ARDUINO WORKSHOP

Bologna, 30 Maggio 2017

Relatore: Ing. Paolo Guidorzi

ARDUINO WORKSHOP

Prima parte

- Introduzione: cos'è Arduino, la storia
- Sistemi embedded, microcontrollori, microprocessori, DSP, FPGA, computer, sensori, attuatori, domotica..
- Legge di Ohm, resistenze, LED, operazionali, sensori, protocolli I^2C e SPI
- Arduino: l'hardware, il firmware, il software (lo «sketch»), la *community*
- Arduino e il mondo esterno: porte analogiche e digitali, l'interfaccia seriale
- Il linguaggio di programmazione di Arduino e il suo ambiente di sviluppo

Seconda parte

- I primi esperimenti, breadboard e millefori, Arduino Playground
- Lettura del valore di un potenziometro
- Dal valore di un potenziometro all'uscita PWM – luminosità di un LED
- Uscita PWM continuamente variabile
- Dal segnale PWM a una tensione continua. «Poor man DAC»
- Utilizzo di un pulsante. Resistenze di pull-up e pull-down
- Il partitore di tensione
- Utilizzo di sensori di tipo resistivo. La fotocellula
- Utilizzo di sensori di tipo resistivo. Sensore di GAS

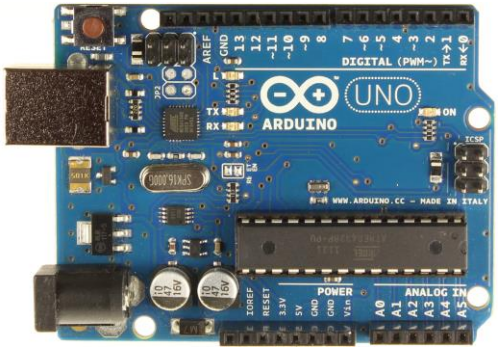
- Display a 7 segmenti (seriale)
- Display grafico TFT a colori (2.8'')
- Collegamento di un altro display grafico TFT (1.8'')
- Collegamento di un altro display grafico TFT (2.2'') usando Arduino MEGA2560
- Convertitori DAC e ADC esterni
- Sensore di temperatura e pressione
- Comunicazione dati da Arduino al computer (tramite porta seriale)
- Comunicazione dati da smartphone ad Arduino tramite Bluetooth
- Utilizzo di un Multiplexer

Terza parte

- Una realizzazione completa: il naso elettronico
- Display grafico
- ADC 16 bit
- Sensore pressione temperatura umidità
- Salvataggio dati su microSD e orologio real-time
- Multiplexer
- Sensori di gas

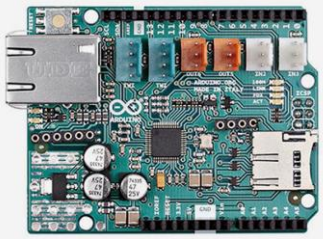
ARDUINO WORKSHOP

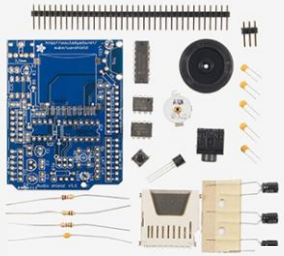
- Arduino è uno strumento di prototipazione rapida che permette di creare piccoli sistemi interattivi stand-alone.
- E' stato creato per artisti, progettisti, studiosi, ricercatori o chiunque abbia bisogno di uno strumento di lavoro per una specifica applicazione.
- Sia l'hardware che il software di Arduino sono open source, come la maggior parte dei progetti già pronti che si possono trovare liberamente online (sui siti www.arduino.cc e www.arduino.org o in altri siti creati dagli utilizzatori)
- E' facile da usare, non occorre essere ingegneri elettronici



ARDUINO WORKSHOP

- Si programma tramite computer con un linguaggio simile al C
- Ha vari ingressi e uscite (digitali e analogici)
- E’ espandibile utilizzando i cosiddetti *Shield*, che ne ampliano le possibilità di utilizzo in modo immediato
- Oppure si possono inventare o creare espansioni per usi particolari

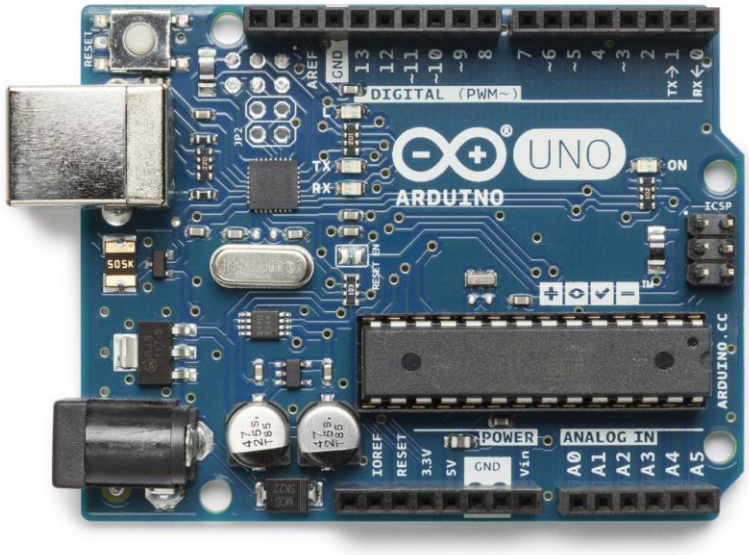




ARDUINO WORKSHOP

Esistono molte versioni di Arduino, con diverso numero di ingressi e uscite e diverse *potenze* di calcolo. Tutti condividono lo stesso linguaggio e ambiente di programmazione.

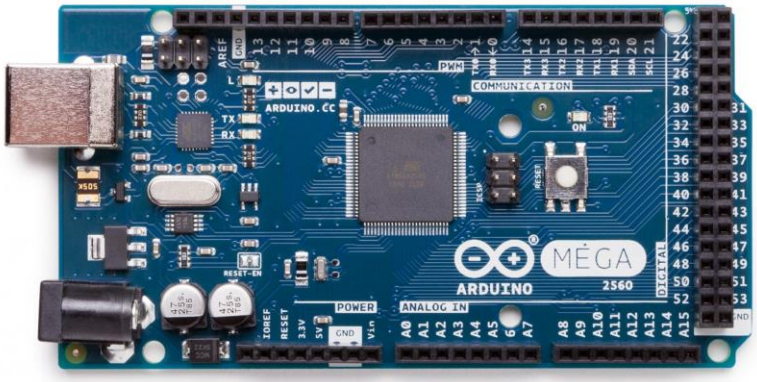
Arduino UNO:



Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

ARDUINO WORKSHOP

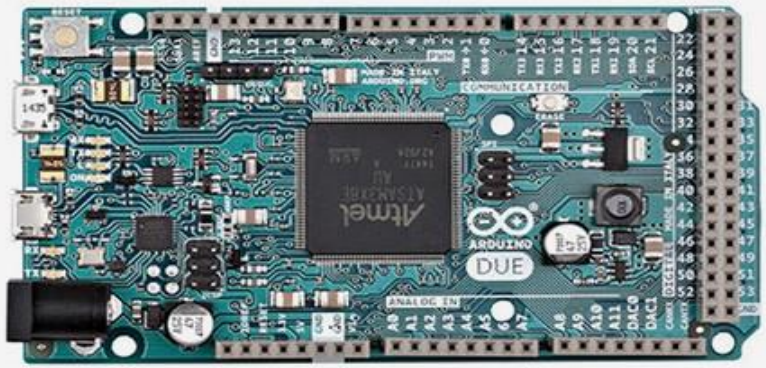
Arduino MEGA256:



Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

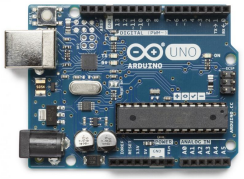
ARDUINO WORKSHOP

Arduino DUE:

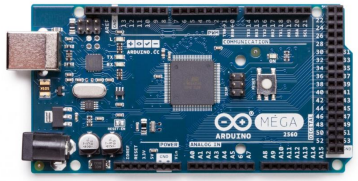


Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

ARDUINO WORKSHOP

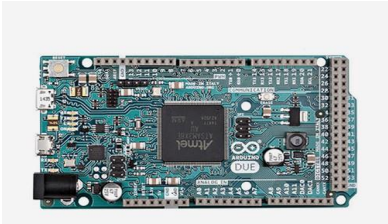


Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

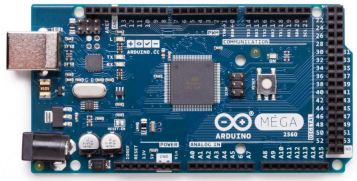


Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

ARDUINO WORKSHOP

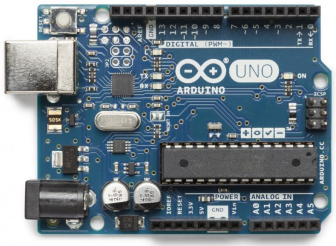


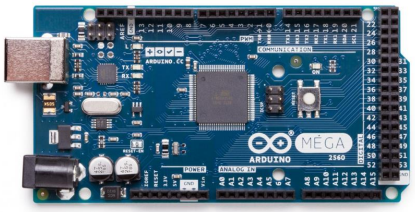
Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

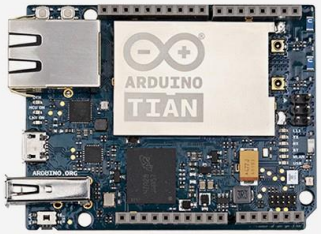


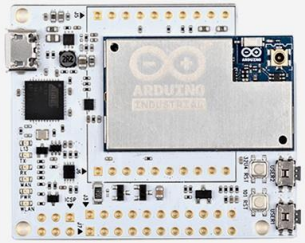
Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

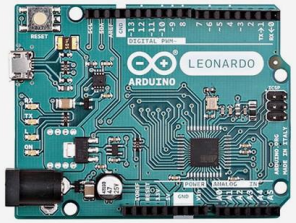
ARDUINO WORKSHOP

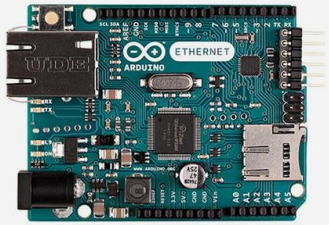


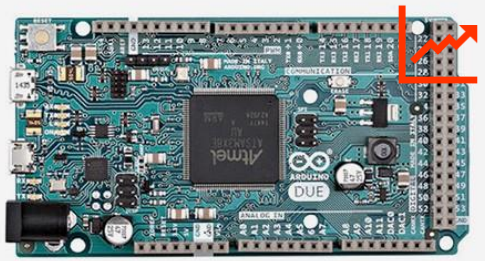












ARDUINO WORKSHOP

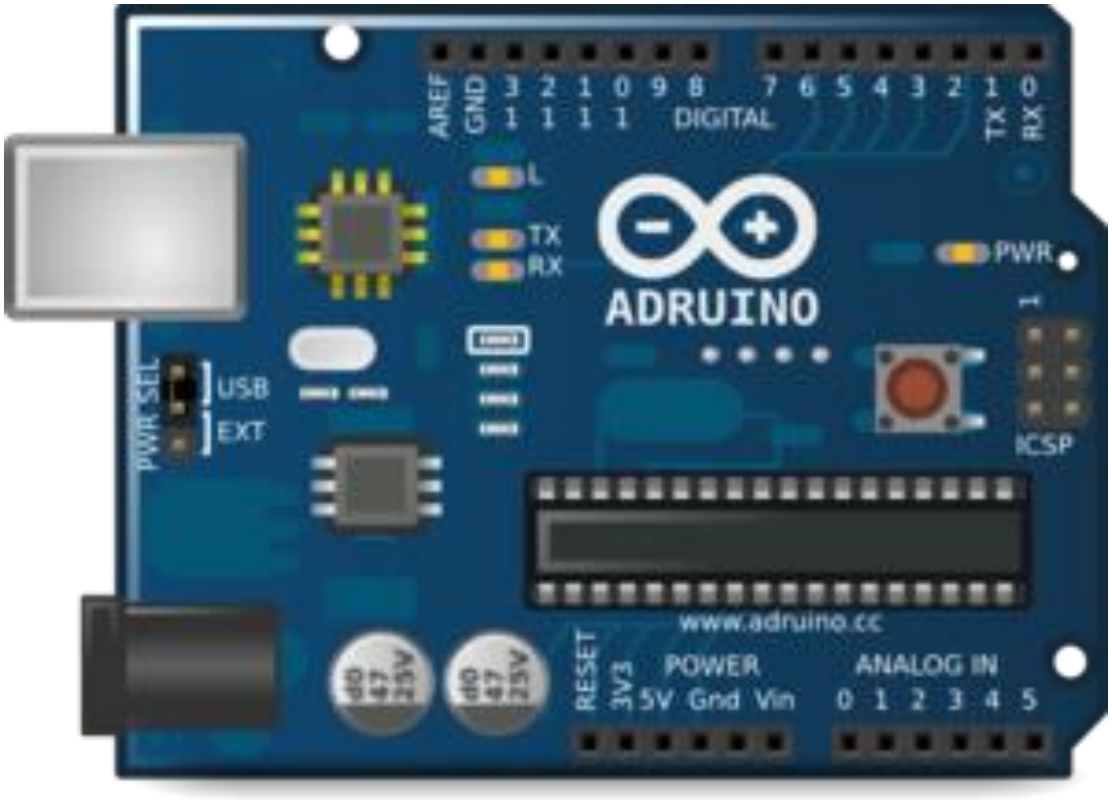
Compare board specs

This simple table shows a quick comparison between the characteristics of all the Arduino boards.

Name	Processor	Operating Voltage/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [KB]
Uno	ATmega328	5 V/7-12 V	16 Mhz	01/06/00	14/06/14	1
Due	AT91SAM3X 8E	3.3 V/7-12 V	84 Mhz	12/02/14	54/12	-
Leonardo	ATmega32u4	5 V/7-12 V	16 Mhz	01/12/00	20/07/14	1
Mega 2560	ATmega2560	5 V/7-12 V	16 Mhz	16/0	54/15	4
Mega ADK	ATmega2560	5 V/7-12 V	16 Mhz	16/0	54/15	4
Micro	ATmega32u4	5 V/7-12 V	16 Mhz	01/12/00	20/07/14	1
Mini	ATmega328	5 V/7-9 V	16 Mhz	01/08/00	14/06/14	1
Nano	ATmega168 ATmega328	5 V/7-9 V	16 Mhz	01/08/00	14/06/14	0.512 1
Ethernet	ATmega328	5 V/7-12 V	16 Mhz	01/06/00	14/04/14	1
Esplora	ATmega32u4	5 V/7-12 V	16 Mhz	-	-	1
ArduinoBT	ATmega328	5 V/2.5-12 V	16 Mhz	01/06/00	14/06/14	1
Fio	ATmega328P	3.3 V/3.7-7 V	8 Mhz	01/08/00	14/06/14	1
Pro (168)	ATmega168	3.3 V/3.35-12 V	8 Mhz	01/06/00	14/06/14	512
Pro (328)	ATmega328	5 V/5-12 V	16 Mhz	01/06/00	14/06/14	1
Pro Mini	ATmega168	3.3 V/3.35-12 V 5 V/5-12 V	8 Mhz 16Mhz	01/06/00	14/06/14	512
LilyPad	ATmega168V ATmega328V	2.7-5.5 V/2.7-5.5 V	8 Mhz	01/06/00	14/06/14	512
LilyPad USB	ATmega32u4	3.3 V/3.8-5V	8 Mhz	01/04/00	09/04/14	1
LilyPadSimple	ATmega328	2.7-5.5 V/2.7-5.5 V	8 Mhz	01/04/00	09/04/14	1
LilyPadSimpleS	ATmega328	2.7-5.5 V/2.7-5.5 V	8 Mhz	01/04/00	09/04/14	1

ADRUINO WORKSHOP

Hardware e software open source..



