Laboratorio microcontrollori e open source
Prima parte

Politecnico Open unix Labs

13 Aprile 2012

Introduzione

Questo corso è una breve introduzione ai microcontrollori, concentrandosi sia sugli aspetti software di programmazione che hardware di costruzione di circuiti.

Verranno presentate due piattaforme per lo sviluppo di applicazioni:

- Arduino che è una piattaforma molto popolare basata su microcontrollori Atmel a 8bit
- STM32, una architattura di microcontrollori più potente a 32bit, usando il sistema operativo Miosix.

Tutto il corso sarà tenuto in ambiente Linux, usando solo strumenti Open Source.

Struttura del corso

Il corso si compone di tre lezioni.

- Lezione 1 (oggi): Basi di Arduino
  - Breve introduzione ai microcontrollori
  - Breve introduzione ad Arduino
  - Ampia sessione di sperimentazione pratica con semplici esempi usando Arduino
- Lezione 2: Basi di STM32 e Miosix
  - Breve introduzione ai microcontrollori STM32
  - Breve introduzione a Miosix
  - Ampia sessione di sperimentazione pratica con semplici esempi usando STM32 e Miosix
- Lezione 3: Progetti avanzati
  - Verranno mostrati progetti più complessi basati sia su STM32 che Arduino

Microcontrollori

Cos'è un computer?

Immaginario collettivo:

- un dispositivo con schermo, tastiera e mouse, in grado di eseguire programmi, connettersi a Internet, etc.

Definizione troppo limitata:

- smartphone e tablet

Microcontrollori

Cos'è un computer?

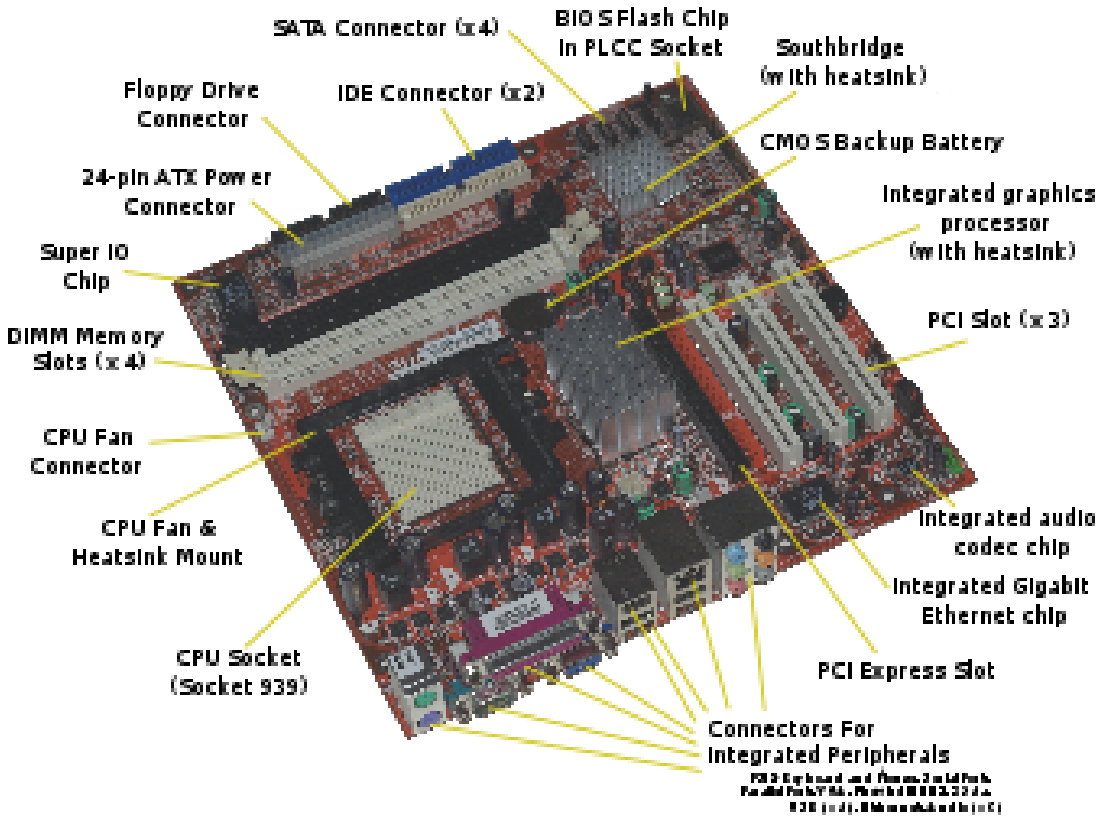
Definizione più teorica:

- un dispositivo con una CPU e della memoria in grado di contenere dati e istruzioni, con un instruction set turing completo

Definizione molto più ampia dell'immaginario collettivo

- sistemi embedded: computer all'interno di altri dispositivi, dall'orologio alla lavatrice

La scheda madre di un tradizionale desktop computer



Il processore, la memoria RAM, e la memoria di massa sono componenti separati.

Microcontrollori

Un microcontrollore è un “computer on a chip”

Il processore, la memoria RAM, e la memoria nonvolatile si trovano nello stesso componente.

Microcontrollori

In un microcontrollore le risorse sono molto più limitate rispetto ad un PC o a uno smartphone.

