ISTITUTO TECNICO INDUSTRIALE STATALE
G.GALILEI DI SAN SECONDO

SISTEMI EMBEDDED
Il caso Arduino

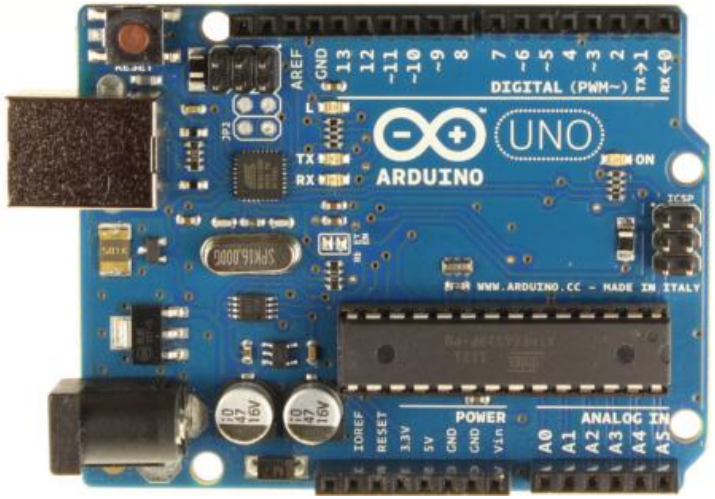
"se ascolto dimentico, se vedo ricordo, se faccio capisco"

INDICE

- Le schede Arduino e le shield
- Sleep, interrupt e consumi
- Storage
- Servo motori

I DIVERSI ARDUINO

Arduino UNO

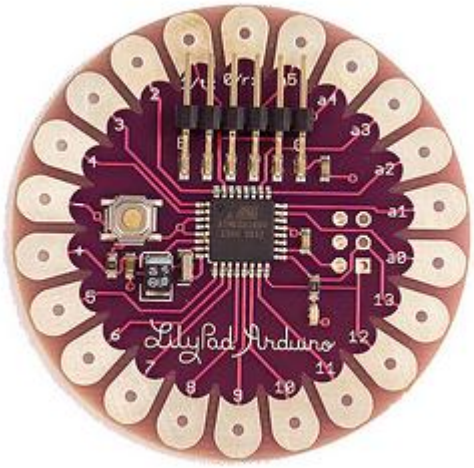


- Versione 3
- Comunicazione seriale facilitata
- Bootloader precaricato
- Reset software automatico
- Protezione contro ''Overcurrent 500mA''

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

I DIVERSI ARDUINO

LilyPad Arduino



- 50mm di diametro
- Lavabile

Microcontroller	ATmega168V ATmega328V
Operating Voltage	2.7-5.5V
Input Voltage (recommended)	2.7-5.5V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
Flash Memory	16 KB (of which 2 KB used by bootloader)
SRAM	1 KB
EEPROM	512 bytes
Clock Speed	8 MHz

I DIVERSI ARDUINO

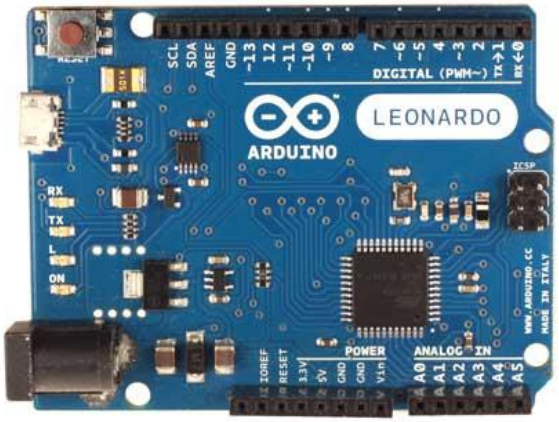
Arduino Nano

- 40mm di diametro
- FTDI solo con USB
- Reset software automatico

Microcontroller	ATmega168 ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V (limits 6-20V)
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	8
DC Current per I/O Pin	40 mA
Flash Memory	16 KB or 32 KB (of which 2 KB used by bootloader)
SRAM	1 KB or 2 KB
EEPROM	512 bytes or 1 KB
Clock Speed	16 MHz

