.3.2: Motor Shield di Infineon.

La Motor Shield (Figura 1.3.2) è unità fondamentale per il controllo di un motore DC (*DirectCurrent*, Corrente in Continua), senza la quale non sarebbe possibile regolarne né la velocità, né la direzione. Inoltre, consente di alimentare il motore con una sorgente di tensione differente da quella che alimenta l'unità di controllo, evitando così possibili danneggiamenti del microcontrollore, incapace di fornire una corrente sufficiente a pilotarlo.

Componente fondamentale, deputato al pilotaggio del motore, è il Ponte-H (H-Bridge), saldato sulla scheda, circuito responsabile della modulazione della corrente media che scorre ai capi del motore. Senza entrare in inutili tecnicismi, ripetendo qualcosa di già noto a molti, la shield utilizza il segnale proveniente dal pin PWM (*Pulse Width Modulation*, Modulazione ad Ampiezza di Impulsi) di Arduino per modulare l'apertura e la chiusura dei MOSFET di potenza all'interno del ponte, modificano la corrente media e dunque la sua velocità.

Volutamente non si è voluti entrare nel dettaglio di funzionamento del ponte, né sulla tecnica di modulazione per il comando dei MOSFET (PWM o PLL, acronimo di *Phased Locked Loop* ovvero Anello ad Aggancio di Fase), si rimanda il lettore più curioso a successivi approfondimenti al riguardo.

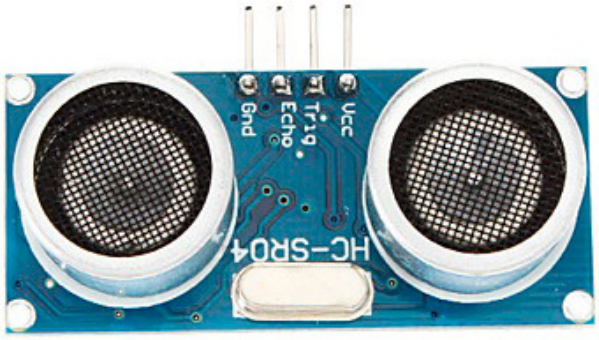


Figura 1.3.3: Sensore ad ultrasuoni HC-SR04.

Il sensore ad ultrasuoni (*HC-SR04*, Figura 1.3.3) è l'aspetto che merita un maggiore livello di dettaglio, essendo il cuore dell'intero sistema. Come già detto in precedenza, utilizza le onde sonore per la rilevazione della distanza da un oggetto. A tale proposito, è fondamentale sapere che le onde sonore quando incontrano l'interfaccia di separazione tra due mezzi materiali danno luogo a fenomeni quali rifrazione e riflessione.

Si parla di rifrazione (Figura 1.3.4) quando l'onda incontra la superficie di separazione tra due mezzi aventi impedenza acustica differente, parte dell'onda viene riflessa, l'altra parte continua la propagazione all'interno del mezzo con un angolo di propagazione differente da quello di incidenza. Il fenomeno causa una perdita di energia, dunque riduce la distanza di propagazione dell'onda acustica. Si tratta di un fenome-

no da evitare, perché non consente la misura della distanza da un oggetto.

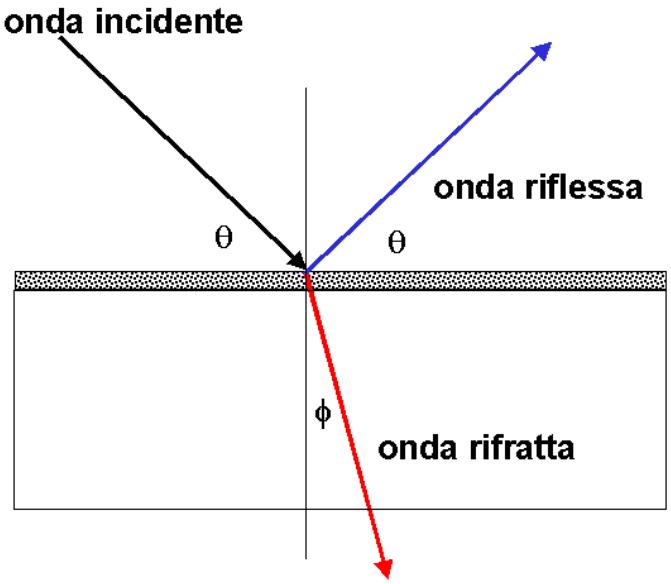


Figura 1.3.4: Schema riassuntivo del fenomeno della rifrazione.

La riflessione, al contrario, è un fenomeno che ben si presta alle operazioni di misura. A tale proposito, quando si parla di riflessione è importante distinguere tra due fenomeni riflessivi (Figura 1.3.5):

- **Riflessione speculare**: l'onda sonora impatta con l'interfaccia di separazione di due mezzi materiali e si riflette, dirigendosi nuovamente verso la sorgente. Le onde riflesse si propagano in un'unica direzione;
- **Scattering o diffusione**: fenomeno funzione del rapporto delle lunghezze d'onda sonore e della dimensione media delle particelle. Le onde riflesse si propagano in tutte le direzioni. Se l'energia delle onde riflesse non è sufficientemente elevata non è possibile individuare un picco d'energia, e quindi non sarà possibile effettuare la misura di distanza.

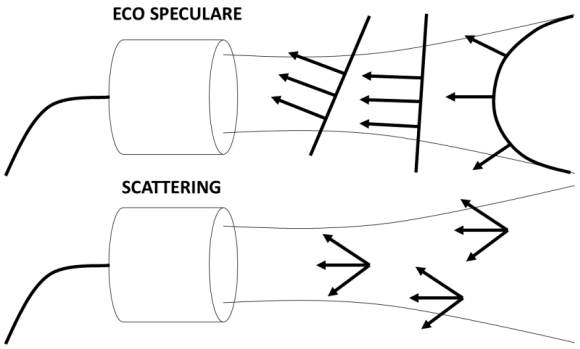


Figura 1.3.5: Differenza tra riflessione speculare e scattering.

Un sensore ad ultrasuoni come l'*HC-SR04* misura il tempo impiegato (*ms*) dalla onde sonore (Figura 1.3.6), emesse da una sorgente, a tornare indietro, a seguito dello scontro con un ostacolo in grado di rifletterle. La limitazione nell'utilizzo del sensore risiede nell'impossibilità d'impiego con oggetti in grado di assorbire le onde acustiche. Nonostante ciò, riesce ad offrire una buona accuratezza ed una stabile lettura della distanza nell'intervallo dai *2* ai *400 cm*, tenuto in considerazione anche il costo (tra i 2/3 €).

Il fascio di onde sonore (Figura 1.3.7), emesse dalla sorgente situata di fianco al ricevitore, ha forma conica, la stessa forma delle onde riflesse da un ostacolo. La forma del fascio fa sì che il sensore riceva molte riflessioni, dai diversi oggetti presenti all'interno dell'ambiente, ciò rende il sensore, da solo, incapace di distinguere un oggetto da un altro o aperture negli oggetti troppo piccole. Per sopperire a tale problema, il sensore è dotato di un'apposita circuiteria di controllo.

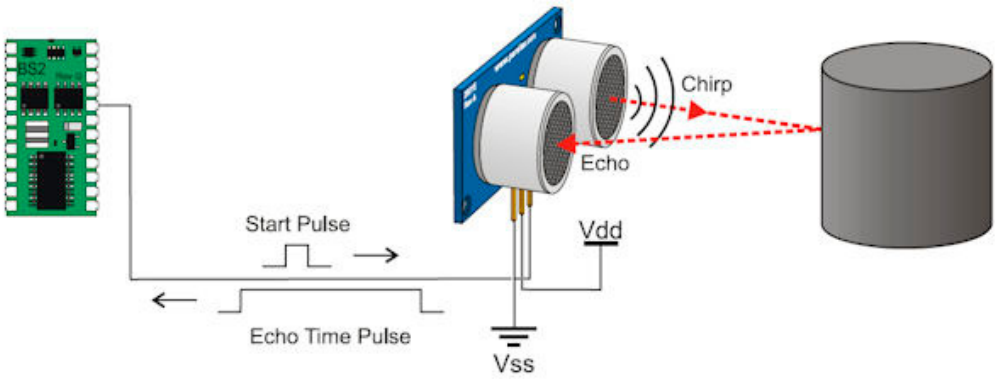


Figura 1.3.6: Schema di funzionamento sensore ad ultrasuoni.

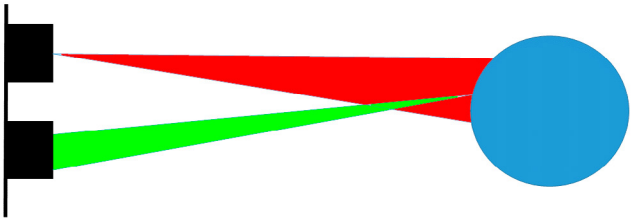


Figura 7: Fascio di onde emesse e ricevute dal sensore ad ultrasuoni.

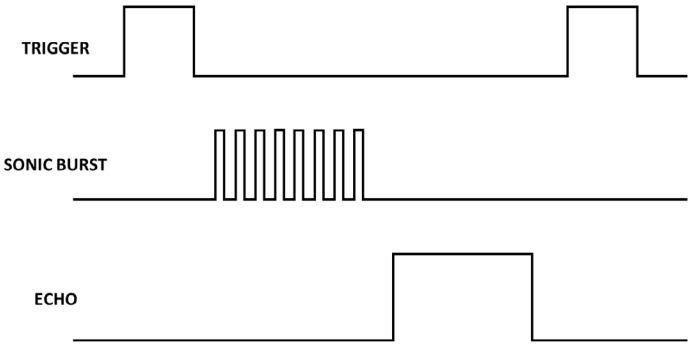


Figura 8: Logica dei segnali dal/al HC-SR04.

Il principio di funzionamento (Figura 1.3.8) è abbastanza semplice:

1. Si porta al valore logico alto il pin di trigger del sensore per un tempo di almeno *10 µs*;
2. Il modulo automaticamente invia un'onda acustica alla frequenza di *40 kHz* e si pone in ascolto di un segnale ad onda quadra di ritorno;
3. Se il segnale torna alla sorgente, in uscita viene restituito il tempo che è intercorso tra l'invio e la ricezione del segnale, da qui si procede al calcolo della distanza utilizzando la formula:

$$Distanza\ (cm) = \frac{Echo\ (\mu s)}{58}$$

LO SCHEMA CIRCUITALE

Dopo aver capito quali sono i componenti del progetto, come funzionano, e quali sono i vincoli fisici che ne condizionano l'operatività, non resta altro da fare che interconnetterli tra loro, alimentandoli secondo quanto specificato all'interno dei *datasheet*.

Al fine di mantenere livelli di tensione di alimentazione stabili, quanto più possibile, è stato utilizzato un regolatore di tensione (Figura 1.4.1), in particolare il *L7805CV* , capace di ricevere in ingresso una tensione continua fino a *35 V* e di produrre in uscita una tensione continua di *5 V* , la stessa utilizzata per alimentare il sensore ad ultrasuoni *HC-SR04*.

Il regolatore di tensione, presente in qualunque circuito elettronico, ha lo scopo di fornire le precise tensioni di alimentazione necessarie per il funzionamento dei circuiti elettronici analogici e digitali presenti all'interno di una rete, evitando così malfunzionamenti e danneggiamenti del sistema dovuti a bruschi sbalzi di tensione. Tali sono le ragioni per cui si è preferito non collegare il sensore ai pin di alimentazione forniti dal sistema di controllo (Arduino UNO), ma attraverso

il regolatore di tensione direttamente al package di batterie garantendo che l'*HC-SR04* funzioni sempre con un alimentazioni pari a *5 V*.

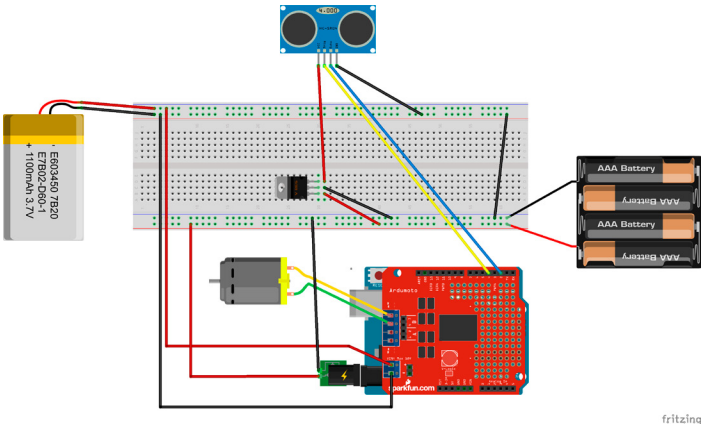


Figura 1.4.1: Schema circuitale del rilevatore sonar.

Inoltre, particolare attenzione è stata posta nel collegare tutti i dispositivi allo stesso riferimento (*ground*, massa), con il fine di evitare misurazioni errate dovute a riferimenti diversi. Come mostrato in Figura 1.4.1, i riferimenti delle batterie, come quelli del regolatore di tensione, del sistema di controllo e della motor shield, sono tutti cortocircuitati tra loro.

Nello schema circuitale (Figura 1.4.1), realizzato utilizzando il software open source Fritzing, sono evidenziate le interconnessioni tra i dispositivi e l'alimentazione:

- I pin del sensore ad ultrasuoni, in particolare *ECHO* e *TRIGGER*, sono collegati rispettivamente a pin digitali 2 e 4 di Arduino;
- Il motore in continua è collegato alla morsettiera montata sulla motor shield.

MOTOR SHIELD DI INFINEON: LE MODIFICHE

Prima di passare all'implementazione del sistema di rivelazione in ambiente *Matlab Simulink* verranno illustrate le modifiche apportate alla motor shield di Infineon.

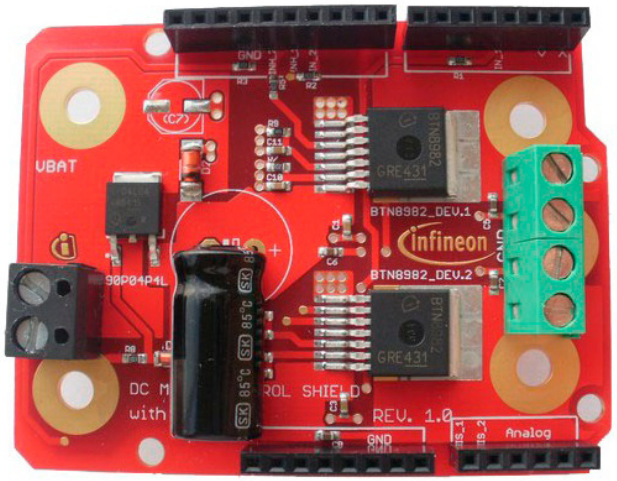


Figura 1.4.2: Motor Shield di Infineon a seguito delle modifiche.

Per prima cosa, è stato necessario dissaldare il condensatore elettrolitico per poi rimontarlo in maniera orizzontale, con il fine di ridurre quanto più possibile l'ingombro consentendo così il montaggio di altri moduli al sistema di controllo come ad esempio un display LCD.

Successivamente, sono state montate delle morsettiere sia sulle piazzole per il pilotaggio dei motori, sia su quelle per il collegamento della sorgente di alimentazione supplementare, il tutto allo scopo di renderne più agevole l'utilizzo piuttosto che montare le boccole a cui poi collegare le relative banane (Figura 1.4.3).

Figura 1.4.3: Connettori, in gergo banane.

Infine sono stati saldati i connettori fondamentali per il montaggio impilato con il sistema di controllo. Il circuito ottenuto è mostrato in Figura 1.4.2.

RILEVATORE IN AMBIENTE MATLAB-SIMULINK

Passiamo ora al vivo della trattazione, illustrando il software implementato in ambiente *Matlab-Simulink* per la realizzazione del rivelatore sonar con Arduino.

Matlab, per chi non lo conoscesse, è ambiente di lavoro molto utilizzato in ambito ingegneristico, e non solo, per il calcolo numerico e l'analisi statistica. Consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, e interfacciarsi con altri programmi. Questi sono soltanto alcuni degli innumerevoli vantaggi che il software mette a disposizione dei propri utenti, e che ne hanno decretato la diffusione.

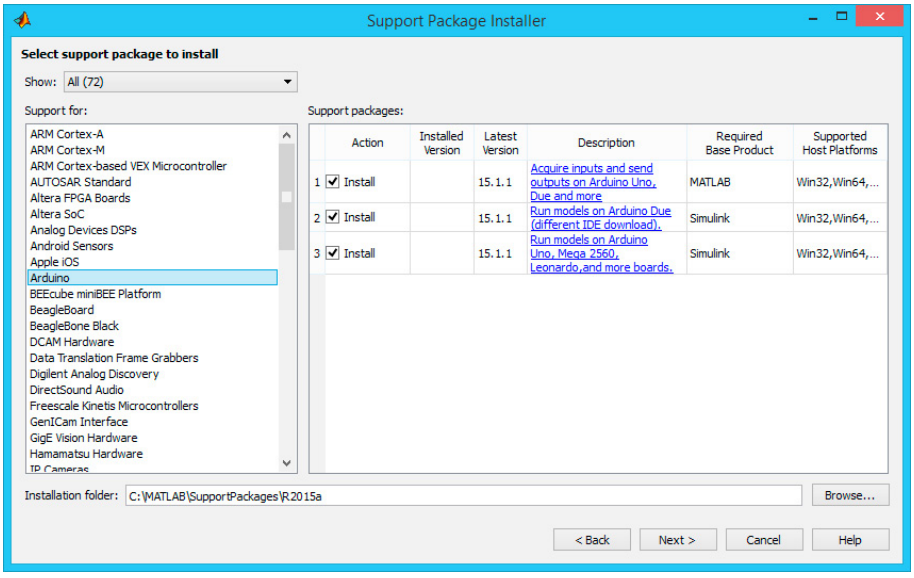


Figura 1.5.1: Support Package Installer di Matlab.

Data la moltitudine di utilizzatori, ed appurate le potenzialità del sistema di controllo Arduino, gli sviluppatori hanno pensato di agevolare il lavoro dei progettisti fornendo un ambiente grafico che consentisse la programmazione del microcontrollore e l'utilizzo di tutti gli strumenti che Matlab mette a disposizione. A tale proposito è stato sviluppata un'apposita libreria, installabile direttamente dal promt dei comandi digitando "*supportPackageInstaller*" (Figura 1.5.1), per poi scegliere il device di interesse (funzione disponibile dalla 2012b e successive).

L'ausilio del supporto IDE di Arduino per Matlab, e di *s-function ad hoc* (appositi blocchetti, disponibili nella libreria di default di Simulink, per l'utilizzo di codice C in Matlab) per l'interfacciamento dei sensori e della motor shield all'ambiente di sviluppo, hanno consentito la realizzazione del software di controllo per il rivelatore sonar riportato in Figura 1.5.2. Si tratta di un software modulare, composto da un blocco

- **SONAR**: misura la distanza da un oggetto;
- **MOTOR**: gestisce il motore in continua (velocità e cambio direzione);
- **XY GRAPH**: realizza lo schermo del rivelatore mostrando, in maniera approssimativa (dovuta alla bassa risoluzione del sensore), la sagoma dell'oggetto di fronte al rivelatore.

Il blocco SONAR (Figura 1.5.3) integra la libreria *NewPing* di Arduino, utilizzata per la sua affidabilità ed ottima struttura, per la misurazione della distanza da un oggetto. Inoltre, il blocco saturazione evita che malfunzionamenti del sensore possano restituire valori numerici fisicamente non misurabili, perché al di fuori dell'intervallo di misurazione descritto dal *datasheet*.

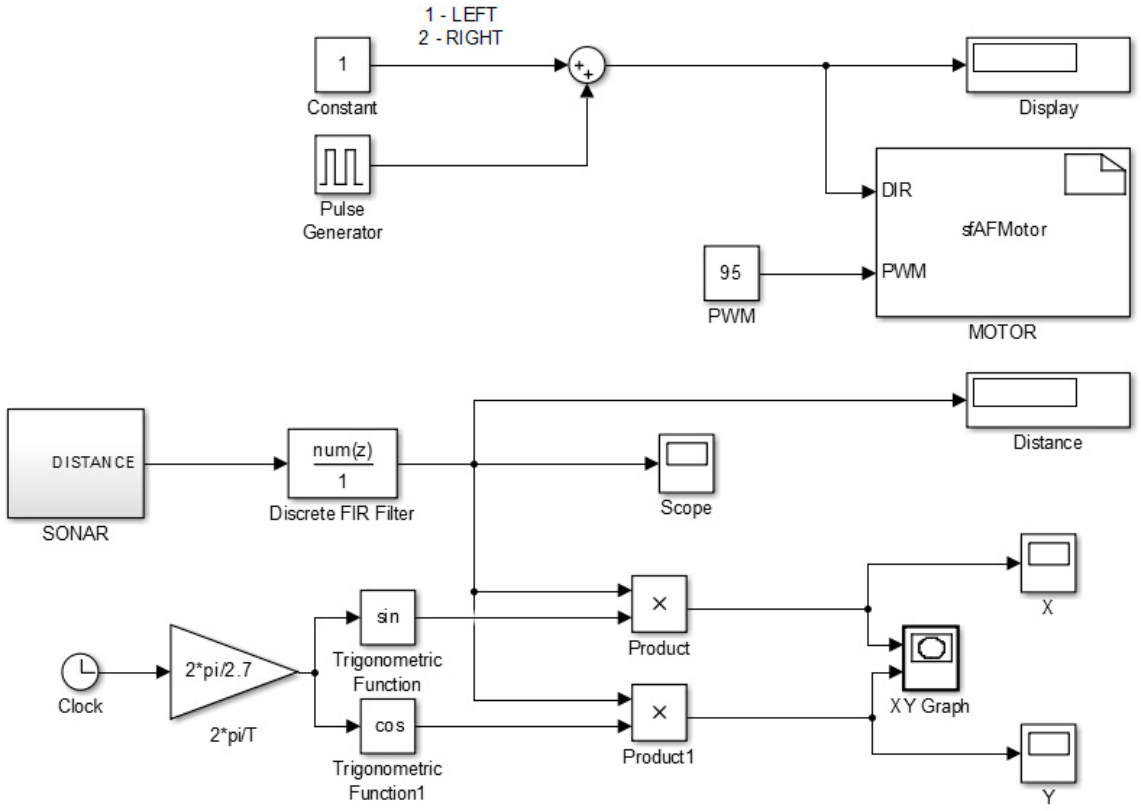


Figura 1.5.2: Software del rivelatore sonar realizzato in Matlab.

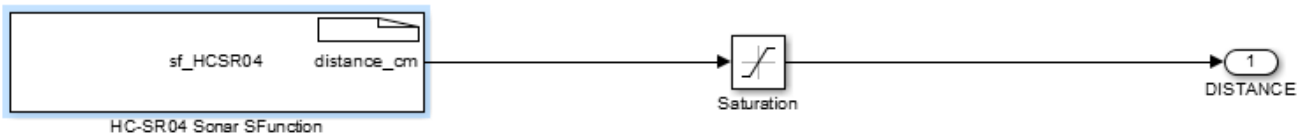


Figura 1.5.3: Blocchetto SONAR in Matlab.

Osservando l'andamento delle misure di distanza, si è notata però la presenza di misure falsate (fluttuazioni su valori continui) dovute a *fault* del sensore: mancato ritorno dell'onda acustica, causato dall'irregolare superficie dell'oggetto, che porta a misure di distanza pari a quella massima misurabile (*400 cm*). Ragion per cui si optato per la diminuzione del ping intervall (distanza temporale tra l'impulso di *TRIGGER* ed il successivo), fino a raggiungere il minimo possibile, in modo da acquisire il maggior numero di misure date poi in ingresso ad un **filtro FIR a media mobile**. Il numero di coefficienti utilizzati per il filtro è pari a 10. Questo riceve in ingresso un array contenente le misure di distanza e sostituisce a ciascuna misura la media di quelle vicine, in modo da eliminare le fluttuazioni (*spike*) nelle misure.

Il blocco MOTOR (Figura 1.5.4), al contrario, è responsabile della movimentazione del motore in continua. La velocità, durante l'intero processo di rilevazione, assume un valore costante pari a 95, scelto sperimentalmente tra il set di valori ammissibili (0 − 255). Per quanto concerne, invece, la direzione essa varia secondo la logica riportata in Figura 1.5.5.

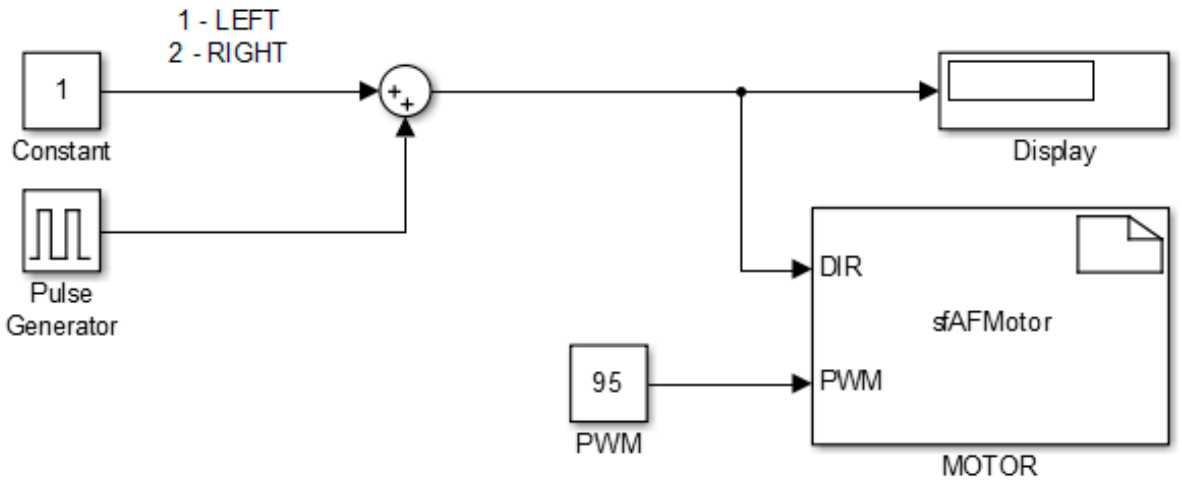


Figura 1.5.4: Blocchetto MOTOR in Matlab.

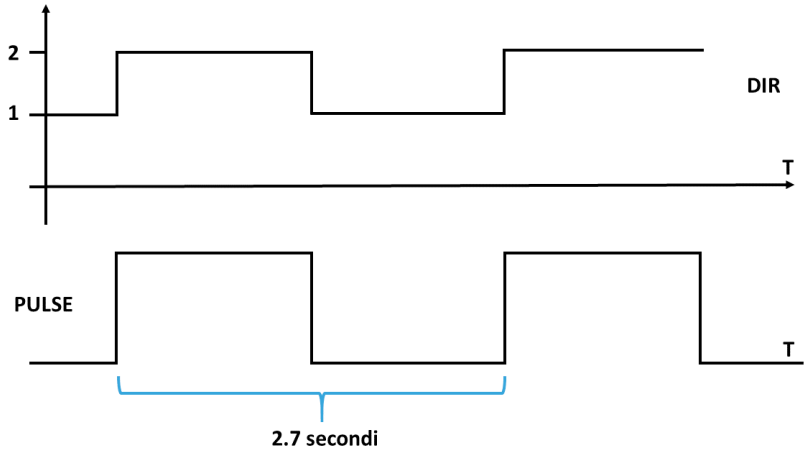


Figura 1.5.5: Logica cambio direzione del blocco MOTOR.

Il motore ruota in senso orario, spostando il sensore da sinistra verso destra, fino a che il segnale che condiziona la direzione (*DIR*) assume valore pari a 2, ovvero fino al fronte di discesa del segnale *PULSE* (segnale che si somma al valore all'unità nel software riportato in Figura 1.5.2). Trascorsi 1.35 secondi (metà periodo dell'onda quadra, anch'esso valore sperimentale) dal fronte di salita di *PULSE*, il segnale *DIR* assume nuovamente valore pari 1, conseguentemente il motore cambia verso di rotazione da orario ad antiorario (da destra verso sinistra) fino al prossimo fronte di salita.

Infine, il software si compone di un ultimo blocco che converte le misure realizzando lo schermo del rilevatore. La distanza viene scomposta luongo le componenti x e y, per poi essere inviata al blocco **XY GRAPH** che provvede a rappresentarle, utilizzando l'espressione

dove

$$x = d \cdot sin\left(\frac{2\pi t}{T_P}\right)$$

$$y = d \cdot cos\left(\frac{2\pi t}{T_P}\right)$$

- *d*: è la distanza dall'oggetto rilevata dal sensore ad ultrasuoni;
- *t*: è la generica variabile tempo;
- T_P : è il periodo del segnale *PULSE*.

RILEVATORE IN PYTHON E SKETCH ARDUINO

Seguendo quanto realizzato in ambiente *Matlab-Simulink* è stato implementato lo Sketch in Arduino (riportato di seguito) per il controllo del motore in continua, e l'acquisizione delle misure di distanza dall'oggetto. Come nelle *s-function* realizzate in Matlab, per il sensore ad ultrasuoni (*HC-SR04*) è stata utilizzata la libreria *NewPing*, mentre per il controllo di velocità e direzione del motore sono stati utilizzate le funzioni *analogWrite* e *digitalWrite*. L'autore non intende entrare nei dettagli, commentando riga per riga il codice prodotto, perché ritiene che il lettore abbia una conoscenza minima di tale linguaggio, in caso contrario rimanda il lettore ad ulteriori approfondimenti al riguardo.

Di seguito è riportato lo script Python per l'acquisizione delle misure distanze su seriale e la realizzazione del grafico XY.

```
import math
import os
import matplotlib.pyplot as plt

serialFromArduino = serial.Serial("/dev/
ttyACM4",115200)
serialFromArduino.flush()

plt.figure(1)
plt.xlabel('Graphic')

while True:
 try:
 val = ord(serialFromArduino.read())
 time=tm_sec(5)
 x=val*sin((2*pi*time)/2.7)
 y=val*cos((2*pi*time)/2.7)
 plt.plot(x, y, 'r--', linewidth=2.0)
 plt.show()

except KeyboardInterrupt:
 exit()

serialFromArduino.close()
```

CONCLUSIONI

La realizzazione di un rilevatore SONAR con Arduino, progetto di relativa semplicità, ha consentito di affrontare concetti chiave come la differenza tra un sistema RADAR ed uno SONAR, il dimensionamento dell'alimentazione, per poi passare ai problemi ed alle relative soluzioni legate all'utilizzo di un sensore ad ultrasuoni *low cost* come il *HC-SR04*. Si tratta di *know how* che fanno la differenza tra un progetto "funzionante" ed uno "fatto bene", consentendo all'applicazione di operare anche in condizioni diverse da quelle dell'esperimento.

Inoltre, l'implementazione del software con due strumenti diversi (*Matlab-Simulink e Sketch-Python*) ha consentito di delineare vantaggi/svantaggi derivanti dai due ambienti di sviluppo, aprendo le porte anche a chi di programmazione sa ben poco con tutti gli annessi e connessi.

In tema di sviluppi futuri si potrebbe pensare all'integrazione del sistema all'interno di piccoli robot, realizzati utilizzando sempre Arduino, combinando così la parte di *sensing* con quella di controllo.

```
#include <NewPing.h>
// Pin assignment for first half-bridge
const int IN_1 = 3; // input (PWM)
const int INH_1 = 12; // inhibit (low = sleep)

// Pin assignment for second half-bridge
const int IN_2 = 11; // input (PWM)
const int INH_2 = 13; // inhibit (low = sleep)

//Variabile to change the motor direction
int timer;
float constant, interval;

#define TRIGGER_PIN 4 // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN 2 // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 400 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400 cm.

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum distance.

void setup() {

 // Set correct input/output state
 pinMode(IN_1, OUTPUT);
 pinMode(INH_1, OUTPUT);
 pinMode(IN_2, OUTPUT);
 pinMode(INH_2, OUTPUT);

 // Set initial output states
 analogWrite(IN_1, 95); // set motor speed to 0
 digitalWrite(INH_1, HIGH); // enable OUT1
 analogWrite(IN_2, 95); // set motor speed to 0
 digitalWrite(INH_2, LOW); // disable OUT2

 //Variabile to change direction
 constant=2.7; interval=constant;

 // initialize serial communication:
 Serial.begin(115200);
 while (!Serial) {
 ; // wait for serial port to connect. Needed for Leonardo only
 }
}

void loop() {

 timer=millis();
 delay(50); // Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest delay between pings.
 unsigned int uS = sonar.ping(); // Send ping, get ping time in microseconds (uS).
```

```
 Serial.println(uS / US_ROUNDTRIP_CM); // Convert ping time to distance in cm and print result (0 =
outside set distance range)

 /* half bridge mode, two unidirectional motors motor 1 connected to OUT1 and GND motor 2 connected
to OUT2 and GND */

 //Change motor direction
 if(timer>=interval){
 digitalWrite(INH_1, LOW); // disable OUT1
 digitalWrite(INH_2, HIGH); // enable OUT2
 interval=interval+constant;
 }
}
```

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/realizzazione-di-un-rilevatore-sonar-con-arduino

Più sprint alla tua auto con Arduino e Processing

di Ernesto Sorrentino

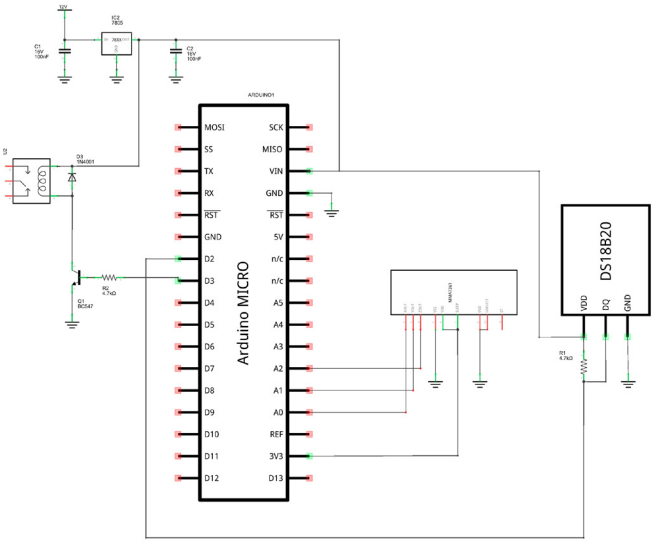
Questo progetto propone un dispositivo per rilevare spostamenti meccanici di una leva e attivare un relè a una determinata posizione stabilita. Può essere utilizzato in molti settori, come in una catena di montaggio per determinare l'eccessiva inclinazione di un piano o per monitorare spostamenti di una leva. Nel seguente articolo, invece, vedremo come utilizzarlo per individuare la prima marcia di un'autovettura in modo tale da disabilitare, tramite relè, il climatizzatore ottenendo più potenza a bassi regimi. In seguito dopo un determinato tempo, settato al PC, si potrà riattivare il clima.

Il trasduttore, da porre sulla parte meccanica in movimento, è collegato al circuito tramite un cavo di prolunga mentre il resto dell'elettronica dovrà essere inscatolata in un contenitore in ABS di piccole dimensioni. **Il sistema, collegato al PC tramite porta USB, può essere programmato per assegnare i tempi e la sensibilità d'intervento del relè**.

https://www.youtube.com/watch?v=C8QnCYx3u_s

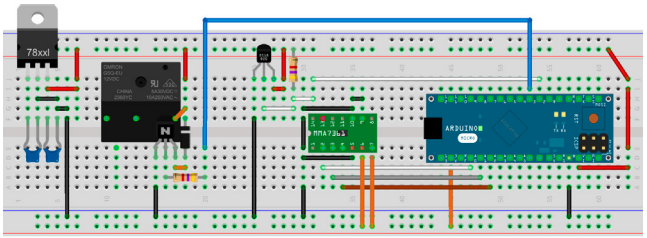
Come già accennato, il sistema è composto di due parti; **hardware e software**.
Nella prima parte sarà esaminato lo schema elettrico e la procedura di montaggio. Nella seconda illustrerò l'utilizzo del programma al PC (per configurare e tarare il circuito) e la realizzazione dello stesso con Processing e PGUI (**tool d'interfaccia grafica per Processing**).

HARDWARE



Il circuito elettronico è costituito da un accelero-

metro analogico, un sensore di temperatura e un relè tutti gestiti con Arduino “micro”. La semplicità dello schema permette una prototipazione anche su “breadbord”.



L’accelerometro, il MMA7361 della **Freescale Semiconductor**, è del tipo capacitivo a tre assi e con uscite analogiche.

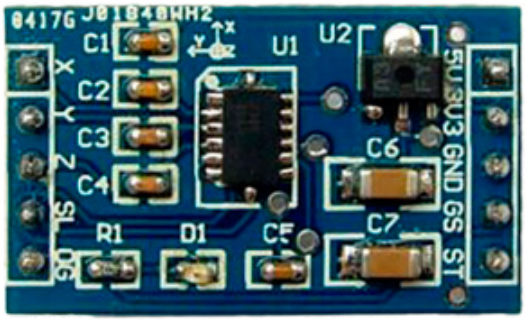
Tra le sue caratteristiche troviamo:

- *consumi ridotti (400 uA, di cui si riducono ulteriormente a 3uA in mode sleep);*
- *range di tensione basso (da 2.2 V a 3.6 V);*
- *alta sensibilità (800 mV/g a 1.5 g);*
- *possibilità di configurare la sensibilità (1.5 g o 6 g);*
- *rilevamento di caduta libera (0 g);*
- *a norma con i requisiti RoHS;*
- *economico.*

Le applicazioni tipiche sono:

- *rilevamento caduta dei HDD;*
- *registrazione eventi per contapassi e gaming;*
- *compensatore di tilt nelle bussole elettroniche e altro ancora.*

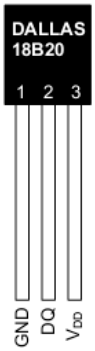
Questo integrato è fornito solo con **package LGA-14** pertanto è scomodo da stagnare su PCB. Inoltre per ovviare all’installazione dei componenti aggiuntivi, come i condensatori da 3.3 nF nelle uscite analogiche (per eliminare eventuali disturbi), ho optato per una versione commerciale già montata su scheda. In più è presente anche un regolatore di tensione per alimentarlo a 5 V.



L’accelerometro è collegato alle porte analogiche di Arduino come segue:

- Asse X alla porta A0;
- Asse Y alla porta A1;
- Asse Z alla porta A2.

Il **DS18B20** della **DALLAS** è un termometro digitale con misurazione della temperatura a 12bit/centigradi. Aggiunto al circuito per monitorare la temperatura d’ambiente e attivare il relè solo per soglie prestabilite tramite tool al PC.



Tra le sue caratteristiche troviamo:

- range di lettura esteso (da -55 °C a 125 °C) con un’accuratezza del +/-0. 5 °C;
- la tensione di operazione può variare da 3 V a 5,5 V;
- la comunicazione dei dati col microcontrollore avviene su un Bus da un solo filo.

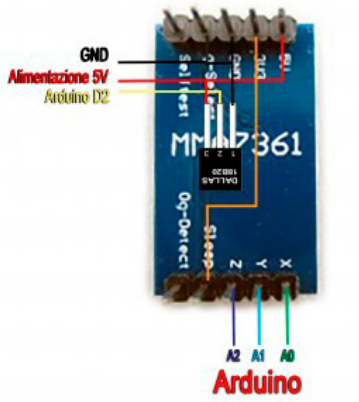
Ogni DS18B20 ha un codice di serie univoco a 64 bit, che consente a più sensori di funzionare sullo stesso Bus 1-wire; **Questo elemento è connesso alla porta D2 dell’Atmega**.

Il relè, tipo **HFD3**, può essere configurato tramite tool per simulare un pulsante o come inter-

ruttore. Configurazioni che possono essere utilizzare per pilotare carichi pesanti o per gestire un comando a bassa potenza, come il pulsante "AC" del clima.

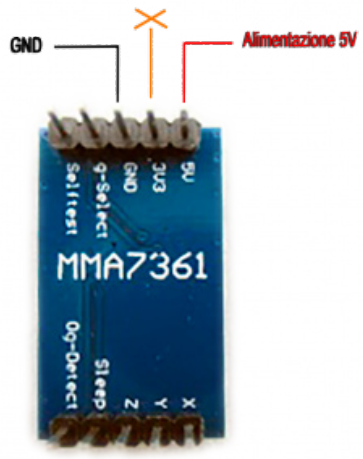
ASSEMBLAGGIO

Entrambi i sensori (**MMA7361 e DS18B20**) devono essere collegati al circuito tramite un cavo di prolunga e impacchettati insiemi con una guaina termo-restringente, ottenendo così un unico involucro da collegare sulla parte meccanica da monitorare.

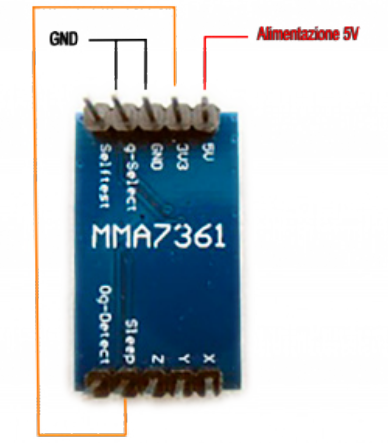


Il cavo dovrà essere composto di **5 fili + calza** (da usare per connettere la massa) si consiglia di utilizzare quelli per apparecchiature audio perché sono più resistenti e flessibili.

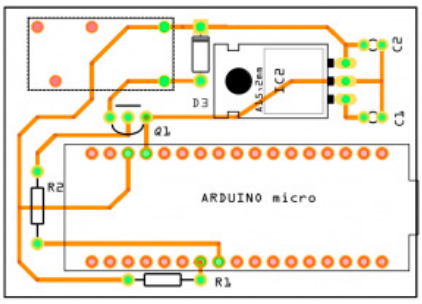
A differenza di com'è riportato dallo schema l'accelerometro, essendo già assemblato, **dovrà essere alimentato a 5 V direttamente da Arduino**.

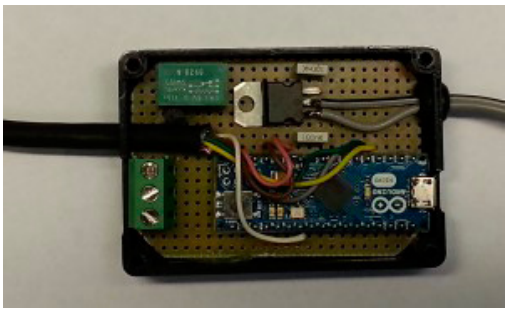


Prima di sigillare i sensori bisogna settare l'MMA7361 **configurando due pin**. L'ingresso "**g-select**" deve essere collegato a massa per impostare il "g-range" a 1,5 g (sensibilità a 800mV/g). Invece l'ingresso "**sleep**" al livello alto (**ATTENZIONE per questo integrato il livello alto è inteso a 3,3V e NON 5V**) per disabilitare la funzione di stand-by.



Il resto del circuito è stato montato, per la sua semplicità, su basetta "mille-fori" e inscatolato in un contenitore in ABS.



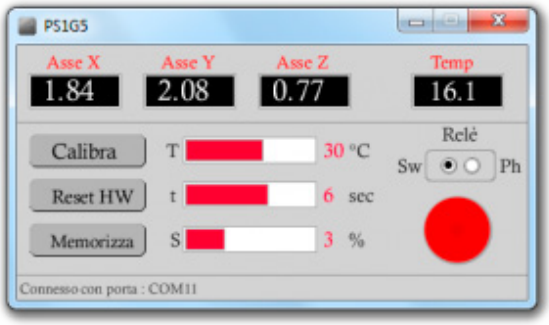


PROGRAMMA

Per configurare la parte hardware è stato realizzato un tool con Processing e PGUI (classe per creare comandi d'interfaccia grafica con

Processing). L'utilizzo è semplice e ci sono pochi comandi da regolare. Avviando lo sketch di Processing noteremo una finestra con l'avviso di scollegare il dispositivo dall'alimentazione esterna e collegarlo a **una qualsiasi porta USB** del PC.

Cliccando sul pulsante "**Connetti**" il programma tenterà una connessione con Arduino e dopo aver stabilito il collegamento cambierà videata passando a, **quella nominata nello sketch, "window 2"**.



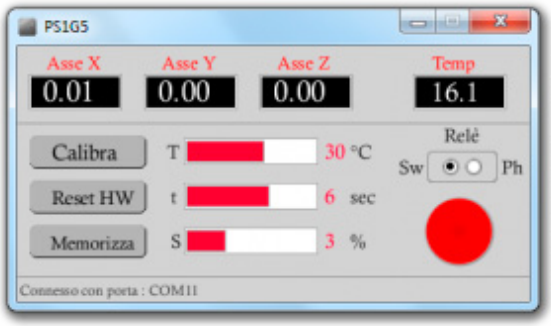
Questa finestra è divisa in tre parti. La prima in alto composta da quattro display; quella centrale con i comandi per interagire col dispositivo e l'ultima parte, in basso, per visualizzare le informazioni di stato dell'hardware.

DISPLAY

I display sono utilizzati per visualizzare le informazioni ricevute da Arduino. I primi tre visualizzano i valori degli assi espresso in "**g**" mentre nel quarto display è visualizzata la **temperatura** dell'ambiente dove situato il sensore.

PULSANTI

Il primo comando, pulsante "**Calibra**", ha la funzione di azzerare i valori degli assi per creare un punto di riferimento; ottimo per determinare **la posizione di "folle" del cambio**.



Per eventuali errori o modifiche, del punto appena creato, è possibile ripristinare i valori degli assi cliccando il pulsante "Reset HW". L'ultimo pulsante "Memorizza" è utilizzato per inviare, ad Arduino, la posizione scelta per far attivare il relè. Tale valore rimarrà in memoria, anche in caso di mancata alimentazione al circuito, fino a quando non verrà sostituito con un nuovo dato. Tutti i dati saranno memorizzati nell'eeprom di Arduino e caricati nei rispettivi vettori (label) a ogni riavvio del dispositivo.

CURSORI

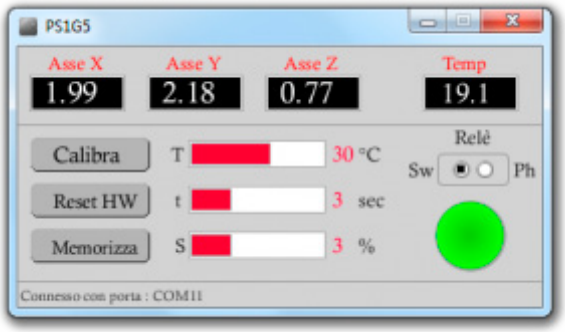
Il primo cursore (T) è utilizzato per determinare la soglia di temperatura minima per far attivare in relè. Come valore da impostare fare sempre riferimento alla temperatura rilevata nell'abitacolo. Il secondo cursore (t) imposta il tempo, espresso in secondi, della durata attivazione relè dopo che la posizione memorizzata è cambiato. Il terzo cursore (S) definisce la tolleranza, espressa in percentuale, dei valori letti dall'accelerometro con il punto memorizzato degli assi. Un'opportuna regolazione dei cursori "t" e "S" aiuta a evitare un falso intervento del relè in piani molto inclinati.

CHECK

Il relè può essere configurato in due modalità; **Switch o push**. Come interruttore, settando il check "Sw", che si attiva quando il sensore di trova nella posizione memorizzata e si disattiva dopo trascorso il tempo "t" e fuori dal range memorizzato. Nell'altra modalità, **settando il check "Ph"**, il relè si comporta come un pulsante; ossia ogni volta che il sensore è nella posizione memorizzata il relè si attiva solo per **200 ms**. Ripeterà la stessa funzione quando il sensore si troverà fuori dal range della posizione memorizzata e dopo trascorso il tempo settato col cursore "t".

LED

Quest'oggetto grafico segnala lo stato del relè, luce rossa per disattivato e verde per attivo.



Terminata la configurazione e memorizzato la posizione del sensore, il programma chiede di riavviare il dispositivo scollegandolo dalla porta USB, **questo permetterà di caricare tutti i dati memorizzati nell'eeprom di Arduino**.

INSTALLAZIONE

I passaggi successivi saranno poco dettagliati dato che ogni veicolo ha una procedura d'installazione a se. Il primo passo, più semplice, è quello di collegare l'accelerometro alla leva del cambio. Basta togliere la cuffia dalla leva (in alcuni casi per toglierla bisogna prima svitare il pomello del cambio) e fascettare il sensore più in lato possibile. In questa posizione avrà un ampio spazio di movimento e di conseguenza più tolleranza d'intervento.

Il secondo passo consiste nel collegare i contatti del relè al carico, quest'operazione può essere svolta in due modi. Il primo consiste nel collegare i contatti del relè in serie al fusibile del climatizzatore, tagliando il filo corrispondente dietro alla scatola fusibile. Per questa scelta bisogna settare il relè in **modalità "Sw"**. Il secondo metodo è quello di collegare i contattati del relè **in parallelo al pulsante "AC"** del clima per simu-

larne la pressione. Con questo metodo il relè deve essere configurato in **modalità "Ph"**. In ultimo bisogna connettere l'alimentazione al circuito, consiglio di usare un positivo sotto chiave (15) in modo da non aver assorbimenti, anche se ridotti, a veicolo spento. Tale segnale può essere prelevato dalla scatola fusibile o, di sicuro presente, dietro il blocchetto della chiave. ***Per eventuali dettagli d'installazione, su ogni tipo di veicolo, potete far richiesta inserendo un commento alla fine dell'articolo***.

SKETCH

Nello sketch di Arduino, **per gestire il sensore di temperatura**, sono state utilizzate due librerie di Miles Burton; **"OneWire" e "DallasTemperature"**. La "OneWire" coordina la trasmissione dati su un solo filo mentre la "DallasTemperature" è adoperata per dialogare con i sensori di temperatura della famiglia "**DSxx**". Il cuore del programma è composto di un ciclo di eventi strutturato nel seguente modo:

- Lettura dati dai due sensori, MMA6371 e DS18B20;
- Invio dati al PC;
- Ricezione dati dal PC;
- Analisi dei dati ricevuti;
- Gestione del relè.

```
void loop() {
 readData();
 sendData();
 reciveData();
 command();
 satusData();
}
```

Nella prima parte, Arduino, esegue la lettura dei dati ricevuti dai sensori

```
void readData(){
 somma_x = 0;
 somma_y = 0;
 somma_z = 0;

 for (int i=0;i<10;i++) {
 val_x = analogRead(A0);
 somma_x = somma_x + val_x;
 val_y = analogRead(A1);
 somma_y = somma_y + val_y;
 val_z = analogRead(A2);
 somma_z = somma_z + val_z;}

 media_x = somma_x / 10;
 media_y = somma_y / 10;
 media_z = somma_z / 10;

 x = media_x-cal_x;
 y = media_y-cal_y;
 z = media_z-cal_z;

 XG = abs(x/165.5);
 YG = abs(y/165.5);
 ZG = abs(z/165.5);

 sensors.requestTemperatures();
 t = (sensors.getTempCByIndex(0)-2);
 dtostrf(t,0,1,T);
}
```

Dopo la memorizzazione nelle rispettive label i dati vengono inviati al PC. La trasmissione avviene tramite il comando "**Serial.print**", inviando più blocchi di dati separati da uno spazio. Ogni blocco corrisponde a un valore letto dai sensori mentre lo spazio di separazione **servirà a Processing per "splittare" i dati ricevuti**.
Dati inviati da Arduino:

```
void sendData(){
 Serial.print(stored);
 Serial.print(" ");
 Serial.print(active);
 Serial.print(" ");
 Serial.print(XG);
 Serial.print(" ");
 Serial.print(YG);
```

```
 Serial.print(“ “);
 Serial.print(ZG);
 Serial.print(“ “);
 Serial.println(T);
}
```

I dati inviati sono :

- Info stato di archiviazione;
- Stato del relè;
- Asse X, Y e Z (espresso in g);
- Temperatura.

Dati ricevuti da Processing:

```
if (myPort.available() > 0) {
  val = myPort.readStringUntil(10);
  String[] a = split(val, ‘ ‘);
  stored = int(a[0]);
  active = int(a[1]);
  text_d1 = a[2];
  text_d2 = a[3];
  text_d3 = a[4];
  text_d4 = a[5];
  myPort.clear(); }
```

Il terzo punto consiste nel controllare la ricezione dei dati e smistare i valori nelle rispettive label. In questo passaggio ho adottato **una tecnica diversa da quella precedente**. I dati **non** sono inviati, da Processing, tutti contemporaneamente ma solo quelli interessati e per riconoscere un determinato valore dagli altri è stato aggiunto un **prefisso alfabetico**, **personalizzato**, come nell’esempio:
a ogni pressione di un pulsante, Processing, invia il numero corrispondente (**label “push”**) aggiungendo il prefisso “**p**”

```
write = “p”+push;
 myPort.write(write);
```

Arduino, grazie al prefisso, riconosce in quale label memorizzare il dato.

```
if(Serial.available() > 0){
  if(Serial.peek() == ‘p’){
  Serial.read();
  push = Serial.parseInt();
  }
}
```

Sono state utilizzate due tecniche di trasmissione dati diverse solo per motivo didattico.

Nella quarta parte del programma si analizzano i dati ricevuti per gestire le funzioni corrispondenti come: **calibrazione, memorizzazione e restore**. Nella quinta e ultima parte dello sketch avviene la comparazione dei valori attuali dell’accelerometro con quelli memorizzati per gestire il relè. Un altro punto fondamentale dello sketch di Arduino è la **memorizzazione di tutti i dati ricevuti nell’eeprom**; scelta affrontata per alimentare il circuito tramite un positivo sotto chiave (15). A ogni avvio del circuito il programma carica tutti dati dell’eeprom alle label corrispondenti. La registrazione dei dati è strutturata come segue:

- Registro 01 e 02 valore di riferimento dell’asse X;
- Registro 03 e 04 valore di riferimento dell’asse y;
- Registro 05 e 06 valore di riferimento dell’asse z;
- Registro 07 valore della temperatura;
- Registro 08 valore del tempo;
- Registro 09 stato del relè;
- Registro 10 e 11 valore di calibrazione dell’asse X;
- Registro 12 e 13 valore di calibrazione dell’asse Y;

- Registro 14 e 15 valore di calibrazione dell'asse Z.

I segnali letti sugli assi dell'accelerometro sono tensioni espresse in bit. Ogni singolo asse genera un valore che può superare i 255 bit, **valore massimo da inserire in una unica cella dell'eeprom**. Pertanto sarà diviso in due registri; nel primo registro saranno inseriti i primi 255 bit e nel secondo i restanti del valore letto.

Gli sketch originali di Processing e Arduino, in download a fine articolo, sono completi di commento in ogni loro parte.

ELENCO MATERIALE

1 Arduino micro;
1 sensore temperatura DS18B20;
1 Accelerometro MMA7361;
1 relè da 5V;
1 regolatore di tensione 7805;
1 Transitor BC547;
2 resistenze da 4.7 Kohm 1/4W;
2 condensatori da 100 nF;
1 diodo 1n4148;

CONCLUSIONI

Il progetto può crescere e migliorare, le idee per evolverlo sono tante. Ad esempio con una piccola modifica allo sketch di Arduino è possibile utilizzare un accelerometro digitale con interfaccia I2C, in questo modo si ridurrà il numero dei fili nel cablaggio tra i sensori e centralina. Un'altra idea potrebbe essere, modificando lo sketch di Processing, aggiungere altri pulsanti per memorizzare più posizioni del cambio.

Download dei file PS1G5

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/piu-sprint-alla-tua-auto-con-arduino-e-processing

La regolazione di temperatura con Arduino

di Nicola Taraschi

L'HARDWARE

L'architettura dell'hardware con ARDUINO è schematizzata nella figura 1. Tre potenziometri permettono l'acquisizione dei valori di setpoint, banda proporzionale o differenziale nella regolazione on-off, e del tempo integrale, che sono i parametri che definiscono le tre regolazioni considerate. Il punto centrale dei potenziometri va al pin analogico di ingresso corrispondente. Nella figura 2 lo schema del circuito di potenza che alimenta le lampadine da 5 watt 12 V, realizzato con mosfet IRF530, che sopporta senza raffreddamento fino a 25 watt. S= segnale di controllo da ARDUINO. L=lampadina, VDD= alimentazione a 12 V.Il segnale S proveniente da un pin analogico di uscita Arduino con tecnica PWM attiva o meno la conduzione del Mosfet. Il diodo fast serve per proteggere il mosfet dalle scariche dei carichi induttivi presenti su un carico ohmico-induttivo e nel caso di carichi puramente ohmici non sarebbe necessario. Il display GROVE è un display seriale che ha bisogno, a parte l'alimentazione a 5V DC, di solo due segnali, e quindi particolarmente comodo nei collegamenti rispetto a quelli tradizionali che hanno bisogno di 6 segnali. Uno shield Grove permette attraverso cavetteria apposita il collegamento rapido ad Arduino. Il sensore di temperatura è un NTC il cui collegamento è raffigurato nella figura 3. Nel caso di questo sensore il segnale è analogico. La tensione del punto A acquisita da Arduino varia in funzione della temperatura e l'apposita routine software la trasforma nel valore di temperatura corrispondente. Il "difetto" dei sensori analogici è nella oscillazione dei valori rilevati a causa dei disturbi. Un miglioramento si ha con i sensori digitali tipo DS18B20 il quale produce un segnale digitale e pertanto esente da oscillazioni.

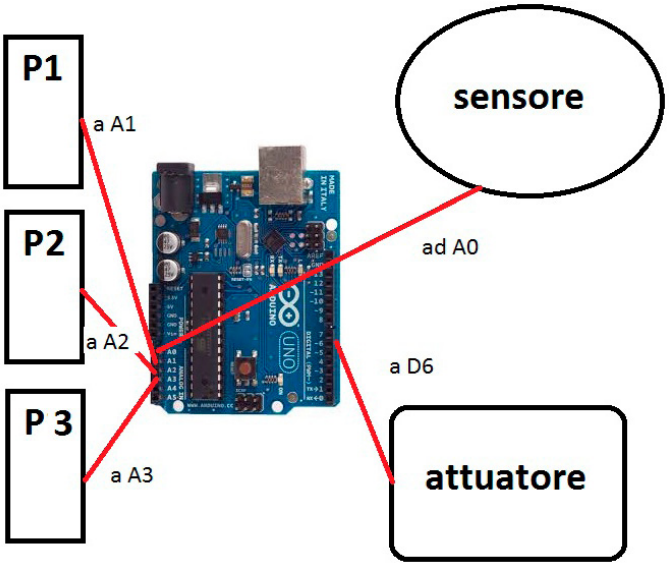


Figura 1: schema collegamenti

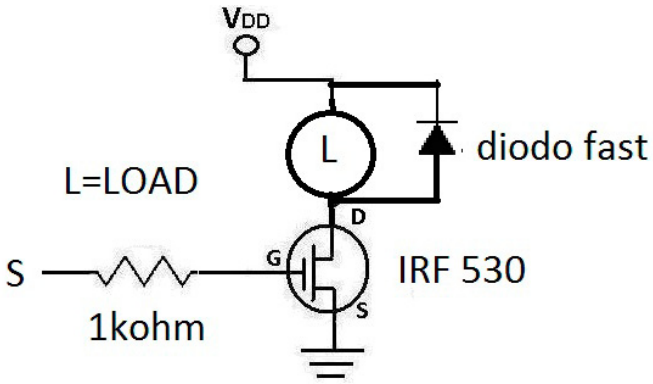


Figura 2: il circuito di potenza con i Mosfet

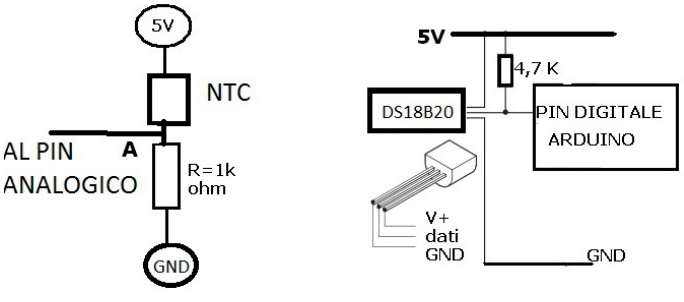


Figura 3:collegamento dei sensori NTC e DS18B20

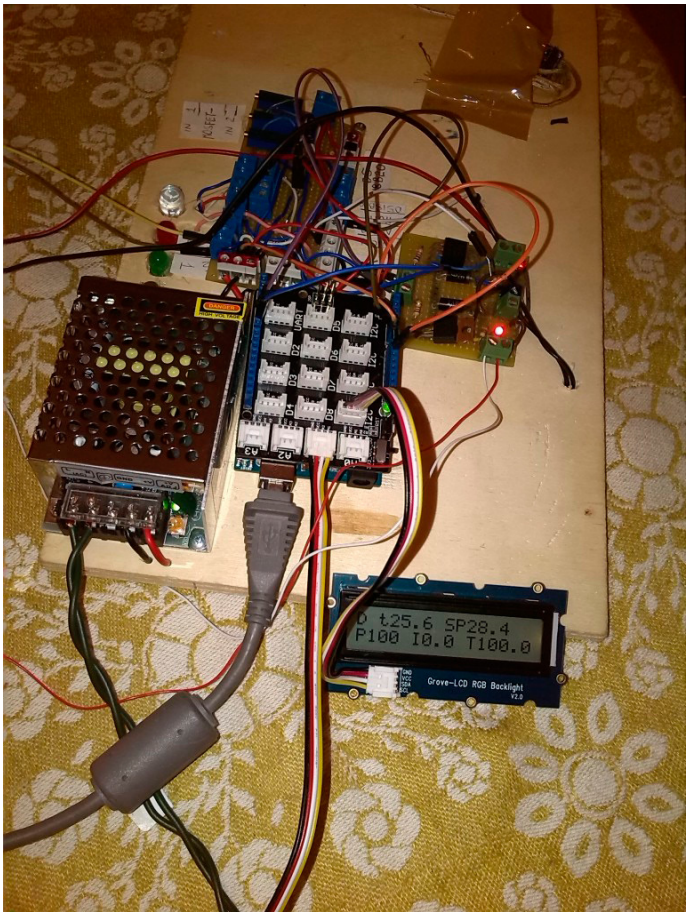


Figura 4: l'apparecchiatura complessiva. L'alimentatore, la scheda ARDUINO con uno shield Grove per l'inserzione del display, e il circuito di potenza.

Figura 5. Le due lampadine e il sensore in un contenitore momentaneamente scoperto

Figura 6:il display GROVE

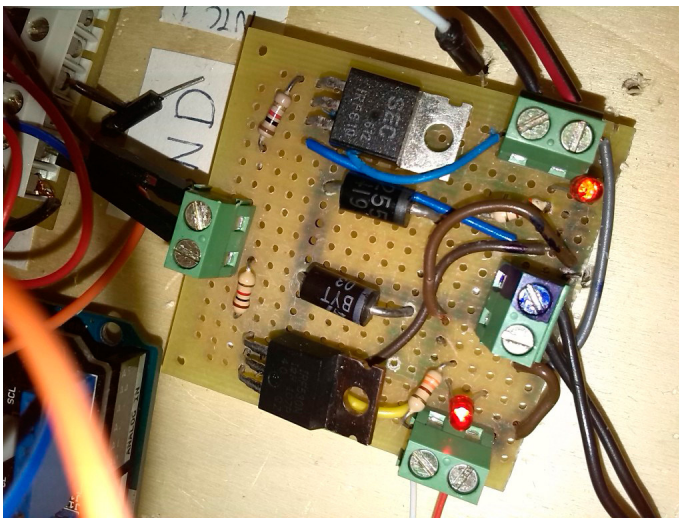


Figura 7: I due mosfet

IL SOFTWARE

Come è noto ogni programma su ARDUINO si sviluppa su due routine fondamentali: SETUP, la routine che viene eseguita solo e sempre al lancio del programma, e LOOP che è la routine che viene eseguita successivamente e sempre finchè l'hardware è alimentato. Tutte le altre routines sono richiamate o da quella di SETUP oppure, comunemente da quella di LOOP. Il software si sviluppa secondo i seguenti blocchi funzionali:

- Inizializzazione
- acquisizione del valore della temperatura
- acquisizione dei parametri
- attesa della richiesta dati dal PC ed esecuzione del tipo di regolazione impostato ed invio dati al PC

Inizializzazione. La routine SETUP inizializza la porta seriale di comunicazione con il PC,(Serial..) configura le uscite come OUTPUT, (pinMode…) inizializza il display LCD, (setup_lcd..)setta il tipo di regolazione secondo due parametri(inizia_reg..). Il primo è true=regolazione diretta, false=regolazione inversa. Ricordiamo che nella regolazione diretta ad un aumento del valore della variabile corrisponde una diminuzione della potenza dell'attuatore e viceversa nella regolazione inversa. Il secondo segue la convenzione 1=on-off, 2=proporzionale e 3=PI.

La routine SETUP

```
void setup()

{

Serial.begin(9600);

pinMode(uscita,OUTPUT);//setta l'uscita

setup_lcd() ; //inizializza il display

//setta il tipo di regolazione

inizia_reg(true,3);

}
```

Si sottolinea che il software è valido qualunque sia la grandezza fisica da controllare. Basta infatti cambiare, oltre l'hardware relativo, i parametri di settaggio della regolazione. Sostituendo la sola routine leggo_temp con altra relativa a diversa grandezza, l'elaborazione ed il programma rimangono invariati.

La routine di lettura dal sensore NTC

```
float leggo_temp(int porta)

{float beta=3450;

float tensione;

float rx ;

float R;

int letto;

letto=analogRead(porta);

tensione=letto*0.00489;

R=(5000-1000*tensione)/tensione;

R=R/10000;

rx=log(R)/log(exp(1));

rx=rx/beta+0.003355705;

rx=1/rx-273;

return rx;}
```

La routine leggi_porta

```
float leggi_porta(int quale, float xmin, float xmax)

{ int aa;

float r;
```

```
aa=analogRead(quale);

r=xmin+(xmax-xmin)*aa/1023;

return r;}
```

La lettura dei parametri viene eseguita attraverso la chiamata alla funzione *leggi_porta*. Nella parentesi sono definiti i parametri passati alla routine nella sua chiamata: la porta analogica dalla quale acquisire il valore di tensione, il minimo ed il massimo valore della grandezza risultante. Quindi a zero volt corrisponderà il valore minimo e a 5 volt, che corrisponde al valore di 1023 riportato da ARDUINO (convertitore a 10 bit), corrisponderà il valore massimo. Valori intermedi seguiranno la legge lineare.

Esecuzione della regolazione.La routine LOOP svolge tutta l'elaborazione. Se vi sono dei dati inviati dal PC: **if (Serial.available()>0)** il software acquisisce i dati : **leggi_x()** , svolge il tipo di regolazione ed invia i dati al PC.

La routine LOOP

```
void loop()

{ int y;

if (Serial.available()>0)

{ leggi_x();

if (tiporeg==1){azione_onoff();}

if (tiporeg==2){azione_propor();}

if (tiporeg==3){azione_pi();}

if (tiporeg>1) {

y=aztot*255/100;// trasforma l'azione di regolazione dal valore max di 100, in percentuale nel valore massimo dell'uscita di ARDUINO=255

analogWrite(uscita,y);// replica la stessa uscita sui 2 attuatori

analogWrite(uscita2,y);}

inviadati();

}
```

Se la regolazione è ON-OFF l'attuatore non è più modulante e l'uscita sulla porta è solo digitale e non analogica.

L'azione ON-OFF

```
if (rdiretta==true){

if (xv>setpoint+bandap/2) {digitalWrite(uscita,LOW);stato=false;};

if (xv<setpoint-bandap/2) {digitalWrite(uscita,HIGH);stato=true;};

}

if (rdiretta==false){

if (xv>setpoint+bandap/2) {digitalWrite(uscita,HIGH);stato=true;};

if (xv<setpoint-bandap/2) {digitalWrite(uscita,LOW);stato=false;};

}
```

L'implementazione dell'azione regolante va approfondita. Le regolazioni On-OFF e P sono regolazioni in cui l'azione regolante è direttamente collegata al valore attuale della grandezza fisica letta. Non è così per la regolazione PI. L'azione integrale è legata all'area complessiva sottesa dalla funzione temperatura ed il setpoint, nel piano temperatura-tempo(figura 8). Ne consegue che l'azione integrale dipende dai valori precedenti della grandezza fisica e dal tempo. L'area può essere ricondotta ad una somma di rettangoli aventi tutti la stessa larghezza, l'intervallo di tempo, e aventi come altezze il valore istantaneo della grandezza fisica .

