Corso Arduino 2015

25 Maggio 2015

Nicola Corna
nicola@corna.info

Lezione 2 - segnali analogici, audio e video

Segnali analogici

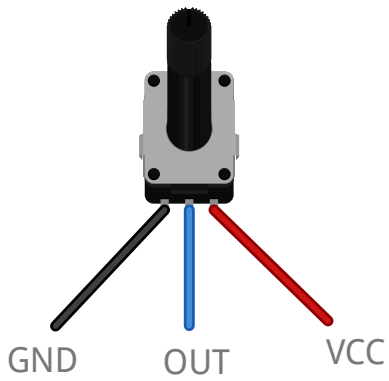
L'altra volta abbiamo lavorato con segnali digitali. Un ingresso digitale può assumere solo 2 valori, HIGH o LOW. Se l'ingresso non è esattamente 0 V o 5 V allora viene approssimato ad uno di questi due valori.

Gli ingressi analogici permettono di leggere segnali che variano in modo continuo tra due estremi (che solitamente su Arduino sono 0 e 5 Volt). La lettura non è veramente "continua" ma l'intervallo viene diviso in tanti[1] piccoli "gradini".

[1] in questo caso 1024

Trimmer

Come possiamo generare un segnale analogico che varia? Ad esempio, con un trimmer (o potenziometro)



Collegato in questo modo, facendo ruotare la manopola da un estremo all'altro, OUT varierà continuamente tra GND e VCC.

Leggere i segnali analogici

Come leggiamo ora il segnale che esce dal potenziometro?
`analogRead(int pin);`

- Legge il `pin analogico con il numero pin`
- Restituisce un numero tra 0 e 1023; va salvato in una variabile:
  `int val = analogRead(pinPot);`
- Il pin va indicato con il numero corrispondente ai pin analogici, senza la A
- I pin analogici vanno da A0 ad A5 su Arduino Leonardo

Leggere i segnali analogici

Come leggiamo ora il segnale che esce dal potenziometro?
`analogRead(int pin);`

- Legge il `pin analogico con il numero pin`
- Restituisce un numero tra 0 e 1023; va salvato in una variabile:
  `int val = analogRead(pinPot);`
- Il pin va indicato con il numero corrispondente ai pin analogici, senza la A
- I pin analogici vanno da A0 ad A5 su Arduino Leonardo

Leggere i segnali analogici

Come leggiamo ora il segnale che esce dal potenziometro?
`analogRead(int pin);`

- Legge il `pin analogico con il numero pin`
- Restituisce un numero tra 0 e 1023; va salvato in una variabile:
  `int val = analogRead(pinPot);`
- Il pin va indicato con il numero corrispondente ai pin analogici, senza la A
- I pin analogici vanno da A0 ad A5 su Arduino Leonardo

Leggere i segnali analogici

Come leggiamo ora il segnale che esce dal potenziometro?
`analogRead(int pin);`

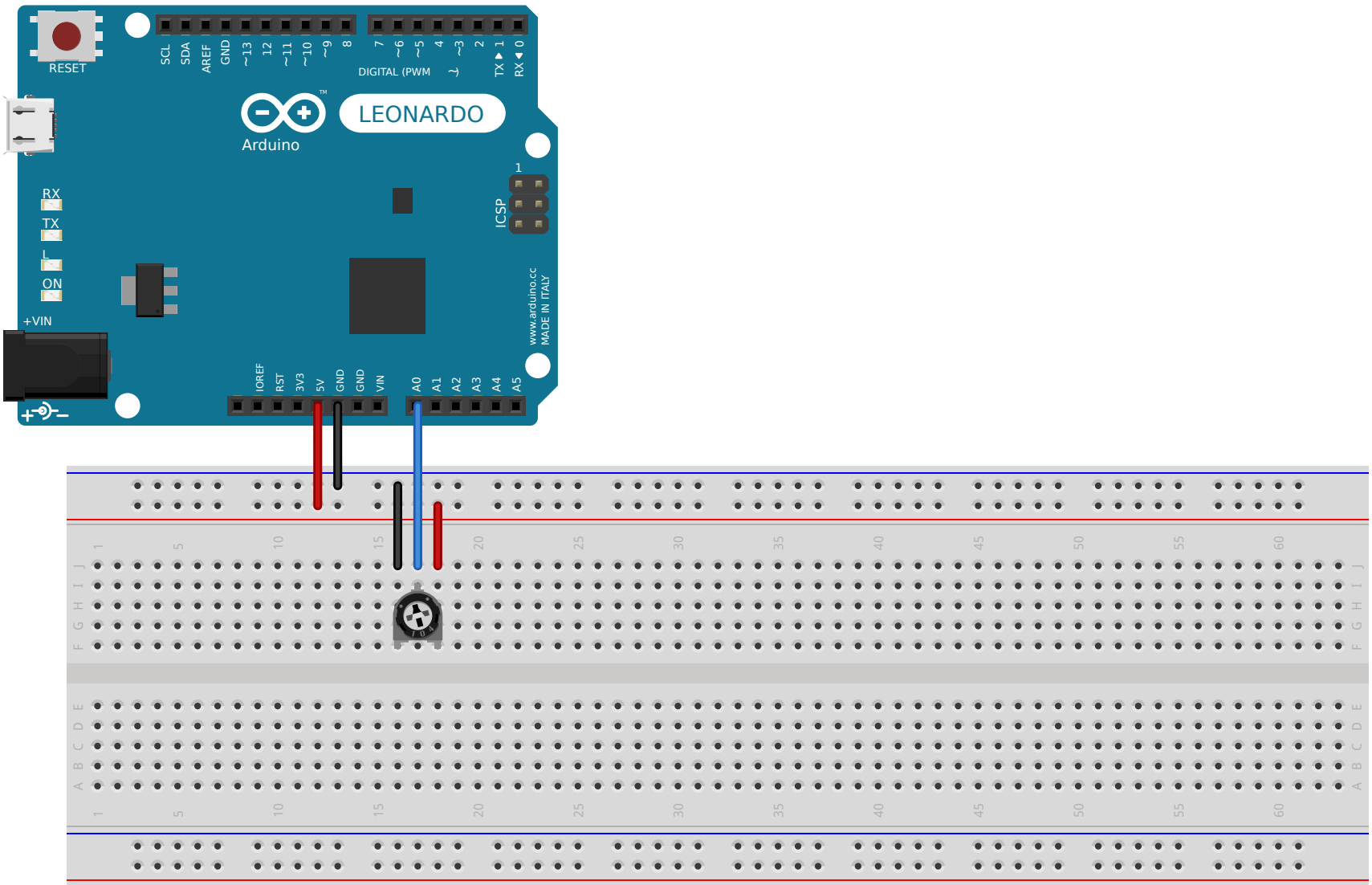
- Legge il `pin analogico con il numero pin`
- Restituisce un numero tra 0 e 1023; va salvato in una variabile:
  `int val = analogRead(pinPot);`
- Il pin va indicato con il numero corrispondente ai pin analogici, senza la A
- I pin analogici vanno da A0 ad A5 su Arduino Leonardo

Tocca a voi

Realizzate un circuito che faccia blinkare il LED (pin 13) ad intervalli regolabili, compresi tra 100 e 1000 ms

Go!