	CPU	RAM	Memoria di massa
Desktop PC	32/64bit, 1..3GHz	>2GByte	Hard disk, >100GByte
Smartphone	32bit, ~1GHz	>100MByte	FLASH, >1GByte
STM32	32bit, ~100MHz	8..192KByte	FLASH, 16..1024KByte
ATmega	8bit, ~10MHz	~1KByte	FLASH, 2..128KByte

Arduino

Arduino è una piattaforma per semplificare la programmazione di microcontrollori.
La piattaforma è composta da

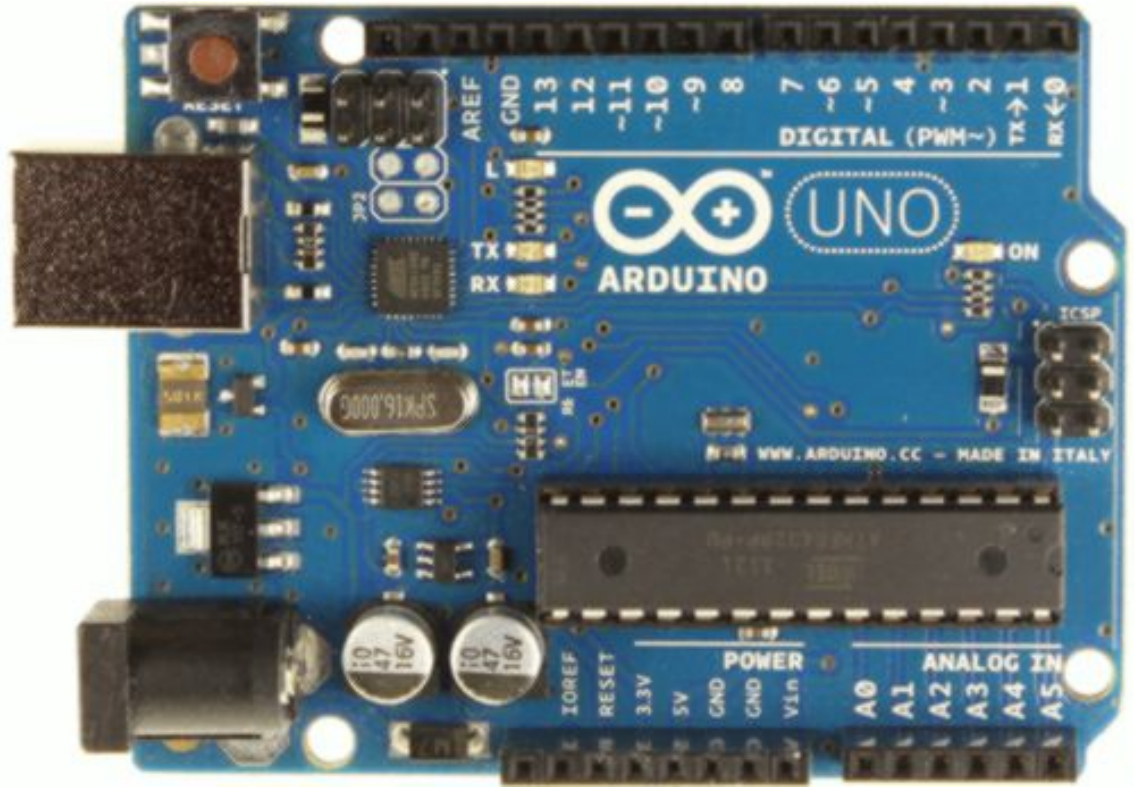
- Una board basata su un microcontrollore ATmega a 8bit
  - Evita di doversi occupare di fornire alimentazione e clock al microcontrollore
  - Tramite un bootloader, consente la programmazione del microcontrollore direttamente via USB, senza bisogno di un programmatore
  - Fornisce accesso ai GPIO del microcontrollore, su due connettori posti ai lati della scheda
  - Consente di connettere altre board "shield" per offrire ulteriori funzionalità
- Un ambiente di sviluppo lato PC
  - Consente la programmazione in un dialetto semplificato del C++
  - Ha una interfaccia minimale, pensata per essere usabile anche da chi non è esperto di programmazione
  - Consente di trasferire i programmi sulla board

Arduino - Board

Ci sono molti tipi di board compatibili con Arduino

- Arduino USB: è la board più comune. E' a sua volta una famiglia di board in quanto ci sono molte versioni di questa board, che si differenziano per la quantità di memoria e altri piccoli dettagli. Ogni board ha un nome, come "Arduino Uno", "Arduino Duemilanove", "Arduino Diecimila", ...
- Arduino Mega, con più GPIO e più memoria
- LilyPad, pensato per applicazioni di wearable hardware
- Altre board meno conosciute, come la prima versione di Arduino, con una porta seriale al posto della USB, o l'"Arduino Nano" pensato per l'uso direttamente su breadboard.