ARDUINO WORKSHOP

Arduino è nato nel 2005 da un'altra piattaforma per prototipazione semplificata, Wiring, creata per non esperti, da Hernando Barragan, di cui Massimo Banzi, creatore di Arduino, era relatore.

Massimo Banzi, David Cuartielles, David Mellis, Tom Igoe, Gianluca Martino

Il nome Arduino nasce dal nome di una caffetteria di Ivrea dove il team si ritrovava nel tempo libero. Arduino d'Ivrea è stato re d'Italia dal 1002 al 1014

Chi sono gli utilizzatori di Arduino?

*I **Maker** sono personaggi interessanti: non sono nerd, anzi sono dei tipi piuttosto fighi che si interessano di tecnologia, design, arte, sostenibilità, modelli di business alternativi. Vivono di comunità online, software e hardware open source ma anche del sogno di inventare qualcosa da produrre autonomamente, per vivere delle proprie invenzioni. In un momento di crisi si inventano il loro lavoro invece che cercarne uno classico.*

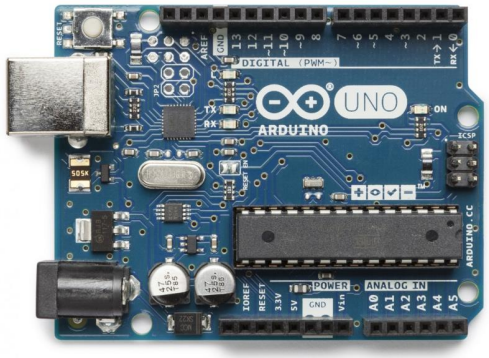
(da un'intervista a Massimo Banzi su *Wired*)

- **Computer**: general purpose, adatto per diverse esigenze. Fa di tutto senza essere ottimizzato per un compito particolare
- **Microprocessore**: unità di elaborazione dati (generica). Richiede elementi esterni (memoria, periferiche, bus, ecc..) per funzionare. Nei computer c'è un microprocessore
- **DSP**: microprocessore specializzato in operazioni matematiche relative al signal processing
- **FPGA**: array di porte logiche programmabili, per certi versi simile al DSP
- **Microcontrollore**: chip che include tutti gli elementi principali per funzionare (memoria volatile e non, bus, ingressi e uscite, ecc..)
- **Sistema embedded**: dispositivo elettronico «intelligente» creato per una funzione precisa (es. un termostato, un telecomando). Creato ed ottimizzato per un solo compito
- **Arduino** utilizza un microcontrollore, quindi la scheda richiede pochi componenti aggiuntivi per funzionare (un quarzo, ovvero il clock di sistema, alcuni regolatori di tensione, un chip per comunicare col computer via USB e poco altro)

ARDUINO WORKSHOP

Dunque quando usare un dispositivo general purpose (computer) e quando uno specializzato (sistema embedded) come Arduino?

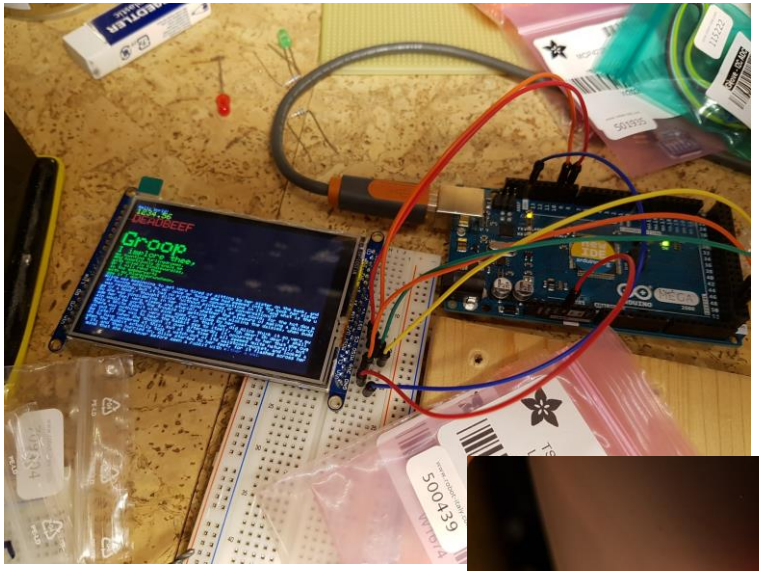
- Portabilità
- Esigenze particolari
- Miniaturizzazione
- Efficienza energetica



Altri esempi di piccoli sistemi embedded di uso comune tra hobbisti e sperimentatori:

Raspberry

PIC

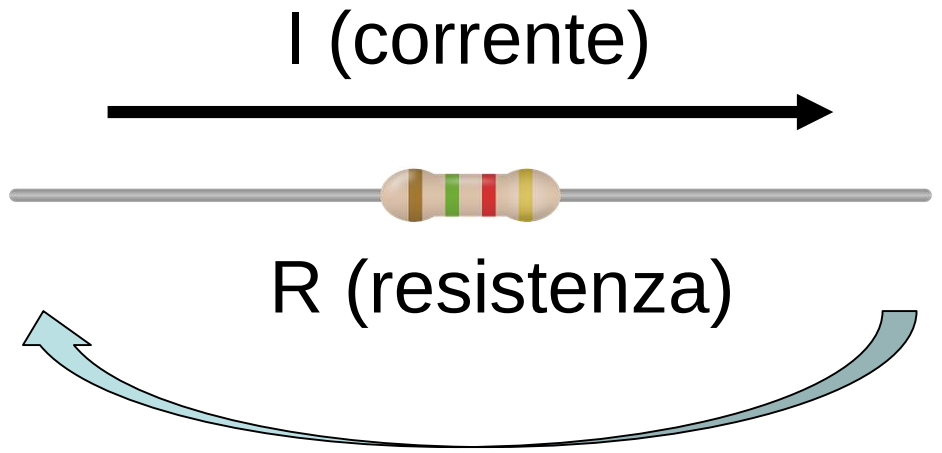

Groop

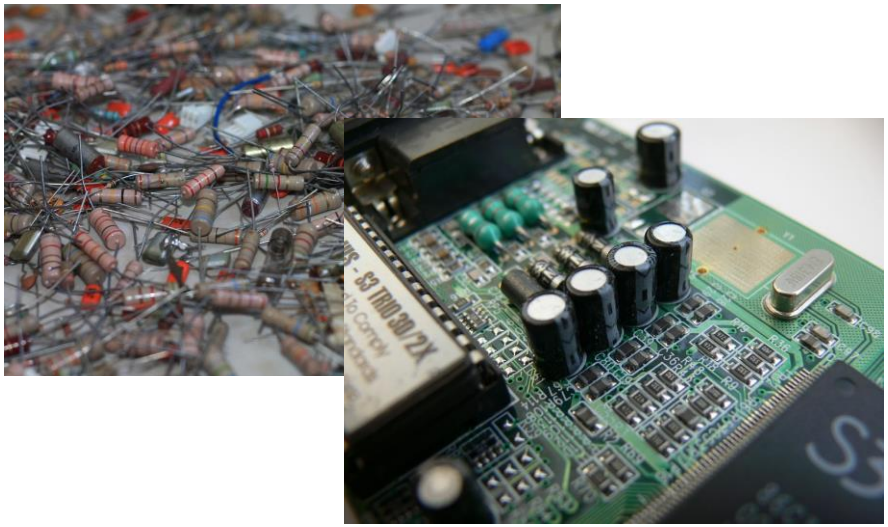
Legge di Ohm V = R * I

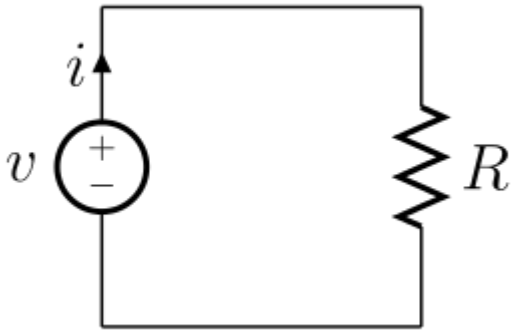
V in Volt

I in Ampere

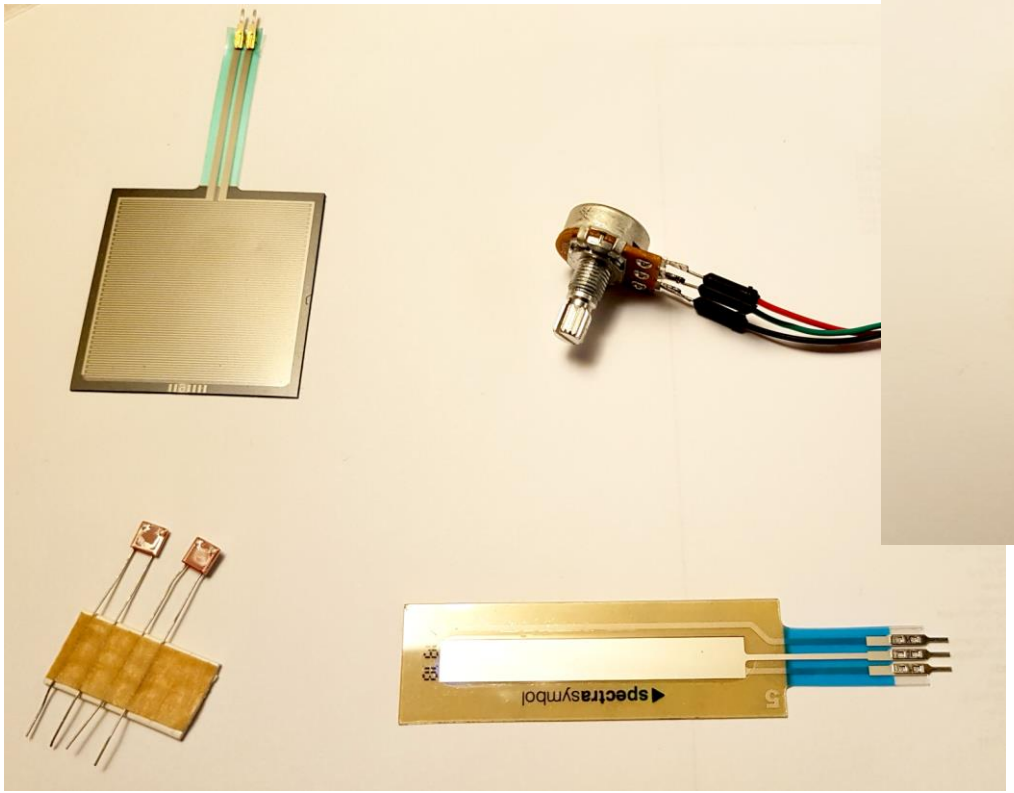
R in Ohm

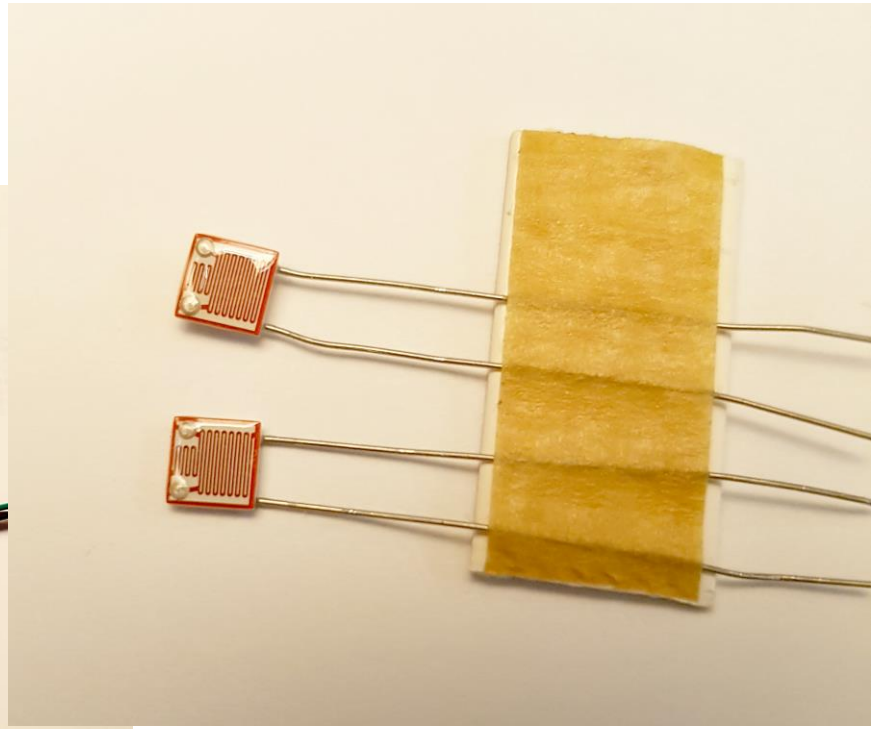






Resistenze variabili di vario tipo



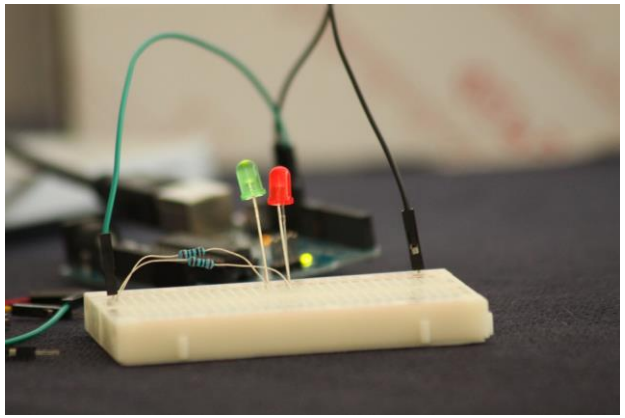


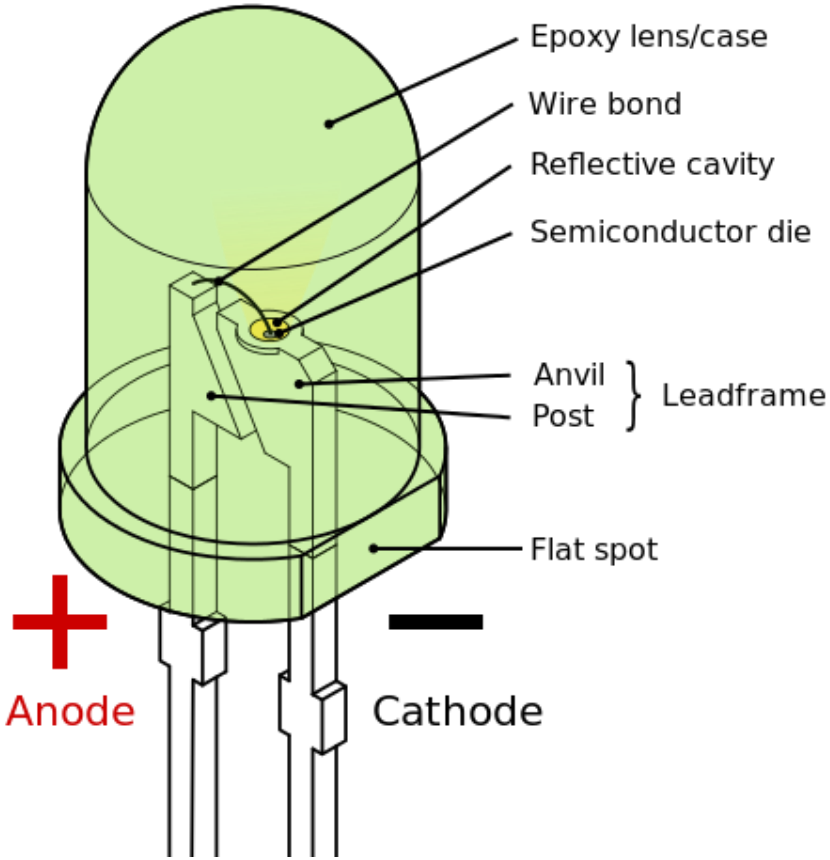
Sensore di forza, potenziometro,

fotocellula, sensore resistivo lineare

LED (Light-Emitting Diode)