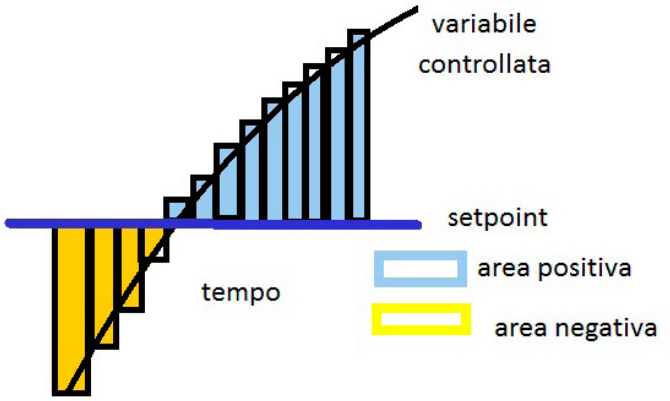


Figura 8 : significato dell'azione integrale

Per l'intervallo di tempo fra un campionamento e l'altro della grandezza fisica va fatto un discorso a parte. Se non vi fosse la comunicazione con il PC , ritenendo trascurabile il tempo di elaborazione, con il comando delay(1000) si potrebbe assegnare un ritardo temporale di 1 secondo od altro definibile, fra un campionamento e l'altro. La comunicazione con il PC si svolge in modo asincrono: il PC invia, ad intervalli regolari un byte sulla porta seriale di ARDUINO che è in attesa if (Serial.available>0). Quando ARDUINO riceve il byte svolge l'elaborazione ed invia i dati al PC. La fase di invio dati è quella relativamente più lunga. Non è quindi noto a priori la durata dell'intervallo di tempo, e, d'altro canto questo tempo è correlato al tempo integrale definito nel settaggio dei parametri. Il comando millis restituisce il tempo, in millisecondi dall'accensione della scheda. La differenza di lettura fra due campionamenti successivi è l'intervallo di tempo cercato. **Le parti di programma riportate sono le più significative e comunque in allegato trovate il software completo**. In questa sede non è possibile una descrizione dettagliata della sintassi del linguaggio software di ARDUINO, una versione di C++, che è possibile approfondire al sito di ARDUINO.

L'ACQUISIZIONE DEI DATI

I dati vengono acquisiti dal PC che li visualizza a schermo(figura 9) ad intervalli di 2 secondi. Tramite l'apposito bottone i dati fino a qui acquisiti vengono inviati ad un file EXCEL, ed è quindi possibile costruire un grafico(figura 10). L'andamento irregolare della temperatura è dovuto all'errore introdotto dall'acquisizione ADC. Il sistema comunque si assesta infine al valore di setpoint. Nella tabella 1 un campione di dati. L'azione proporzionale è legata allo scostamento ed è 50% quando la temperatura è uguale a quella di setpoint, zero quando la temperatura è setpoint+banda proporzionale/2, 100 quando la temperatura è setpoint-banda proporzionale/2. L'azione integrale è legata alla somma delle aree sottese moltiplicata per una costante, dipendente dal tempo integrale introdotto come parametro. La costante è negativa nel caso di regolazione diretta e positiva nel caso di regolazione inversa. Quando la temperatura è maggiore del setpoint il contributo all'area totale è positivo e quindi l'azione integrale diminuisce e viceversa. La figura 11 riporta il grafico della temperatura al variare del setpoint. La figura 12 visualizza altro grafico, ma in questo caso la sonda è una sonda digitale DS18B20. La sonda digitale comunica ad Arduino il valore di temperatura e non una tensione. Pertanto il segnale è "pulito" e si nota l'assenza del frastagliamento del grafico della temperatura.

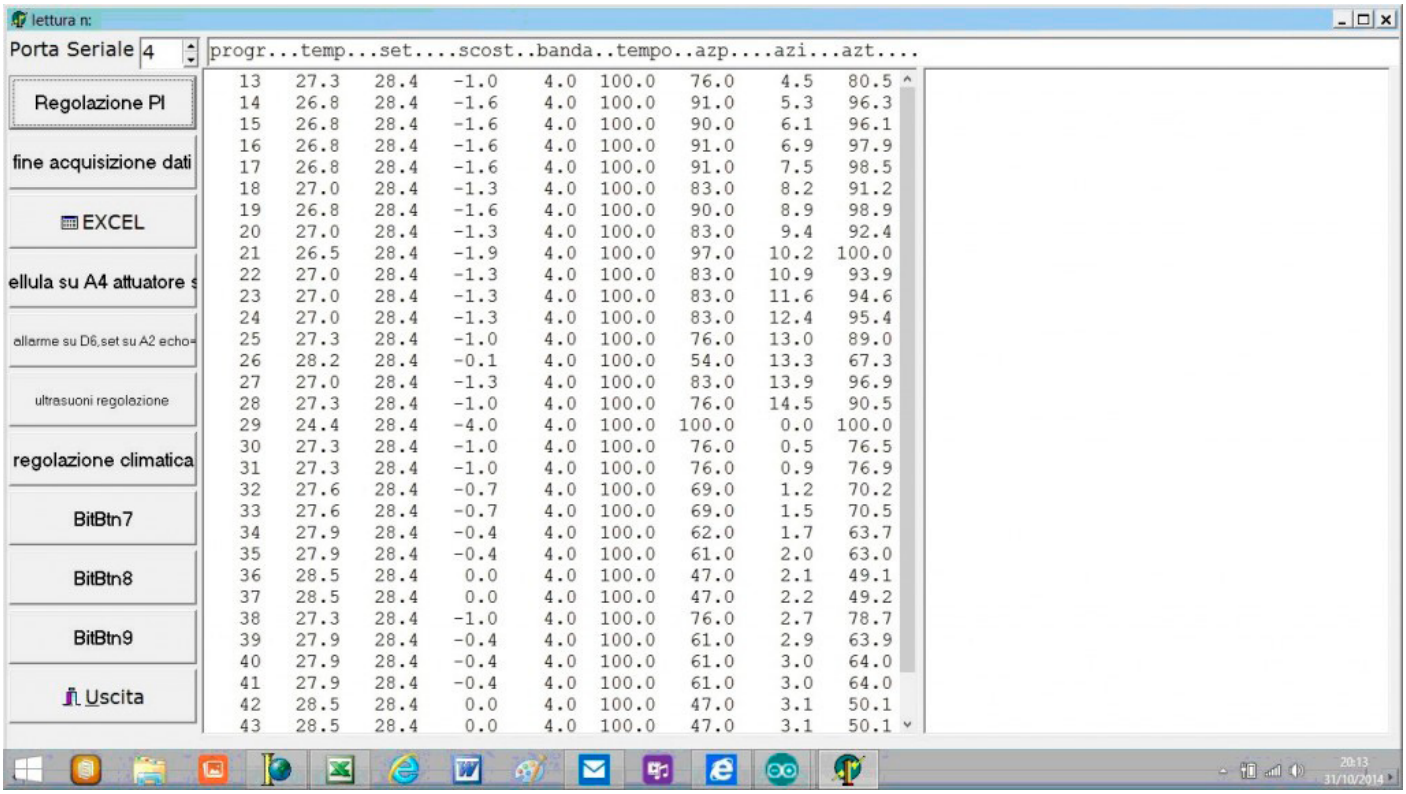


Figura 9: La videata su DELPHI

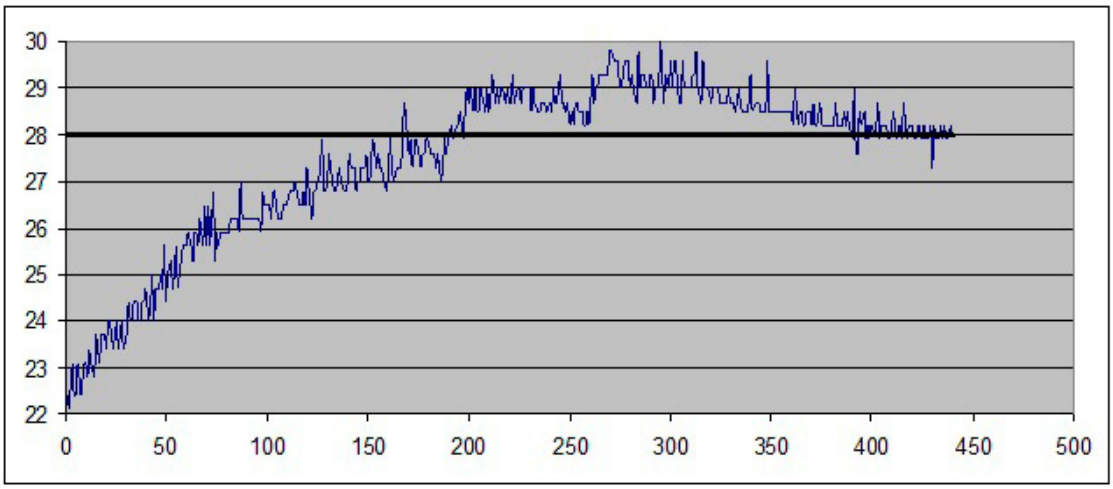


Figura 10:Il grafico in EXCEL dei dati acquisiti da DELPHI . In ascissa il tempo in secondi, in ordinata la temperatura

Temperatura	Scostamento	Azione proporzionale	Azione integrale	Azione totale
27,6	-0,3	58	33,1	91,1
27,9	0	51	33,1	84,1
27,6	-0,3	58	33,7	91,7
27,3	-0,6	65	33,9	98,9
27,6	-0,3	58	34	92
27,3	-0,6	65	34,5	99,5
27,9	0	51	34,5	85,5

27,6	-0,3	58	34,5	92,5
27,9	0	51	34,5	85,5
28,2	0,2	44	34,4	78,4
28	0	51	34,5	85,5
28,2	0,2	44	34,2	78,2
28,5	0,5	37	33,9	70,9
27,9	0	51	33,7	84,7
28,5	0,5	37	33,5	70,5

setpoint=28°C

banda proporzionale=4°C

tempo integrale=100 sec

Tabella 1: campione di dati

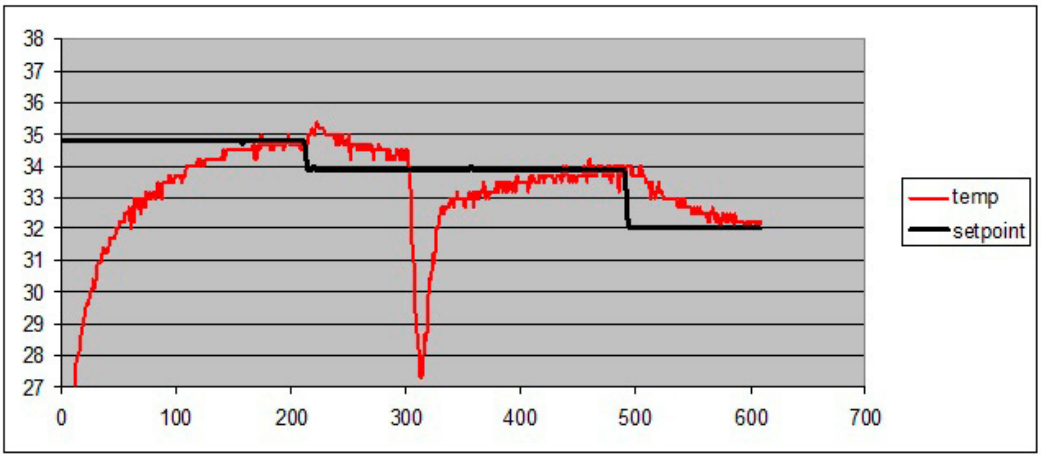


Figura 11: andamento della temperatura con variazioni del setpoint. Il picco in basso corrisponde all'apertura del coperchio della scatola ed il conseguente brusco raffreddamento. In ascissa il tempo in secondi, in ordinata la temperatura

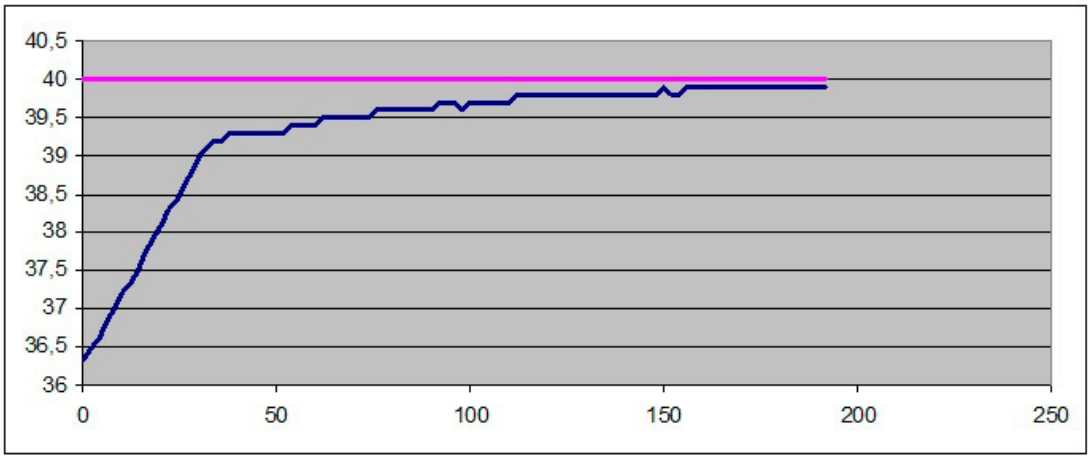


Figura 12:grafico della temperatura con sonda DS18B20. In ascissa il tempo in secondi, in ordinata la temperatura

L'ACQUISIZIONE DEI DATI

Arduino dimostra affidabilità sia in termini di hardware che di software e non è un semplice giocattolo per hobbisti. Una limitazione hardware è l'inserimento dei collegamenti, cui si può ovviare con schede shield che sostituiscono i connettori di Arduino con morsettiere. Le applicazioni possono pertanto essere allargate a controllo di apparecchiature e sensori di tipo semiprofessionale.

ALLEGATO

regolazione

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/la-regolazione-di-temperatura-con-arduino

MEMS, dalla teoria alla pratica: usiamo un Accelerometro con Arduino

di Piero Boccadoro

Come detto in apertura, questa è solo la seconda parte dell'intervento e così come detto la scorsa volta, lo scopo doveva essere quello di far vedere, applicati, in maniera pratica e concreta i concetti che la teoria ha permesso di trasformare in sistemi. Quello che vedremo, tra un attimo, è la prova, il test di un accelerometro che è stato configurato ed utilizzato con Arduino.

Prima di proseguire, però, vorrei fare un breve

ACCENNO ALLE NANOTECNOLOGIE

Questo argomento è certamente molto vasto ed è impossibile trattarlo in un unico articolo. Ci vorrebbe un corso dedicato. Però quello che è importante sottolineare è che esiste ed è concreta la possibilità di realizzare strutture submicrometriche che lavorino in diversi campi, secondo diversi principi e soprattutto inseguendo le caratteristiche più spinte che riusciamo ad immaginare.

La tecnologia del silicio in questi casi risulta quanto mai inefficace dal momento che le dimensioni sono troppo piccole perché si possa sempre pensare di utilizzare questo materiale; esistono processi più idonei, più precisi e con delle rese assolutamente più interessanti.

Esistono, altresì, materiali più performanti e più adatti a questo genere di esigenze ed applicazioni.

In questo panorama che si presenta sfaccettato e complesso si inseriscono perfettamente i nanotubi di carbonio. Ed è proprio di questo che abbiamo accennato a Firenze.

Esistono le cosìddette forme "allotropiche".

Di allotropi del carbonio sono conosciuti alcuni tipi: carbonio amorfo, grafite, diamanti e fullereni.

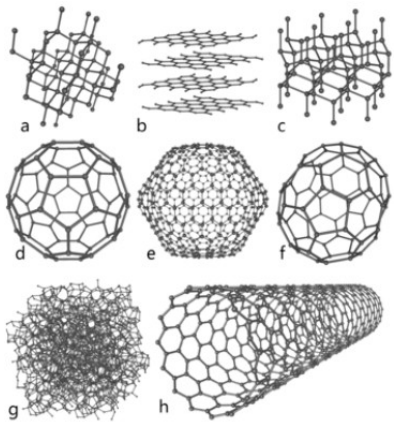
Quando parliamo della sua forma amorfa, ci riferiamo essenzialmente alla grafite nella quale rimane inalterata la struttura cristallina.

A pressione normale il carbonio prende forma di grafite; qui ogni atomo è legato ad altri tre come possiamo vedere più avanti.

A pressioni molto alte il carbonio forma un allotropo chiamato diamante. Qui ogni atomo è legato ad altri quattro. Tutti sappiamo che la forza del legame tra gli atomi all'interno di questa struttura rappresenta sostanzialmente il più solido dei legami possibili e conosciuti.

Pressione e temperatura, quindi, governano le transizioni tra una configurazione dell'altra, portando alla formazione di legami più o meno intensi e alla descrizione di geometrie e strutture cristalline sempre più complesse.

I tipi sono descritti nella seguente immagine:



dove:
a) diamante;
b) grafite;
c) lonsdaliete;
d) fullerene C60;
e) fullerene C540;
f) fullerene C70;
g) amorfo;
h) nanotubi.

I nanotubi sono particolari fullereni che dimostrano di avere un diametro estremamente piccolo ed un rapporto lunghezza/diametro pari a diversi ordini di grandezza. La struttura è qualitativamente simile alla grafite ma invece della configurazione esagonale sono globalmente pentagonale tranne all'inizio ed alla fine. Lì si hanno strutture a forma di cupola che riprendono la struttura del carbonio C60.

Lungi dall'essere state completamente analizzati, le proprietà dei fullereni restano ancora tutte da scoprire. Ciò nonostante, diversi esperimenti li indicano come collocati direttamente sui GATE dei MOSFET di nuova generazione. Tale idea non è l'unica a vederli impiegati perché le nanotecnologie, più in generale, strizzano l'occhio a diverse grandezze e possibilità tra cui il computer quantistico, le strutture quantum confined ed i dispositivi basati sull'effetto tunnel risonante.

VEDIAMONE SUBITO UNO

Per provare ad essere più pratico possibile, eccovi un esempio di accelerometro.

L'LSM303 è un accelerometro tri-assiale tilt compensated:

- **Tensione di alimentazione (analogica): 2.5 V to 3.3 V**
- **Tensione di alimentazione (digitale) IOs: 1.8 V**
- **Power-down mode;**
- **3 canali per magnetic field e 3 canali per accelerazione;**
- **fondo scala del campo magnetico (in Gauss): da ±1.3 a**
- **±8,1;**
- **fondo scala selezionabile dinamicamente tra: ±2g, ±4g e**
- **±8g;**
- **profondità di dato: 16-bit;**
- **interfaccia I2C;**
- **2 generatori indipendenti e programmabili per rilevazione**
- **free-fall e motion detection;**
- **embedded self-test;**
- **funzine sleep-to-wakeup (accelerometro);**
- **rilevazione dell'orientazione 6D.**

Il motivo per cui è stato scelto questo accelerometro è presto detto: si tratta di un componente che conosco perché ce l'ho disponibile, l'ho studiato e col quale ho lavorato. Inoltre si tratta di un dispositivo straordinariamente coerente con l'argomento di oggi. Inoltre ve ne parlo perché è dotato di un'interfaccia di comunicazione molto comoda, l'I2C.

L'altra volta parlavamo di campi applicativi quindi, entrando nello specifico, diciamo che il particolare modello di accelerometro che abbiamo indicato in precedenza, così come indica la documentazione ufficiale, è molto indicato per: "Compensated compassing", "Map rotation", "Position detection", "Motion-activated functions", "Free-fall detection", "Intelligent power-saving for handheld devices", "Display orientation", "Gaming and virtual reality input devices", "Impact recognition and logging", "Vibration monitoring and compensation".

UN CASO PRATICO

Come abbiamo detto, specifiche e campi appli-

cativi vanno di pari passo. Nel recente passato mi sono occupato delle applicazioni spaziali. **Le specifiche che avevo sono riportate per intero in questo documento di Thales Alenia Aerospace.**

Stiamo immaginando di utilizzare un accelerometro che sia anche magnetometro, giroscopio. E questo perché abbiamo bisogno di un sistema che sia in grado di darci **il maggior numero di informazioni possibili e diventare il cuore pulsante di una Inertial Measurements Unit (IMU).** Bene, possiamo pensare che tra i campi applicativi di cui abbiamo parlato in precedenza ci sia una che conosciamo molto bene, ovvero la funzionalità di orientazione del display, che su tutti gli smartphone è ampiamente implementata.

Come si fa? Che cosa ci serve? Noi vogliamo fare in modo da leggere il valore in uscita dal sensore per rilevare che lo spostamento è avvenuto.

Uno spostamento qualsiasi? Ovviamente no! Perché se fosse uno spostamento qualsiasi, la prospettiva potrebbe non essere corretta e a quel punto nascerebbero altre domande, per esempio: “ha senso ruotare lo schermo di 90° quando l’osservatore non ha fatto una richiesta di questo spostamento?” Quindi supponiamo di fissare la soglia a 90°.

Per fare questo, ben poco delle caratteristiche del dispositivo sarà utile: per esempio ci servirà sicuramente la tensione operativa, il range full-scale nonché la precisione. Bene, andiamo a vedere che cosa ci dice il datasheet del componente.

GUARDIAMO IL DATASHEET

Qualunque sia il componente, a prescindere dall’applicazione, in qualunque campo vi stiate muovendo, conoscere il dispositivo sarà la chiave verso il successo. Imparare a comprendere le caratteristiche del dispositivo è ovviamente, anche per questo, anche in questo caso, valgono tutte le considerazioni fatte in precedenza.

Particolare attenzione va posta rispetto alla precisione: quando diciamo che un dato viene rappresentato con un certo numero di cifre significative, specie all’uscita di un sensore che propone già un’uscita digitale, significa che quello è il numero teorico massimo di bit grazie ai quali il numero può essere rappresentato.

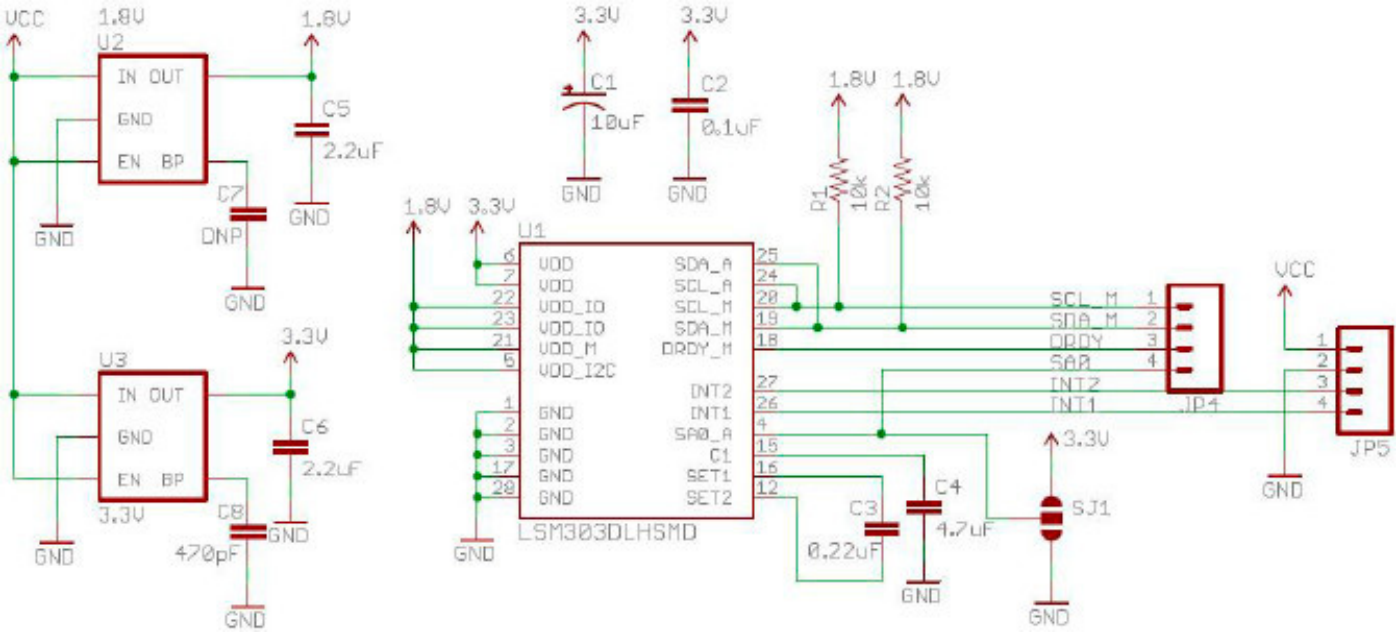
Questa indicazione, però, bisogna ricordarlo sempre, non è da prendere come verbo perché quella precisione ha sicuramente dei fattori limitanti, tra i quali le componenti di offset piuttosto che i rumori piuttosto che le vibrazioni casuali e quindi una serie di componenti che vanno a rendere meno significative specialmente le ultime cifre.

Ecco per quale motivo quello che abbiamo elencato in precedenza non è certamente tutto e solo quello che serve perché c’è molto di più.

Successivamente col dispositivo ci dobbiamo anche”parlare” e per farlo ci interessa molto sapere come dobbiamo comunicare, Quale interfaccia utilizzare, o quale delle interfacce disponibili è il caso di utilizzare, specie in funzione del resto del sistema. Ma anche il tipo di uscita, se analogica o digitale, e così via dicendo.

Dopodiché ancora non abbiamo finito perché serve sapere come si effettua il montaggio e per questo bisogna partire da quello che è riportato nella documentazione riguardo il package.

Onde evitare problematiche di qualunque tipo, bisogna tenere a mente che esistono anche altre specifiche.



LA BOARD

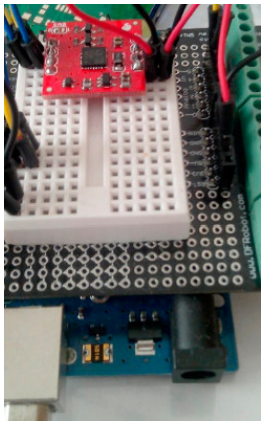
Ma adesso arriviamo al punto in cui questo intervento diventa pratico.

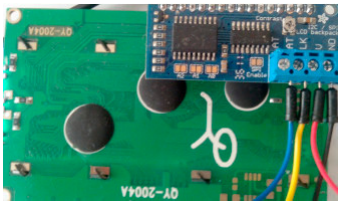
Vi faccio vedere che cosa serve per riuscire ad interfacciarsi con questo tipo di strumenti (prima) e per utilizzarli (poi) grazie anche a schede piuttosto semplici.

Per fare questa prova ho utilizzato Arduino, in maniera tale da fare tutto velocemente, anche e soprattutto per voi, per non appesantirvi.

Per comodità, l'accelerometro è stato già configurato all'interno di un sistema così fatto.

Avevo un'altra possibilità e cioè interfacciarmi direttamente con un accelerometro montato su una scheda della ST però magari questo ve lo faccio vedere la prossima volta.





Qui c'è poco da commentare se non che:

LSM303 ---------- Arduino

Vin	3.3V
GND	GND
SDA	A4
SCL	A5

Il display LCD è connesso tramite I2C (VCC = 5V).

Il codice

Tutti conoscete certamente l'IDE di Arduino e sapete anche come si creano le librerie.

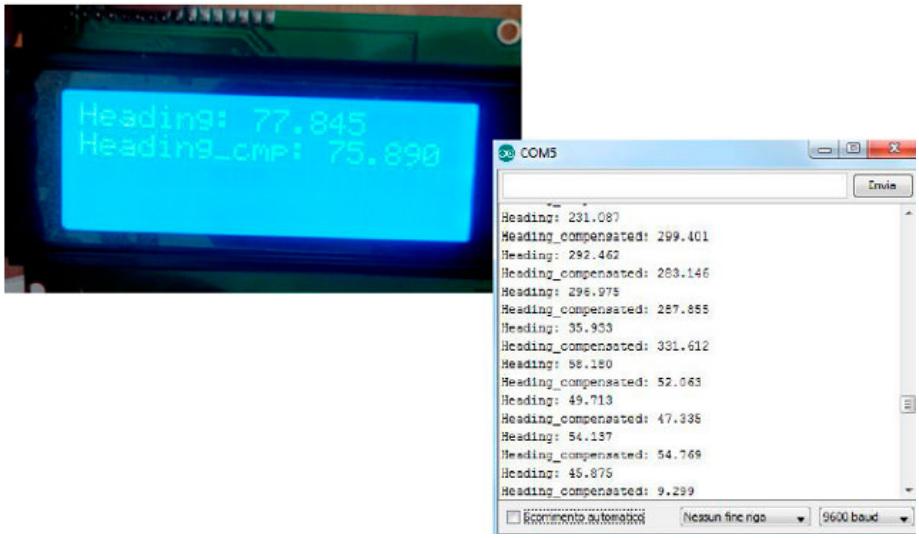
Questa però deve essere una simulazione snella per cui più che creare una libreria ho fatto in modo da scrivere una breve documentazione del componente, ovviamente consultando attentamente il datasheet. Ho scritto soltanto quello che serve nell'immediato per cui non prendetela come una documentazione completa.

Se la cosa vi interessa, il listato è riportato qui e potete simularlo voi stessi.

```
#define SCALE 2 // valore full scale dell'accelera-
zione (valori possibili: 2, 4 o 8)
/* Definizione indirizzo LSM303 */
#define LSM303_MAG 0x1E // SA0 grounded
#define LSM303_ACC 0x18 // SA0 grounded
#define X 0
#define Y 1
#define Z 2
/* Variabili globali */
int accel[3]; // valori grezzi dell'accelerazione
int mag[3]; // valori grezzi del magnetometro
float realAccel[3]; // valori dell'accelerazione calco-
lati

/* Definizioni dei registri dell'LSM303 */
#define CTRL_REG1_A 0x20
#define CTRL_REG2_A 0x21
#define CTRL_REG3_A 0x22
#define CTRL_REG4_A 0x23
#define CTRL_REG5_A 0x24
#define HP_FILTER_RESET_A 0x25
#define REFERENCE_A 0x26
#define STATUS_REG_A 0x27
#define OUT_X_L_A 0x28
#define OUT_X_H_A 0x29
#define OUT_Y_L_A 0x2A
#define OUT_Y_H_A 0x2B
#define OUT_Z_L_A 0x2C
#define OUT_Z_H_A 0x2D
#define INT1_CFG_A 0x30
#define INT1_SOURCE_A 0x31
#define INT1_THS_A 0x32
#define INT1_DURATION_A 0x33
#define CRA_REG_M 0x00
#define CRB_REG_M 0x01
#define MR_REG_M 0x02
#define OUT_X_H_M 0x03
#define OUT_X_L_M 0x04
#define OUT_Y_H_M 0x05
#define OUT_Y_L_M 0x06
#define OUT_Z_H_M 0x07
#define OUT_Z_L_M 0x08
#define SR_REG_M 0x09
#define IRA_REG_M 0x0A
#define IRB_REG_M 0x0B
#define IRC_REG_M 0x0C

/* Definizione delle funzioni */
void initLSM303(int fs)
{
 LSM303_write(0x27, CTRL_REG1_A); // 0x27 =
  modalità normale, tutti gli assi attivi
  if ((fs==8)||(fs==4))
  LSM303_write((0x00 | (fs-fs/2-1)<<4),
  CTRL_REG4_A); // imposta full-scale
 else
  LSM303_write(0x00, CTRL_REG4_A);
 LSM303_write(0x14, CRA_REG_M); // 0x14 =
 output rate mag 30Hz
  LSM303_write(0x00, MR_REG_M); // 0x00 =
 continouous conversion mode
}

void printValues(int * magArray, int * accelArray)
{
 /* stampa dei valori "come sono" */
 Serial.print(accelArray[X], DEC);
 Serial.print("\t");
 Serial.print(accelArray[Y], DEC);
 Serial.print("\t");
 Serial.print(accelArray[Z], DEC);
 Serial.print("\t\t");
 Serial.print(magArray[X], DEC);
 Serial.print("\t");
 Serial.print(magArray[Y], DEC);
 Serial.print("\t");
 Serial.print(magArray[Z], DEC);
 Serial.println();
 delay(1000);
}

float getHeading(int * magValue) {
 // paragrafo 1.2 dell'Application Note AN3192
 // pitch e roll = 0 => level position
 float heading = 180*atan2(magValue[Y],
magValue[X])/PI;
 if (heading <0)
  heading += 360;
 return heading;
}

float getTiltHeading(int * magValue, float * accel-
Value) {
 // Appendice A dell'Application Note AN3192
 float pitch = asin(-accelValue[X]);
 float roll = asin(accelValue[Y]/cos(pitch));
 float xh = magValue[X] * cos(pitch) +
magValue[Z] * sin(pitch);
 float yh = magValue[X] * sin(roll) * sin(pitch) +
magValue[Y] * cos(roll) -
  magValue[Z] * sin(roll) * cos(pitch);
 float zh = -magValue[X] * cos(roll) * sin(pitch) +
```

```
magValue[Y] * sin(roll)
    + magValue[Z] * cos(roll) * cos(pitch);
  float heading = 180 * atan2(yh, xh)/PI;
  if (yh >= 0)
    return heading;
  else
    return (360 + heading);
}

void getLSM303_accel(int * rawValues)
{
  rawValues[Z] = ((int)LSM303_
read(OUT_X_L_A) << 8) | (LSM303_
read(OUT_X_H_A));
  rawValues[X] = ((int)LSM303_
read(OUT_Y_L_A) << 8) | (LSM303_
read(OUT_Y_H_A));
  rawValues[Y] = ((int)LSM303_
read(OUT_Z_L_A) << 8) | (LSM303_
read(OUT_Z_H_A));
  // assegnamento corretto in funzione degli assi
}
void getLSM303_mag(int * rawValues)
{
  Wire.beginTransmission(LSM303_MAG);
  Wire.write(OUT_X_H_M);
  Wire.endTransmission();
  Wire.requestFrom(LSM303_MAG, 6);
  for (int i=0; i<3; i++)
    rawValues[i] = (Wire.read() << 8) | Wire.read();
}

// Lettura del sensore
byte LSM303_read(byte address)
{
  byte appo; // questa è solo una variabile di appog-
gio
  if (address >= 0x20)
    Wire.beginTransmission(LSM303_ACC);
  else
    Wire.beginTransmission(LSM303_MAG);
  Wire.write(address);
  if (address >= 0x20)
    Wire.requestFrom(LSM303_ACC, 1);
  else
    Wire.requestFrom(LSM303_MAG, 1);
  while(!Wire.available());
  appo = Wire.read();
  Wire.endTransmission();
  return appo;
}
```

```
// Scrittura dei dati
void LSM303_write(byte data, byte address)
{
  if (address >= 0x20)
    Wire.beginTransmission(LSM303_ACC);
  else
    Wire.beginTransmission(LSM303_MAG);
  Wire.write(address);
  Wire.write(data);
  Wire.endTransmission();
}

void setup() {
  Serial.begin(9600);
  Wire.begin();
  lcd.begin(20,4);
  lcd.setBacklight(HIGH);
  initLSM303(SCALE);
}

void loop() {
  getLSM303_accel(accel);
  while(!(LSM303_read(SR_REG_M) & 0x01));
  getLSM303_mag(mag); // prelevo i valori e li
salvo nell'array mag
  for (int i=0; i<3; i++) {
    realAccel[i] = accel[i] / pow(2, 15) * SCALE; //
calcolo i valori reali, in unità [g]
  }
  /* Stampa del level e dell'heading tilt-compensa-
ted */
  Serial.print("Heading: ");
  Serial.println(getHeading(mag), 3); // funzione
solo se il sensore è a livello
  Serial.print("Heading_compensated: ");
  Serial.println(getTiltHeading(mag, realAccel), 3);
// effetto della compensazione
  // e poi faccio le stesse cose sull'LCD, visto che la
seriale serviva solo di debug
  lcd.print("Heading: ");
  lcd.print(getHeading(mag), 3);
  lcd.setCursor(0,1);
  lcd.print("Heading_cmp: ");
  lcd.print(getTiltHeading(mag, realAccel), 3);
  delay(500);
  lcd.clear();
}
```



Il codice funziona in maniera tale da implementare soltanto poche funzioni, come dicevo, quelle che servono.

TEST

E per il ciclo “un’immagine vale più di mille parole”:

CHIUSURA

Bene, con questo abbiamo concluso.
Quello che c’è da capire e da sapere sui MEMS certamente non potevo presentarvelo per intero però spero di avervi dato un’idea: l’elettronica analogica e la programmazione rappresentano un tutt’uno abbastanza completo e solidale.
Esistono una serie di dispositivi che possono essere utilizzati per applicazioni tra le più disparate. Dalla robotica fino alla domotica, dalle applicazioni biomedicali fino alla dosimetria, dalle applicazioni consumer fino all’avionica.
Tutto dipende, ovviamente, dal vostro budget e da che cosa volete fare, purtroppo esattamente in quest’ordine.

Riferimenti:

MEMS
MEMS Book
MEMX
MPCollab
Appunti e dispense del corso di Sistemi Micro e Nano Elettronici della prof.ssa Caterina Cimiinelli (Politecnico di Bari)
Fonti consultate il: 10/05/2013

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/mems-dalla-teoria-alla-pratica-usiamo-accelerometro-con-arduino

Temperature & pressure monitoring con Arduino

di Piero Boccadoro

Per me è sempre stata una sfida interessante: misurare la temperatura all'interno di una stanza, vedere come vari nel tempo e cercare di caratterizzare tutte le stanze di un'abitazione in funzione del fatto che sia effettivamente comodo e possibile abitarle durante tutte le stagioni.

Credo sarebbe molto utile per tutti coloro che vogliono capire, per esempio, come migliorare l'efficienza energetica della propria casa, adibendo ciascuna stanza all'uso più opportuno in funzione proprio di queste caratteristiche, magari scoprendo che non è nemmeno necessario modificare le proprie abitudini. La distribuzione degli spazi, dei mobili, la vivibilità, più in generale, potrebbero trasformare la "casa di sempre" in una esperienza abitativa diversa, magari migliore.

Questa sfida, in realtà, si può affrontare con un sistema abbastanza più complesso di quello che vedremo oggi ma che, certamente, si può realizzare anche grazie a quello che vi faremo vedere, utilizzandolo come base.

IL SENSORE

Stiamo parlando di un sensore barometrico, il BMP085 della Bosch, che è un dispositivo di grande precisione ed a basso consumo di potenza. Il range di misura va da 300 fino a 1100 hPa con una precisione assoluta pari a 0.03 hPa. Per la massima chiarezza, ricordiamo che

1 hPa = 1 mbar

1 Pa = 1 N m^-2 = 1 kg m^-1 s^-2

La tecnologia sulla quale è basato permette una ottima linearità, elevata precisione ma, soprattutto resistenza alle interferenze elettromagnetiche. Altra caratteristica di grande interesse è la stabilità sul lungo periodo. Non viene, infatti, evidenziata alcuna sostanziale variazione dei parametri funzionali del dispositivo in funzione del tempo di utilizzo.

Il sensore può lavorare con tensioni di polarizzazione comprese tra 1.8 V e 3.6 V.

È stato progettato per essere connesso direttamente ad un microcontrollore che disponga di un ingresso I2C ed è, infatti, grazie a questo tipo di connessione che lo utilizzeremo per questo esperimento.

La scheda che viene impiegata permette di utilizzare un pin header da 6. VDDD e VDDA del sensore sono connesse insieme e portate ad un singolo pin; questa è una scelta che, in linea generale, potrebbe non essere il caso di fare ma che, per la nostra applicazione specifica, non dovrebbe creare particolari problematiche.

Se il sensore, invece, dovesse essere inserito all'interno di un sistema più complesso e che lavori con segnali misti, sarà certamente necessario distinguere l'alimentazione digitale da quella analogica.

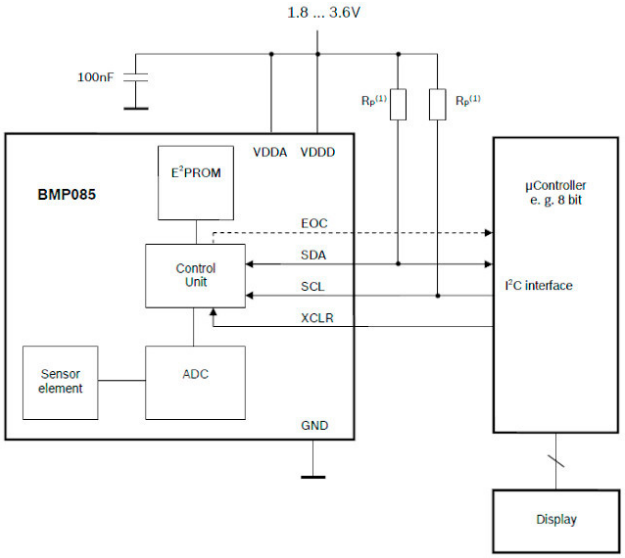
Concordemente a quanto vedremo, viene utilizzato un resistore da 4.7 kΩ come semplice rete di pull-up.

Tornando, brevemente, alle caratteristiche, tra le più interessanti ci sono:

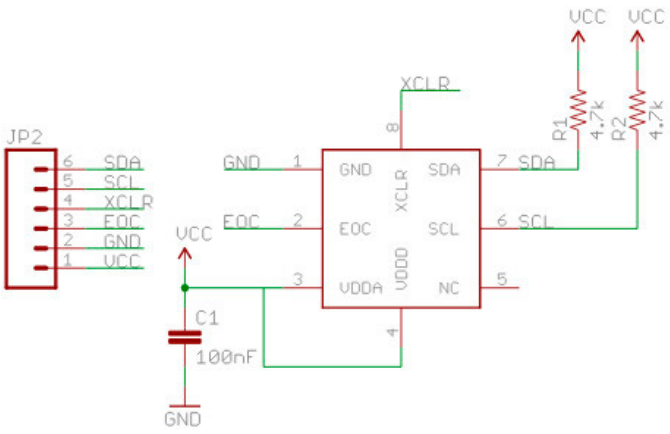
- interfaccia I2C;
- ampio range di pressione;
- diverse tensioni di alimentazione;

- basso consumo di potenza (5 uA @ 1 sample/s);
- basso rumore di misura;
- dimensioni estremamente contenute (16.5 x 16.5 mm).

Veniamo adesso alla configurazione; il datasheet dimostra subito il metodo di configurazione del sensore ed esso prevede l'utilizzo di semplice resistori e condensatori per poterlo rendere operativo. Ecco una figura esplicativa:



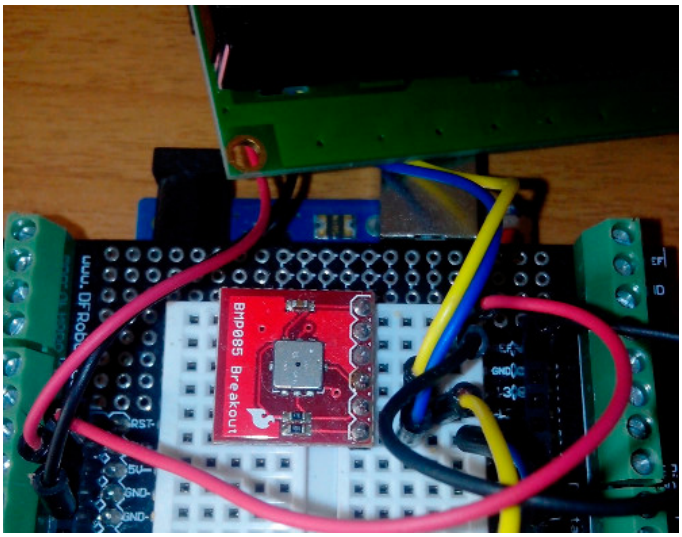
Rp, i resistori di pull-up, possono essere di vario valore, da 2.2 kΩ fino a 10 kΩ; valore tipico è 4.7 kΩ. Ecco, quindi, che, esattamente come riportato nel datasheet, sulla breakout board in uso, il valore della resistenza è lo stesso che vediamo nello schematico.



CONFIGURIAMOLO

Abbiamo deciso di lavorare con Arduino, versione UNO rev.3. Questa scheda la conoscete molto bene (ok, alzi la mano chi non l'ha mai vista...) per cui certamente non vi servono presentazioni (qualcuno non ha alzato la mano?). Tuttavia vale la pena di specificare che l'interfaccia di comunicazione I2C è possibile che venga utilizzata solo grazie ai pin A4 e A5.

Per farlo, i collegamenti dovranno essere quelli che vedete in figura (non fatevi impressionare, lo schield che ho usato NON era necessario ma lo trovo molto comodo per sperimentare al volo i prototipi!):

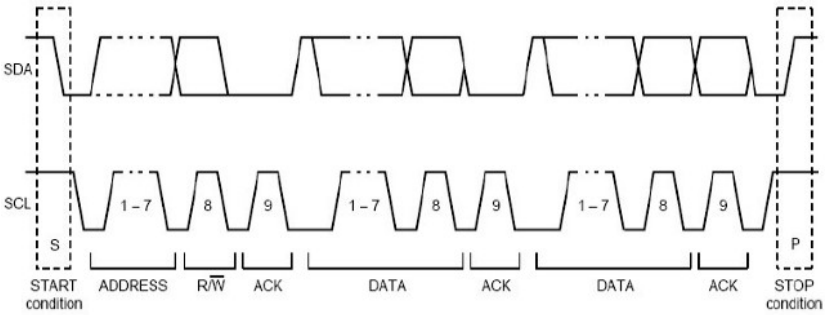


Possiamo utilizzare, per questo esperimento, anche un monitor LCD 20x4. Vedremo che sarà molto utile impiegare il Monitor Seriale con funzione di debug per verificare il corretto invio dei dati ma, naturalmente, scopo del progetto è quello di proporre i dati all'utente ed il monitor a cristalli liquidi rappresenta certamente il metodo più semplice per farlo.

Dal momento che disponiamo di un LCD backpack (che vi consiglio caldamente!) che consente il collegamento del monitor stesso tramite interfaccia I2C, la versione della libreria in uso sarà modificata e permetterà all'utilizzo delle classiche funzioni anche con questo metodo

di collegamento.

Vi rimandiamo, per completezza, all'indirizzo Internet presso il quale potrete reperire il datasheet del componente e la libreria modificata per le comunicazioni su I2C del display. Qui di seguito, invece, un'immagine riassuntiva del protocollo di comunicazione utilizzato.



Il codice è abbondantemente commentato (come avrete sicuramente notato) ma è importante farvi comprendere come queste funzioni siano, almeno in questo caso, figlie di una documentazione che dire completa è dir poco.

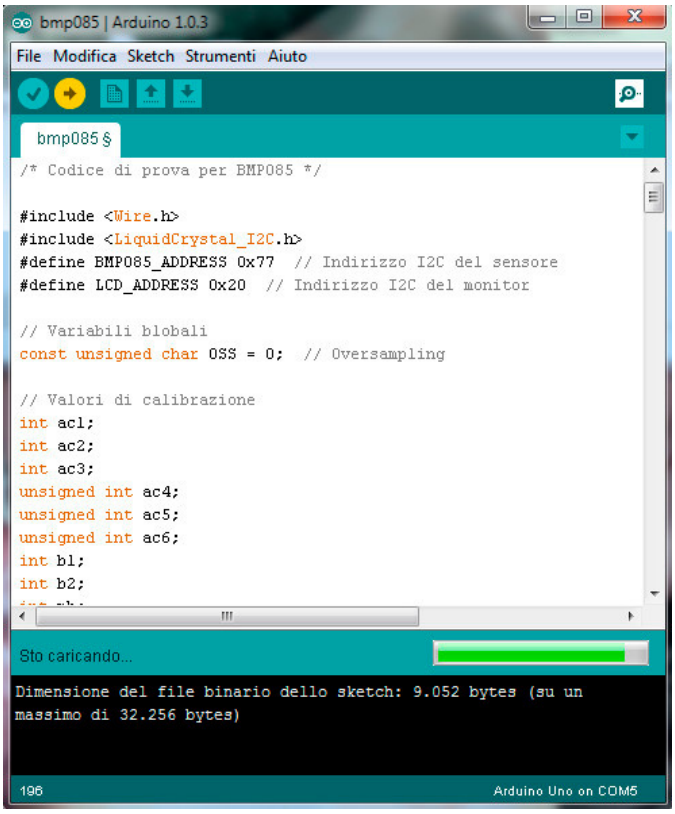
Volendo, inoltre, espandere le possibilità di questo codice una delle prime cose che si può certamente fare è quella di implementare il calcolo dell'altitudine, che può risultare dalla seguente formula:

$$altitude = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

IL CODICE

Dal momento che si tratta soltanto di una piccola dimostrazione e non già di un progetto completo, il codice che vi riportiamo qui di seguito non contiene una vera e propria libreria del componente ma soltanto le funzioni fondamentali che ci servono in questo caso.

Ecco come fare a farlo funzionare:



Viene qui, anche, tiportato che una variazione di 1 hPa corrisponde ad una variazione di altitudine di 8.43 m.

Come potete vedere dai collegamenti, in pratica monitor e sensore sono in parallelo, distinti solamente dal loro indirizzo così come si è visto nel codice.

CONCLUSIONI

L'esperimento che vi abbiamo fatto vedere quest'oggi è molto semplice. Si tratta di configurare il sensore ed effettuare una lettura periodica.

A tal proposito, trattandosi di temperatura, il periodo è da impostare ma naturalmente fare la lettura ogni secondo è altamente inutile: la temperatura è una variabile che varia molto lentamente nel tempo e che ha escursioni abbastanza contenute quindi anche una lettura ogni 6 ore è più che sufficiente!

La parte più complicata, evidentemente, è la realizzazione e l'implementazione delle funzioni utili per effettuare l'interfacciamento.

Per realizzare quel sistema più complesso del

```
#include
#include
#define BMP085_ADDRESS 0x77  // Indirizzo I2C del sensore
#define LCD_ADDRESS 0x20  // Indirizzo I2C del monitor

// Variabili blobali
const unsigned char OSS = 0;  // Oversampling

// Valori di calibrazione
int ac1;
int ac2;
int ac3;
unsigned int ac4;
unsigned int ac5;
unsigned int ac6;
int b1;
int b2;
int mb;
int mc;
int md;

long b5;

// variabili da misurare
short temperature;
long pressure;

// Dichiaro lo schermo LCD
LiquidCrystal_I2C lcd(LCD_ADDRESS, 20,4);

// Con questa funzione memorizzo tutti i valori nelle variabili globali
// I valori di calibrazione servono per calcolare temperature e pressioni
// La funzione va richiamata all'inizio del programma
void bmp085Calibration()
{
  ac1 = bmp085ReadInt(0xAA);
  ac2 = bmp085ReadInt(0xAC);
  ac3 = bmp085ReadInt(0xAE);
  ac4 = bmp085ReadInt(0xB0);
  ac5 = bmp085ReadInt(0xB2);
  ac6 = bmp085ReadInt(0xB4);
  b1 = bmp085ReadInt(0xB6);
  b2 = bmp085ReadInt(0xB8);
  mb = bmp085ReadInt(0xBA);
  mc = bmp085ReadInt(0xBC);
  md = bmp085ReadInt(0xBE);
}

// Calcolo della temperatura
// Valori misurati in decimi di °C (vedi datasheet)
short bmp085GetTemperature(unsigned int ut)
```

```
{
  long x1, x2;

  x1 = (((long)ut - (long)ac6)*(long)ac5) >> 15;
  x2 = ((long)mc << 11)/(x1 + md);
  b5 = x1 + x2;

  return ((b5 + 8)>>4);
}

// Calcolo della pressione
// Qui vengono utilizzati i valori di calibrazione
// I valori sono misurati in Pa
long bmp085GetPressure(unsigned long up)
{
  long x1, x2, x3, b3, b6, p;
  unsigned long b4, b7;

  b6 = b5 - 4000;
  // Calcolo di B3
  x1 = (b2 * (b6 * b6)>>12)>>11;
  x2 = (ac2 * b6)>>11;
  x3 = x1 + x2;
  b3 = (((((long)ac1)*4 + x3)<>2;

  // Calcolo di B4
  x1 = (ac3 * b6)>>13;
  x2 = (b1 * ((b6 * b6)>>12))>>16;
  x3 = ((x1 + x2) + 2)>>2;
  b4 = (ac4 * (unsigned long)(x3 + 32768))>>15;

  b7 = ((unsigned long)(up - b3) * (50000>>OSS));
  if (b7 < 0x80000000)
    p = (b7<<1)/b4;
  else
    p = (b7/b4)<<1;

  x1 = (p>>8) * (p>>8);
  x1 = (x1 * 3038)>>16;
  x2 = (-7357 * p)>>16;
  p += (x1 + x2 + 3791)>>4;

  return p;
}

// Lettura di 1 byte del sensore (indirizzo)
char bmp085Read(unsigned char address)
{
  unsigned char data;

  Wire.beginTransmission(BMP085_ADDRESS);
```

```
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 1);
  while(!Wire.available());

  return Wire.read();
}

// Lettura di 2 bytes del sensore, il primo è l'inidirizzo il secondo è indirizzo+1
int bmp085ReadInt(unsigned char address)
{
  unsigned char msb, lsb;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 2);
  while(Wire.available()<2);
  msb = Wire.read();
  lsb = Wire.read();

  return (int) msb<<8 | lsb;
}

// Lettura dei valori di temperatura (non compensati)
unsigned int bmp085ReadUT()
{
  unsigned int ut;

  // La scrittura di 0x2E nel registro 0xF4 è una lettura del dato
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x2E);
  Wire.endTransmission();

  delay(5); // pausa di 5 ms a fronte dei 4.5 richiesti

  // Lettura da 0xF6 e 0xF7
  ut = bmp085ReadInt(0xF6);
  return ut;
}

// Lettura dei valori di pressione
unsigned long bmp085ReadUP()
{
  unsigned char msb, lsb, xlsb;
  unsigned long up = 0;

  // Scrivere 0x34+(OSS<<6) nel registro 0xF4 vuol dire leggere la pressione
```

```
  // ATTENZIONE al sovracampionamento!
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x34 + (OSS<<6));
  Wire.endTransmission();

  delay(2 + (3<> (8-OSS);

  return up;
}

// Inizia il programma
void setup()
{
  Serial.begin(9600);
  Wire.begin();
  lcd.begin();
  bmp085Calibration();
}

void loop()
{
  temperature = bmp085GetTemperature(bmp085ReadUT())/10;
  pressure = bmp085GetPressure(bmp085ReadUP());

  //Iniziamo a visualizzare i dati
  Serial.print("Temperature: ");
  Serial.print(temperature, DEC);
  Serial.println(" °C");
  Serial.print("Pressure: ");
  Serial.print(pressure, DEC);
  Serial.println(" Pa");
  Serial.println("\r");
  lcd.setBacklight(HIGH);
  lcd.print("Temperature: ");
  lcd.print(temperature, DEC);
  lcd.println(" °C");
  lcd.print("Pressure: ");
  lcd.print(pressure, DEC);
  lcd.println(" Pa");
  cd.println(" m");
  delay(1000);
  lcd.clear();
}
```

quale avevamo accennato all'inizio saranno necessari almeno ancora un **RTC**, in modo tale da collegare le letture all'orario e quindi al periodo della giornata così come alla data, e quindi al periodo dell'anno.

Inoltre il sensore potrebbe non essere unico ma potremmo prelevare dati **dall'interno e dall'esterno** dell'appartamento, in maniera tale da correlare anche la "reazione" dell'ambiente all'esterno. Ovviamente per le misure esterne avrebbero bisogno di sensori più complessi perché dovrebbero essere, magari, **wireless** e quindi serve anche che siano alimentati e c'è da risolvere la questione del "**come**?". Evidentemente i sensori potrebbero diventare anche di più di due e quindi si potremmo dover gestire contemporaneamente dati provenienti da n elementi sensibili distinti, identificati tutti univocamente dal loro indirizzo.

Tutto questo non può essere completo se non si analizzano anche i venti cui l'edificio è esposto, che dipendono soprattutto dall'altezza dell'abitato.

Tutti questi dati, una volta raccolti, potrebbero essere memorizzati su una memoria locale ed ancora inviati ad un webserver.

Insomma, se si comincia davvero a pensare a quali possono essere gli sviluppi futuri per questo genere di sistema, arriviamo ad una stazione metereologica non più tanto in miniatura.

Un gran bel progetto partendo da così poco, vero?

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/temperature-pressure-monitoring-con-arduino

Pilotare motori passo-passo con Arduino

di Cristian Padovano

Un motore passo-passo, anche definito **stepper**, è un tipo di motore sincrono che può, a seconda degli avvolgimenti eccitati dalla corrente continua, dividere i suoi movimenti in passi (step). Questo tipo di motori non ha spazzole e, rispetto ai normali motori CC, ha il vantaggio di poter fare movimenti precisi, tenendo fermo l'albero con poco margine di errore. Ne esistono di due tipi, riconoscibili in base al numero di fili che indicano le fasi:

- **unipolari** (5, 6 o 8 fili)
- **bipolari** (4 o 8 fili)

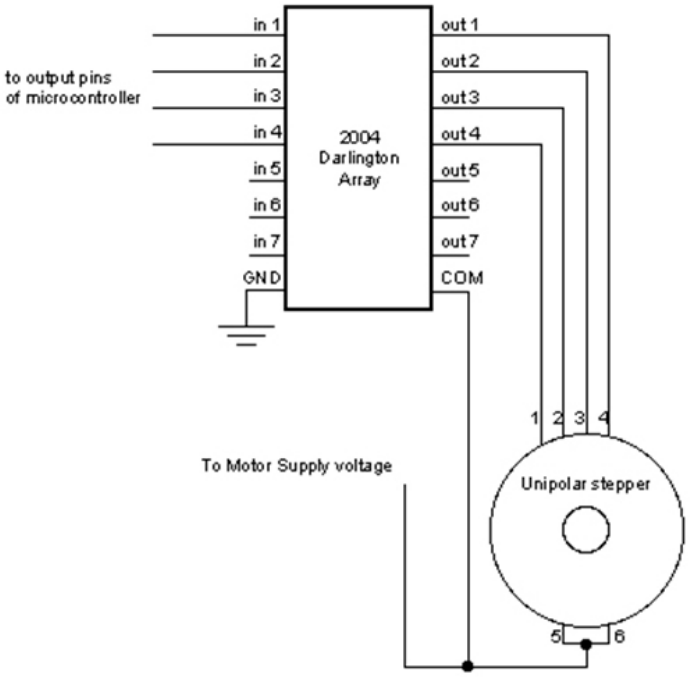
Nel nostro caso andremo ad utilizzare un motore unipolare del tipo **Nema 17** da 200 passi 0,44A. Il motore avrà una corrente di 0,44A perchè l'**integrato** che useremo potrà gestire una corrente massima di 0.5A.

L'integrato ULN2003A (qui trovate il suo datasheet) è composto da 7 transistor Darlington e, come è stato detto, può gestire una corrente massima di 0.5A e tensioni massime in uscita di 50V. **E' da sottolineare che per ogni motore che vogliamo usare bisognerà utilizzare un integrato: infatti, ULN2003A può gestire 1 motore stepper oppure 2 motori CC semplici**.

Passiamo ora allo schema elettrico da usare. Lo schema è questo proposto di seguito

Nell'immagine è indicato l'integrato 2004 ma il datasheet è lo stesso del 2003A quindi sono identici.

Dallo schema elettrico si può notare che verranno usati gli input 1, 2, 3, 4 per ricevere il segnale da Arduino e i rispettivi output (13, 14, 15 e 16) per collegare le 4 fasi del motore. Useremo poi il

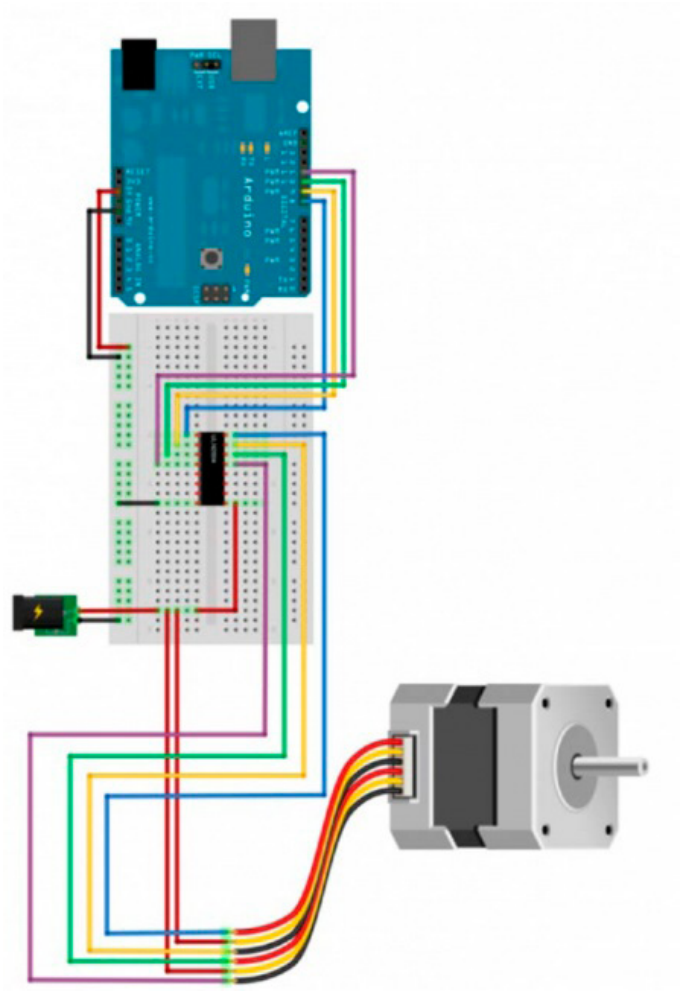


pin 8 a massa (GND) e il rispettivo (COM) come comune con il motore e l'alimentazione dello stesso. Per l'alimentazione, possiamo utilizzare una tensione supportata da entrambi oppure alimentare separatamente i due componenti, cioè utilizzare 5v per il Darlington ed un alimentatore a parte per il motore.

Per capire meglio i collegamenti generali e costruire un circuito di test, si può osservare la seguente immagine esplicativa

In questo caso si può notare come l'alimentazione sia presa esternamente, anche se è presente un collegamento con i 5V di Arduino. La fonte, però, come si può leggere dal datasheet dell'integrato, non deve superare i 30V in entrata.

Tenendo in considerazione che stiamo usando dei motori che funzionano a 5V 0.44A, si potrebbe anche usare la sola alimentazione che arriva da Arduino per entrambi i dispositivi in quanto il



computer riesce a pilotare entrambi senza causare sovraccarichi della porta USB.

Nello schema generale, i pin 8, 9, 10 e 11 di Arduino sono collegati con i pin 1, 2, 3 e 4 dell'integrato; i restanti pin dell'ULN2003A sono collegati come precedentemente descritto.

A questo punto, costruito il nostro circuito, possiamo passare ai test ma prima ancora serve effettuare la stesura del codice di Arduino. Sul sito ufficiale si possono trovare numerosi tutorial e parti di codice utili per qualsiasi evenienza. Prendiamo in esame il codice per pilotare un solo motore:

- le linee 23-26 indicano la corrispondenza tra la fase del motore e il pin al cui è collegata ad Arduino
- la linea 27 indica il ritardo di rotazione, in millisecondi, tra uno step e quello successivo
- le linee 29-34 impostano i pin di Arduino in modalità OUTPUT
- le linee 36 e 57 indicano la funzione di "loop" in modo da far compiere un ciclo continuo all'albero del motore
- le linee 37-40 indicano ad Arduino di invare un segnale HIGH sul pin 8. Quest'ultimo, collegato con l'ULN2003A, collegherà la fase corrispondente del motore facendolo avanzare. Le stesse righe indicano ad Arduino anche di inviare un segnale LOW ai pin dal 9 a 11, mentre la riga 40 impone di aspettare 500 ms prima di procedere con altre istruzioni
- le linee 42-56 impostano un segnale HIGH in successione ai pin 9, 10 e 11 attendendo, per ogni passo, un tempo di 500 millisecondi

Come si può vedere, pilotare un **motore stepper** non è molto complicato e il codice descritto può essere ampliato per pilorare più di un motore. Questo tipo di motori è molto utile per far muovere, nel caso di una stampante 3D, tutti i meccanismi, dalle barre filettate alle cinghie all'estrusore, per la loro **precisione** e per la **maggiore coppia motrice** che hanno rispetto ai normali motori a corrente continua. Inoltre questo tipo di motori non ha spazzole, a differenza dei normali motori, e quindi ha una maggiore vita. Uno svantaggio di questi però è l'eccessivo costo, ma si può ovviare a ciò cercando su siti di materiale usato, come ad esempio qui, oppure procedere in sano spirito **Open** e recuperare alcuni motori da vecchie stampanti Canon (dalla mia esperienza le Hp hanno sempre semplici motori CC) oppure da scanner.

```
/* Stepper Copal
 * -------------
 *
 * Program to drive a stepper motor coming from a 5'25 disk drive
 * according to the documentation I found, this stepper: "[...] motor
 * made by Copal Electronics, with 1.8 degrees per step and 96 ohms
 * per winding, with center taps brought out to separate leads [...]"
 * [http://www.cs.uiowa.edu/~jones/step/example.html]
 *
 * It is a unipolar stepper motor with 5 wires:
 *
 * - red: power connector, I have it at 5V and works fine
 * - orange and black: coil 1
 * - brown and yellow: coil 2
 *
 * (cleft) 2005 DojoDave for K3
 * http://www.0j0.org | http://arduino.berlios.de
 *
 * @author: David Cuartielles
 * @date: 20 Oct. 2005
 */

int motorPin1 = 8;
int motorPin2 = 9;
int motorPin3 = 10;
int motorPin4 = 11;
int delayTime = 500;

void setup() {
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
}

void loop() {
  digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);
  delay(delayTime);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);
  delay(delayTime);
}
```

BUONA PROGETTAZIONE!

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/pilotare-motori-passo-passo-con-arduino

Controllare I/O multipli con pochi pin di Arduino

di Luigi D'Acunto

La **scalabilità** è la capacità di un sistema di "crescere" o "decrescere" (aumentare o diminuire di scala) in funzione delle necessità e delle disponibilità.

A chi lavora o si diletta con i microcontrollori, con **Arduino** o semplicemente con le porte digitali, sarà sicuramente capitato almeno una volta di avvertire l'esigenza di realizzare un prototipo che goda di tale proprietà.

La necessità di rendere un sistema scalabile trova la sua massima espressione nei casi pratici in cui bisogna gestire un numero, in linea di principio imprecisato (seppur stimabile), di input e/o output (come ad esempio sensori ed attuatori).

Il buonsenso in questi casi porterebbe il progettista a cercare di quantificare a priori il numero di ingressi/uscite da gestire ed assegnare a ciascuno di questi un pin dell'unità di elaborazione che in questo articolo supporremo essere un **PIC**, oppure una Board di **Arduino**.

Quest'approccio tuttavia pone un severo **limite** al manufatto completo:

- *L'impossibilità di aggiungere un nuovo elemento di ingresso o uscita senza modificare l'hardware.*

I circuiti logici di cui quest'articolo si interesserà e che risolvono tale problema sono detti *Multiplexer* (**MUX**) e *Demultiplexer* (**DEMUX**).

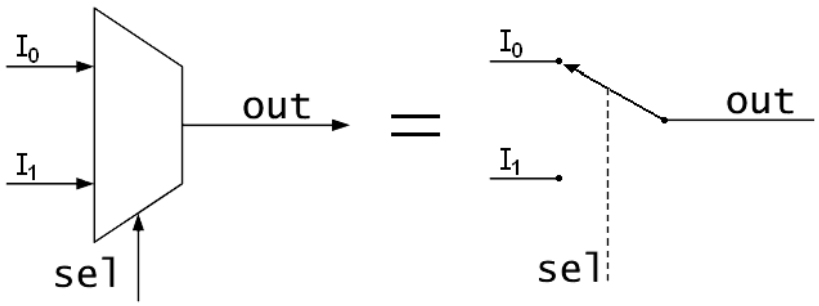
Un **Multiplexer**, nella sua versione più diffusa, è una circuito logico a **N ingressi** e **1 uscita**. Attraverso un opportuno indirizzamento digitale è possibile selezionare uno qualsiasi degli N ingressi, che diventa così l'Output del dispositivo.

Un **Demultiplexer** svolge la funzione duale, ovvero ha **N uscite** e **1 ingresso**. Attraverso un opportuno indirizzamento digitale, il microcontrollore può selezionare una qualunque delle uscite.

ESEMPIO DI UN MUX A 1 BIT E A 1 USCITA

La versione di Multiplexer più semplice che si può considerare utilizza un solo bit di selezione **sel** con il quale è possibile collegare sulla porta di uscita una ed una soltanto delle due porte di ingresso **Io** e **I1**.

Sotto l'immagine è riportata la funzione in logica di Boole che descrive quanto enunciato sopra.



Out = Sel* I0 + Sel I1

Osservando le tavole di verità riportate di seguito, si evince che è possibile selezionare l'ingresso **I0 (D0)** quando il bit di selezione è posto a valore logico **ALTO** e l'ingresso **I1(D1)** quando il bit di selezione è posto a valore logico **BASSO**.

S	Q
0	D0
1	D1

Oltre al pin di selezione, spesso, i MUX in commercio possiedono un **BIT** di **ENABLE**, per agevolare la gestione di un Multpilexing sincrono e minimizzare i consumi di potenza.