I DIVERSI ARDUINO

Arduino Leonardo

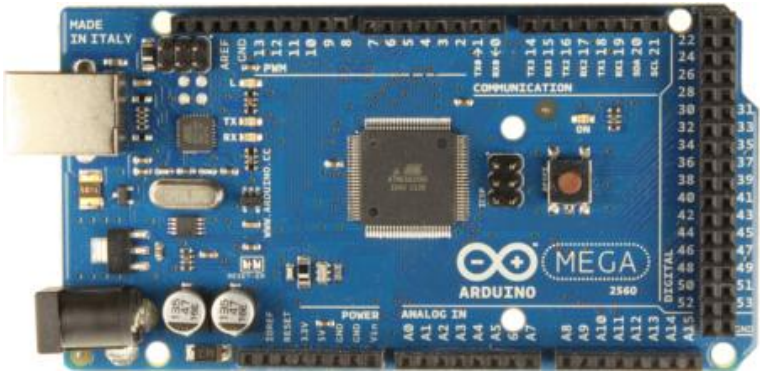


Microcontroller	ATmega32u4
Operating Voltage	5V
Input Voltage (recommended)	7-12V (limits 6-20V)
Digital I/O Pins	20 (of which 7 provide PWM output)
Analog Input Pins	12
DC Current per I/O Pin	40 mA
Flash Memory	32 KB (of which 4 KB used by bootloader)
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

- Implementa anche comunicazione USB (es. visibile come mouse o tastiera dal pc)
- 6 canali digitali/analogici (4,6,8,9,10,12)
- Reset software automatico
- Necessario inizializzare il bootloader

I DIVERSI ARDUINO

Arduino Mega 2560

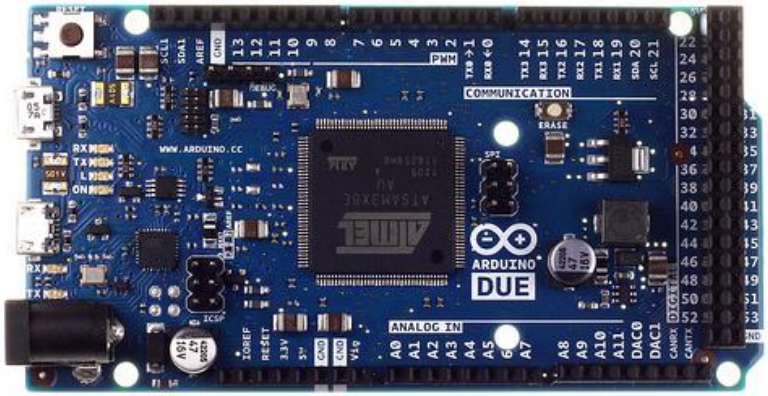


Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V (limits 6-20V)
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
Flash Memory	256 KB (of which 8 KB used by bootloader)
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

- Come l'Arduino UNO solo con più pin di I/O
- La I2C (pin 20 e 21) non è allocata come sull'Arduino UNO (pin A4 e A5)

I DIVERSI ARDUINO

Arduino Due



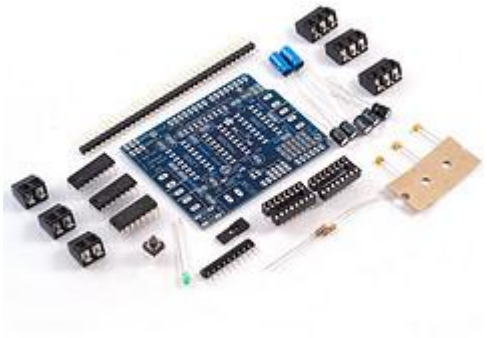
Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V (limits 6-20V)
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Pins	12 Input e 2 Output
DC Current per I/O Pin	800 mA
Flash Memory	512 KB (of which 8 KB used by bootloader)
SRAM	96 KB
EEPROM	4 KB
Clock Speed	84 MHz