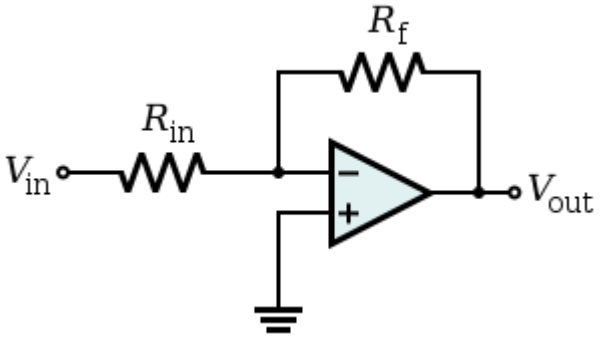
- E' un diodo
- La corrente lo attraversa solo in una direzione
- Occorre limitare la corrente massima con una resistenza





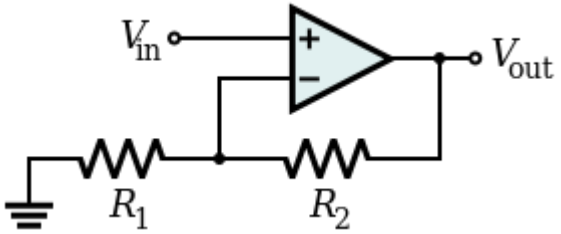
Amplificatori operazionali

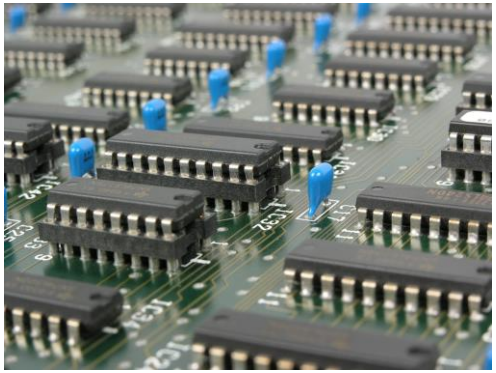
Configurazione invertente: $V_{out} = -\frac{R_f}{R_{in}} V_{in}$

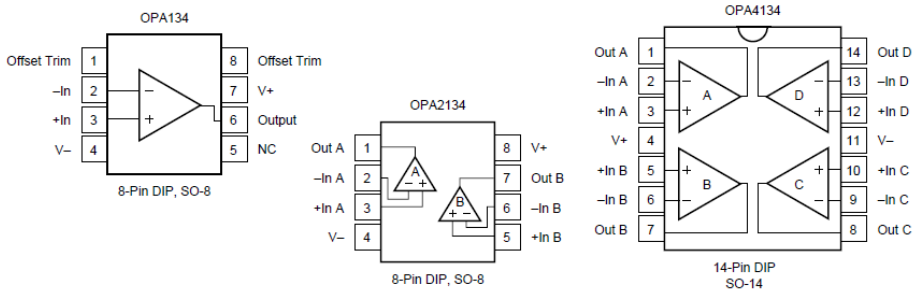


Configurazione non invertente: $V_{out} = \left(1 + \frac{R_2}{R_1}\right) V_{in}$

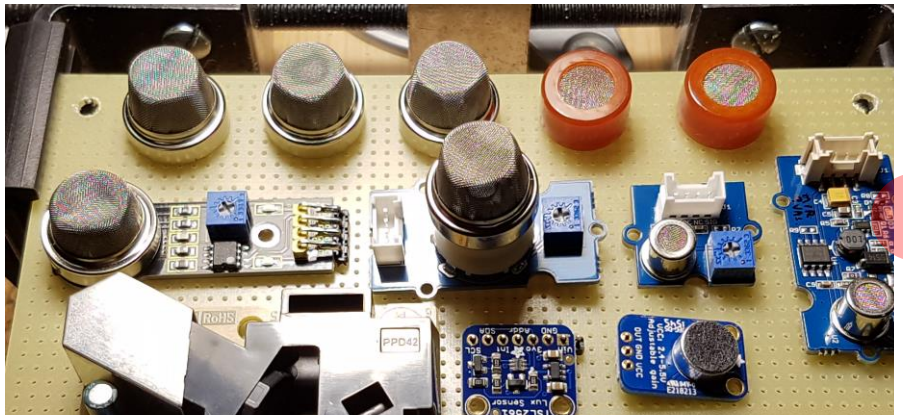
(se R1=infinito e R2=0 si ottiene un buffer a guadagno unitario)

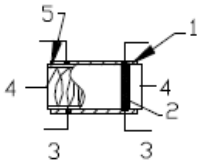


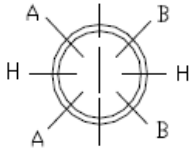


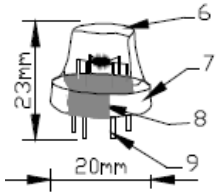


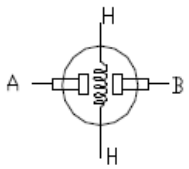
Sensori di gas (analogici)

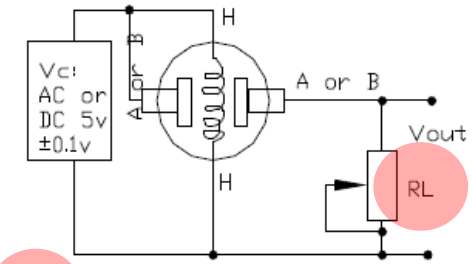








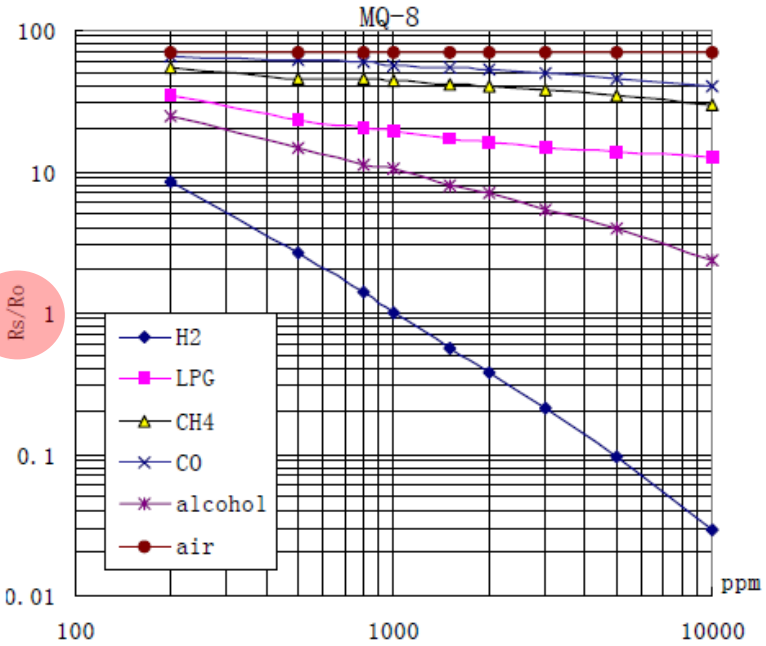


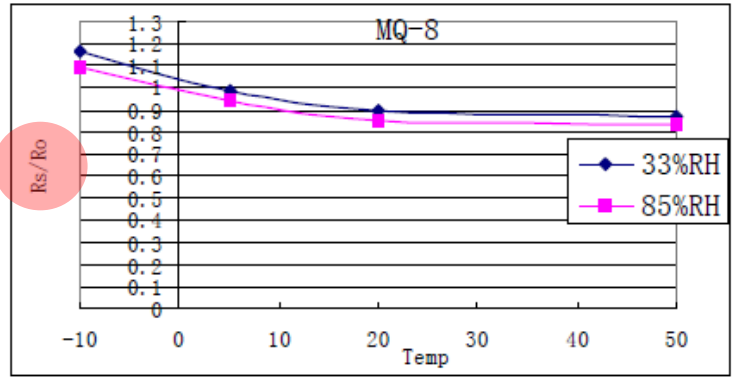


Ro: sensor resistance at 1000ppm of H_2 in the clean air.

Rs:sensor resistance at various concentrations of gases.

Rs	Sensing Resistance	10K Ω - 60K Ω (1000ppm H_2)





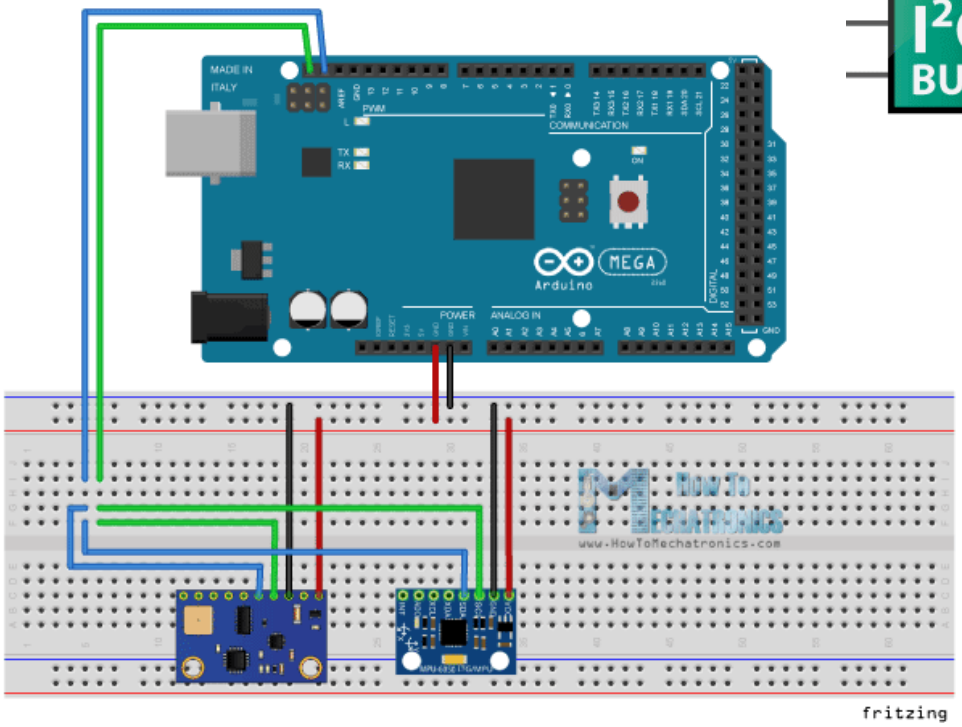
Dispositivi che comunicano in modo digitale

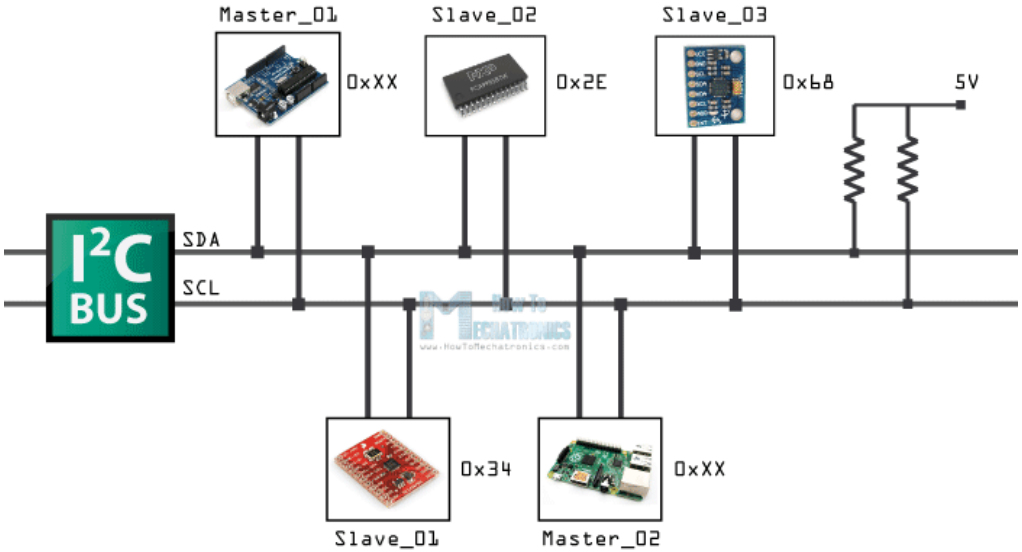
I protocolli più usati per la comunicazione tra dispositivi elettronici «intelligenti» all'interno di un circuito sono due: **I²C** e **SPI**

I²C: sviluppato a fine degli anni '70
bus a 2 fili: **SDA** (Serial Data line)
SCL (Serial Clock line)
i dispositivi sul bus si collegano a questi 2 fili

SPI: Serial Peripheral Interface
bus a 4 fili: **MOSI** (Master Out Slave In), **MISO** (Master In Slave Out), **SCK** (Clock), **SS** (Slave Select, **SS1**, **SS2**, .., **SSn**)