Arduino - Board



Arduino - IDE

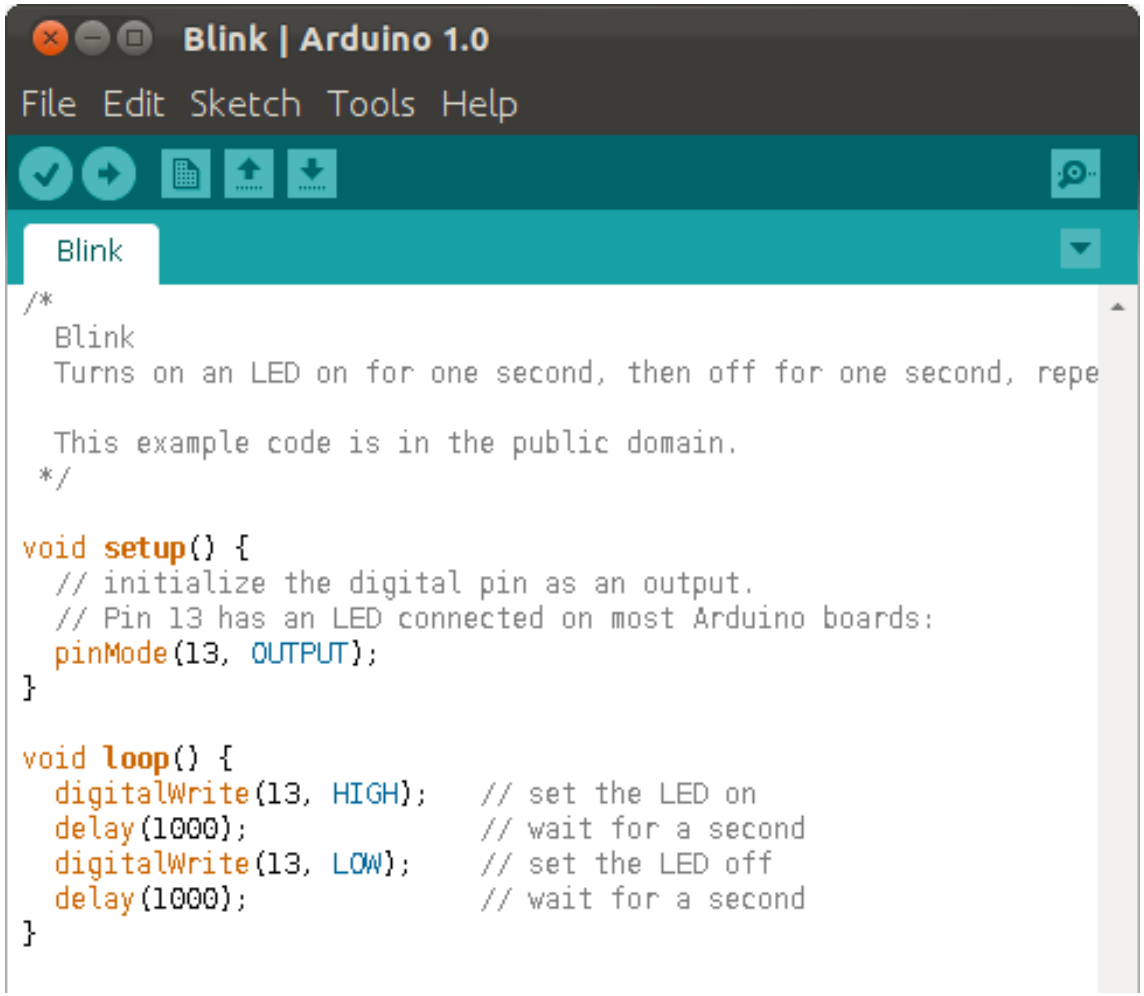
L'ambiente di sviluppo per Arduino si scarica da *http://www.arduino.cc/*

E' disponibile per Linux, Mac OS X, Windows

Contiene esempi preconfezionati per imparare a programmare le board

La documentezione delle librerie è invece disponibile online all'URL *http://arduino.cc/en/Reference/HomePage*

Arduino - IDE


Blink | Arduino 1.0
File Edit Sketch Tools Help
Blink
/*
Blink
Turns on an LED on for one second, then off for one second, repe
This example code is in the public domain.
*/
void setup() {
// initialize the digital pin as an output.
// Pin 13 has an LED connected on most Arduino boards:
pinMode(13, OUTPUT);
}
void loop() {
digitalWrite(13, HIGH); // set the LED on
delay(1000); // wait for a second
digitalWrite(13, LOW); // set the LED off
delay(1000); // wait for a second
}