- Processore a 32-bit
- Supporto a comunicazione CAN
- Integrazione di un modulo DAC a 12bit

ALCUNE SHIELD IN COMMERCIO

Wireless Shield

Ethernet Shield



Motor Shield

Xbee

WiFi

Proto Shield

ESEMPI DI SKETCH CON LE SHIELD

NETWORKING

-> Web Server e client Ethernet

-> Web Server e client Wireless

-> Rete Xbee

Ma se voglio sapere come utilizzare la Motor Shield?
Cerco su internet (link) !!!

CONSUMI

Con lo sketch ‘’Blink’’ è stato possibile effettuare un test sul consumo del led 13 presente sulla scheda Arduino. Nella figura a destra, quando il led è spento, la scheda consuma 48mA, mentre quando si accende, consuma 51,5mA.

Con una batteria da 1000 mA si potrebbe alimentare Arduino che attende dentro un delay per circa 20 ore (1000 / 48 = 20,8333)!!

INTERRUPT

Come risolvere il problema?

I pin di Arduino Uno sensibili alle variazioni di segnale sono i pin2 e pin3 che rispettivamente vengono specificati nella funzione **`attachInterrupt`** come 0 e 1.

Nella funzione `attachInterrupt` , oltre a specificare il pin che vogliamo monitorare dobbiamo specificare il nome della funzione da eseguire quando si scatena l'interruzione e il modo in cui l'interruzione deve essere interpretata.

La variabili **state** viene dichiarata utilizzando l'attributo **volatile**, questo perchè la funzione gestita dall'interrupt viene eseguita in contemporanea al blocco **loop()** ed è necessario che in fase di compilazione la variabile sia caricata nella RAM.

INTERRUPT

La variazione di stato del segnale applicato all'Arduino può essere gestita in vari modi, a seconda del parametro utilizzato per la gestione dell'interrupt. La tabella seguente mostra i parametri che determinano come deve essere generata l'interruzione.

LOW	Il segnale viene scatenato quando il livello del segnale è basso
CHANGE	Il segnale viene scatenato ogni volta che si verifica un cambiamento di livello sul pin
RISING	Il segnale viene scatenato ogni volta che il livello passa da basso ad alto
FALLING	Il segnale viene scatenato ogni volta che il livello passa da alto a basso

CODICE DI ESEMPIO PER INTERRUPT

```
int pin = 8;
//Notare l'uso di 'volatile'
volatile int state = LOW;

void setup(){
    pinMode(pin, OUTPUT);
    digitalWrite(pin, LOW);

    //Creo un gestore di interrupt e lo associo al pin3
    attachInterrupt(1, GestInt, LOW);
}

void loop(){
    //fai qualcosa...
    delay(10);
}

void GestInt(){
    state = !state;
    digitalWrite(pin, state);
}
```

INTERRUPT TIMER

Timer1

Questa libreria consente di configurare il timer interno dell'ATmega328. Ci sono 3 timer disponibili per Arduino. L'uso dei timer è utile per generare segnali PWM. L'accuratezza del timer dipende dalla velocità del processore e dalla sua frequenza. Il Timer1 dispone inoltre di un prescaler che può essere settato secondo la tabella sottostante.