Quest'esempio, seppure autoesplicativo del funzionamento di un **MUX**, è ben lungi dall'evidenziarne le potenzialità, poichè il risparmio in termini di '**PIN**' è di uno soltanto. Si raccomanda pertanto, per comprendere maggiormente le potenzialità del Multiplexing/Demultiplexing, la lettura del paragrafo: "*Configurazione dei pin e collegamento ad ATMEGA328*".

Le tipologie di MUX e DEMUX variano fortemente e occorre sceglierli in relazione al tipo di applicazione e possono implementare al loro interno anche funzioni matematiche semplici come le somme e avere più di un'uscita, oppure un'uscita e il suo complemento.

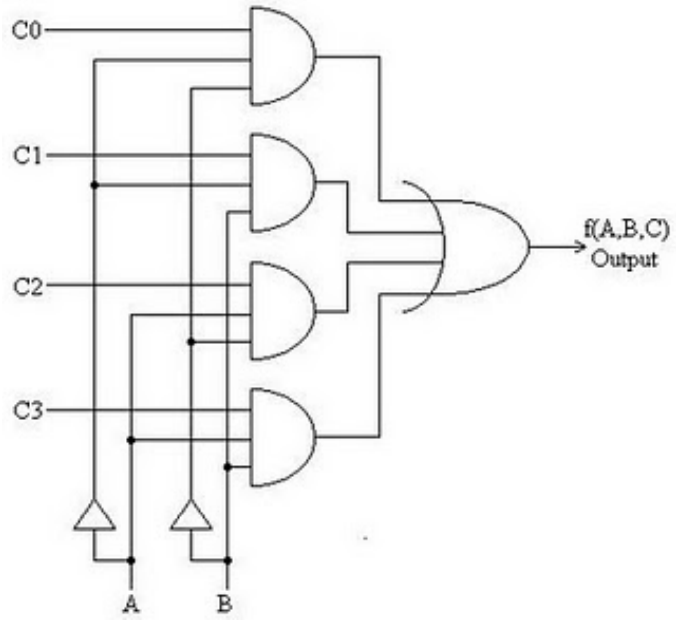
COME SCEGLIERE UN MUX O UN DEMUX?

Nella scelta di un MUX o DEMUX, la prima scrematura deve essere fatta in base alla tipologia di applicazione, ovvero in base al numero di 2^N di I/O che bisogna gestire (che determina il numero N di bit). A questo punto è possibile osservare le caratteristiche statiche e dinamiche del dispositivo e scegliere quello che più si adatta all'applicazione, a parità di prezzo, in termini di velocità di risposta, range di temperatura etc.

COME È FATTO UN MUX?

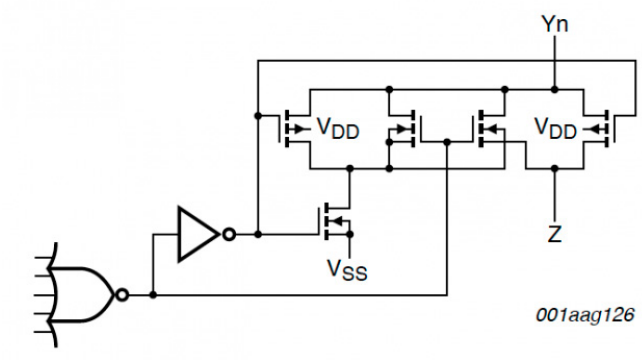
Un MUX è formato da porte logiche che possono essere costruite in diverse tecnologie, come ad esempio CMOS (la più diffusa in assoluto) e JFET (principalmente per applicazioni audio) e sfruttare diverse logiche booleane.

Nell'esempio di sotto si può osservare come, **2 Bit** di selezione (**A** e **B**) possano discriminare i 4 ingressi (**C0,C1,C2,C3**) attraverso una logica a porte **AND** la cui uscita è inviata a una **XOR**.



UN CASO PRATICO: L'HFE4067B COME DEMULTIPLEXER

Di seguito è possibile osservare lo schema logico di un interruttore del **MUX/DEMUX HEF4067B** prodotto da **NXP** a **16 canali** (**4 bit** di selezione) e disponibile in package **DIP24** (molto comodo per testarlo su breadboard) e **SO24**. Nell'esempio considereremo l'utilizzo di tale dispositivo come **Demultiplexer digitale** collegato a un microcontrollore **ATMEL ATMEGA328**, lo stesso di **Arduino**.



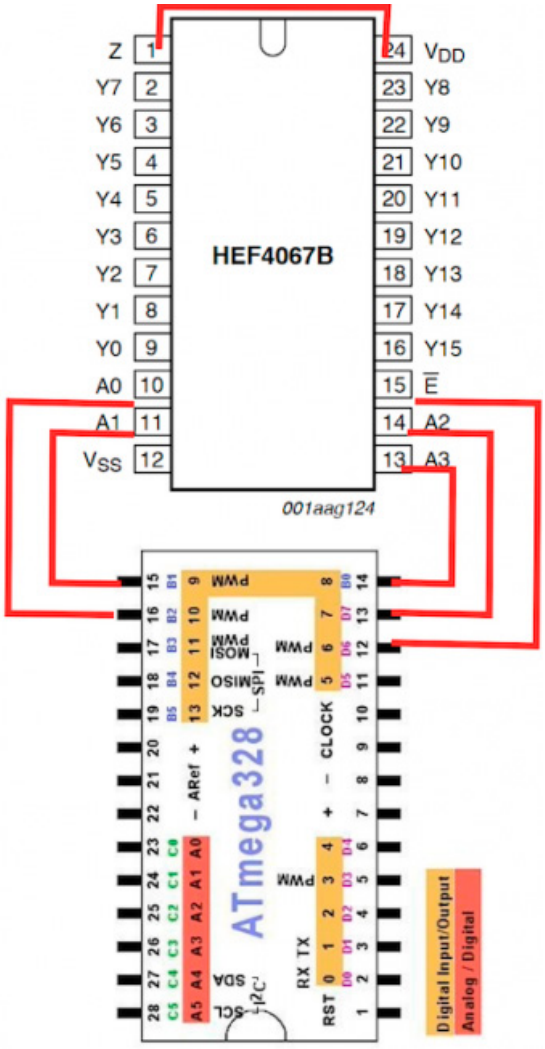
CONFIGURAZIONE DEI PIN E COLLEGAMENTO AD ATMEGA328

Il pinout di questo componente presenta i canali I/O numerati da **Y0** a **Y15**, i 4 bit di selezione (**A0,A1,A2,A3**), un bit di I/O comune **Z**, i due pin di alimentazione (**Vdd** e **Vss=GND**) e infine un pin di **ENABLE**.

Utilizzando questo dispositivo come demultiplexer **si possono comandare fino a 16 canali utilizzando soltanto 4 pin del microcontrollore** per l'indirizzamento, più un pin di Enable, per un totale di **5 pin per gestire 16 possibili uscite**.

Attraverso questa configurazione è possibile utilizzare l'**ATMEGA328** (il popolare microcontrollore utilizzato da **Arduino**), con i pin dal **12** al **17** impostati come **OUTPUT** per attuare le uscite da **Y0** a **Y15** del **DEMUX**.

A0	A1	A2	A3	SEL
0	0	0	0	Y0
0	0	0	1	Y1
0	0	1	0	Y2
0	0	1	1	Y3
0	1	0	0	Y4
0	1	0	1	Y5
0	1	1	0	Y6
0	1	1	1	Y7
1	0	0	0	Y8
1	0	0	1	Y9
1	0	1	0	Y10
1	0	1	1	Y11
1	1	0	0	Y12
1	1	0	1	Y13
1	1	1	0	Y14
1	1	1	1	Y15



Dopo aver collegato **Vdd** alla tensione di alimentazione (tipicamente **5 V**) e **Vss** a **GND**, occorre implementare una rete di **pull-down** per le uscite digitali Y0…Y15 del HEF4067B.

Infatti, queste, quando sono in stato di non-selezione, sono **flottanti** e quindi soggette a spiacevoli fenomeni di accoppiamento di tipo elettrostatico.

Per ovviare a questo problema è sufficiente mettere un resistore di valore molto grande (**100kΩ-1MΩ**) tra l'uscita da pilotare e la massa.

A questo punto è sufficiente indirizzare tramite codice le uscite desiderate per fornire un output digitale all'uscita 0 piuttosto che alla 15. Per fare ciò, è necessario osservare la tavola di verità del Demultiplexer riportata in precedenza.

Da notare che lo stato delle uscite rimane inal-

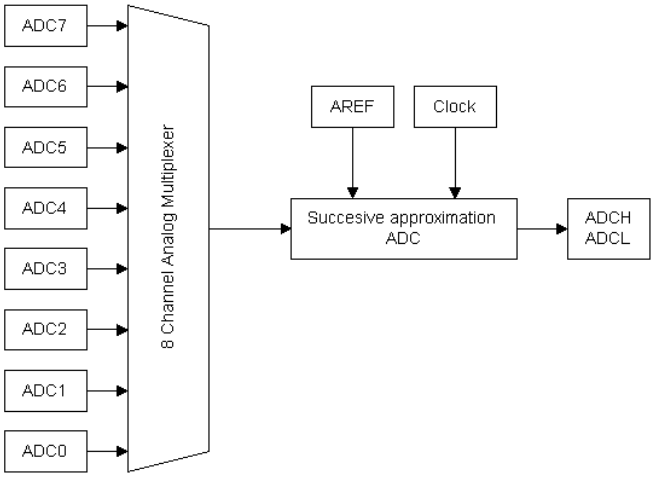
terato fino a quando il pin di **ENABLE** non viene portato a un valore alto.
Pertanto l'algoritmo per fornire tensione a una ipotetica uscita arbitraria, in questo caso quella corrispondente al pin numero 11 sarà il seguente:

A0 —> 1
A1 —> 0
A2 —> 1
A3 —> 1
E —> 1

In questo modo è possibile ad esempio pilotare l'accensione e lo spegnimento di 16 LED in maniera indipendente utilizzando soltanto 5 pin del microcontrollore.
Complicando un po' la logica di pilotaggio è possibile, inoltre, mettere due o più MUX/DEMUX in cascata per estendere ulteriormente il numero di canali controllabili.

QUANDO USARE UN MUX/DEMUX?

Sicuramente quando il controllo in tempo reale non è un vincolo stringente, quando si vogliono usare al meglio le poche risorse hardware che si hanno piuttosto che aggiungerne di nuove (come ad esempio un solo ADC condiviso da più ingressi) e sicuramente quando si ha un gran numero di dispositivi che si comportano in maniera identica da gestire separatamente, siano essi dei sensori tutti uguali da cui acquisire periodicamente dati, oppure delle uscite da attuare a passi alterni.



QUANDO NON USARE UN MUX/DEMUX?

Quando i pin del microcontrollore a disposizione sono maggiori del numero di elementi che bisogna controllare, oppure si vuole utilizzare un microcontrollore con un numero minore di PIN di proposito.

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/controllare-io-multipli-con-pochi-pin-di-arduino

Speciale Arduino PROGETTI

Analizzatore MIDI con Arduino

di slovati

L'INTERFACCIA MIDI

L'interfaccia **MIDI**, ampiamente utilizzata nel campo della produzione musicale a livello elettronico, si basa sulla trasmissione seriale di byte alla frequenza fissa di 31250 baud [1]. A differenza della RS-232, l'interfaccia non utilizza soglie di tensione predefinite per i livelli basso e alto associati ai bit dati e ai bit di start e stop. Viceversa, l'interfaccia è basata su un loop di corrente da 5 mA creato tra l'uscita (MIDI Out) di un dispositivo e l'ingresso (MIDI In) di un'altro dispositivo. Un flusso di corrente corrisponde allo stato logico zero, mentre l'assenza di corrente corrisponde allo stato logico uno. In **Figura 1** è mostrata in dettaglio l'implementazione di questa interfaccia. Le interfacce di ingresso e uscita (MIDI In e MIDI Out) richiedono due pin ciascuna, e due dispositivi MIDI richiedono pertanto due soli fili per il collegamento. Uno dei pin di uscita è collegato in modo permanente alla tensione +5 V tramite una resistenza da 220 Ω. Per trasmettere lo stato logico zero, il secondo pin di uscita viene portato a massa, sempre tramite una resistenza da 220 Ω. Ne consegue che una piccola corrente scorre tra i pin del connettore MIDI In dell'altro dispositivo. Per trasmettere

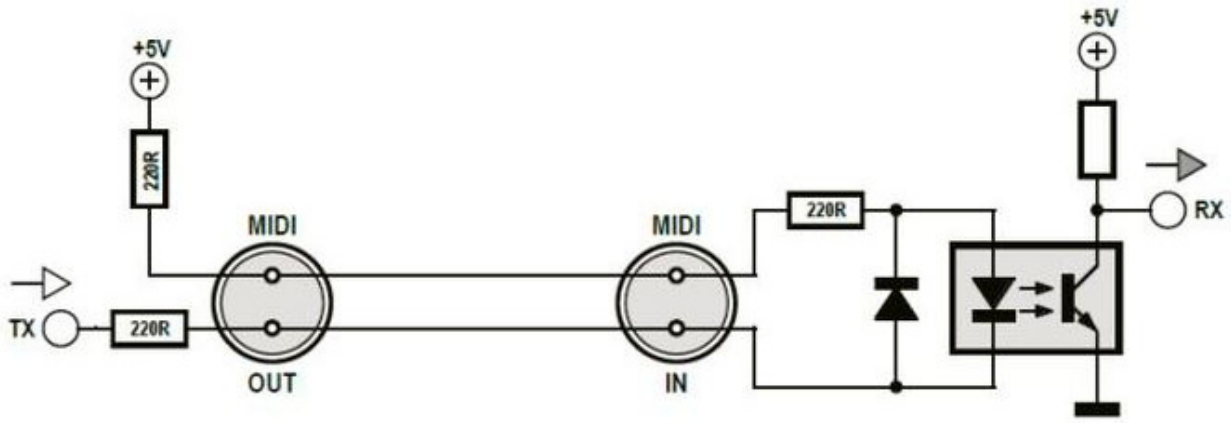


Figura 1: un esempio di interfaccia MIDI

invece lo stato logico uno, il secondo pin di uscita viene portato alla tensione di +5 V, e di conseguenza non si genera alcun flusso di corrente. **Il funzionamento è pertanto, entro certi limiti, compatibile con quello di un'interfaccia UART che lavora con i livelli logici TTL**. Sul lato MIDI In dell'interfaccia è presente un fotoaccoppiatore dotato di fototransistor. Quando una corrente attraversa il cavo MIDI, questo transistor "tira" l'uscita verso massa; nello stato di riposo, l'uscita si porta alla tensione di +5 V. Possiamo perciò collegare questo segnale direttamente all'ingresso RX di un microcontrollore. I due collegamenti utilizzati per l'ingresso e per l'uscita fanno capo ai pin 4 e 5 di una presa DIN a 5 poli. Dal punto di vista esterno, pertanto, i connettori MIDI In e Out sono identici. Tuttavia, occorre utilizzare delle prese separate per ogni dispositivo che richieda di essere collegato sia in ingresso che in uscita.

L'HARDWARE

Le caratteristiche menzionate in precedenza implicano che sia relativamente semplice aggiungere la funzionalità MIDI, sia in ingresso che in uscita, a un circuito basato su microcontrollore. La soluzione proposta in questo articolo si basa sull'utilizzo combinato di una board Arduino Uno e dell'Extension Shield di Elektor, alle quali viene collegata l'interfaccia MIDI tramite un connettore ECC (Embedded Communication Connector) [2]. Il circuito risultante è visualizzato in **Figura 2**. Sul lato sinistro è visibile la parte relativa al MIDI Input, con un accoppiatore ottico di tipo 6N137. La sua uscita è collegata al pin 6 del connettore ECC (K2), il quale a sua volta è collegato al pin RX del microcontrollore. Il pin 5 del connettore ECC porta il segnale TX del microcontrollore al modulo MIDI, utilizzato per pilotare direttamente la presa di uscita MIDI (K3). La scheda a microcontrollore fornisce l'alimentazione +5 V al circuito tramite il pin 9 del connettore ECC, mentre un apposito led indica la presenza dell'alimentazione.

In **Figura 3** è mostrato il circuito stampato approntato da Elektor per il progetto; il PCB è stato appositamente perforato in modo tale da poterlo facilmente separare in due parti [3]. La parte destra può essere utilizzata come interfaccia MIDI In, mentre quella a sinistra come interfaccia MIDI Out (con i segnali MIDI disponibili sul connettore K5), oppure come una scheda flessibile con K4 come connettore di ingresso.

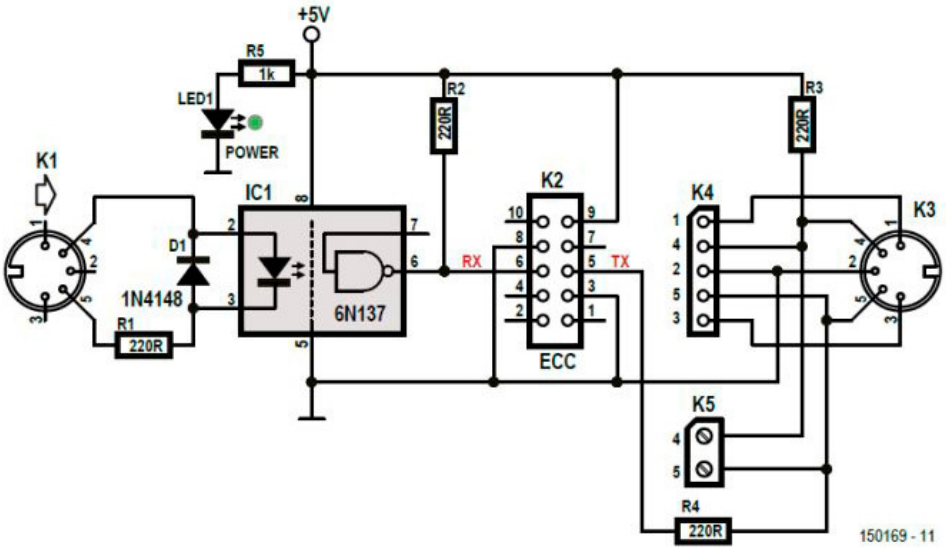
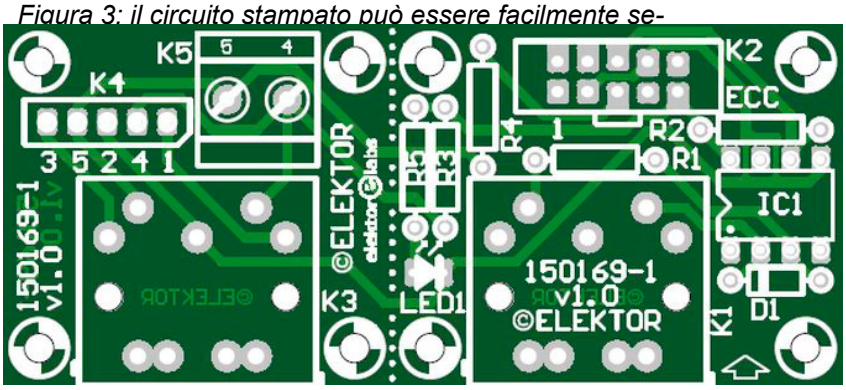


Figura 2: schema circuitale del modulo MIDI In/Out

Figura 3: il circuito stampato può essere facilmente separato in due parti



Il prototipo realizzato da Elektor Labs (visibile in **Figura 4**), utilizza un pin header per K4, ma sarebbe molto più conveniente utilizzare un header socket, in modo tale da ridurre il rischio di corto circuiti accidentali. Il PCB è disponibile presso l'Elektor Store [4], e l'assemblaggio del circuito non dovrebbe presentare grosse difficoltà.

ELENCO COMPONENTI

L'elenco completo dei componenti richiesti per la realizzazione del circuito è il seguente.

RESISTENZE

R1,R2,R3,R4 = 220 Ω

R5 = 1kΩ

SEMICONDUTTORI

D1 = 1N4148

LED1 = LED, verde, 3mm

IC1 = 6N137, DIP8

VARIE

K1,K3 = presa DIN per montaggio su PCB, 180°

K2 = boxheader a 10 vie (2x5), passo 0,1"

K4 = pin header a 5vie, passo 0,1"

K5 = connettore a 2 vie, passo 0,2"

PCB #150169-1 v1.0

IL SOFTWARE

Il modulo MIDI è collegato al connettore ECC dell'Extension Shield tramite un cavo piatto a 10 fili, mentre l'Extension Shield è a sua volta installato sulla board Arduino Uno. **Il microcontrollore Atmega328P presente sull'Arduino Uno può ora ricevere i byte MIDI tramite l'interfaccia UART**, ma

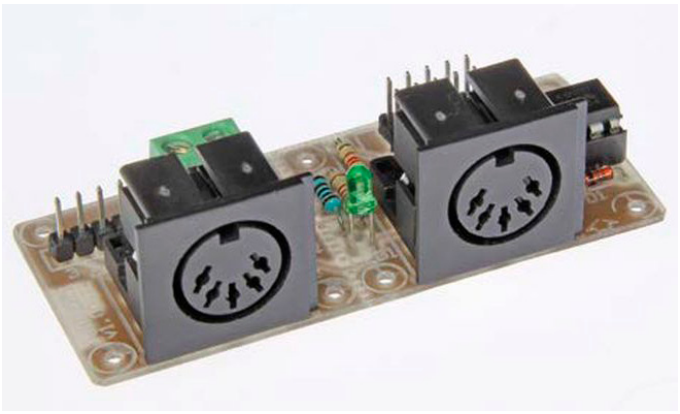


Figura 4: il prototipo sviluppato da Elektor Labs con in evidenza le due prese DIN

ovviamente avremo anche bisogno di un opportuno software [4] per completare il progetto. Fortunatamente, il protocollo MIDI non è così complicato (si veda l'apposito capitolo dedicato allo

stesso, più avanti nell'articolo). La maggiorparte dei messaggi MIDI è infatti composta da tre soli byte, ed è pertanto relativamente semplice eseguire la decodifica di ogni messaggio.

Il software esegue le seguenti operazioni principali:

1. inizializza l'UART, impostando la velocità a 31250 baud;
2. memorizza i byte ricevuti in un buffer circolare, tramite una gestiore ad interrupt;
3. controlla periodicamente lo stato del buffer circolare, rilevando la presenza di un nuovo messaggio MIDI analizzandone il primo byte. Nel caso sia stato ricevuto un nuovo messaggio, lo decodifica e chiama una funzione specifica per indicare che un nuovo messaggio MIDI è stato correttamente ricevuto e decodificato;
4. visualizza su display il contenuto del messaggio ricevuto. Un messaggio rimane visualizzato sul display sino a quando non viene ricevuto un nuovo messaggio; in tal caso, il contenuto del display viene aggiornato.

La soluzione ideale, dal punto di vista software, sarebbe quella di avere un modulo software separato che si occupi di ciascuna delle attività sopra menzionate, al fine di migliorare la riusabilità e portabilità del software stesso, e semplificarne la manutenzione. Le dipendenze (incluse quelle temporali) tra i moduli dovrebbero inoltre essere ridotte al minimo (torneremo su questo aspetto più avanti, nel corso dell'articolo). **L'Embedded Firmware Library (EFL) [5] rappresenta la scelta ideale per un software dimostrativo**, dal momento che essa include già un'implementazione del buffer circolare e una libreria apposita per la gestione del display. Quello che rimane da fare è scrivere una piccola libreria MIDI per gestire l'operazione numero 3 della lista menzionata in precedenza.

LA DECODIFICA MIDI

I messaggi MIDI vengono decodificati tramite l'utilizzo di una piccola libreria contenuta nei file MidiEFL.h/.c. Come tutte le altre liberie di tipo EFL utilizzate nei progetti Elektor (ad esempio la libreria ElektorBus), anche questa è stata progettata per funzionare con una moltitudine di canali fisici di comunicazione differenti, e potrebbe persino essere utilizzata anche per decodificare i messaggi MIDI ricevuti su una connessione TCP/IP. Ciò che interessa, è che i byte ricevuti vengano immagazzinati in un buffer circolare, la cui posizione viene comunicata alla libreria durante la fase di inizializzazione nel modo seguente:

```
MIDI_LibrarySetup(UARTInterface_Send, 0,
        UARTInterface_GetRingbuffer(0), MIDIIn_Process);
```

Il primo parametro indica la funzione da chiamare quando si vuole trasmettere un messaggio MIDI. Il secondo parametro indica che verrà utilizzata l'interfaccia UART sia per trasmettere che per ricevere i dati (su Arduino Uno, in realtà, esiste soltanto una interfaccia UART). Il terzo parametro specifica il **buffer circolare** utilizzato per la ricezione, mentre il quarto ed ultimo parametro è un puntatore a una funzione callback che andrà implementata nel codice. Questa funzione verrà chiamata automaticamente dalla libreria MIDI quando un nuovo messaggio verrà ricevuto e decodificato. Il loop principale del programma deve invocare in modo periodico la funzione di libreria *MidiProtocol_Engine()*, che si occupa della gestione del punto 3 della lista menzionata in precedenza.

Non appena un messaggio viene ricevuto, esso viene memorizzato nella struttura *ReceivedMidiData di tipo MidiData*. Il main del codice può ottenere un puntatore a questa struttura chiamando la funzione di libreria *GetReceivedMidiData()*, e accedere così al contenuto decodificato. Nella funzione callback menzionata precedentemente, abbiamo ora accesso a tutti gli elementi del messaggio MIDI più recente, e possiamo quindi visualizzarli sul display. Un esempio di codice potrebbe essere il seguente:

```
MidiData* ReceivedMidiData = MIDI_GetReceivedMidiData();
Display_WriteNumber(0, 0, ReceivedMidiData->Channel);
```

Il risultato di queste linee di codice è quello di visualizzare sulla prima riga del display il numero di canale del messaggio ricevuto. Purtroppo, la libreria EFL per il display contiene funzioni per visualizzare stringhe e numeri su una specifica riga del display, ma sempre partendo dalla posizione più a sinistra della riga stessa. Il display presente sull'Extension Shield per Arduino possiede soltanto due righe, e vorremmo poter visualizzare quattro differenti tipi di informazioni su di esso. Per risolvere questo problema, la libreria è stata estesa in modo tale da poter specificare dove visualizzare le informazioni in formato testo o numerico, avendo a disposizione otto possibili campi di destinazione. Questi campi sono numerati da 0 a 7 (si osservi la **Figura 5**), e questo numero viene chiamato "posizione" del campo. Le posizioni possono essere anche utilizzate per selezionare un particolare led o tasto all'interno di un blocco. I comandi di uscita per controllare lo stato di un led (ON=1, OFF=0) all'interno di un blocco di led, e per visualizzare il corrispondente messaggio in una specifica posizione del display sono i seguenti:

```
SwitchLED (LEDblocknumber, position, ON);
Display_WritePosition (displaynumber, position, "ON");
```

Le coordinate e le dimensioni (in caratteri) dei campi possono essere specificate utilizzando la funzione *Display_SetPosition(uint8 DisplayPosition, uint8 row, uint8 column, uint8 columnmax)*. Per semplificare ulteriormente le cose, durante l'inizializzazione la libreria del display imposta due campi su ogni riga del display, ciascuno con dimensione pari alla metà di ogni riga. Nel nostro caso avremo pertanto 4 campi, tutti della stessa dimensione, nei quali potremo visualizzare i quattro elementi più importanti di ogni messaggio ricevuto (si osservi la **Figura 6**). Per ridurre al minimo l'occupazione di memoria, display differenti appartenenti a uno stesso sistema non possono essere configurati in modo diverso l'uno dall'altro, e ogni campo non può occupare più di una riga. Ovviamente, la libreria può anche essere modificata per rimuovere queste restrizioni, qualora fosse necessario.

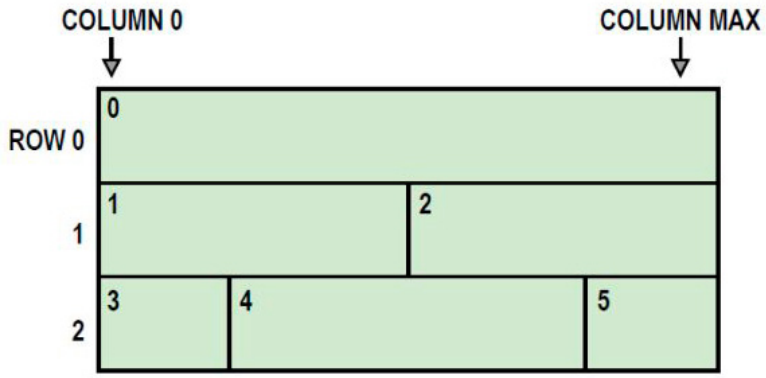


Figura 5: la libreria per il display estesa consente di configurare i campi all'interno dell'area del display, dove si possono visualizzare stringhe o numeri. In questo esempio è mostrato un display di 3 righe, divise in 6 campi numerati da 0 a 5

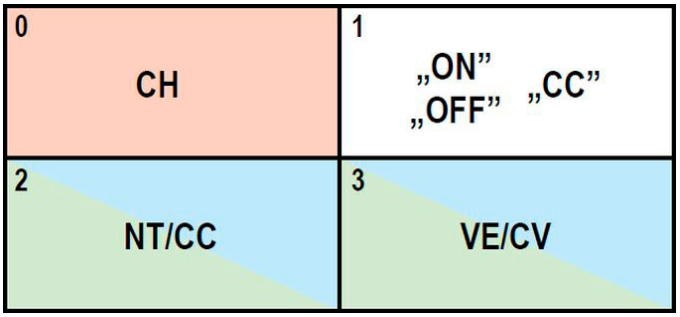


Figura 6: i vari elementi che compongono un messaggio MIDI vengono visualizzati su Quattro campi del display: CH=canale, NT=nota (stringa), VE=velocità, CC=numero del controller, CV=valore del controller

USCITA FLESSIBILE

Poichè la scrittura sul display è implementata direttamente nella funzione chiamata dalla libreria MIDI quando un nuovo messaggio è stato decodificato, si è volutamente creato un accoppiamento stretto tra i due moduli, in particolare dal punto di vista temporale. Ciò non è sempre desiderabile. Una soluzione migliore potrebbe essere quella di non aggiornare il display immediatamente. E' possibile che questa soluzione possa essere introdotta in versioni future del software, includendo anche attività ad elevata priorità. Ad esempio, si potrebbero loggare i byte ricevuti sull'interfaccia MIDI, associando loro anche un timestamp, oppure si potrebbe estendere il progetto realizzando un mini sintetizzatore, in grado di convertire i byte ricevuti in suoni. La soluzione che permette di realizzare queste estensioni è quella di creare un elemento in una tabella speciale (chiamata "StateTable") per ciascuno dei differenti elementi MIDI da visualizzare. Oltre al valore corrente di ogni elemento, verrà mantenuta una flag che indica se il valore è cambiato rispetto all'ultima visualizzazione sul display. La funzione chiamata alla ricezione di ogni messaggio MIDI, si occuperà anche della scrittura nella StateTable dei nuovi valori degli elementi del messaggio MIDI. Se un valore differisce da quello precedente, la flag "STATE_UPDATED" viene alzata. A questo punto occorre controllare periodicamente quando il display richiede un aggiornamento oppure no. Si noti come la frequenza con cui è eseguita questa operazione è indipendente dalla frequenza di arrivo dei messaggi MIDI. La funzione che si occupa di questa operazione è la Reaction_Process(), implementata nel nuovo modulo EFL InOutEFL.c. La funzione analizza ogni elemento della tabella, e, per ogni elemento che ha la flag settata, esegue la chiamata alla corrispondente funzione di output. Le funzioni di output sono contenute nella tabella OutputTable, che deve essere preventivamente inizializzata. Una terza tabella, chiamata ReactionTable, contiene I collegamenti tra le altre due, garantendo che esista una funzione di output per ogni parametro il cui valore può cambiare. E' altresì possibile fornire un indice in una quarta tabella, chiamata EncoderTable.

Un esempio può aiutare a comprendere le relazioni esistenti tra queste tabelle, per cui analizzeremo ora la modalità con cui viene processata una nuova nota MIDI appena ricevuta. All'inizio del programma vengono impostati alcuni elementi della tabella come indicato nel seguito.

```
S_MidiIn_Note = State_Add(0, 0, 127, STATE_MINMAXMODE_OVERFLOW);
O_Write_Pos2 = Output_Add(Display_WritePosition, 0, 2);
E_Midi_Note = Encoder_Add(Midi_NoteEncode);
Reaction_AddOutput(S_MidiIn_Note, STATE_UPDATED, O_Write_Pos2, E_Midi_Note);
```

Quando viene ricevuto un nuovo valore (che deve essere compreso nel range tra 0 e 127), esso viene copiato nella StateTable, e la flag STATE_UPDATED viene settata. Ciò implemen-

tato nella funzione callback come segue:

```
MidiData* ReceivedMidiData = Midi_GetReceivedMidiData();
State_Update(S_MidiIn_Note, ReceivedMidiData->Note);
```

La funzione Reaction_Process() viene chiamata periodicamente nel loop principale, al fine di controllare se la flag è settata o meno. Se ciò avviene, viene chiamata la funzione encoder configurata per gestire il cambio di valore della variabile: in questo caso la funzione encoder è **Midi_NoteEncode(), implementata nella libreria EFL MIDI**. Questa funzione riceve come parametro il valore della nota, compreso tra 0 e 127, e lo converte in una stringa del tipo "C#4". Questa stringa è a sua volta passata come parametro alla funzione di output Display_WritePosition(), che scrive il testo nella posizione 2 del display 0. Il comportamento del software può essere modificato cambiando il contenuto delle tabelle, anche dinamicamente, mentre il software è già in esecuzione.

INDIPENDENZA DALL'HARDWARE

Questo progetto dimostra come le applicazioni basate sulle librerie EFL siano modulari e indipendenti dall'hardware utilizzato. All'inizio del programma, nella funzione *ApplicationSetup()*, viene anzitutto eseguita l'inizializzazione delle tabelle. Successivamente, tutto viene gestito in modo automatico, e i vari moduli possono funzionare in modo disaccoppiato l'uno dall'altro. Possiamo portare facilmente il codice su un'altra scheda, dove ad esempio potremmo utilizzare il display 1 al posto dello 0 per visualizzare le informazioni ricevute sull'interfaccia MIDI. Oppure, potremmo voler cambiare la posizione in cui sono visualizzate le varie informazioni, oppure ancora inviare le informazioni decodificate su una UART differente, anzichè mostrarle sul display. Tutte queste varianti possono essere implementate tramite semplici modifiche al codice di setup, lasciando inalterato il resto del programma.

IL PROTOCOLLO MIDI

Nonostante quello che molti possano pensare, il protocollo MIDI è in realtà piuttosto semplice. La maggior parte dei messaggi MIDI consiste infatti di tre soli byte. Il primo byte include il codice del comando nei quattro bit più significativi, e il canale MIDI (da 0 a 15) nei quattro bit meno significativi. Il bit più significativo di questo byte è sempre settato, mentre i due byte successivi del messaggio hanno sempre il bit più significativo a zero (e possono quindi rappresentare valori compresi tra 0 e 127). Ciò permette di individuare agevolmente l'inizio di un messaggio all'interno di uno stream di byte ricevuti. Il software che accompagna il progetto [4] è in

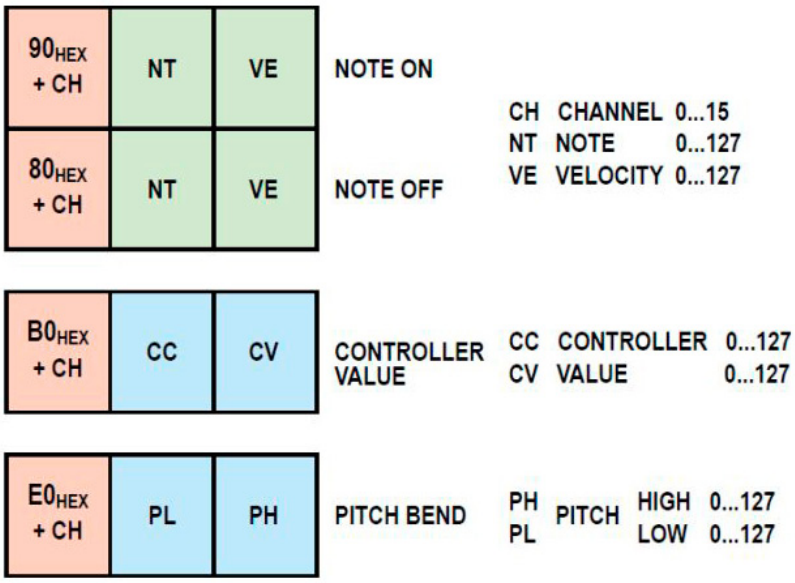


Figura 7: struttura di alcuni comandi MIDI

grado di riconoscere i quattro comandi MIDI più importanti. Due di questi (quelli con codice 90 hex e 80 hex) corrispondono ai messaggi “nota off” e “nota on”. Il secondo byte contiene l’ampiezza della nota, espressa in semitoni con un intero tra 0 e 127; in questo modo dodici valori coprono un’ottava. Il terzo byte (anch’esso compreso tra 0 e 127), contiene il valore di velocità, e indica quanto velocemente sulla tastiera è stato premuto oppure rilasciato il tasto. Il comando B0 hex identifica il messaggio “cambio di controllo”, ed è seguito dal numero del controller (compreso tra 0 e 127) e dal nuovo valore del controllo (ancora compreso tra 0 e 127). Il numero di controller viene assegnato ad ogni parametro che influenza il suono prodotto da un sintetizzatore, in modo tale che essi possano essere controllati in tempo reale, persino durante una performance musicale. Ad esempio, il controller numero 1 è riservato per il controllo della modulazione sulla maggiorparte delle tastiere. Maggiori dettagli su questo agomento possono essere trovati in [6]. Il comando E0 hex identifica invece il messaggio “pitch bend change”. Questa funzione richiede una risoluzione molto fine, per cui il range è compreso tra 0 e 16383 (14 bit). Il secondo byte del messaggio contiene i sette bit meno significativi del valore, mentre il terzo byte contiene i sette bit più significativi (figura 7).

LA LIBRERIA EFL INOUT

Possiamo comprendere i vantaggi che derivano dall’utilizzo di questa libreria considerando un semplice esempio. Si supponga che si voglia progettare un alimentatore, in cui ci sono quattro variabili chiamate U_setpoint, U_actual, I_actual, e I_max. L’apparato include dei controlli (tasti, encoder rotativi, potenziometri) che permettono di modificare i valori di U_setpoint e I_max, mentre i valori di U_setpoint, U_actual, e così via devono poter essere visualizzati sul display. Vorremmo inoltre essere in grado di eseguire il codice su board differenti, a scapito di piccole variazioni dello stesso. Ad esempio, **una board potrebbe disporre di un potenziometro** per impostare il valore di U_setpoint, mentre un’altra potrebbe disporre di tasti “up” e “down”. Naturalmente il codice richiederà qualche piccola modifica, ma l’idea di fondo è quella di rendere il porting il più semplice possibile, isolando le parti di programma dipendenti dall’hardware nella funzione *ApplicationSetup()*. Il secondo obiettivo è quello di disaccoppiare tra loro, sia dal punto di vista software che da quello temporale, i vari processi coinvolti nelle operazioni di input, modifica dei valori di stato, e output. Se le letture di un valore sono acquisite alla frequenza di mille campioni al secondo, non ha senso aggiornare il display ogni volta che arriva un nuovo valore. Il terzo obiettivo consiste nel minimizzare lo sforzo programmativo. I valori di ingresso utilizzati per impostare dei parametri richiedono sempre di essere validati rispetto a un valore minimo e massimo, e il framework dovrebbe farsi carico automaticamente di questi controlli, sollevando dal compito il programmatore.

La nuova common library EFL (InOutEFL.h/.c) mantiene aggiornate diverse tabelle, che devono essere inizializzate all’ingresso nel programma. Le funzioni State_Add(), Output_Add(), e così via, sono utilizzate per creare un nuovo elemento nella tabella corrispondente; esse restituiscono l’indice dell’elemento appena aggiunto, che può essere successivamente utilizzato come parametro quando si aggiunge un nuovo elemento in un’altra tabella. Questo approccio è simile alla creazione di un circuito stampato: aggiungere elementi a una tabella equivale ad

aggiungere delle piste, creando così delle connessioni tra ingressi, variabili di stato, ed uscite. Il cuore dell'architettura è rappresentato dalla **StateTable**. Ogni elemento di questa tabella contiene il valore a 16 bit di una variabile dell'applicazione, i suoi valori minimo e massimo, e fino ad otto flag. La **InputTable** collega gli eventi in ingresso (come ad esempio la pressione di un particolare tasto, specificata dal numero di blocco e dalla posizione del tasto all'interno del blocco), con un indice della StateTable. Inoltre, essa contiene il valore dello step, che misura l'entità con cui deve essere modificata la variabile quando l'evento si verifica, e la flag che deve essere settata quando l'evento avviene. **Nella InputTable è ad esempio possibile specificare che, quando viene premuto un certo tasto**, la variabile U_setpoint (che potrebbe rappresentare una tensione in millivolt) debba essere incrementata di 100. Quando il valore viene incrementato, viene eseguito un controllo affinchè il nuovo valore risulti all'interno dei limiti ammissibili. Se il valore supera il limite superiore, esso può essere sia limitato al valore massimo, oppure resettato al valore minimo, a seconda della configurazione prescelta. La seconda alternativa permette di implementare una modalità di "wrap-around", utile in alcune applicazioni. Se il valore validato è cambiato, viene impostata una flag particolare, nel nostro caso sceglieremo la flag STATE_UPDATED. La **OutputTable** è la controparte della InputTable. Essa contiene le funzioni di output, insieme al numero del blocco e alla posizione dell'elemento di output al suo interno. Può essere utilizzata a questo scopo qualunque funzione, purchè abbia una signature di questo tipo:

```
functionname(uint8 blocknumber, uint8 position, int16 numericalvalue)
```

oppure:

```
functionname(uint8 blocknumber, uint8 position, char* textstring)
```

La **EncoderTable** contiene invece le funzioni encoder che convertono i valori numerici in stringhe di testo. Queste possono essere implementate in uno qualunque dei moduli EFL, purchè abbiano una signature del tipo:

```
char* functionname(int16 numericalvalue)
```

Ogni elemento della tabella **ReactionTable** collega un indice della StateTable (o, in modo equivalente, una variabile dell'applicazione), una flag, una funzione di output e, opzionalmente, un encoder, tutti memorizzati come indici alle rispetive tabelle. Per assicurarsi che le funzioni di output siano attivate, è necessario invocare in modo regolare la funzione *Reaction_Process()*. Questa funzione analizza gli elementi della ReactionTable, e per ciascuno di essi controlla se la flag di attivazione associata nella StateTable è settata oppure no. In caso affermativo, viene invocata la funzione di output. Infine, viene resettata la flag nella StateTable. Procedendo con il nostro esempio, potremmo definire un elemento della OutputTable in modo tale da visualizzare del testo nella posizione 0 del display 0. Potremmo anche implementare una funzione encoder che converte un valore in millivolt in una stringa di testo con il formato "x.yyy V". Questa funzione andrà aggiunta come nuovo elemento della EncoderTable. Un

nuovo elemento della ReactionTable può ora essere utilizzato per collegare U_setpoint, la flag STATE_UPDATED, e gli indici degli elementi appena aggiunti a OutputTable e EncoderTable. Se ora premiamo il tasto e incrementiamo U_setpoint di 100 mV, verranno chiamate la funzione *Reaction_Process()* e successivamente la funzione encoder, con conseguente visualizzazione del testo sul display. Modificare il codice in modo tale da utilizzare il formato punto decimale europeo (la virgola) al posto del punto è pressochè immediato: occorre scrivere una versione leggermente modificata della funzione encoder, aggiungerla alla EncoderTable, e successivamente cambiare il corrispondente indice nella ReactionTable. Questa operazione può essere eseguita anche mentre il programma è in esecuzione, ad esempio quando si vuole cambiare la lingua dell'interfaccia utente (figura 8).

RIFERIMENTI

[1] http://en.wikipedia.org/wiki/MIDI
[2] www.elektor-magazine.com/140182
[3] www.elektor-labs.com/project/150169-midi-analyzer-light.14481.html
[4] www.elektor-magazine.com/150169
[5] www.elektor-magazine.com/120668
[6] www.midi.org/techspecs/midimessages.php

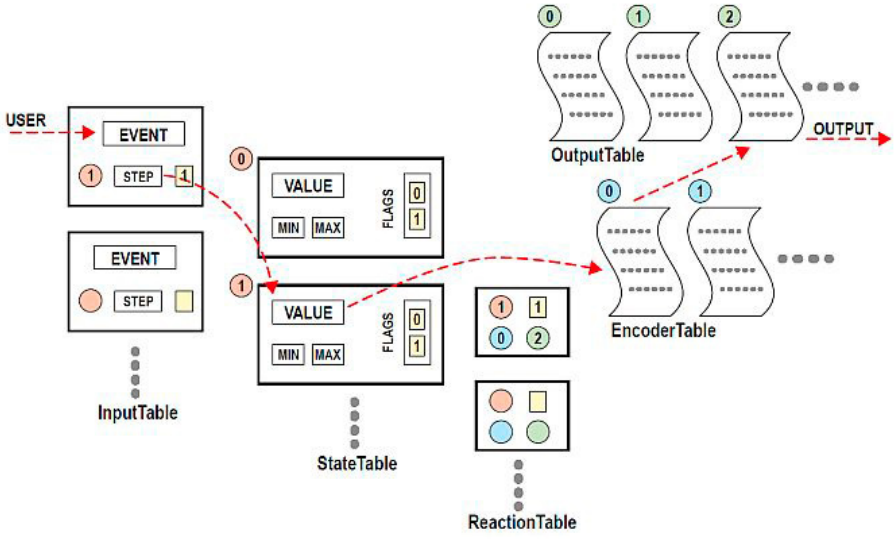


Figura 8: architettura sw della libreria

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/analizzatore-midi-con-arduino

Progetto serra domotica open source

di Antonio La Mura

1. DISPOSITIVI UTILIZZATI

Facendo una prima overview del progetto è possibile dividere il tutto in tre parti fondamentali Master, Slave e Web Server, meglio descritte successivamente. Per quanto riguarda il Master è stato realizzato utilizzando un Arduino Uno, un modulo di trasmissione wireless 2.4GHz NRF24L01 e una shield wifi CC3000. Lo Slave invece utilizza lo stesso modulo di trasmissione NRF24L01, un sensore di umidità e temperatura DHT22 e la shield per il controllo di un motore brushed della Infineon.

Infine il Master comunica tramite wifi con un web Server dedicato al quale è possibile accedere tramite una app realizzata per Android.

1.1 ARDUINO UNO

Attualmente il progetto è ancora in fase di prototipazione, pertanto si è deciso di iniziare a sviluppare il tutto con il classico Arduino Uno. Il prossimo step sarà di utilizzare per gli Slave degli Arduino Nano o Mini e per il Master un Arduino Galileo che abbiamo a disposizione. La scelta di utilizzare Galileo è dovuta sia al fatto di poter disporre di una elevata potenza della scheda, che per gestire più Slave è necessaria, sia al vantaggio di poter utilizzare Linux. Pertanto Il progetto presentato qui è un primo step di controllo domotico di una serra, che poi andrà inevitabilmente potenziato.

Figura 1 - Arduino UNO

1.2 SHIELD PER CONTROLLO MOTORE IN DC INFINEON

Grazie all'iniziativa di EMCelettronica ci è stato possibile utilizzare anche questa comodissima shield della Infineon per il controllo di un motore DC. In questa fase è stato utilizzato un motore recuperato da una vecchia stampante, quest'ultimo ci serve per aprire e chiudere una finestra della serra quando la temperatura è troppo elevata.

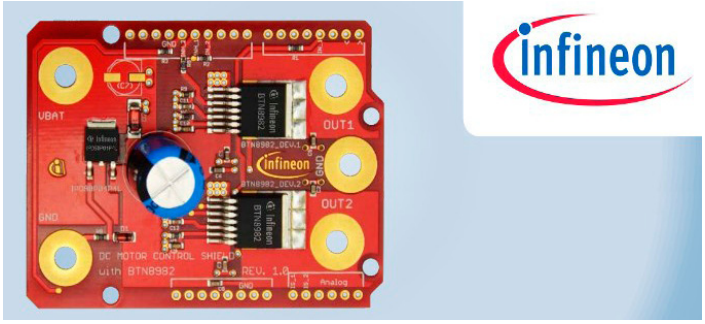


Figura 2 - Motor Shield Infineon

1.3 MISURA TEMPERATURA E UMIDITÀ AMBIENTALE

Per il controllo dei parametri ambientali è stato utilizzato un DHT22, che ci permette di avere una misura sia dell'umidità e sia della temperatura. Parametri che poi saranno trasmessi al Master che li invierà al Web Server. Non è stata effettuata una vera taratura di questa misura, ma semplicemente un confronto tra i valori ottenuti dal sensore e quelli ottenuti da un termometro e da un igrostato, essendo questi molto simili non è stata effettuata alcuna compensazione.

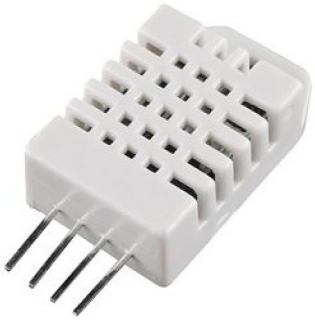


Figura 3 - DHT22

1.4 NRF24L01

Infine sono stati utilizzati due moduli NRF per la comunicazione wireless tra Master e Slave, per trasmettere le informazioni lette dal sensore (Slave to Master) e per trasmettere le nuove impostazioni fornite dall'utente (Master to Slave).
È stato formulato un semplice protocollo di comunicazione, per assicurare che i dati trasmessi fossero consegnati. Quanto detto avviene tramite un controllo effettuato sia dal Master che dallo Slave, ciò permette una elevata efficienza di comunicazione e comunque un basso costo computazionale.

2. MASTER

Di seguito è riportato il Flow Chart del programma:

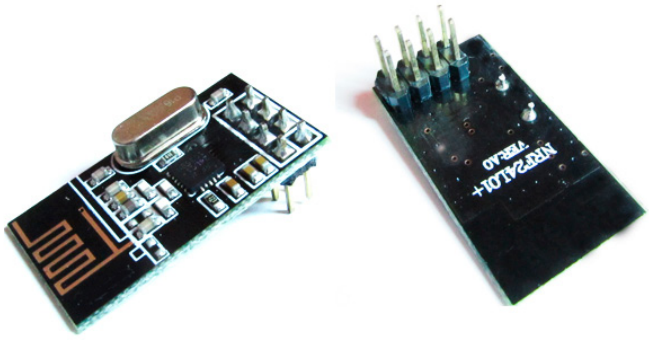


Figura 4 - NFR24L01

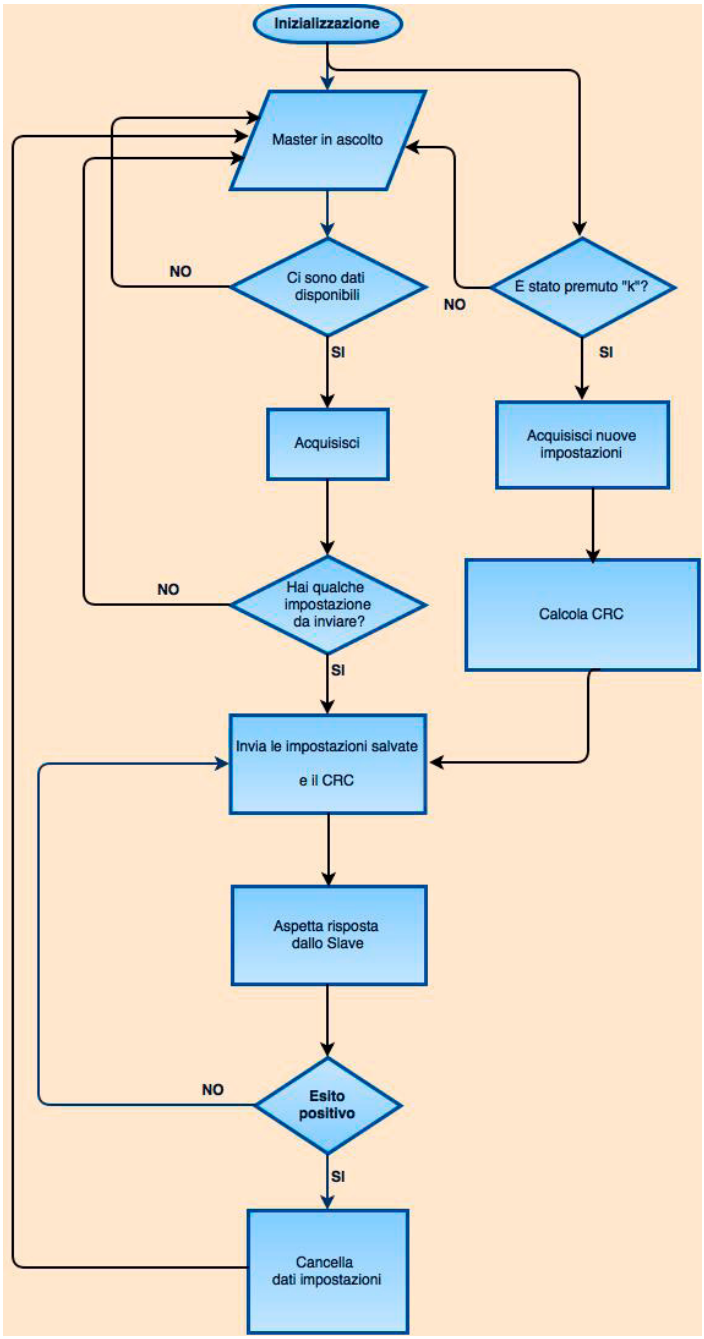


Figura 5 - Flow Chart Master

Il Master è sempre in ascolto, appena ci sono dati disponibili li acquisisce e li stampa a video. Intanto verifica se sono presenti nuovi settaggi delle impostazioni da inviare allo Slave.

NB: il master invia le nuove impostazioni solo quando lo Slave gli invia le letture del sensore. Infatti, secondo il nostro protocollo è sempre lo Slave ad avviare la comunicazione.

Se sono presenti nuovi settaggi in lista, ne calcola prima il CRC a 8 bit, poi allega in coda alle impostazioni anche quest'ultimo, che servirà allo Slave per verificare se i dati ricevuti sono corretti o meno.

Fatto ciò il Master aspetta un'altra risposta dallo Slave per confermare la corretta ricezione dei dati. Se ciò non fosse, i dati non vengono cancellati ma restano in memoria e saranno inviati successivamente al prossimo contatto da parte dello Slave.

Infine ritorna nello stato iniziale di ascolto, per ricominciare un nuovo ciclo.

La funzione che calcola il CRC a 8 bit è un algoritmo basato sulle formule di Dallas – Maxim.

2.1 COMUNICAZIONE LATO MASTER

Vediamo ora un po' di codice, in questo paragrafo descriviamo i processi controllati dal Master:

```
while(!Mirf.dataReady()){
  //Aspetto qualche informazione
 }
 Mirf.getData(data);

 Serial.print("RAW DATA: ");
 Serial.println((char*)data);
 char* command = strtok((char *)data, ";");
 int count = 0;
 while (command != 0){

  //Divido le informazioni
  switch (count){
  case 0:
   snprintf(name, sizeof(name), "%s", command);
   break;

  case 1:
   temp = atof(command)/10;
   break;

  case 2:
   hum = atof(command)/10;
   break;
  }

  command = strtok(0, ";");
  count++;
 }
```

Il *while* alla prima riga mette il Master in attesa, appena arriva un dato lo sketch prosegue. Con il *Mirf.getData(data)* acquisiamo il dato e lo salviamo in *data*, che è un vettore di *tipo byte*. Il dato ricevuto ha la seguente struttura "*nome;temperatura;umidità*", lo salviamo in un puntatore a char e sfruttiamo alcune funzioni delle stringhe. Quella usata qua è *stringtokenizer* che ci permette di dividere il vettore ricevuto quando incontriamo appunto un token, nel nostro caso ";". Il comando *atof(x)*, ascii to float, ci permette di convertire x in un float. Il primo parametro ricevuto è il nome dello Slave che sta trasmettendo, gli altri due sono temperatura e umidità lette nella serra.

```
if(Serial.read() == 'k'){

 Serial.println("Inserisci umidità max");
  while (Serial.available() == 0);
  humax = Serial.parseFloat();
  Serial.flush();

  Serial.println("Inserisci umididà min");
  while (Serial.available() == 0);
  humin = Serial.parseFloat();
  Serial.flush();

 Serial.println("Inserisci la temperatura max");
```

```
  while (Serial.available() == 0);
  tumax = Serial.parseFloat();
  Serial.flush();

 Serial.println("Inserisci la temperatura min");
  while (Serial.available() == 0);
  tumin = Serial.parseFloat();

 Serial.print("Letto: ");
  Serial.println(tumin);
  Serial.flush();
}
```

E ancora:

```
Mirf.setTADDR((byte *)name);

  snprintf(threshold, sizeof(toSend),
"%s;%d;%d;%d;%d;%d", name, (int)humax, (int)
humin, (int)tumax, (int)tumin, (int)crc);
   Serial.print("Soglie: ");
  Serial.println(threshold);
  Mirf.send((byte *)threshold);

   //Imposto un timeout oltre il quale stoppa la
trasmissione
   while(Mirf.isSending()){
   }
   Serial.println("Finished sending");
   delay(10);
   time = millis();
   while(!Mirf.dataReady()){
    if ( ( millis() - time ) > 1000 ) {
     Serial.println("Timeout Setting parametri");
     return;
    }
   }
```

Nella prima parte quando viene scritto il carattere "*k*" sulla seriale, il programma ci chiederà di inserire le nuove soglie minime e massime di umidità e temperatura che poi saranno inviate allo Slave.

Mentre nella seconda parte col comando *snprintf(threshold, sizeof(toSend), "%s;%d;%d;%d;%d;%d", name, (int)humax, (int)humin, (int)tumax, (int)tumin, (int)crc)* creiamo un vettore da inviare allo Slave, formato da il nome di chi trasmette, le soglie impostate e il CRC calcolato dal Master. Per comodità tutti parametri trasmessi sono trasformati in interi.

Nell'ultima parte è impostato un *timeout* per stoppare la trasmissione dopo un certo tempo.

2.2 PROCESSI MASTER

Un controllo che effettua il Master è che la trasmissione sia andata a buon fine per fare ciò calcola il CRC.

Generalmente per verificare la bontà dei dati ricevuti si ricorre alle *somme di controllo*, quelle più semplici sono quelle algebriche che però hanno alcuni limiti. Pertanto si utilizzano algoritmi appositi, conosciuti con l'acronimo di CRC (Cyclic Redundancy Check). Questo sfrutta una rete logica, all'interno della quale vengono introdotti i dati che scorrono in un registro. Durante lo scorrimento, vengono prelevati e combinati fra loro dei bit da determinate posizioni del registro: il risultato di questa operazione viene combinato con il bit in ingresso. In questo caso è stato utilizzato un CRC8, ovvero si ha in uscita una somma ad 8 bit. Il codice è il seguente:

```
//Calcolo il CRC-8, mi creo quindi un array di byte
byte bhmax = (byte)humax;
byte bhmin = (byte)humin;
byte btmax = (byte)tumax;
byte btmin = (byte)tumin;

byte crc32_str[4] = {
bhmax, bhmin, btmax, btmin      };

crc = CRC8(crc32_str);
Serial.println("CRC: ");
Serial.println(crc);
```

E ancora:

```
//Calcolo CRC-8 - algoritmo basato sulle formule
di CRC-8 di Dallas/Maxim
byte CRC8(const byte *data) {
byte crc = 0x00;

while (*data) {
byte extract = *data++;
for (byte tempI = 8; tempI; tempI--) {
byte sum = (crc ^ extract) & 0x01;
crc >>= 1;
if (sum) {
crc ^= 0x8C;
}
extract >>= 1;
}
}
return crc;
}
```

3. SLAVE

Di seguito è riportato il Flow Chart del programma:

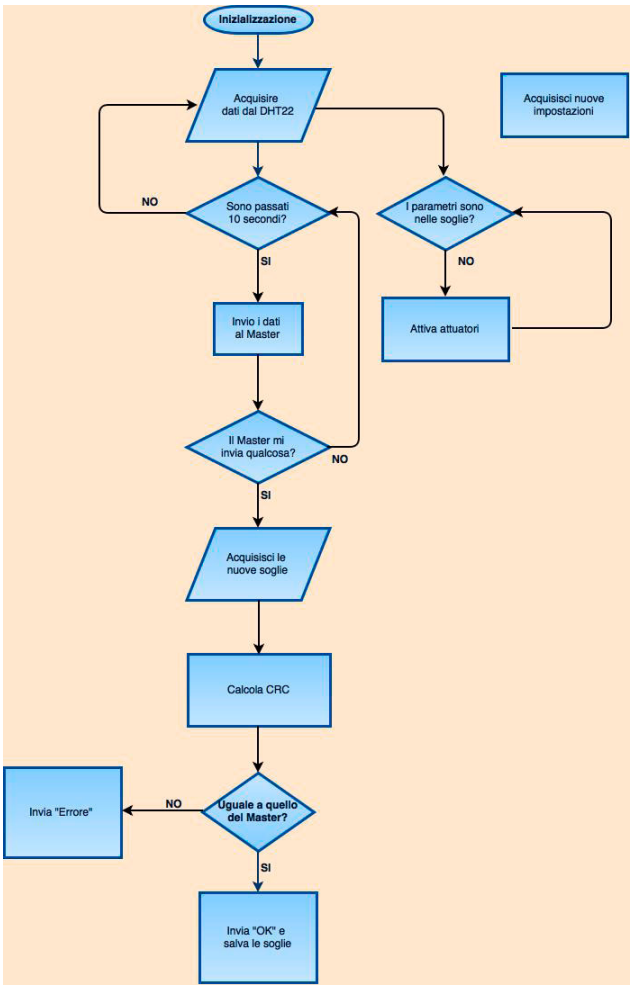


Figura 6 - Flow Chart Slave

Inizialmente vengono effettuate tutte le inizializzazioni sia per i sensori sia per la comunicazione. Inizia quindi, l'acquisizione di umidità e temperatura del DHT22, ed inizia il timer per l'invio dei dati misurati. Non avendo una necessita di real time inviamo i dati ogni 10 secondi.

Pertanto verrà avviata la conversazione con il Master, una volta inviati i parametri lo Slave si mette in ascolto per ricevere eventuali impostazioni. Se così fosse acquisisce questo pacchetto, ne estrae le informazioni di cui ha bisogno e si calcola il CRC e verifica se è uguale a quello ricevuto dal Master. Se così fosse risponde con "*OK*", altrimenti da "*Errore*".

Intanto il microcontrollore continua a verificare se i parametri letti siano o meno all'interno delle soglie massime e minime. Agendo di conseguenza sugli attuatori, che in questo caso sono un motore che apre e chiude una finestra, una pompa per l'irrigazione ed un riscaldatore.

3.1 PROCESSI SLAVE

Vediamo ora un po' di codice, in questa paragrafo descriviamo i processi controllati dal Master:

```
//Gestisco l'umidificatore, il riscaldatore e la vento-
la in funzione delle soglie e dei valori letti
//Riscaldatore
if(myDHT22.getTemperatureC() >= tmax-1){
digitalWrite(RISCALDATORE, HIGH);
}
if(myDHT22.getTemperatureC() <= tmin+1){
digitalWrite(RISCALDATORE, LOW);
}

//Umidificatore
if((int)myDHT22.getHumidity() >= hmax-1){
digitalWrite(UMIDIFICATORE, HIGH);
}
if((int)myDHT22.getHumidity() <= hmin+1){
digitalWrite(UMIDIFICATORE, LOW);
}

//Aprire finestra
```

```
if(myDHT22.getTemperatureC() >= tmax+4){
duty_motor = 20;
analogWrite(IN_2, duty_motor);
delay(TCONST);
}
if(myDHT22.getTemperatureC() <=
(tmax+tmin)/2 || finecorsa == LOW){
reset_ports();
duty_motor = 0;
analogWrite(IN_2, duty_motor);
delay(TCONST);
}
```

Controllo i parametri in serra e attivo o disattivo i vari attuatori, l'ultimo *if* mi attiva il motore con un duty del 20%. Dato che devo aprire e chiudere una finestra non ho bisogno di elevata velocità, ma di elevata coppia pertanto lavoro a duty fisso e basso.
Poi controllo che i dati vengano inviati ogni 10 secondi:

```
//Invio i parametri ogni 10 secondi
if(currentMillis - previousMillis > interval) {
previousMillis = currentMillis;
Mirf.setTADDR((byte *)"server");
snprintf(toSend, sizeof(toSend), "%s;%d;%d",
name, (int)humidity, (int)temperature);
Serial.println(toSend);
Mirf.send((byte *)toSend);

//Imposto un timeout oltre il quale stoppa la tra-
smissione
while(Mirf.isSending()){
}
Serial.println("Finished sending");
delay(10);

while(!Mirf.dataReady()){
if ( ( millis() - time ) > 1000 ) {
return;
}
}
}
```

Ed infine controllo che il CRC ricevuto sia uguale a quello che mi calcolo ora:

```
Mirf.getData(data);
char* command = strtok((char *)data, ";");
int count = 0;
while (command != 0){
//Divido le informazioni
switch (count){
case 0:
snprintf(name, sizeof(name), "%s", command);
break;

case 1:
hmax = atof(command)/10; //atof(char* ) mi con-
verte un tipo char* in double
break;

case 2:
hmin = atof(command)/10;
break;

case 3:
tmax = atof(command)/10;
break;

case 4:
tmin = atof(command)/10;
break;

case 5:
crc_ric = atoi(command);
break;
}
command = strtok(0, ";");
count++;
}
//Calcolo il CRC-8, mi creo quindi un array di byte
/**********************************************
*****************************/
byte bhmax = (byte)hmax;
byte bhmin = (byte)hmin;
byte btmax = (byte)tmax;
byte btmin = (byte)tmin;

byte crc32_str[4] = {bhmax, bhmin, btmax,
btmin};

crc = CRC8(crc32_str);
Serial.println("CRC: ");
Serial.println(crc);
```

```
/*********************************************
*****************************/

//Controllo i dati ricevuti
/*********************************************
*****************************/
if((byte)crc_ric == crc){
Serial.println("CONTROLLO DATI OK!");
Mirf.setTADDR((byte *)name);
Mirf.send((byte *)"OK");
}
else {
Serial.println("CONTROLLO DATI ERRORE!");
Mirf.setTADDR((byte *)name);
Mirf.send((byte *)"Errore");
}
/*********************************************
*****************************/
```

Utilizzando il CRC si riesce con due sole trasmissioni a validare la bontà dei dati ricevuti.

3.2 COMUNICAZIONE LATO SLAVE

La comunicazione lato Slave è molto semplice infatti quest'ultimo non fa altro che inviare le letture dei dati ogni 10 secondi e aspettare se il Master ha qualche impostazione da inviare.

```
//Invio i parametri ogni 10 secondi
if(currentMillis - previousMillis > interval) {
previousMillis = currentMillis;
Mirf.setTADDR((byte *)"server");
snprintf(toSend, sizeof(toSend), "%s;%d;%d",
name, (int)humidity, (int)temperature);
Serial.println(toSend);
Mirf.send((byte *)toSend);

//Imposto un timeout oltre il quale stoppa la tra-
smissione
while(Mirf.isSending()){
}
Serial.println("Finished sending");
delay(10);

while(!Mirf.dataReady()){
if ( ( millis() - time ) > 1000 ) {
return;
}
}
}

while(!Mirf.dataReady()){
if ( ( millis() - time ) > 1000 ) {
return;
}
}
```

L'intera comunicazione dello Slave avviene in questa parte dello sketch.

4. SERVER

La componente Server del nostro progetto si occupa di ricevere ed immagazzinare le rilevazioni temporaneamente memorizzate sul dispositivo Master e di comunicare a quest'ultimo eventuali cambi di configurazione inoltrati dall'utente tramite App Mobile (Cap. 5). Si compone di una componente software Java e di un database di tipo MySQL.

Il server viene lanciato su una macchina dedicata e si predispone a ricevere pacchetti in entrata sia dal Master che da App Mobile. All'arrivo di un package per prima cosa ne analizza l'integrità tramite controllo CRC, poi ne analizza il mittente: se proviene da App Utente, quindi si tratta di un set di comandi da inviare al Master, lo aggiunge alla coda dei pacchetti in uscita e il flusso continua; se invece si tratta di un pacchetto in arrivo dal Master, viene estratta la parte delle rilevazioni e viene salvata in due tabelle del DB, una contenente i dati in formato ROW per eventuali analisi e un'altra contenente i dati strutturati tramite un parser dedicato pronti per la visualizzazione. Viene inviata poi una risposta al Master che indica l'avvenuta ricezione e che può quindi

procedere a cancellare i suoi dati temporanei. Al termine della memorizzazione viene anche controllato se nella coda di pacchetti in uscita è presente un set di comandi da inviare al Master che ha appena comunicato, in tal caso invece della risposta OK viene inoltrato al Master il set di comandi, aspettando poi un OK da quest'ultimo per concludere il processo di comunicazione.