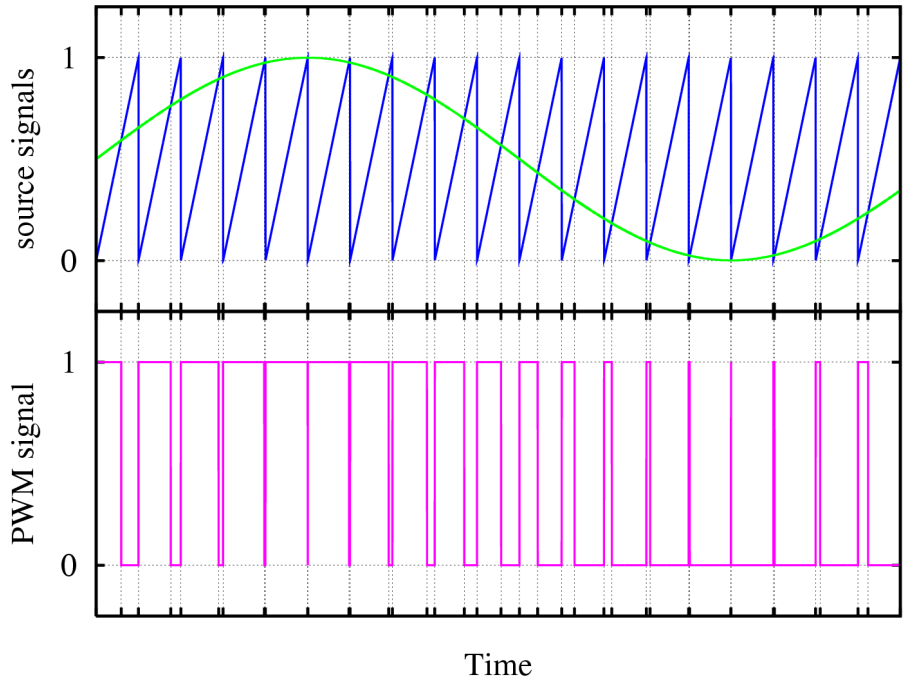
Circuito potenziometro



Soluzione

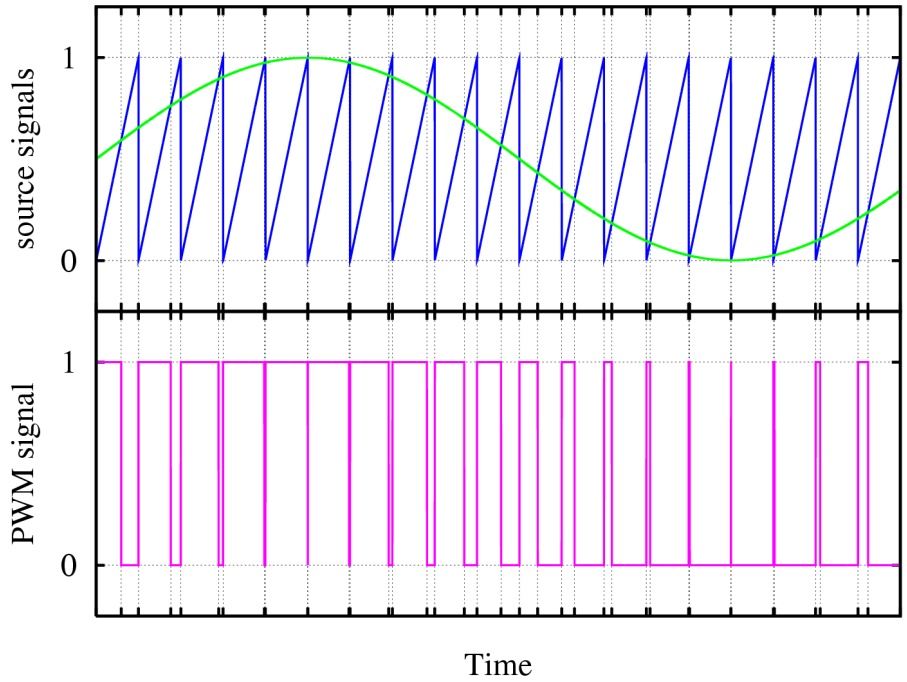
```
int ledPin = 13;
int potPin = 4;
int interval = 1000;
void setup(){
  pinMode(ledPin, OUTPUT);
}
void loop(){
  interval = 100 + analogRead(potPin) * 900.0/1023.0;
  digitalWrite(ledPin, HIGH);
  delay(interval);
  digitalWrite(ledPin, LOW);
  delay(interval);
}
```

PWM



- La PWM (Pulse Width Modulation) serve a fare emettere un'approssimazione di un voltaggio analogico, usando un voltaggio digitale
- La funzione che fa questo su Arduino è analogWrite

PWM



- La PWM (Pulse Width Modulation) serve a fare emettere un'approssimazione di un voltaggio analogico, usando un voltaggio digitale
- La funzione che fa questo su Arduino è analogWrite

analogWrite

- Sintassi: `analogWrite(pin, value)`
- `pin` è il numero del pin che si vuole usare, si posssono usare solo i pin con il simbolo ~ sulla board
- `value` è un numero tra 0 e 255
- Quando value è 0, l'uscita sarà sempre LOW, quando value è 255, l'uscita sarà sempre HIGH
- La frequenza dell'onda quadra è 490Hz sulla maggior parte degli Arduino, mentre alcuni pin sono invece a 980Hz[2].

[2] Su Leonardo sono i pin 3 e 11, su UNO sono il 5 e 6

analogWrite

- Sintassi: `analogWrite(pin, value)`
- `pin` è il numero del pin che si vuole usare, si posssono usare solo i pin con il simbolo ~ sulla board
- `value` è un numero tra 0 e 255
- Quando value è 0, l'uscita sarà sempre LOW, quando value è 255, l'uscita sarà sempre HIGH
- La frequenza dell'onda quadra è 490Hz sulla maggior parte degli Arduino, mentre alcuni pin sono invece a 980Hz[2].

[2] Su Leonardo sono i pin 3 e 11, su UNO sono il 5 e 6

analogWrite

- Sintassi: `analogWrite(pin, value)`
- `pin` è il numero del pin che si vuole usare, si posssono usare solo i pin con il simbolo ~ sulla board
- `value` è un numero tra 0 e 255
- Quando value è 0, l'uscita sarà sempre LOW, quando value è 255, l'uscita sarà sempre HIGH
- La frequenza dell'onda quadra è 490Hz sulla maggior parte degli Arduino, mentre alcuni pin sono invece a 980Hz[2].