Max Period = (Prescale)(1/Frequency)*(2^17)*
Time per Tick = (Prescale)(1/Frequency)*

For 16MHz:

Prescale	Time per counter tick	Max Period
1	0.0625 uS	8.192 mS
8	0.5 uS	65.536 mS
64	4 uS	524.288 mS
256	16 uS	2097.152 mS
1024	64uS	8388.608mS

INTERRUPT TIMER

```
#include "TimerOne.h"

void setup(){
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);
  // initialize timer1, and set a 1/2
     second period
  Timer1.pwm(9, 512);
  // setup pwm on pin 9, 50% duty cycle
  Timer1.attachInterrupt(callback);
  // attaches callback() as a timer
     overflow interrupt
}

void callback(){
  digitalWrite(10, digitalRead(10) ^ 1);
}

void loop(){
  // your program here...
}
```

Approfondimenti
Link1
Link 2

SLEEP

Arduino è dotato di due porte di interrupt: pin 2 e 3.
Grazie a questi due interrupt, Arduino è in grado di "svegliarsi" e riprendere l'esecuzione del codice. È anche possibile eseguire codice speciale a seconda di quale interrupt ha innescato la sveglia. Eventi sul USART (la porta seriale) riattivano Arduino. Perché questo funzioni, Arduino deve essere in `POWER_MODE_IDLE`, l'unico modo che non disattiva l'USART.
Anche se questa modalità non dà grandi risparmi energetici è possibile utilizzare le funzioni fornite in avr/power.h

```
power_adc_disable(),
power_spi_disable(),
power_timer0_disable(),
power_timer1_disable(),
power_timer2_disable(),
power_twi_disable()
```

per disattivare altri moduli hardware e quindi ottenere maggiore risparmio di energia (link).

SLEEP

Quando Arduino è in `SLEEP_MODE_PWR_DOWN` l'unico modo per riattivarlo è con un interrupt watchdog timer o con un cambio di livello sui piedini 2 o 3.
Un cambio di livello implica che il pin deve essere tenuto in quello stato per un certo periodo di tempo prima che l'interrupt venga attivato.
Nella routine di servizio di interrupt (ISR) l'interrupt deve essere disabilitato altrimenti l'ISR verrà richiamato più volte finché non ci sarà un cambio di stato.

```
void pin2_isr(){
  detachInterrupt(0);
  pin2_interrupt_flag = 1;
}
```

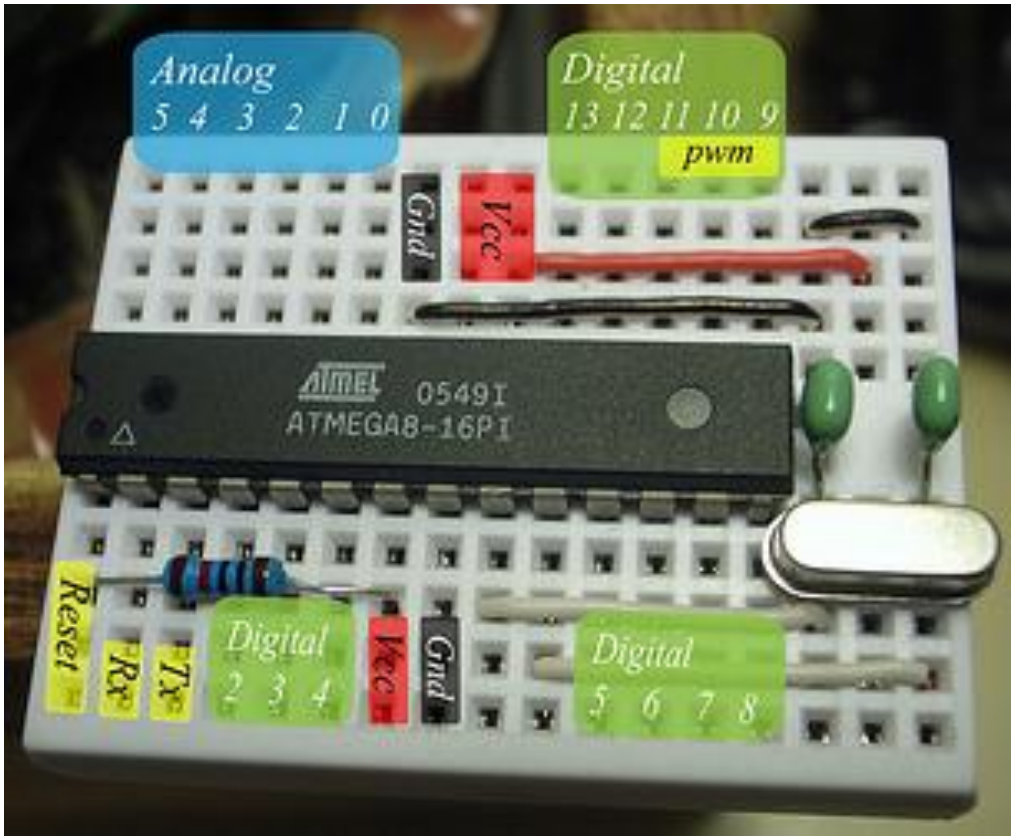
Come conseguenza, l'interrupt deve essere riattivato nel codice una volta che il pin è tornato a uno stato normale, cioè per un livello basso, verificare che il pin sia andato alto prima di fissare l'interrupt nuovamente.