4.1 COMUNICAZIONE SERVER

Il Server riceve e inoltra pacchetti su protocollo TCP/IP istanziando un oggetto ServerSocket, secondo il seguente esempio:

```
String clientSentence;
String capitalizedSentence;
ServerSocket welcomeSocket = new ServerSocket(6789);

while(true)
{
Socket connectionSocket = welcomeSocket.accept();
BufferedReader inFromClient =
new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
clientSentence = inFromClient.readLine();
System.out.println("Received: " + clientSentence);

……..
}
```

Dopo aver costruito la risposta, viene inoltrata al mittente nel seguente modo

```
DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());

outToClient.writeBytes(response);
```

5. APP MOBILE

L'app mobile è l'interfaccia dell'utente per consultare i dati memorizzati sul Server ed effettuare eventuali cambi di configurazione al Master (soglie di umidità e temperatura, timeout, …..). Si compone di una app Android scritta in linguaggio Java.

L'App al suo avvio, mostra un menù tramite il quale è possibile visualizzare le rilevazioni memorizzate sul Server, indicando un range temporale o di sensori che si vuole visualizzare. Il risultato appare come grafico e mostra l'andamento temporale delle rilevazioni.

E' inoltre possibile scegliere dal menu di effettuare il cambio dei parametri, in questo caso vengono mostrati a video le configurazioni attuali, al salvataggio viene inoltrato un pacchetto al Server che provvederà ad instradarlo al dispositivo Master.

5.1 COMUNICAZIONE APP MOBILE

La comunicazione della App Mobile verso il Server avviene su protocollo TCP/IP, con un meccanismo analogo a quello del server.

CONCLUSIONI

Manca da sviluppare solo la comunicazione tra Master e Server.

L'idea che c'è dietro a questo progetto è quella di riportare l'agricoltura alla portata di tutti. Infatti disporre di un sistema di questo tipo, significa poter coltivare a casa tutto quello che si vuole riducendo di molto il tempo e la fatica da dedicarci.

Questo per ora è solo un prototipo semplice

dell'idea che vogliamo sviluppare, ma la nostra intenzione era quella di trasmettere la nostra visione di agricoltura.

È possibile, con un sistema ben sviluppato risparmiare anche più del 30% delle risorse naturali, ciò consente sia di abbattere i costi e sia di ridurre l'impatto ambientale.

Antonio La Mura & Umberto Festa

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-serra-domotica-open-source

Inverter ad onda sinusoidale Open Source con scheda Infineon Arduino motor shield

di vetrucci

PREPARAZIONE DELLA SCHEDA INFINEON MOTOR SHIELD

La scheda Infineon Motor Shield arriva senza piedini saldati per cui, per prima cosa, ho provveduto a saldare le strip maschio destinate ad accoppiarsi con gli I/O di Arduino. Vista la grande corrente (30A) in entrata ed in uscita dai capi di potenza, la scheda non presenta per quest'ultimi le tipiche piccole piazzole per strip ma fori più grandi con ampie zone in rame da entrambi i lati, io ho optato per avvitarci sopra quattro boccole1 per terminali a banana in modo da rendere comodi i collegamenti provvisori necessari in fase di prototipazione.

Il montaggio di tali boccole, anche tenute il più alto possibile (fig. 1), purtroppo, mal si adatta alla scheda Arduino Uno, visto che i dadi sottostanti tendono comunque ad urtare la porta di alimentazione e la porta USB dell'Arduino.

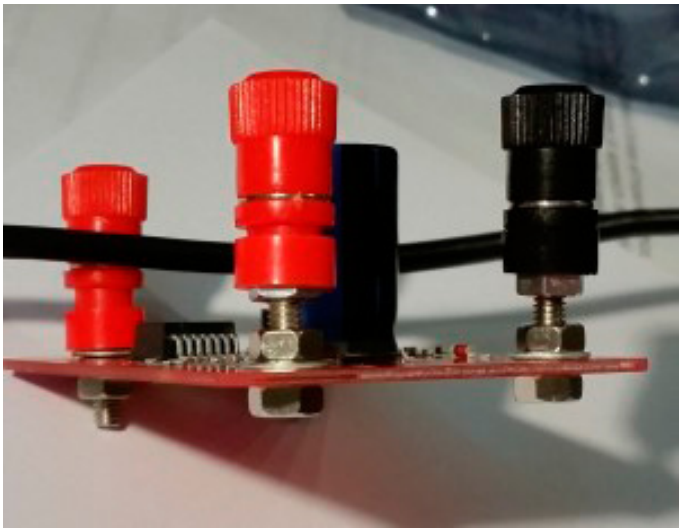


Fig. 1 – boccole montate in modo da creare il minimo ingombro sotto lo shield.

Ho risolto usando strip maschio leggermente più lunghe e ponendo nastro isolante sia sul dorso della presa USB che sul dado che inevitabilmente va a contatto con essa (fig. 2), per la presa dell'alimentazione ci sono meno problemi perché esternamente presenta materiale isolante.

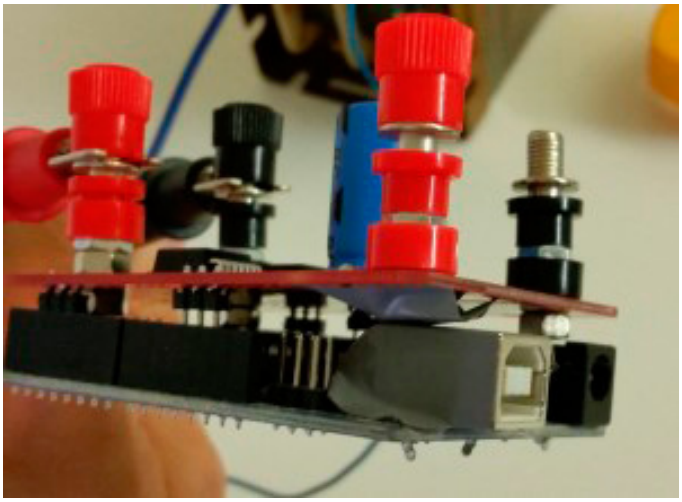


Fig. 2 – Nastratura (in grigio) per evitare contatti indesiderati.

MODALITÀ DI CONTROLLO DEI 2 HALF BRIDGE

La scheda presenta 2 integrati BTN8982TA che sono dei mezzi ponti (half bridge), per creare una tensione sinusoidale si può agire in due modi, il primo è comandare contemporaneamente i due half bridge con segnali sinusoidali sfasati di 180 gradi, il secondo è quello di far agire una sola metà del ponte alla volta tenendo costante la tensione in uscita dall'altra metà del ponte. (fig. 4)

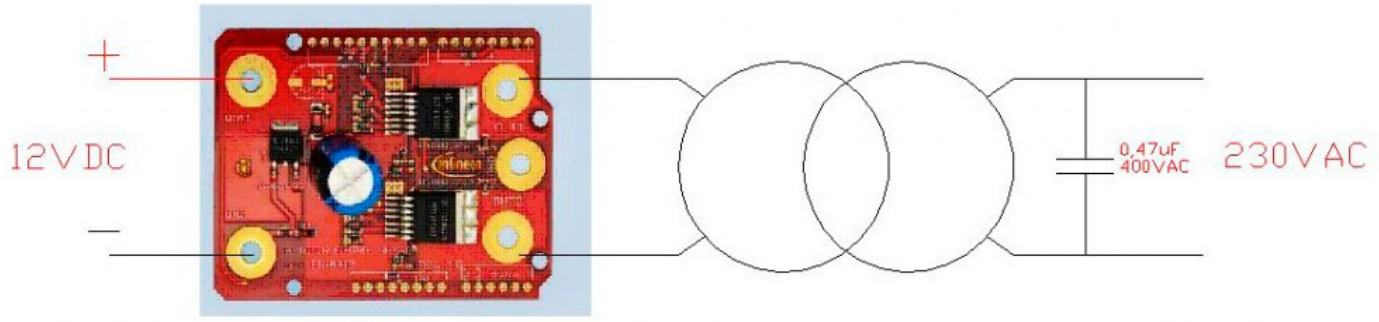


Fig. 3 - Schema elettrico solo due componenti oltre alle schede Arduino Uno e Infineon motor shield

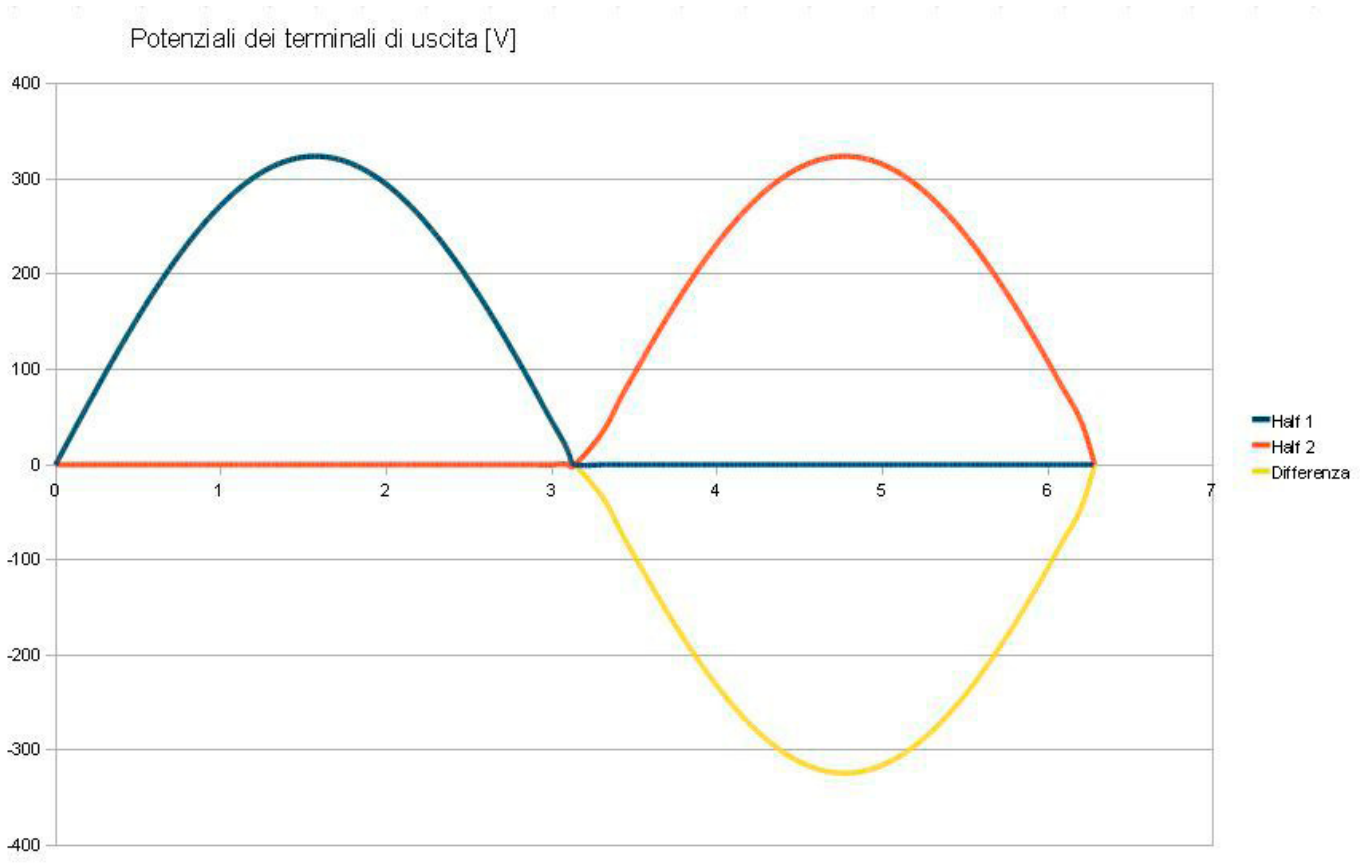


Fig. 4 – potenziale dei due terminali OUT e loro differenza, per un valore efficace di 230V il valore massimo è superiore ai 300V.

La seconda soluzione è migliore perché tenendo fisso uno dei due valori si evitano metà delle perdite di commutazione, per cui nella programmazione del firmware ho cercato di implementare questa tipo di controllo.

CONSIDERAZIONI SUI VALORI DI TENSIONE E POTENZA

L'inverter assorbe una corrente continua in ingresso e ne produce una alternata in uscita, fin qui nulla di nuovo, ma che valore presenta la tensione alternata? Per saperlo occorre ripassare un po' di teoria, il valore col quale si indica la tensione alternata, se non specificato altrimenti, è il valore efficace ovvero la radice quadrata della media dei quadrati, indicata in inglese con la scritta RMS (Root Mean Square), per un'onda sinusoidale (e solo per quella) il valore efficace

si può ottenere facilmente prendendo il valore di picco di una semionda (detto valore massimo) e dividendolo per radice di 2.

Vediamo di chiarire con un esempio pratico se diamo in ingresso 12V DC ad un ponte ad H in uscita possiamo avere al massimo una sinusoide che va da +12V a -12V il cui valore massimo è 12V, di conseguenza il valore efficace risulta di di 12/1,41 = 8,51V.

Se vogliamo realizzare un inverter in grado di erogare i 230Vac della rete domestica dovremo applicare un trasformatore con adeguato rapporto di trasformazione che, al contrario di quanto si può pensare, non è 230/8,51, in quanto la maggior parte dei trasformatori è pensata per ridurre la tensione e non per alzarla e presenta un rapporto di trasformazione reale diverso da quello di targa per compensare le cadute di tensione.

Ad esempio il trasformatore che ho utilizzato nel prototipo presenta Sn=100VA e rapporto nominale 220/12 e per ottenere 230 V in uscita ho dovuto alimentarlo con 20V in corrente continua.

Da questi ragionamenti ne consegue che per realizzare un inverter che dalla tensione di batteria auto ci porti a 230V occorre un rapporto di trasformazione di targa di 230/7,2V che non è un valore standard. In conclusione consiglio di utilizzare un rapporto di trasformazione standard con valore secondario di poco inferiore a 7,2 e di regolare la tensione alternata in ingresso tramite firmware in modo da ottenere sperimentalmente la tensione di uscita desiderata.

Per quanto riguarda la potenza massima del nostro inverter abbiamo il limite di 30A per cui se alimentiamo a 12V la potenza massima in ingresso sarà di 12x30=360W, anche il trasformatore dovrà presentare una potenza nominale maggiore o uguale a 360VA ed in uscita avremo verosimilmente almeno 300W erogabili 24h al giorno oppure potenze maggiori per tempi brevi.

PROGRAMMAZIONE

La realizzazione del firmware con l'IDE Arduino ha richiesto la soluzione di diversi problemi che non avevo preventivato, la difficoltà è dovuta ai 50Hz che, pur sembrando pochi, richiedono di creare una sinusoide in soli 20ms con istruzioni da eseguire in real time.

Non ho potuto utilizzare la funzione delay() in quanto accetta in ingresso solo valori interi di millisecondo e mi avrebbe consentito di approssimare la sinusoide con meno di 20 gradini (ritardo minimo 1 ms) per cui, dopo varie ricerche, ho trovato il listato della funzione delay() e modificandolo sono riuscito ad ottenere funzioni delay "personalizzate" con ritardo di 10 o 100us. Solo successivamente, in fase di revisione dell'articolo ho scoperto l'esistenza della funzione delayMicroseconds() che mi avrebbe evitato la fatica.

La forma d'onda in uscita comunque non era soddisfacente e ho scoperto che ciò era dovuto alla frequenza delle uscite PWM di Arduino che presentano di default un valore molto basso per lo scopo (488Hz). In rete ho trovato una funzione che permette di variare questo parametro ed ho portato la frequenza del PWM a 3904Hz.

Per creare la sinusoide al posto di calcolare la funzione seno con "sin()" eseguito n volte al ciclo ho optato per creare un vettore di 17 elementi con già inseriti i valori del seno di angoli da 0 a 90°. Ripercorrendo il vettore avanti e indietro una volta comandando una metà ponte una volta l'altra metà ho ricreato in uscita una sinusoide approssimata con ben 64 gradini.

Come visibile in fig. 5 la forma d'onda della tensione presenta molti picchi dovuti alla commutazione PWM per cui ho pensato di filtrarli appli-

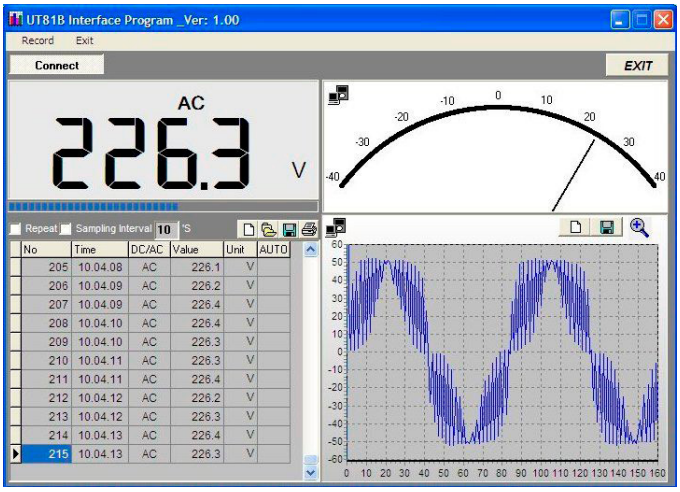


Fig. 5 – Forma d'onda dell'uscita con collegato un carico resistivo da 40W.

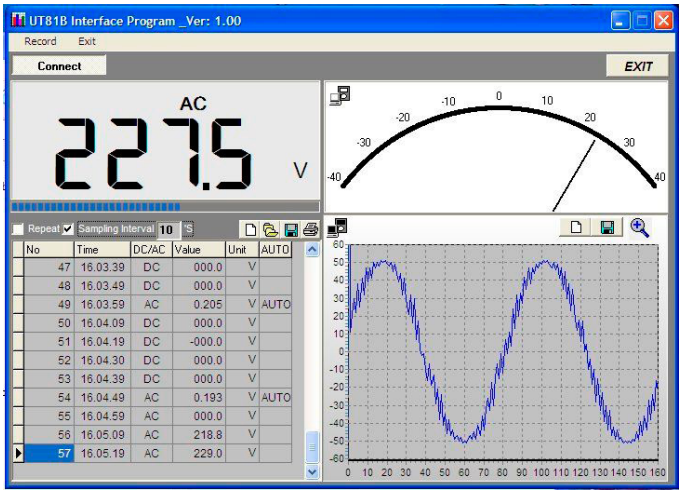


Fig. 6 – Forma d'onda dell'uscita con condensatore da 0,47uF e carico resistivo da 40W, la forma d'onda risulta sensibilmente migliorata.

cando in uscita un condensatore poliestere con tensione di lavoro 400VAC e 0,47uF ed ecco i risultati:

CONCLUSIONI

Con questo prototipo ho dimostrato che è possibile utilizzare la scheda Infineon, originariamente pensate per i motori in corrente continua, anche per realizzare un inverter. Prossimi passi possibili per sviluppare ulteriormente il progetto sono quelli di poter variare tensione e frequenza allo scopo di controllare motori asincroni.

1 I terminali di potenza sono cinque, ma per questo progetto non mi serviva la massa in uscita e quindi non ho installato la relativa boccola.

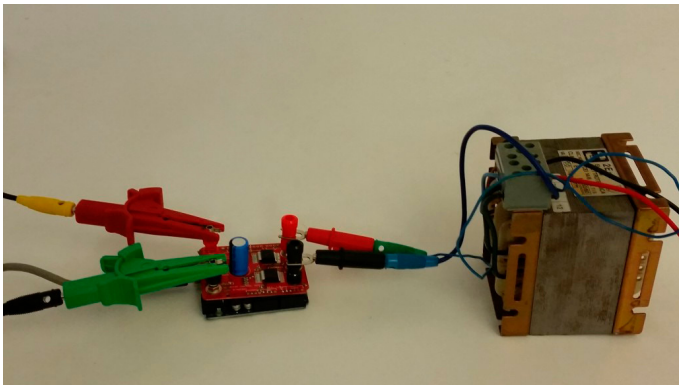


Fig. 7 - Il prototipo funzionante (ancora senza condensatore)

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/inverter-ad-onda-sinusoidale-open-source-con-scheda-infineon-arduino-motor-shield

Camera slider DIY low cost con Arduino [Progetto completo]

di Vincenzo Motta

L' obbiettivo di questo articolo è fondamentalmente quello di mostrarti il mio prototipo completamente testato e funzionante e renderlo **"open"** per chiunque volesse realizzarlo e migliorarlo.

Esistono varie soluzioni sul mercato, da quelle professionali e semi-professionali di marchi ormai noti a kit assemblabili stile 'fai da te', ma sono forse prodotti piuttosto cari per time-lapser e videomaker "entry-level". La 'mission' di questo articolo è sicuramente quella di ridurre notevolmente i costi, realizzando a casa propria un camera slider.

COS'È UN CAMERA SLIDER?

Il camera slider (chiamato anche "dolly") è una delle attrezzature più utilizzate nel cinema, nel film making e soprattutto nella fotografia. L'invenzione del dolly sembra attribuibile al regista e tecnico canadese-statunitense Allan Dwan, anche se qualcuno sostiene sia invece un brevetto del 1912 dell'italiano Giovanni Pastrone.

Si tratta di un sistema che si possa spostare in modo liscio e fluido per muovere una macchina da presa per realizzare video in movimento o per muovere una reflex per realizzare time-lapse in movimento. Solitamente un sistema del genere ha ruote che scorrono su binari, ma molti usano carrelli o sedie a rotelle. Io ho testato la mia centralina su un sistema dotato di un carrello che, grazie a dei cuscinetti a sfera, si muove su dei binari.

Un esempio tipico di slider **non** motorizzato è il seguente:

Per adattare la mia centralina con motore ad un camera slider non motorizzato, come quello sopra mostrato, ho chiesto supporto ad HackSpace Catania per la realizzazione di alcuni sostegni stampati in 3D. Nello specifico è stato realizzato un supporto in materiale PLA che grazie all' inserimento di due viti sorreggesse il mio motore stepper. Il motore stepper è stato poi ancorato a una cinghia agganciata al carrello dello slider.

Se non hai la possibilità di avere a tua disposizione una stampante 3D, esistono infiniti modi per realizzare un carrello, potresti anche adottare la soluzione **"skate dolly"** di Stavros Koulis da cui ho preso ispirazione per future evoluzioni del progetto e che ti mostro qui di seguito:

skete dolly non motorizzato

skate dolly motorizzato

BENEFICI DELL'UTILIZZO DEL CAMERA SLIDER

- fare piccoli movimenti di camera in maniera precisa e armoniosa
- giocare con la profondità di campo
- seguire un soggetto in movimento
- rivelare l'ambientazione che ci circonda
- creare dinamicità alla ripresa

TIME-LAPSE

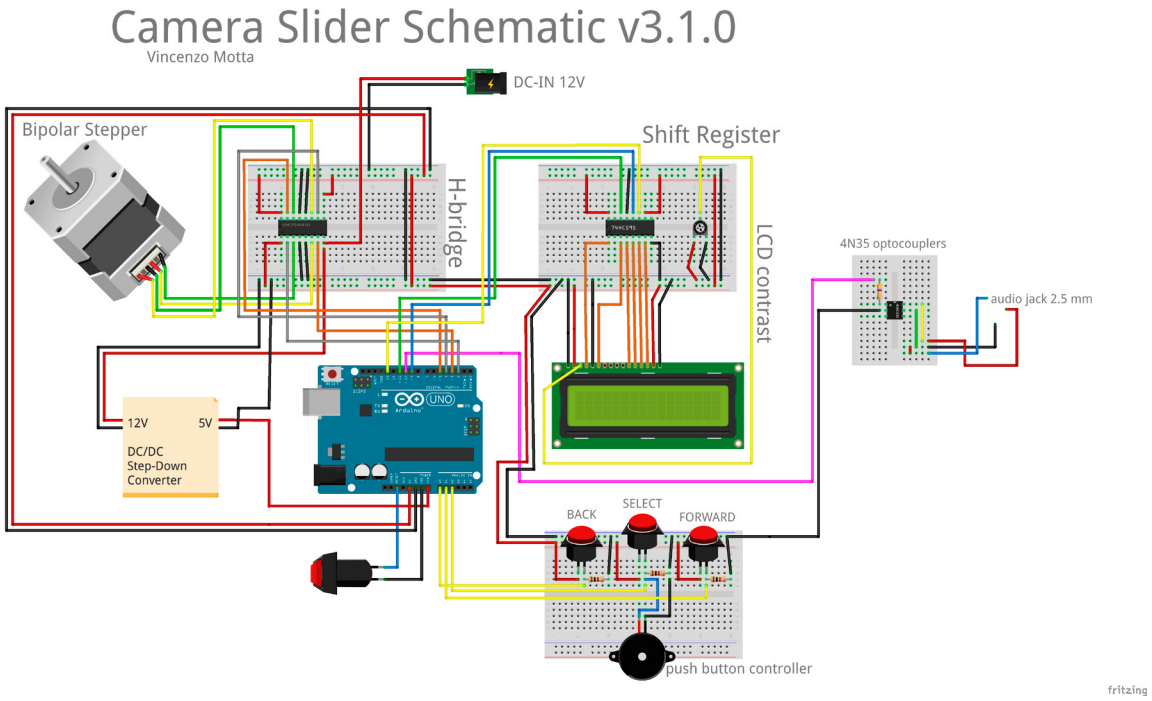
La più importante applicazione di un sistema camera slider è sicuramente la fotografia time-lapse.

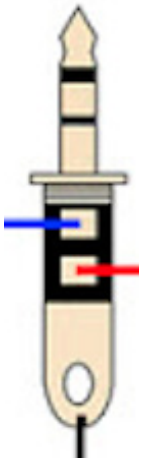
La **fotografia time-lapse** (dall'inglese 'time': "tempo" e 'lapse': "intervallo", quindi fotografia ad intervallo di tempo), o semplicemente **time-lapse**, è una tecnica cinematografica nella quale la frequenza di cattura di ogni fotogramma è molto inferiore a quella di riproduzione. A causa di questa discrepanza, la proiezione con un frame rate standard di 24 fps fa sì che il tempo, nel filmato, sembri scorrere più velocemente del normale.

Un filmato time-lapse può essere ottenuto processando una serie di fotografie scattate in sequenza e opportunamente montate o attraverso video che verranno poi accelerati. Filmati di livello professionale, vengono prodotti con l'ausilio di videocamere e fotocamere provviste di intervallometri ovvero di dispositivi di regolazione, del frame rate di cattura o della frequenza degli scatti fotografici, su uno specifico intervallo temporale; alcuni intervallometri sono connessi al sistema di controllo del movimento della telecamera in modo da ottenere effetti di movimento, quali panning e carrellate, coordinati a differenti frame rate. Il time-lapse trova un largo impiego nel campo dei documentari naturalistici. Mediante questa tecnica cinematografica, è infatti possibile documentare eventi non visibili ad occhio nudo o la cui evoluzione nel tempo è poco percettibile dall'occhio umano, come il movimento apparente del sole e delle stelle sulla volta celeste, il trascorrere delle stagioni, il movimento delle nuvole o lo sbocciare di un fiore.
La fotografia time-lapse è considerata una tecnica opposta alla fotografia ad alta velocità e non deve essere confusa con l'animazione a passo uno. Un esponente di rilievo nel campo del time-lapse è il regista e direttore della fotografia statunitense Ron Fricke che ha utilizzato questa tecnica nel cortometraggio IMAX *Chronos* (1985) e nel film *Baraka* (1992).

LA CENTRALINA ELETTRONICA

Di seguito il particolare del jack con i relativi collegamenti colorati:





COMPONENTI UTILIZZATI

- 1 Arduino Uno
- 1 Motore Stepper Bipolare(4 fili) mod. Nema 17
- 1 Ponte H integrato SN754410NE + zoccoletto
- 1Shift Register 74HC595 + zoccoletto
- 1 Display LCD dim.16×2 (16 colonne, 3 righe) con retroilluminazione blu
- 1 Trimmer da 10 kohm
- 4 Switch a pulsante
- 3 Resistori da 1kohm
- 1 Resistore 470kohm(protezione IR led fotoaccoppiatore)
- 1 Buzzer
- Convertitore DC/DC Step-Down da 12V a 5V
- Connettore Jack 2.5mm
- Fotoaccoppiatore 4N35

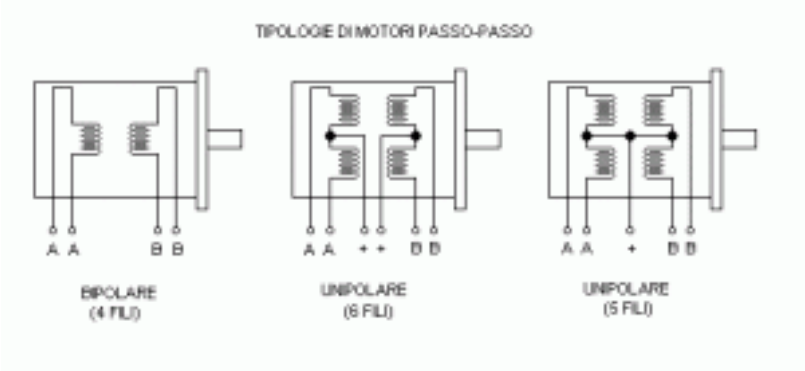
IL MOTORE STEPPER

I motori passo passo (o stepper) sono classificati in due tipologie distinte:

- **unipolari**
- **bipolari**

i motori unipolari sono più semplici da pilotare in quanto la sequenza della rotazione è esperessa dal succedersi dell'eccitazione delle bobine di cui è composto; i motori bipolari sono di maggiore complessià in quanto la rotazione avviene con un campo magnetico generato dalla corrente che percorra in entrambi i versi le bobine.

Da sottolineare la maggiore forza sviluppabile da un motore di tipo bipolare nei confronti dell'unipolare in quanto a insistere sulla rotazione sono 2 avvolgimenti in contemporanea e non uno solo.



Il numero di fili di cui un motore passo-passo possono essere 4,5,6 o 8 e sono direttamente legati alla caratteristica unipolare (5,6 o 8 conduttori) o bipolare (4 o 8 conduttori) del motore stesso.

I motori a 4 fili sono di tipo bipolare, come deducibile dallo schema in figura .

Pilotare questi motori è possibile facendo percorrere alla corrente gli avvolgimenti in entrambi i versi in modo alternato, inducendo la produzione del campo magnerico necessario alla corretta rotazione.

Ho deciso di utilizzare un motore passo-passo anziché un motore in corrente continua per avere una precisione maggiore.

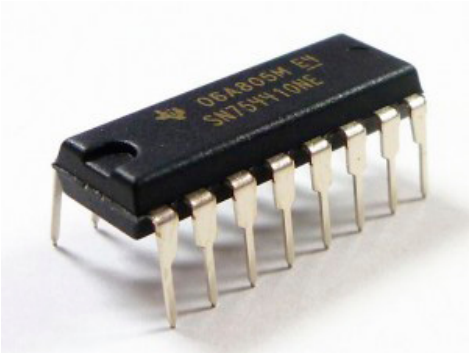
Questo perché **il motore in corrente continua lavora in tempi, mentre il motore passo-passo in distanze** (gli step).

Lavorando solo in tempi, i motori in corrente continua sono meno precisi dei motori passo passo. La precisione e la ripetibilità del motore in corrente continua dipende dal carico del motore stesso che deve essere costante (cosa alquanto difficile su una guida da circa 1 metro). D'altro canto i motori a corrente continua hanno un movimento più fluido. I motori passo passo lavorando a piccoli movimenti (step), hanno un funzionamento ad impulsi (se si aumenta la velocità diventano però impercettibili). Poiché lo scopo del progetto era quello di avere precisione di movimento ho optato per l' utilizzo del motore passo passo.

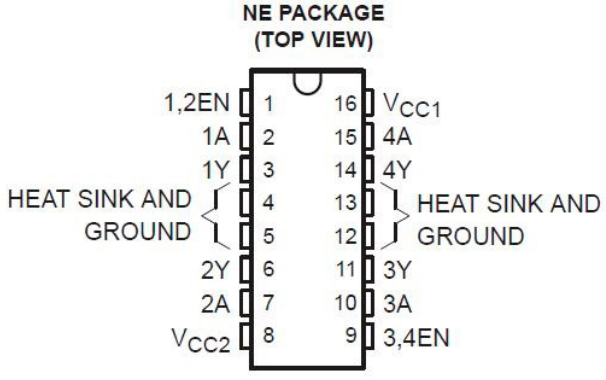
In particolare ho optato per un **motore che assorbisse circa 800mA per fase alimentato con una tensione di 12 V**.

PILOTARE IL MOTORE: L' SN-754410NE

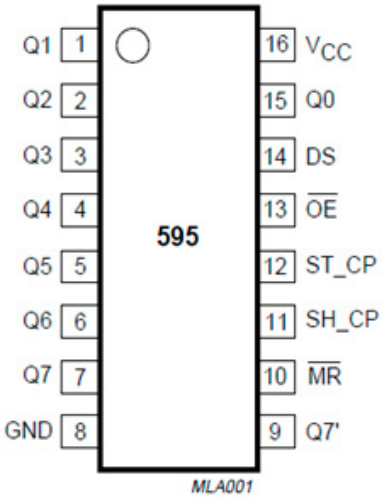
L' SN754410NE è un popolare H-Bridge (ponte-H) che può essere utilizzato per pilotare 2 motori in corrente continua, 1 motore stepper, solenoidi, grandi rele' o altri carichi induttivi. Sopporta una corrente di picco di 2A e un carico continuo di 1.1 Ampere per ogni stadio. Accetta tensioni di carico da 4.5V a 36V. Internamente è equipaggiato con i diodi e una protezione termica. Possono essere accoppiati 2 chips (uno sopra l'altro) per aumentare la corrente. Per smaltire al meglio il calore, dovuto alla gestione di grosse correnti in gioco, bisogna prevedere un dissipatore. E' compatibile pin to pin con l'L293D che è un altro popolare ponte-H.



qui di seguito il pinout:



è da notare che i piedini 4 e 13 si occupano di dissipare il calore se questo integrato viene saldato in un PCB dotato di piste per la dissipazione del calore.



PINNING

PIN	SYMBOL	DESCRIPTION
1	Q1	parallel data output
2	Q2	parallel data output
3	Q3	parallel data output
4	Q4	parallel data output
5	Q5	parallel data output
6	Q6	parallel data output
7	Q7	parallel data output
8	GND	ground (0 V)
9	Q7'	serial data output
10	$\overline{MR}$	master reset (active LOW)
11	SH_CP	shift register clock input
12	ST_CP	storage register clock input
13	$\overline{OE}$	output enable (active LOW)
14	DS	serial data input
15	Q0	parallel data output
16	V_{CC}	positive supply voltage

74HC595 SHIFT REGISTER

Il 74HC595 è uno Shift Register ad 8 bit che ho usato per pilotare il display LCD utilizzando soltanto 3 pin del microcontrollore anziché 6. In altre parole, questo componente mi permette di ridurre il numero di pin utilizzati da un dispositivo. Tra Arduino e lo shift register vi è una comunicazione di tipo SPI. Sarà infatti importante nello sketch Arduino andare a includere la libreria per la comunicaizone SPI in modo da garantire il corretto funzionamento dell' LCD.

Per stabilire una corretta comunicazione bisogna capire 'chi da ordini e chi li riceve', bisogna quindi stabilire chi è il 'Master' e chi lo 'Slave' nella comunicazione.
Arduino è il "Master" e lo shift register è lo "Slave". A questo scopo abbiamo bisogno che i pin:

- Data (MOSI),
- Clock (SLCK)
- Latch (Slave Select)

siano collegati.

Il MISO (Master In Slave Out) non è necessario perché non abbiamo intenzione di leggere i dati dal display LCD.

Durante la comunicazione i dati vengono trasferiti nello shift register (74HC595) attraverso i pin N°14 (Serial data in) e 11 (Shift register Clock). Una volta finito l' invio dei dati, basta portare a livello alto il pin 12 (Storage Clock o Latch Clock) per trasferire i dati dallo shift register al latch. Il pin 13 (Output Enable – attivo basso) consente il trasferimento dei dati dal latch alle porte di uscita quando viene portato a livello basso: avendo noi collegato tale pin fisso a massa, i dati li ritroveremo sulle uscite non appena diamo il colpo di clock sul latch. Il pin 10 è il master reset: ponendolo a livello basso resetta il contenuto dello shift register, per cui lo teniamo alto per sicurezza.
Il dato serializzato viene trasferito durante la transizione del clock da livello alto a livello basso. I bit vengono inviati dal più significativo verso il meno significativo, per cui dobbiamo tenere conto di questo nella routine di serializzazione per fare in modo che i bit si ritrovino sulle uscite nel giusto ordine.

Qui di seguito la connessione che ho eseguito tra l' 74HC595 e Arduino:

- *(MOSI)*: connetti il pin 14 (DS) del

74HC595 al pin 11 di Arduino.

- *(SCK)*: connetti il pin 11 (SHCP) del 74HC595 al pin 12 di Arduino.
- Il *Latch*: connetti il pin 12 (STCP) del 74HC595 al pin 9 di Arduino.

CONVERTITORE STEP- DOWN

Il convertitore Step-Down è un importantissimo componente in questo sistema. Lo step-down si occupa di convertirmi i 12V di tensione in ingresso in 5V di tensione in uscita. Nel progetto che presento in questo articolo è stato necessario utilizzare qualcosa che mi disaccoppiasse l'alimentazione tra Arduino e il motore stepper, poiché, a causa del fatto che il motore stepper è un carico induttivo soggetto a spike di corrente durante l' eccitazione delle fasi. Gli spike di corrente negli avvolgimenti del motore provocano del display LCD uno "sfarfallio" poco gradevole per una buona visione dell' LCD e garanzia di una breve durata del dispositivo.

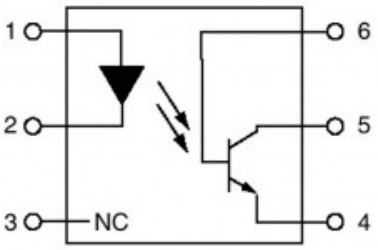
COLLEGAMENTO DELLA CENTRALINA ELETTRONICA ALLA FOTOCAMERA

Per non rischiare di danneggiare la camera, ho interposto un foto-accoppiatore modello 4N35 tra Arduino e la camera.

Questo componente permette di separare fisicamente due circuiti elettronici. Il foto accoppiatore non è altro che una specie di relè elettronico. All'interno è presente un led ed un sensore.

Facendo accendere il led (circuito 1) un sensore (nel circuito 2 separato dal circuito 1) chiuderà un interruttore. In questo modo non ci sarà nessun contatto elettrico tra Arduino e la nostra fotocamera.

Il foto accoppiatore, essendo per l'appunto un LED non puo' essere collegato direttamente ad un pin di Arduino. Bisogna interporre una resistenza, nel progetto ho utilizzato una resistenza da 470kΩ. Oltre alla resistenza, in serie, puoi inserire un LED in modo tale da avere una segnalazione visiva del comando di scatto.



PIN 1. ANODE
2. CATHODE
3. NO CONNECTION
4. EMITTER
5. COLLECTOR
6. BASE

Connessione:

- il pin 1 al pin 10 di Arduino che utilizzerai per pilotare il foto-accoppiatore, passando da una resistenza di 470Kohm;
- il 2 a massa (GND);
- il pin 4 ad un estermo del cavo del jack di scatto remoto della tua macchina fotografica;
- il pin 5 all' altro estremo del cavo del jack di scatto remoto;

Nello schema dell' intero sistema avrai notato che 2 pin sono ponticellati assieme, sono i contatti dell' auto-focus che deve essere premuto perchè la macchina metta a fuoco e non deve essere aperto durante lo scatto.

COME FUNZIONA?

Come ormai hai ben capito, un camera slider ti permette di eseguire scatti in modalità time lapse, questo è possibile grazie alla sincronizzazione tra numero passi del motore e scatto della camera. Attraverso l' LCD è possibile visualizzare un menù con le seguenti voci:

- **Configura**
- **Check**: permette di fare un check delle impostazioni eseguite in fase di configurazione
- **Reset Carrello**: permette di impostare il punto di "start" del carrello sul binario. Muovendoti con i pulsanti "BACK" e "FORWARD" puoi decidere dove posizionare di preciso il punto di start del carrello.
- **Avvio**: fa partire il carrello e la sequenza di scatti impostata

Giunti alla voce "Configura" si entra nel sottomenù:

- **Passi x scatto**: imposta il numero di passi del motore stepper tra uno scatto e l'altro della camera
- **Intervallo passi**: imposta il ritardo (in secondi) tra due passi successivi del motore stepper
- **Numero scatti**: imposta il numero di scatti che la nostra camera dovrà eseguire grazie all' ausilio dell' input mandato da Arduino al fotoaccoppiatore .

È possibile muoversi all' interno del menù grazie all' uso dei pulsanti illustrati nello schematico del progetto:

- **BACK** (<< indietro)
- **FORWARD** (avanti>>)
- **SELECT** (seleziona)

Quando premerai il pulsante SELECT il buzzer emetterà un suono per segnalare l' avvenuta selezione di una voce del menù.

IL CODICE

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



// Input Digitali
int prev = A0;
int next = A1;
int conf = A2;

// Stepper
int motorPin1 = 5;
int motorPin2 = 4;
int motorPin3 = 2;
int motorPin4 = 3;
int nFase=1;

// Led
int pinLed = 10;

// Menu Level
char* menu_principale[4]  =
{"Configura","Check","Reset Carrello","Avvio"};
char* submenu[3]        = {"passi x
scatto","Intervallo passi","Numero scatti"};

// Init level
int ngiri=0;
float interv=0;
int scatti=0;

int posizione=0;

// initialize the library with the numbers of
the interface pins
LiquidCrystal lcd(9);

void setup() {
  lcd.begin(16, 2);
  lcd.print("Camera Slider");
  lcd.setCursor(0, 1);
  lcd.print("Vincenzo Motta");
  delay( 1000 );
  lcd.clear();
  lcd.print( menu_principale[posizione] );

  digitalWrite( pinLed, LOW );

  pinMode(motorPin1, OUTPUT);
```

```
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);


  pinMode(pinLed, OUTPUT);
}

void loop() {
  lcd.setCursor(0, 1);
  int pnext = analogRead( next );
  int pprev = analogRead( prev );
  int pconf = analogRead( conf );

  if ( pnext > 1000 || pprev > 1000 || pconf >
1000)
  {
    if ( pnext > 1000 ) { posizione++; lcd.cle-
ar(); }
    if ( pprev > 1000 ) { posizione--; lcd.cle-
ar(); }

    if ( posizione > 3 ) posizione = 0;
    if ( posizione < 0 )  posizione = 3;

    lcd.print( menu_principale[posizione] );
    //lcd.print(millis()/1000);
    delay(200);

    if ( pconf > 1000 ) {
      lcd.clear();
      switch ( posizione )
      {
        case 0:
          ngiri=0;
          interv=0;
          scatti=0;
          Setting();
          lcd.clear();
          lcd.print( menu_principale[0] );
        break;

        case 1:
          View();
          lcd.clear();
          lcd.print( menu_principale[1] );
        break;

        case 2:
          ResetCarrello();
          lcd.clear();
          lcd.print( menu_principale[2] );
        break;

        case 3:
          Go();
          lcd.clear();
          lcd.print( menu_principale[3] );
        break;
      }
    }
  }
}

void Setting()
{
  int i = 0;
  boolean message = true;

  while ( i < 3 )
  {
    int pnext = analogRead( next );
    int pprev = analogRead( prev );
    int pconf = analogRead( conf );

    if ( message )
    {
      lcd.setCursor(0, 0);
      lcd.print( submenu[i] );
      lcd.setCursor(0, 1);
      message = false;
    }

    if ( pnext > 1000 || pprev > 1000 || pconf >
1000)
    {
      if ( pnext > 1000 )
      {
        if ( i == 0 ) { ngiri++; lcd.setCursor(0,
1); lcd.print( ngiri ); }
        if ( i == 1 ) { interv += 0.5; lcd.setCur-
sor(0, 1); lcd.print( interv ); lcd.setCursor(6,
1); lcd.print( "sec" ); }
        if ( i == 2 ) { scatti++; lcd.setCursor(0,
1); lcd.print( scatti ); }
      }
      if ( pprev > 1000 )
      {
        if ( i == 0 ) { ngiri--; lcd.setCursor(0, 1);
lcd.print( ngiri ); }
        if ( i == 1 ) { interv -= 0.5; lcd.setCur-
sor(0, 1); lcd.print( interv ); lcd.setCursor(6,
```

```
1); lcd.print( "sec" ); }
      if ( i == 2 ) { scatti--; lcd.setCursor(0,
1); lcd.print( scatti ); }
    }
    if ( pconf > 1000 )
    {
     lcd.clear();
     i++;
     message = true;
    }
   }
  delay( 200 );
 }
}

void View()
{
 for (int i=0; i<3; i++)
 {
  lcd.clear();
  lcd.print( submenu[i] );
  lcd.setCursor(0, 1);

  if ( i == 0 ) { lcd.print( ngiri ); }
  if ( i == 1 ) { lcd.print( interv ); lcd.setCur-
sor(6, 1); lcd.print( "sec" ); }
  if ( i == 2 ) { lcd.print( scatti ); }

  delay( 1000 );
 }
}

void ResetCarrello()
{
 lcd.clear();
 lcd.print( "+/- sposta carr" );
 lcd.setCursor(0, 1);
 lcd.print( "C esce" );
 int i = 0;

 while ( i < 1 )
 {
  int pnext = analogRead( next );
  int pprev = analogRead( prev );
  int pconf = analogRead( conf );

  if ( pnext > 1000 )
  {
   nFase++;
   if ( nFase > 4 ) nFase = 1;
   gira( nFase );
  }
  if ( pprev > 1000 )
  {
   nFase--;
   if ( nFase < 1 ) nFase = 4;
   gira( nFase );
  }
  if ( pconf > 1000 )
  {
   lcd.clear();
   i++;
  }
  delay ( 200 );
 }
}

void Go()
{
 lcd.print( "Avvio del dolly" );
 lcd.clear();

 for ( int sc=0; sc <= scatti; sc++ )
 {
  lcd.setCursor(0, 0);
  lcd.print( "scatto: " );
  lcd.setCursor(8, 0);
  lcd.print( sc );

  for ( int ng=1; ng <= ngiri; ng++ )
  {
   lcd.setCursor(0, 1);
   lcd.print( "passo: ");
   lcd.setCursor(8, 1);
   lcd.print( ng );

   gira( nFase );
   nFase++;
   if ( nFase > 4 ) nFase = 1;

   delay( interv * 1000 );
  }

  digitalWrite( pinLed, HIGH );
  delay( 1000 );
  digitalWrite( pinLed, LOW );

 }

}

void gira( int nFase ) {
```

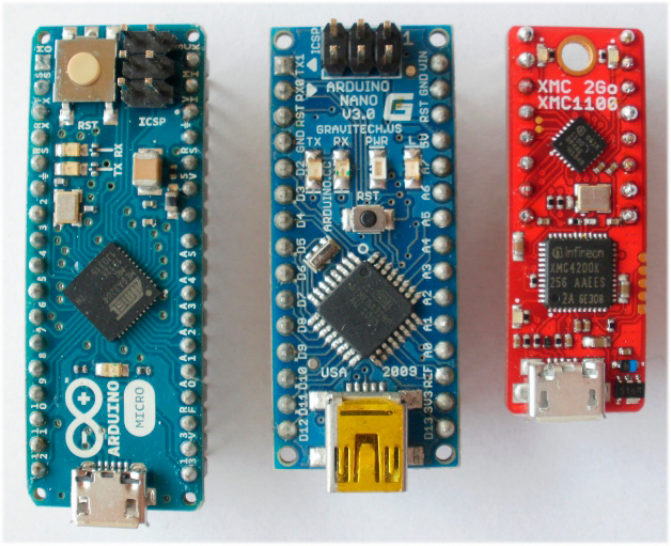
```
  switch( nFase )
  {

  case 1:
    digitalWrite(motorPin1, HIGH);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  break;

  case 2:
    digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, HIGH);
  digitalWrite(motorPin4, LOW);
  break;

  case 3:
    digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, HIGH);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, LOW);
  break;

  case 4:
    digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  digitalWrite(motorPin3, LOW);
  digitalWrite(motorPin4, HIGH);
  break;