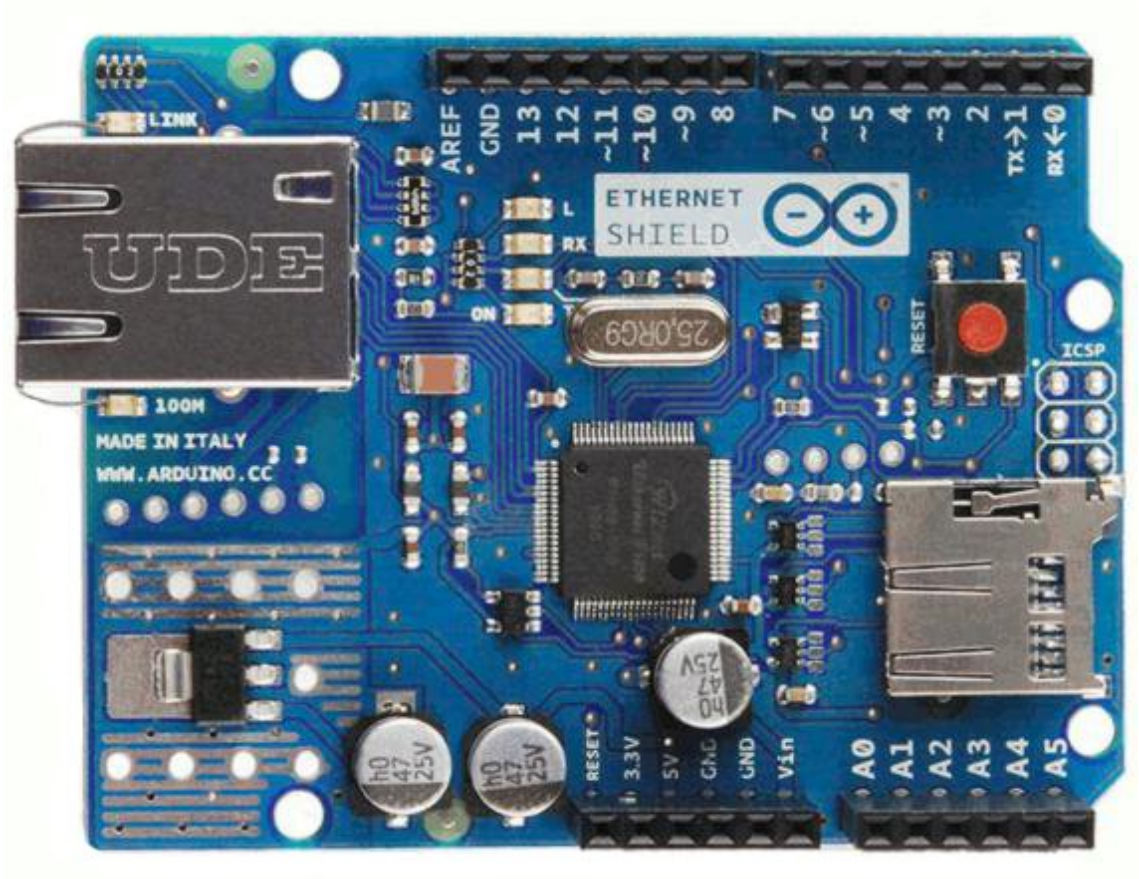
Arduino - Shield

Gli shield sono board agguintive, che aggiungono funzionalità alla board principale, come ad esempio connettività ethernet, pilotaggio di motori per applicazioni di robotica.

Spesso sono corredati di una libreria software per semplificarne l'utilizzo.

In questo corso non verranno usati shield.

Arduino - Shield



Sperimentazione pratica

Completata questa breve introduzione ai microcontrollori e ad Arduino, si può subito passare alla sperimentazione pratica.

Progettino #1: pulsanti e LED

Specifiche del progettino:

Collegare un pulsante e un LED a un Arduino.
Il led deve lampeggiare quando si preme il pulsante.

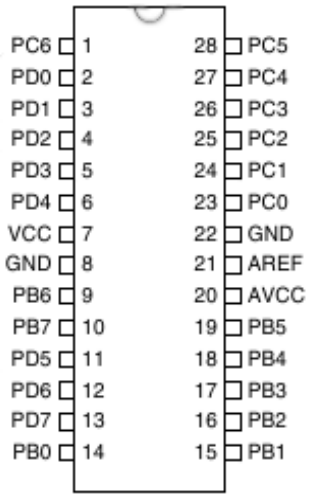
Progettino #1: pulsanti e LED

Organizzazione del progettino

- Teoria: i GPIO
- Software: le funzioni `setup()`, `loop()` e `delay()`
- Software: come accedere ai GPIO da Arduino
- Hardware: accendere un LED
- Hardware: leggere lo stato di un pulsante
- Pratica: costruzione del circuito

Progettino #1: pulsanti e LED: GPIO

I GPIO sono una delle periferiche più semplici di un microcontrollore, il nome sta per "General Purpose Input/Output" e la loro funzione è quella di poter controllare il livello logico su dei piedini di un circuito integrato tramite software.



La figura a destra mostra il microcontrollore ATmega328, quello usato nell'Arduino. Come si può notare, ai lati ci sono due file da 14 contatti elettrici, detti "piedini". Sulla sinistra è evidenziata la funzione di questi piedini.

Progettino #1: pulsanti e LED: GPIO

A parte alcuni piedini necessari a formire l'alimentazione al microcontrollore (indicati con VCC, AVCC e GND), ed un piedino specifico per l'ADC (AREF), tutti gli altri sono usabili come GPIO.

Caratteristiche dei GPIO

- sono configurabili in software come uscita, in questa modalità di funzionamento è possibile produrre sotto controllo software una tensione di uscita di zero Volt (livello logico 0), oppure una tensione pari a quella di alimentazione del microcontrollore, solitamente 5V o 3.3V (livello logico 1). La corrente che possono fornire è limitata, ma comunque sufficiente ad accendere un LED, oppure...
- ...sono configurabili come ingresso, in questa modalità è possibile leggere via software se la tensione ai loro capi è un livello logico 0 oppure 1.
- Sono raggruppati in porte, in modo da poter leggere/scrivere più GPIO contemporaneamente. Spesso le porte sono indicate con delle lettere, mentre i singoli GPIO all'interno delle porte con dei numeri. Quindi i GPIO hanno dei nomi come PA0, PA1, PB0, PB1. Le porte sono solitamente composte da 8, 16 o 32 GPIO.
- In Arduino, ogni GPIO ha un numero, riportato a lato delle board.

Progettino #1: pulsanti e LED: Arduino basics

Contrariamente a C/C++ dove ogni programma deve avere una funzione `main()`, un programma in arduino deve avere due funzioni, `setup()` e `loop()`:

```
void setup()
{

}

void loop()
{

}
```

Progettino #1: pulsanti e LED: Arduino basics

`setup()`

- Viene chiamata una sola volta quando si accende la board
- E' un buon posto dove scrivere codice per inizializzare le periferiche

`loop()`

- Viene chiamata ripetutamente finchè non si spegne la board
- All'interno si scrive la logica dell'applicazione

Il runtime Arduino esegue le seguenti operazioni

```
int main()
{
    setup();
    for(;;) loop();
}
```

Progettino #1: pulsanti e LED: Arduino basics

La funzione `delay()` tiene occupata la CPU per un tempo in millisecondi, passatogli come parametro.