-> `SleepExample`

ARDUINO A CONSUMO ZERO

Ovviamente per avere un Arduino a consumo zero devo ottimizzare i dispositivi collegati e magari pensare ad una versione Stand-Alone in modo da eliminare dispositivi che non mi servono (per esempio l'interfaccia seriale o i controllori per la comunicazione I2C o SPI).

Con una soluzione del genere si potrebbero realizzare dispositivi outdoor con delle autonomie piuttosto elevate (anche un paio di anni se si ottimizzano le sleep)



STORAGE

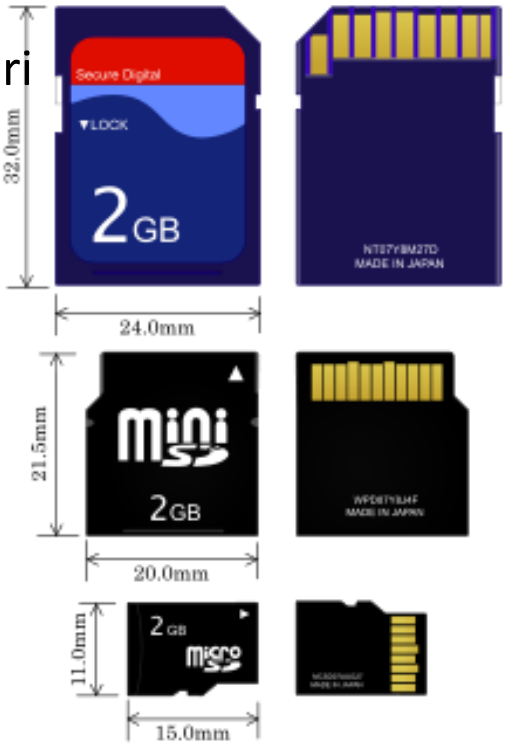
Arduino dispone di alcuni sistemi di storage. Uno interno al microcontrollore, basato su una **memoria EEPROM** (Arduino UNO ha un EEPROM di 1KB). In ciascuna cella puoi memorizzare un valore che va da 0 a 255, è un dato di tipo *integer* ossia un numero intero.

-> EEPROM

1. *per poter scrivere in ogni cella occorrono 3,3 millisecondi, per cui tienine conto quando memorizzerai i tuoi dati nella EEPROM;*
2. *le EEPROM possono essere scritte per un massimo di 100.000 volte, per scrittura si intende ogni volta che cambi il valore in una cella, considerando che solitamente utilizzi la EEPROM solo per memorizzare dei dati di configurazione, questo valore è decisamente elevato;*

STORAGE

Mentre l'altra soluzione potrebbe essere quella di utilizzare una shield con supporto per **schede SD** e salvare i dati su di essa. Le shield con slot per micro SD consentono di montare fino a schede di 2Gb utilizzabile per scrivere e leggere i propri file.