[2] Su Leonardo sono i pin 3 e 11, su UNO sono il 5 e 6

analogWrite

- Sintassi: `analogWrite(pin, value)`
- `pin` è il numero del pin che si vuole usare, si posssono usare solo i pin con il simbolo ~ sulla board
- `value` è un numero tra 0 e 255
- Quando value è 0, l'uscita sarà sempre LOW, quando value è 255, l'uscita sarà sempre HIGH
- La frequenza dell'onda quadra è 490Hz sulla maggior parte degli Arduino, mentre alcuni pin sono invece a 980Hz[2].

[2] Su Leonardo sono i pin 3 e 11, su UNO sono il 5 e 6

analogWrite

- Sintassi: `analogWrite(pin, value)`
- `pin` è il numero del pin che si vuole usare, si posssono usare solo i pin con il simbolo ~ sulla board
- `value` è un numero tra 0 e 255
- Quando value è 0, l'uscita sarà sempre LOW, quando value è 255, l'uscita sarà sempre HIGH
- La frequenza dell'onda quadra è 490Hz sulla maggior parte degli Arduino, mentre alcuni pin sono invece a 980Hz[2].

[2] Su Leonardo sono i pin 3 e 11, su UNO sono il 5 e 6

Ciclo for

- Sintassi: `for (int i = 0; i < 255; i++) {`
  `//codice che usa la variabile i`
  `}`
- `int i=0` definisce una variabile chiamata i e gli assegna il valore 0
- `i < 255` è la condizione del ciclo, finché è vera il ciclo for continua
- `i++` è l'istruzione che viene eseguita al termine di ogni iterazione

Ciclo for

- Sintassi:
```
for (int i = 0; i < 255; i++) {
  //codice che usa la variabile i
}
```
- `int i=0` definisce una variabile chiamata i e gli assegna il valore 0
- `i < 255` è la condizione del ciclo, finché è vera il ciclo for continua
- `i++` è l'istruzione che viene eseguita al termine di ogni iterazione

Ciclo for

- Sintassi:
```
for (int i = 0; i < 255; i++) {
  //codice che usa la variabile i
}
```
- `int i=0` definisce una variabile chiamata i e gli assegna il valore 0
- `i < 255` è la condizione del ciclo, finché è vera il ciclo for continua
- `i++` è l'istruzione che viene eseguita al termine di ogni iterazione

Ciclo for

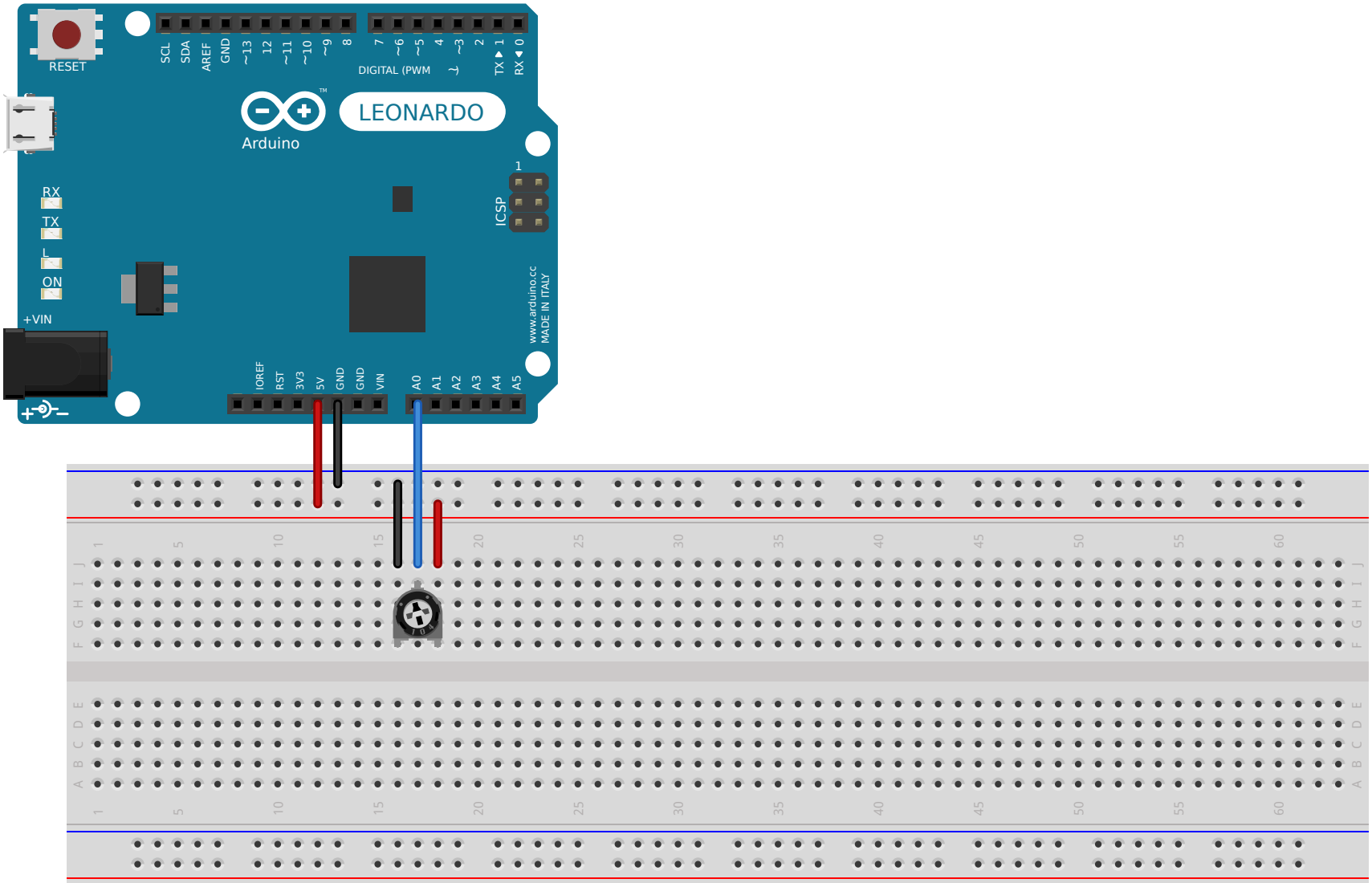
- Sintassi:
```
for (int i = 0; i < 255; i++) {
  //codice che usa la variabile i
}
```
- `int i=0` definisce una variabile chiamata i e gli assegna il valore 0
- `i < 255` è la condizione del ciclo, finché è vera il ciclo for continua
- `i++` è l'istruzione che viene eseguita al termine di ogni iterazione

Fading LED

Realizzate un programma che fa accendere e spegnere gradualmente il LED interno (pin 13).
Bonus: controllate la velocità di fading con il potenziometro

Go!