  }
}
```

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/camera-slider-diy-low-cost-con-arduino-progetto-completo

Arduino Micro e BMP180 Bosch per realizzare una Weather Station

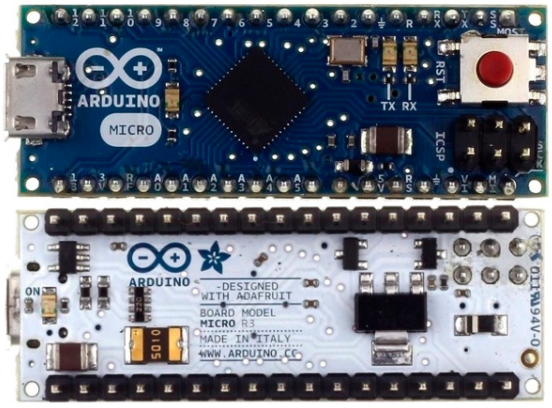
di Vincenzo Motta

Di schede di controllo di piccole dimensioni ve ne sono molte in commercio, nel caso di Arduino in un precedente articolo **Costruzione del robot LittleBot – Scheda di controllo**, avevamo visto per esempio **Arduino Nano**, mentre in **un'altro articolo** si vede un possibile utilizzo di una scheda **XMX 2Go** prodotta dalla Infineon.



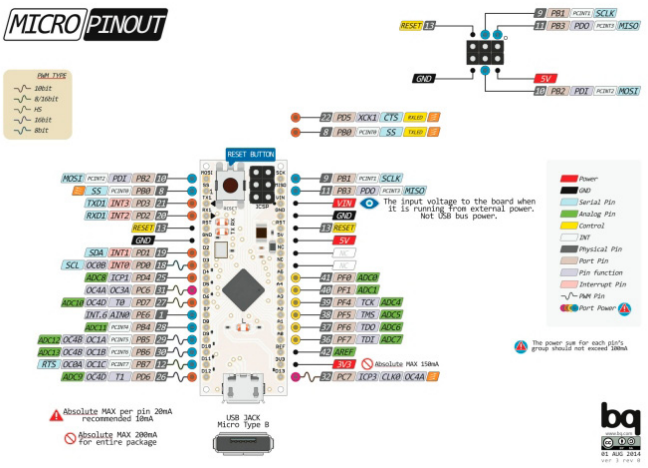
Confronto di dimensioni tra Arduino Micro, Nano e la scheda XMC 2Go

Vediamo ora di analizzare in dettaglio le caratteristiche offerte da **Arduino Micro**

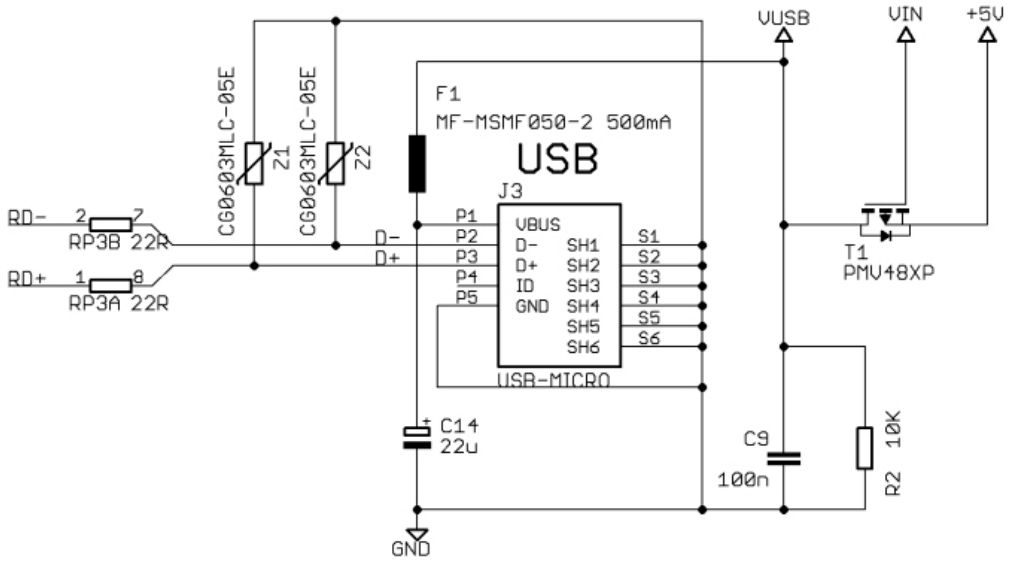


CARATTERISTICHE SALIENTI

Microcontroller	Atmel ATmega32u4
Velocità di clock	16 MHz
Tensione di funzionamento	5 V
Tensione in ingresso	7-12 V (consigliati)
Tensione in ingresso	6-20 V (limiti)
Pin I/O digitale	20 (7 forniscono in uscita segnali PWM e 10 possono essere usati come input analogici e 3 solo digitali)
Pin analogici	6
Corrente continua per I/O	40 mA
Corrente per Pin alimentati a 3.3V:	50mA
Flash Memory	32 KB (di cui 4 KB utilizzati dal bootloader)
SRAM	22.5 KB
EEPROM	1 KB
Dimensioni	48 x 18 mm



Passiamo quindi ad analizzare le varie sezioni in cui può essere suddiviso lo **schema elettrico** della scheda: **alimentazione, processore, memoria, pin d'input e output, connettore ICSP, led di segnalazione**.



La funzione della resistenza R2 da 10kΩ è quella di non far fluttuare il segnale di comando del mosfet T1.
L'alimentazione esterna (non USB) può essere fornita da un alimentatore DC o tramite una batteria, questi dovranno essere collegati ai pin **Gnd** e **Vin**.

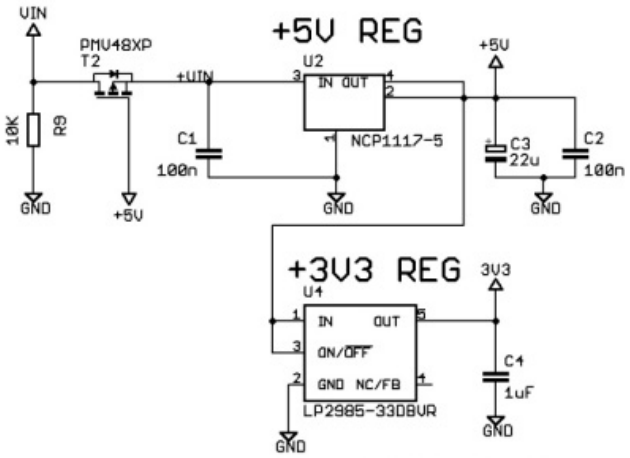
SEZIONE ALIMENTAZIONE

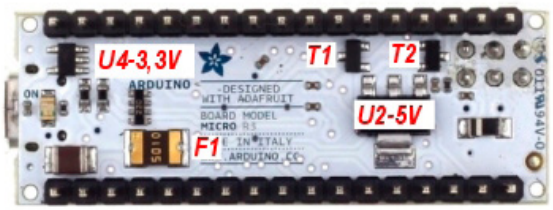
Arduino Micro può essere alimentato tramite **due diverse fonti di energia**: prelevata tramite il connettore dell'interfaccia **USB** micro o con un alimentatore esterno connesso al **Pin Vin**.

La fonte di alimentazione è selezionata automaticamente, tramite l'attivazione del mosfet, **T1** tipo **FND340P** o **PMV48XP**, usato come switch, che si occupa di connettere o meno i 5V USB alla scheda.

L'alimentazione da USB è protetta da un POLYFUSE azzerabile **F1** tipo **MF-MSF050-2** da 500 mA, che protegge la porta USB del computer da cortocircuiti e sovracorrenti.
Sebbene la maggior parte dei computer abbia già una protezione interna, il fusibile fornisce un ulteriore livello di protezione.
In questo caso, se una corrente maggiore di 500 mA è prelevata dalla porta USB, il fusibile interromperà automaticamente il collegamento fino a quando il sovraccarico non sarà rimosso.
La tensione è livellata e filtrata tramite il condensatore C9 da 100 nF e l'elettrolitico C14 da 22uF.

La scheda può funzionare con una tensione che teoricamente può essere compresa tra i 6 e i 20 volt.
In pratica però, se si forniscono meno di 7V, la tensione continua generata potrebbe essere inferiore ai 5V e in questo caso il funzionamento della scheda può essere instabile.
D'altra parte, fornire una tensione maggiore di 12V, può portare al surriscaldamento con relativo danneggiamento del regolatore di tensione.
Per questo motivo l'intervallo raccomandato di tensione è compreso **tra i 7 e i 12 volt**.
La parte del circuito di regolazione è formata per quanto riguarda l'uscita a +5V, dal'integrato **U2** tipo **NCP1117-5**, che è un regolatore di tensione fisso a basso dropout con una corrente d'uscita massima di 1 A.
La tensione d'uscita è livellata tramite i condensatori C1 e C2 da 100nF e dall'elettrolitico C3 da 22μF.
Nel caso fosse disponibile la tensione di +5V fornita tramite la porta USB, oppure da fonte esterna tramite l'apposito PIN, l'attivazione del secondo mosfet, **T2** tipo **FND340P** o **PMV48XP**, usato anche in questo caso come switch, si occupa di bypassare l'integrato **U2**.





Nel caso invece, la tensione da USB non fosse presente, la tensione sarebbe prelevata tramite l'ingresso **Vin**.

La funzione della resistenza R9 da 10kΩ è quella di non far fluttuare il segnale di comando del mosfet.

Arduino Micro è anche in grado di fornire in uscita una tensione di +3,3V, questa non è direttamente utilizzata dalla scheda ma è presente per utilizzi esterni.

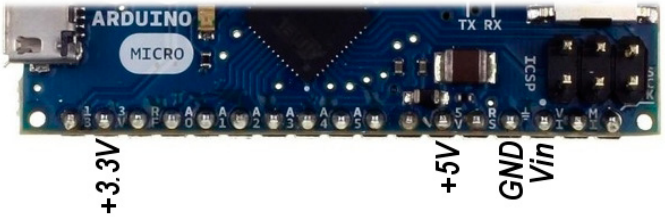
Per ottenere questa tensione che è ottenuta riducendo la tensione di +5V, è presente l'integrato **U4** tipo **LP2985-33DBVR**, si tratta di un regolatore di tensione fissa a basso rumore e basso dropout con possibilità di opzione shutdown, nel nostro caso non utilizzata dato che il pin risulta permanentemente connesso alla fonte di alimentazione.

La massima corrente d'uscita teorica è di 150 mA, anche se dalle caratteristiche della scheda è consigliato un assorbimento massimo di 50 mA.

La tensione è livellata dal condensatore elettrolitico C4 da 1uF.

Riassumendo i pin di alimentazione presenti sulla scheda Arduino Micro, sono i seguenti:

- **VI**. La tensione d'ingresso alla scheda Arduino quando sta utilizzando una sorgente di alimentazione esterna (in contrapposizione a 5 volt dalla connessione USB o altra fonte di alimentazione regolata).
- **5V**. tensione stabilizzata utilizzata per alimentare il microcontrollore e altri componenti sulla scheda. Questo può venire sia da VIN tramite un regolatore a bordo o essere fornita da USB o da un fonte esterna con un valore di 5V.
- **3V**. Sono forniti 3.3 volt generati dal regolatore di bordo. Assorbimento massimo è di 50 mA.
- **GND** pin di terra.



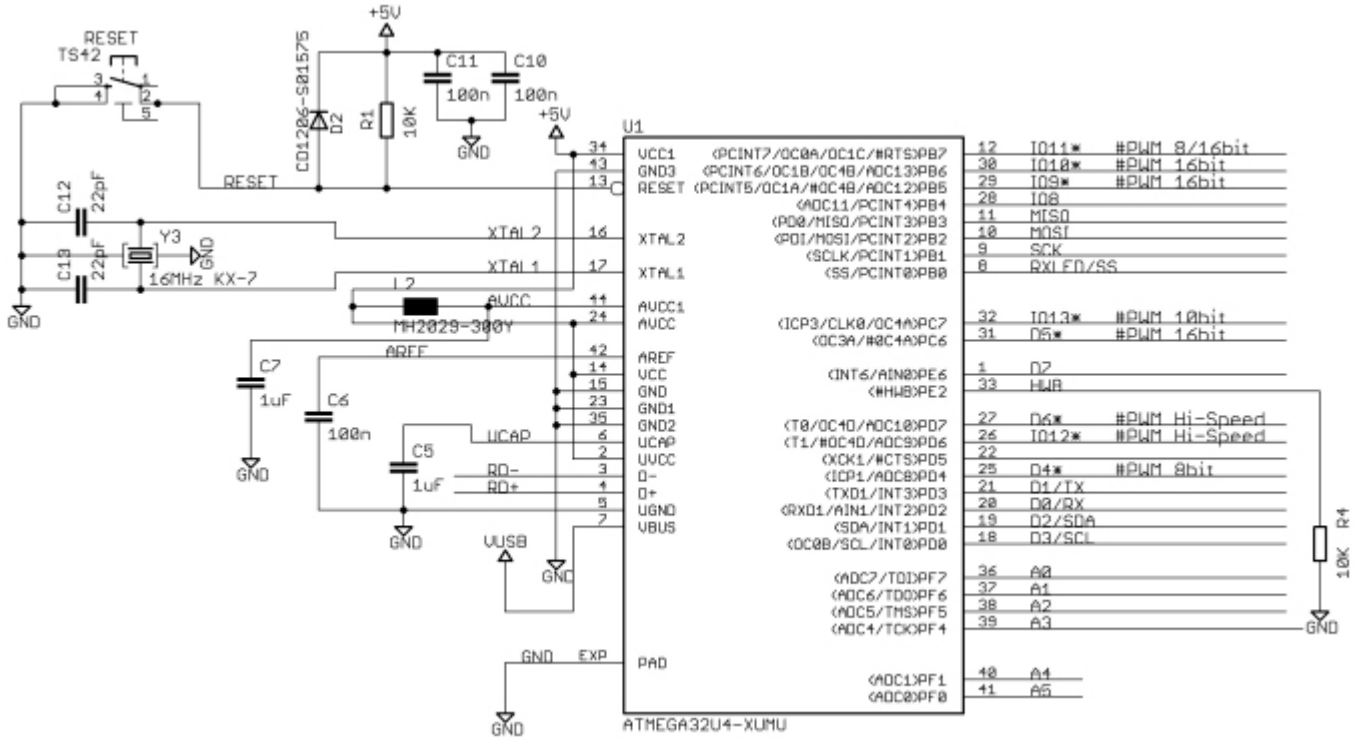
SEZIONE PROCESSORE

La scheda utilizza lo stesso tipo di processore della **Arduino Leonardo** e **Arduino Esplora**, un microcontrollore prodotto dalla AVR tipo **ATMEGA32U4** siglato sullo schema **U1**, che opera a una frequenza di **16 MHz**: è connesso a una porta USB ed è in grado di agire come un dispositivo client USB, come un mouse o una tastiera.

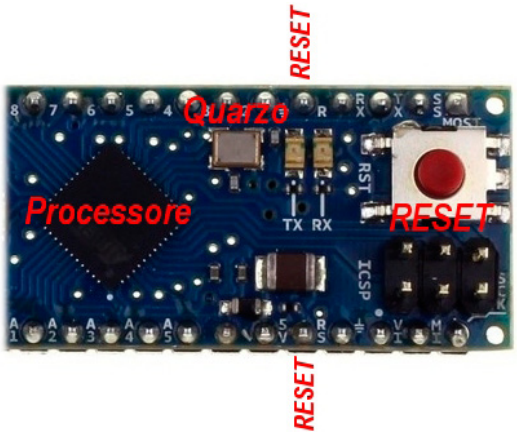
L'ATMEGA32U4 appare al computer come una porta COM virtuale, lo stesso processore funziona anche come dispositivo USB secondo lo standard 2.0.

L'integrato supporta, inoltre, la comunicazione **SPI** a cui si può accedere tramite la **libreria SPI**.

Nel circuito è presente un pulsante di reset (duplicato



anche su due pin della scheda), con cui è possibile riavviare la scheda.



MEMORIA

Il processore **ATMEGA32U4** ha 32 KB di memoria flash, anche chiamata flash memory, di cui 4 KB sono utilizzati per il bootloader.

Sono poi presenti 2,5 KB di **SRAM** (acronimo di **S**tatic **R**andom **A**ccess **M**emory), che è un tipo di RAM che non richiede refresh.

E' inoltre presente 1 KB di memoria **EEPROM** (acronimo di **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory), in cui è possibile memorizzare piccole quantità di dati che devono essere mantenuti quando è tolta l'alimentazione elettrica (per esempio la configurazione di un dispositivo).

Per la lettura/scrittura di quest'ultimo tipo memoria è possibile utilizzare la **libreria EEPROM**.

PIN DI INPUT E OUTPUT

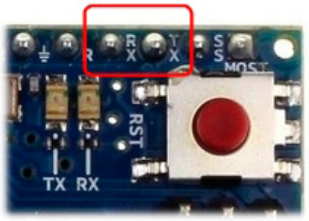
Ciascuno dei 20 I/O pin digitali di Arduino Micro può essere utilizzato sia come ingresso che come uscita, per fare questo si utilizzeranno le funzioni **pinMode ()**, **digitalWrite ()**, e **digitalRead ()**.

I pin operano a 5 volt e ognuno può fornire o ricevere un massimo di 40 mA di corrente.

Internamente è presente una resistenza di pull-up (sconnessa di default) del valore di 20-50 kΩ.

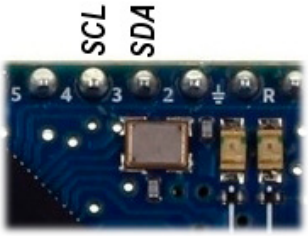
Alcuni pin hanno funzioni specializzate.

Serial: 0 (RX) e 1 (TX). La scheda è dotata di due pin per la comunicazione seriale di tipo TTL a 5V. I pin N.0 e N.1 sono collegati direttamente ai corrispondenti pin del processore 32U4 Piedino 21 TXD1 e 20 RXD1.



TWI: 2 (SDA) e 3 (SCL). La scheda supporta il protocollo hardware TWI detto anche I2C formato da due linee seriali di comunicazione:

- **SDA** (**S**erial **DA**ta line) per i dati;
- **SCL** (**S**erial **C**lock **L**ine) per il clock.



Per utilizzarla occorre far ricorso alla **Liberia Wire**.

Interrupt esterni: 0 (RX), 1 (TX), 2 e 3 Questi pin possono essere configurati per attivare un interrupt su un valore basso, un fronte di salita o di discesa, o una variazione di valore. Vedere la funzione **attachInterrupt ()** per i dettagli.

Pin PWM, i pin 3, 5, 6, 9, 10, 11 e 13 sono utilizzabili come uscite **PWM** (acronimo di **P**ulse **W**idth **M**odulation) a 8 bit utilizzando la funzione **analogWrite()**.

Ingressi analogici: A0-A5, A6 – A11 (su pin digitali 4, 6, 8, 9, 10, e 12) Arduino Micro può essere configurato per avere sino a 12 ingressi analogici, i pin da **A0 a A5** sono direttamente etichettati sui pin stessi, mentre per gli altri, da **A6 a A11**, sono disponibili accedendo tramite codice di programmazione utilizzando le costanti da A6 ad A11 poiché sono condivise rispettivamente sui pin digitali 4, 6, 8, 9, 10, e 12.

Tutto ciò può anche essere utilizzato come I /O digitale.

Ogni ingresso analogico fornisce 10 bit di risoluzione (cioè 1024 valori differenti).

Per default la misura ingressi analogici è per valori di tensione compresi tra 0 e +5 volt, anche se è possibile cambiare la tensione massima utilizzando il pin **AREF** e la funzione **analogReference ()**.

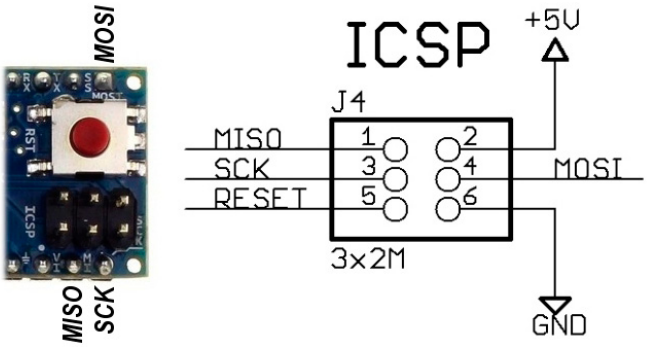
Oltre ai pin sin qui citati, sono presenti un paio di altri pin con funzioni speciali, questi sono:

AREF: Tensione di riferimento per gli ingressi analogici. Utilizzato con il comando **analogReference ()**.

Reset: Portare questa linea LOW per resettare il microcontrollore. Tipicamente è utilizzata per aggiungere un pulsante di reset, quando quello presente sulla scheda non è raggiungibile.

CONNETTORE ICSP

A lato del pulsante di reset, è presente il connettore **J4** denominato **ICSP** (acronimo di **I**n **C**ircuit **S**erial **P**rogramming) per l'eventuale programmazione senza rimozione dal circuito del processore.



I pin del connettore sono inoltre duplicati sulla scheda e sono etichettati **MISO**, **MOSI** e **SCK**.

LED DI SEGNALAZIONE

Sulla scheda sono presenti quattro led:

– **due di colore giallo (RX1 eTX1)** che segnalano il traffico di dati sulla linea seriale, il led RX1 è connesso al pin RX_LED / SS, che indica l'attività

Pin J4	Pin del Arduino Micro	Funzione
1	MISO	MISO (Master Input Salve Output)
2	+5V	+5V
3	SCK	SCK (Serial Clock Signal)
4	MOSI	MOSI (Master Output Slave Input)
5	RESET	RESET
6	GND	GND

di trasmissione durante la comunicazione USB, ma può essere anche utilizzato come slave.

– **uno di colore verde (L3)** collegato al pin digitale 13. Quando il pin è HIGH, il LED è acceso, quando il pin è LOW, esso è spento.

– **uno di colore blu (ON1)**, situato nella parte inferiore della scheda, segnala la presenza di alimentazione

Tutti i led sono di tipo SMD è hanno una resistenza di limitazione dal valore di 1kΩ.

DIFFERENZE TRA ARDUINO MICRO E ARDUINO UNO

Terminata la descrizione di come funziona la scheda, passiamo ora ad analizzare alcune differenze che esistono tra una normale scheda **Arduino UNO** e una scheda **Arduino MICRO**.

In generale, si programma e si utilizza la scheda Arduino Micro come si farebbe con altre schede Arduino. Vi sono, tuttavia, alcune differenze importanti.

Come abbiamo visto, la scheda Arduino Micro ha un solo processore sia per l'esecuzione del programma che per la comunicazione USB con il computer.

Arduino Uno e altre piattaforme utilizzano invece il classico FT232 o un Atmel dedicato, come il modello **ATMEGA8U2** su **Arduino UNO R3**, per questa funzione, il che significa che la connessione USB al computer rimane connessa indipendentemente dallo stato del microcontrollore principale.

Combinando queste due funzioni in un unico processore, la porta seriale è finalmente libera per il programma utente in qualsiasi momento, lo stack USB prevede anche la parte HID che permette quindi ad Arduino Micro di emulare un mouse o una tastiera.

Occorre però segnalare che qualche volta, vi potrebbero essere dei problemi di restart quando si apre la connessione seriale, per cui si potrebbe perdere la connessione con la scheda quando si avvia il programma.

RINUMERAZIONE DELLA SERIALE AL RESET

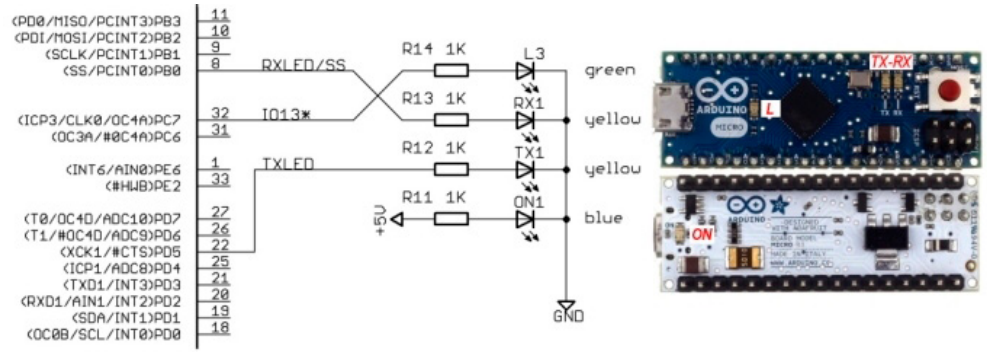
Dal momento che la scheda non ha un chip dedicato per gestire la comunicazione seriale, significa che la porta seriale è virtuale, è solamente una routine software, sia sul sistema operativo, che sulla scheda stessa.

Proprio come il computer crea un'istanza del driver della porta seriale quando si collega qualsiasi Arduino, la scheda Arduino Micro creerà un'istanza seriale ogni volta che esegue il suo bootloader.

Questo significa che ogni volta che si reimposta la scheda, la connessione seriale USB sarà interrotta e in seguito ristabilita.

La scheda scomparirà momentaneamente dall'elenco delle porte seriali e la lista si rinumererà.

Qualsiasi programma che ha una connessione seriale aperta con la scheda perderà la sua connessione.



EMULAZIONE DI TASTIERA E MOUSE

Un vantaggio di utilizzare un singolo chip per l'esecuzione dei programmi e il collegamento USB è aumentata dalla flessibilità nella comunicazione con il computer.

Mentre la scheda appare come una porta seriale virtuale con il sistema operativo (chiamato anche **CDC**) per la programmazione e la comunicazione (come con Arduino Uno), può anche comportarsi come un (**HID** acronimo di **H**uman **I**nterface **D**evice) come una tastiera o come un mouse.

SEPARAZIONE DELLA COMUNICAZIONE SERIALE DALLA PORTA USB

Sulla scheda Arduino Micro la comunicazione con il PC è realizzata con un driver seriale virtuale tramite la porta USB, non vi è collegamento ai pin fisici **0** e **1** come è sulla Arduino Uno.

Per utilizzare la porta seriale hardware UART TTL (5V) (**pin 0 e 1, RX e TX**), occorre utilizzare **Serial1**.

Il software Arduino include un monitor seriale che consente di inviare e ricevere dati dalla scheda Arduino.

I LED RX e TX sulla scheda lampeggeranno quando vi sarà traffico attraverso la connessione USB al computer (ma non per la comunicazione seriale sui pin 0 e 1).

La libreria **SoftwareSerial** consente la comunicazione seriale su altri pin digitali di Arduino Micro.

Come abbiamo visto l'ATMEGA32U4 oltre alla comunicazione seriale, supporta sia la comunicazione **I2C** (TWI) sia la **SPI**.

INSTALLAZIONE DEL DRIVER

Per installare il driver occorre collegare la scheda Arduino Micro al computer, per fare questo è necessario un cavo USB dotato di un connettore di tipo **Micro-B**.

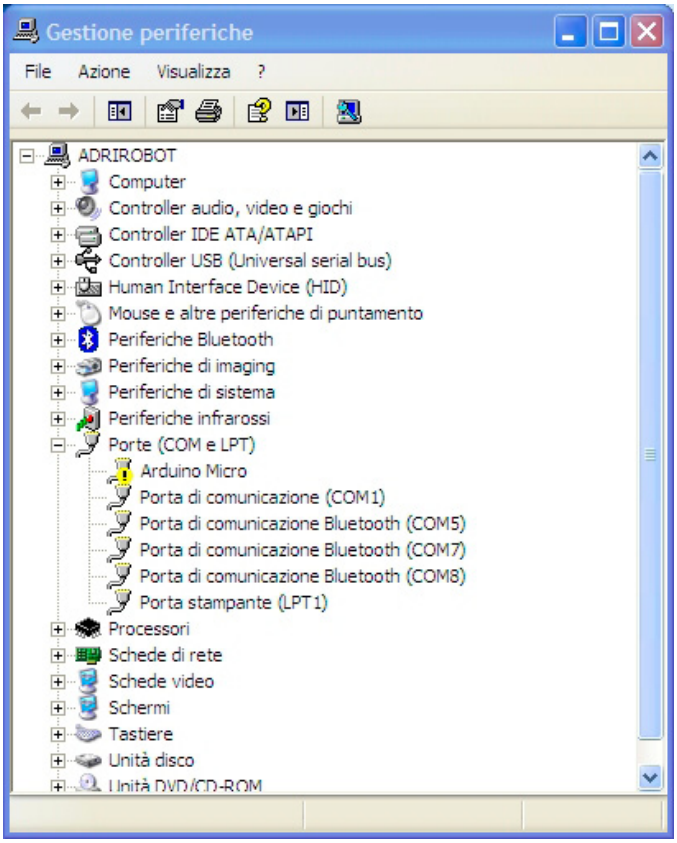
Come abbiamo visto, il cavo USB permette sia la programmazione sia l'alimentazione della scheda.

Le seguenti istruzioni si riferiscono alla versione per WINDOWS, sia per la versione XP che le successive, con piccole differenze nelle finestre di dialogo.

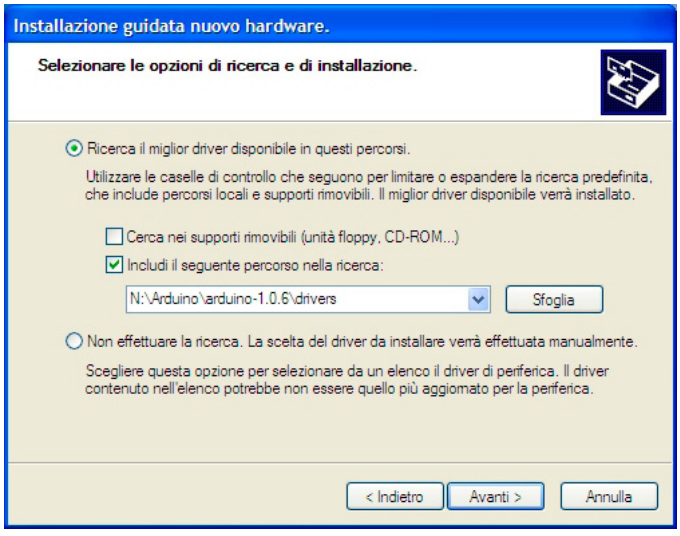
Una volta collegata la scheda, occorre attendere che Windows avvii il processo d'installazione del driver.

Se l'installazione non si avvia automaticamente, individuate tramite la Gestione periferiche di Windows (Start> Pannello di controllo> Hardware) nell'elenco **Arduino Micro**, che sarà evidenziato da un punto esclamativo giallo.

Fare clic con il tasto destro e scegliere **Aggiorna driver**.

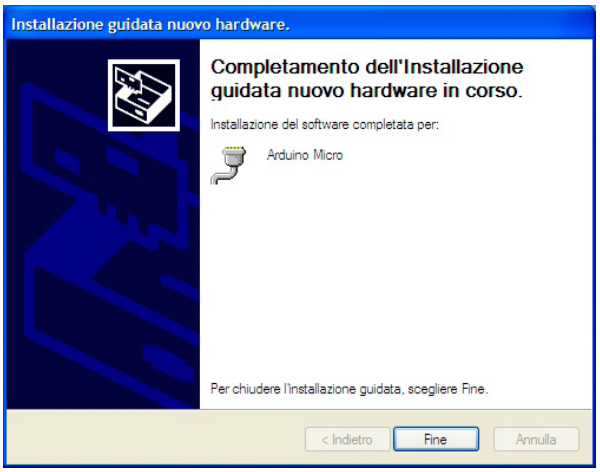


Nella schermata successiva, scegliere “Cerca il software del driver” e fare clic su Avanti.
Fare clic sul pulsante Sfoglia. Un’altra finestra appare: passare alla cartella con il software Arduino che si è appena scaricata.
Selezionare la cartella in cui sono i driver e fare clic su OK, quindi fare clic su Avanti.

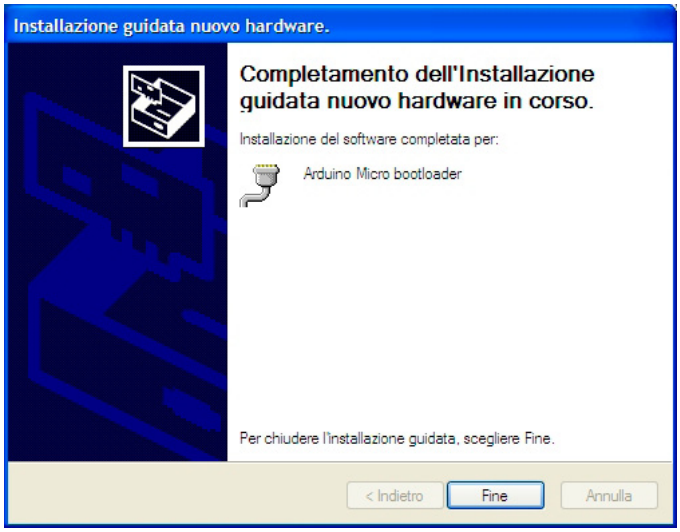


Si riceverà una notifica che la scheda non ha superato il test di Windows Logo. Fare clic sul pulsante Continua.

Dopo qualche istante, una finestra vi segnalerà che la procedura guidata per ‘installazione del software per Arduino Micro è stata completata. Premere il pulsante Chiudi.

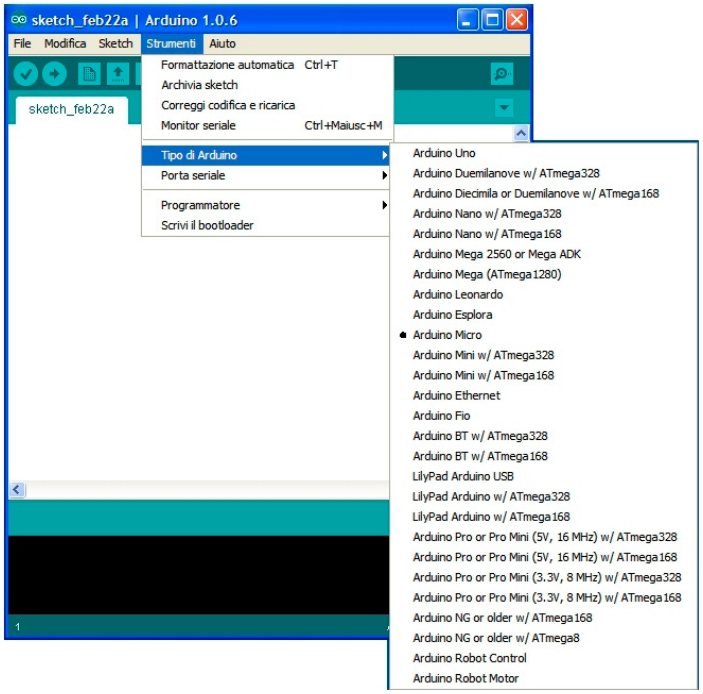


Al primo caricamento di un programma sarà richiesto d’installare il driver del Bootloader.
La procedura è simile a quella appena vista, il driver è posto nella stessa cartella impostata in precedenza.



PROGRAMMAZIONE

Arduino Micro può essere programmata con il **software IDE Arduino, ora alla versione 1.0.6**, che dovrebbe essere ben conosciuto dai lettori.
Per utilizzare la scheda occorrerà semplicemente selezionare "**Arduino Micro**" dal menu Strumenti> Tipo di Arduino.



Occorrerà inoltre indicare a quale porta seriale è collegato, selezionandola dal menu Strumenti> Porta seriale.
Come abbiamo visto l'ATMEGA32U4 su Arduino Micro è programmato con un bootloader che permette di caricare il nuovo codice senza l'utilizzo di un programmatore hardware esterno.
La comunicazione avviene utilizzando il protocollo AVR109.
È possibile comunque bypassare il bootloader e programmare il microcontrollore attraverso il connettore ICSP utilizzando eventualmente un altro Arduino come programmatore.

TEST DELLA SCHEDA

Per testare la scheda, non caricheremo il solito programma blink, ma qualcosa di più complesso: visualizzeremo su un **display TFT** i dati letti da un **sensore di BMP180** che fornisce i valori di: pressione, temperatura e altitudine sul livello del mare.
Perciò oltre alla scheda Arduino Micro sono necessari i seguenti componenti :

- Una breadboard;
- La scheda Arduino Micro;
- Display LCD TFT Arduino;
- Sensore di pressione BP180;
- Alimentatore per breadboard;
- Cavi colorati maschio-mascio per realizzare i collegamenti;
- Una batteria 9V per l'alimentazione.

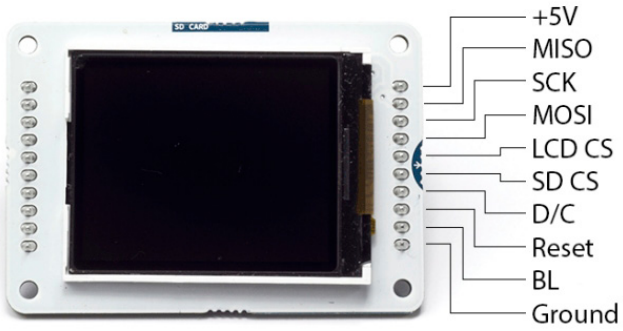
Vediamo le caratteristiche dei principali componenti utilizzati.

DISPLAY LCD TFT ARDUINO

Il display qui utilizzato è lo stesso presentato nell'articolo **Scopriamo la nuova scheda Arduino Esplora**.
Si tratta di un **display LCD tipo TFT** (Thin Film Transistor), retroilluminato che misura 1,77″ pari a circa 45 mm di diagonale, con risoluzione di 160 x 128 pixel.
Il modulo misura 40x44mm circa e nella parte posteriore è presente uno slot per schede mi-

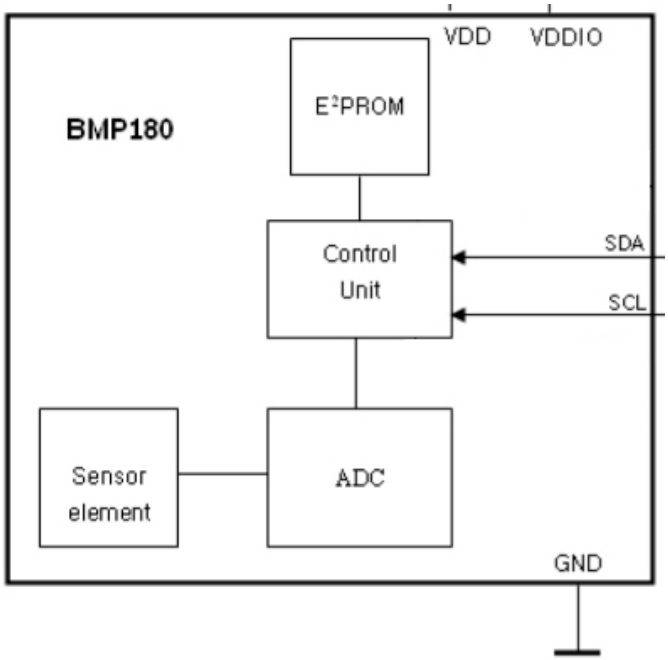
cro-SD, che tra le altre cose, permette di memorizzare le immagini bitmap da visualizzare sullo schermo.

Per la comunicazione, il display utilizza un **protocollo SPI**, per facilitare la gestione del display. E' disponibile **un'apposita libreria** che è inclusa in quelle disponibili per l'IDE 1.0.5 e successive.

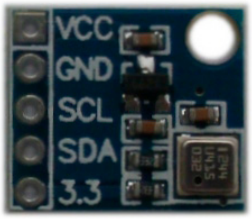


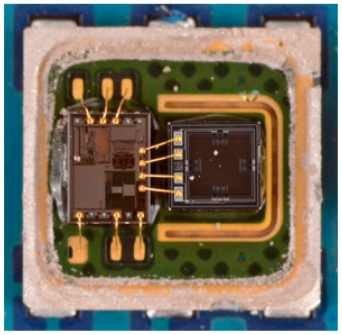
SENSORE DI PRESSIONE BMP180

Il sensore utilizzato è il **BMP180 prodotto dalla BOSCH**, permette di misurare pressioni da 300 a 1100hPa (da -500m a +9000m relativi al livello del mare), la tensione di alimentazione è compresa tra 1.8-3,6 (VDD) e 1.62-3.6V (VDDIO), il consumo è di soli 5uA con una lettura al secondo.



Il sensore è contenuto in un package tipo LGA con una dimensione di 3,6×3,8 mm.





Oltre al valore di pressione, il sensore fornisce anche la misurazione della temperatura; i dati sono trasmessi tramite un'interfaccia I2C, con un indirizzo fisso 0x77.

La calibrazione è fatta durante la produzione e i valori sono salvati nella memoria del sensore.

Per facilitare l'uso del sensore, questo si trova di solito già montato su una piccola basetta, dove oltre al sensore, trovano posto i pochi altri componenti necessari al suo funzionamento.

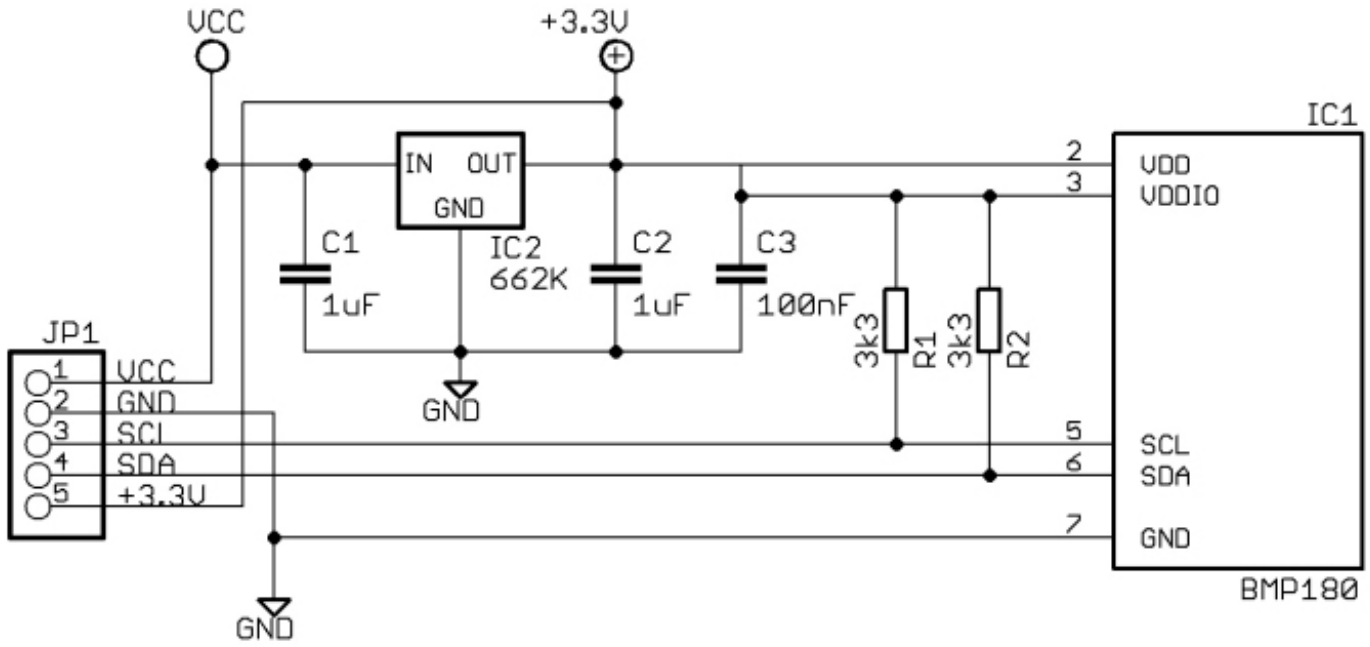
Il modulo utilizzato dispone anche di un piccolo regolatore di tensione **tipo 662K** che fornisce in uscita una tensione di 3,3 V, che permette di alimentare il modulo anche con una tensione di 5V.

ALIMENTATORE PER BREADBOARD

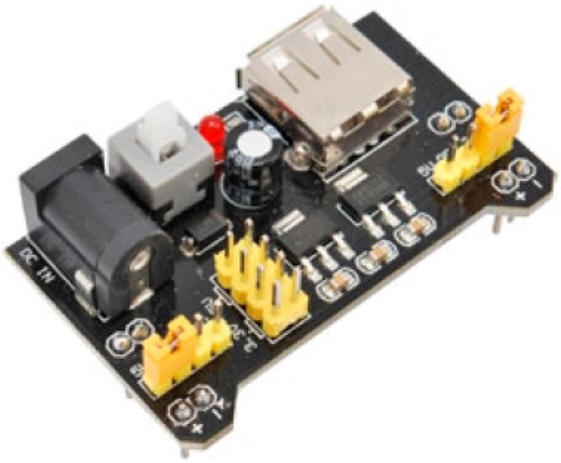
Per alimentare i vari componenti utilizzati, si è utilizzato un'alimentatore compatto (dimensioni: 55 x 35mm) studiato appositamente per essere impiegato con le breadboard.

Al modulo è possibile applicare una tensione compresa tra 6,5 e 12 Vdc, per ottenere in uscita due tensioni distinte a 3,3 V e 5 V con una corrente massima di circa 700 mA.

Ha un interruttore on/off, uscita USB 5V (non utilizzata nel nostro caso), LED di alimentazione e jumper di selezione tensione. Potrete trovarlo in rete cercando "**YwRobot MB-V2**"



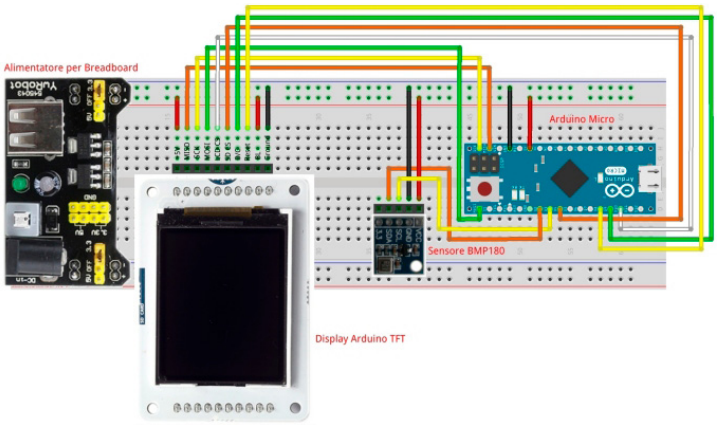
Schema del modulo utilizzato nel test



Pin Arduino Micro	Pin Display TFT	PIN Sensore BMP180
+5V	+5V, BL	+5V
GND	GND	GND
MISO	MISO	
SCK	SCK	
MOSI	MOSI	
SDA		SDA
SCL		SCL
D4	SD CS	
D10	LCD CS	
D9	D/C	
D8	Reset	

SCHEMA DI COLLEGAMENTO

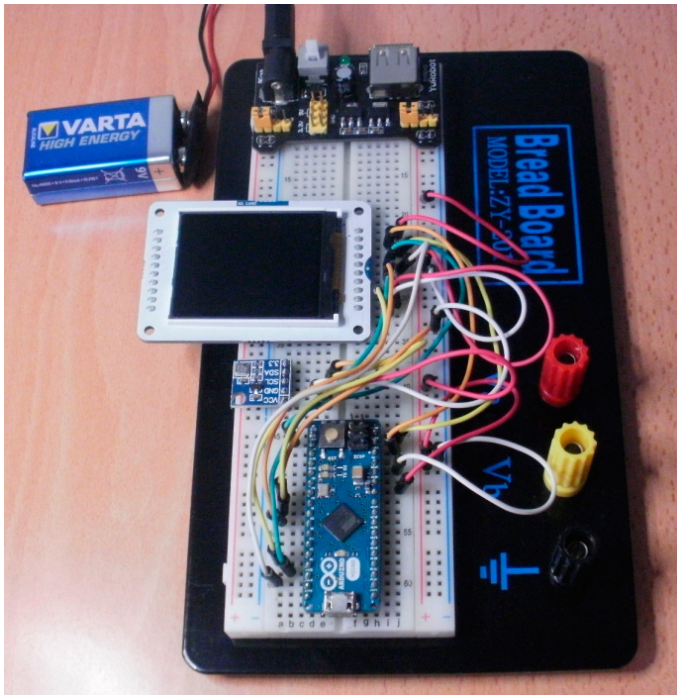
Nell'immagine sotto riportata sono visibili i collegamenti da realizzare per l'effettuare il test.



LIBRERIA PER UTILIZZO SENSORE BMP180

Per facilitare l'uso del sensore è disponibile un'apposita libreria (**bmp180**), questa funziona in unione alla **libreria Wire.h** già presente come libreria standard dell'IDE di Arduino che permette l'utilizzo del protocollo I2C.

In rete ve ne sono molte versioni, quella utilizza-



ta per questo test possiede vari comandi:

- Initialize();
- GetUncompensatedTemperature();
- GetUncompensatedPressure();
- Compensate Temperature();
- CompensatePressure();
- SoftReset();
- GetTemperature();
- GetPressure();
- GetAltitude();
- SetResolution();
- IsConnected()
- EnsureConnected()
- GetErrorText().

Nel programma di test saranno utilizzati solamente i seguenti:

Initialize() – Inizializza il sensore

IsConnected() – verifica che il sensore sia effettivamente connesso.

EnsureConnected() – verifica che sia possibile connettersi con il sensore.

SoftReset() – garantisce che quando si è collegato il sensore vi sia un avvio pulito.

GetTemperature() – acquisisce dal sensore il dato della temperatura.

GetPressure() – acquisisce dal sensore il dato della pressione in Pascal.

GetAltitude() – acquisisce dal sensore il dato dell'altitudine in metri, il risultato ottenuto deve essere corretto per la pressione relativa al punto in cui avviene la misurazione, nel programma deve essere inserito il valore locale nella variabile seaLevelPressure, che di default è pari alla pressione a livello del mare pari a 101325 Pa;

Nota: il dato della pressione è fornito in Pascal (Pa) nel Sistema internazionale pari a 1 newton su metro quadrato (1 N/m²). In ambito meteorologico, la pressione atmosferica si misura in centinaia di Pascal (o ettopascal, abbreviato con hPa). Si ha: 1 013,25 millibar = 101 325 Pa = 1 013,25 hPa

Per utilizzare la libreria occorrerà scaricarla e salvarla all'interno della cartella Library dell'IDE di Arduino.

Successivamente per utilizzarla all'interno del programma si dovranno aggiungere le seguenti righe:

```
#include <Wire.h>
#include <BMP180.h>
```

PROGRAMMA DI TEST

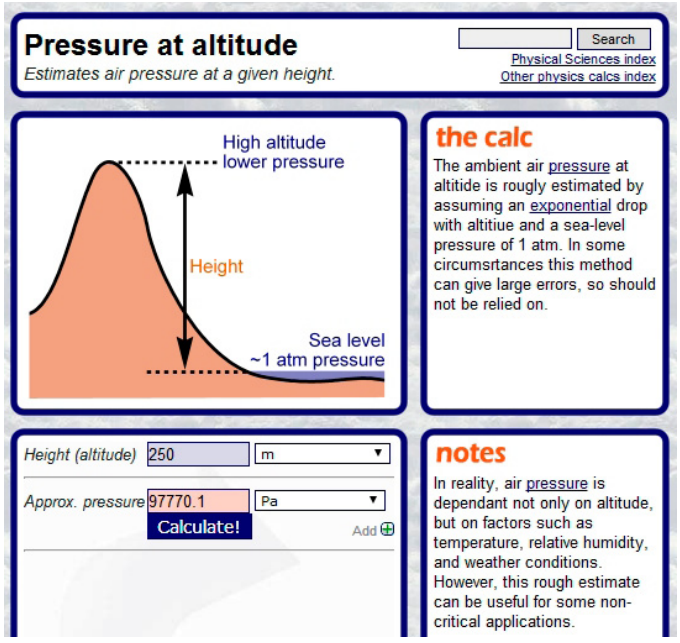
Il programma di test (**Sensore_BMP180**) mostrerà sul display TFT, dopo una pagina iniziale di presentazione, i valori letti dal sensore BMP180.

Per il funzionamento del programma oltre alle librerie **Wire.h** e **BMP180.h** citate prima per la gestione del sensore occorre utilizzare anche altre due librerie che sono già incluse in quelle

standard dell'IDE versione 1.0.6, queste sono **TFT.h** e **SPI.h** che occorrono per la gestione del display LCD.

Il listato è sufficientemente commentato per comprenderne il funzionamento.

Per ottenere i valori corretti dell'altitudine occorre inserire il valore relativo alla propria posizione nella variabile **seaLevelPressure**, che di default contiene il valore 101325 (espresso i Pa), per determinare il valore si può utilizzare l'utility trovata in rete.



Il risultato del calcolo è solamente una stima in cui si assume un calo esponenziale della pressione considerando una pressione a livello del mare di 1 atm (101325 Pa).

Questo valore potrebbe non essere corretto in quanto la pressione dell'aria dipende non solo dell'altitudine, ma anche da altri fattori quali temperatura, umidità relativa, e le condizioni atmosferiche. Tuttavia, questa stima può essere utile per applicazioni non critiche.

Nota: Per i calcoli impostare il dato in uscita in Pa.

```
#include <TFT.h>  // Arduino LCD library
#include <SPI.h>
// gestione sensore BMP180
#include <Wire.h>
#include <BMP180.h>

// Definizione dei pin utilizzati
// per il display TFT
#define cs   10
#define dc   9
#define rst  8

#define indicatorLed  13
// array per contenere i dati di pressione
char PressurePrintout[5];
String Pressure;
// array per contenere i dati di temperatura
char TemperaturePrintout[5];
String Temperature;
// array per contenere i dati di altitudine
char AltitudePrintout[5];
String Altitude;

// Crea un'istanza per il senore BMP180.
BMP180 barometer;
// Inserire la pressione a livello del mare della
// vostra posizione a Pascal.
float seaLevelPressure = 101325;

// create an instance of the library
TFT TFTscreen = TFT(cs, dc, rst);

void setup() {
  // Scrittura pagina di presentazione
  TFTscreen.begin();
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(2);
  TFTscreen.stroke(255,255,0);
  TFTscreen.text(" Test sensore",0,0);
  TFTscreen.text("   BMP180",0,20);
```

```
  TFTscreen.text("Arduino Micro",0,40);
  TFTscreen.stroke(0,255,255);
  TFTscreen.text("  adrirobot",0,60);

  // inizalizzazione della libreria I2C per la
  // comunicazione con il sensore BMP180.
  Wire.begin();
  // Impostazione LED indicatore.
  pinMode(indicatorLed, OUTPUT);
  // Creiamo un'istanza del nostro sensore BMP180.
  barometer = BMP180();
  // Si verifica che ci si può collegare al sensore.
  if(barometer.EnsureConnected())
  {
   TFTscreen.setTextSize(1);
   TFTscreen.stroke(0,255,0);
   TFTscreen.text("BMP180 collegato",30,100);
   // Accensione led se sensore collegato
   digitalWrite(indicatorLed, HIGH);

   // Dopo il, collegamento, il sensore viene
   //resettato per garantire un avvio pulito.
   barometer.SoftReset();
   // Ora inizializziamo il sensore.
   barometer.Initialize();
  }
  else
  {
   TFTscreen.setTextSize(1);

   TFTscreen.stroke(255,0,0);
   TFTscreen.text("BMP180 non collegato",20,100);
   // Spegnimento led se sensore non collegato
   digitalWrite(indicatorLed, LOW);
  }
  // Tempo di visualizzazione schermata iniziale
  delay(2000);
  TFTscreen.background(0, 0, 0);
}

void loop() {
  if(barometer.IsConnected)
  {
   //Lettura e visualizzazione pressione in Pascal.
   TFTscreen.stroke(255,0,0);
   TFTscreen.setTextSize(2);
   TFTscreen.text("Pressione:",0,0);
   TFTscreen.setTextSize(3);
   Pressure = String (barometer.GetPressure());
   Pressure.toCharArray(PressurePrintout, 6);
   TFTscreen.text(PressurePrintout, 0, 18);
   TFTscreen.text("Pa",115,18);

   // Lettura e visualizzazione temperatura in gradi
Celsius.
   TFTscreen.stroke(0,255,0);
   TFTscreen.setTextSize(2);
   TFTscreen.text("Temperatura:",0,45);
   TFTscreen.setTextSize(3);
   TFTscreen.stroke(0,255,0);
   Temperature = String (barometer.GetTemperatu-
re());
   Temperature.toCharArray(TemperaturePrinto
ut, 6);
   TFTscreen.text(TemperaturePrintout, 0, 63);
   TFTscreen.circle(110, 65, 2);
   TFTscreen.text("C",115,63);

   //Lettura e visualizzazione quota attuale in me-
tri.
   TFTscreen.stroke(255,255,0);
   TFTscreen.setTextSize(2);
   TFTscreen.text("Altitudine:",0,90);
   TFTscreen.setTextSize(3);
   Altitude = String (barometer.
GetAltitude(seaLevelPressure));
   Altitude.toCharArray(AltitudePrintout, 6);
   TFTscreen.text(AltitudePrintout, 0, 108);
   TFTscreen.text("m",115,108);

   // Attesa tra le misure
   delay(1000);