Libreria <SD.h>

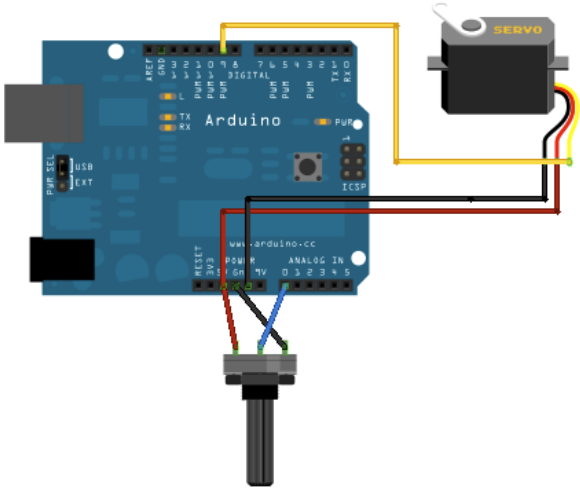
Bisogna abilitare la scrittura sull'SD. Sull'Ethernet Shield il pin di abilitazione è il 4.

```
if (!SD.begin(4)) { return; }
```

-> SD

SERVO MOTORI

Uno dei campi d'applicazione preferito d'Arduino è la robotica. Essa si basa sulla combinazione di sensori e motori per ottenere un dispositivo automatico che interagisca con il mondo esterno.



Alcuni motori utilizzati sono i **servo motori.** Essi sono particolari tipi di motore (elettrici, pneumatici, idraulici) generalmente di piccola potenza, le cui condizioni operative, a differenza dei motori tradizionali, sono soggette ad ampie e spesso repentine variazioni sia nel campo della velocità che della coppia motrice, alle quali si deve adattare con la massima rapidità e precisione.

Colore	Funzione
Nero o marrone	Negativo di alimentazione (GND)
Rosso	Positivo di alimentazione (+Vcc)
Giallo o bianco	Segnale di comando (ingresso)

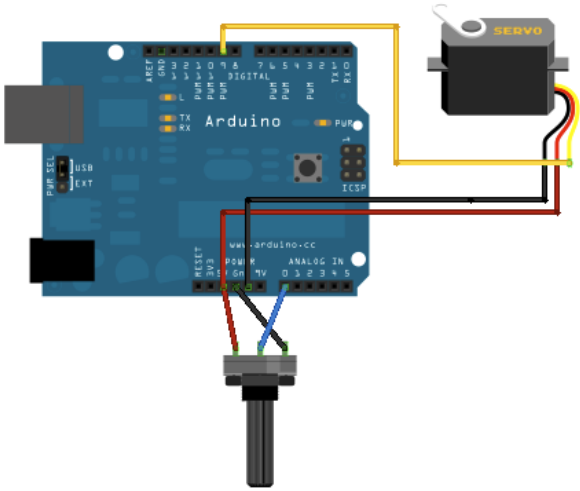
SERVO MOTORI – Istruzioni principali

```
Libreria <Servo.h>
Servo myServo
```

`myServo.Attach();`

Associa la variabile servo ad uno specifico pin.

```
myServo.attach(pin);
myServo.attach(pin, min,max);
```



Min (opzionale): durata minima dell'impulso, in microsecondi, corrispondente al minimo grado di rotazione (0 gradi) del servo (il valore predefinito è 544).
Max (opzionale): durata massima dell'impulso, in microsecondi, corrispondente alla massima rotazione (180 gradi) del servo (il valore predefinito è 2400).