Circuito Fading LED



Soluzione

```
int ledPin = 13;
void setup()  {
  pinMode(ledPin, OUTPUT);
}
void loop()  {
  for(int fadeValue = 0 ;fadeValue <= 255;
      fadeValue +=5) {
    analogWrite(ledPin, fadeValue);
    delay(30);
  }
  for(int fadeValue = 255 ; fadeValue >= 0;
      fadeValue -=5) {
    analogWrite(ledPin, fadeValue);
    delay(30);
  }
}
```

Timing

- Oltre alla funzione `delay`, ci sono altre funzioni per la gestione del tempo utili su Arduino
- `delayMicroseconds(nMicros)`: come la delay ma il delay è in microsecondi
  - Ha una risoluzione di circa 3 microsecondi
- `millis()`: ritorna il valore di millisecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 50 giorni
- `micros()`: ritorna il valore di microsecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 70 minuti

Timing

- Oltre alla funzione `delay`, ci sono altre funzioni per la gestione del tempo utili su Arduino
- `delayMicroseconds(nMicros)`: come la delay ma il delay è in microsecondi
  - Ha una risoluzione di circa 3 microsecondi
- `millis()`: ritorna il valore di millisecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 50 giorni
- `micros()`: ritorna il valore di microsecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 70 minuti

Timing

- Oltre alla funzione `delay`, ci sono altre funzioni per la gestione del tempo utili su Arduino
- `delayMicroseconds(nMicros)`: come la delay ma il delay è in microsecondi
  - Ha una risoluzione di circa 3 microsecondi
- `millis()`: ritorna il valore di millisecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 50 giorni
- `micros()`: ritorna il valore di microsecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 70 minuti

Timing

- Oltre alla funzione `delay`, ci sono altre funzioni per la gestione del tempo utili su Arduino
- `delayMicroseconds(nMicros)`: come la delay ma il delay è in microsecondi
  - Ha una risoluzione di circa 3 microsecondi
- `millis()`: ritorna il valore di millisecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 50 giorni
- `micros()`: ritorna il valore di microsecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 70 minuti

Timing

- Oltre alla funzione `delay`, ci sono altre funzioni per la gestione del tempo utili su Arduino
- `delayMicroseconds(nMicros)`: come la delay ma il delay è in microsecondi
  - Ha una risoluzione di circa 3 microsecondi
- `millis()`: ritorna il valore di millisecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 50 giorni
- `micros()`: ritorna il valore di microsecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 70 minuti

Timing

- Oltre alla funzione `delay`, ci sono altre funzioni per la gestione del tempo utili su Arduino
- `delayMicroseconds(nMicros)`: come la delay ma il delay è in microsecondi
  - Ha una risoluzione di circa 3 microsecondi
- `millis()`: ritorna il valore di millisecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 50 giorni
- `micros()`: ritorna il valore di microsecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 70 minuti

Timing

- Oltre alla funzione `delay`, ci sono altre funzioni per la gestione del tempo utili su Arduino
- `delayMicroseconds(nMicros)`: come la delay ma il delay è in microsecondi
  - Ha una risoluzione di circa 3 microsecondi
- `millis()`: ritorna il valore di millisecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 50 giorni
- `micros()`: ritorna il valore di microsecondi passati da quando la scheda è stata accesa
  - Il numero va in overflow dopo circa 70 minuti

Timing (2)

In particolare la funzione millis() può essere utile a fare programmi come il blinking LED che però non "sprecano tempo" con la `delay()`.
Poiché i valori restituiti da millis e micros possono essere molto grandi (pensate a quanti millisecondi ci sono in 50 giorni) se vogliamo salvare il numero di millisecondi/microsecondi dovremo farlo in variabili di dimensioni maggiori rispetto alla solita `int`

```
unsigned long millisecondi = millis();
```

Polling vs Interrupt

Tutti i programmi che abbiamo scritto finora rilevavano gli input tramite polling. In pratica, il programma ad ogni passaggio chiedeva al pin "In che stato sei?" e agiva di conseguenza. Alcuni input digitali permettono di agire in modo diverso, tramite interrupt. In pratica, quando c'è un determinato cambiamento di stato su un pin, viene eseguita una funzione. Gli interrupt sono una feature abbastanza avanzata e non sempre sono il modo migliore per gestire gli input digitali. In ogni caso...

attachInterrupt

- attachInterrupt(interrupt, ISR, mode)
- interrupt: il numero dell'interrupt che si vuole abilitare. Il numero di interrupt non corrisponde al numero di pin, usate questa tabella

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

- ISR: il nome della funzione da eseguire. La funzione non deve ricevere nessun parametro e non deve ritornare niente.

attachInterrupt

- attachInterrupt(interrupt, ISR, mode)
- interrupt: il numero dell'interrupt che si vuole abilitare. Il numero di interrupt non corrisponde al numero di pin, usate questa tabella

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

- ISR: il nome della funzione da eseguire. La funzione non deve ricevere nessun parametro e non deve ritornare niente.

attachInterrupt

- attachInterrupt(interrupt, ISR, mode)
- interrupt: il numero dell'interrupt che si vuole abilitare. Il numero di interrupt non corrisponde al numero di pin, usate questa tabella

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

- ISR: il nome della funzione da eseguire. La funzione non deve ricevere nessun parametro e non deve ritornare niente.

attachInterrupt (2)

- mode: il cambiamento di stato su cui deve essere attivato l'interrupt
  - CHANGE: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW o viceversa
  - RISING: l'interrupt viene attivato ogni volta che lo stato passa da LOW a HIGH
  - FALLING: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW
  - LOW: l'interrupt viene attivato ogni volta che lo stato è LOW

attachInterrupt (2)

- mode: il cambiamento di stato su cui deve essere attivato l'interrupt
  - CHANGE: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW o viceversa
  - RISING: l'interrupt viene attivato ogni volta che lo stato passa da LOW a HIGH
  - FALLING: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW
  - LOW: l'interrupt viene attivato ogni volta che lo stato è LOW

attachInterrupt (2)

- mode: il cambiamento di stato su cui deve essere attivato l'interrupt
  - CHANGE: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW o viceversa
  - RISING: l'interrupt viene attivato ogni volta che lo stato passa da LOW a HIGH
  - FALLING: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW
  - LOW: l'interrupt viene attivato ogni volta che lo stato è LOW

attachInterrupt (2)

- mode: il cambiamento di stato su cui deve essere attivato l'interrupt
  - CHANGE: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW o viceversa
  - RISING: l'interrupt viene attivato ogni volta che lo stato passa da LOW a HIGH
  - FALLING: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW
  - LOW: l'interrupt viene attivato ogni volta che lo stato è LOW

attachInterrupt (2)