Esiste anche la funzione `delayMicroseconds()` per ritardi di tempo più brevi.

Per conoscere il tempo trascorso dall'avvio di Arduino esiste la funzione `millis()` che restituisce i millisecondi trascorsi dall'inizio del programma.

Progettino #1: pulsanti e LED: GPIO e Arduino

I GPIO del microcontrollore sono connessi ai due connettori ai lati della scheda. I GPIO sono suddivisi in analogici e digitali

Sulla scheda è riportato un numero accanto a ogni GPIO. Questo numero è usato per identificare il GPIO in software.

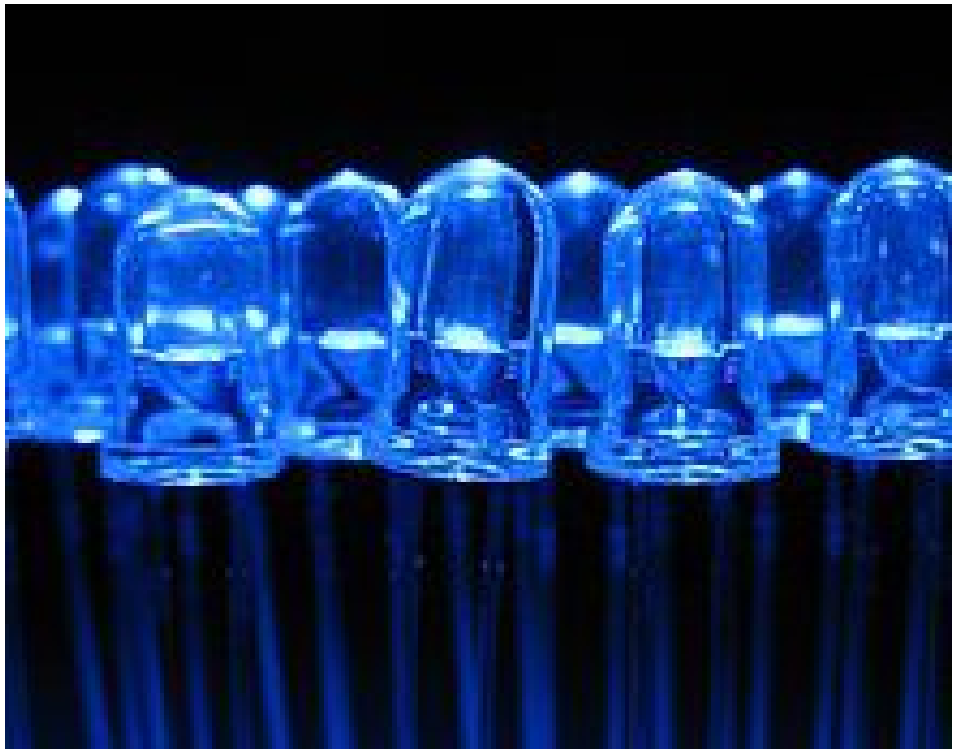
Progettino #1: pulsanti e LED: GPIO e Arduino

Operazioni eseguibili con un GPIO *digitale*

- `pinMode(<numero GPIO>, INPUT);` Configura il GPIO come ingresso
- `pinMode(<numero GPIO>, OUTPUT);` Configura il GPIO come uscita
- `digitalWrite(<numero GPIO>, HIGH);` Porta un GPIO configurato come uscita a livello logico alto (+5V)
- `digitalWrite(<numero GPIO>, LOW);` Porta un GPIO configurato come uscita a livello logico alto (0V)
- `digitalRead(<numero GPIO>);` Chiamata su un GPIO configurato come ingresso, ritorna HIGH o LOW

Progettino #1: pulsanti e LED: il LED

Un led è un componente elettronico, un diodo ad emissione luminosa. É un componente base e semplice da utilizzare, è comodo anche quando bisogna fare test molto grezzi per capire se c'è tensione in certi punti (questo perchè consuma pochissimo e influisce poco). Un led è dotato di due piedini, uno più lungo dell'altro. Quello più lungo è detto anodo e corrisponde al polo positivo (+), mentre quello più corto è detto catodo e corrisponde al polo negativo (-).



Progettino #1: pulsanti e LED: il LED

Collegando direttamente un led all'arduino la corrente che scorre è troppo elevata, con il rischio di bruciare il led o l'arduino. Per evitare sovraccarichi e non fornire troppa corrente al led possiamo mettere in serie al led (dal lato in cui va a massa o dal lato in cui va al pin, è indifferente) una resistenza che limiti la corrente che scorre nel led (che non deve in ogni caso mai superare al massimo i 20mA, pena l'esplosione/bruciatura del led).
Il calcolo della resistenza da mettere in serie possiamo farlo in base alla seguente formula: $R = \frac{V_{applicata} - V_{led}}{A_{led}}$ dove A_{led} è la corrente che vogliamo far scorrere nel led, e la poniamo pari ad esempio a *0,015 A*. La V_{led} invece è la tensione ai capi del led quando è acceso. Questo parametro dipende dal colore del led. I led rossi hanno il valore più basso (circa *1.8 V*), quelli gialli e verdi un valore di circa *2.2 V* mentre i led bianchi e blu un valore di *3.6 V*.