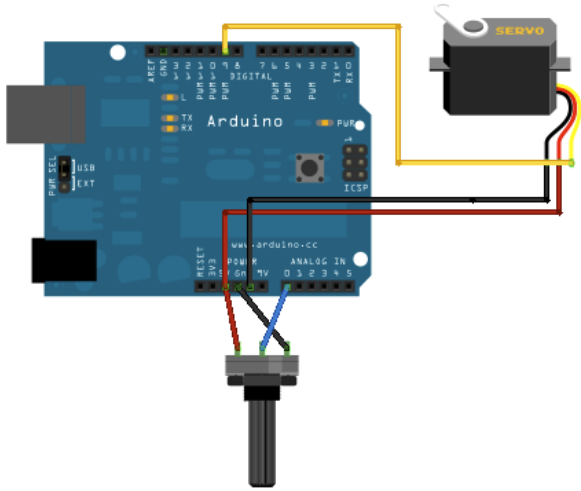
SERVO MOTORI – Istruzioni principali

```
Libreria <Servo.h>
Servo myServo
```

`myServo.Attached();`

Verifica l'associazione tra la variabile servo ed il pin.

Ritorna vero se il servo è associato al pin; falso in caso contrario.

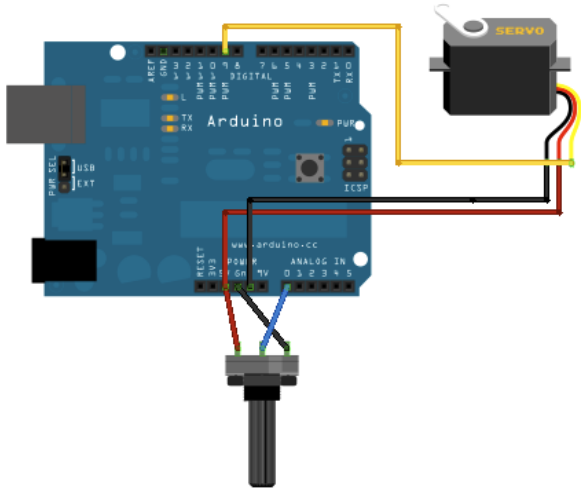


SERVO MOTORI – Istruzioni principali

```
Libreria <Servo.h>
Servo myServo
```

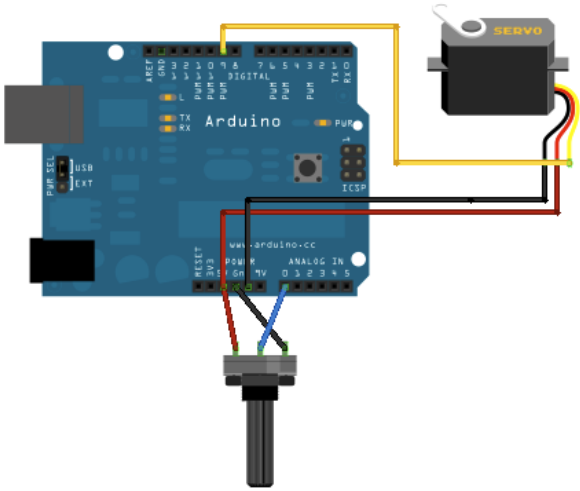
`myServo.Detach();`

Dissocia la variabile servo al pin specificato. Se tutte le variabili servo non sono associate, i pin PWM possono essere usati come normali uscite "Analogiche" con l'istruzione analogWrite().