- mode: il cambiamento di stato su cui deve essere attivato l'interrupt
  - CHANGE: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW o viceversa
  - RISING: l'interrupt viene attivato ogni volta che lo stato passa da LOW a HIGH
  - FALLING: l'interrupt viene attivato ogni volta che lo stato passa da HIGH a LOW
  - LOW: l'interrupt viene attivato ogni volta che lo stato è LOW

Cose a cui stare attenti

- L'ISR ha priorità sulla funzione loop
  - Quindi se mettete come mode LOW, verrà continuamente eseguita l'ISR e non il loop
- Durante l'esecuzione dell'ISR, il contatore millis() non si incrementa
- Le variabili che devono essere modificate all'interno dell'ISR devono avere il modificatore `volatile`

Cose a cui stare attenti

- L'ISR ha priorità sulla funzione loop
  - Quindi se mettete come mode LOW, verrà continuamente eseguita l'ISR e non il loop
- Durante l'esecuzione dell'ISR, il contatore millis() non si incrementa
- Le variabili che devono essere modificate all'interno dell'ISR devono avere il modificatore `volatile`

Cose a cui stare attenti

- L'ISR ha priorità sulla funzione loop
  - Quindi se mettete come mode LOW, verrà continuamente eseguita l'ISR e non il loop
- Durante l'esecuzione dell'ISR, il contatore millis() non si incrementa
- Le variabili che devono essere modificate all'interno dell'ISR devono avere il modificatore `volatile`

Cose a cui stare attenti

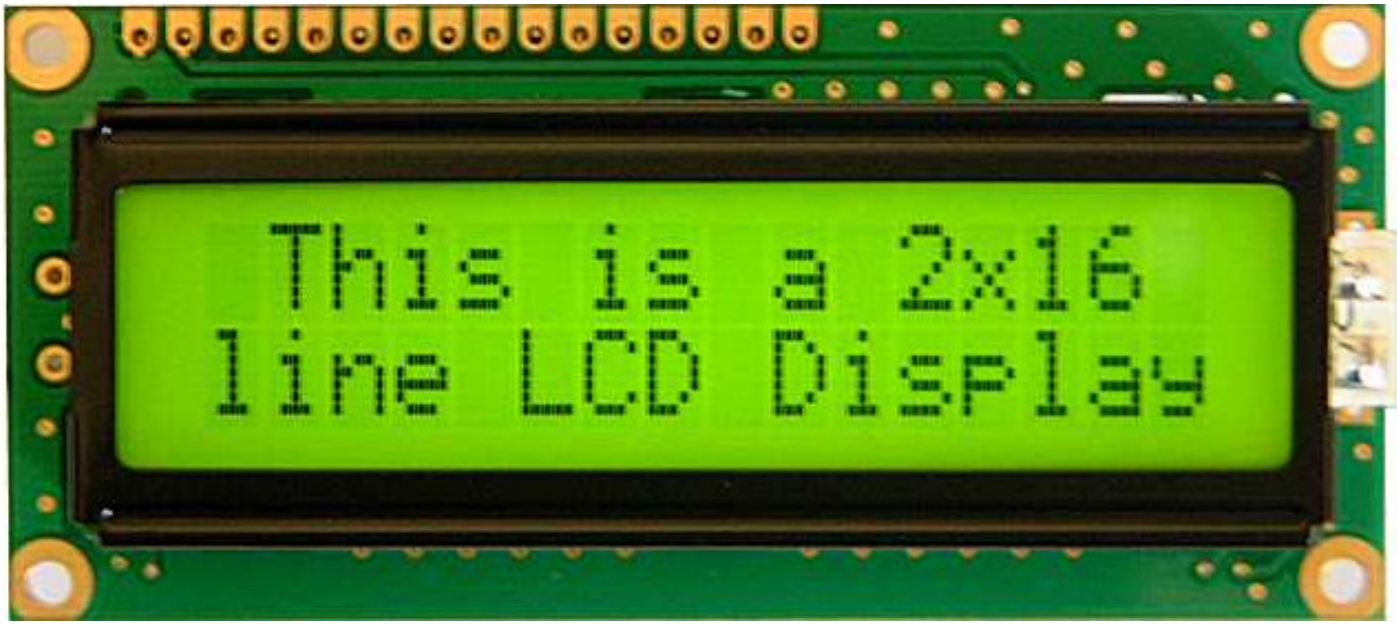
- L'ISR ha priorità sulla funzione loop
  - Quindi se mettete come mode LOW, verrà continuamente eseguita l'ISR e non il loop
- Durante l'esecuzione dell'ISR, il contatore millis() non si incrementa
- Le variabili che devono essere modificate all'interno dell'ISR devono avere il modificatore `volatile`

Esempio Interrupt

```
int pin = 13;
volatile int state = LOW;
void setup() {
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}
void loop() {
  digitalWrite(pin, state);
}
void blink() {
  state = !state;
}
```

Display LCD

I display LCD ci permettono di comunicare facilmente con il “mondo esterno”.



Inizializzazione

- Per prima cosa dobbiamo includere il codice necessario per far funzionare il display LCD

```
#include <LiquidCrystal.h>
```

- Poi dobbiamo dire ad Arduino a che pin è collegato il display: possiamo farlo con la funzione LiquidCrystal

```
LiquidCrystal lcd (rs, enable, d4, d5, d6, d7);
```

- All'interno della funzione `setup()` possiamo poi inizializzare il display con

```
lcd.begin(int columns, int rows);
```

Inizializzazione

- Per prima cosa dobbiamo includere il codice necessario per far funzionare il display LCD

  ```
  #include <LiquidCrystal.h>
  ```

- Poi dobbiamo dire ad Arduino a che pin è collegato il display: possiamo farlo con la funzione LiquidCrystal

  ```
  LiquidCrystal lcd (rs, enable, d4, d5, d6, d7);
  ```

- All'interno della funzione `setup()` possiamo poi inizializzare il display con

  ```
  lcd.begin(int columns, int rows);
  ```

Inizializzazione

- Per prima cosa dobbiamo includere il codice necessario per far funzionare il display LCD

  ```
  #include <LiquidCrystal.h>
  ```

- Poi dobbiamo dire ad Arduino a che pin è collegato il display: possiamo farlo con la funzione LiquidCrystal

  ```
  LiquidCrystal lcd (rs, enable, d4, d5, d6, d7);
  ```

- All'interno della funzione `setup()` possiamo poi inizializzare il display con

  ```
  lcd.begin(int columns, int rows);
  ```

Inizializzazione (2)

- Nel nostro caso abbiamo
  - rs collegato al pin D12
  - e (enable) collegato al pin D11
  - d4...8 collegati rispettivamente ai pin D5...2
  - 16 colonne e 2 righe
- Di conseguenza dovremo scrivere
  `LiquidCrystal lcd (12, 11, 5, 4, 3, 2);`
- e dovremo inizializzare il display con
  `lcd.begin(16, 2);`
- A questo punto il display è inizializzato e possiamo scriverci sopra; come?