Progettino #1: pulsanti e LED: Pulsanti

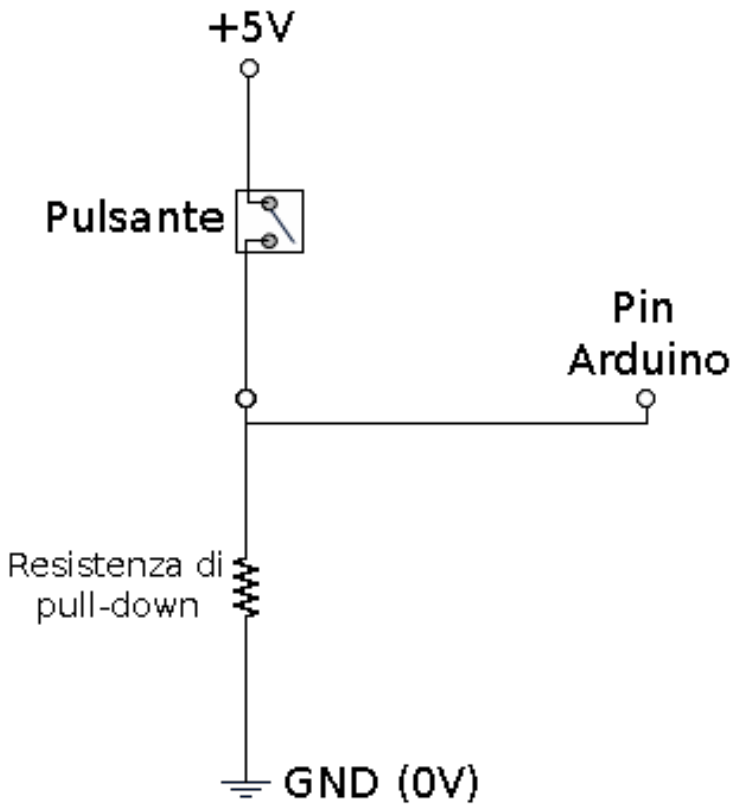
Un pulsante è un dispositivo elettromeccanico, che presenta un contatto elettrico che si chiude quando viene premuto.
Come possiamo collegare un pulsante a un microcontrollore?
Possiamo collegare un pin del pulsante a $5\ V$ e l'altro pin collegarlo a un GPIO di ingresso dell' Arduino. Sembra tutto a posto, ma in realtà non funzionerebbe, perchè:

- Quando il pulsante è premuto, il GPIO è collegato a 5V, e `digitalRead()` ritornerebbe HIGH.
- Quando il pulsante non è premuto è un circuito aperto, e il GPIO non sarebbe collegato a niente.

Cosa succede quando un GPIO non è collegato a niente? Il valore è indeterminato, e `digitalRead()` potrebbe ritornare sia HIGH che LOW.

Progettino #1: pulsanti e LED: Pulsanti

Per risolvere il problema è necessario che quando il pulsante non è premuto arrivi comunque massa (*0 V*). Per farlo basta che colleghiamo allo stesso GPIO una *resistenza che vada a massa*. In questo modo quando premiamo il pulsante arrivano *5 V* che vanno sul GPIO e poi tramite la resistenza a massa; mentre quando non è premuto il pulsante arriva massa dalla resistenza.



Progettino #1: pulsanti e LED

Pratica: costruzione del circuito

Progettino #2: Comunicazione con il PC

Specifiche del progettino:

Usando il pulsante del circuito precedente scrivere un programma che comunica al PC, ogni secondo, lo stato del pulsante (premuto, non premuto).

Progettino #2: Comunicazione con il PC

Organizzazione del progettino

- Teoria: La Seriale
- Software: Arduino e la classe Serial
- Pratica: costruzione del circuito

Progettino #2: Comunicazione con il PC: La seriale

Per "seriale" in questo caso si intende lo standard RS232. E' uno standard molto antico, risalente agli anni '60 per collegare i terminali ai primi mainframe computer.

Negli anni '90 è stata usata ampiamente per collegare i modem 56K ai PC.

Oggi è sparita dai portatili, ma in ambito embedded è ancora usatissima.

Perchè?

Progettino #2: Comunicazione con il PC: La seriale

Perchè è semplice, praticamente tutti i microcontrollori la supportano, e scrivere un driver per la periferica seriale di un microcontrollore richiede circa 100 linee di codice C. In confronto, non tutti i microcontrollori supportano la USB, e comunque scrivere un driver USB richiede migliaia di righe di codice.

Per risolvere il problema che i PC non hanno più la seriale, esistono circuiti integrati che si presentano da un lato come USB, e dall'altro come seriale, convertendo i dati tra i due protocolli.

Quindi, anche se l'Arduino ha fisicamente una porta USB, in realtà quando si scrive il software sia lato PC che lato microcontrollore l'interfaccia è come se fosse una seriale.

Progettino #2: Comunicazione con il PC: Serial

Per usare la seriale da Arduino si fa ricorso alla classe Serial (vi avevo detto che i programmi Arduino sono programmi C++, no?)

Per chi non conosce i linguaggi a oggetti, gli oggetti sono agglomerati di dati e funzioni per accedere ai dati, detti metodi. Quando si chiama un metodo, bisogna specificare l'istanza dell'oggetto, ad esempio

```
Serial.begin(9600);
```

chiama il metodo (=funzione) begin della classe Serial.

Per chi conosce troppo bene C++, sa cos'è un metodo statico e si aspettava un :: da qualche parte, è vero, Serial non è una classe, è un istanza di classe dichiarata nel runtime Arduino.