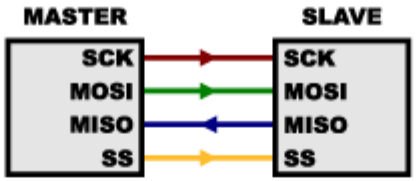
BUS I²C

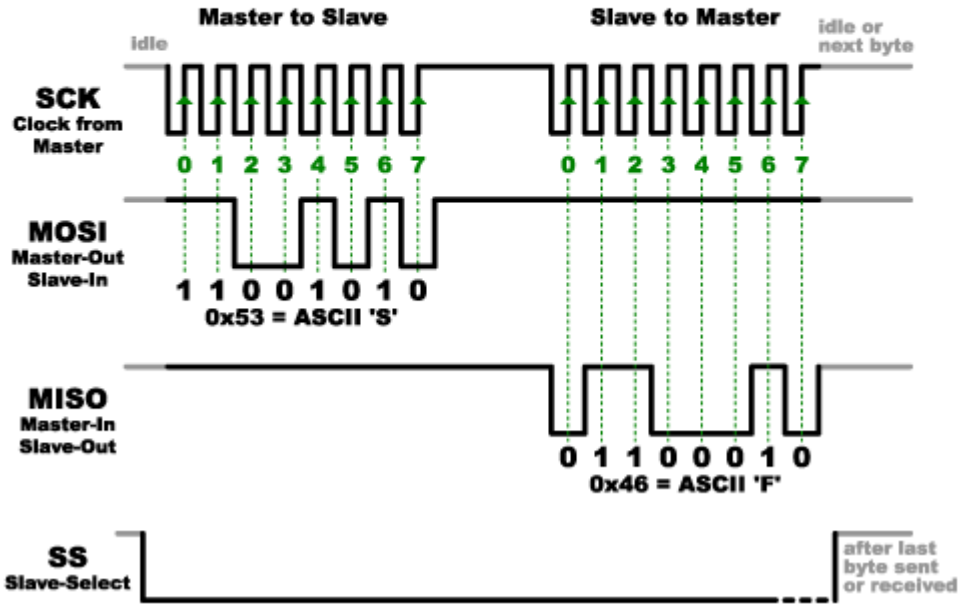


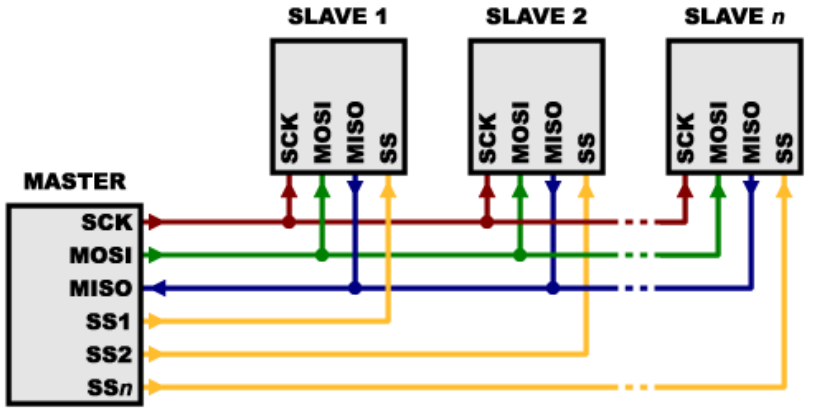


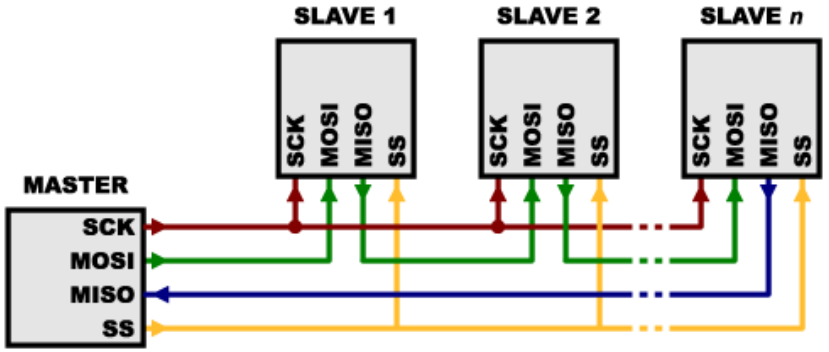
http://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/

BUS SPI









https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

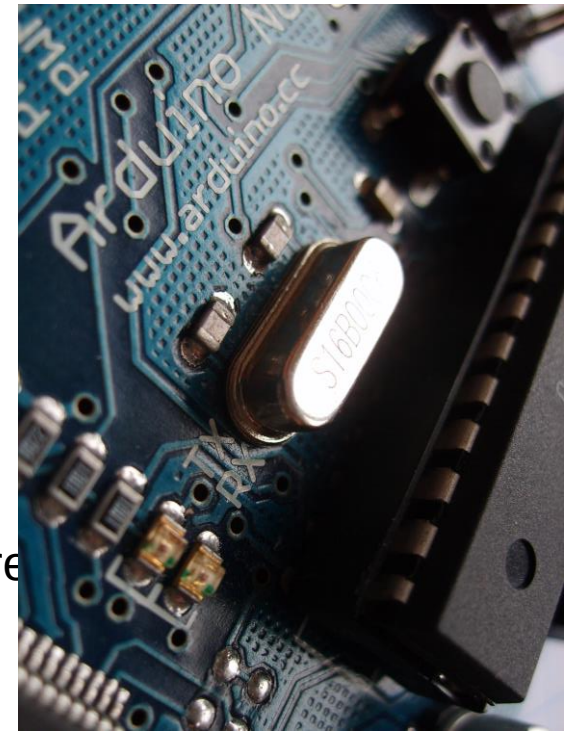
Confronto tra BUS I²C e BUS SPI

- Entrambi sono di tipo Master / Slave. Il Master inizia sempre la comunicazione

- **I²C** usa 2 fili: **SDA** (Serial Data line) e **SCL** (Serial Clock line).
  E' relativamente lento (100-400 kHz)
  Possono esserci multipli Master e Slave sulla linea

- **SPI** usa 4 (o più) fili: **MOSI** (Master Out Slave In), **MISO** (Master In Slave Out), **SCK** (Clock), **SS** (Slave Select, **SS1**, **SS2**, .., **SSn**).
  E' veloce, può arrivare a 25 MHz
  Un solo Master e multipli Slave

Arduino è composto di 3 elementi:

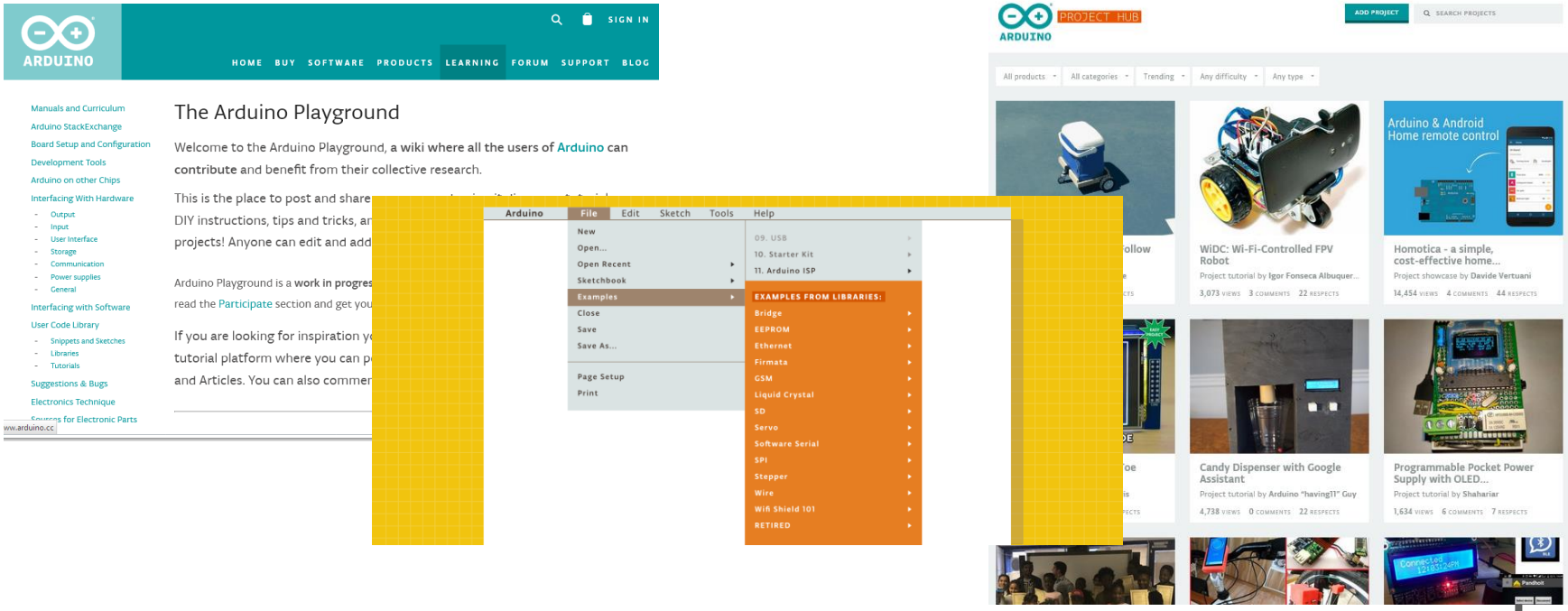


- Hardware
- Software
- Community

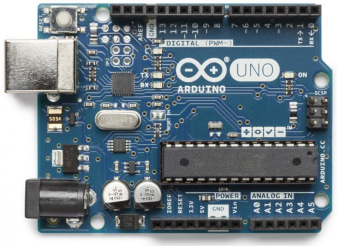
1. L'hardware, cioè le schede fisiche, possono variare in termini di numero di porte in/out, potenza del microcontrollore, ma si programmano tutte con lo stesso linguaggio (C semplificato) e tramite lo stesso ambiente di sviluppo.
2. Il software caricato sul microcontrollore è formato da 2 parti: un firmware, che resta residente e invariato (simile al BIOS del computer) ed esegue le funzioni di base, tra cui permettere comunicare col computer tramite porta USB e caricare il software sviluppato dall'utente, e il programma dell'utente («sketch»)

ARDUINO WORKSHOP

3. La community è il vero punto di forza di Arduino. Il repository di progetti di altri utenti e il forum del sito ufficiale sono ottimi punti di partenza per non partire *da zero.* La filosofia *open source* dell'intero ecosistema Arduino spinge e invita l'utilizzatore a condividere in modo aperto e gratuito (con vari tipi di licenza) i propri progetti con l'intera community. Inoltre l'ambiente di sviluppo (la «IDE» di Arduino) include già migliaia di esempi funzionanti.



Connessioni di Arduino UNO verso e dal mondo esterno

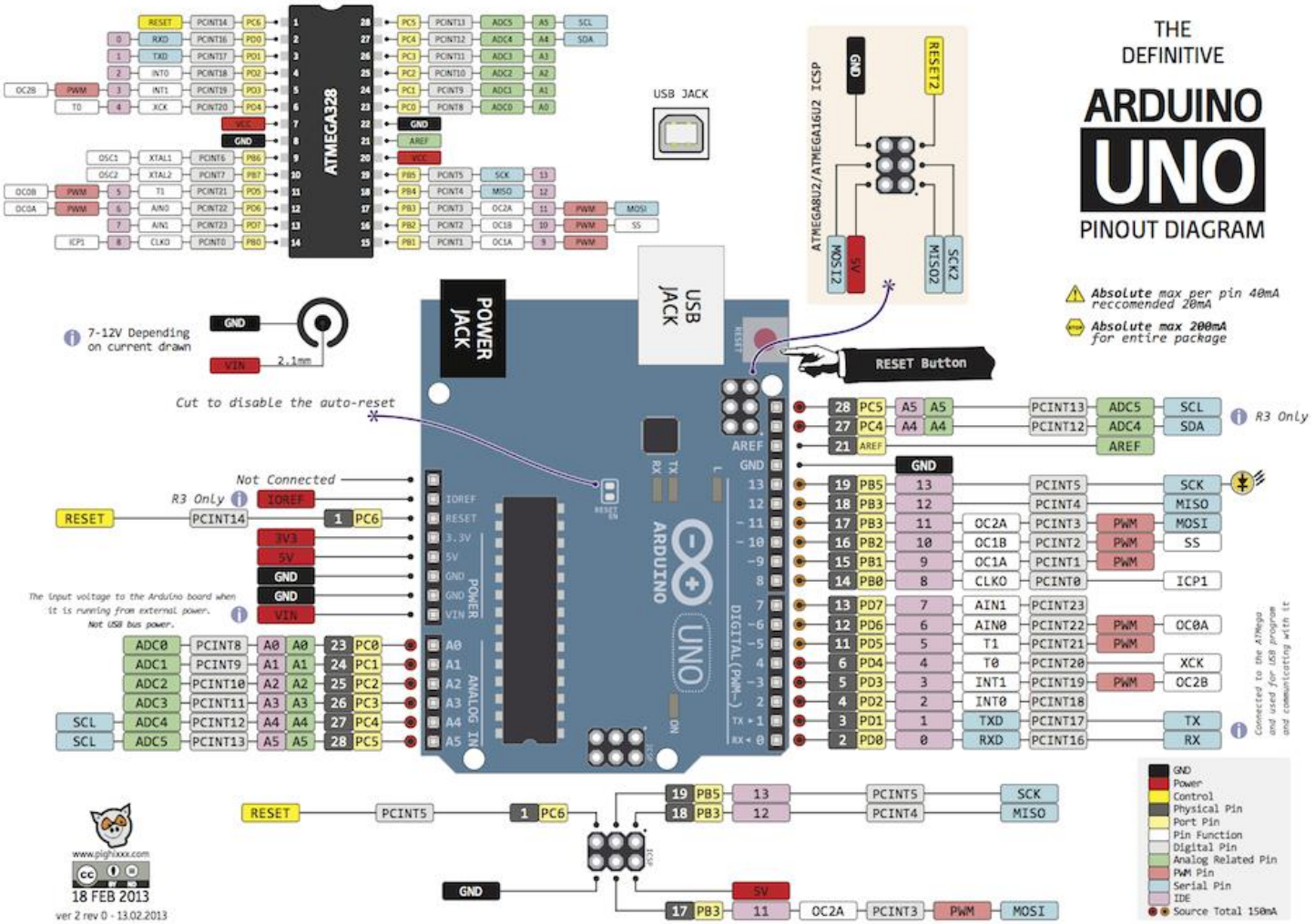


Atmega168 Pin Mapping

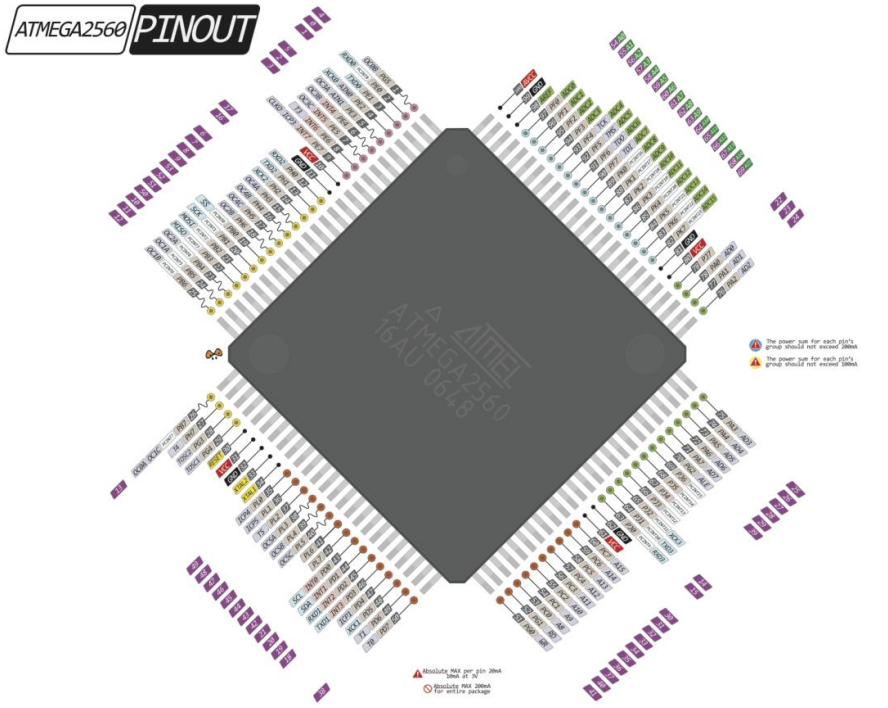
Arduino function				Arduino function
reset	(PCINT14/RESET) PC6	1	28 PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27 PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26 PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25 PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24 PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23 PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22 GND	GND
GND	GND	8	21 AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20 AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19 PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18 PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17 PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16 PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15 PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