   // Cancellazione dati per nuova misura
   TFTscreen.stroke(0,0,0);
   TFTscreen.text(PressurePrintout, 0, 18);
   TFTscreen.text(TemperaturePrintout, 0, 63);
   TFTscreen.text(AltitudePrintout, 0, 108);
  }
}
```

FILMATO ILLUSTRATIVO

E' possibile vedere l'esecuzione del programma visionando il filmato:

Arduino Micro – Lettura sensore di pressione BMP180

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

CONCLUSIONI

Siamo al termine di quest'articolo di presentazione di questo piccolo/grande Arduino, abbiamo anche mostrato l'utilizzo di un sensore con cui il lettore potrebbe creare, con l'utilizzo di altri sensori quali sensore di umidità e un modulo clock, **una piccola stazione meteo**, oppure realizzare magari un altimetro come visibile in **questo filmato**.

Non resta che liberare la propria immaginazione.

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-micro-e-bmp180-per-realizzare-una-weather-station

Progetto Giardiniere: Gestire una serra domestica con Arduino

di Gaspare Santaera

IL GIARDINIERE: L'IDEA

Il progetto Giardiniere nasce come espansione di Thumbelina, un progetto che proponeva la realizzazione di un software capace di scrivere in modo automatico uno sketch per Arduino, per la gestione combinata di sensori e trasduttori. Il software è stato sviluppato e testato su un piccolo modello di serra arrivando a conquistare il secondo premio, nella challenge indetta dal CNA Italia all'interno dell'Hackathon, tenutasi in occasione dell'Internet Festival a Pisa lo scorso Ottobre.

Partendo da Thumbelina, quindi il Giardiniere, che rispetto alla madre ha un nome meno fantasioso ma più esplicativo, è un progetto per la creazione e la gestione di una serra verticale da balcone. Nello specifico il sistema è composto da quattro vasche di terra e si occupa del monitoraggio dei valori di umidità del terreno, della quantità di luce e della temperatura all'interno della serra, provvedendo ove sia necessario all'irrigazione, all'illuminazione ed alla ventilazione di quest'ultima.

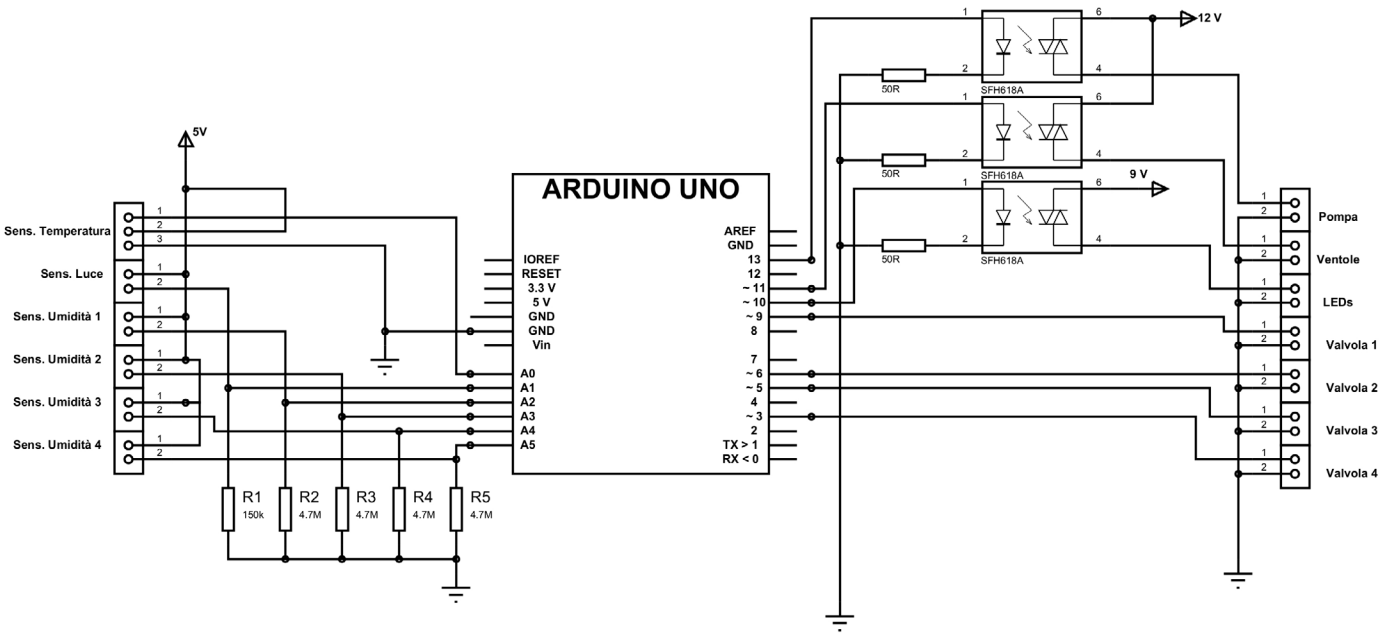
In questo articolo verrà presentato lo schema elettrico che utilizza un Arduino Uno, il firmware scritto per quest'ultimo, ed infine verrà data una breve descrizione delle valvole utilizzate per l'irrigazione che sono state progettate ad-hoc per questa applicazione e realizzate con l'ausilio di una stampante 3D. Tutti i CAD delle valvole nonché i CAD di tutti gli altri pezzi, stampati in 3D, della struttura sono allegati nel file Stl.

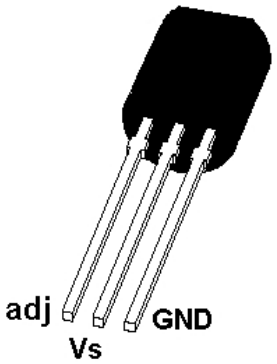
SCHEMA ELETTRICO

Nella figura viene riportato lo schema elettrico dove per semplicità l'Arduino Uno viene rappresentato come un blocco sul quale sono stati riportati i pin presenti sulla scheda. Nella parte sinistra dello schema vengono rappresentati gli ingressi, ovvero i sensori, mentre nella parte a destra vengono rappresentate le uscite, ovvero gli attuatori.

Per quanto riguarda gli ingressi, sono stati usati sei diversi sensori a loro volta collegati a sei ingressi analogici di Arduino; è presente un sensore di temperatura, un sensore di luminosità e quattro sensori di umidità, uno per ogni vasca o vaso.

Il sensore di temperatura usato è un classico LM335, un sensore analogico che restituisce sul suo piedino "adj" una tensione proporzionale alla temperatura misurata.





Simbolo del sensore d'Umidità

Fotocellula

Per quanto riguarda il sensore di luminosità è stata usata una banalissima fotocellula, questa si comporta come una resistenza di valore variabile in funzione della luce, nello specifico in assenza di luce avrà una resistenza relativamente bassa (10/15 Kohm), mentre in caso di luce aumenta la propria resistenza arrivando anche a superare i 150 KOhm. Usando nel progetto il sensore di luminosità come un semplice interruttore, atto ad accendere le luci artificiali quando quelle naturali scendono al di sotto di una certa luminosità, nel partitore di tensione insieme alla fotocellula viene usata una resistenza da 150 Kohm, ottenendo così un buon compromesso tra lo stato di luce e quello del buio.

Considerazioni leggermente diverse vengono fatte invece per i sensori di umidità, in questo caso la resistenza presente nel partitore di tensione è di circa 4.7 MOhm in quanto la resistenza elettrica del terreno è notevolmente superiore. Il sensore d'umidità, infatti, si basa sulla capacità del terreno di cambiare la propria resistenza al variare della propria umidità. Un terreno secco presenta una grossa resistenza elettrica, sperimentalmente è stata misurata una resistenza di 7/8 Mohm, che scende all'aumentare della percentuale d'acqua, sperimentalmente è stata misurata una resistenza di 4 Mohm. Quest'effetto è dovuto alla capacità dell'acqua di aumentare la conduttività del terreno, in quanto portatrice di sali minerali ricchi di cationi ed anioni, e al tempo stesso di compattare il terreno riducendo la distanza tra le molecole di quest'ultimo.

Sensore d'Umidità (Simbolo Circuitale – Modello Commerciale – Sensore Usato nel Progetto)

Nella figura vengono mostrati rispettivamente: uno schema del sensore; un modello commerciale; il sensore usato in questo progetto. Concettualmente i tre casi sono identici, come già detto si tratta di misurare la resistenza elettrica tra due conduttori ad una certa distanza, quindi qualunque soluzione che preveda dei conduttori fissati nel terreno è valida. Ovviamente i valori di resistenza misurati saranno funzione oltre che dell'umidità e del tipo di terreno usato anche della distanza tra i conduttori, sarà quindi buona norma eseguire alcune misure a vuoto sul terreno al fine di trovare il migliore valore per la resistenza da inserire nel partitore. Le misure di resistenza riportate nel precedente paragrafo si riferiscono al mio caso con il tipo di sensore mostrato in figura e una miscela di terriccio standard di quelli normalmente acquistabili al supermercato.

Nella parte destra dello schematico vengono mostrate le connessioni tra Arduino e vari attuatori, in questo progetto viene usata una pompa al fine di aumentare la portata tra i contenitori d'acqua e le vasche del terreno, quattro ventole (una per ogni vasca) per la ventilazione della serra, otto luci (due per ogni vasca) per l'illuminazione e otto valvole (due per ogni vasca) per l'irrigazione.

La pompa, le ventole e i led vengono controllati tramite tre optoisolatori SFH618A (usati anche qui) a loro volta pilotati da tre uscite digitali di Arduino, mentre le otto valvole (quattro gruppi da due) vengono pilotate tramite quattro uscite PWM.

In particolare per quanto riguarda la pompa è stato riutilizzato un vecchio modello smontato dal parabrezza di un'automobile ed alimentato a 12V, tale scelta, oltre ad essere abbastanza economica, aggiunta alla gravità (i contenitori d'acqua si trovano sulla parte superiore della struttura) garantisce una buona portata in tutte le vasche.

Pompa

Anche per le ventole si è optato per una soluzione di recupero, installando quattro ventole da 12V recuperate da alcuni alimentatori per PC

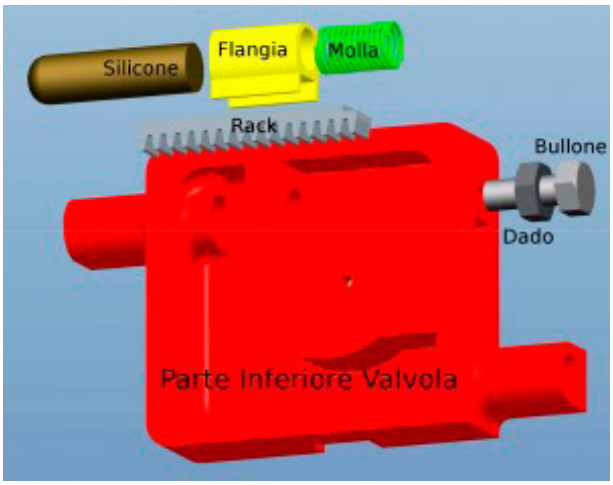
Ventola

L'illuminazione è affidata ad un gruppo di led,

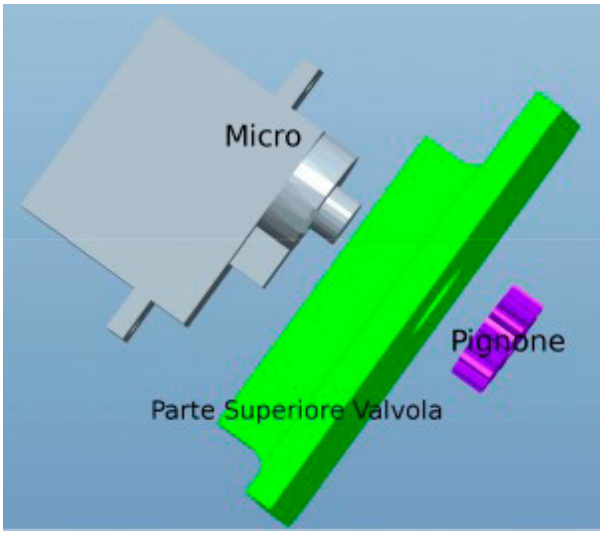
alimentati a 9 Volt, anche questi recuperati da alcune lampade da giardino normalmente acquistabili in qualche ipermercato o catena di ferramenta.

LED

Infine, discorso diverso va fatto per le valvole, innanzitutto per il tipo di controllo, in questo caso è necessario un segnale PWM e in secondo luogo per la provenienza di quest'ultime. Infatti a differenza della pompa, delle ventole e dei led le valvole usate nel progetto sono quasi completamente progettate e costruite in casa mediante una stampante 3D. Come già detto nel file Stl, trovate un archivio rar con tutti gli stl dei pezzi da stampare per montare la valvola oltre ai pezzi usati per il montaggio della struttura e la scatola contenente l'elettronica (qui un link interessante su una stampante 3D commerciale).



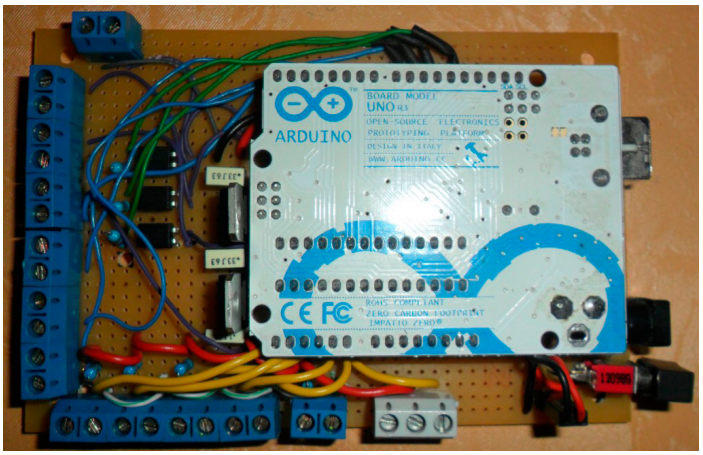
Esploso: Parte Inferiore Valvola



Esploso: Parte Superiore della Valvola

La struttura della valvola è molto semplice, le figure mostrano i CAD esplosi di quest'ultima, la parte inferiore è composta da una parte rigida contenente due raccordi per il tubo dell'acqua in ingresso e in uscita e una parete centrale atta a dividere la parte a contatto con l'acqua dalla parte a contatto con il servo (montato sulla parte superiore della valvola). All'interno della parte asciutta è presente una flangia alla quale è incollato a sinistra uno stantuffo di silicone che chiude la valvola e a destra una molla che serve ad aiutare il servo a spingere lo stantuffo per chiudere la valvola o in caso di servo in folle chiude automaticamente la valvola prevenendo eventuali perdite d'acqua. La molla è stata fissata al telaio mediante l'utilizzo di un sistema bullone-dado, la flangia e di conseguenza lo stantuffo, che apre o chiude la valvola, viene mossa mediante un sistema pignone-cremagliera dove quest'ultima è rigidamente attaccata alla flangia. Il pignone invece si trova sulla parte superiore della valvola ed è rigidamente attaccato al servo. Ricapitolando in ogni valvola un servo pilotato da Arduino spinge avanti ed indietro uno stantuffo aprendo e chiudendo opportunamente

la valvola. Il file Stl contiene tutti i file stampati in 3D per la realizzazione della struttura e delle valvole.



Scheda di Controllo

IL FIRMWARE

Il firmware di pari passo allo schema elettrico è anch'esso semplice, dopo una fase iniziale di sori e dei vari pin usati da sensori e trasduttori, il programma entra in ciclo while infinito. In ogni ciclo vengono dapprima letti tutti i valori dei sensori, successivamente vengono eseguiti i vari controlli sui valori letti. Il primo sensore ad essere controllato è quello di temperatura, se il valore letto è superiore ad una certa soglia in tick prestabilita vengono accese le ventole per la ventilazione della serra. In questo caso le ventole rimarranno accese per 60 secondi e uno speciale contatore verrà incrementato di 60.

Successivamente vengono via via controllati i quattro sensori di umidità, in questo caso se il valore letto supera il valore di soglia prestabilito, viene dapprima attivata la pompa, successivamente viene abilitato il servo presente sulla valvola e mandato a quest'ultimo un valore in modo da aprire la valvola e far scorrere l'acqua all'interno di questa. Tutto il processo dura 5 secondi al termine dei quali il contatore viene incrementato di 5.

Infine, viene controllato il valore letto dalla fotocellula, se il sistema rileva una sufficiente quantità di luce il valore del contatore viene confrontato con un valore di soglia di luce e nel caso in cui il valore del contatore è superiore a quest'ultimo le piante hanno avuto luce per più tempo di quanto prefissato e quindi sarebbe inutile accendere i led. Viceversa se il sistema rileva una bassa quantità di luce e dal controllo sul contatore del tempo si evince che le piante hanno avuto luce per un tempo inferiore alla soglia prestabilita, il sistema provvederà all'accensione dei led.

Dalla descrizione della logica di funzionamento del sensore di luce, si ben intuisce che il sistema è stato tarato affinché modificando due semplici numeri si riesca facilmente a stabilire quanta luce, tra naturale ed artificiale, dare alle piante ogni giorno, nonché il perché di una variabile che costantemente tiene il conto dei secondi.

Dopo aver controllato tutti i sensori, il programma si congela per 300 secondi, aggiornando sempre il contatore, prima di iniziare un nuovo ciclo.

Cliccando su Giardiniere è possibile scaricare il file .rar contenente il firmware.

CONCLUSIONI

In questo progetto si è visto come poter sviluppare in modo semplice, pratico ed economico una piccola serra domestica da tenere magari sul balcone. L'idea di fondo del progetto è semplice tanto quanto la sua realizzazione, tuttavia non essendo un esperto di botanica oltre alle caratteristiche sviluppate (irrigazione, illumina-

zione e ventilazione) al momento non ho molte altre idee su quali azioni o misure si possano prendere al fine di curare meglio le piante o di aumentarne la crescita. Per questo ultimo scopo mi rivolgo alla fantasia e all'esperienza degli altri utenti della community, buon divertimento!

Struttura

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-giardiniere-gestire-una-serra-domestica-con-arduino

Un telecomando TV per comandare un robot cingolato

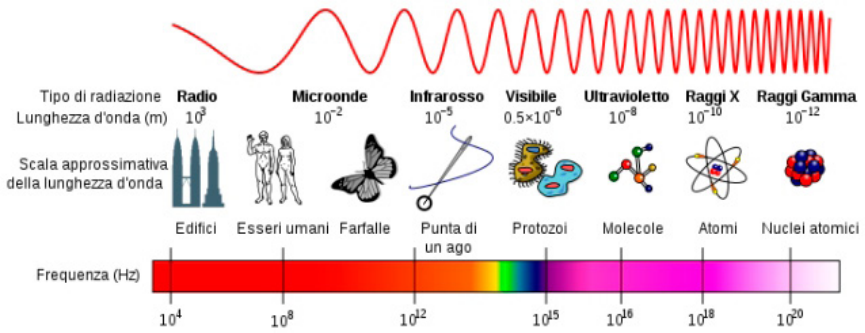
di adrirobot

LA LUCE INFRAROSSA

La luce infrarossa o radiazione infrarossa (IR) è una radiazione elettromagnetica con banda di frequenza dello spettro elettromagnetico inferiore a quella della luce visibile, con una lunghezza d'onda compresa tra 700 nm e 1 mm (banda infrarossa).

Il termine significa "sotto il rosso" (dal latino infra, "sotto"), perché il rosso è il colore visibile con la frequenza più bassa.

Per noi esseri umani, questa luce è invisibile perché la sua lunghezza d'onda è al di sotto dello spettro di sensibilità del nostro apparato di sensing.



Un metodo per "vedere" questa luce è quello di utilizzare una videocamera o fotocamera digitale (va bene anche quella di un telefonino).

Basta puntare il telecomando verso una macchina fotografica, premere un pulsante qualsiasi e sarà visibile la luce intermittente del LED.

Ci sono molte fonti di luce infrarossa, poiché tutto ciò che irradia calore, irradia anche luce infrarossa e questo potrebbe creare interferenze (rumore di fondo) tra il telecomando e il sensore.

Il sole è la fonte più brillante di tutte, ma ve ne sono altre come: lampadine, resistenze di riscaldamento, candele, impianto di riscaldamento centralizzato, anche il nostro corpo irradia luce infrarossa.

Perciò, occorre prendere alcune precauzioni per garantire che il nostro messaggio IR arrivi al ricevitore senza errori.

LA MODULAZIONE

Per eliminare il problema del rumore di fondo, è necessario utilizzare la modulazione.

Con la modulazione facciamo sì che la sorgente infrarossa emetta il segnale a una particolare frequenza.

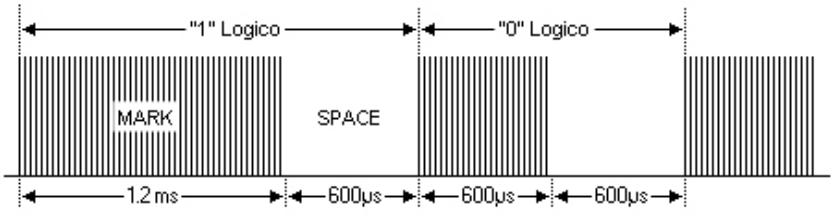
Basterà sintonizzare il ricevitore infrarosso su quella frequenza, in modo da ignorare tutto il resto.

Per analogia si può fare l'esempio del lampeggio delle luci gialle a lato dei cantieri che attirano l'attenzione anche in pieno giorno.

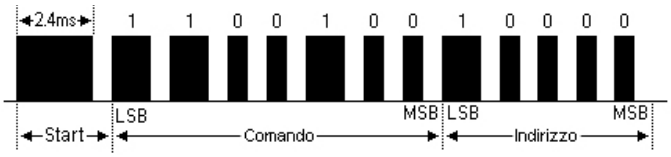
Nella comunicazione seriale di solito parla di «**MARK**» e «**SPACE**».

Il MARK è il segnale di default, che è lo stato di off nel caso del trasmettitore.

Nessuna luce è emessa durante lo stato di SPACE. Durante lo stato di MARK del segnale di luce IR è pulsata e si spegne a una particolare frequenza, di solito compresa tra 30 e i 60 kHz.



Sul lato ricevente un “SPACE” è rappresentato da un elevato livello di uscita del ricevitore. Un “MARK” è rappresentato automaticamente da un livello basso.
Si prega di notare che i “MARK” e gli “SPACE” non sono gli 1 e gli 0 che vogliamo trasmettere.
Il vero rapporto tra i MARK e gli SPACE e tra 1 e 0 dipende dal protocollo che viene utilizzato.

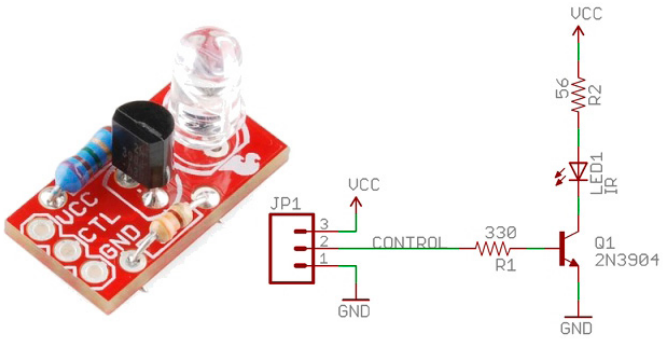


Nell’immagine sopra è riportato un tipico treno di impulsi del protocollo SIRC (SONY), con questo protocollo il **LSB** (**L**east **S**ignificant **B**it - bit meno significativo) viene trasmesso per primo.
Il primo impulso è ampio 2.4ms, seguito da uno spazio standard di 0.6ms.
Oltre a segnalare l’inizio di un messaggio SIRC questo segnale di Start viene utilizzato anche per regolare il guadagno del ricevitore IR.
Viene trasmesso, quindi, il comando di 7-bit, seguito dall’indirizzo del dispositivo di 5-bit.
In questo caso sono trasmessi l’Indirizzo 1 e il Comando 19.
I comandi sono ripetuti ogni 45ms (misurato da Start a Start) per tutto il tempo per cui il tasto sul telecomando viene tenuto premuto.

IL TRASMETTITORE

Il trasmettitore o telecomando è di solito portatile ed alimentato a batteria, deve consumare meno energia possibile, mentre deve fornire un segnale IR più forte possibile per ottenere una distanza di controllo accettabile.
Sul mercato si trovano chip che sono progettati per essere utilizzati come trasmettitori a infrarossi, di solito sono dedicati solo a uno dei molti protocolli che sono stati inventati.
Ultimamente sono però utilizzati dei microcontrollori visto il loro basso prezzo e soprattutto la loro versatilità di utilizzo.
Nel funzionamento è previsto che, se non si preme alcun tasto, siano mantenuti in modalità a basso assorbimento e solo quando si preme un tasto, il processore viene risvegliato per l’invio del segnale IR appropriato.
Per il clock del processore non sono di solito utilizzati i quarzi, in quanto molto fragili e con tendenza alla rottura in caso di cadute; in questo caso si utilizzano i risonatori ceramici, che sono leggermente meno precisi ma molto più resistenti agli urti.
Per il pilotaggio del led, la corrente che lo attraversa, può variare da 50 mA a oltre 1A.
Più è alta la corrente, maggiore sarà la distanza raggiunta dal segnale. Un buon compromesso può essere trovato tra i parametri dei LED, la durata della batteria e la distanza massima di controllo.
La corrente di alimentazione del LED può essere alta poiché gli impulsi di pilotaggio sono molto brevi, si consideri comunque che la dissipazione di potenza media del LED non deve superare il suo valore massimo.
È necessario verificare che la corrente massima di picco per il LED, dato che è riportato sulla scheda tecnica del led, non sia superata.
Un esempio di pilotaggio può essere un piccolo modulo fornito dalla **SparkFun**, il modulo denominato **Max Power IR LED Kit**; è formato da uno diodo led emittente luce infrarossa con una lunghezza d’onda di 950nm, in serie è presente la resistenza R2 da 56Ω che è calcolata considerando i parametri di tensione di alimentazione, caduta di tensione sul led e corrente massima che deve scorrere al suo interno.
L’accensione è garantita da un transistor NPN

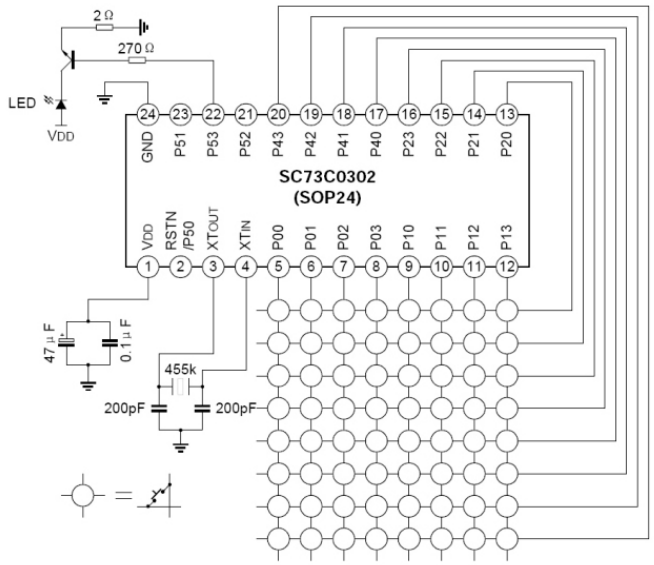
tipo **2N3904** sulla cui base è presente una resistenza R1 da 330 Ω.



Nella nostra applicazione, sarà utilizzato un telecomando fornito a suo tempo per comandare il robot Panettone della DeAgostini.

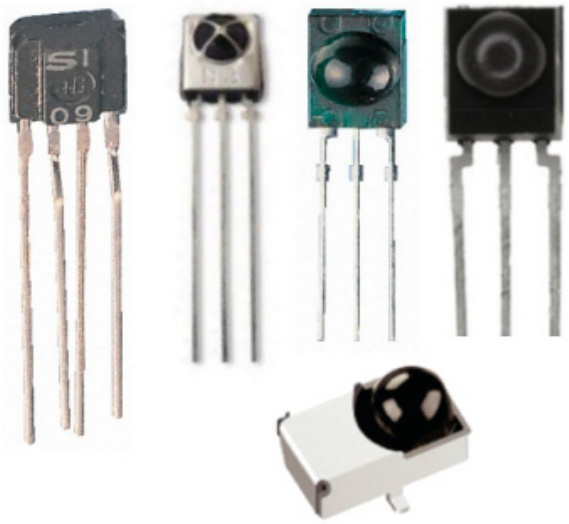
Cuore del circuito di comando è l'integrato **SC73C0302-008** di produzione cinese che codifica i segnali della tastiera con il protocollo Sony-compatibile e li invia tramite un led trasmittente ad infrarosso. L'alimentazione è fornita tramite 2 batterie 1,5V stilo.

Il telecomando presenta 27 tasti a cui sono associate varie serigrafie.

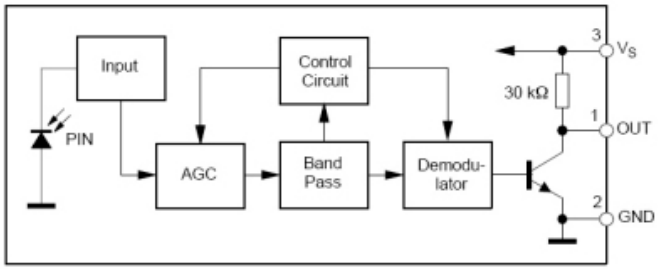


IL RICEVITORE

Esistono diversi modelli di ricevitore sul mercato, il più importante criterio di selezione è la frequenza di modulazione utilizzata.



Analizziamo prima di tutto il loro funzionamento: nella foto sotto potete vedere lo schema a blocchi del ricevitore TSOP18XX.



Il segnale IR ricevuto è prelevato dal diodo di rilevazione a infrarossi PIN, questo segnale è amplificato e limitato nel primo blocco di Input.

Nel blocco AGC il segnale è trattato in modo da ottenere un livello di impulso costante, indipendentemente dalla distanza del telecomando.

Il segnale AC è poi inviato al filtro Passa Banda (Band Pass) che è sintonizzato sulla frequenza di modulazione del telecomando.

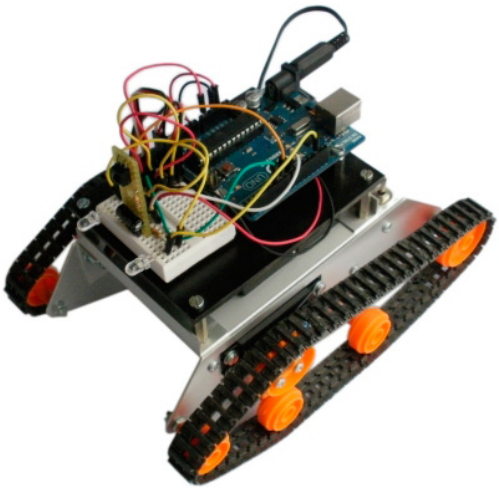
Queste frequenze possono andare, per esempio, nel modello TSOP18XX (da 30kHz a 40kHz).

Type	fo	Type	fo
TSOP1830	30 kHz	TSOP1833	33 kHz
TSOP1836	36 kHz	TSOP1837	36.7 kHz
TSOP1838	38 kHz	TSOP1840	40 kHz
TSOP1856	56 kHz		

Nel blocco demodulatore avviene la rilevazione della presenza della frequenza di modulazione e, se questa è presente, l'uscita sarà attivata e si avrà il segnale sul pin OUT.
Esistono diversi produttori di ricevitori IR tra i quali: Siemens, Vishay e Telefunken.

Esempio di applicazione

Come esempio di utilizzazione, si è pensato di realizzare un semplice robot che sarà poi comandato tramite i comandi impartiti da un telecomando.
Il modulo ricevente potrà essere montato su una semplice bread board.



Queste sono solo alcune delle caratteristiche del robot proposto:

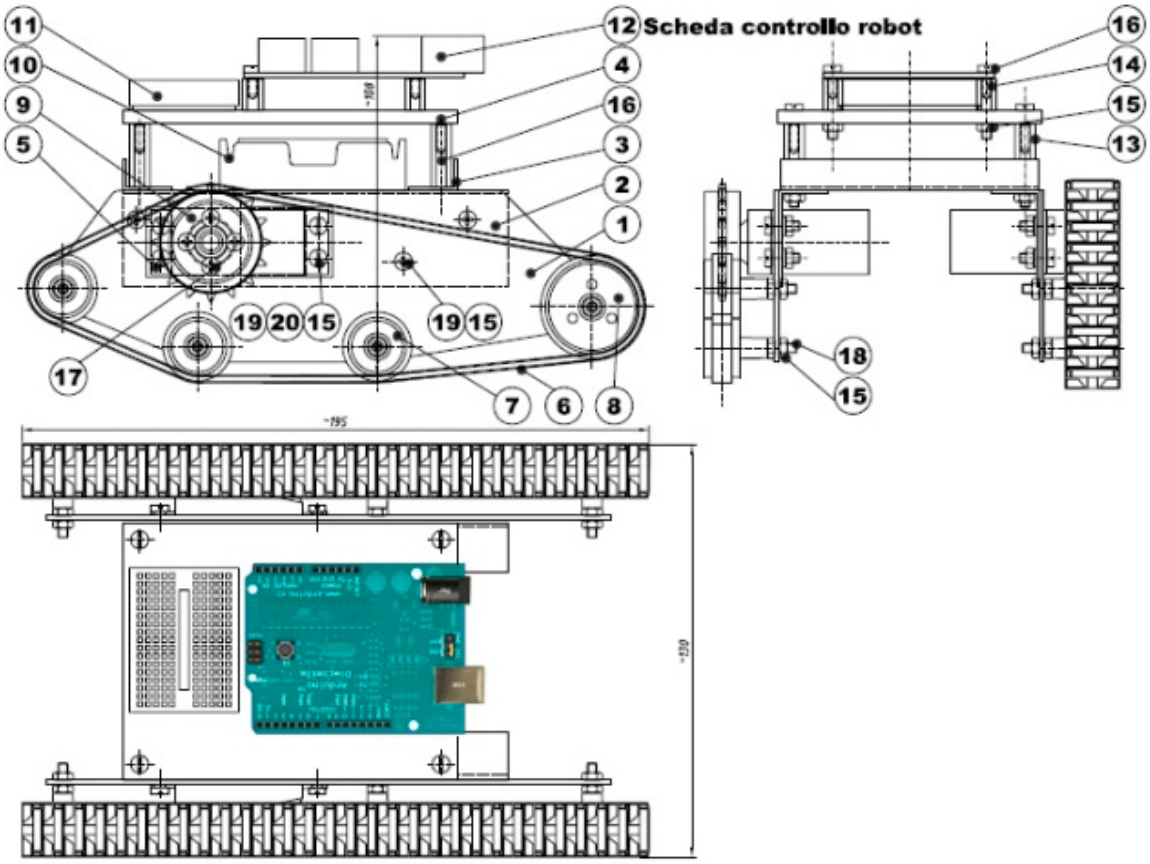
MONTAGGIO DEL ROBOT

Per la costruzione del robot inizieremo dalla lista delle parti necessarie, per la localizzazione sul robot fare riferimento al disegno sottostante.

[1] Piastre laterali - n° 2 pezzi;
[2] Profilati fissaggio servomotori - n° 2 pezzi;
[3] Traverse - n° 2 pezzi;
[4] Piastra supporto scheda controllo - n° 1 Pezzo;
[5] Servomotori a rotazione continua completi di squadrette – n° 2 pezzi;

Processore scheda di comando	Arduino UNO - ATmega328
Uscite	Comando servomotori Comando led illuminazione
Sensori	Sensore decodificatore segnali IR tipo TSOP 1836 (36 KHz)
Alimentazione	6V tramite 4 batterie 1,5V tipo AA
Motorizzazione	2 servomotori a rotazione continua
Cingoli	In gomma, prodotti dalla Tamiya
Misure	195x130x108 mm
Peso	465 g

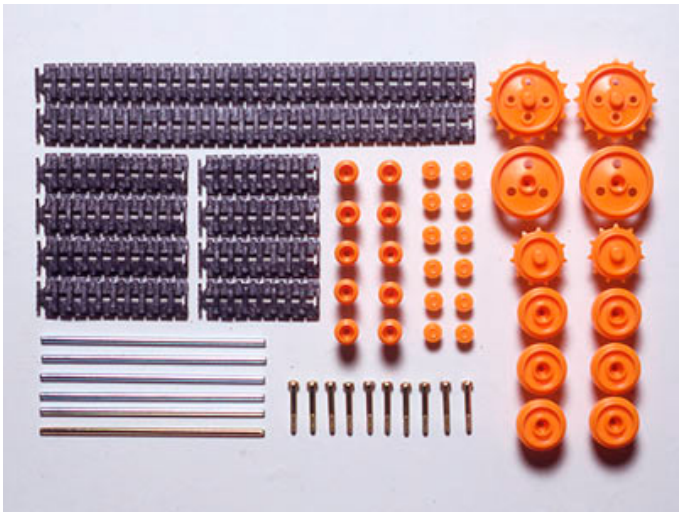
[6] Cingoli 66 maglie – n° 2 pezzi;
[7] Ruote medie folli Ø21,5 - n° 6 pezzi;
[8] Ruote grandi folli Ø32 – n° 2 pezzi;
[9] Pignoni grandi – n° 2 pezzi;
[10] Portabatteria per 4 batterie tipo AA - n° 1 pezzo;
[11] Mini breadboard 17x10 – n° 1 pezzo;
[12] Scheda di controllo utilizzata Arduino UNO;
[13] Distanziali esagonali M3 L= 20mm – n° 4 pezzi;
[14] Distanziali esagonali M3 L= 10mm – n° 4 pezzi;
[15] Dadi M3 – n°38 pezzi;
[16] Viti testa cilindrica con intaglio M3 L= 8 – n° 8 pezzi;
[17] Viti testa croce M3 L= 25 – n° 8 pezzi;
[18] Viti testa croce autofilettante M3 L= 8 – n° 8 pezzi;
[19] Viti testa cilindrica con intaglio M3 L= 10 – n° 14 pezzi;
[20] Rosette piane Ø3,2– - n° 8 pezzi;



[21] Rondelle antisvitamento Ø3,2– - n° 8 pezzi;
[22] Modulo sensore a infrarossi tipo TSOP1836;
[23] Mammut 2 poli;
[24] Interruttore deviatore a Levetta sub miniatura;
[25] Spina DC Dimensioni 2,1 x 5,5 x 9,5mm, terminali a saldare.

CINGOLI

I particolari utilizzati sono all'interno della confezione **Tamiya 70100 Track and Wheel Set**;



sono presenti vari componenti come: spezzoni di cingoli in gomma di varie lunghezze, alberi, ruote e pignoni

La lunghezza di cingoli, pignoni e ruote di dimensioni diverse, consentono flessibilità nella progettazione del proprio robot.

Per il nostro robot si sono utilizzati:

- tutti gli spezzoni per formare due cingoli (1 lungo+2 medi+2 piccoli) 66 maglie **[6]** ;
- n° 6 ruote piccoli folliØ21,5 **[7]** ;
- n° 2 ruote grandi folli Ø32 **[8]** ;
- n° 2 pignoni grandi **[9]**.

SERVOMOTORE

Per la motorizzazione del robot, si sono utilizzati due servomotori **[5]** di taglia normale, da questi è stato eliminato il blocco della rotazione.

Nel prototipo sono stati utilizzati **due servomotori Parallax**, ma in rete è possibile trovare vari riferimenti dove viene spiegato come effettuare la modifica su normali servomotori.

Alcune caratteristiche dei servo impiegati:

- dimensioni: 40.5 × 20.0 × 38,0 millimetri;
- peso: 43 g;
- alimentazione: 4,8-6V;
- velocità A 6V: 50 rpm;
- stallo A6V coppia: 3,0 kg/cm;
- lunghezza del cavo: 270 millimetri.

LA BATTERIA E I COMPONENTI PER IL CABLAGGIO ELETTRICO

L'alimentazione del robot è fornita da **4 batterie da 1,5V** tipo AA per un totale di 6V inserite all'interno di un portabatteria **[10]** opportunamente fissato al telaio del robot.

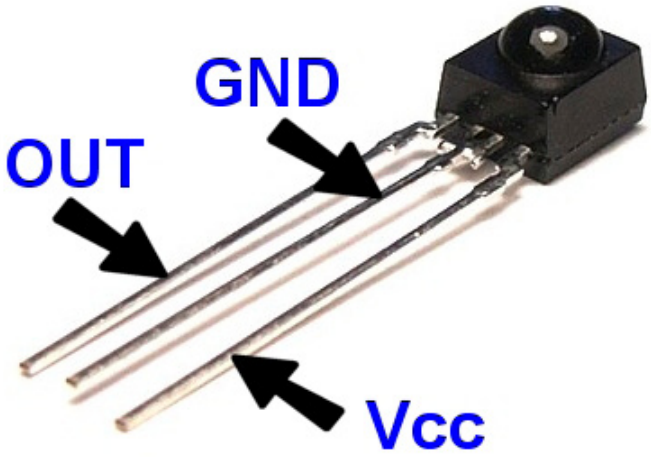
È previsto un interruttore deviatore a levetta subminiatura **[24]** per l'accensione e un spina DC **[25]** dimensioni 2,1 x 5,5 x 9,5mm (terminali da saldare per il collegamento con la scheda Arduino).

Per il collegamento delle varie parti si utilizza un mammut 2 poli **[23]**.

MODULO RICEVITORE INFRAROSSI

Il sensore utilizzato nel progetto è il tipo **TSOP1836**, che fa parte della serie **TSOP18XX** prodotto dalla Vishay, questo potrà essere sostituito con l'equivalente **TSOP 4836** (**da Conrad codice 171107-62**).

Questi sensori, sono ideali come ricevitore per sistemi di telecomando a infrarossi.



Come in precedenza descritto, all'interno del dispositivo, è presente il diodo PIN e il preamplificatore; sono montati su uno stampato inserito all'interno di un contenitore di resina epossidica che ha anche la funzione di filtro IR.

Il segnale di uscita può essere demodulato direttamente e decodificato da un microprocessore.

Il principale vantaggio è il funzionamento affidabile anche in ambienti disturbati e la protezione contro impulsi di uscita incontrollati.

Alcune caratteristiche:

- rilevatore IR e preamplificatore in un unico contenitore;
- filtro interno per la frequenza PCM;
- compatibilità TTL e CMOS;
- uscita attiva bassa;
- schermatura contro i disturbi di natura elettrica;
- maggiore immunità contro tutti i tipi disturbo della luce;
- nessuna comparsa di disturbi sugli impulsi sull'uscita;

- tempo di assestamento dopo l'accensione molto breve (<200s).

Il sensore è sensibile solamente ad infrarossi con lunghezza d'onda di 950 nm e con una frequenza di 36 kHz.

Questo previene le interferenze IR da sorgenti di infrarossi come la luce solare e l'illuminazione domestica.

La luce solare è vista come un'interferenza in corrente continua (0 Hz), mentre l'illuminazione domestica è vista come un'interferenza a 100 Hz, poiché è al di fuori della banda passante del filtro elettronico; tale frequenza è completamente ignorata dal rivelatore IR.

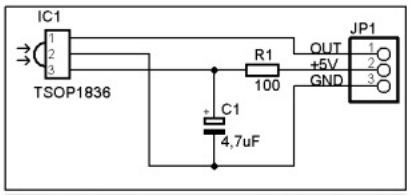
Si può utilizzare il sensore così com'è, oppure si può acquistare un apposito modulo su cui, di solito, sono montati dei componenti passivi che prevedono dei pin di collegamento.

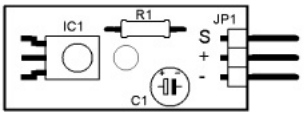
È possibile realizzarne uno su una millefori oppure con **un proprio CS**.

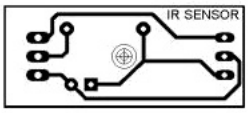
Dato che l'amplificatore interno all'integrato è impostato su un guadagno molto elevato, il sistema potrebbe iniziare ad oscillare molto facilmente; per questo motivo è presente la resistenza **R1** e il condensatore **C1** in modo da disaccoppiare la linee di alimentazione.

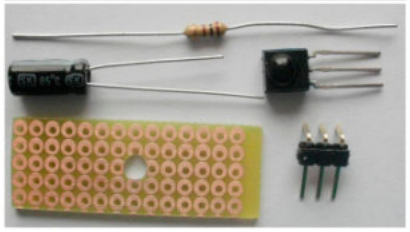
LED ILLUMINAZIONE

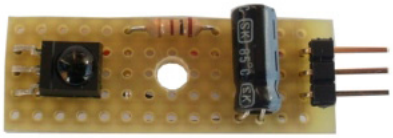
Due led ad alta luminosità completano il robot.







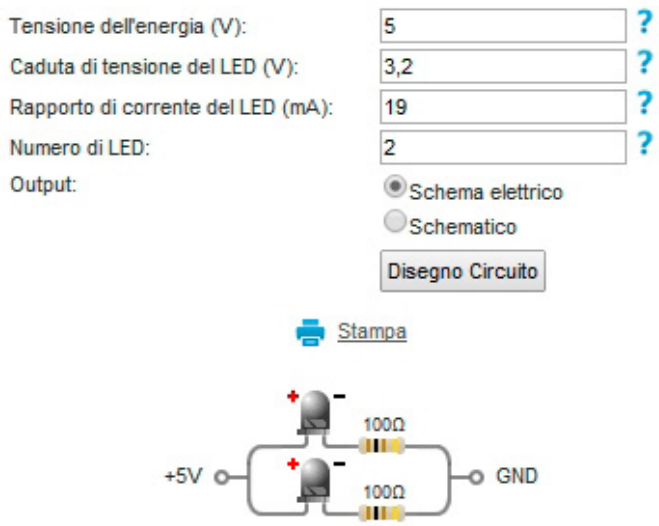




Questi sono attivati dal pin digitale 13 che è anche quello che attiva il led presente sulla **scheda Arduino UNO**.

I led sono connessi in parallelo con la propria resistenza di limitazione del valore di 100 Ω, con tale valore la corrente assorbita dal pin è ancora nel range disponibile che è di 40 mA.

Per il dimensionamento si è utilizzata **questa risorsa della rete**:



Le caratteristiche del led utilizzato tipo L5WCN5 o equivalente sono:

- dimensioni: 5mm;
- colore + materiale emettitore: super bianco GaN/SiC;
- lunghezza d'onda: 465nm;
- intensità luminosa: 20000mcd (If = 20mA);
- angolo di visione: 20 °;

- tensione diretta: 3.2V.

Costruzione particolari del telaio

Il materiale base per la realizzazione del telaio del robot è rappresentato da profilati in alluminio di varie forme e dimensioni che potrà essere trovato presso una grossa ferramenta.

Occorre il seguente materiale:

- angolare L in alluminio 30x15x1 mm;
- angolare L in alluminio 15x10x1 mm;
- piastra alluminio spessore 1,5 mm;
- piastra PVC espanso spessore 3 mm.

Per le lavorazioni, sono necessari un seghetto per metalli e una lima da ferro.

Per le forature, occorre un trapano a colonna con le punte dei diametri indicati.

Si raccomanda di effettuare i tagli e le forature serrando i pezzi su una morsa interponendo eventualmente del materiale plastico per evitare di rovinare la superficie dei pezzi.

Le operazioni indicate producono dei trucioli di alluminio, quindi è fortemente consigliato l'utilizzo dei DPI di protezione: per esempio occhiali protezione in policarbonato e guanti per manutenzione, questi potranno essere trovati presso la stessa Ferramenta dove troverete i profilati.

Con questi attrezzi si costruiscono i seguenti particolari:

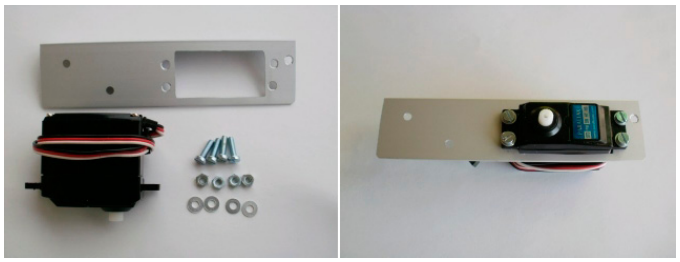
- **[1]** Piastre laterali - n° 2 pezzi;
- **[2]** Profilati fissaggio servomotori - n° 2 pezzi;
- **[3]** Traverse - n° 2 pezzi;
- **[4]** Piastra supporto scheda controllo - n° 1 Pezzo;

Per la loro costruzione si farà riferimento alle misure riportate sui **disegni presenti negli allegati**.

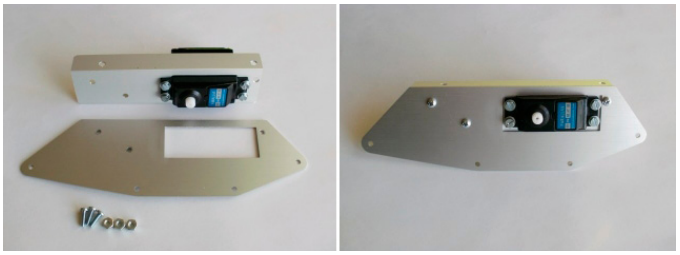
ASSEMBLAGGIO DEI PARTICOLARI MECCANICI

Per il fissaggio dei vari particolari sono inoltre necessari:

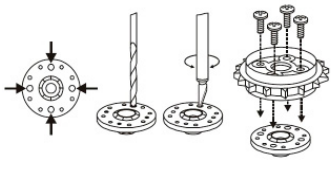
- un cacciavite con la punta a taglio e uno con la punta a croce
- un paio di pinze

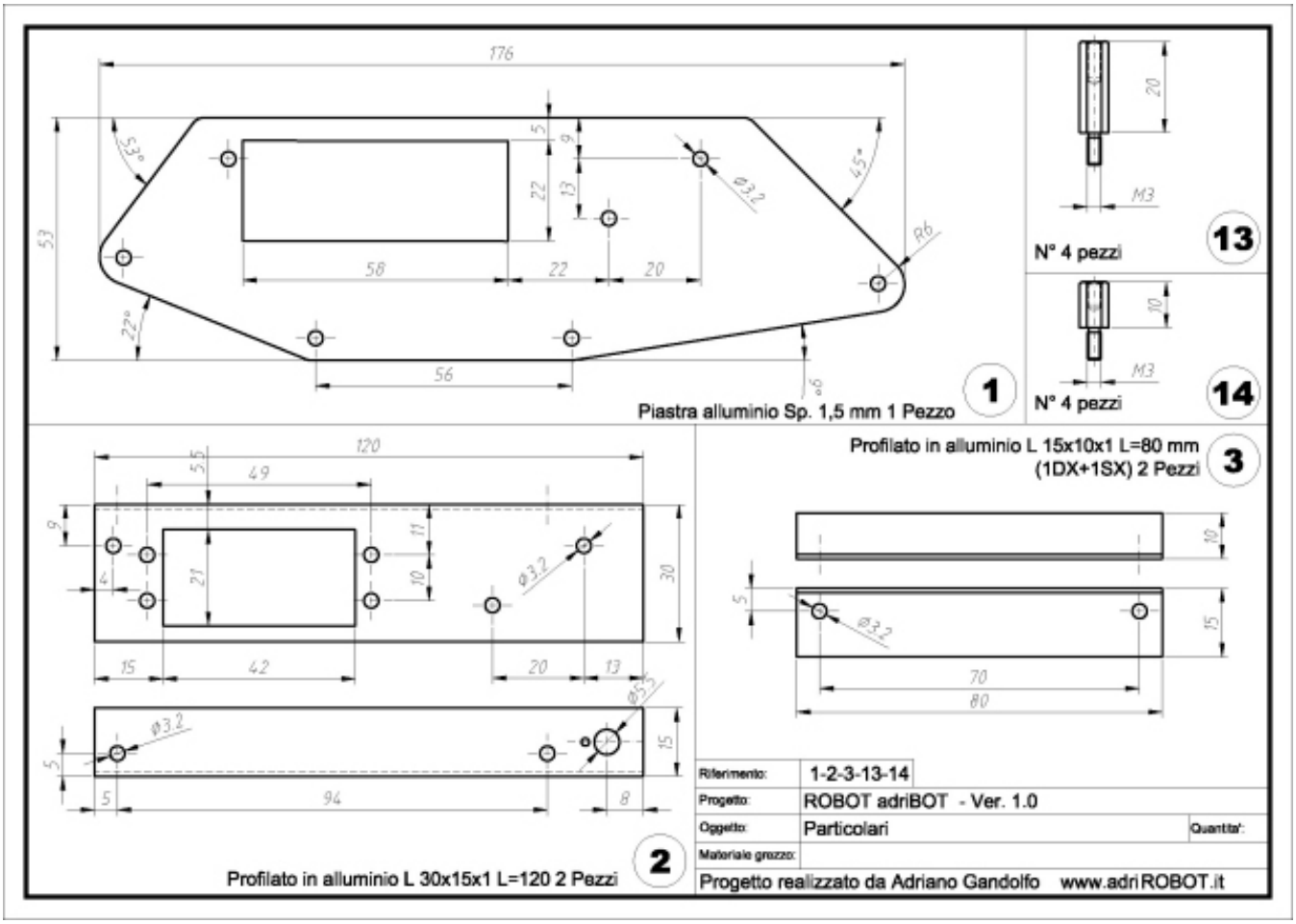


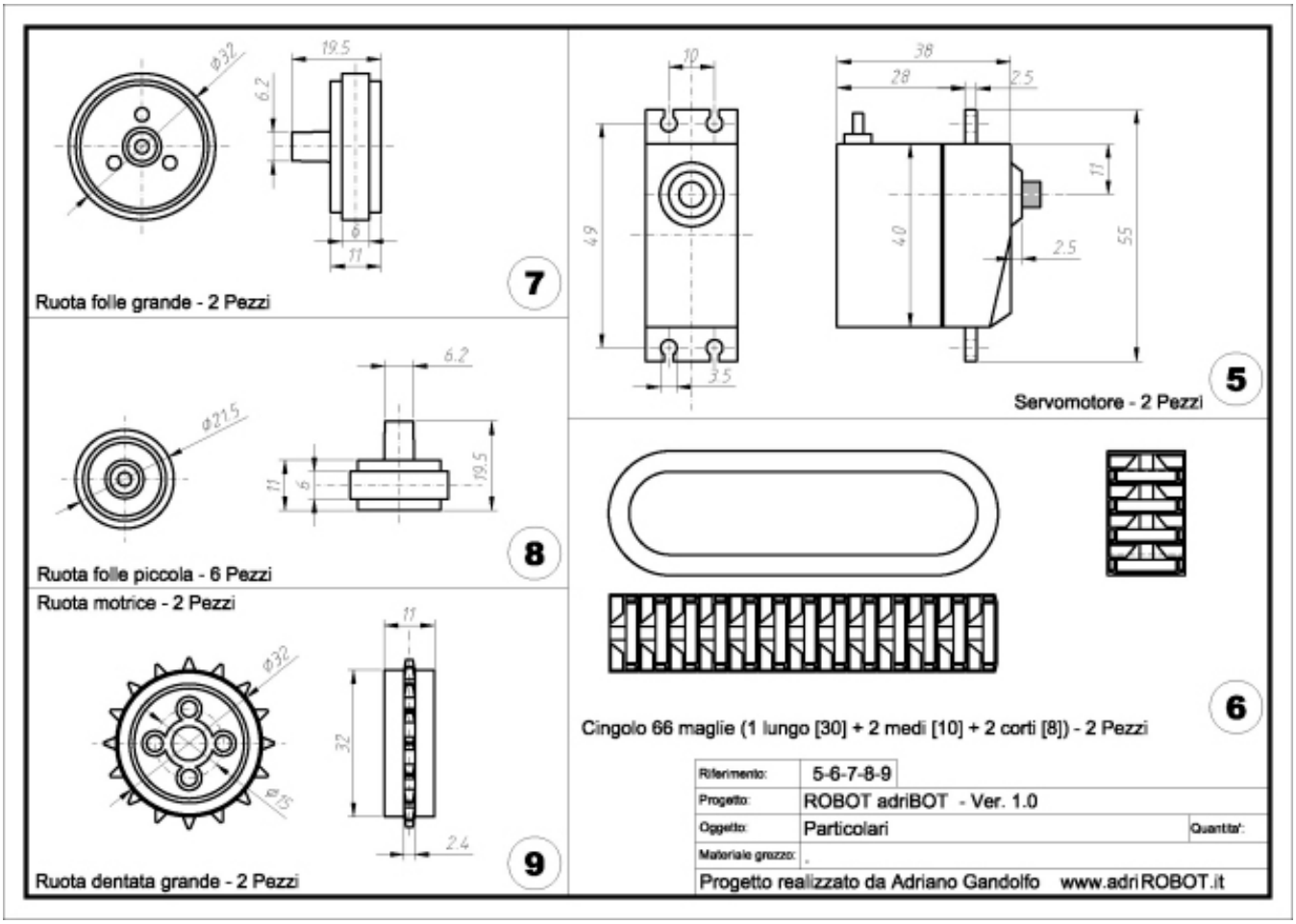
Il montaggio incomincia con le traverse laterali **[2]**, i due servomotori a rotazione continua **[5]** ognuno mediante :n°4 viti M3x8 **[16]**, n°4 dadi M3 **[15]** e n°4 rondelle piane ø3,2 **[19]**.

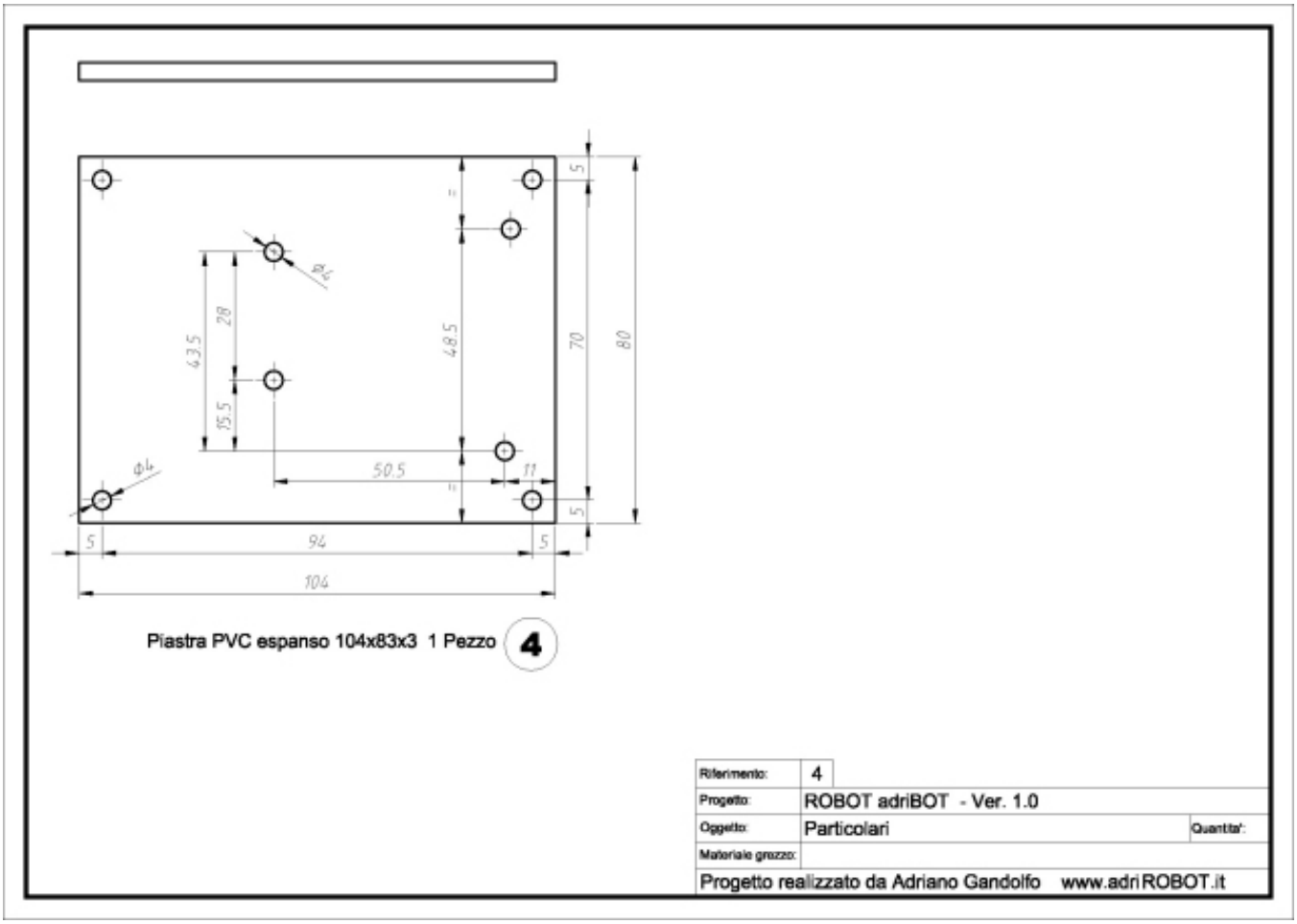


Si fisserà quindi al telaio laterale completo di servomotore la piastra di alluminio laterale **[1]** utilizzando 3 viti lunghe M3x10 **[19]** e 4 dadi M3 **[15]**.



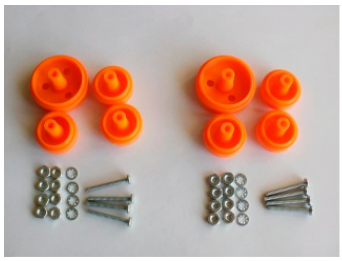

176
53°
53
22
13
5
9
58
22
20
Ø3.2
45°
R6
22°
56
6
Piastra alluminio Sp. 1,5 mm 1 Pezzo
1
20
M3
N° 4 pezzi
13
10
M3
N° 4 pezzi
14
120
55
49
9
4
21
11
10
30
15
42
20
13
Ø5.2
5
94
8
Profilato in alluminio L 30x15x1 L=120 2 Pezzi
2
Profilato in alluminio L 15x10x1 L=80 mm
(1DX+1SX) 2 Pezzi
3
70
80
Riferimento: 1-2-3-13-14
Progetto: ROBOT adriBOT - Ver. 1.0
Oggetto: Particolari
Quantita':
Materiale grezzo:
Progetto realizzato da Adriano Gandolfo www.adriROBOT.it

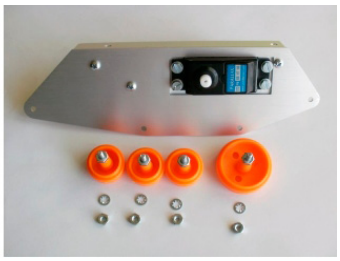

Ø32
19.5
6.2
6
11
Ruota folle grande - 2 Pezzi
7
Ø21.5
6.2
19.5
11
6
Ruota folle piccola - 6 Pezzi
8
Ruota motrice - 2 Pezzi
Ø32
11
32
Ø15
2.4
Ruota dentata grande - 2 Pezzi
9
10
49
3.5
38
28
2.5
11
40
2.5
55
Servomotore - 2 Pezzi
5
Cingolo 66 maglie (1 lungo [30] + 2 medi [10] + 2 corti [8]) - 2 Pezzi
6
Riferimento: 5-6-7-8-9
Progetto: ROBOT adriBOT - Ver. 1.0
Oggetto: Particolari
Quantita':
Materiale grezzo:
Progetto realizzato da Adriano Gandolfo www.adriROBOT.it



Per il fissaggio delle ruote motrici **[9]** al servomotore, occorre allargare quattro fori presenti sulle squadrette che saranno state fornite con i servomotori **[5]**, i fori dovranno essere più piccoli della vite in modo che, quando saranno inserite le viti queste possano fare presa. È possibile eseguire questi fori con una punta di trapano, oppure utilizzando la lama di un cutter per ingrandire i fori come mostrato nella figura.

Se si utilizza il cutter, assicurarsi di non ingrandire eccessivamente i fori, magari mediante l'alesatura con il coltello sia sul lato superiore che inferiore, in questo modo il foro sarà più omogeneo.

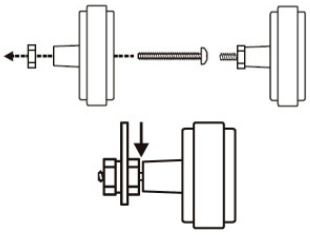


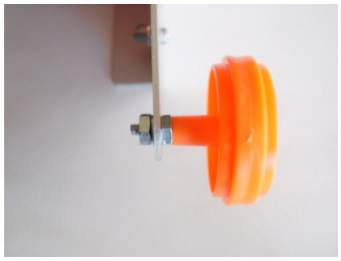


Si passerà poi al fissaggio delle ruote folli necessarie per la guida del cingolo per ogni lato del robot.

Sono necessarie una ruota grande **[8]**, 3 medie **[7]**, n°4 viti M3x25 **[17]** complete di dadi M3 **[15]** e rondelle antisvitamento **[21]**.

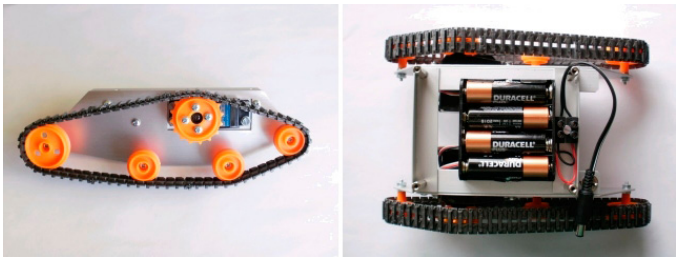
Si potranno unire le squadrette **[9]** al pignone mediante le viti autofilettanti **[18]**, quindi si fisseranno le varie ruote folli.



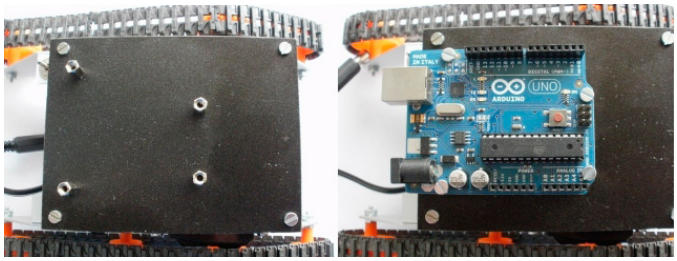


Per il fissaggio delle ruote si procederà serrando il dado sulla ruota poi allentandolo di 1/4 o di 1/3 di giro. Il gruppo ruota va collegato al telaio con un dado e una rondella di bloccaggio all'interno del telaio. Quando i due dadi su ciascuna ruota sono serrati, controllare che ci sia un piccolo spazio tra la ruota e il dado esterno per consentire una facile rotazione, come mostrato nella figura.

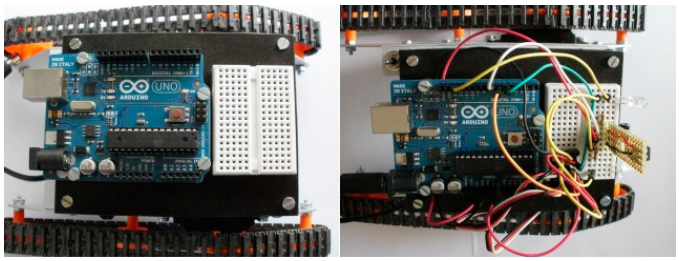
Il montaggio dei due cingoli **[6]** richiede 66 maglie per cui sono necessari: n°1 pezzo da 30 maglie, n°2 pezzo da 10 maglie, n° 2 pezzo da 8 maglie.



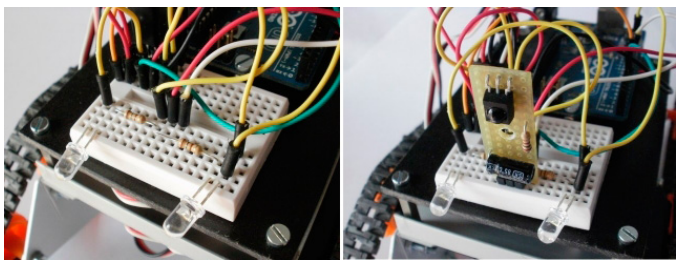
A questo punto i due cingoli potranno essere installati sulle rispettive ruote, esercitando una leggera trazione sul cingolo stesso.
È necessario controllare che ogni ruota possa girare senza sforzo e senza incepparsi.
Se dovessero bloccarsi, allentando il dado e rimontando per mantenere il piccolo spazio come indicato precedentemente si risolverà il problema. I due telai devono essere uniti con le traverse **[3]** fissando il tutto mediante i 4 distanziali esagonali L= 20 **[13]** e utilizzando 4 dati M3 **[15]**.
Si fisserà quindi il portabatteria **[10]** e l'interruttore **[24]** al telaio completando il cablaggio mediante la morsettiera a mammut **[23]**, il tutto terminerà con il connettore polarizzato **[25]** che andrà poi inserito nella presa DC di Arduino.
Si noti che il cavo positivo della batteria andrà collegato al pin centrale del connettore polarizzato.



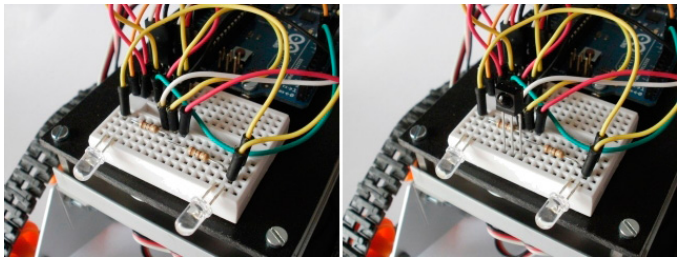
È possibile montare adesso la piastra supporto scheda **[4]** con n° 4 viti M3x8 **[16]** su cui si fisseranno 4 distanziali esagonali L=10 mm **[14]** fissati con 4 dadi M3 **[15]**.
La scheda Arduino UNO **[12]** sarà fissata con n° 4 viti M3x8 **[16]**.



La mini breadboard **[11]** si fisserà alla piastra mediante la sua base autoadesiva.
Si passerà quindi al cablaggio seguendo lo schema riportato nel prossimo paragrafo.



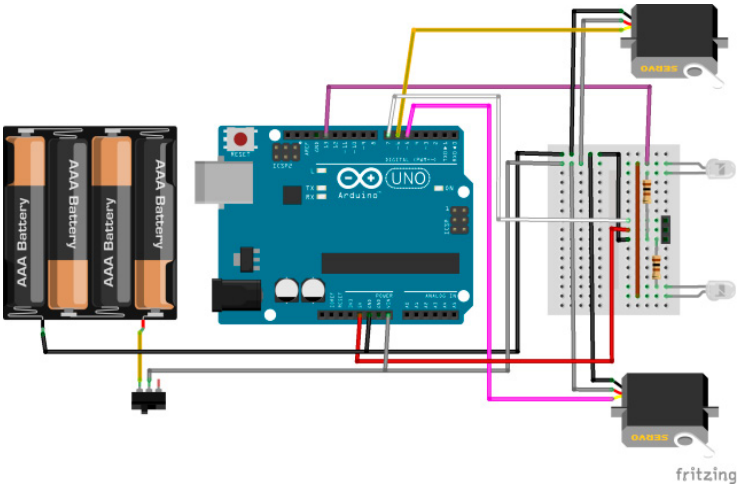
Nella foto è riportato lo schema di collegamento del modulo sensore **[22]**, che prevede il contatto di alimentazione centrale, quello di GND sulla sinistra e l'uscita del segnale sulla destra

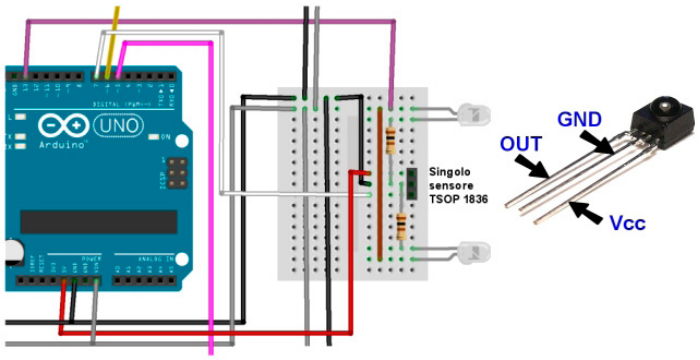


Nella foto è riportato invece lo schema di collegamento del semplice sensore, che possiede il contatto di alimentazione sulla destra, quello di GND al centro e l'uscita del segnale sulla sinistra.

SCHEMA DI COLLEGAMENTO

Nella figura sotto riportata sono presenti i collegamenti da eseguire: l'alimentazione dei servomotori è fornita dalle batterie derivate da Vin, mentre il sensore e il led sono alimentati a +5V.
I segnali sono forniti/letti dai pin digitali 5 e 6 per i servomotori, 7 per il sensore, 13 per i led.





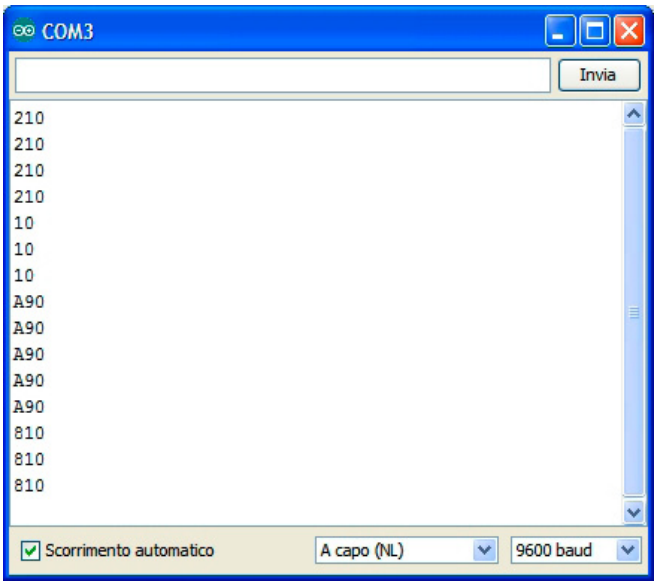
Nel caso si preferisse l'utilizzo del solo sensore **TSOP1836**, occorrerà modificare leggermente il cablaggio in quanto, rispetto al modulo, il pin di alimentazione è a sinistra, il pin di massa è al centro e l'uscita del segnale è a sinistra.

LIBRERIA IREMOTE

Per la gestione del sensore IR occorre fare riferimento al lavoro fatto da **Ken Shirriff** per cui occorrerà prima di tutto, scaricare **l'apposita libreria**.
Il primo programma da utilizzare è quello necessario per la verifica del funzionamento del sensore e per la determinazione del codice a cui sono associati i tasti del telecomando, quindi si caricherà lo sketch **IRrecvDemo** che si trova tra gli esempi della **libreria IRremote**.
Nel programma occorrerà modificare il numero della porta cui è collegati il sensore:

```
int RECV_PIN = 7;  Pin a cui è collegato il sensore
```

a questo punto attivando il monitor seriale, si potrà vedere il numero di codice del comando inviato dal telecomando.



Comandi disponibili in Ricezione	Comandi disponibili in Trasmissione

IRrecv irrecv(receivePin)
irrecv.enableIRIn()
irrecv.decode(&results)
irrecv.resume()
irrecv.blink13(true)
	IRsend irsend;
	irsend.sendNEC(IRcode, numBits);
	irsend.sendSony(IRcode, numBits);
	irsend.sendRC5(IRcode, numBits);
	irsend.sendRC6(IRcode, numBits);
	irsend.sendRaw(rawbuf, rawlen, frequency);

Nota: non tutti i tipi di telecomando sono riconosciuti dalla libreria, i modelli devono essere conformi alla codifica: NEC, Sony, RC5, RC6.

```
#include <IRremote.h>
int RECV_PIN = 7;  Pin a cui è collegato il sensore
IRrecv irrecv(RECV_PIN);
decode_results results;
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
}
void loop() {
  if (irrecv.decode(&results)) {
    Serial println(results.value, HEX);
    irrecv.resume(); // Receive the next value
  }
}
```

È possibile scaricare il **progragramma già modificato**, presente negli allegati.

IL PROGRAMMA DI GESTIONE DEL ROBOT

Una volta determinato a quali codici corrispondono i tasti del proprio telecomando, si caricherà il programma di gestione che **potrete trovare negli allegati**.

Il programma prevede l'utilizzo, oltre che della **libreria IRemote**, anche la **libreria Servo** per la gestione dei servomotori, questa è già disponibile in quanto è tra le librerie standard dell'IDE.

Per i comandi della libreria servo si potrà far riferimento all'apposita **sezione presente sul sito Arduino**.

Il programma prevede prima di tutto la configurazione dei servomotori e l'inizializzazione del ricevitore IR.

Nel loop viene letto il codice decodificato dal sensore e in base al valore, è attivata la relativa routine di azionamento dei due servomotori.

Per le varie assegnazioni di codici sarà necessario modificare l'istruzione

```
if ( results.value == 0xA90 )
```

e sostituire il valore **A90**, che corrisponde alla pressione del tasto **STOP** per il telecomando utilizzato, quello del telecomando in vostro possesso.

È inoltre possibile inserire nuovi comandi semplicemente inserendo altre linee.

```
if ( results.value == 0xXXXX ){
  NUOVA ROUTINE COMANDO();
 }
```

NOTA:

Per il suo funzionamento, la libreria IrRemote utilizza un Timer sia per la trasmissione che per la ricezione.

Se si desidera utilizzare un'altra libreria che richiede lo stesso Timer, è necessario modificare **IRremoteInt.h** per fare in modo che **IrRemote** utilizzi un timer differente.

Per questo motivo, per esempio, il comando **Tone ()** per la creazione di suoni tramite buzzer, senza le opportune modifiche, non funzionerà.

```
#include <Servo.h> // Caricamento libreria per la crazione degli
                   //oggetti Servo che generano segnali PWM
Servo leftDrive;   // Creazione per controllo servomotore sinistro
Servo rightDrive;  // Creazione per controllo servomotore destro

#include <IRremote.h> // Caricamento libreria per gestione sensore IR
int receiver = 7; // Definizione del pin a cui è collegato il sensore IR
IRrecv irrecv(receiver); // Inizializzazione della libreria IRemote
decode_results results; // Decodifica del risultato che Arduino riceve dal sensore
// per ottenere un valore numerico utilizzabile nel confronto.
#define ledpin 13 // Definizione del pin a cui sono collegati i led di illuminazione

void setup()
{
 leftDrive.attach(5);  // attribuisce il servo sul pin 5 all'oggetto servo
 rightDrive.attach(6);  // attribuisce il servo sul pin 6 all'oggetto servo
 pinMode(ledpin, OUTPUT); //Imposta il pin dei led come Output
 irrecv.enableIRIn(); // si utilizza il metodo irrecv.enableIRIn(
// della libreria IRemote perchè legga i valori provenienti dal sensore
 Serial.begin(9600); // Imposta la velocità della seriale per controllo
}

void loop()
{
 if (irrecv.decode(&results)) // È stato ricevuto un segnale IR?
 {
  Serial.println(results.value, HEX); // Mostra eventualemnte sul monitor seriale
 // il valore esadecimale ricevuto
  irrecv.resume();// prosegue mettendo il sensore nuovamente in modalità di ascolto
 }
 if ( results.value == 0x810 ){
  driveForward();
 } // a seconda dei valori decodificati esegue una diversa routine

 if ( results.value == 0x210 ){
  driveBackward();
 }
 if ( results.value == 0xA90 ){
  STOP();
 }
 if ( results.value == 0x410 ){
  turnLeft();
 }
 if ( results.value == 0xA10){
  turnBackLeft();
 }
 if ( results.value == 0x10 ){
  turnRight();
 }
 if ( results.value == 0xC10 ){
  turnBackRight();
```

```
  }
  if ( results.value == 0xE10 ){
    digitalWrite(ledpin, HIGH);
  } //Accende i led frontali
  if ( results.value == 0x910 ){
    digitalWrite(ledpin, LOW);
  } //Spegne i led frontali
 }
//Routine azionamento
 void turnRight() //Il robot gira a destra
 {
  leftDrive.write(180);
   rightDrive.write(180);
 }
 void turnBackRight() //Il robot indietro a destra
 {
  leftDrive.write(90);
  rightDrive.write(0);
 }
 void turnLeft() //Il robot gira a sinistra
 {
  leftDrive.write(0);
  rightDrive.write(0);
 }
 void turnBackLeft() //Il robot gira indietro a sinistra
 {
  leftDrive.write(180);
  rightDrive.write(90);
 }
 void driveForward() //Il Robot va avanti
 {
  leftDrive.write(0);
  rightDrive.write(180);
 }
 void driveBackward() //Il robot va indietro
 {
  leftDrive.write(180);
  rightDrive.write(0);
 }
 void STOP()// Il robot si ferma
 {
  leftDrive.write(90);
  rightDrive.write(90);
 }
```

FILMATO ILLUSTRATIVO

Il filmato mostra l'utilizzo del robot.

https://watch/?v=YSH3z4BSsHU

CONCLUSIONI

Abbiamo approfondito in quest'articolo l'uso di un sensore di raggi infrarossi per decodificare gli impulsi forniti da un telecomando.

Si è inoltre realizzato un robot, che potrà essere potenziato con altri sensori che potranno permettergli, per esempio, di muoversi autonomamente su terreno.

In rete potrete trovare innumerevoli esempi, magari con l'utilizzo di sensori sonar, per cui lasciamo al lettore eventualmente proporre modifiche e nuove implementazioni.

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/telecomando-tv-comandare-robot-cingolato

G.F.A.rduino (Generatore di Funzioni Arbitrarie con Arduino)

di Ernesto Sorrentino

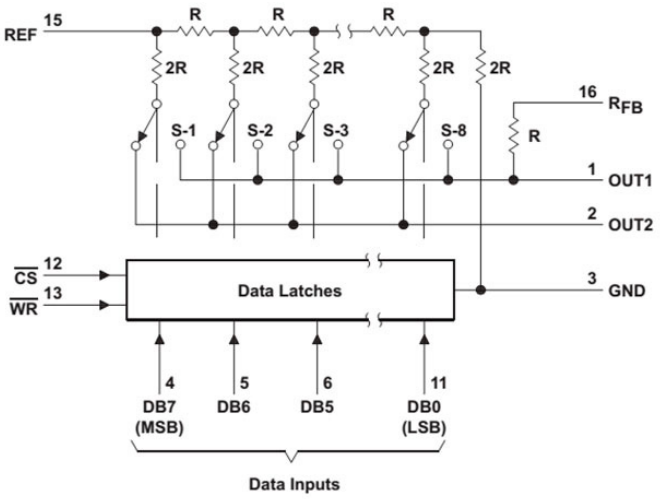
CARATTERISTICHE

Lo scopo principale che mi sono imposto durante la progettazione era di rendere questo strumento **un dispositivo di facile utilizzo**, eliminare tutti quei pulsanti per settare la forma d'onda o per impostare il range della frequenza e renderlo **leggero** per poterlo spostare comodamente tra casa e lavoro; obiettivo centrato in pieno **utilizzando un** Arduino UNO **e un encoder rotativo**. Tutti i comandi sono gestiti dalla rotazione e dal pulsante dell'encoder e per rendere l'utilizzo ancora più pratico, le regolazioni e le impostazioni sono visualizzate su un display TFT a colori di Arduino.

https://www.youtube.com/watch?v=OYSejYt5OvU

IL CIRCUITO ELETTRONICO

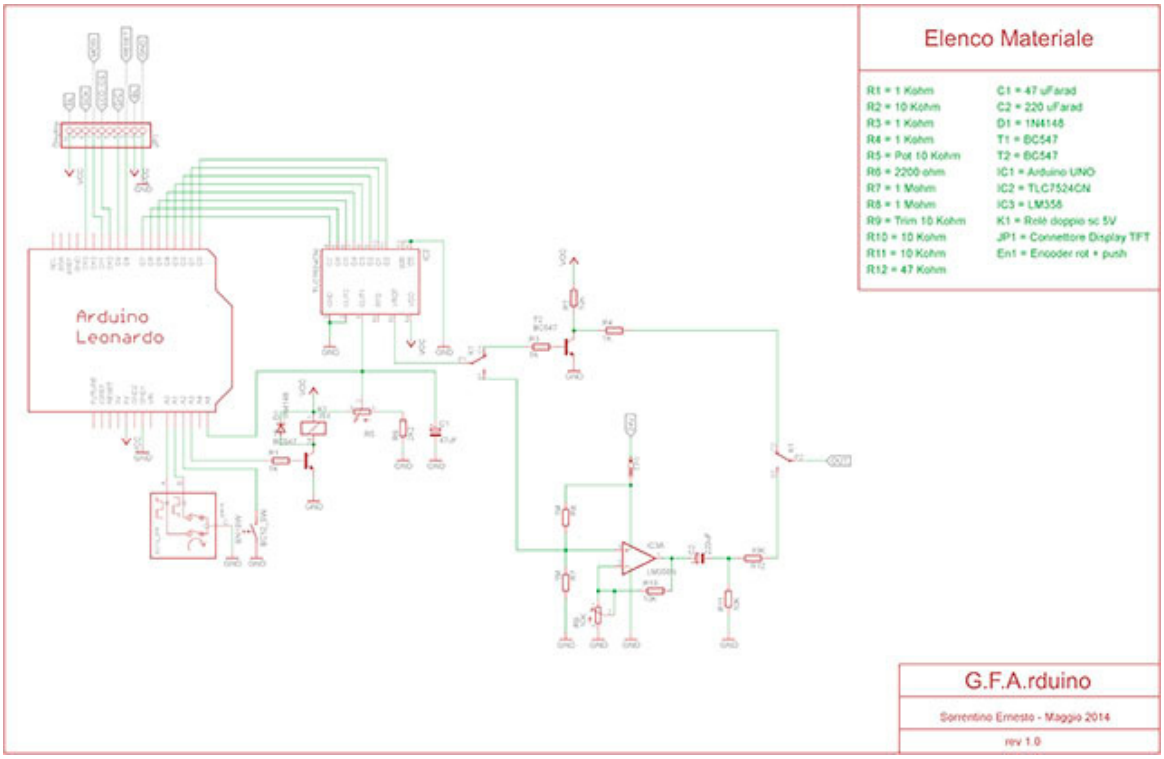
Per la creazione della forma d'onda ho utilizzato un DAC tipo R2R ossia il TLC7524CN. Per non dilungarmi troppo rimando, per chi non conosce la tecnica della conversione digitale/analogico con scala R2R, a quest'articolo di EOS. L'integrato TLC7524CN sostituisce la scala di resistenze ed è possibile scegliere riferimenti di tensioni sia per lo zero sia per il Vcc Max.



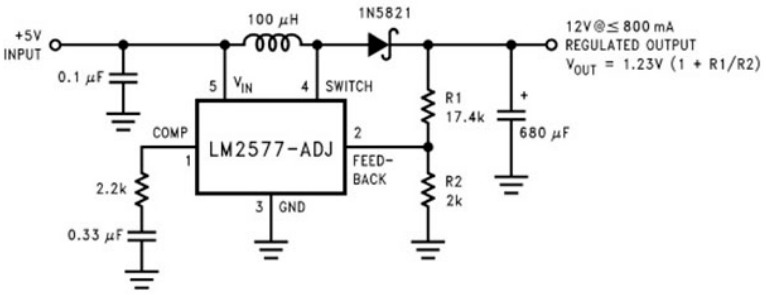
I comandi d'input sono collegati direttamente alla **PORTA D** dell'Atmega.

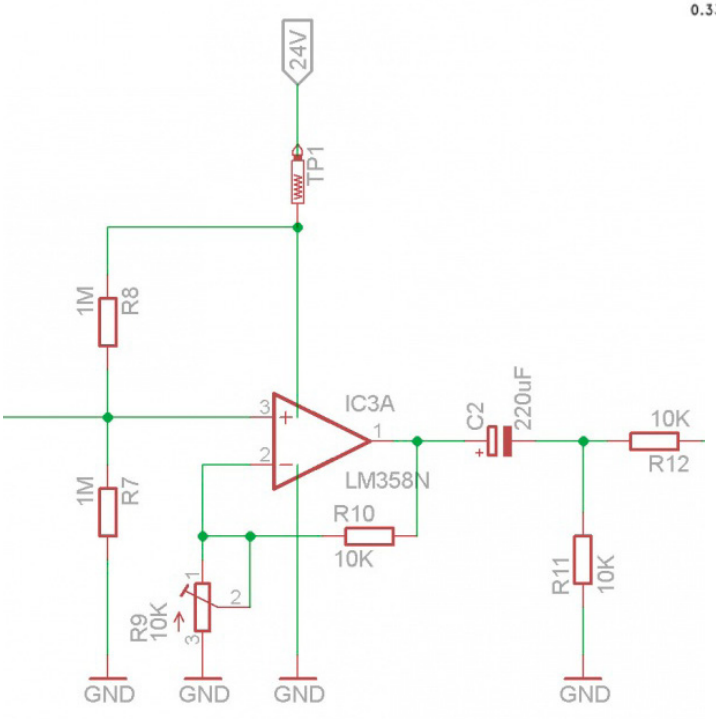
L'onda è generata con una risoluzione di 100 parti, ciò significa che il microcontrollore deve generare almeno un ciclo di 100 impulsi per creare un'onda a 1 Hz, pertanto **dato che il limite di ON/OFF delle porte di Arduino è 100 KHz la frequenza massima dell'onda sarà di 1 Khz**. Per aumentare il limite della frequenza ho optato nella riduzione della risoluzione dell'onda, realizzando un'equazione capace di ridurre la risoluzione in modo automatico man mano che incrementa la frequenza. Ho impostato come limite di **Fmax a 5600 Hz** perché ritenevo che a tale frequenza l'onda fosse abbastanza accettabile ma **è possibile, cambiando un'impostazione nello sketch, aumentare ancora la frequenza sempre però a discapito della risoluzione**.

L'onda in uscita generata dal DAC sarà amplificata dall'operazionale (LM358) in configurazio-



ne non invertente con un rapporto fisso, **il trimmer R9 collegato al pin 2 è utilizzato per regolare la soglia massima di amplificazione**.

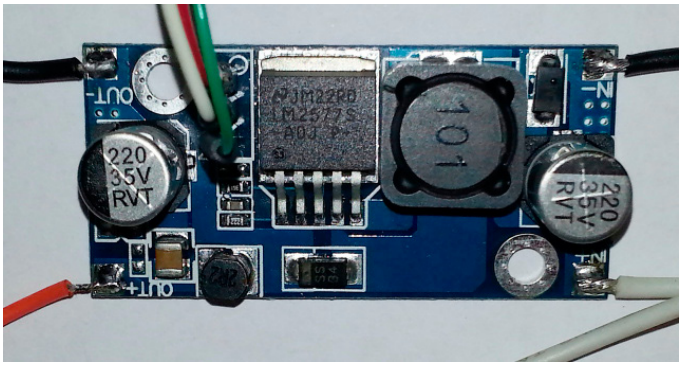




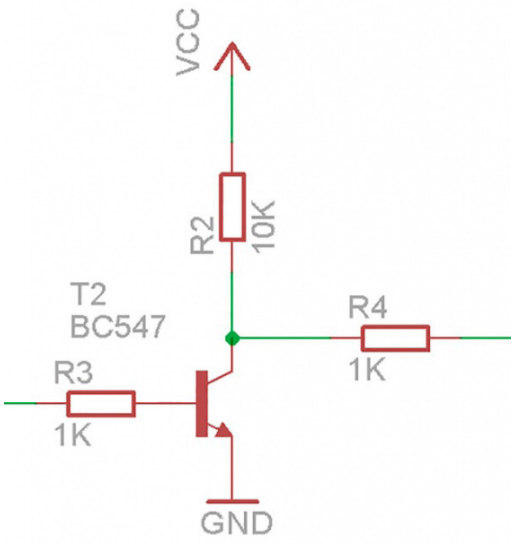
Si noti, dallo schema, che tutta l'elettronica è alimentata da Arduino (5V) mentre tale l'operazionale è alimentato con una tensione maggiore (24V), per ottenere un'ampiezza del segnale fino a 15V picco/picco. Per erogare una tensione di 24V e rendere il circuito portatile ho deciso di utilizzare un DC-DC StepUp, anche questo dispositivo è alimentato da Arduino (5V) e genera una tensione in uscita regolabile fino a 30 V con corrente max di 500 mA **(soglia massima di erogazione da Arduino)**. Un DC-DC StepUp può essere facilmente costruito con l'integrato LM2755 come da schema seguente:

Tuttavia ho scelto una versione già montata reperibile sulla rete a un prezzo modico.

Questa versione presenta un trimmer da 10 K per regolare la tensione in uscita, una volta impostata la tensione a 24 V il trimmer non verrà più utilizzato. Per variare l'ampiezza del segnale,



invece, useremo il potenziometro da 10 K collegato al pin 1 del TLC7524CN, che **regola il livello MSB dei dati** sulla scala R2R. Invece **per il segnale TTL ho utilizzato un transistor BC547** pilotato dal DAC, una soluzione per ottenere un segnale ben squadrato e con ampiezza di 5 V.



REGOLAZIONI

Le regolazioni da compiere sono solo per l'amplificazione del segnale, eseguire i seguenti passi:

1. regolare il trimmer del DC-DC StepUp per ottenere una tensione a carico di 24V;
2. posizionare il trimmer dell'operazionale a metà corsa;
3. avviare il generatore in modalità sinusoidale;
4. con un oscilloscopio controllare la distorsione del segnale e correggerla col trimmer dell'operazionale;
5. se l'onda presenta dei picchi superflui diminuire leggermente la tensione in uscita del StepUp, invece se l'onda risulta tagliata nell'estremità aumentare leggermente la tensione in uscita del StepUp.

Regolazione ultimata.

ELENCO MATERIALE UTILIZZATO

1 - Arduino UNO
1 - Arduino LCD TFT 1,77"
1 - Encoder rotativo con pulsante
1 - DAC TLC7524
2 - Amp OP LM358
1 - Relè due vie 5V
N - materiale vario come resistenze, condensatori e transistor acquistabile presso conrad.it

IL PROGRAMMA

Il cuore del programma sta nell'utilizzo della funzione interrupt del Timer per pilotare i dati sulle porte D dell'Atmega collegato al TLC7524, **per questo progetto ho scelto il Timer1 per i suoi 16 bit**. L'interrupt è una routine di servizio (**ISR - Interrupt Service Routine**) al di fuori del programma originale che si avvia ad un specifico evento, in questo caso si avvia ogni volta che il contatore del Timer raggiunge una frequenza stabilita. A ogni evento d'interrupt, l'Atmega, invia sulla PORTD 1 delle 100 parti per la creazione dell'onda. Il registro per assegnare il valore da comparare al contatore del Timer è l'**OCR1A** (relativo al Timer1).
La formula di default è:

```
OCR1A = (16000000 / (prescaler * frequenza desiderata di interrupt)) -1
```

dove 16000000 è la frequenza di clock di Arduino. In questo progetto invece la formula è stata impostata nel seguente modo:

```
OCR1A = (16000000 / (1 * (frequenza desiderata* 100))-1
```

Si noti che il prescaler è impostato a 1 e la frequenza è moltiplicata per 100 che sono le parti che compongono l'onda. Nello sketch risulta nel seguente modo:

```
OCR = (16000000/(Hz*K_Hz))-1;        // riga n°164
```

Il valore di Hz è letto dall'encoder, la lettura avviene grazie ad una libreria creata da "Paul Stoffregen". Questa libreria conteggia il valore di rotazione dell'encoder in un contatore reso disponibile sia per la **lettura** tramite il comando:

```
Encoder.read();
```

sia per la **scrittura**, per impostare dei limiti, col comando:

```
Encoder.write(valore);
```

Per approfondire, ecco alcuni dettagli.

Dallo sketch si può notare che il ciclo fondamentale sono proprio queste due funzioni, la lettura dell'encoder e il settaggio del registro OCR1A.

```
void loop() {
encoder();
setting(Hz, K);
...}
```

FORMA D'ONDA

Per la generazione delle forme d'onda ho realizzato quattro liste, una per ogni tipo di forma d'onda escluso il TTL.

```
byte sine[]= {127, 134, 142, 150, 158, ....};
byte triangle[] = {127, 132, 138, 143, .....};
byte saw[] = {255, 252, 250, 247, 245,....};
byte squar[] = {255, 255, 255, 255, ........};
```

Ogni lista è composta di 100 valori compresi tra 0 e 255. Dove 0 corrisponde a 0 V in uscita del DAC mentre 255 corrisponde a 5 V. È possibile creare una curva nuova rispettando però la **condizione che lo zero del segnale corrisponda a 127, ossia a circa 2.5 V**; questo permetterà di ottenere un segnale, grazie al condensatore di disaccoppiamento "**C2**", in tensione duale. È consigliato sostituire la curva nuova con una di quelle già presenti.

IMPOSTAZIONE UTENTE

Nello sketch ci sono alcune impostazioni che possono essere gestite dall'utente per personalizzare il progetto, come:

- ***user_Hz_min***: si può impostare il valore minimo della frequenza da selezionare con l'encoder;
- ***user_Hz_max***: si può impostare la frequenza massima regolabile con l'encoder. Di default è impostata a 5600, può essere aumentata ma a discapito della risoluzione dell'onda;
- ***user_Hz_start***: l'utente può impostare la frequenza d'inizio all'avvio del programma.