Inizializzazione (2)

- Nel nostro caso abbiamo
  - rs collegato al pin D12
  - e (enable) collegato al pin D11
  - d4...8 collegati rispettivamente ai pin D5...2
  - 16 colonne e 2 righe
- Di conseguenza dovremo scrivere
  `LiquidCrystal lcd (12, 11, 5, 4, 3, 2);`
- e dovremo inizializzare il display con
  `lcd.begin(16, 2);`
- A questo punto il display è inizializzato e possiamo scriverci sopra; come?

Inizializzazione (2)

- Nel nostro caso abbiamo
  - rs collegato al pin D12
  - e (enable) collegato al pin D11
  - d4...8 collegati rispettivamente ai pin D5...2
  - 16 colonne e 2 righe
- Di conseguenza dovremo scrivere
  `LiquidCrystal lcd (12, 11, 5, 4, 3, 2);`
- e dovremo inizializzare il display con
  `lcd.begin(16, 2);`
- A questo punto il display è inizializzato e possiamo scriverci sopra; come?

Inizializzazione (2)

- Nel nostro caso abbiamo
  - rs collegato al pin D12
  - e (enable) collegato al pin D11
  - d4...8 collegati rispettivamente ai pin D5...2
  - 16 colonne e 2 righe
- Di conseguenza dovremo scrivere
  `LiquidCrystal lcd (12, 11, 5, 4, 3, 2);`
- e dovremo inizializzare il display con
  `lcd.begin(16, 2);`
- A questo punto il display è inizializzato e possiamo scriverci sopra; come?

Inizializzazione (2)

- Nel nostro caso abbiamo
  - rs collegato al pin D12
  - e (enable) collegato al pin D11
  - d4...8 collegati rispettivamente ai pin D5...2
  - 16 colonne e 2 righe
- Di conseguenza dovremo scrivere
  `LiquidCrystal lcd (12, 11, 5, 4, 3, 2);`
- e dovremo inizializzare il display con
  `lcd.begin(16, 2);`
- A questo punto il display è inizializzato e possiamo scriverci sopra; come?

Inizializzazione (2)

- Nel nostro caso abbiamo
  - rs collegato al pin D12
  - e (enable) collegato al pin D11
  - d4...8 collegati rispettivamente ai pin D5...2
  - 16 colonne e 2 righe
- Di conseguenza dovremo scrivere

```
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
```

- e dovremo inizializzare il display con

```
lcd.begin(16, 2);
```

- A questo punto il display è inizializzato e possiamo scriverci sopra; come?

Inizializzazione (2)

- Nel nostro caso abbiamo
  - rs collegato al pin D12
  - e (enable) collegato al pin D11
  - d4...8 collegati rispettivamente ai pin D5...2
  - 16 colonne e 2 righe
- Di conseguenza dovremo scrivere

```
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
```

- e dovremo inizializzare il display con

```
lcd.begin(16, 2);
```

- A questo punto il display è inizializzato e possiamo scriverci sopra; come?

Inizializzazione (2)

- Nel nostro caso abbiamo
  - rs collegato al pin D12
  - e (enable) collegato al pin D11
  - d4...8 collegati rispettivamente ai pin D5...2
  - 16 colonne e 2 righe
- Di conseguenza dovremo scrivere
  `LiquidCrystal lcd (12, 11, 5, 4, 3, 2);`
- e dovremo inizializzare il display con
  `lcd.begin(16, 2);`
- A questo punto il display è inizializzato e possiamo scriverci sopra; come?

lcd.print

- Per scrivere sul display LCD possiamo usare la funzione `lcd.print(data);`
- `data` può essere sia del testo che una variabile contenente un numero
  - `lcd.print("Hello POuL!")` scrive sul display "Hello POuL!"
  - `lcd.print(numero)` scrive sul display il contenuto della variabile "numero"

lcd.print

- Per scrivere sul display LCD possiamo usare la funzione `lcd.print(data);`
- `data` può essere sia del testo che una variabile contenente un numero
  - `lcd.print("Hello POuL!")` scrive sul display “Hello POuL!”
  - `lcd.print(numero)` scrive sul display il contenuto della variabile “numero”

lcd.print

- Per scrivere sul display LCD possiamo usare la funzione `lcd.print(data);`
- `data` può essere sia del testo che una variabile contenente un numero
  - `lcd.print("Hello POuL!")` scrive sul display “Hello POuL!”
  - `lcd.print(numero)` scrive sul display il contenuto della variabile “numero”

lcd.print

- Per scrivere sul display LCD possiamo usare la funzione `lcd.print(data);`
- `data` può essere sia del testo che una variabile contenente un numero
  - `lcd.print("Hello POuL!")` scrive sul display “Hello POuL!”
  - `lcd.print(numero)` scrive sul display il contenuto della variabile “numero”

lcd.clear

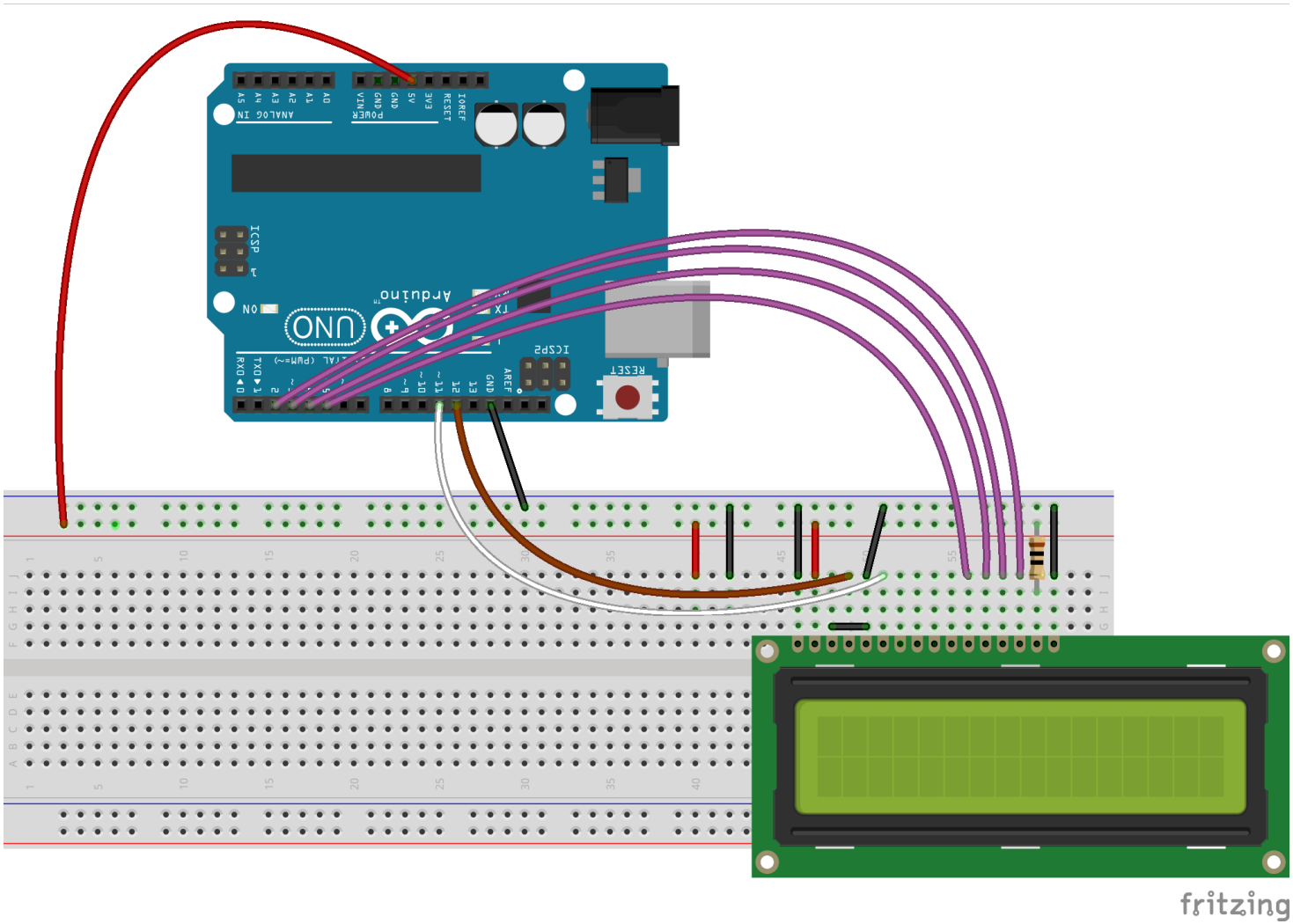
- Possiamo inoltre cancellare tutto quello che c'è sullo schermo con la funzione `lcd.clear()`