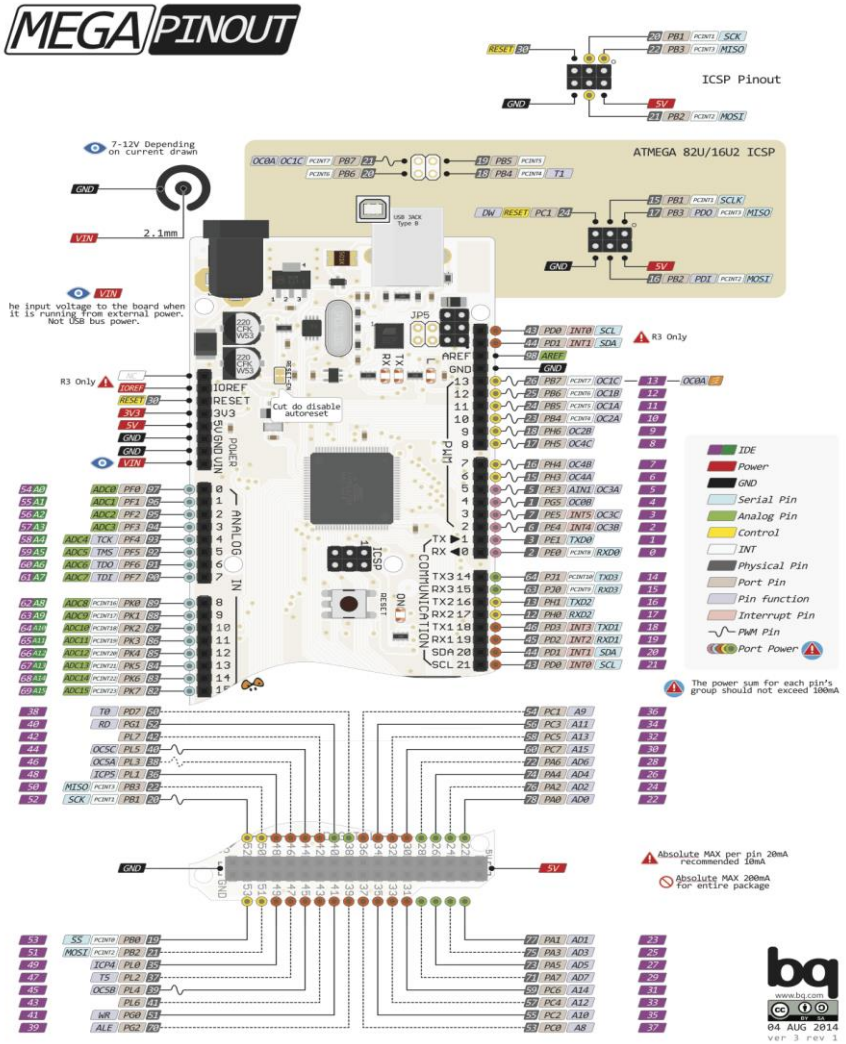
Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

ARDUINO WORKSHOP



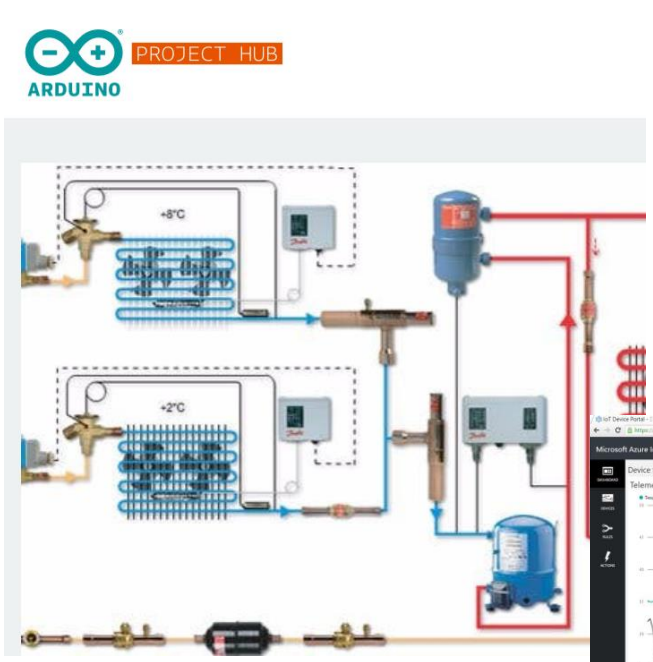
www.pighixxx.com

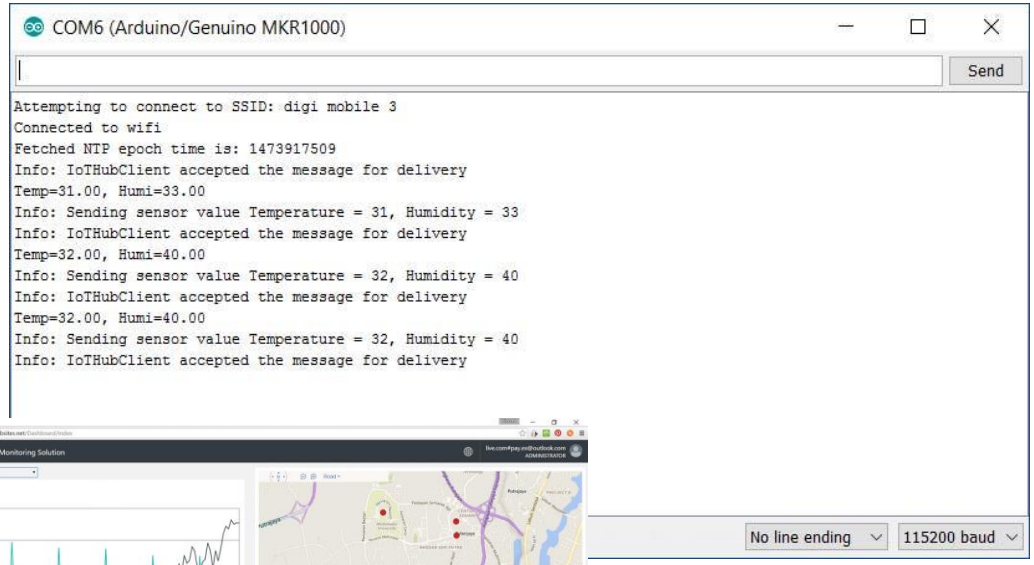

ATMEGA2560 PINOUT

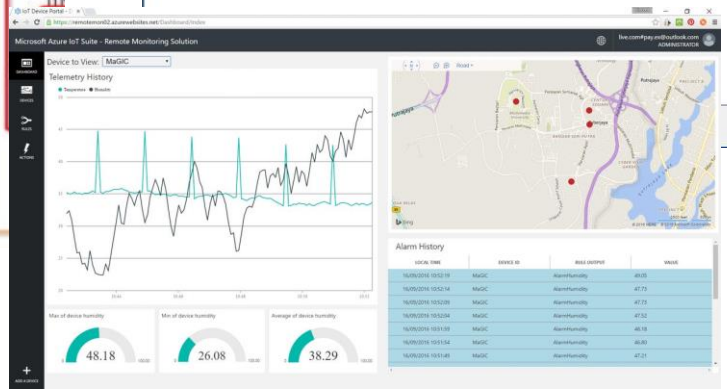

MEGA PINOUT
ICSP Pinout
ATMEGA 82U/16U2 ICSP
7-12V Depending on current drawn
VIN
2.1mm
Cut do disable autoreset
ANALOG IN
POWER
COMMUNICATION
Absolute MAX per pin 20mA
Absolute MAX 200mA for entire package
04 AUG 2014

ARDUINO WORKSHOP

Arduino, oltre a comunicare col mondo esterno attraverso le sue porte, analogiche e digitali (eventualmente connesse ad altri dispositivi come display, sensori, trasduttori, relè, ..), può anche scambiare dati col computer tramite la sua porta seriale (attraverso la USB). Ciò si usa spesso in fase di debugging dello *sketch*, per visualizzare valori di variabili o lo stato dell'elaborazione.



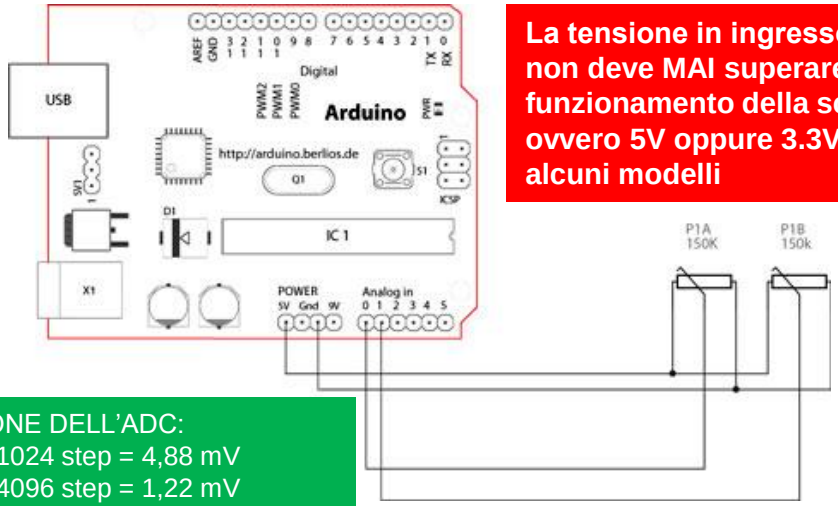




https://create.arduino.cc/projecthub/wesee/project-kool-temperature-and-humidity-remote-monitoring-e5ddae

Ingressi analogici di Arduino

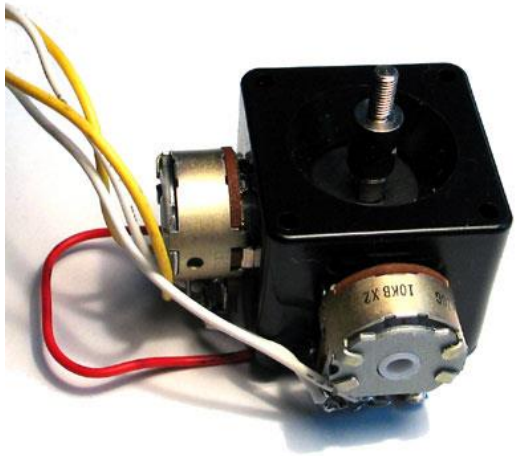
Arduino ha già alcuni ingressi *analogici*, ovvero connessi ad alcuni convertitori ADC (all'interno del microcontrollore), nel caso di UNO e MEGA, sono a 10 bit, quindi capaci di discretizzare una tensione in ingresso in 1024 intervalli.
In caso sia necessario avere una risoluzione maggiore, basta utilizzare un ADC esterno e farlo comunicare con Arduino con un bus I^2C oppure SPI



La tensione in ingresso all'ADC non deve MAI superare quella di funzionamento della scheda, ovvero 5V oppure 3.3V per alcuni modelli

RISOLUZIONE DELL'ADC:
10 bit: 5V / 1024 step = 4,88 mV
12 bit: 5V / 4096 step = 1,22 mV
16 bit: 5V / 65536 step = 0,07 mV
24 bit: 5V / 16777216 step ≈ 0,0003 mV

Interfacing a Joystick



Arduino DUE ha due ADC a 12 bit e due DAC a 12 bit nella scheda, grazie al microcontrollore più evoluto

- Teorema del campionamento
- Frequenza di Nyquist
- Filtri antialiasing

https://www.arduino.cc/en/Tutorial/JoyStick

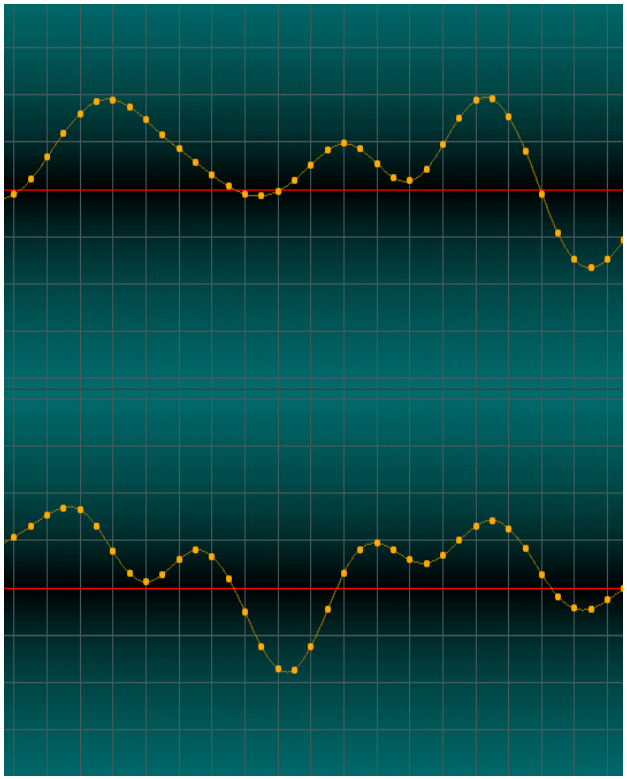
Ingressi e uscite digitali di Arduino

Tutti i modelli di Arduino presentano alcuni ingressi e uscite digitali e analogiche. Il numero e tipologia di questi ingressi dipende dal modello stesso. Per esempio:

Arduino UNO
- 14 porte digitali (6 PWM), configurabili come IN o OUT
- 6 convertitori A/D con risoluzione 10 bit (1024 valori)

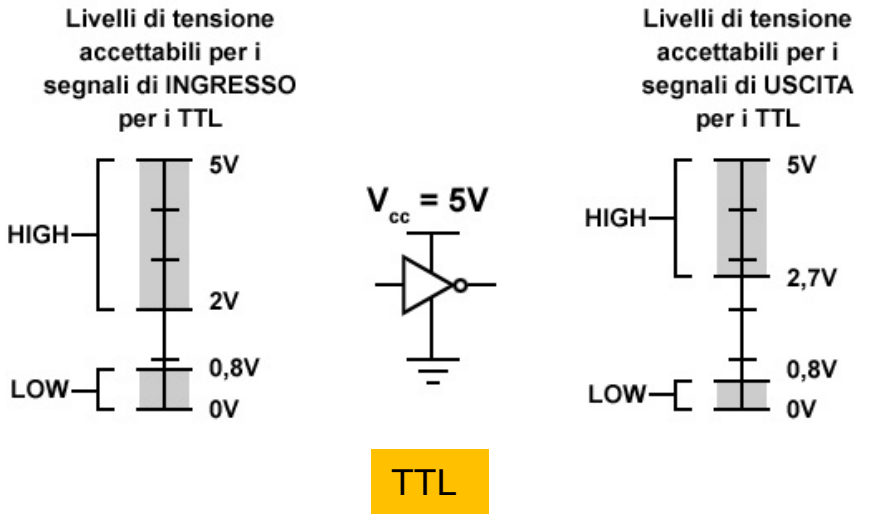
Arduino Mega256
- 54 porte digitali (15 PWM), configurabili come IN o OUT
- 16 convertitori A/D con risoluzione 10 bit (1024 valori)

Arduino DUE
- 56 porte digitali (12 PWM), configurabili come IN o OUT
- 12 convertitori A/D con risoluzione 12 bit (4096 valori)
- 2 convertitori D/A con risoluzione 12 bit (4096 valori)