```
#define user_Hz_min 5          // Frequenza minima selezionabile
#define user_Hz_max 5600       // Frequenza massima selezionabile
#define user_Hz_start 100      // Frequenza iniziale dopo l'avvio del programma
#define Vmax_out 9.7           // Tensione max in uscita dall'ampl. operazionale
```

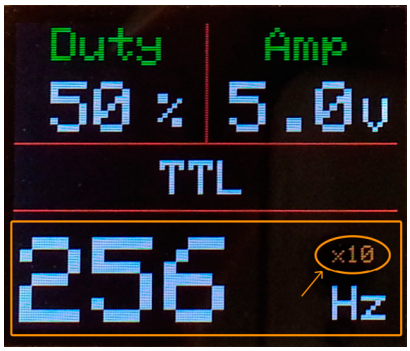
REGOLAZIONI

Anche nel programma c'è una piccola impostazione da ultimare e riguarda la parte del Voltmetro. La lettura dell'ampiezza del segnale non avviene direttamente all'uscita dell'amplificatore operazionale **LM358**, una scelta fatta per evitare distorsioni del segnale e componenti aggiuntivi come inseguitori di tensioni. Ho preferito **leggere la tensione al pin 1 del DAC che regola la soglia MSB dei dati**, ovvio che la lettura non è reale ma conoscendo la tensione massima in uscita dell'amplificatore operazionale è possibile regolare lo script per ottenere un risultato preciso da visualizzare sul display. **L'impostazione va settata inserendo il valore della tensione massima, letta con un multimetro, in uscita dall'amplificatore LM358 selezionando l'onda "Quadra". Tale valore deve essere inserito alla voce "Vmax_out" dell'impostazione utente**.

I COMANDI

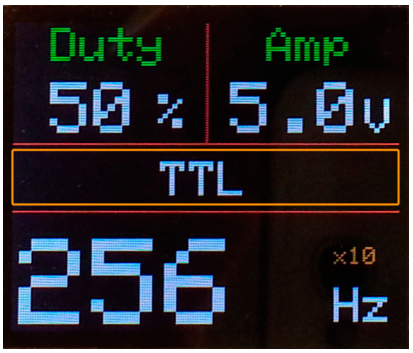
Tutte le impostazioni sono state divise in tre menù visibili sul display in tre sezioni, nel seguente modo:

1° sezione: Gestione frequenza (impostazione di default);



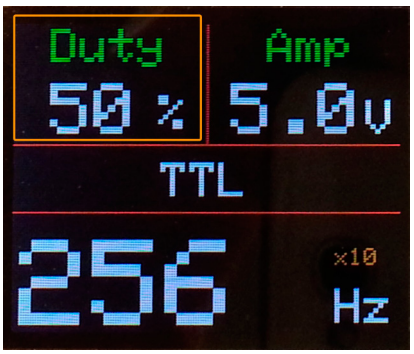
In questo menù, oltre alla regolazione della frequenza, si può impostare il fattore di moltiplicazione (1X, 10X o 100X).

2° sezione : Scelta forma d'onda;



In questa sezione si imposta la forma d'onda da generare.

3° sezione: Gestione Duty-Cycle (solo per TTL);



In quest'ultimo menù si regola il duty-cycle e la frequanza in contemporanea, valido solo per il segnale TTL.

Per passare da una sezione all'altra bisogna premere il pulsante dell'encoder e tenerlo premuto per almeno un secondo mentre per modificare la sezione scelta, bisogna premere una sola volta il pulsante dell'encoder. Tutte le scelte sono selezionabili in ordine come descritto sopra e a ripetizioni in loop. Quando si passa da

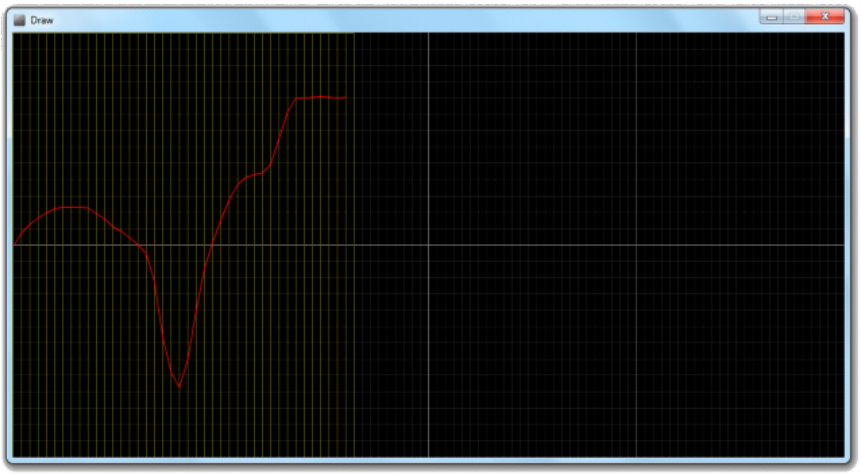
una sezione all'altra il testo, della sezione scelta, cambia colore in giallo per indicare la possibilità di regolazione.

REALIZZAZIONE

Purtroppo per motivi di tempo non sono riuscito a realizzare uno stampato ma lo schema è stato disegnato con Eagle per tanto chiunque può crearne uno con pochi e semplici passaggi. Tra gli allegati sono presenti due librerie di Eagle, uno per il modulo Arduino e l'altro per l'encoder rotativo. Per chi volesse realizzare un PCB consiglio questa guida pubblicata su EOS.

PROSSIMAMENTE

Il progetto può crescere e migliore, le idee per evolverlo sono tante come realizzare un applicazione con Processing per disegnare nuove curve e caricarle sul SD, presente sulla board del display, tramite la porta seriale di Arduino.



progetto in fase di lavorazione, disponibile al più presto nel prossimo articolo.

ALLEGATO

G_F_A_rduino (documentazione)

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/gfarduino-generatore-di-funzioni-arbitrarie-con-arduino

Arduino abbassa il volume degli spot TV! [Fai-Da-Te]

di Piero Boccadoro

INTRODUZIONE

Qualche tempo fa, su queste pagine, pubblicammo un articolo che parlava di una possibile soluzione al problema degli spot televisivi dal volume decisamente troppo invadente. La sua pubblicazione andò ad aggiungersi alle tante per le quali realizzammo il concorso Make-4Cash. Ma anche senza cash, il progetto, almeno il prototipo, ora è completo.

Le considerazioni preliminari da fare che danno vita a questo progetto sono le stesse che vennero fatte proprio nell'articolo di presentazione del concept: gli spot televisivi, ma spesso anche alcune porzioni dei film d'azione, alcune scene, sono davvero fastidiosi. Potrebbe essere utile ed interessante realizzare un sistema che sia in grado di gestire in maniera intelligente queste situazioni, di fatto realizzando un equalizzazione con il volume che di solito viene considerato accettabile ovvero quello del parlato o di scene comunque non altrettanto movimentate oppure appariscenti.

E' da queste idee e con queste premesse che nasce il progetto che fu presentato all'epoca e che oggi è completo funzionante. Ha subito alcune modifiche, è stato rivisto e pesantemente ottimizzato ed è per questo che ora lo proponiamo.

COME FUNZIONA

Il funzionamento di questo progetto è abbastanza semplice dal momento che è dotato di un microfono, un pulsante, due LED e poco più. Il principio di funzionamento è semplicemente un meccanismo a soglia che rilevi il volume attuale, lo compari con un valore di confronto ed agisca di conseguenza. Se siamo sopra la soglia prestabilita di rumorosità accettabile, il sistema deve inviare il comando per abbassare il volume mentre in caso contrario per alzarlo.

Nell'immagine di copertina, un test fatto sulla riproduzione di un file audio qualsiasi, per darvene una breve dimostrazione.

Tutto questo deve essere fatto mantenendo il dispositivo il più vicino possibile alla cassa del televisore. Una delle caratteristiche di questo progetto è proprio la sensibilità del microfono che viene ottimizzata mantenendo il dispositivo non più lontano di 5 cm dalla cassa. Sono state fatte diverse prove durante la realizzazione del progetto e vale la pena di anticipare adesso che, naturalmente, minore è la distanza migliore è stato il risultato e la capacità del progetto di seguire le variazioni e l'andamento del livello del volume nel corso del tempo. Vi consigliamo, pertanto, se sceglierete di provare a cimentarvi in questa sfida, di partire da queste distanze così come vi consigliamo di partire da questo hardware per poi modificarlo.

COSA FARE

Prima di poter partire con la realizzazione del codice ed il test è necessario compiere alcuni passi per ottimizzare i tempi. La prima cosa che serve è l'acquisizione dei comandi. Come nel mio caso, il televisore non gestisce il protocollo

RC5 o RC6 e per questo motivo, dopo diverso tempo, è stato necessario pensare di ricostruire la forma d'onda a mano.
Una volta fatto questo, che verrà descritto più avanti, si è passati alla realizzazione del circuito partendo dallo schematico che vedrete tra un attimo.
Scritto il programma, poi, tutto ha funzionato per il meglio.
Quindi per prima cosa bisogna studiare qual è l'hardware di cui si ha bisogno, scrivere la BOM, stilare lo schematico, lavorare sui codici ad infrarosso, scrivere il programma ed effettuare i vari test.
Bene, ora che abbiamo le idee chiare su che cosa dobbiamo fare, facciamolo!

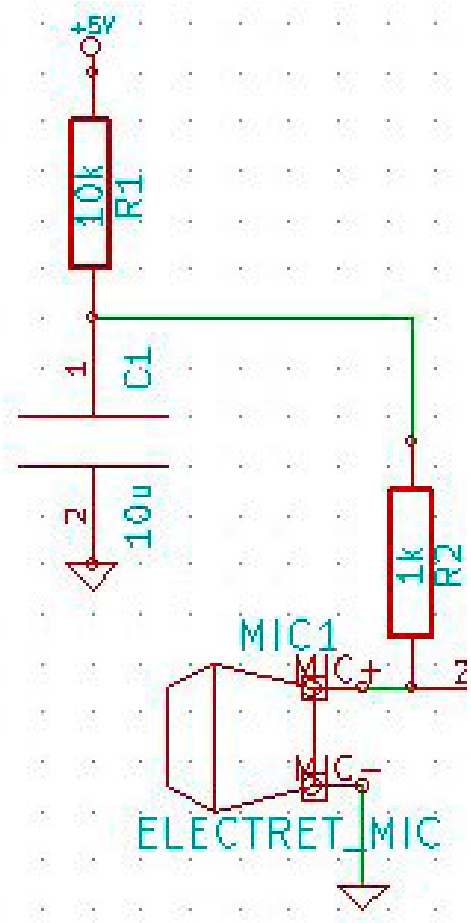
L'HARDWARE

Non c'è progetto senza l'hardware di riferimento. Ed ecco che cosa vi occorre:

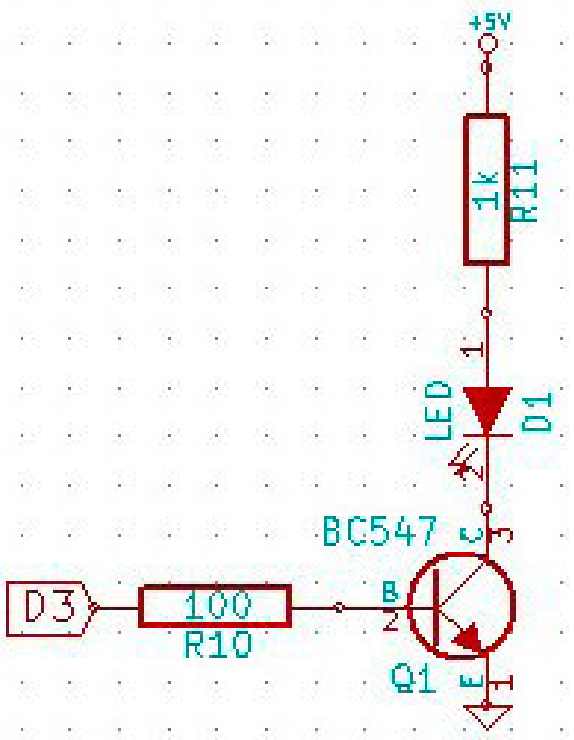
- **1 microfono Electret;**
- **2 resistenze da 100 Ohm;**
- **6 resistenze da 1 kOhm;**
- **1 resistenza da 10 kOhm;**
- **1 resistenza da 110 kOhm;**
- **1 resistenza da 220 kOhm;**
- **1 condensatore da 220 uF;**
- **1 condensator3 da 4.7 nF;**
- **3 condensatori da 10 nF;**
- **1 LM324N (Quad OpAmp);**
- **1 pulsante;**
- **1 diodo LED IR;**
- **1 ricevitore IR 38 kHz (TSOP2236);**
- **1 Arduino;**
- **fili vari.**

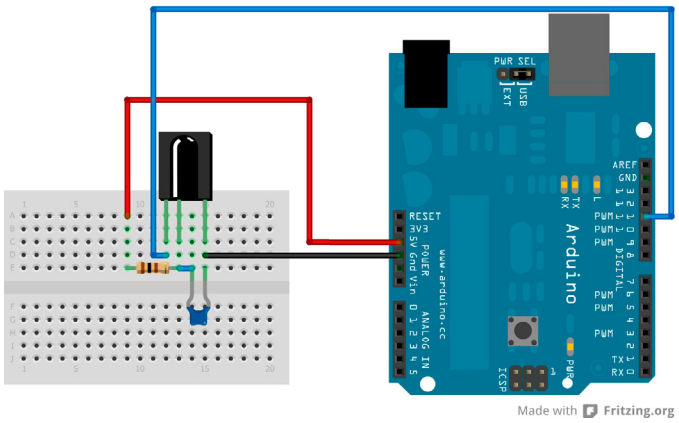
Una piccola osservazione: l'alimentazione per il funzionamento di questo progetto è stata sempre prelevata tramite l'USB del computer. Se dobbiamo pensare di renderlo autonomo, certamente servirà un'alimentazione dedicata e quindi una batteria.

Una volta che e stata realizzata la connessione del microfono, serve subito lavorare sull'audio e per fare questo è stato usato un filtro in configurazione Sallen Key del second'ordine. Utilizzeremo delle immagini per farvi vedere come siamo andati avanti nella realizzazione ma in generale sappiate che quello che non trovate qui nell'articolo esplicitamente riportato è contenuto negli allegati che troverete più avanti. In ogni caso, state tranquilli: la documentazione è completa.
Dicevamo della sezione audio, la quale ci conduce direttamente ad uno degli ingressi di Arduino.

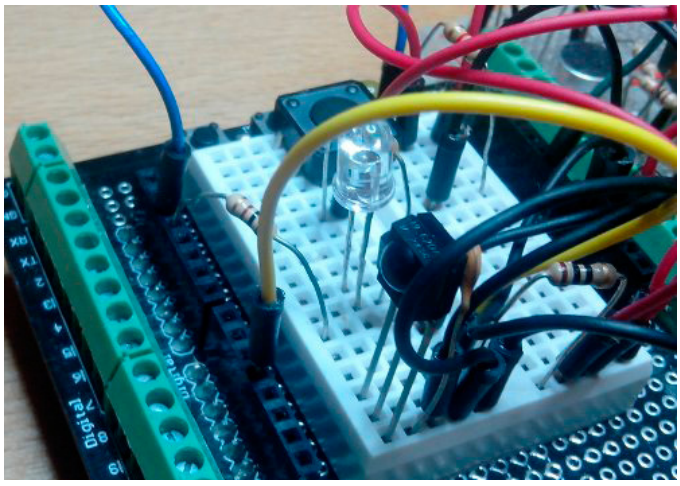


Una volta fatto questo non abbiamo finito perché bisogna creare circuiti di condizionamento per tutta la sezione che gestisce gli infrarossi. Ed eccoli:

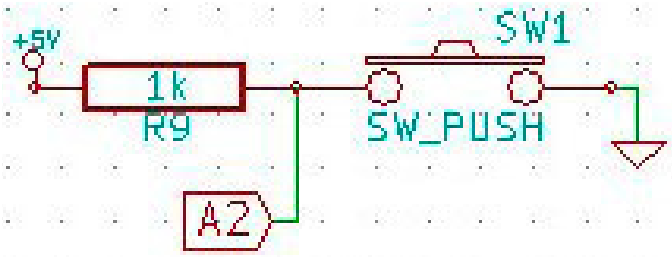




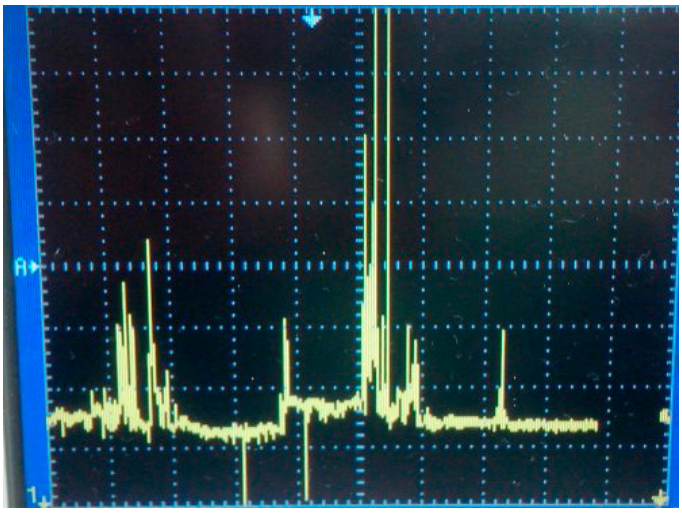
Che, messo in pratica e realizzato, vien fuori così:



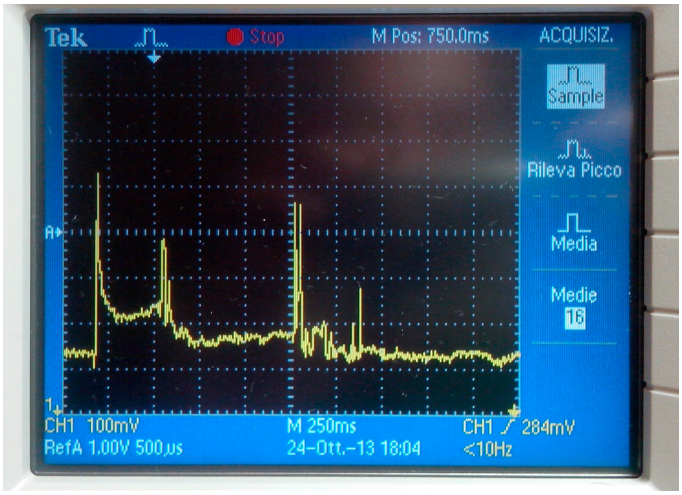
E non è ancora tutto perché abbiamo detto che abbiamo bisogno di un pulsante. Ecco come è stato collegato



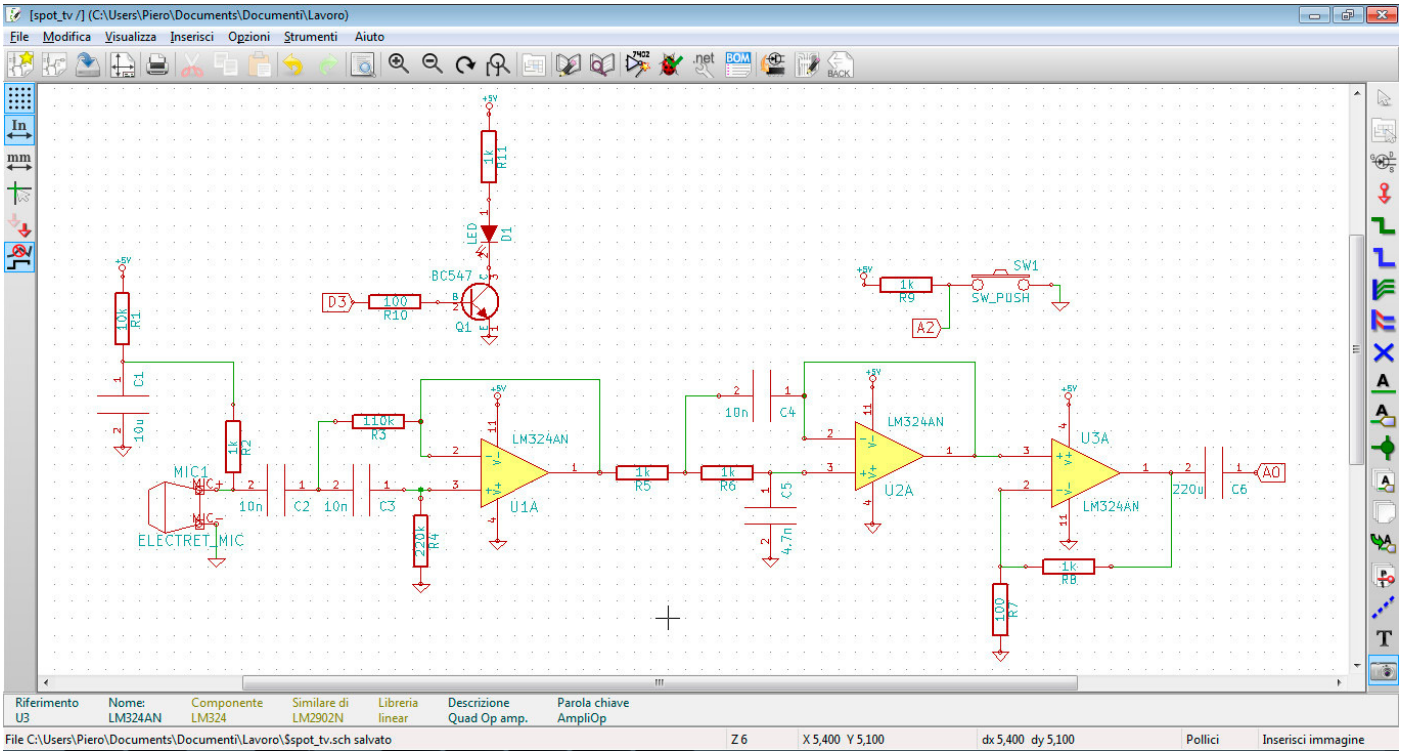
Una volta realizzate tutte queste connessioni e alcuni dei risultati che abbiamo ottenuto utilizzando l'oscilloscopio:



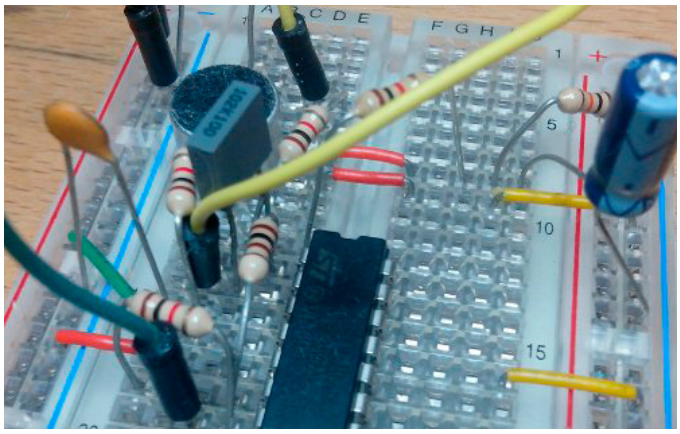
L'oscilloscopio inizia a mostrare ben altre forme d'onda, nel corso dei test e piano piano si distingue sempre più chiaramente ciò che è rumore da ciò che non lo è:



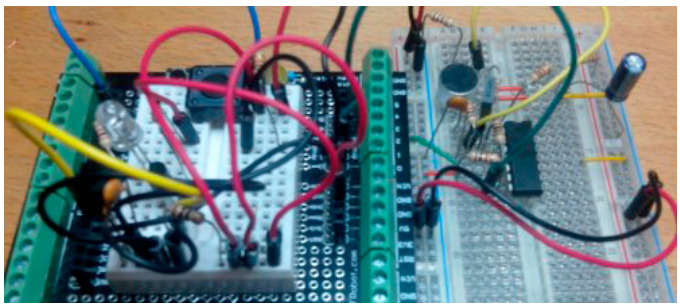
Nell'immagine che avete appena visto, è stato riportato l'audio registrato dal microfono ascoltando dei suoni.



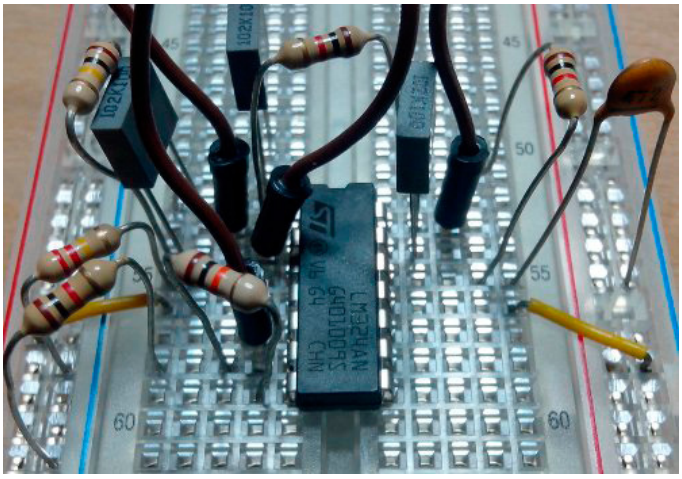
Per iniziare i test, è stato realizzato un semplice filtro passa-basso (metà del Sallen Key)



Assemblato il prototipo, e verificato il funzionamento



realizzato il Sallen Key del secondo ordine
Il tutto è stato assemblato opportunamente a dare il prototipo completo.



Bene, ora l'hardware è pronto.

SCOPRIAMO I CODICI

Devo, per poter fare in modo che tutto funzioni, scoprire come sono fatte le forme d'onda in uscita dal telecomando in corrispondenza dei due che dovrà essere in grado di inviare cioè l'abbassamento e l'innalzamento del volume. Per fare questo è stato utilizzato il seguente codice con cui programmare Arduino.

```
#include <IRremote.h>

int ricevitore = 10; // al pin 10 è connesso il ricevitore IR
IRrecv irrecv(receiver); // crea l'istanza 'irrecv' come da libreria
decode_results results;

void setup()
{
  Serial.begin(9600); // Il debug dell'applicazione vien fatto da seriale
  irrecv.enableIRIn(); // Inizializza il ricevitore
  pinMode(10, INPUT); // Il pin 10 serve come input
}

void loop()
{
  if (irrecv.decode(&results)) // have we received an IR signal?
  {
    Serial.println(results.value, HEX); // mostra il codice sulla seriale in esadecimale
    irrecv.resume(); // riceve il valore successivo
  }
}
```

Questo corrisponde ad una configurazione molto semplice in cui il ricevitore ad infrarossi è collegato direttamente da Arduino grazie all'utilizzo di semplici resistenze condensatori.

parlando dei codici che sono stati ricevuti, eccoli:

vol +:

```
68733A46 (HEX),
0110 1000 0111 0011 0011 1010 0100 0110 (BIN),
1752382022 (RAW)
```

vol -:

```
83B19366 (HEX),
1000 0011 1011 0001 1001 0011 0110 0110 (BIN),
2209452902 (RAW)
```

È possibile che i codici ottenuti debbano essere rivisti dal momento che la notazione esadecimale non va bene ed infatti sono stati riportati sia con la notazione esadecimale sia in binario sia in formato grezzo.

È evidente che i dati raw servono a poco se non esiste un metodo univoco per l'interpretazione ma invece convertire in binario ha diversi vantaggi, non ultimo il fatto che sull'oscilloscopio si vedono i bit!

Ecco per quale motivo è stato importante cercare un metodo per effettuare una conversione da esadecimale a binario e, tanto per utilizzare un esempio che noi abbiamo già fatto eco un paio di comandi in MATLAB che sicuramente vi faranno raggiungere lo scopo, se lo avete a disposizione.

```
bin_str = dec2bin(hex2dec(hex_str))
```

se facessimo un esempio potremmo scrivere

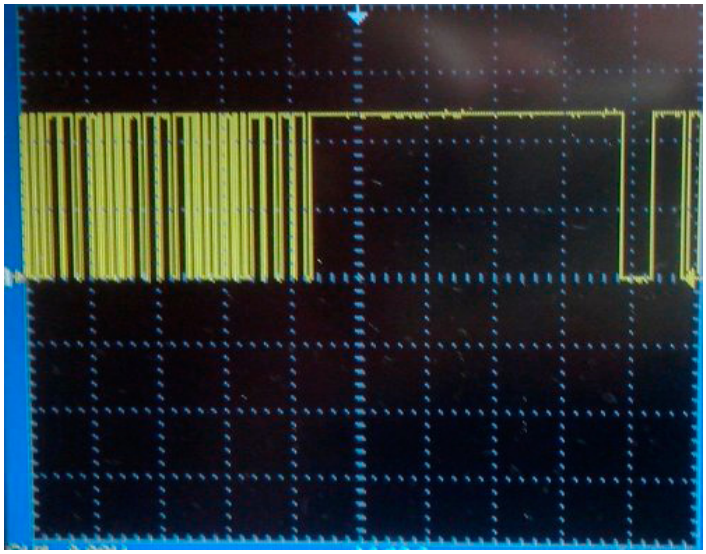
```
hex_str = 'AF5'
bin_str = dec2bin(hex2dec(hex_str))
bin_str = 101011110101
```

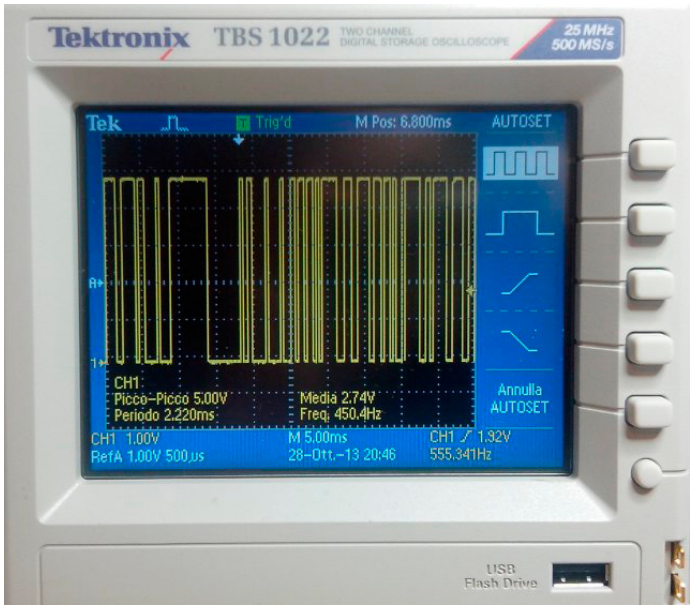
Simpatico, non trovate?

Utilizzando la sonda dell'oscilloscopio sul pin d'uscita è stato possibile scoprire com'erano fatte le forme d'onda.

E qui è cominciato il lavoro più impegnativo, ovvero la ricostruzione delle stesse.

La libreria che è stata impiegata, in mera teoria, sarebbe stata capace di garantirmi l'utilizzo di





semplici vettori all'interno dei quali inserire i dati corrispondenti ai bit ma con questa configurazione di dati grezzi non è stato possibile. Lascio ai lettori il compito di verificarne la funzionalità.

Per quanto riguarda il supporto, la libreria è in grado di gestire diversi protocolli di comunicazione tra cui quelli proprietari di aziende come **NEC** e **Panasonic** ma anche **RC5** ed **RC6**. Esistono diverse funzioni definite all'interno della libreria che permettono alla stessa di funzionare in maniera egregia. Ciò nondimeno, dal momento che il televisore a disposizione rientrava in queste casistiche, non ho potuto provarlo direttamente ma consiglio a voi di farlo.

IL CODICE

A questo punto tutto ciò che manca è la stesura del codice, sapendo che per la lettura del livello del volume sarà sufficiente scrivere un comando del tipo:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int livello = analogRead(A0);
  Serial.println(livello); // Lavoro con i valori assoluti
  float tensione = livello * (5.0 / 1023.0);
  Serial.println(tensione); // Corrispondente livello di tensione
  delay(500);
}
```

e che poco altro sarà davvero indispensabile per la funzionalità completa del programma (lo trovate in allegato).

Naturalmente il codice proposto è certamente perfettibile. Bisogna ottimizzarlo e sicuramente è indispensabile costruire le forme d'onda in maniera diversa, lavorando con la libreria in maniera tale da renderla compatibile con tanti formati diversi, anche quelli non standard. In questo modo la si può "universalizzare" ed utilizzare in qualsiasi altro caso.

Vale la pena di osservare che la soglia di rumorosità viene impostata ad un valore compreso fra 0 e 1023 dal momento che l'ADC disponibile su Arduino è a 10 bit.

CONCLUSIONI

Bene,siamo in chiusura. Avete visto che tutto sommato è stato abbastanza semplice realizzare questo progetto. Lo schema di funzionamen-

to è piuttosto elementare, l'hardware in gioco non è particolarmente costoso e le configurazioni sono, tutto sommato, standard. La parte più interessante sulla quale lavorare è stata l'acquisizione dei segnali, che ha portato via diverso tempo dal momento che, nel mio caso specifico, il protocollo si è rivelato piuttosto particolare.
Ad ogni modo, esistono certamente diversi margini di miglioramento per questo progetto che possono renderlo, in definitiva, molto più completo.
Sono a disposizione nei commenti per rispondere ad ogni eventuale domanda ed eventualmente guidarvi nella realizzazione di questo progetto.

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-abbassa-volume-degli-spot-tv-fai-da-te

Stazione Meteo Online con Arduino

di Giovanni Lorenzini

In commercio esistono molti dispositivi in grado di acquisire (ma non sempre salvare) i valori climatici, ma spesso non posseggono i requisiti dei quali necessitiamo, o in altri casi sono semplicemente troppo costosi.
Questo è il primo motivo che mi ha spinto a realizzare la mia piccola stazione meteo.
Inoltre l'idea di creare qualcosa di funzionale e meravigliosamente versatile era troppo allettante.

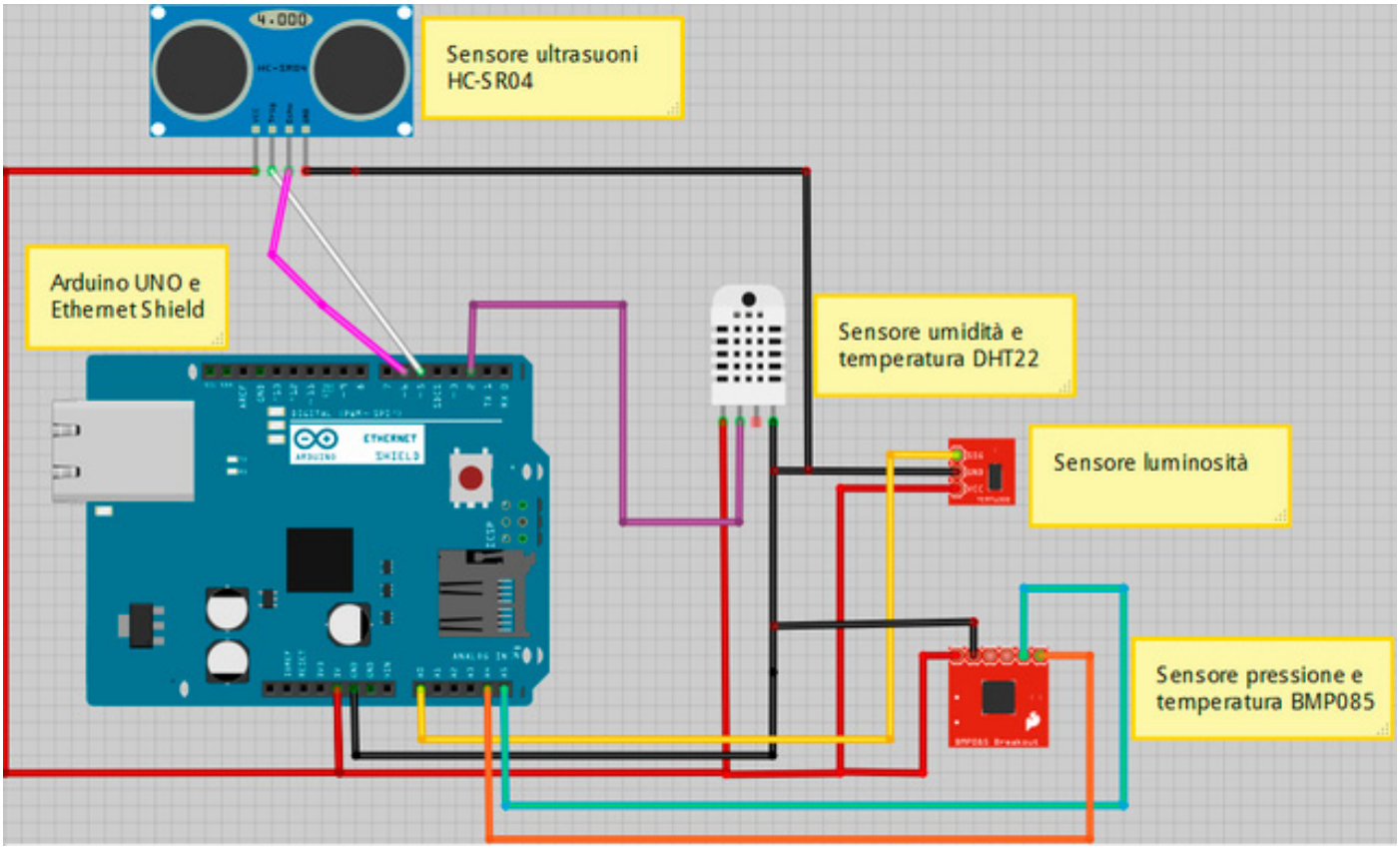
I compiti principali che svolge questa stazione meteo sono:
- Acquisire dati dai sensori
- Preparare i dati per essere memorizzati
- Ogni tot secondi, memorizzare i dati su un server online

I componenti necessari per realizzare questo progetto sono:
- Arduino (io ho utilizzato la board Arduino Uno, ma può essere utilizzata qualsiasi altra piattaforma basata su Arduino)
- Ethernet Shield
- Sensore di temperatura e umidità DHT-22
- Sensore di pressione BMP085
- Sensore di luminosità MF DIY LM358
- Sensore ad ultrasuoni HC-SR04
- Cavetteria varia

Quindi, i parametri che verranno gestiti sono:
- Temperatura
- Umidità
- Pressione atmosferica
- Luminosità dell'ambiente
- Altezza della neve

Per chi non lo sapesse, Arduino è una piattaforma Hardware e Software con cui è possibile imparare i principi fondamentali dell'elettronica e della programmazione.
Questa fantastica board si basa su un circuito stampato che integra un microcontrollore con pin connesso alle porte I/O, un regolatore di tensione ed un'interfaccia USB che permette la comunicazione con il computer per la programmazione del microcontrollore.

Arduino può essere utilizzato per moltissimi progetti, come ad esempio costruire robot, rilevare dati da sensori e svolgere delle azioni correlate, gestire dei servo motori, e come direbbe Ivan Scordato, "L'unico limite è la propria fantasia".
Il sensore di temperatura DHT22 misura la temperatura nel range -40 °C - +80 °C con un accuratezza di 0.5 °C. Misura l'umidità da 0% a 100% con un accuratezza da 2% a 5%. È un sensore lento, che bisogna leggere massimo ogni due secondi, ma da' buoni risultati.



Il sensore di pressione barometrica BMP085 permette di ottenere misurazioni nel range 300 - 1100hPa con un'accuratezza fino a 0.003hPa, è basato sulla tecnologia piezo-resistiva che fornisce grande accuratezza, robustezza e stabilità nel tempo. Misura la temperatura ambientale, di cui farò una media con la temperatura rilevata dal sensore DHT22 per ottenere una maggiore precisione.

Il sensore ad ultrasuoni HC-SR04 è una scheda compatta per la rilevazione di distanza misurata con il sistema a ultrasuoni.

Funziona inviando un segnale ad ultrasuoni e aspettando il suo ritorno (che avviene se il raggio ha rimbalzato su un ostacolo); a quel punto, sarà letto dallo stesso sensore. È capace di leggere una distanza variabile dai 2 ai 400 cm, con un margine di errore di 0,3 mm.

Per interfacciarsi con Arduino basterà utilizzare i suoi 4 pin: VCC, GND, il pin Trig e il pin Echo. Fornendo sul pin Trig un impulso positivo di circa 10 uS il trasmettitore invierà un segnale a ultrasuoni che dopo aver rimbalzato sull'oggetto in esame viene captato dal ricevitore.

Invece sul pin Echo verrà restituito un segnale logico "alto" con durata proporzionale alla distanza.

La formula con cui avviene il calcolo della distanza è: Distanza(cm)=Echo(us)/58

Adesso vediamo come collegare tutti i componenti ad Arduino:

Il software che bisogna caricare su Arduino è il seguente:

```
//Stazione Meteo con Arduino 12/2013
//Ivan Scordato e Giovanni Lorenzini

//dichiaro il pin di entrata del sensore di luminosità in una variabile
int pinfotoresistenza = A0;

//includo la libreria del sensore DHT22 di temperatura e umidità...
//...e dichiaro a che pin è collegato
#include “DHT.h”
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

//includo le librerie wire e del sensore di temperatura e pressione BMP085
#include “Wire.h”
#include “Adafruit_BMP085.h”
Adafruit_BMP085 bmp;
//scl--> A5 sda-->A4

//dichiaro in due variabili i pin del sensore ad ultrasuoni HC-SR04 per misurare...
//...l’altezza della neve
const int TRIG_PIN = 5;
const int ECHO_PIN = 6;

//includo le librerie della ethernet shield
#include
#include

//variabili per collegarsi a internet
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};

IPAddress ip(192,168,50,100); //indirizzo IP disponibile sulla rete

IPAddress myDns(192,168,1,1); //tuo DNS

char server[] = “www.fabbricaopensource.altervista.org”; //sito web

EthernetClient client;

unsigned long lastConnectionTime = 0; //l’ultima volta che ti sei connesso...
//...al server in millisecondi
boolean lastConnected = false;
const unsigned long postingInterval = 60L*1000L; //la L è necessaria per...
//...usare i numeri di tipo long

//dichiaro la stringa dove scrivo i dati per caricarli sul web
String strURL = “”;

void setup()
{
```

```
  //inizializzo i sensori sensori
  dht.begin(); //inizializzo il sensore di umidità
  bmp.begin(); //inizializzo il sensore di pressione
  pinMode(TRIG_PIN, OUTPUT); //dichiaro come output il pin trig
  pinMode(ECHO_PIN, INPUT); //dichiaro come input il pin echo

  Serial.begin(9600); //stabilisco una connessione con il pc

  delay(1000); //attendo un secondo

  Ethernet.begin(mac, ip, myDns);

  //invio al pc il mio IP
  Serial.print("My IP address: ");
  Serial.println(Ethernet.localIP());

}

void loop()
{

   if (client.available()) {
   char c = client.read();
   Serial.print(c);
  }

   if (!client.connected() && lastConnected) {
   Serial.println();
   Serial.println("Disconnessione...");
   client.stop();
  }

   if(!client.connected() && (millis() - lastConnectionTime > postingInterval)) {
   aggiornoDati(); //carico i dati sul server
  }

  lastConnected = client.connected();

//calcolo il punto di rugiada
double puntoDiRugiada(double temperatura, double umidita)
{
  double a = 17.271;
  double b = 237.7;
  double temp = (a * temperatura) / (b + temperatura) + log(umidita/100);
  double Td = (b * temp) / (a - temp);
  return Td;
}

void aggiornoDati() //carico i dati sul server
{
  //leggo la luminosità dal pin analogico A0
```

```
  int luminosita = map(analogRead(pinfotoresistenza), 0, 1023, 0, 100); //eseguo un...
//...procedimento che riduce il range da 0-1023 a 0-100...
//...per esprimere la luminosità in percentuale

  //leggo l'umidità e la scrivo in una variabile
  float umidita = dht.readHumidity();

  //leggo la pressione e la scrivo in una variabile
  float pressione = bmp.readPressure();

  //leggo la temperatura da due sensori, faccio la media e la scrivo in una variabile
  float temperatura = (bmp.readTemperature() + dht.readTemperature());
  temperatura = (temperatura / 2.0);

  //misuro l'altezza della neve
  float neve, durata; //dichiaro la variabile neve e la variabile durata
  float cmPerMicrosecondi = 0.0331 + ( 0.000062 * temperatura); //calcolo i cm/ms...
//...del suono in base alla temperatura
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10); //faccio un impulso di dieci microsecondi sul pin trig
  digitalWrite(TRIG_PIN, LOW);
  durata = pulseIn(ECHO_PIN,HIGH); //mi metto in ascolto sul pin eco e calcolo la...
//...durata dell'impulso
  neve = (durata*cmPerMicrosecondi/2.0); //calcolo la distanza con la formula...
//...durata*(cm/ms)/2, diviso due perchè il suo va, rimbalza contro...
//...un oggetto e ritorna,...
//...quindi compie due volte il tragitto
  float menoNeve = 200.0 - neve;

  if (client.connect(server, 80))
  {
    Serial.println("Connessione...");

    //creo l'url utilizzando una stringa
    strURL ="GET  /meteo/salva.php?temp="; //url
    strURL+=temperatura;
    strURL+="&umi=";
    strURL+=umidita;
    strURL+="&pres=";
    strURL+=pressione;
    strURL+="&lum=";
    strURL+=luminosita;
    strURL+="&neve=";
    strURL+=menoNeve;
    strURL+="&rug=";
    strURL+=puntoDiRugiada(temperatura, umidita);
    strURL+=" HTTP/1.1";

    //invio la richiesta al server
```

```
    client.println(strURL);
    client.println("Host: www.fabbricaopensource.altervista.org"); //sito web
    client.println("User-Agent: arduino-ethernet");
    client.println("Connection: close");
    client.println();

    lastConnectionTime = millis();

    delay(1000);
    Serial.println(strURL);
  }
  else
  {
    Serial.println("Errore Connessione");
    Serial.println("Disconnessione...");
    client.stop();
  }
}
```

Per salvare i nostri dati su un database MySQL, abbiamo bisogno di un dominio e di un database.

I file che dovranno essere caricati sul server sono due: **salva.php** e **visualizza.php**.

Il primo file riceve i dati acquisiti da Arduino tramite metodo GET, **mentre l'altro** sarà la pagina dove possiamo vedere i dati che sono stati acquisiti.

Tutto il codice è commentato e quindi (spero) di facile comprensione.

--- FILE salva.php

```
PHP
// dichiaro le variabili che ricevo da arduino
$temperatura = $_GET['temp'];
$umidita = $_GET['umi'];
$pressione = $_GET['pres'];
$luminosita = $_GET['lum'];
$neve = $_GET['neve'];
$rugiada = $_GET['rug'];

$tmsp = strtotime("now");
$date = date('d m Y',$tmsp);

// effettuo la connessione al mysql
$host = "host";
$database = "database";
$password = "password";
$link = mysql_connect($host, $database, $password);
if (!$link) {
   die('non connesso : ' . mysql_error());
}

// effettuo la connessione al database mysql
$db_selected = mysql_select_db('NOME_DATABASE', $link);
if (!$db_selected) {
   die ('Non è possibile connettersi al database : ' .
mysql_error());
}

// seleziono la tabella nella quale memorizzare i dati
$query = mysql_query("SELECT * FROM dati");
 $fetch = mysql_fetch_array($query);
// inserisco i valori
if (mysql_query("INSERT INTO
//unite il codice tra i ...
dati(temperatura,umidita,pressione,luminosita,...
...neve,rugiada,data)...
...VALUES('$temperatura','$umidita','$pressione',...
...$luminosita','$neve','$rugiada','$data')...
... ")or die(mysql_error()))
 { // se tutto è andato a buon fine...
        echo "OK";
 }
```

```
else{
echo “ERROR”;

}

/PHP
```

--- FILE visualizza.php

```
PHP
// effettuo la connessione al mysql
$host = “host”;
$database = “database”;
$password = “password”;
$link = mysql_connect($host, $database, $password);
if (!$link) {
   die(‘non connesso : ‘ . mysql_error());
}

// effettuo la connessione al database mysql
$db_selected = mysql_select_db(‘my_fabbricaopensource’, $link);
if (!$db_selected) {
   die (‘Non è possibile connettersi al database : ‘ . mysql_error());
}
/PHP
//fate tasto destro ispezione elemento per vedere il codice della tabella



PHP
$query = mysql_query(“SELECT * FROM dati”);
while($row = mysql_fetch_array($query)){
  $data = $row[‘data’];
  $temperatura = $row[‘temperatura’];
  $umidita = $row[‘umidita’];
  $luminosita= $row[‘luminosita’];
  $pressione = $row[‘pressione’];
  $neve = $row[‘neve’];
  $rugiada = $row[‘rugiada’];

     echo “”;
}
/PHP
```

```
Data: Temperatura: Umidità: Luminosità Pressione: Altezza della neve: Punto di rugiada:
$data $temperatura $umidita $luminosita $pressione $neve $rugiada
```

Quello che è stato appena presentato, è il codice che dovrà essere caricato sul server, mentre **per creare la tabella nel database mysql** potete utilizzare la seguente query:

```
CREATE TABLE `NOME_DATABASE`.`meteo` (
`id` INT NOT NULL ,
`temperatura` VARCHAR( 100 ) NOT NULL ,
`umidita` VARCHAR( 100 ) NOT NULL ,
`pressione` VARCHAR( 100 ) NOT NULL ,
`luminosita` VARCHAR( 100 ) NOT NULL ,
`neve` VARCHAR( 100 ) NOT NULL ,
`rugiada` VARCHAR( 100 ) NOT NULL ,
`data` DATE NOT NULL ,
PRIMARY KEY ( `id` )
) ENGINE = MYISAM
```

Considerazioni e osservazioni

Di questa stazione meteorologica ne ho realizzato due esemplari.

Una l'ho posizionata presso la mia abitazione in montagna, mentre l'altra è stata impiantata da un mio amico in valle, dove abita. In questo modo ad entrambi è possibile conoscere in tempo reale **i dati di due posti diversi**.

Quello che ho presentato è il codice generico. Infatti, nel caso in cui vogliate utilizzare più di una stazione meteo, dovrete modificare il codice aggiungendo un valore identificativo, altrimenti i dati che saranno salvati sul database verranno confusi tra loro.

Adesso, potrete leggere quì di seguito un piccolo approfondimento sul Meteo di Sebastiano Carpentari (il mio amico che possiede la seconda stazione meteo)

LA VELOCITÀ DEL SUONO E LA TEMPERATURA

In natura le onde possono essere di vario tipo: onde sismiche, elettromagnetiche, sonore, gravitazionali. Le onde di per se non possono crearsi se non immerse in un mezzo il quale permette loro di muoversi tramite forza elastica che scaturisce dalla deformazione del mezzo stesso (le onde elettromagnetiche e a livello teorico quelle gravitazionali possono diffondersi anche nel vuoto). Per concretizzare quando detto fin qui, immaginiamo di lanciare un sasso in uno specchio d'acqua. Il risultato è noto a tutti: si verranno a formare delle circonferenze concentriche in costante allontanamento dal punto di partenza. Il mezzo che permette alle onde di formarsi è proprio l'acqua. Lo stesso discorso lo possiamo fare per l'aria, quand'essa viene colpita da una vibrazione emessa da una sorgente sonora. Le caratteristiche classiche di un suono sono tre: intensità (energia dell'onda), altezza (tonalità dell'onda, anche chiamata frequenza o armonica fondamentale) e il timbro (composizione in termini di frequenze armoniche superiori dell'onda).

Ma a che velocità si propaga un'onda sonora nei vari mezzi presenti in natura (noi terremo presente l'aria, l'acqua e l'acciaio, che funge da solido)?

La velocità di propagazione di un'onda sonora dipende fondamentalmente dal mezzo che deve attraversare e non dalle caratteristiche del suono. È stato ormai dimostrato il fatto che la velocità delle onde aumenta all'aumentare della temperatura e della densità del mezzo e per quando

riguarda quelli presi in esame, essa è:

MEZZO	VELOCITA' SPERIMENTALE
ACQUA a 20°C	1480 m/s
ACCIAIO a 20°C	5980 m/s

Tabella 1 Velocità del suono in alcuni mezzi campione

Ma concentriamoci un momento sulla velocità delle onde sonore in base alla temperatura. L'equazione che ci permette di venir a conoscenza della stessa è, rispettivamente per l'aria e per l'acqua, la seguente:

$$v_{aria} = 340^{m}/_{s} + 0{,}6 \cdot T$$

$$v_{acqua} = 340^{m}/_{s} + 4{,}2 \cdot T$$

Dove per 0,6 e 4,2 si intende tasso di variazione della velocità con la temperatura (la sua unità di misura è m/(s*K) .

ALCUNI PARAMETRI FONDAMENTALI NELLA METEOROLOGIA

I parametri che concorrono a una buona previsione meteorologica sono molti. Di seguito ne sono elencati solamente alcuni, rilevati anche dalla stazione meteo creata con Arduino:

- **Punto di rugiada**: questo parametro, conosciuto anche come "dew point", indica la temperatura alla quale bisognerebbe portare una massa d'aria unitaria, a pressione costante, affinché essa, raggiunga la saturazione (100% umidità relativa). La formula che ci permette di calcolarla è la seguente:

$$T_d = \sqrt[8]{\frac{H}{100}} \cdot [112 + (0{,}9 \cdot T)] + (0{,}1 \cdot T) - 112$$

- **Umidità relativa**: in meteorologia, l'umidità relativa ci indica in via percentuale la quantità di vapore contenuta nell'atmosfera e viene definita come il rapporto tra la densità del vapore contenuto in atmosfera e la densità del vapore saturo alla temperatura della miscela. Lo strumento che la rileva è chiamato igrometro.
- **Pressione**: per riassumere in breve il concetto di pressione atmosferica possiamo dire che essa è la pressione che viene esercitata dalla colonna d'aria su un qualsiasi punto della superficie terrestre. A livello del mare essa è pari a 1013,25 mBar, cioè 1Atm. Un accorgimento da tener presente quando si parla di pressione è il fatto che essa varia con la quota e con le perturbazione dell'equilibrio atmosferico come le aree di bassa pressione (o cicloni) oppure gli anticicloni.
- **Temperatura**: essa è il parametro meglio conosciuto dalla maggior parte delle persone. La temperatura serve a descrivere lo "stato termico" di un sistema e le unità di misura sono molte. Le più usate sono: Kelvin, Celsius e Fahrenheit.

Possibili modifiche e sviluppi futuri

Quella che è stata presentata in questo articolo, è una versione base che può essere usata in molti contesti.

E' possibile modificarla facilmente ed ampliare il suo funzionamento.

Ad esempio, è possibile aggiungere la memorizzazione dei dati acquisiti su una memory card

"micro sd" (L'ethernet shield è già predisposta) affinchè, nel caso in cui dovesse mancare la connessione internet, automaticamente i dati acquisiti vengono memorizzati sulla memoria locale.

Sul sito si potrebbe inoltre creare un grafico che mostra i dati che sono stati rilevati l'ultimo giorno e quelli rilevati nell'ultimo mese.

Ringraziamenti

Voglio ringraziare Ivan Scordato che mi ha aiutato a realizzare questo progetto, specialmente nella parte del codice PHP e nella scrittura dell'articolo stesso.

ALLEGATI

Stazione Meteo

Stazione Meteo con 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/stazione-meteo-online-con-arduino

Crepuscolare fai-da-te con Arduino

di Piero Boccadoro

Come tutti ormai ben sapete, e come molti di noi condividono, la filosofia dell'open source è alla base di ogni mia considerazione e siccome sono affascinato dalla potenza e dalla bellezza del concetto stesso di condivisione mi sono affacciato, con grande entusiasmo, al mondo di Arduino.

Un esame all'università ha rappresentato lo stimolo decisivo a fare l'acquisto che, finora, si sta rivelando la spesa più riuscita e stimolante che io abbia fatto.

Per il momento, però, voglio descrivere ciò che s è proposto come semplice obiettivo hobbistico, ovvero **creare un sensore di luminosità che pilotasse, almeno in miniatura, un impianto crepuscolare** da semplice neofita.

Vi racconterò, quindi, la genesi del progetto.

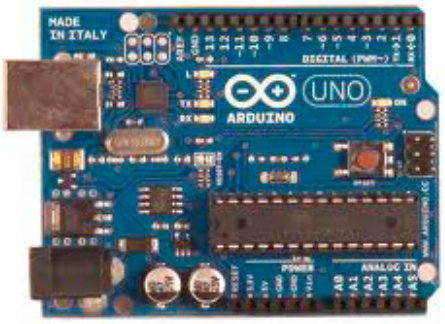
L'idea, quando ho deciso di fare l'acquisto, era quella di procurarmi l'occorrente per completare il dispositivo.

Lo Store di Arduino è molto molto ben fornito e tra le mille possibilità offerte, c'è lo Starter Kit, un piccolo set di strumenti e dispositivi molto semplice in cui ci sono resistenze, condensatori, pulsanti vari e poi semplici fili, reostati... Insomma, un po' di tutto.

Una volta ricevuto il pacco, arrivato in tempi molto celeri, ho iniziato a provare e sperimentare.

Avevo già letto che i codici di Arduino si chiamano sketch, che il linguaggio di programmazione era da imparare ex novo, che c'era una grande community molto attiva, che c'era tanto da poter imparare, che c'erano tanti progetti già svolti, spiegati e commentati, che su you-tube si trovano dimostrazioni di progetti funzionanti... insomma, mi ero documentato su quanto successo e quali risultati arduino avesse ottenuto.

Le informazioni raccolte avevano portato a farmi un'idea, poi confermata, che il modo giusto per spiegare i che si tratta è "Arduino is not for quitters". Grandi potenzialità, dunque, ma, come ogni progetto Open Source, alta richiesta di impegno da parte di chi lo usa... Come è giusto che sia!



Appena collegato Arduino al pc, ho avuto il mio primo problema: Arduino e il pc non comunicano! Ho iniziato compilando e facendo l'upload di uno sketch predefinito, uno degli esempi preconfezionati ma... Arduino si bloccava, il programma andava in crash e non ne venivo a capo. Mi sono collegato sul sito ed ho capito immediatamente che il problema era relativo all'uso delle porte COM, alla politica di assegnazione degli slot e che Arduino andava in conflitto con il Bluetooth del mio pc, il cui modulo di comunicazione è esterno e USB.

Così, una scheda di prototipazione esterna mi ha insegnato che la gestione degli indirizzi di comunicazione e dei protocolli non deve in alcun modo essere lasciata nella manie del sistema operativo e che certe cose devo tenerle sotto

occhio io in prima persona.
Appresa questa importante lezione, sono andato avanti ed ho iniziato ad occuparmi di capire come funzionassero gli ingressi, le uscite, l'alimentazione e dopo una ventina di minuti di studio sulla scheda, ho iniziato a fare i collegamenti per creare un semplice sensore di luce con l'LDR incluso nella scatola ed un LED.
Dopo aver creato il primo sketch e la prima configurazione funzionante, ho deciso di migliorare questo prototipo ed ho immaginato un sensore di luminosità con 3 gradi di "sensibilità" che accendesse altrettanti LED a seconda della luce presente.
Ed è proprio questo progetto che ora descriverò.
Sulla board Arduino, ci sono due file di pin ai lati. Guardando Arduino dal lato della porta USB, ci sono sulla sinistra 13 ingressi digitali (che generalmente vengono usati come uscite) e sulla destra quelli analogiche (che di solito fungono da ingressi).
Accanto ai pin analogici, poi, ce ne sono alcuni contrassegnati come "power", tra i quali c'è il punto a 5V e GND.

PER REALIZZARE IL CREPUSCOLARE SI PROCEDA COME SEGUE:

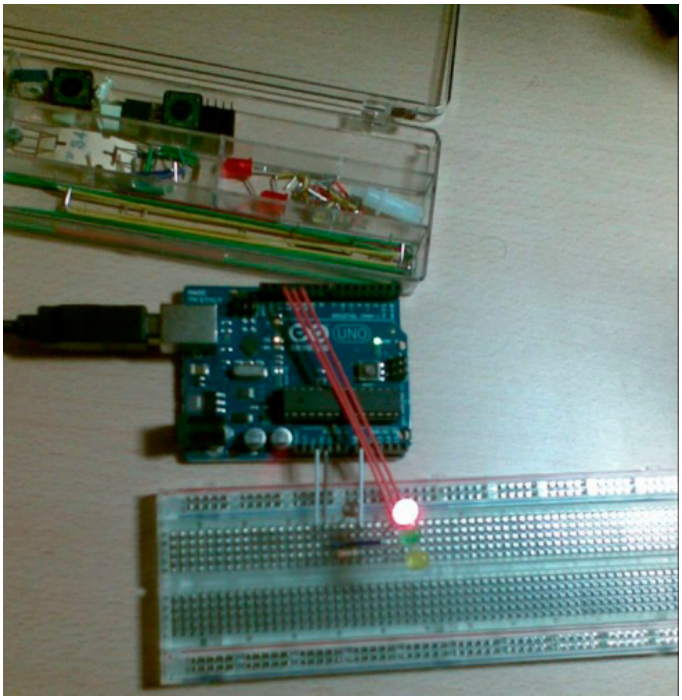
- prendendo due fili e portando i 5V e GND su un pad della breadboard compresa nello starter kit. (Il consiglio che do, quando si lavora su breadboard, è di usare sempre le "colonne" + e - per questo passaggio. Si arriva fin troppo presto a scoprire che l'ordine è fondamentale!!!)
- attaccando l'LDR ai 5V e connettendo l'altro pin ad un'altra fila, che va all'ingresso A0.
- la resistenza va connessa tra il secondo pin dell'LDR e GND (per questa esperienza io ho usato una resistenza da 22kOhm.)
- ora vanno collegati i LED. Vengono usati i connettori delle linee digitali numeri 13, 12 e 11. Ciascuno dei led va connesso tra linea di riferimento e GND.

Ora veniamo al codice, vero cuore del crepuscolare. Lo sketch è il seguente:

```
// Sensore di luminosità versione con 3 LED
int ledPin1 = 13; // Dichiaro il LED sull'uscita digitale 13
int ledPin2 = 12; // Dichiaro un secondo LED sull'uscita digitale 12
int ledPin3 = 11; // Dichiaro un terzo LED sull'uscita digitale 11
int sensore = 0; // Dichiaro l'LRD su ingresso analogico 0
int val; // dichiaro una variabile di appoggio
void setup()
{
Serial.begin(9600); //apro la porta seriale per mettere in comunicazione il circuito col pc Il baudrate è fisso a 9600 bits per secondo
pinMode(ledPin1, OUTPUT); // imposto il LED1 come uscita
pinMode(ledPin2, OUTPUT); // imposto il LED2 come uscita
pinMode(ledPin3, OUTPUT); // imposto il LED3 come uscita
}
void loop()
{
val = analogRead(sensore); // leggo i valori associati al sensore
Serial.println(val); // scrivo i valori letti del sensore nella variabile val
int range=val/4; // ricavo il valore di val modulo 4
delay(10); // effettuo la lettura ogni centesimo di secondo
switch (range) {
```

```
case 0: // tutti i LED sono accesi => luminosità nulla
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
digitalWrite(ledPin3, HIGH);
break;
case 1: //due LED su 3 sono accesi => luce scarsa
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
digitalWrite(ledPin3, LOW);
break;
case 2: //2 LED su 3 sono spenti => luce moderata
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
digitalWrite(ledPin3, LOW);
break;
case 3: //tutti i LED sono spenti => luce naturale
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
digitalWrite(ledPin3, LOW);
break;
}
}
```

Visto che il codice è commentato, non ho molto altro da dire.
Solo un piccolo appunto: il controllo sull'intensità luminosa scandito ogni centesimo di secondo non è affatto necessario! Lo si può comunque fare ogni decimo di secondo. Questo diminuisce la frequenza delle interrogazioni e libera il bus di comunicazione. Ovvio è che, in questa applicazione specifica questo non ha granché importanza ma se si utilizza la scheda per progetti più "corposi" anche questo aspetto va tenuto in conto.



PROGETTO CREPUSCOLARE, IN CONCLUSIONE:

Arduino è una scheda semplice, efficiente, funzionale e divertente. Rende l'elettronica e la programmazione appetibili e fruibili anche a chi non ha competenze specifiche e la sua versatilità è garanzia di efficienza.
Il costo contenuto, la facilità di interfacciamento e la varietà delle applicazioni a cui può esser riferita, fanno di questa scheda un utilissimo supporto.

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/crepuscolare-fai-da-te-con-arduino

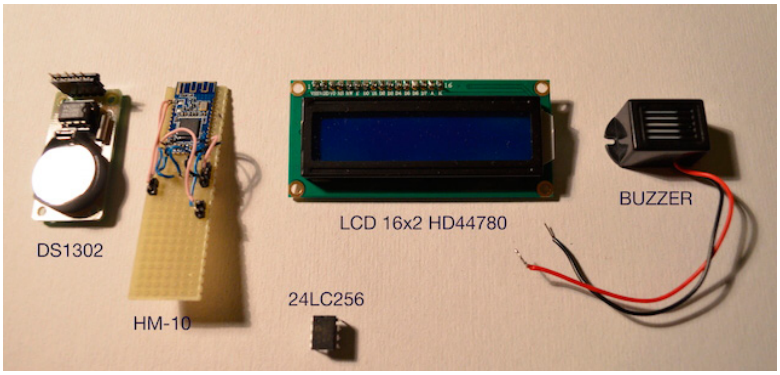
Realizziamo una Smart Sveglia Bluetooth con Arduino

di Luigi D'Acunto

Premesso che la 'mission' di questo articolo non è la realizzazione di un prototipo di allarme bluetooth, ma il 'proof of concept', con cui spero di aprire la mente dei lettori, iniziamo a parlare di questo progetto entrando subito nel vivo ed elencando i componenti utilizzati nella Smart Sveglia.

COMPONENTI UTILIZZATI

- **Arduino UNO;**
- **DS1302 (RTC);**
- **24LC256 (EEPROM);**
- **CR2032 (BATTERIA 3V);**
- **HM-10 (MODULO BLUETOOTH LE);**
- **Display LCD 16x2;**
- **Trimmer 10k;**
- **Quarzo 32.768 kHz;**
- **Buzzer x1;**
- **Resistori: 2.2kOhm x1, 470Ohm x1, 47Ohm x1;**
- **LED x1;**



IL REAL TIME CLOCK

Ogni nostra azione quotidiana è scandita dall'inesorabile incedere del tempo.

Questa necessità si riflette allo stesso modo nei circuiti digitali che eseguono comandi a ritmo di clock. L'oscillatore che regola il tempo di esecuzione dei comandi macchina dei microcontrollori è spesso esterno per ragioni di stabilità e dimensioni ed è in grado di scandire intervalli temporali dell'ordine dei nanosecondi con una eccellente risoluzione.

Tuttavia, in assenza di un riferimento temporale che leghi l'intervallo temporale scandito dall'oscillatore al tempo coordinato universale (UTC), è impossibile sfruttare un oscillatore come se fosse un vero e proprio orologio.

In questo contesto il **real-time clock** trova la sua ragion d'essere in quanto capace di portare il tempo anche quando il dispositivo all'interno del quale è integrato è spento.

Tutto ciò è possibile sfruttando un doppio circuito di alimentazione costituito dall'alimentazione primaria e dalla **batteria tampone** in grado di alimentare l'integrato, che consuma pochi nanoWatt, per diversi anni.

Solitamente la batteria tampone è una piccola cella al litio ma sistemi più recenti tendono ad utilizzare dei supercapacitori, in quanto questi ultimi possono essere integrati nel processo di assemblaggio del PCB. Il primo dispositivo commerciale dotato di RTC fu il PC/AT di IBM nel 1984 che adottava il chip MC146818 di Motorola. Attualmente i chip più utilizzati allo scopo tra gli hobbisti sono sicuramente i **DS1302** e i **DS1307** (in vendita su Conrad).

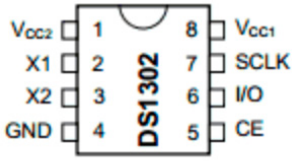
Non è un caso che questi microcontrollori richiedano un quarzo esterno con frequenza di 32.768

kHz, che, essendo una potenza di 2 (2^15), è più semplice da manipolare con dei contatori digitali.

Nella Smart Sveglia ho utilizzato un **DS1302**, già dotato di quarzo da 32.768kHz e batteria al litio **CR2023** da 3V.

image credits: playzone.ch

I pin digitali di Arduino 5, 6, 7 sono stati collegati ai pin **SLCK** (Serial Clock), **I/O** e **CE** (Chip Enable) per implementare l'interfaccia **3 Wire**.



DIP (300 mils)

image credits: datasheets

Se non si utilizzano le linee digitali di Arduino, già dotate di resistori di pull-up, è necessario aggiungerli in modo da preservare l'integrità della trasmissione.

A questo punto, per testare il chip, è stata utilizzata una libreria per Arduino aggiunta su **GitHub** da Matt Sparks che crea un oggetto **DS1302** di cui possono essere richiamati metodi di lettura o scrittura.

```
#include < DS1302.h >
#include < stdio.h >

#define kCePin 5 // DS1302 - Chip Enable
#define kIoPin 6 // DS1302 - Input/Output
#define kSclkPin 7 // DS1302 - Serial Clock
DS1302 rtc(kCePin, kIoPin, kSclkPin);

void printTime()
{
Time t = rtc.time();
char buf[50];
snprintf(buf, sizeof(buf), "%04d-%02d-%02d
%02d:%02d:%02d",t.yr, t.mon,
t.date, t.hr, t.min, t.sec);
Serial.println(buf);
}

void setup()
{
Serial.begin(9600);
rtc.writeProtect(false); //Abilita la scrittura sul chip
dando un
valore LOW al pin CE
rtc.halt(false); // Mette in sleep in clock se posto a
TRUE
Time t(2014, 5, 5, 1, 38, 50, Time::kSunday);
rtc.time(t);
}

void loop()
{
printTime();
delay(1000);
}
```

Con questo semplice sketch è possibile settare l'ora e il giorno di riferimento per il DS1302, monitorare il tempo e visualizzarlo ogni secondo sulla porta seriale.

La chiave di volta di questo sketch sono le righe

```
Time t(2014, 5, 5, 1, 38, 50, Time::kSunday);
rtc.time(t);
```

con le quali vengono impostate ora e data corrente sull'RTC. Chiaramente questa operazione andrebbe eseguita una tantum e non ogni qualvolta viene programmato il microcontrollore.

E' pertanto opportuno impostare un flag di controllo sul primo utilizzo del controllore in cui impostare l'orario corrente sul DS1302.

L'orario attuale viene letto attraverso il metodo **rtc.time()** richiamato nella funzione **printTime()**.

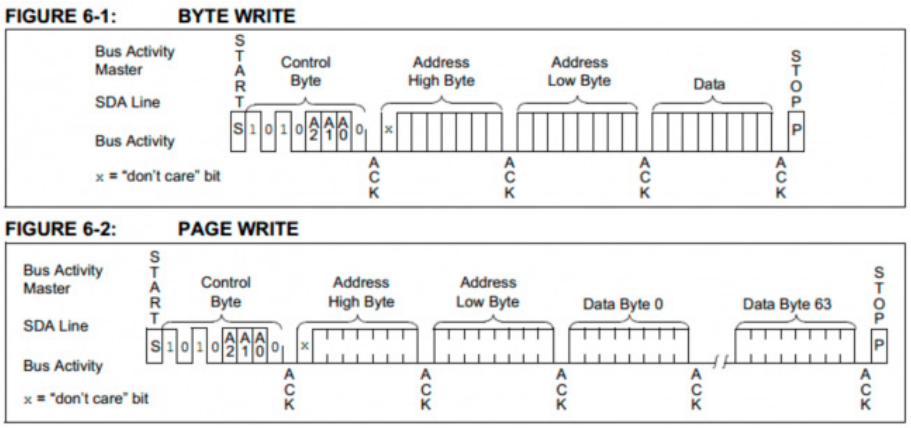


image credits: Microchip Datasheet

LA EEPROM

Per la realizzazione della Smart Sveglia una **EEPROM esterna** è del tutto **NON necessaria**. Se ne potrebbe tranquillamente fare a meno ed utilizzare quella interna ad Arduino che ha una capacità di **512 bytes**; tuttavia, ho ritenuto opportuno utilizzarne una esterna per completezza di trattazione, in quanto la ridotta memoria può risultare in un limite alle potenzialità della sveglia, ad esempio se si vogliono memorizzare più sveglie o informazioni.

Una **EEPROM** (*Electrically Erasable Programmable Read Only Memory*) è una memoria di tipo non volatile che può essere scritta oppure letta.

Solitamente, per la sua capacità ridotta, è utilizzata per memorizzare impostazioni di configurazione. E' sconsigliato utilizzarla come se fosse una RAM per via del limitato tempo di vita medio legato al numero di cicli di scrittura (tipicamente 100000).

Il componente che ho scelto per la Smart Sveglia è il chip **24LC256** prodotto da **Microchip**, realizzato in tecnologia CMOS ed in vendita su Conrad.

Il chip in questione è in grado di memorizzare fino a **32 kBytes (64 volte in più di quella interna ad Arduino)**, ha tempi di latenza massimi in scrittura pari a **5 ms** (un dettaglio assolutamente non trascurabile) e la sua memoria è **divisa in pagine da 64 bytes** ciascuna (ovvero con una gestione ottimizzata di codice e risorse si potrebbe pensare di scrivere una intera pagina con un solo ciclo riducendo, così, i tempi di latenza).

Il datasheet garantisce inoltre il componente per oltre un milione di cicli di scrittura e la corretta conservazione dei dati per oltre 200 anni!