LCD

Realizzate un cronometro che ogni secondo scriva il numero di secondi trascorsi dall'accensione di Arduino
Bonus: collegate nuovamente il trimmer dell'esercizio precedente e, in aggiunta ai secondi trascorsi, fate in modo che Arduino scriva anche il valore del pin analogico (magari anche in percentuale) sullo schermo LCD

Go!

Circuito LCD



Resistenza: 10 Ω

Soluzione

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
int contatore = 0;
void setup()  {
  lcd.begin(16, 2);
}
void loop()  {
  lcd.clear();
  lcd.print(contatore);
  ++contatore;
  delay(1000);
}
```

Soluzione con millis()

```
#include <LiquidCrystal.h>
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
int contatore = 0;
int ledPin = 13;
unsigned long inizio;
void setup()  {
  pinMode(ledPin, OUTPUT);
  lcd.begin(16, 2);
}
```

Soluzione con millis() (2)

```
void loop() {
  lcd.clear();
  lcd.print(contatore);
  ++contatore;
  inizio = millis();
  while(millis() - inizio < 1000) {
    digitalWrite(ledPin, HIGH);
    delay(100);
    digitalWrite(ledPin, LOW);
    delay(100);
  }
}
```

Seriale

Abbiamo visto il display LCD, ora vediamo la seriale, una linea dati che ci permette di comunicare con il pc.

- Serial.begin(9600): da inserire nella setup, inizia la comunicazione seriale a baud (velocità) 9600 bit al secondo
- Serial.println("Hello POuL!"): scrive la string "Hello POuL!" sulla seriale e va a capo
- Serial.print("Hello POuL!"): uguale ma non va a capo alla fine

Seriale

Abbiamo visto il display LCD, ora vediamo la seriale, una linea dati che ci permette di comunicare con il pc.

- Serial.begin(9600): da inserire nella setup, inizia la comunicazione seriale a baud (velocità) 9600 bit al secondo
- Serial.println("Hello POuL!"): scrive la string "Hello POuL!" sulla seriale e va a capo
- Serial.print("Hello POuL!"): uguale ma non va a capo alla fine

Seriale

Abbiamo visto il display LCD, ora vediamo la seriale, una linea dati che ci permette di comunicare con il pc.

- Serial.begin(9600): da inserire nella setup, inizia la comunicazione seriale a baud (velocità) 9600 bit al secondo
- Serial.println("Hello POuL!"): scrive la string "Hello POuL!" sulla seriale e va a capo
- Serial.print("Hello POuL!"): uguale ma non va a capo alla fine

Seriale (2)

- Su Arduino UNO la seriale viene scritta sia sulla seriale virtuale (quella collegata al PC) che su quella hardware (collegata ai pin 0 e 1 della board)
- Su Arduino Leonardo, la classe Serial scrive solo sul pc. Per scrivere sulla porta hardware bisogna usare Serial1.
- Oltre a stringhe costanti, ovviamente, si possono stampare variabili (utile per risolvere problemi nel codice)

Seriale (2)

- Su Arduino UNO la seriale viene scritta sia sulla seriale virtuale (quella collegata al PC) che su quella hardware (collegata ai pin 0 e 1 della board)
- Su Arduino Leonardo, la classe Serial scrive solo sul pc. Per scrivere sulla porta hardware bisogna usare Serial1.
- Oltre a stringhe costanti, ovviamente, si possono stampare variabili (utile per risolvere problemi nel codice)

Seriale (2)

- Su Arduino UNO la seriale viene scritta sia sulla seriale virtuale (quella collegata al PC) che su quella hardware (collegata ai pin 0 e 1 della board)
- Su Arduino Leonardo, la classe Serial scrive solo sul pc. Per scrivere sulla porta hardware bisogna usare Serial1.
- Oltre a stringhe costanti, ovviamente, si possono stampare variabili (utile per risolvere problemi nel codice)

Monitor seriale

Per leggere quello che viene scritto dalla seriale, potete usare il monitor seriale dell'Arduino IDE.

- Strumenti → Porta Seriale e selezionate la porta seriale giusta
- Strumenti → Monitor Seriale e selezionate in basso a destra il baud giusto (default 9600)

Monitor seriale

Per leggere quello che viene scritto dalla seriale, potete usare il monitor seriale dell'Arduino IDE.

- Strumenti $\rightarrow$ Porta Seriale e selezionate la porta seriale giusta
- Strumenti $\rightarrow$ Monitor Seriale e selezionate in basso a destra il baud giusto (default 9600)

Esempio seriale

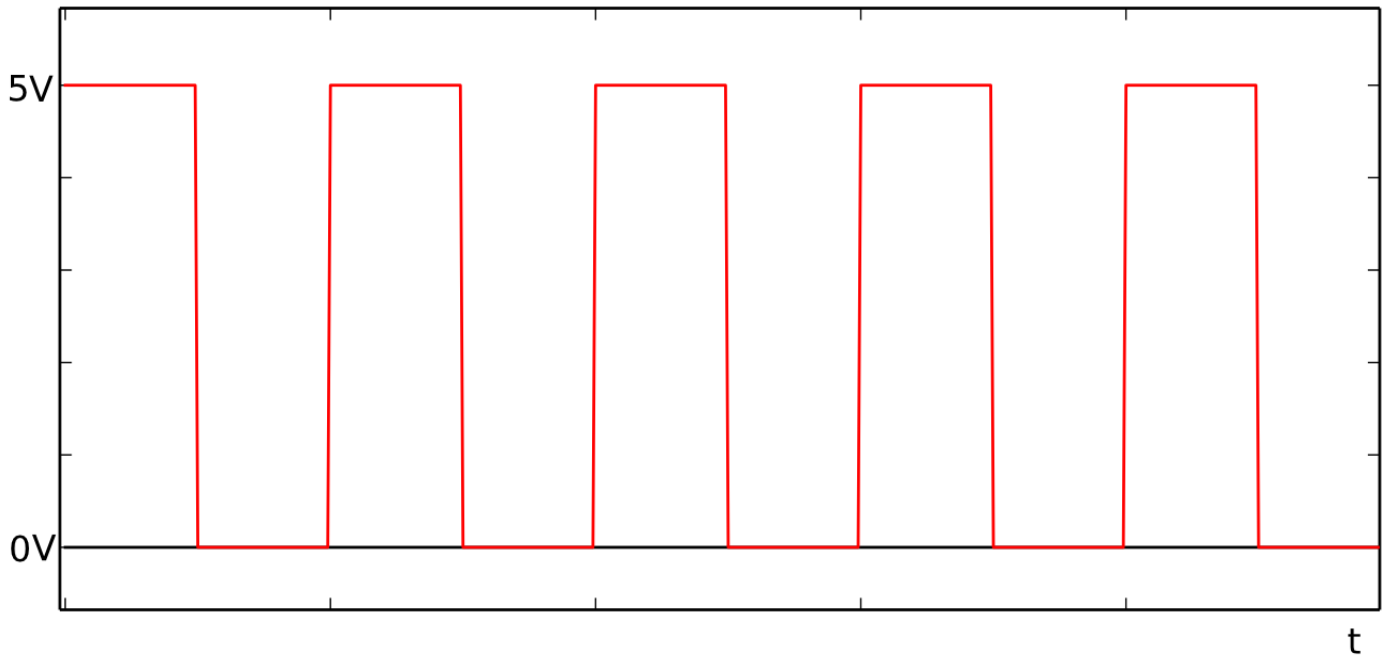
Scrivete un programma che stampa ogni due secondi “Hello ” seguito da un numero che incrementa ogni volta che viene stampato e va a capo. Bonus: fate due stampe, una ogni 2 secondi e una ogni 3, usando la millis(). Fate inoltre in modo che ad ogni trasmissione il LED (pin 13) lampeggi per qualche centinaio di millisecondi.

Soluzione