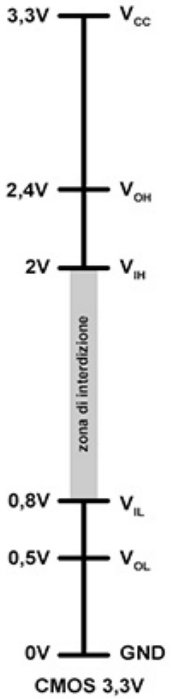


- Teorema del campionamento
- Frequenza di Nyquist
- Filtri antialiasing

Elettricamente, i livelli di ingresso e uscita digitali corrispondono a valori di tensione di 5 V (oppure 3.3 V per alcuni modelli di Arduino) per il livello HIGH e 0 V per il livello LOW. Esistono in realtà dei range di tolleranza sopra e sotto ai quali sono riconosciuti i valori logici HIGH e LOW:



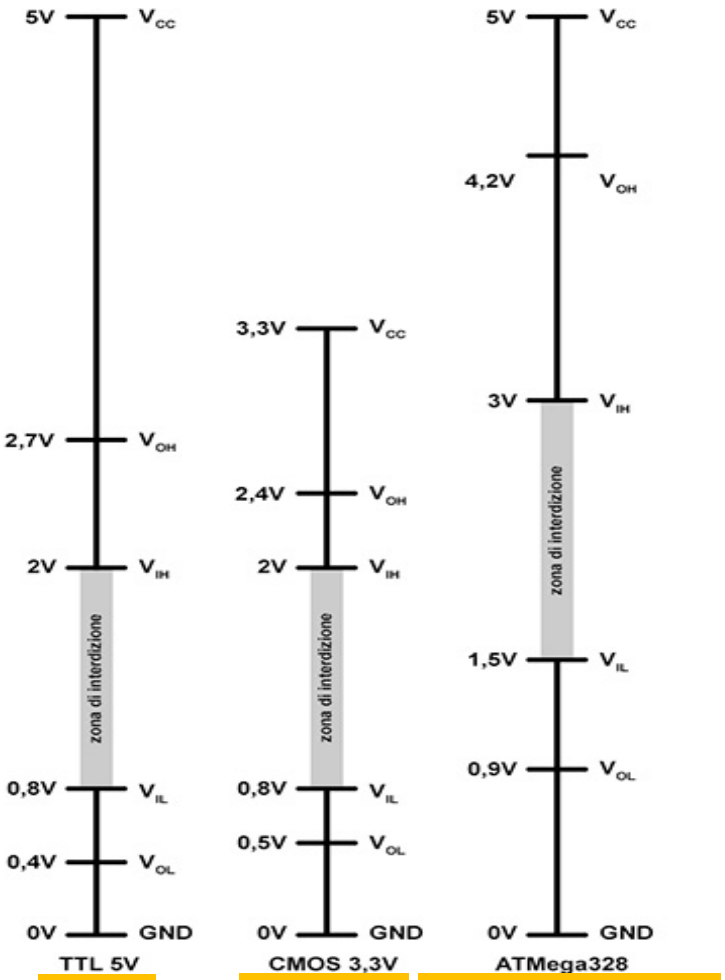
TTL



CMOS

ARDUINO (5V)

http://www.maffucci.it/2014/11/05/livelli-logici-ttl-e-cmos-cosa-si-nasconde-dietro-un-high-o-low-di-una-digitalwrite-di-arduino/



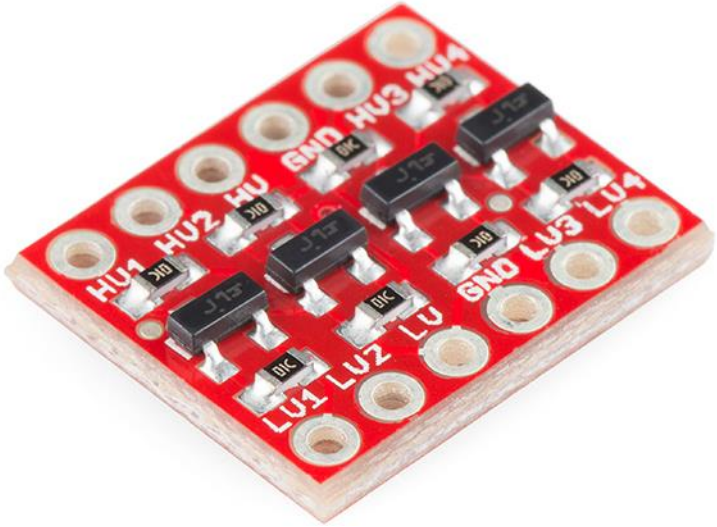
TTL CMOS ARDUINO (5V)

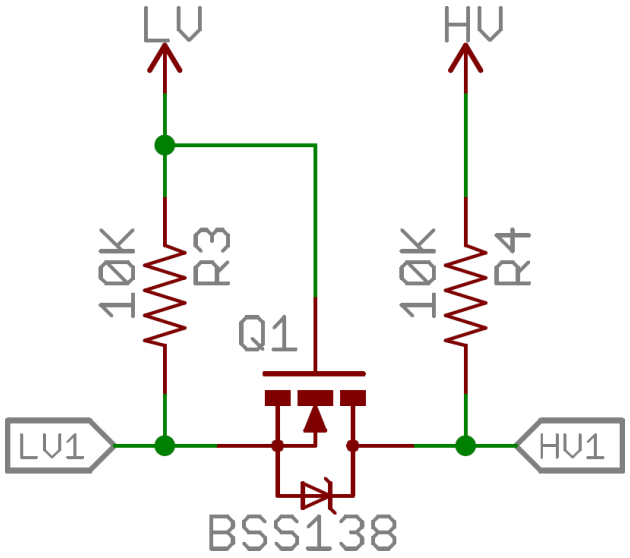
http://www.maffucci.it/2014/11/05/livelli-logici-ttl-e-cmos-cosa-si-nasconde-dietro-un-high-o-low-di-una-digitalwrite-di-arduino/

Conversione di livelli logici

Se si collegano ad Arduino dispositivi che non usano gli stessi livelli logici occorre effettuare un *level shifting,* che si può effettuare con appositi componenti o usando *shield* o schedine create apposta per questo compito. La conversione da 5 V a 3.3 V potrebbe essere effettuata anche con un partitore di tensione formato da 2 resistenze, ma chiaramente non il viceversa. Nota: i livelli logici di comunicazione, cioè le linee digitali, non necessariamente corrispondono alle tensioni di alimentazione del dispositivo.

Esistono componenti e shield che permettono la conversione di livello **bidirezionale** (BD-LLC):

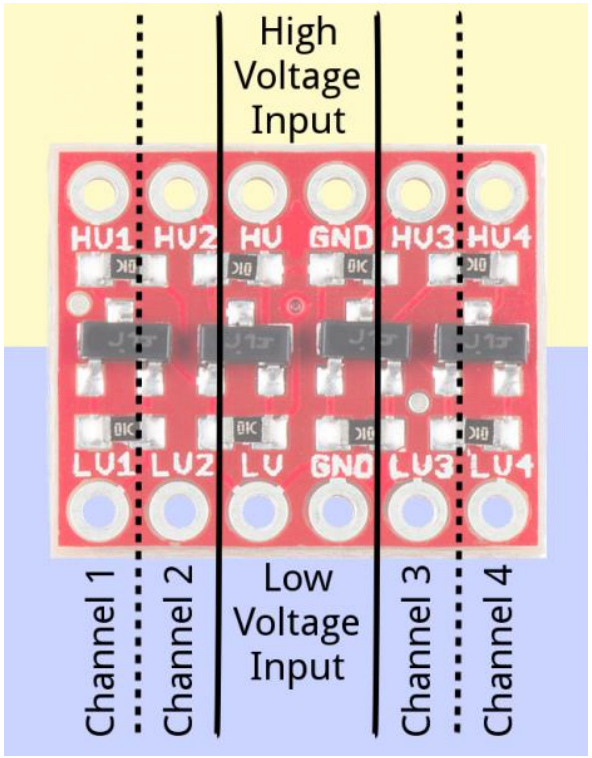




https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

Questa schedina (e altre simili) convertono i livelli da logica a 5 V a logica a 3.3 V e viceversa.

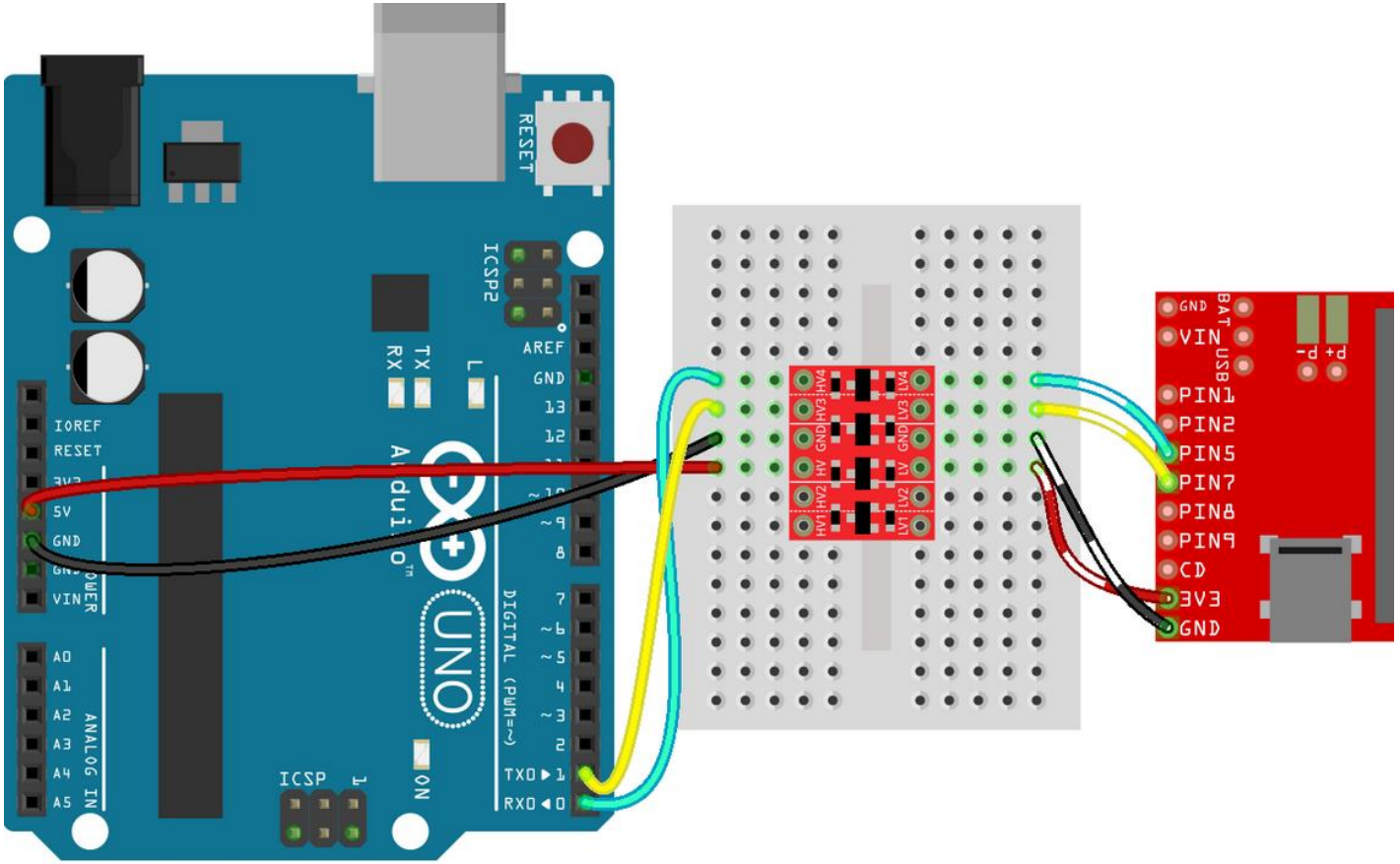
Occorre collegare alla schedina il valore di tensione alto (5 V), quello basso (3.3 V) e la massa. I livelli in logica 5 V in entrata dalle porte HVn saranno convertiti in livelli in logica 3.3 V alle porte LVn, oppure i livelli in logica 3.3 V in entrata dalle porte LVn saranno convertiti in livelli in logica 5 V alle porte HVn. In questa scheda ci sono 4 porte disponibili.



https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

ARDUINO WORKSHOP

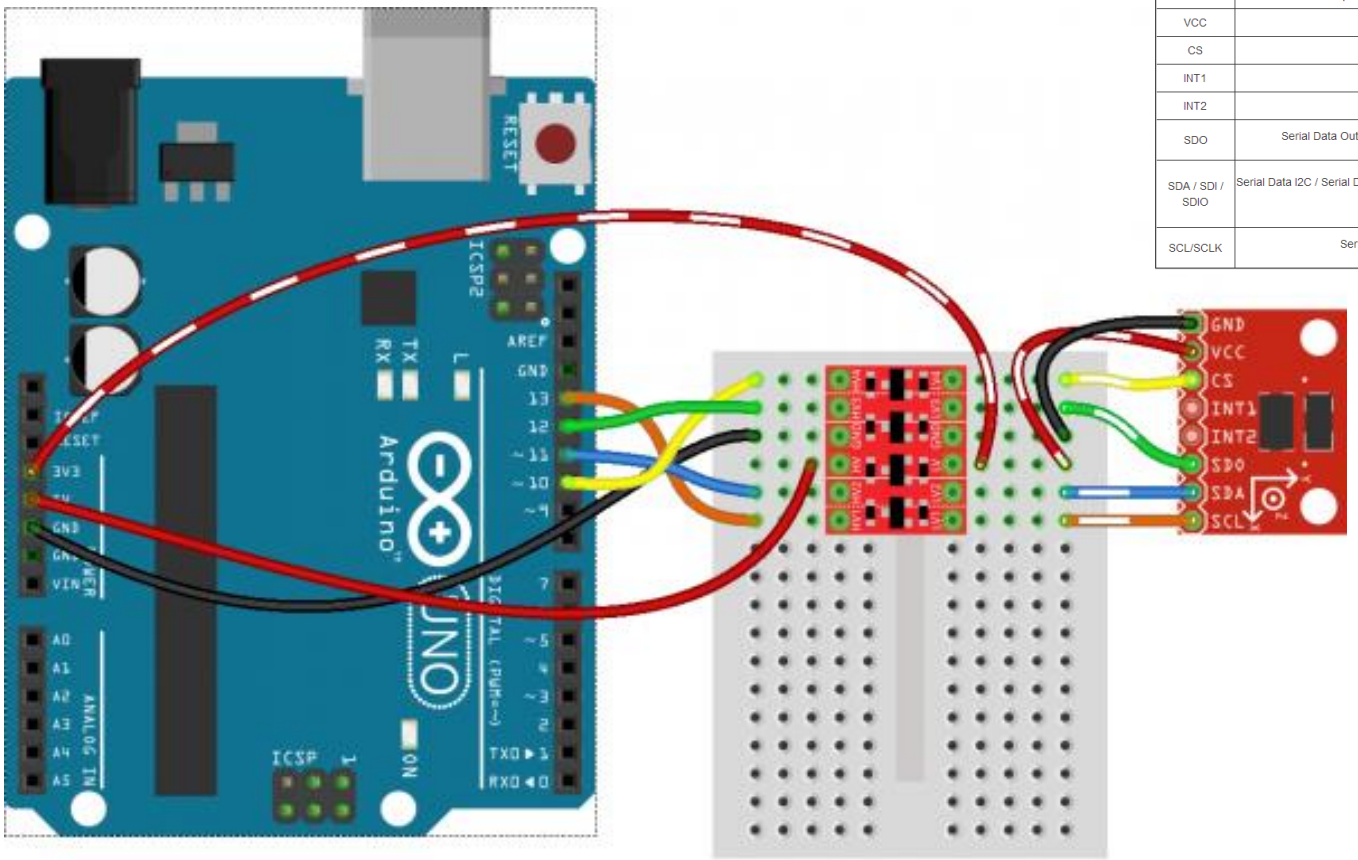
Esempio di collegamento tra Arduino UNO (a logica 5 V) con un dispositivo funzionante a logica 3.3 V. Comunicazione dalle porte seriali (TX0 e RX0). Si noti che anche Arduino ha un pin in cui sono disponibili direttamente i 3.3 V (per alimentare eventuali dispositivi che richiedono questa tensione)



https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

Esempio di collegamento tra Arduino UNO (a logica 5 V) con un dispositivo funzionante a logica 3.3 V. Comunicazione tramite protocollo SPI. Si usano 4 fili: MOSI (master out, slave in), MISO (master in, slave out), SCLK (serial clock) e CS (chip select).