Progettino #2: Comunicazione con il PC: Serial

I metodi più comunemente usati di Serial sono:

- `begin(<baudrate>)` Inizializza la seriale, <baudrate> è un numero che specifica la velocità della seriale. 9600 è un valore molto usato in ambiente Arduino
- `print(<value>)` Invia un valore attraverso la seriale, questo valore può essere visto dal PC o essere letto da un programma che gira sul PC. <value> può essere un numero, o una stringa.
- `println(<value>)` Esattamente come `print()`, ma invia un $\backslash n$
- `available()` ritorna il numero di caratteri che il PC ha inviato all'Arduino. Utile per inviare dati dal PC all'Arduino.
- `read()` Legge un carattere che il PC ha inviato all'Arduino.

Progettino #2: Comunicazione con il PC: Serial

Serial è per Arduino l'equivalente di `printf()` e `scanf()` per un programma C.

L'arduino non ha schermo e tastiera, quindi i dati vengono comunicati al PC.

Esistono dei programmi detti "terminal" emulators, come screen o minicom che sono usabili per "prendere in prestito" la tastiera e lo schermo del PC e usarli per comunicare con un microcontrollore. L'IDE Arduino ha un terminale integrato.

In aggiunta, si possono scrivere programmi lato PC che aprono la porta seriale e "parlano" con l'Arduino.

Progettino #2: Comunicazione con il PC

Pratica: costruzione del circuito

Progettino #3: ADC e PWM

Specifiche del progettino:

Collegare una fotoresistenza e un LED a un Arduino.
Scrivere un programma che, ogni 100ms, manda il valore di tensione della fotoresistenza. Osserviamo come cambiano i valori a seconda della luce che colpisce la fotoresistenza. Usiamo il LED in modo che sia acceso al 100% quando è buio, acceso al 50% quando c'è poca luce e spento quando c'è molta luce.

Progettino #3: ADC e PWM

Organizzazione del progettino

- Teoria: ADC
- Teoria: PWM
- Software: la funzione `analogRead()`
- Software: la funzione `analogWrite()`
- Hardware: la fotoresistenza
- Pratica: costruzione del circuito

Progettino #3: ADC e PWM: ADC

L'ADC è un componente con un ingresso analogico (tensione variabile tra un minimo e un massimo), e un uscita digitale a n bit che rappresenta la tensione in ingresso come un numero tra 0 e 2^n.

Il microcontrollore ATmega32 ha un ADC da 10bit. Il range di tensioni va da 0 a 5V

Progettino #3: ADC e PWM: PWM

Il PWM è una tecnica per emulare un segnale analogico avendo a disposizione solo un segnale digitale.

Il trucco consiste nel generare un onda quadra a duty cycle variabile. Il duty cycle è il rapporto tra il tempo in cui il segnale sta a livello logico 1 e il periodo dell'onda quadra.

La tensione media sarà pari all'integrale della forma d'onda in un periodo.

Questa tecnica funziona molto bene per LED e motori elettrici.

Progettino #3: ADC e PWM: `analogRead(<numero GPIO>)`

Per leggere un valore analogico con Arduino occorre:

- Collegare il segnale da leggere a degli ingressi specifici, segnati sulla board come "ANALOG IN". I segnali analogici possono essere connessi solo a questi piedini.
- Lato software, esiste la funzione `analogRead()`. Questa funzione prende come parametro il numero dell'ingresso analogico (quello riportato sulla board), e ritorna un numero tra 0 e 1023.
  0 significa 0V, 1023 significa 5V.

Progettino #3: ADC e PWM: `analogWrite()`

Il PWM è disponibile solo su alcuni dei GPIO *digitali* di Arduino. A seconda della board sono indicati con la scritta "PWM" o con un "~".

Il PWM si attiva con la funzione `analogWrite(<numero GPIO>, <valore>)` dove <numero GPIO> è un GPIO che supporta il PWM, mentre <valore> è un numero tra 0 e 255. Con il valore 0 il GPIO rimane sempre a livello logico 0, con 255 rimane sempre a livello logico 1, con un valore intermedio viene generata un'onda quadra a circa 490Hz con un duty cycle proporzionale al valore.

Progettino #3: ADC e PWM: la fotoresistenza

La fotoresistenza è un componente che varia la sua resistenza a seconda della luce. Al buio ha una resistenza molto alta (circa 1MOhm), mentre quando viene illuminata da una forte luce ha una resistenza molto bassa (circa 100Ohm).

Per sapere quanta luce è presente è possibile quindi fare un *partitore* con una fotoresistenza e una normale resistenza. Per farlo basta collegare un capo della fotoresistenza a +5V, e collegare una resistenza tra l'altro capo della fotoresistenza e massa.

La tensione nel punto centrale del partitore sarà circa 0V al buio, e circa 5V in pena luce.

Progettino #3: ADC e PWM

Pratica: costruzione del circuito

Progettino #4: Display LCD

Specifiche del progettino:

Collegare un LCD da 2 righe per 16 caratteri compatibile HD44780 ad Arduino. Stampare “Hello world” sulla prima riga. Sulla seconda riga visualizzare il numero di secondi trascorsi dall’avvio di Arduino sullo schermo.

Progettino #4: Display LCD

Organizzazione del progettino

- Teoria: I display LCD compatibili HD44780.
- Software: Arduino e la classe LiquidCrystal
- Pratica: costruzione del circuito

Progettino #4: Display LCD: HD44780

HD44780 è uno standard *de facto* per i display LCD a caratteri (ossia non in grado di mostrare immagini). Il nome deriva dal primo circuito integrato ad implementare questo protocollo.