SERVO MOTORI – Istruzioni principali

```
Libreria <Servo.h>
Servo myServo
```



`myServo.Read();`

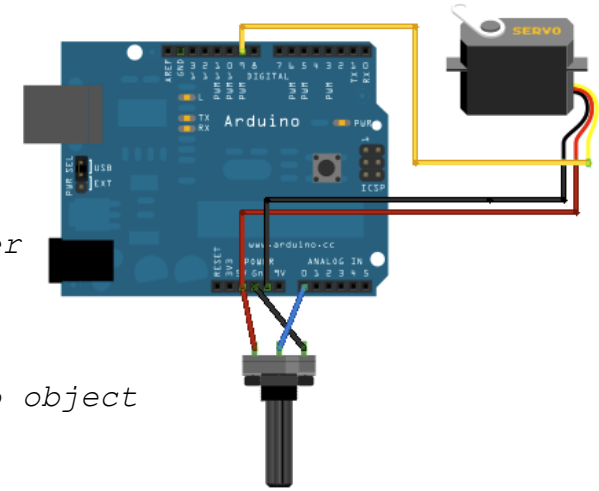
Legge l'attuale posizione del servo corrispondente all'ultima posizione passata con l'istruzione write(). Ritorna l'angolo del servo da 0 a 180 gradi.

`myServo.Write(angle);`

Invia il valore in gradi relativo alla posizione del perno del servo. Un valore 0 corrisponde alla massima rotazione a sinistra, mentre 180 equivale alla massima rotazione a destra; L'esatta corrispondenza tra valore in gradi inviato e l'effettiva rotazione del servo viene specificata dai valori Max e Min nella dichiarazione dell'istruzione attach(); tali valori devono essere ricavati mediante prove pratiche, in quanto possono anche variare da servo a servo.

SERVO MOTORI - Esempio

```
// Controlling a servo position using a potentiometer
#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int potpin = 0;  // analog pin used to connect the potentiometer
int val;    // variable to read the value from the analog pin

void setup() {
  myservo.attach(9); //attaches the servo on pin 9 to the servo object
}

void loop() {
  //reads the value of the potentiometer (value between 0 and 1023)
  val = analogRead(potpin);

  // scale it to use it with the servo (value between 0 and 180)
  val = map(val, 0, 1023, 0, 179);

  // sets the servo position according to the scaled value
  myservo.write(val);
  delay(15);
}
```



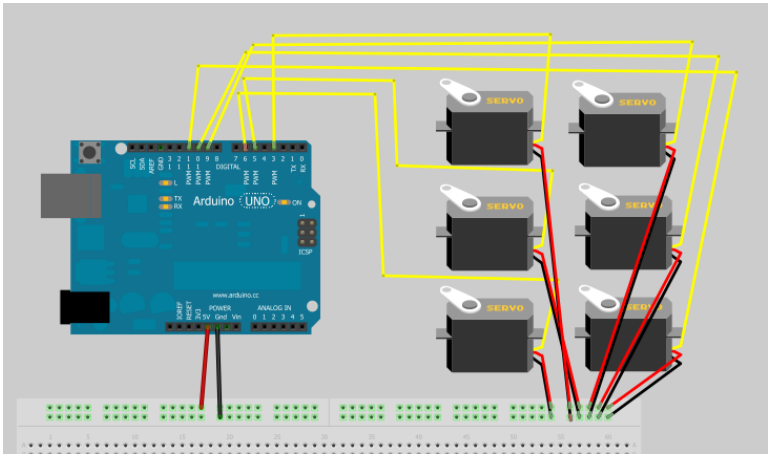
-> Servo

SERVO MOTORI – Esempio avanzato

Nel costruire un robot, ad esempio, è necessario che tu sappia pilotare più di un servo contemporaneamente, immagina il sistema di locomozione utilizzato dalla nasa per i rover Sojourner, Pathfindere Opportunity, per la missione su Marte, con 6 ruote da controllare, oppure ad un robot antropomorfo industriale con 8 giunti indipendenti.

Con Arduino ovviamente non si ha la pretesa di realizzare un prodotto tanto complesso (la rotazione delle singole ruote sarebbe affidata ad esempio a dei motori in corrente continua, ma la capacità di sterzare si potrebbe gestirla con dei servomotori).

Lo sketch realizzato è uno sketch di esempio, nulla che montato su un rover possa consentirgli di muoversi; il suo scopo è quello di pilotare i 6 servo impartendo dei comandi provenienti dalla porta seriale.



-> ProtoRover

RIFERIMENTI

www.mancio90.it

mirkomancin90@gmail.com