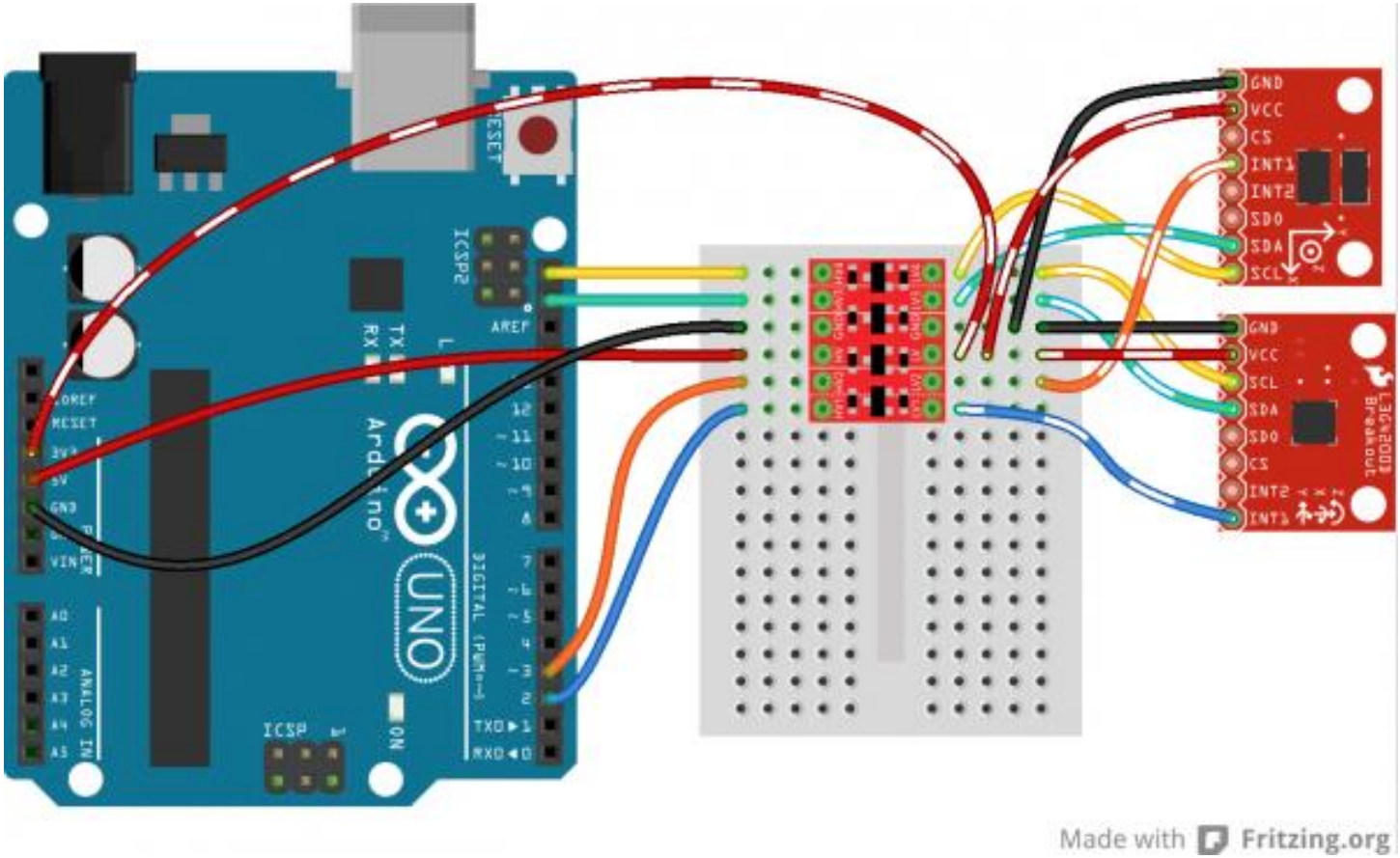
Mnemonic	Description
GND	This pin must be connected to ground
VCC	Supply Voltage
CS	Chip Select
INT1	Interrupt 1 Output
INT2	Interrupt 2 Output
SDO	Serial Data Output (SPI 4-Wire) / I2C Address Select
SDA / SDI / SDIO	Serial Data I2C / Serial Data Input (SPI 4-WIRE) / Serial Data Input and Output (SPI 3-Wire)
SCL/SCLK	Serial Communications Clock



https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

ARDUINO WORKSHOP

Esempio di collegamento tra Arduino UNO (a logica 5 V) con un dispositivo funzionante a logica 3.3 V. Comunicazione tramite protocollo I^2C. In questo caso i dati che transitano nei 2 fili richiesti dal protocollo (SDA e SCL) sono bidirezionali, supportati comunque dal modulo di conversione.

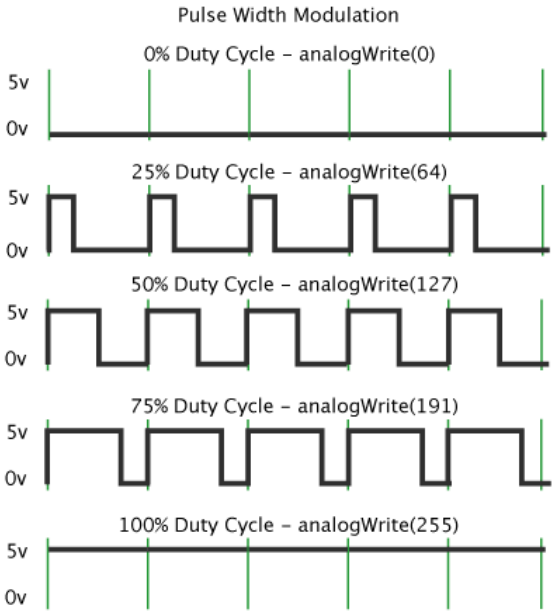


https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

PWM

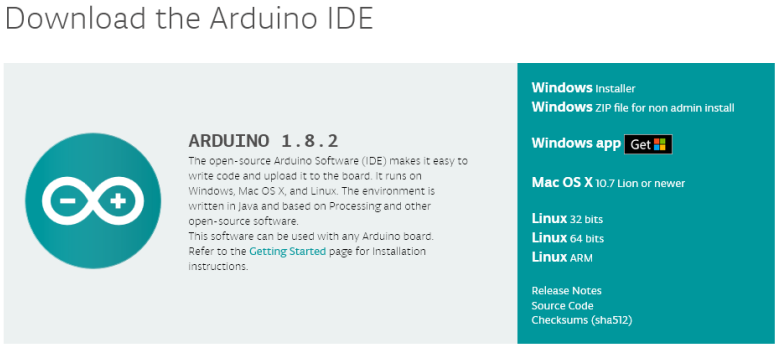
Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.

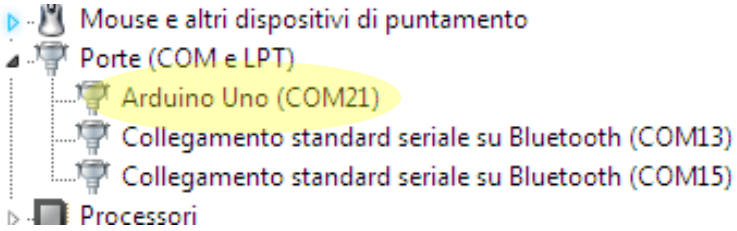


L'ambiente di sviluppo di Arduino (l'IDE)

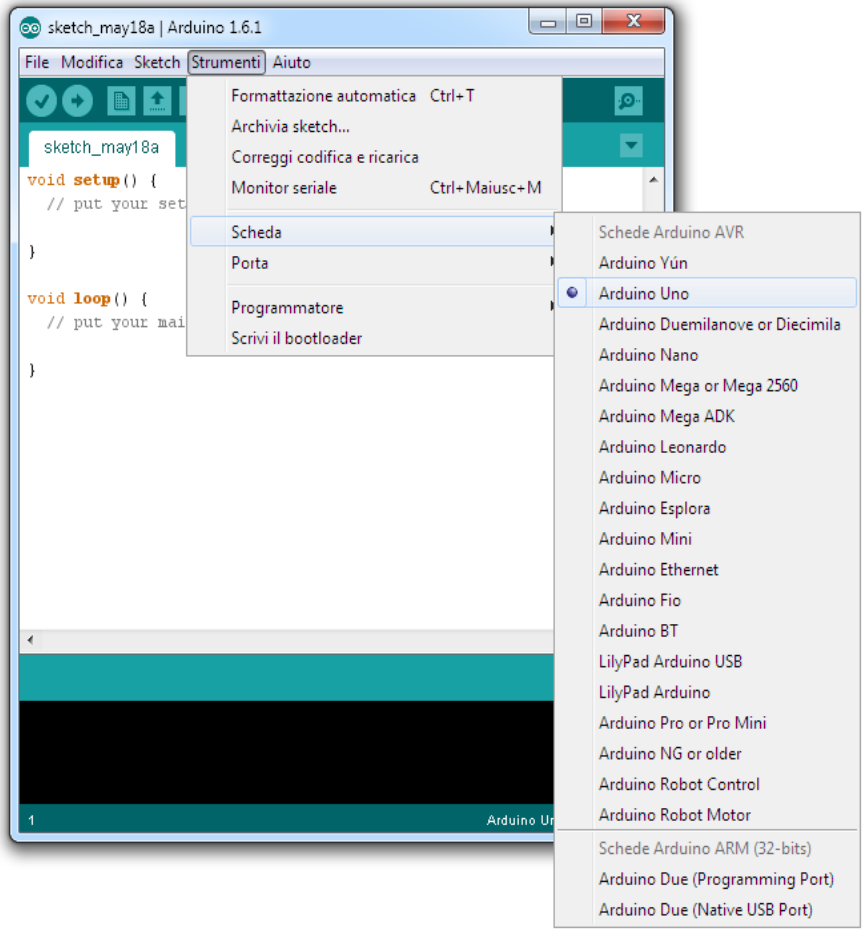
Arduino si connette al computer tramite USB. L'IDE (Integrated development environment) è un semplice editor di testo che permette di editare il software dell'utente, verificarlo, compilarlo e caricarlo sulla scheda Arduino. L'IDE è multipiattaforma, quindi disponibile per Windows, OSX e Linux.

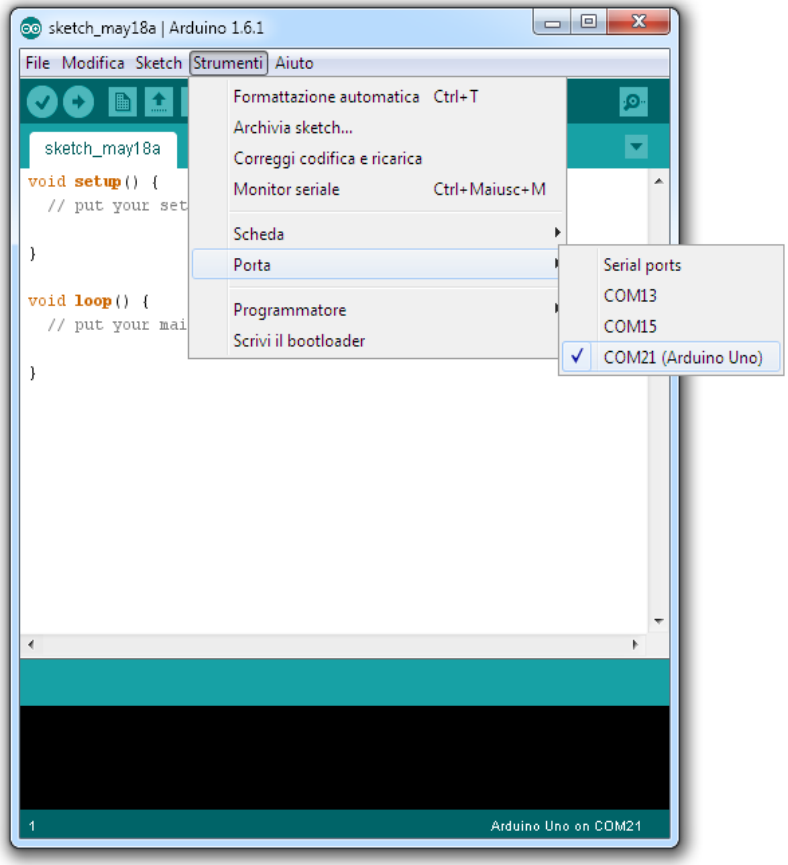


Dopo avere installato l'IDE, inclusi i relativi driver, e avere connesso la propria scheda Arduino alla porta USB del computer, questa sarà riconosciuta come una porta seriale.



Ora, nell'IDE andrà selezionato il tipo di scheda che si sta usando e la relativa porta seriale (virtuale)

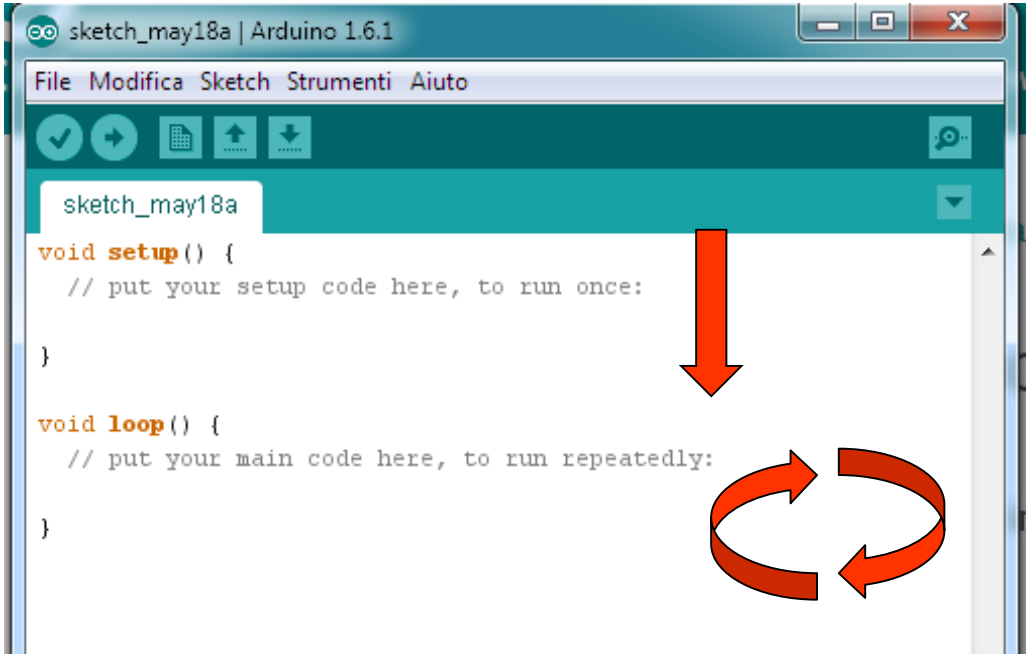




READY TO GO!!