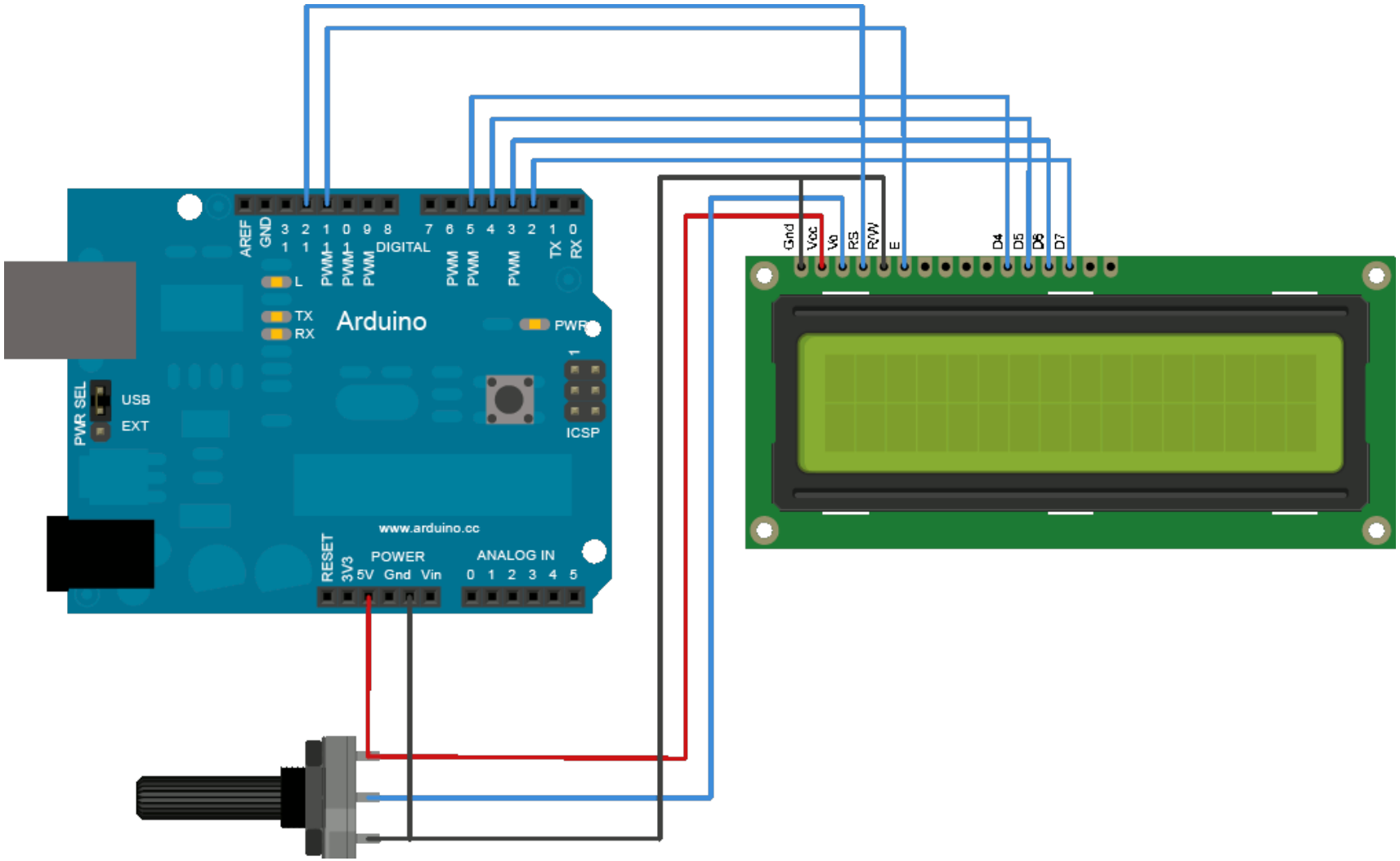
Trivia: HD44780 è anche il nome di una stella, o per la precisione di un sistema binario, due stelle che ruotano attorno a un asse comune.

Progettino #4: Display LCD: HD44780

Questi LCD hanno 14 o 16 piedini, a seconda della presenza di una retroilluminazione per renderli visibili al buio.

- 1 Da collegare a massa
- 2 Da collegare a +5V
- 3 Contrasto, collegare un trimmer tra +5V e massa
- 4 Register Select, da collegare a un GPIO digitale dell'Arduino
- 5 Read/Write (R/W), da collegare a massa (write)
- 6 Clock (Enable), da collegare a un GPIO digitale dell'Arduino
- 7 Bit 0 non collegare
- 8 Bit 1 non collegare
- 9 Bit 2 non collegare
- 10 Bit 3 non collegare
- 11 Bit 4, da collegare a un GPIO digitale dell'Arduino
- 12 Bit 5, da collegare a un GPIO digitale dell'Arduino
- 13 Bit 6, da collegare a un GPIO digitale dell'Arduino
- 14 Bit 7, da collegare a un GPIO digitale dell'Arduino

Progettino #4: Display LCD: Cablare il display



Progettino #4: LiquidCrystal

Per usare un display da Arduino si fa ricorso alla classe LiquidCrystal, le funzioni per poter utilzzare questi display sono

- `LiquidCrystal lcd(rs, enable, d4, d5, d6, d7)` Crea un oggetto per gestire un display collegato ai pin specificati.
- `begin(<column>, <row>)` Inizializza il display specificando il numero di colonne e righe.
- `print(<value>)` Invia una stringa (o un valore numerico) da visualizzare sul display.
- `setCursor(<column>, <row>)` Posiziona il cursore sulla colonna e la riga specificate.

Progettino #4: Display LCD

Pratica: costruzione del circuito