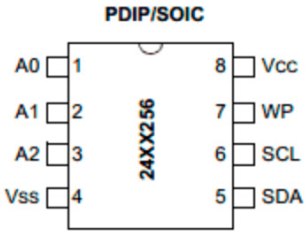


image credits: Microchip Datasheet

Il protocollo per accedere alla EEPROM è l'**I2C** (Inter Integrated Circuit) supportato nativamente da Arduino, che **richiede soltanto 2 pin: SCL** (*Serial Clock Line*), poiché è un bus sincrono, e **SDA** (*Serial DAta line*).

I pin **A0**, **A1** e **A2** servono ad **impostare l'indirizzo**

```
#include < Wire.h >

void writeEEPROM(int deviceaddress, unsigned int eeaddress, byte data )
{
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8)); // MSB
  Wire.send((int)(eeaddress & 0xFF)); // LSB
  Wire.send(data);
  Wire.endTransmission();
  delay(5);
}

byte readEEPROM(int deviceaddress, unsigned int eeaddress )
{
  byte rdata = 0xFF;
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8)); // MSB
  Wire.send((int)(eeaddress & 0xFF)); // LSB
  Wire.endTransmission();
  Wire.requestFrom(deviceaddress,1);
  if (Wire.available()) rdata = Wire.receive();
  return rdata;
}
```

del chip all'interno del bus.

L'**I2C** infatti ha **7 bit di indirizzo** e può pertanto supportare **fino a 128 nodi**.

Vi sono, però, 16 indirizzi protetti e non accessibili, pertanto il numero massimo di dispositivi che possono essere connessi sul bus è di 112.

Tuttavia, il **24LC256** possiede **soltanto 3 bit di indirizzamento**, infatti gli altri 4 bit (quelli più significativi) **vengono considerati** (di default) **pari a 1010**. Così facendo, il **numero massimo di EEPROM** che possono essere aggiunte al bus **viene limitato a 2^3=8**. Il pin **WP** (*Write Protection*) se posto a valore logico alto, in questo caso **Vcc**, impedisce la scrittura, preservando i dati precedentemente memorizzati, e, pertanto, nell'esempio in questione è stato posto a massa. Analogamente sono stati posti a massa i pin **A0**, **A1**, **A2**, assegnando, così, al chip **indirizzo 1010000 sul bus I2C**.

I pin **SCL** e **SDA** sono stati collegati ai **pin analogici** di **Arduino 4 e 5**, come suggerito dalla documentazione della libreria Wire.h di Arduino. In questo modo si può evitare di inserire resistori di pull-up poiché già presenti sulla linea analogica di Arduino e abilitati attraverso la libreria suddetta. Un'eccellente sketch per controllare la EEPROM si può trovare sul forum di Arduino utilizzando la libreria EEPROM.h, tuttavia ho sfruttato un altro riferimento che utilizza la libreria Wire.h.

Attraverso queste due funzioni è possibile leggere o scrivere sulla EEPROM caratterizzata da un indirizzo I2C univoco '*deviceaddress*' un byte 'data' nella locazione di memoria della EEPROM '*eeaddress*'.

E' stato inserito un **delay di 5ms** nella funzione **writeEEPROM** poiché questo è il tempo di latenza massima garantito dal datasheet per la

scrittura in memoria.

```
#define eeprom1 0x50    //INDIRIZZO DEL
24LC256
void setup(void)
{
  Serial.begin(9600);
  Wire.begin();
  unsigned int address = 0;
  writeEEPROM(eeprom1, address, 'A');
  Serial.print(readEEPROM(eeprom1, address),
DEC);
}
```

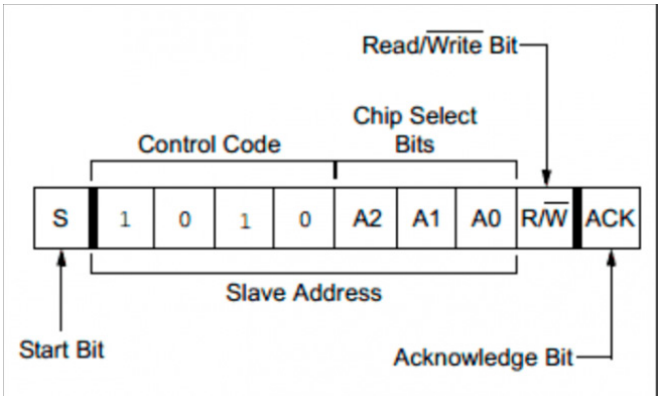


image credits: Microchip Datasheet

Nel codice mostrato sopra, il codice **HEX 0x50** corrisponde alla eeprom indirizzata con i bit **A0**, **A1**, **A2** a massa, infatti

1	0	1	0	**A2**	**A1**	**A0**
1	0	1	0	0	0	0

0*(2^0)+**0***(2^1)+**0***(2^2)+**0***(2^3)+**1***(2^4)+0*(2^5)+1*(2^6)=16+64=**80** in decimale che corrisponde a **50** in esadecimale.

A questo punto occorre definire anche l'indirizzo della locazione di memoria (la variabile '*address*') in cui si vuole andare a scrivere il byte da salvare in memoria (in questo caso la lettera '**A**'). Chiaramente essendo la EEPROM in questione da **32K l'indirizzo** che possiamo scegliere può variare **tra 0 e 31999**.

Con le funzioni **writeEEPROM** e **readEEPROM** si può **salvare/leggere** in memoria un **singolo byte** e questo è sicuramente poco pratico per manipolare delle stringhe.

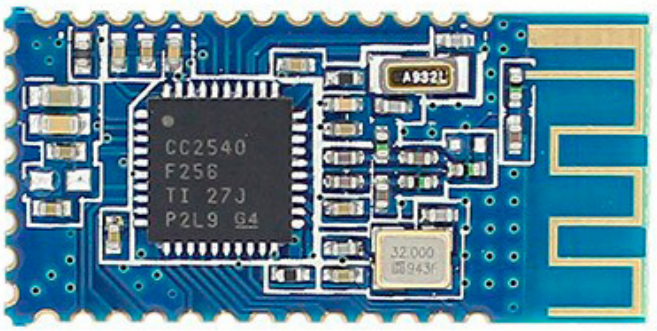
Si può pensare allora di scrivere delle **routine** per richiamare ciclicamente queste funzioni.

Quelle mostrate di seguito richiedono come parametri la dimensione delle aree di memoria da leggere (in numero di byte) e l'indirizzo da cui iniziare a leggere, oltre che quello della EEPROM sul bus I2C.

La routine **writeEEPROMstring**, tuttavia, **non è ottimizzata** per la scrittura in memoria a pagine e pertanto il **tempo di latenza** per la scrittura di una stringa è pari a **5ms*lunghezza_stringa**.

IL BLUETOOTH HM-10

Il modulo bluetooth **HM-10**, prodotto da *JNHuaMao Technology Company*, utilizza il chip di Texas Instruments **CC2540** e implementa lo stack protocollare del **Bluetooth 4.0 Low Energy** e, pertanto, si sposa bene con l'applicazione da Smart Sveglia per via del basso consumo energetico.



Nonostante questo chip sia dotato di **34 pin**, nel corso di quest'articolo mi concentrerò soltanto su quelli per il funzionamento minimale di questo dispositivo, ovvero i pin di alimentazione, quelli per l'**UART**, il pin di **RESET** e quello per il **LED** (utile in fase di debugging ma non necessario).

```
void writeEEPROMstring(int deviceAddress, unsigned int startingEepromAddress,
unsigned int memorySize,char *string)
{
for(int i=0;i < memorySize;i++)
writeEEPROM(deviceAddress,startingEepromAddress+i,string[i]);
}

String readEEPROMstring(int deviceAddress, unsigned int startingEepromAddress,
unsigned int memorySize)
{
char string[memorySize+1];
for(int i=0;i < memorySize;i++)
string[i]=charValue(readEEPROM(deviceAddress,startingEepromAddress+i));
}
string[memorySize]='\0';
return string;
}

char charValue(int integer)
{
return char(integer-0);
}
```

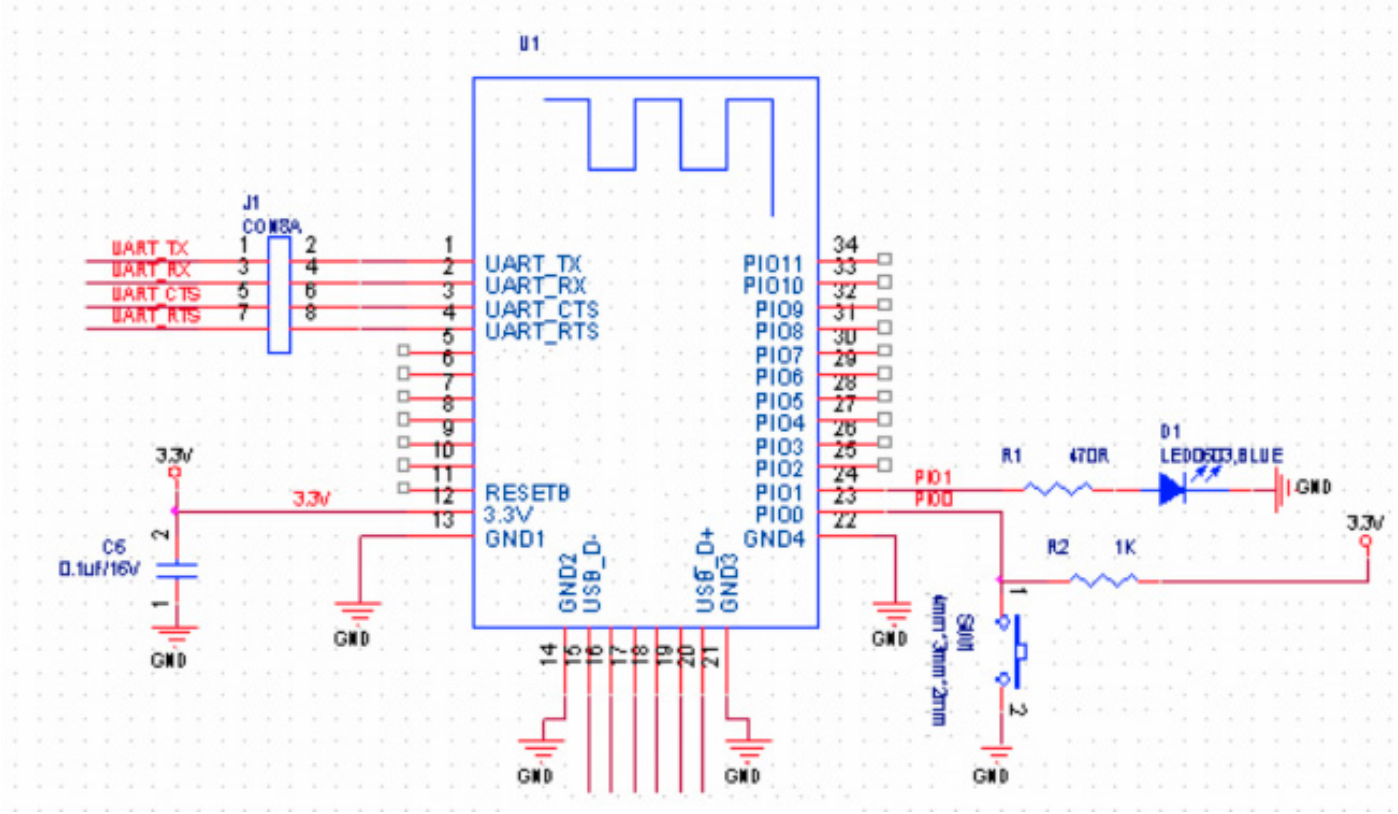


image credits: Microduino

Per l'**UART** (*Universal Asynchronous Receiver-Transmitter*) sono necessari, in questo esempio, soltanto i pin **UART_TX e UART_RX**, mentre i pin **UART_CTS e UART_RTS** possono essere lasciati floating.

Attenzione: l'HM-10, a differenza del resto della circuiteria trattata in questo articolo, DEVE essere alimentato a 3.3V.

In questo esempio ho collegato i **pin 1 e 2** dell'**HM-10** rispettivamente **ai pin 10 e 11 di Arduino**, ho omesso il condensatore anti-disturbi tra alimentazione e massa, connesso il **pin 24 a un LED** verde tramite un resistore da **47 Ohm** e il **pin 23 alla** tensione **3.3V di Arduino tramite un resistore da 3.3 kOhm, con in parallelo un interruttore di tipo Normally Open (NO)**.

Una volta effettuate le connessioni, il LED lampeggerà in modo continuato, per evidenziare che la **periferica HM-10** è in modalità *Advertising*, fino a quando un **Bluetooth Master** non effettua l'accoppiamento (*pairing*).

A questo punto il LED manterrà la sua luce fissa per l'intera durata della connessione al Master.

Per implementare la **UART** mantenendo le funzionalità della porta seriale per il debugging è consigliabile **non utilizzare i pin 0 e 1 di Arduino**, ma la **seriale emulata** (in questo esempio dai pin 10 e 11) dalla libreria SoftwareSerial.h.

```
#include < SoftwareSerial.h >

#define kRxSwSer 10 // RX Software Serial - Bluetooth Comm
#define kTxSwSer 11 // TX Software Serial - Bluetooth Comm

SoftwareSerial mySerial(kRxSwSer, kTxSwSer);

void setup()
{
Serial.begin(9600); //INIZIALIZZO LA SERIALE
while(!Serial){;}
mySerial.begin(9600); //INIZIALIZZO LA SERIALE EMULATA
}

void loop()
{
char buffer[100];
int k=0;
while (mySerial.available()){
buffer[k]=mySerial.read();
k++;
}
if(k!=0){
buffer[k]='\0';
}
Serial.print(buffer);
}
```

Con questo codice è possibile stampare a video, tramite la seriale, i dati che si ricevono sull'UART tramite la seriale emulata dai pin 10 e 11.

IL DISPLAY LCD 16X2

Ogni sveglia che si rispetti, oltre a svolgere la funzione di allarme, mostra all'utente l'orario, per questo ho deciso di inserire un display LCD alfanumerico nel progetto.

Il display utilizzato è diviso in **16 colonne e 2 righe (16x2)** ed è basato sul noto controller HITACHI HD44780.

Questo genere di display monta un'interfaccia di comunicazione standard a 16 pin e può essere comandato sia in modalità **8 bit** (utilizza 8 pin per la comunicazione) sia **4 bit** (utilizza 4 pin per la comunicazione ma è più lenta poichè ogni messaggio è diviso in due blocchi).

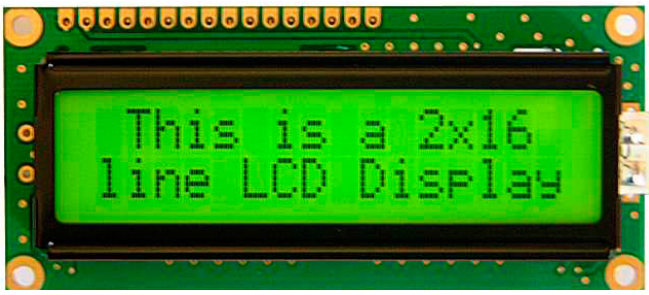


image credits: 3.bp

```
#include  < LiquidCrystal.h >

#define kRSLCD 13//LCD RS
#define kELCD   12//LCD E
#define kD4LCD 9 //LCD D4
#define kD5LCD 8 //LCD D5
#define kD6LCD 3 //LCD D6
#define kD7LCD 2 //LCD D7

LiquidCrystal lcd(kRSLCD, kELCD, kD4LCD, kD5LCD, kD6LCD, kD7LCD);

void setup()
{
lcd.begin(16, 2);
delay(1000);
lcd.setCursor(0, 0);
lcd.print("EOS DISPLAY");
}
```

Seguendo le indicazioni riportate sul datasheet risulta davvero semplice connettere il display ad un microcontrollore, in particolare, nel caso in esame, ho utilizzato la modalità di funzionamento a **4 bit**, lasciando i pin **D0**, **D1**, **D2**, **D3 floating**.

Il pin **RW** (*Read Write*) è stato collegato a massa poiché il display viene, in questo esempio, utilizzato soltanto in scrittura e non in lettura.

I pin **RS** (*Register Select*) ed **E** (*Enable*) del display sono stati collegati rispettivamente ai pin **13** e **12** di **Arduino**.

I pin **D4**, **D5**, **D6**, **D7** del display sono stati collegati rispettivamente ai pin **9**, **8**, **3**, **2** di Arduino.

Il pin **Vo** del display è usato per la **regolazione del contrasto** ed è stato connesso al terminale centrale di un **trimmer da 10 kOhm** posto **tra Vcc e GND**.

Infine, i pin A e K dell'LCD, utilizzati per la retroilluminazione, sono stati connessi rispettivamente a Vcc tramite un resistore da 470 Ohm e a GND.

Attenzione: Senza il resistore, la retroilluminazione del display si brucerebbe.

Questa è realizzata in tecnologia LED e, pertanto, fissa la tensione ai suoi capi, in questo caso a **4.2 V**.

Se **Vcc=5V**, allora, la corrente destinata alla retroilluminazione, per la **legge di Ohm**, è pari a **(5-4.2)/470= 1.8 mA** un valore sufficientemente lontano da quello massimo tollerato dai LED, di certo poco accurato, ma rappresenta una soluzione precauzionale da attuare qualora non si possegga il datasheet dell'LCD.

Il controller **HD44780** richiede una inizializzazione abbastanza complessa con tempi di latenza ben precisi, fortunatamente la libreria LiquidCrystal.h di Arduino svolge questo lavoro per noi.

In questo esempio, dopo aver creato un oggetto '*lcd*' della classe **LiquidCrystal** e averlo inizializzato come un display con 16 colonne e 2 righe, tramite il metodo **lcd.begin(16, 2)**, è possibile visualizzare il testo "*EOS DISPLAY*".

Il metodo richiamato nel codice **setCursor(x,y)**

```
//
//  BTAlarmViewController.h
//  coreBluetoothiPhone
//
//  Created by Luigi D'Acunto on 01/05/14.
//  Copyright (c) 2014 Luigi D'Acunto. All rights reserved.
//

#import < UIKit/UIKit.h >
#import < CoreBluetooth/CoreBluetooth.h >

@interface BTAlarmViewController : UIViewController
@property (weak, nonatomic) IBOutlet UIDatePicker *datePick;
@property (weak, nonatomic) IBOutlet UIButton *enableButton;
- (IBAction)enableAction:(id)sender;

@property (strong, nonatomic) CBCentralManager *myCentralManger;
@property (strong, nonatomic) CBPeripheral *myPeripheralBLE;
@property (strong, nonatomic) CBCharacteristic *myCharacteristic;
@property (strong,nonatomic)  NSString *deviceType;
@end
```

prevede due parametri **x** e **y** che indicano rispettivamente la riga e la colonna del display a partire dalle quali iniziare a scrivere il testo.

L'APP IOS

Partendo dal precedente articolo sul Bluetooth Low Energy e i dispositivi Apple, andiamo a modificare il codice già esistente, predisposto per il BC127, per connettere un iPhone al modulo HM-10.

Nella classe **BTViewController** và fatta una piccola modifica poiché la periferica HM-10 ha **un solo servizio** con **un'unica caratteristica** che, a differenza del BC127, prevede una **modalità di scrittura senza risposta** (*CBCharacteristicWriteWithoutResponse*).

Pertanto andiamo a sostituire *CBCharacteristicWriteWithResponse* con *CBCharacteristicWriteWithoutResponse*.

Il codice Xcode allegato all'articolo è già ottimizzato per l'HM-10.

Nello *storyboard* ho trasformato l'unico *ViewController* finora presente nel *rootViewController* di un *NavigationController* e aggiunto un *segue* con il nome *BTAlarmViewController* collegato al pulsante SVEGLIA.

Ho inserito quindi da interfaccia grafica un oggetto **UIDatePicker** e un oggetto **UIButton** cui ho associato il metodo **enableAction** nell'header di **BTAlarmViewController** riportato di seguito.

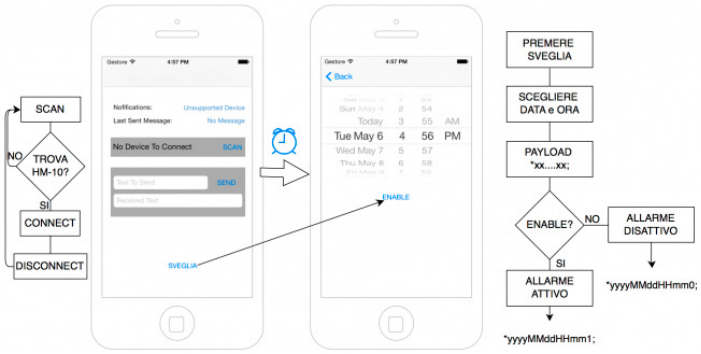
HEADER

Le proprietà relative al Bluetooth vengono importate dal **ViewController** trattato in precedenza.

Nel file di implementazione, invece, ho gestito la possibilità di abilitare/disabilitare l'allarme lavorando sul toggle del tasto ENABLE.

Ho aggiunto, inoltre, alla stringa che viene invia-

ta dall'iPhone un carattere di inizio (*) e fine (;) comando per poter discriminare dalla periferica bluetooth la lunghezza del messaggio in arrivo.



Come evidenziato dal mokup la stringa viene inviata all'HM-10 soltanto se viene premuto il pulsante ENABLE/DISABLE che, oltre a concatenare le stringhe di data e ora in formato yyyyMMddHHmm, aggiunge un flag booleano, che indica se l'allarme è attivo o meno, e il payload (*;).
La stringa finale inviata dal telefono all'HM-10 e, che dovrà essere elaborata da Arduino, risulta allora:

***yyyyMMddHHmmE;**

IMPLEMENTATION

```
//
//  BTAlarmViewController.m
//  coreBluetoothiPhone
//
//  Created by Luigi D'Acunto on 01/05/14.
//  Copyright (c) 2014 Luigi D'Acunto. All rights
reserved.
//

#import "BTAlarmViewController.h"

@interface BTAlarmViewController ()

@end

@implementation BTAlarmViewController
```

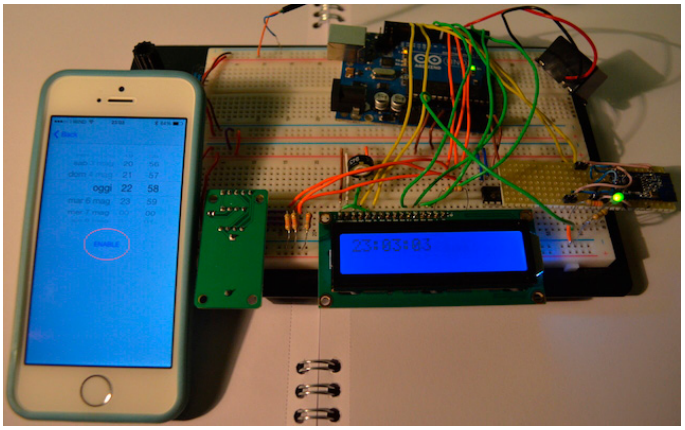
```
- (id)initWithNibName:(NSString *)nibNameOr-
Nil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil];
    if (self) {

    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    NSDate *currentTime = [NSDate date];
    NSDateFormatter *dateFormatter = [[NSDate-
Formatter alloc] init];
    [dateFormatter setDateFormat:@"ddMMyyyy
hh:mm"];
    NSString *min = [dateFormatter stringFromDa-
te: currentTime];

    NSString *max = @"22042024 12:00";
    NSDateFormatter *dateFormat = [[NSDateFor-
matter alloc] init];
    [dateFormat setDateFormat:@"ddMMyyyy
hh:mm"];
    NSDate *datemin = [dateFormat
dateFromString:min];
    NSDate *datemax = [dateFormat
dateFromString:max];
    self.datePick.maximumDate = datemax;
    self.datePick.minimumDate = datemin;
}

- (IBAction)enableAction:(id)sender {
    if([self.enableButton.titleLabel.text
isEqualToString:@"ENABLE"]){
        [self.enableButton setTitle:@"DISABLE"
forState:UIControlStateNormal];
        [self sendMessage:1];
    }
    else if([self.enableButton.titleLabel.text
isEqualToString:@"DISABLE"]){
        [self.enableButton setTitle:@"ENABLE"
forState:UIControlStateNormal];
        [self sendMessage:0];
    }
```

```
}
-(void)sendMessage:(NSUInteger) enable{

    NSDateFormatter *dateFormatter = [[NSDate-
Formatter alloc] init];
   [dateFormatter
setDateFormat:@"yyyyMMddHHmm"];
   NSString *dateString = [dateFormatter string-
FromDate: self.datePick.date];
   NSString *stringSS = [NSString stringWithFor
mat:@"*%@%d;",dateString,enable];
   NSLog(@"%@",stringSS);

   NSData* data = [stringSS dataUsingEncoding:N
SUTF8StringEncoding];
   if([self.deviceType isEqualToString:@"BC127"])
   {
      [self.myPeripheralBLE writeValue:data
forCharacteristic:self.myCharacteristic
type:CBCharacteristicWriteWithResponse];
      NSLog(@"Send String To BC127");
   }
   else if([self.deviceType
isEqualToString:@"HM10"])
   {
      [self.myPeripheralBLE writeValue:data
forCharacteristic:self.myCharacteristic
type:CBCharacteristicWriteWithoutResponse];
      NSLog(@"Send String To HM10");
   }
}
@end
```

LA SMART SVEGLIA

Mettendo insieme i vari frammenti di codice (più qualche funzione extra) e di hardware riportati in questo articolo e, aggiungendo un piccolo **buzzer piezoelettrico** tra il **pin 4** di Arduino e **GND**, si può procedere alla realizzazione della Smart Sveglia.

Il codice che ho scritto prevede la creazione di una struttura nominata alarmType



```
typedef struct
{
int year;
int mon;
int day;
int hh;
int mm;
int ss;
boolean enabled;
} alarmType;
```

che contiene le informazioni su un orario di allarme e un flag booleano che indica se la la sveglia è stata abilitata o meno.
Definisco poi indirizzi e locazioni di memoria della EEPROM e inizializzo una variabile di tipo alarmType.

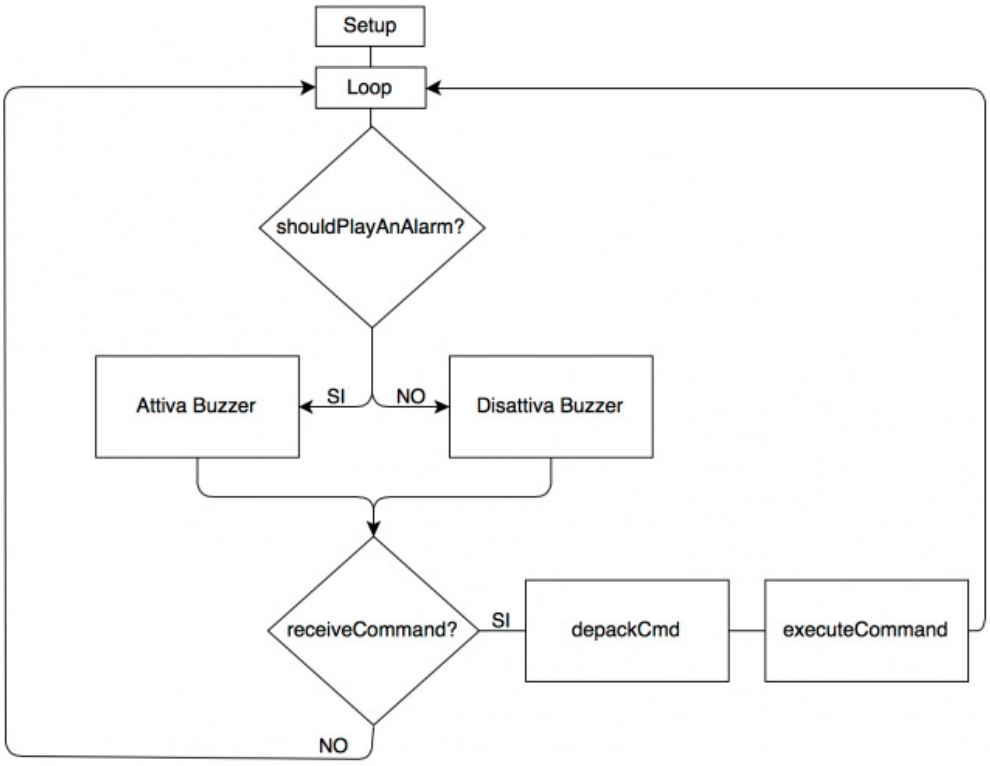
```
//INDIRIZZI LOCAZIONI MEMORIA EEPROM
#define ADD_FIRSTTIME 0
#define ADD_DATE 1
#define ADD_TIME 11

//DIMENSIONI LOCAZIONI MEMORIA EE-
PROM (aggiungere sempre +1!)
#define ADD_DATE_SIZE 11
#define ADD_TIME_SIZE 9

#define kBuzzer 4 // BUZZER

alarmType alarm;
```

```
..... INSERIRE IN SETUP .....
pinMode(kBuzzer, OUTPUT);
digitalWrite(kBuzzer, LOW);

if(readEEPROM(EEPROM1, ADD_FIRSTTIME)==1){ //NOT FIRST TIME
Serial.print("Not First Time");
readAlarmFromEEPROM();
}
else{ //FIRST TIME
Serial.print(" First Time");
writeEEPROM(EEPROM1, ADD_FIRSTTIME,1);
Time t(2014, 5, 1, 17, 56, 0, Time::kMonday);
rtc.time(t);
delay(200);
saveTestAlarmToEEPROM();
delay(200);
}
}
..... INSERIRE IN SETUP .....

void saveTestAlarmToEEPROM()
{
alarm.year = 2014;
alarm.mon = 4;
alarm.day = 30;
alarm.hh = 12;
alarm.mm = 46;
alarm.ss = 40;
alarm.enabled = 0;
alarmSet(alarm);
}

void alarmSet (alarmType al)
{
String dateString = int2string(al.year)+int2string(al.mon)+int2string(al.day);
String timeString = int2string(al.hh)+int2string(al.mm)+int2string(al.ss);

char date[40];
char time[40];

dateString.toCharArray(date,dateString.length()+1);
timeString.toCharArray(time,timeString.length()+1);

writeEEPROMstring(EEPROM1,ADD_DATE,ADD_DATE_SIZE,date);
writeEEPROMstring(EEPROM1,ADD_TIME,ADD_TIME_SIZE,time);
}

void readAlarmFromEEPROM ()
{
```

```
String time = readEEPROMstring(EEPROM1,ADD_TIME,ADD_TIME_SIZE);
String date = readEEPROMstring(EEPROM1,ADD_DATE,ADD_DATE_SIZE);

alarm.year = date.substring(0,4).toInt();
alarm.mon = date.substring(4,6).toInt();
alarm.day = date.substring(6,8).toInt();
alarm.hh = time.substring(0,2).toInt();
alarm.mm = time.substring(2,4).toInt();
alarm.ss = time.substring(4,6).toInt();
}
```

Nel blocco di setup viene controllato un **flag memorizzato nella EEPROM** nella locazione di memoria ad **indirizzo 0** (*ADD_FIRSTTIME*).
Tale flag, **se posto pari a 0**, indica che quello in corso è il **primo utilizzo della EEPROM** e, solo in questo caso, si procede ad impostare l'orario corrente sul RTC manualmente e, eventualmente, a salvare un allarme di test disabilitato sulla EEPROM.
Viene inoltre impostato il **pin relativo al Buzzer come uscita**, e posto a valore logico basso.

Nel Loop, invece, la funzione **sprtinf(**cmd Buffer,**receiveCommand(**cmdBuffer**))** inserisce, se presente, un comando nell'array di char **cmdBuffer**, una volta privato del payload dalla funzione **depackCmd**(char*cmd).
Il comando, che non prevede alcun tipo di eccezione, è atteso nel formato specificato nell'app iOS (**yyyyMMddHHmmE**).
Una volta ricevuto, questo viene scomposto in sottostringhe **yyyy MM dd HH mm E** dalla funzione **executeCommand**(char *cmd), che, convertite in interi, vanno a popolare la variabile globale alarm.
La struct alarm viene, al ciclo successivo, utilizzata dalla funzione **shouldPlayAnAlarm**(alarmType *alarm) che compara i suoi campi con data e ora correnti lette dal RTC e, qualora risultassero uguali e il flag **alarm.enabled** fosse pari a **true**, comunicherebbe ad Arduino di **attivare il Buzzer**.



```
void loop()
{
delay(1000);
if(shouldPlayAnAlarm(alarm)==true){
Serial.print("\nALARM!\n");
lcd.setCursor(0, 0);
lcd.print("ALARM!");
digitalWrite(kBuzzer, HIGH);
}
else{
digitalWrite(kBuzzer, LOW);
lcd.setCursor(0, 0);
lcd.print(printTime());
Serial.print(printTime());
}

char cmdBuffer[20];
sprintf(cmdBuffer,receiveCommand(cmdBuffer));
}

int shouldPlayAnAlarm(alarmType al)
{
Time t = rtc.time();
if(t.yr==al.year&&t.mon==al.mon&&t.date==al.day&&t.hr==al.hh&&t.min==al.mm)
if(al.enabled ==true)
return 1;
return 0;
}

void executeCommand(char *cmd)
{
char strTemp[20];
strTemp[0]=cmd[0];
strTemp[1]=cmd[1];
strTemp[2]=cmd[2];
strTemp[3]=cmd[3];
strTemp[4]='\0';
alarm.year = atoi(strTemp);
strTemp[0]=cmd[4];
strTemp[1]=cmd[5];
strTemp[2]='\0';
alarm.mon = atoi(strTemp);
strTemp[0]=cmd[6];
strTemp[1]=cmd[7];
strTemp[2]='\0';
alarm.day = atoi(strTemp);
strTemp[0]=cmd[8];
strTemp[1]=cmd[9];
strTemp[2]='\0';
```

```
alarm.hh = atoi(strTemp);
strTemp[0]=cmd[10];
strTemp[1]=cmd[11];
strTemp[2]='\0';
alarm.mm = atoi(strTemp);
strTemp[0]=cmd[12];
strTemp[1]='\0';
if(atoi(strTemp)==1)
alarm.enabled=1;
else
alarm.enabled=0;

}
char* receiveCommand(char *buffer)
{
int k=0;
while (mySerial.available()){
buffer[k]=mySerial.read();
k++;
}
if(k!=0)
{
buffer[k]='\0';
executeCommand(depackCmd(buffer));
return "OK";
}
else
return NULL;
}

char* depackCmd(char* cmd)
{
char strTemp[20];
int k=1,endFound=0;
for(int i=0;i if(cmd[i]=='*'){
k--;
}
else{
strTemp[k]=cmd[i];
if(strTemp[k]==';'){
strTemp[k]='\0';
endFound=1;
break;
}
k++;
}
}
if(endFound==0)
Serial.print("Missing End");
return strTemp;
}
```

CONCLUSIONI

Abbiamo realizzato insieme una Smart Sveglia programmabile da remoto tramite Bluetooth Low Energy da Smartphone iOS. Lo scopo di questo piccolo progetto non è quello di realizzare un prodotto finito e funzionante bensì di aprire la mente ai lettori riguardo potenzialità e problematiche legate alla scrittura su una EEPROM e alle esigenze di sincronizzazione tra orario UTC e Arduino.

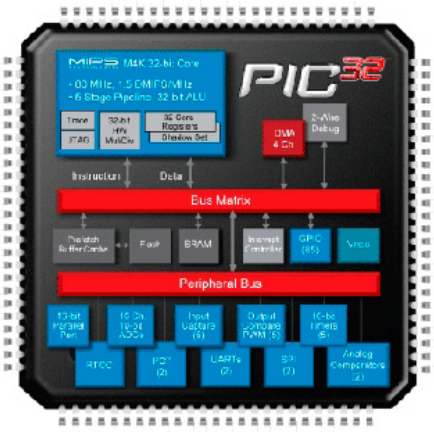
https://www.youtube.com/watch?v=842gnyS4QTU

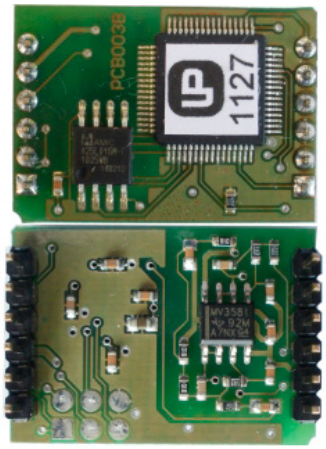
L'autore è a disposizione nei commenti per eventuali approfondimenti sul tema dell'Articolo.
Di seguito il link per accedere direttamente all'articolo sul Blog e partecipare alla discussione:
http://it.emcelettronica.com/realizziamo-smart-sveglia-bluetooth-con-arduino

Costruiamo un Voice Shield per far parlare Arduino [Progetto Completo Open Source]

di adrirobot

Il progetto proposto permette di dare la parola ad Arduino; sarà così facile, per esempio, sentire il valore letto tramite una porta analogica attraverso l'altoparlante connesso alla scheda. Il tutto è possibile utilizzando uno speciale shield su cui è installato un modulo sonoro **LPM11162** con cui è possibile memorizzare e riprodurre file audio in formato wave. Lo shield misura 43x59 mm e, su di esso, sono presenti uno stadio alimentatore, un'interfaccia per l'adattamento delle tensioni di comunicazione tra il processore Arduino e il modulo LPM11162, uno stadio amplificatore e, infine, una sezione sensori e una di I/O.





SCHEMA ELETTRICO

Cuore dello shield è il Modulo **LPM1162**, che si presenta come un piccolo circuito (28x20 mm) dotato di due file da sei pin. Sul modulo sono presenti pochi componenti tra cui il processore, un PIC32MX320F, e una memoria flash da 2Mbyte che permette di memorizzare un massimo di 95 secondi di registrazione in formato wave 11 KHz-16bit mono (tempo massimo per tutti i file), mentre il numero massimo di file è di 512 e non c'è limitazione alle dimensioni di ciascun file.

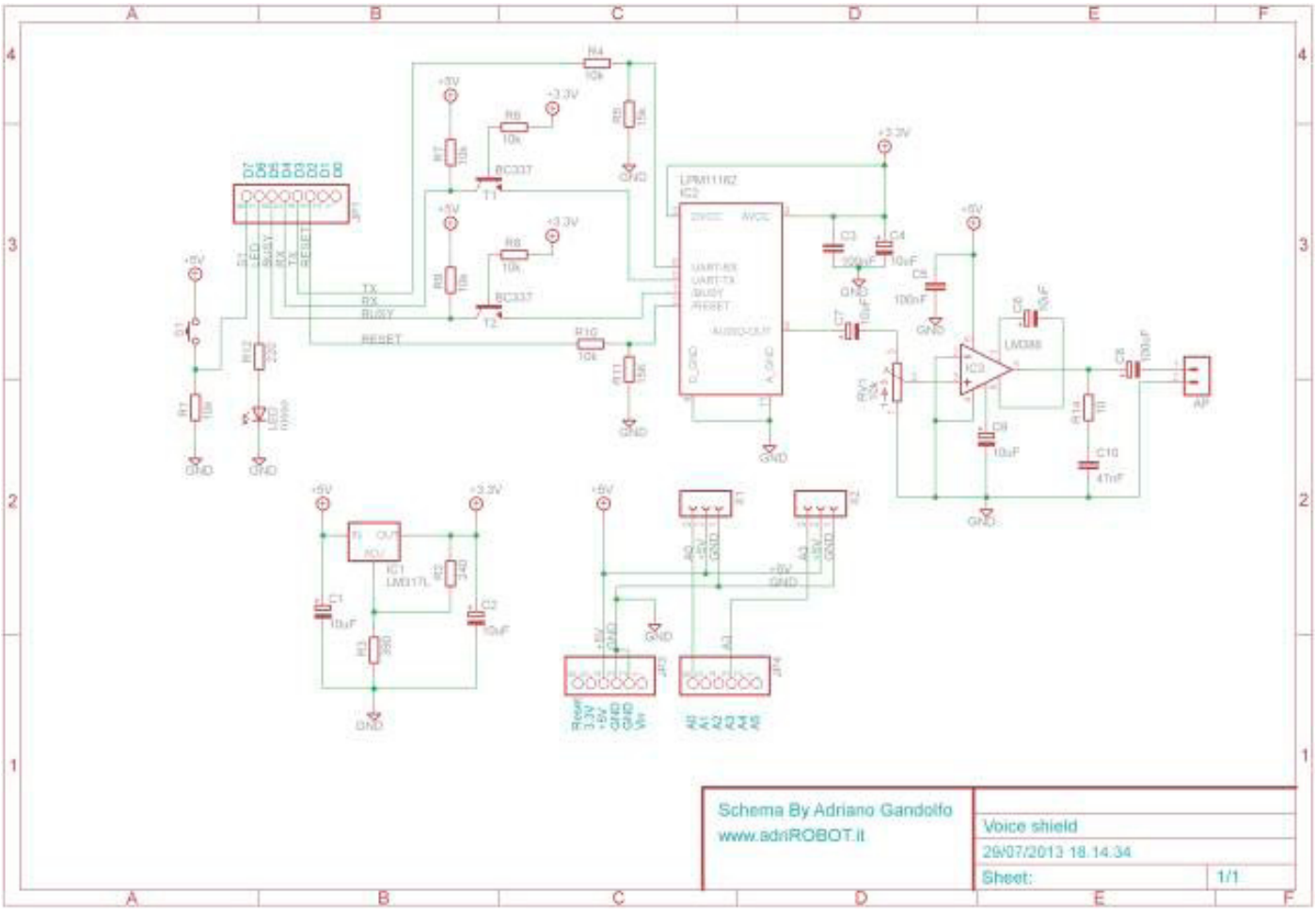
Per gestire il modulo sono necessari pochi e semplici comandi seriali che vengono inviati tramite le quattro linee RX, TX, Busy, Reset.

Per maggiori informazioni e per l'acquisto del modulo vedere il sito del produttore **LP Elettronica**

DESCRIZIONE DEL CIRCUITO

Per la descrizione del circuito partiamo dallo stadio di alimentazione del modulo **LPM11162** che necessita della tensione di 3,3 V, senza utilizzare quella fornita da Arduino.

Nel nostro caso, si utilizza la tensione derivata da quella a +5V che è ridotta utilizzando un classico regolatore **LM317LZ** (corrente massima fornita 100mA), il valore della tensione è impostato tramite le due resistenze da 240Ω e 390Ω. Dato che il modulo LPM11162 funziona con una tensione di alimentazione di +3,3 V, anche le comunicazioni seriali e i segnali di reset e busy sono e devono essere a questa tensione. Visto che Arduino ha una logica a +5V, è necessario adattare i livelli uscenti ed entranti dal modulo. Per quelli entranti (RX - Reset), è sufficiente un



partitore resistivo 10K/15K che adatta la tensione d'uscita 5 V di Arduino ai 3,3 V con cui il modulo LPM11162 è alimentato. Per quelli uscenti, si sono utilizzati due transistor **BC337** che traslano il livello d'uscita 3,3 V del modulo audio ai 5 V di Arduino.

Per amplificare il segnale audio analogico in uscita dal pin 9 del modulo LPM11162, è utilizzato un integrato amplificatore, in questo caso un **LM386**. Questo circuito integrato che si presenta in un contenitore plastico ad 8 pin (DIP8), è molto diffuso perché con pochi componenti esterni è in grado di pilotare direttamente un piccolo altoparlante: con 5 V di alimentazione riesce a fornire una potenza di circa 300 mW su un altoparlante da 8 Ohm. Per quanto riguarda il guadagno, questo è impostato a 200 tramite il condensatore C6 da 10 µF connesso ai pin 1 ed 8.

Il condensatore, infatti, influisce su questo valore dal momento che esso funge da bypass per il resistore e la componente resistiva in gioco è quella della impedenza complessa che varia come 1/C.

La regolazione del volume è ottenuta con il trimmer RV1 da 10 kOhm, il condensatore da 47 nF e la resistenza da 10 Ω in serie sull'uscita dell'amplificatore, sono necessari per rendere stabile l'amplificatore come indicato nel Datasheet. In serie all'altoparlante è presente il condensatore di disaccoppiamento C9 che costituisce un blocco per la componente in continua evitando che la medesima venga inviata all'altoparlante.

Per il collegamento dei sensori sono presenti due connettori, **X1** connesso all'ingresso Analogico A0, e **X2** connesso all'ingresso A3.

Abbiamo poi altre due possibilità di interagire con la scheda, rappresentati da **un pulsante**

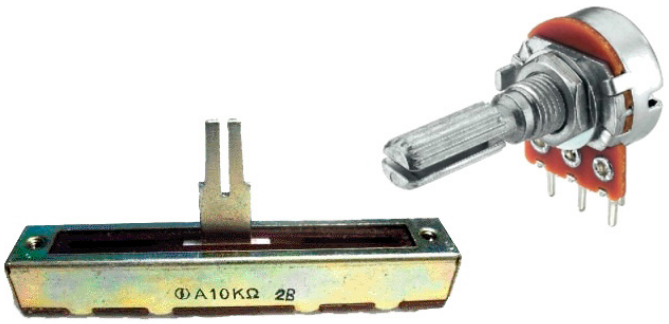
connesso alla porta digitale D7 e da **un led** dotato di resistenza limitatrice connesso alla porta digitale D6.

CONNETTORI PER SENSORI

Ai connettori X1 e X2, è possibile collegare sensori con uscita analogica come un sensore di temperatura LM35; questo si presenta con tre terminali: uno per l'alimentazione, uno di massa e uno per l'uscita della tensione proporzionale alla temperatura rilevata che è pari a 10 mV per ogni grado centigrado, ed è calibrato in gradi Celsius. Per trasformare la tensione letta si può utilizzare la formula:

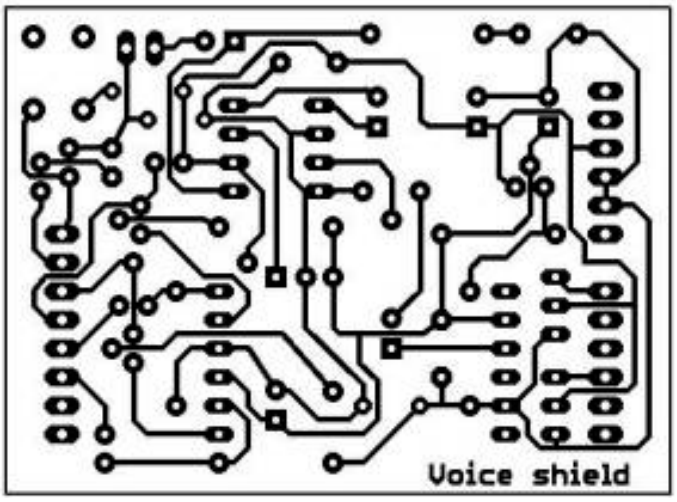
temp = (5.0 * analogRead(tempPin) * 100.0) / 1024

Si consideri che, con un campionamento a 10 bit, Arduino ha una risoluzione in tensione di circa 5 mV, e considerando che **LM35** fornisce 10 mV per ogni grado centigrado, la massima precisione che si può ottenere, è di mezzo grado.
Altro tipo di dispositivo collegabile al connettore è un potenziometro; si possono utilizzare indifferentemente i potenziometri in contenitore rotativo o lineare.



Anche in questo caso per la lettura della porta analogica, si userà il comando **analogRead()** che converte la tensione d'ingresso da 0 a 5 volt, in un valore digitale. Ruotando l'albero del potenziometro, si cambia la quantità di resistenza su entrambi i lati del pin centrale del potenziometro. Questo cambia la resistenza relativa tra il pin centrale e i pin esterni, dando una tensione diversa all'ingresso analogico. Quando l'albero è girato completamente in una direzione, non vi è resistenza tra il pin centrale e il pin collegato a GND. La tensione al pin centrale è quindi 0 volt. Quando l'albero è girato tutto nella direzione opposta, non vi è resistenza tra il pin centrale e il pin collegato a +5 volt. La tensione al pin centrale è quindi di 5 volt. Perciò analogRead() restituirà un numero tra 0 e 1023 che è proporzionale alla quantità di tensione applicata al pin analogico.

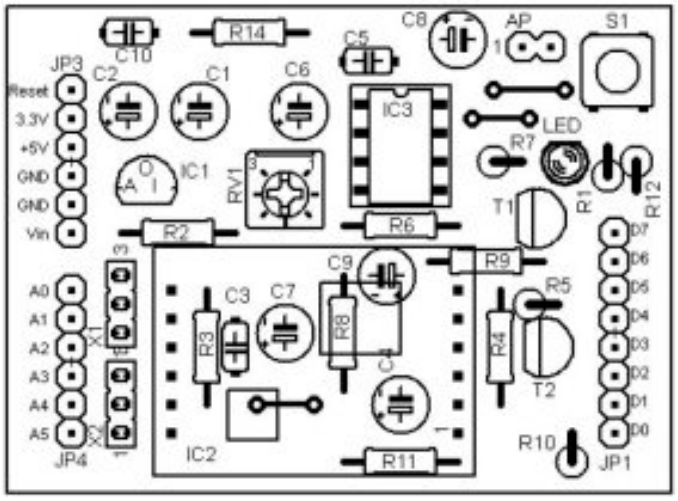
CIRCUITO STAMPATO



L'immagine vista riporta la traccia del circuito stampato, questo è del tipo monofaccia e sono previsti solamente tre ponticelli.

Piano di montaggio

Di seguito invece è visibile l'immagine del piano di montaggio.



LISTA PARTI

Part list Voice shield con LMP11162

n°	Q.tà	Riferimento	Descrizione	Part
1	6	C1,C2,C4,C6,C7,C9	Condensatore	10 µF elettrolitico
2	2	C3,C5	Condensatore	100 nF multistrato
3	1	C8	Condensatore	100 µF elettrolitico
4	1	C10	Condensatore	47nF poliestere
5	1	IC1	Regolatore di tensione	LM317L
6	1	IC2	Modulo audio	LPM11162
7	1	IC3	Amplificatore	LM386
8	2	T1,T2	Transistor	BC337
9	1	LED	Led	Led rosso Ø 3mm
10	7	R1,R4,R6,R7,R8,R9,R10	Resistenza	¼W 10kΩ
11	1	R2	Resistenza	¼W 240Ω
12	1	R3	Resistenza	¼W 390Ω
13	2	R5,R11	Resistenza	¼W 15kΩ
14	1	R12	Resistenza	¼W 220 Ω
15	1	R14	Resistenza	¼W 10 Ω
16	1	RV1	Trimmer	10kΩ
17	1	S1	Pulsante	Pulsante CS
18	2	X1,X2	Pinstrip	maschio 1x3
19	1	JP1	Pintrip	femmina 1x08
20	2	JP3,JP4	Pintrip	femmina 1x06
21	1	AP	Pintrip	maschio 1x2

REALIZZAZIONE PRATICA

Per la costruzione della scheda, si procederà iniziando dalla realizzazione del circuito stampato, la traccia è scaricabile in scala 1:1 mediante il link (in fondo all'articolo).

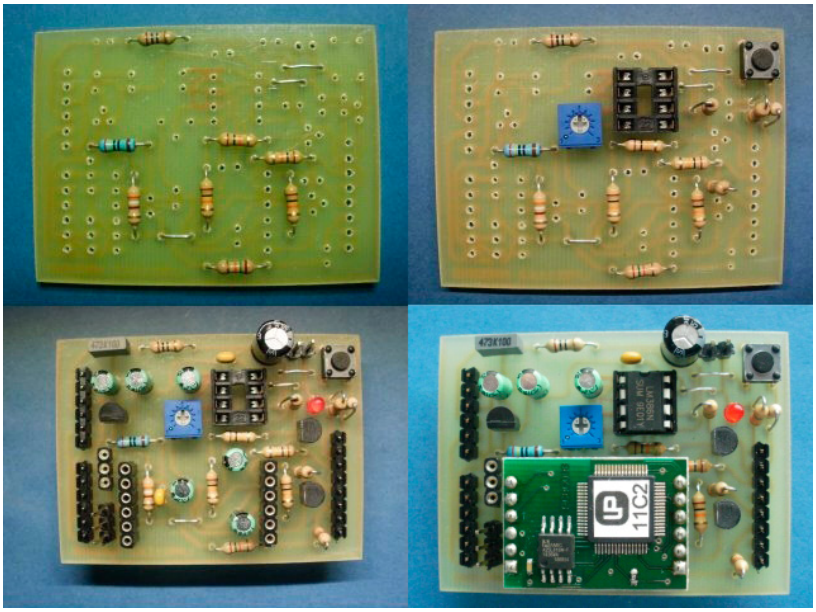
Occorre ricordare che la stampa dovrà essere fatta deselezionando la funzione che adatta il foglio alla pagina. Per la sua realizzazione, si utilizzerà una basetta in vetronite (monofaccia) di dimensioni 43x59 mm, il metodo potrà essere quello della fotoincisione o del trasferimento termico utilizzando i cosiddetti fogli blu (**PRESS-N-PELL**), in questo caso ricordo che l'immagine delle tracce del circuito dovrà essere speculare. Una volta inciso il rame, si verificherà in controluce o mediante l'utilizzo di un multimetro, che non vi siano cortocircuiti soprattutto tra le piste più vicine. Si passerà quindi alla foratura della stessa, utilizzando principalmente una punta da 0,8 mm, mentre, se ne utilizzerà una da 1 mm per le pin strip. In seguito, si potrà passare al posizionamento e alla saldatura dei componenti seguendo lo schema di montaggio visto prima.

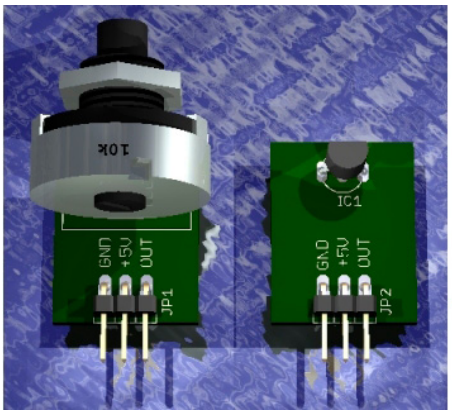
Per la saldatura, si utilizzerà un piccolo saldatore a punta fine, della potenza di circa 25 – 30 W. S'inizierà dai vari ponticelli, continuando con le resistenze; si potrà, quindi, procedere con il pulsante, lo zoccolo dell'integrato, i condensatori. Si concluderà con le pin strip e i connettori.

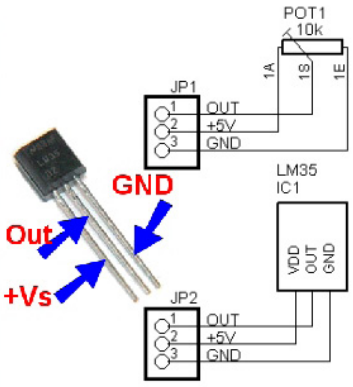
Terminata la saldatura, si potranno inserire gli integrati IC2 (Modulo LPM1162) e IC3 (LM386) nell'apposito zoccolo facendo attenzione alla tacca di riferimento.

SENSORI

Come detto sopra, alla scheda possono essere connessi vari sensori, la cui uscita sia analogica con un range di tensione compreso tra 0 e +5V. Nella figura sotto, sono visibili due esempi: un potenziometro (utilizzato poi nel test) e un sensore di temperatura LM35.

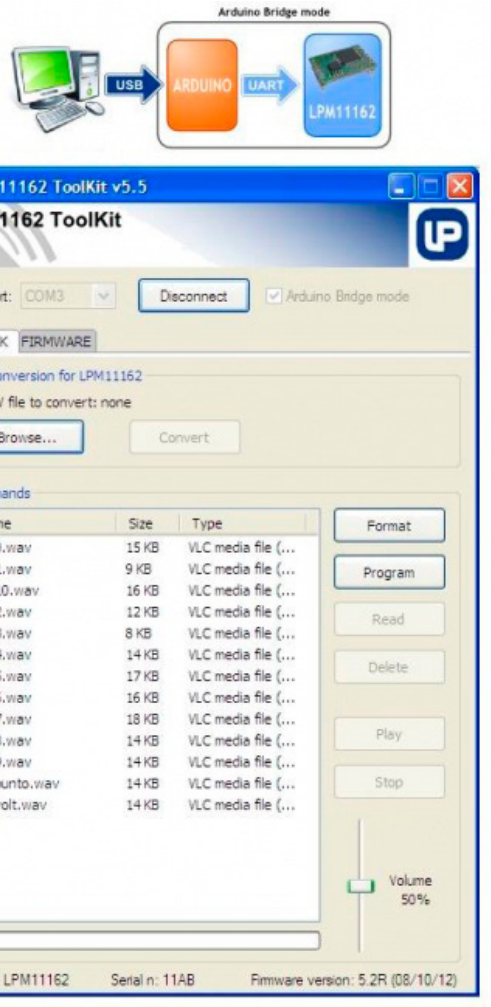






Firmware.

Per la programmazione tramite la scheda Arduino, è disponibile la **modalità Bridge**: Arduino diventa programmatore per il modulo audio LPM11162 e non sono necessari dispositivi esterni. Quindi il modulo LPM11162 è collegato ad Arduino che a sua volta è collegato al PC tramite una COM virtuale (connessione USB).



PROGRAMMA TOOLKIT

Per trasferire i file wav nella memoria del modulo LPM11162, è disponibile presso il sito del produttore, uno speciale programma chiamato **LPM11162 ToolKit**; con quest'applicazione per PC l'uso del modulo diventa semplice.

Con questo programma è possibile:

- Convertire i file wave nel formato mono-11KHz-16bit.
- Programmare i file wave sul LPM11162
- Riprodurre i file wave programmati con regolazione del volume.
- Verificare la versione di firmware dei moduli LPM11162.
- Aggiornare il firmware all'ultima versione

LIBRERIA ARDUINO

Per la gestione del modulo, è disponibile, sempre presso il sito del produttore, la **LPM11162 Arduino Library** che è una libreria open-source per il controllo dei moduli audio LPM11162 con Arduino.

La libreria fornisce alcune semplici funzioni e invia comandi seriali al modulo audio per riprodurre i file programmati. Sono disponibili le seguenti funzioni:

begin(baudrate)

Configura la libreria e l'hardware per comunicare con il modulo audio LPM11162 con il baudrate selezionato, tipicamente 9600.

end()

All'opposto di begin() rilascia interamente le risorse hardware e software impegnate per la comunicazione con il modulo audio LPM11162.

play(fileName)

Inizia la riproduzione del file WAV il cui nome è specificato come parametro della funzione.

stop()

Interrompe la riproduzione del file WAV che è in corso.

volume(volume)

Regola il livello del volume di riproduzione audio passando un numero tra 0 e 100 come parametro per indicare il volume.

isBusy()

Legge lo stato del pin /BUSY del modulo LPM11162. Questa informazione è utile per sapere se la riproduzione di un file WAV è ancora in corso.

reset(resetLevel)

Pilota direttamente il pin /RESET del modulo LPM11162.

synch()

Verifica la presenza del collegamento con il Toolkit per la programmazione dei file WAV sul modulo LPM11162.

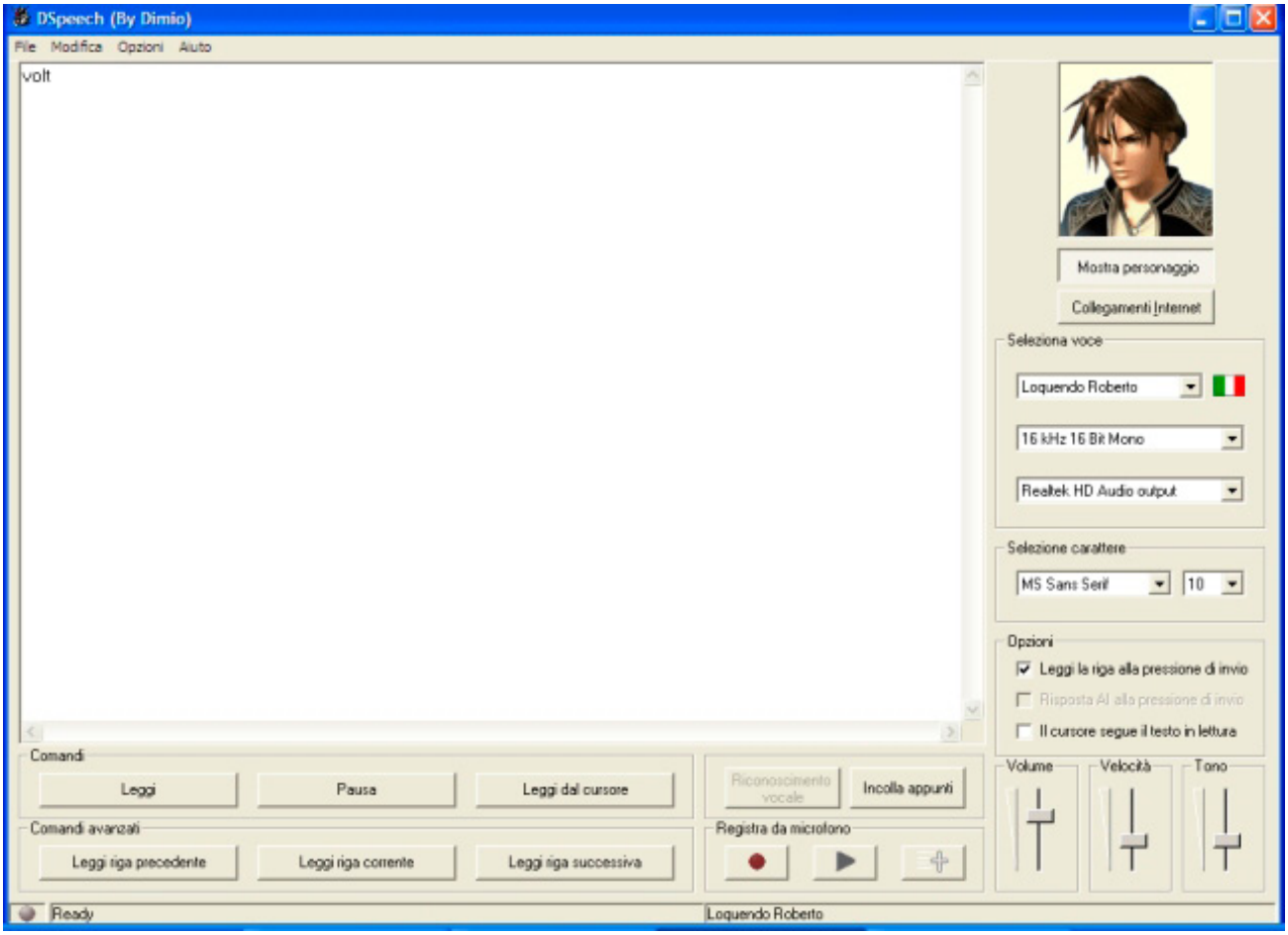
bridge()

Con questa funzione la libreria Arduino entra in modalità Bridge.

PROGRAMMA PER CREAZIONE FILE WAV

Se, oltre ai numeri e alle parole presenti nel programma demo, si vogliono registrare altri numeri o parole da pronunciare, si può utilizzare un qualsiasi programma di registrazione audio, ma utilizzando un programma di **TTS** cioè Text To Speech tutto si semplifica poiché è sufficiente scrivere quello che vogliamo far pronunciare e il programma, utilizzando la sintesi vocale che è la tecnica per la riproduzione artificiale della voce umana, creerà per voi il file che potrà essere salvato in formato wav. Quello che consiglio si chiama **DSpeech** è stato scritto da Dimitrios Coutsoumbas (Dimio). Il programma che tra l'altro è freeware, oltre alle funzioni TTS, possiede le funzionalità di **ASR** (Automatic Speech Recognition) integrate. E' cioè, in grado di leggere ad alta voce il testo scritto e di scegliere le frasi da pronunciare a seconda delle risposte vocali dell'utente.
Non ha bisogno di essere installato e occupa poca memoria. Permette di salvare l'output sotto forma di un file Wav, Mp3, Aac, Wma o Ogg.

Per scaricarlo e per le istruzioni di come installarlo vedere il seguente link.



PROGRAMMA DI PROVA

Segue un esempio di programma, in questo caso è letto il valore di tensione variato tramite un potenziometro collegato al connettore X2. Il valore di tensione è poi "pronunciato" tramite l'altoparlante connesso all'uscita. La procedura di caricamento è la seguente.

Lanciare l'IDE e trasferire il programma nella memoria di Arduino. A caricamento terminato, lanciare il programma ToolKit e utilizzando la funzione bridge trasferire nella memoria del modulo LPM11162 i file wav presenti nel file zip allegato a quest'articolo, essi sono riferiti ai numeri, la parola "punto" e "volt". Sconnettere il programma toolkit e resettare la scheda Arduino. A questo punto ruotando l'alberino del potenziometro, si udrà dall'altoparlante il valore di tensione letta

https://www.youtube.com/watch?v=iDwlAd46qlI

```
#include <SoftwareSerial.h>

#include <LPM11162.h>

#define LPM11162_RX     3 //Arduino TX (out)

#define LPM11162_TX     4 //Arduino RX (in)

#define LPM11162_RESET  2 //(out)

#define LPM11162_BUSY   5 //(in)
```

```
#define pin_tensione   A3 // Pin analogico lettura


LPM11162 audio(LPM11162_RX, LPM11162_
TX, LPM11162_RESET, LPM11162_BUSY);

int pinval = 0;

int oldpinval =0;


void setup()

{

 Serial.begin(9600);

 if ( audio.synch() )

 {

  audio.bridge();

 }

 audio.begin(9600);

}


void loop()

{

 pinval = analogRead(pin_tensione);  // Legge il
pin analogico

 if(pinval != oldpinval)

 {

  oldpinval = pinval;

  SayVolts(pinval);    // Converte in una tensione
e dice il valore

 }

 delay (2000);
}


void SayVolts(int volts)

{

 int BigNumber = 0;

 int SmallNumber = 0;


 BigNumber = abs(volts/204);   // Converte il
valore analogico in tensione

 Serial.println (BigNumber);

 SmallNumber = volts - abs(BigNumber*204); //
Converte i decimali ad una sola cifra

 SmallNumber = abs(204-SmallNumber)/2;

 SmallNumber = abs(100-SmallNumber)/10;

 Serial.println (SmallNumber);

 if (BigNumber != 0)

 {

  String intero= String (String (BigNumber) +
".wav");

  char frase_array[20];

  intero.toCharArray( frase_array, sizeof(frase_
array) );

  audio.play(frase_array);

  while( audio.isBusy() );

  delay (1000);

 }

 audio.play("punto.wav");        // Dice "punto"
```

```
  while( audio.isBusy() );

  delay (1000);


  String decimale = String (String (SmallNumber)
+ ".wav");

  char frase_array[20];

  decimale.toCharArray( frase_array, sizeof(frase_
array) );

  audio.play(frase_array);

  while( audio.isBusy() );

  delay (1000);


  audio.play("volt.wav");       // Dice "volt"

  while( audio.isBusy() );

  delay (1000);

}
```

- Schema elettrico PDF
- Layout PDF
- Circuito stampato PDF
- Schema elettrico e layout sorgente EAGLE
- Sketch per Arduino
- File wav da memorizzare sul Modulo LPM11162.

CONCLUSIONI

Il programma presentato è molto semplice, ma è possibile crearne altri, per esempio uno che leggendo la tensione in uscita dal sensore di LM35, converte il valore letto e fa pronunciare quest'ultimo tramite l'altoparlante.

Documentazione completa Open Source

La documentazione completa è disponibile, a questo link: **voice_shield_arduino.zip**

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/costruiamo-voice-shield-far-parlare-arduino-progetto-completo-open-source

TUTORIAL

- Just another "Getting Started with Arduino"
- Come scrivere una libreria per Arduino
- Sviluppo Software per Arduino con Atmel Studio
- Programmare Arduino UNO con Atmel Studio
- Un encoder per Arduino
- SMAs: gestiamo questi materiali "intelligenti" con Arduino
- Realizzazione di un rilevatore SONAR con Arduino
- Più sprint alla tua auto con Arduino e Processing
- La regolazione di temperatura con Arduino
- MEMS, dalla teoria alla pratica: usiamo un Accelerometro con Arduino
- Temperature & pressure monitoring con Arduino
- Pilotare motori passo-passo con Arduino
- Controllare I/O multipli con pochi pin di Arduino

PROGETTI

- Analizzatore MIDI con Arduino
- Progetto serra domotica open source
- Inverter ad onda sinusoidale Open Source con scheda Infineon Arduino motor shield
- Camera slider DIY low cost con Arduino [Progetto completo]
- Arduino Micro e BMP180 Bosch per realizzare una Weather Station
- Progetto Giardiniere: Gestire una serra domestica con Arduino
- Un telecomando TV per comandare un robot cingolato
- G.F.A.rduino (Generatore di Funzioni Arbitrarie con Arduino)
- Arduino abbassa il volume degli spot TV! [Fai-Da-Te]
- Stazione Meteo Online con Arduino
- Crepuscolare fai-da-te con Arduino
- Realizziamo una Smart Sveglia Bluetooth con Arduino
- Costruiamo un Voice Shield per far parlare Arduino [Progetto Completo Open Source]