Il linguaggio di programmazione di Arduino

Tutti i programmi utente caricabili su Arduino sono formati da almeno due parti (più eventuali altre funzioni definite dall'utente).



Le parti «obbligatorie» si chiamano:

setup() e **loop**()

- La funzione **setup**() esegue la parte di codice racchiusa tra le parentesi { } una sola volta, all'avvio o reset della scheda Arduino. Qui va inserito il codice di inizializzazione del programma o delle periferiche connesse alla scheda.
- La funzione **loop**() esegue in loop infinito la parte di codice racchiusa tra le parentesi { } Qui andrà inserito il codice principale del nostro programma.
- Eventuali altre funzioni si possono scrivere e richiamare

La sintassi del C implementato in Arduino

Sul sito ufficiale è presente il manuale del linguaggio di programmazione
https://www.arduino.cc/en/Reference/HomePage

Tutti i programmi di Arduino sono composti di questi 3 elementi:

- **Strutture**
- **Valori** (variabili e costanti)
- **Funzioni**

La struttura di base del programma è formata dalle due funzioni Setup e Loop, ma all'interno di queste possono essere presenti altre strutture di controllo come per esempio *if… else* oppure *do... while* oppure *for.*

La sintassi del C prevede che le parentesi graffe { } delimitino la porzione di codice eseguita dalle varie strutture di controllo.

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Further Syntax

- ; (semicolon)
- {} (curly braces)
- // (single line comment)
- /* */ (multi-line comment)
- #define
- #include

Bitwise Operators

- & (bitwise and)
- | (bitwise or)
- ^ (bitwise xor)
- ~ (bitwise not)
- << (bitshift left)
- >> (bitshift right)

Arithmetic Operators

- = (assignment operator)
- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo)

Boolean Operators

- && (and)
- || (or)
- ! (not)

Pointer Access Operators

- * dereference operator
- & reference operator

Comparison Operators

- == (equal to)
- != (not equal to)
- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)

Compound Operators

- ++ (increment)
- -- (decrement)
- += (compound addition)
- -= (compound subtraction)
- *= (compound multiplication)
- /= (compound division)
- %= (compound modulo)
- &= (compound bitwise and)
- |= (compound bitwise or)

Data Types

- void
- boolean
- char
- unsigned char
- byte
- int
- unsigned int
- word
- long
- unsigned long
- short
- float
- double
- string - char array
- String - object
- array

Constants

- HIGH | LOW
- INPUT | OUTPUT | INPUT_PULLUP
- LED_BUILTIN
- true | false
- integer constants
- floating point constants

Conversion

- char()
- byte()
- int()
- word()
- long()
- float()

Variable Scope & Qualifiers

- variable scope
- static
- volatile
- const

Utilities

- sizeof()
- PROGMEM

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

Analog I/O

- analogReference()
- analogRead()
- analogWrite() - *PWM*

Due & Zero only

- analogReadResolution()
- analogWriteResolution()

Random Numbers

- randomSeed()
- random()

Bits and Bytes

- lowByte()
- highByte()
- bitRead()
- bitWrite()
- bitSet()
- bitClear()
- bit()

Advanced I/O

- tone()
- noTone()
- shiftOut()
- shiftIn()
- pulseIn()

Time

- millis()
- micros()
- delay()
- delayMicroseconds()

Math

- min()
- max()
- abs()
- constrain()
- map()
- pow()
- sqrt()

Trigonometry

- sin()
- cos()
- tan()

Characters

- isAlphaNumeric()
- isAlpha()
- isAscii()
- isWhitespace()
- isControl()
- isDigit()
- isGraph()
- isLowerCase()
- isPrintable()
- isPunct()
- isSpace()
- isUpperCase()
- isHexadecimalDigit()

External Interrupts

- attachInterrupt()
- detachInterrupt()

Interrupts

- interrupts()
- noInterrupts()

Communication

- Serial
- Stream

USB (32u4 based boards and Due/Zero only)

- Keyboard
- Mouse

if / else

if/else allows greater control over the flow of code than the basic **if** statement, by allowing multiple tests to be grouped together. For example, an analog input could be tested and one action taken if the input was less than 500, and another action taken if the input was 500 or greater. The code would look like this:

```
if (pinFiveInput < 500)
{
  // action A
}
else
{
  // action B
}
```

else can proceed another **if** test, so that multiple, mutually exclusive tests can be run at the same time.

for statements

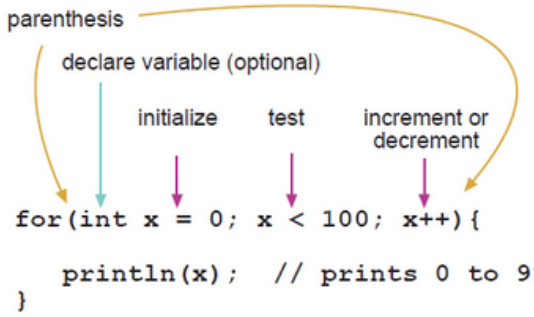
Description

The **for** statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The **for** statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the **for** loop header:

```
for (initialization; condition; increment) {

//statement(s);    }
```



The **initialization** happens first and exactly once. Each time through the loop, the **condition** is tested; if it's true, the statement block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes false, the loop ends.

Example

```
// Dim an LED using a PWM pin
int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

void setup()
{
  // no setup needed
}

void loop()
{
   for (int i=0; i <= 255; i++){
      analogWrite(PWMpin, i);
      delay(10);
   }
}
```

```
for(int x = 2; x < 100; x = x * 1.5){
println(x);
}
```

Generates: 2,3,4,6,9,13,19,28,42,63,94

Another example, fade an LED up and down with one **for** loop:

```
void loop()
{
   int x = 1;
   for (int i = 0; i > -1; i = i + x){
      analogWrite(PWMpin, i);
      if (i == 255) x = -1;              // switch direction at peak
      delay(10);
   }
}
```

switch / case statements

Like **if** statements, **switch...case** controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The **break** keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

Example

```
switch (var) {
    case 1:
      //do something when var equals 1
      break;
    case 2:
      //do something when var equals 2
      break;
    default:
      // if nothing else matches, do the default
      // default is optional
    break;
  }
```

while loops

Description

while loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the **while** loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

Syntax

```
while(expression){
  // statement(s)
}
```

Expression true: esegue statement(s)

Expression false: non esegue statement(s) ed esegue le istruzioni successive al while

Parameters

expression - a (boolean) C statement that evaluates to true or false

Example

```
var = 0;
while(var < 200){
  // do something repetitive 200 times
  var++;
}
```

do - while

The **do** loop works in the same manner as the **while** loop, with the exception that the condition is tested at the end of the loop, so the **do** loop will *always* run at least once.

```
do
{
    // statement block
} while (test condition);
```

Example

```
do
{
  delay(50);           // wait for sensors to stabilize
  x = readSensors();   // check the sensors

} while (x < 100);
```

break

break is used to exit from a **do**, **for**, or **while** loop, bypassing the normal loop condition. It is also used to exit from a **switch** statement.

Example

```
for (x = 0; x < 255; x ++)
{
    analogWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold){      // bail out on sensor detect
       x = 0;
       break;
    }
    delay(50);
}
```

continue

The continue statement skips the rest of the current iteration of a loop (**do**, **for**, or **while**). It continues by checking the conditional expression of the loop, and proceeding with any subsequent iterations.

Example

```
for (x = 0; x < 255; x ++)
{
    if (x > 40 && x < 120){       // create jump in values
        continue;
    }

    analogWrite(PWMpin, x);
    delay(50);
}
```

return

Terminate a function and return a value from a function to the calling function, if desired.

Syntax:

return;

return value; // both forms are valid

Parameters

value: any variable or constant type

The return keyword is handy to test a section of code without having to "comment out" large sections of possibly buggy code.

```
void loop(){

// brilliant code idea to test here

return;

// the rest of a dysfunctional sketch here
// this code will never be executed
}
```

Examples:

A function to compare a sensor input to a threshold

```
int checkSensor(){
    if (analogRead(0) > 400) {
        return 1;
    else{
        return 0;
    }
}
```

goto: meglio non usarlo

Variable Scope

Variables in the C programming language, which Arduino uses, have a property called *scope*. This is in contrast to early versions of languages such as BASIC where every variable is a *global* variable.

A global variable is one that can be *seen* by every function in a program. Local variables are only visible to the function in which they are declared. In the Arduino environment, any variable declared outside of a function (e.g. setup(), loop(), etc.), is a global variable.

When programs start to get larger and more complex, local variables are a useful way to insure that only one function has access to its own variables. This prevents programming errors when one function inadvertently modifies variables used by another function.

It is also sometimes handy to declare and initialize a variable inside a *for* loop. This creates a variable that can only be accessed from inside the for-loop brackets.

Example:

```
int gPWMval;  // any function will see this variable

void setup()
{
  // ...
}

void loop()
{
  int i;    // "i" is only "visible" inside of "loop"
  float f;  // "f" is only "visible" inside of "loop"
  // ...

  for (int j = 0; j <100; j++){
  // variable j can only be accessed inside the for-loop brackets
  }

}
```

pinMode()

Description

Configures the specified pin to behave either as an input or an output. See the description of digital pins for details on the functionality of the pins.

As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode INPUT_PULLUP. Additionally, the INPUT mode explicitly disables the internal pullups.

Syntax

pinMode(pin, mode)

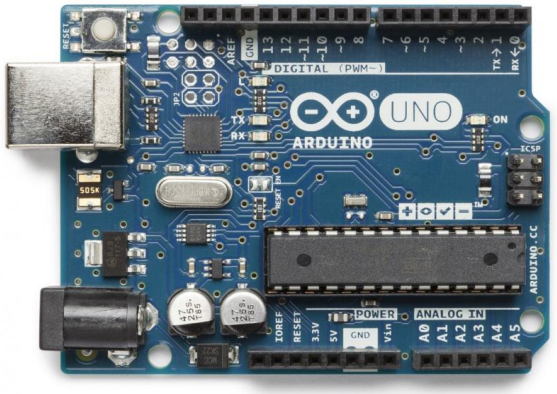
Parameters

pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT, or INPUT_PULLUP. (see the digital pins page for a more complete description of the functionality.)

Returns

None



Atmega168 Pin Mapping

Arduino function				Arduino function
reset	(PCINT14/RESET) PC6	1	28 PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27 PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26 PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25 PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24 PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23 PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22 GND	GND
GND	GND	8	21 AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20 AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19 PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18 PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17 PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16 PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15 PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

digitalWrite()

Description

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor. See the digital pins tutorial for more information.

NOTE: If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull-up resistor, which acts like a large current-limiting resistor.

Syntax

digitalWrite(pin, value)

Parameters

pin: the pin number

value: HIGH or LOW

Returns

none

digitalRead()

Description

Reads the value from a specified digital pin, either HIGH or LOW.

Syntax

digitalRead(pin)

Parameters

pin: the number of the digital pin you want to read (*int*)

Returns

HIGH or LOW

Nota: per evitare che un pin di ingresso a cui non è connesso niente (ad esempio un interruttore aperto) resti in uno stato ignoto o incerto si deve utilizzare una resistenza di pull-up o di pull-down (funzione presente anche via software, cfr. pinMode INPUT_PULLUP)

analogWrite()

Description

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to **analogWrite()**, the pin will generate a steady square wave of the specified duty cycle until the next call to **analogWrite()** (or a call to **digitalRead()** or **digitalWrite()** on the same pin). The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support analogWrite() on pins 9, 10, and 11.

The Arduino Due supports analogWrite() on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.

You do not need to call pinMode() to set the pin as an output before calling analogWrite().

The *analogWrite* function has nothing to do with the analog pins or the *analogRead* function.

Syntax

analogWrite(pin, value)

Parameters

pin: the pin to write to.

value: the duty cycle: between 0 (always off) and 255 (always on).

Returns

nothing

analogRead()

Description

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference().

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax

analogRead(pin)

Parameters

pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Returns

int (0 to 1023)

Note

If the analog input pin is not connected to anything, the value returned by analogRead() will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).

#include

#include is used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.

The main reference page for AVR C libraries (AVR is a reference to the Atmel chips on which the Arduino is based) is here.

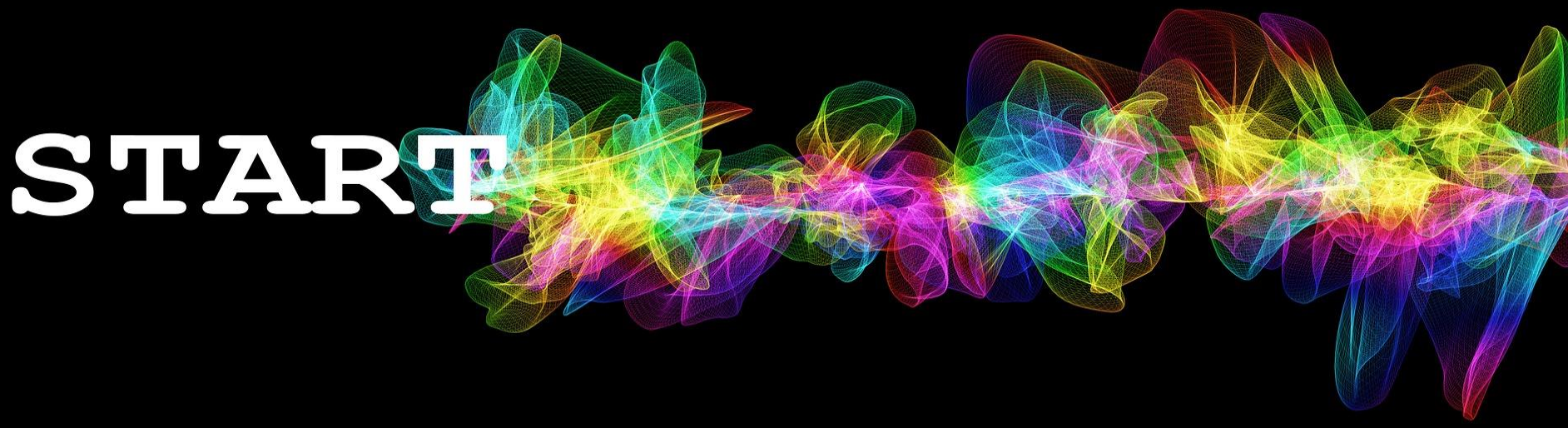
Note that **#include**, similar to **#define**, has no semicolon terminator, and the compiler will yield cryptic error messages if you add one.

Example

This example includes a library that is used to put data into the program space *flash* instead of *ram*. This saves the ram space for dynamic memory needs and makes large lookup tables more practical.

```
#include <avr/pgmspace.h>

prog_uint16_t myConstants[] PROGMEM = {0, 21140, 702   , 9128,  0, 25764, 8456,
0,0,0,0,0,0,0,0,29810,8968,29762,29762,4500};
```


START

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Ing. Paolo Guidorzi

Dipartimento di Ingegneria Industriale

paolo.guidorzi@unibo.it

http://acustica.ing.unibo.it/Staff/paolo/index.html