```
int val = 0;
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Hello ");
  Serial.println(val);
  val++;
  delay(1000);
}
```

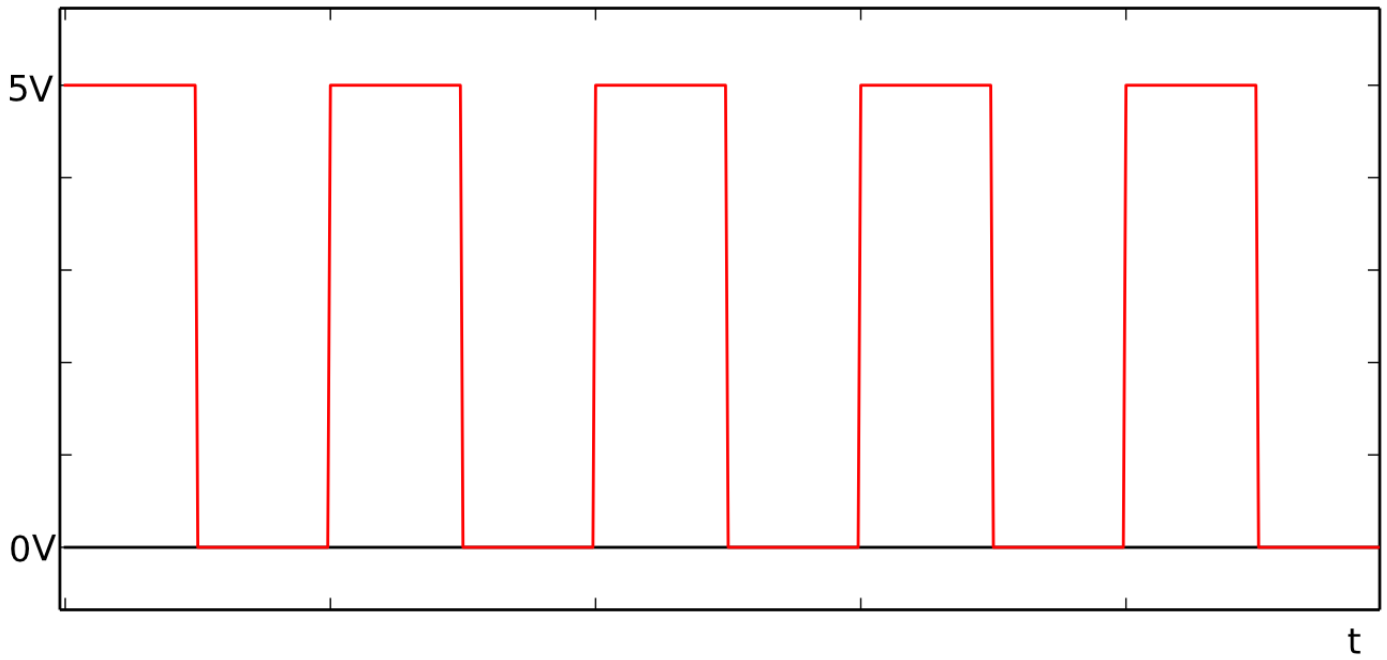
Altoparlante

- Con un po' di circuiteria extra possiamo anche collegare un altoparlante ad Arduino
- Purtroppo Arduino Uno e Leonardo non hanno una "vera" uscita analogica, per cui possiamo generare in modo semplice solo delle onde quadre a frequenze variabili



Altoparlante

- Con un po' di circuiteria extra possiamo anche collegare un altoparlante ad Arduino
- Purtroppo Arduino Uno e Leonardo non hanno una "vera" uscita analogica, per cui possiamo generare in modo semplice solo delle onde quadre a frequenze variabili



tone

- Possiamo generare onde quadre a frequenza variabile con la funzione `tone(pin, frequency);`
  - Solo un'onda quadra alla volta può essere generata
  - L'onda quadra continuerà fino a quando non si chiama la funzione `noTone(pin);`
- In alternativa si può anche specificare la durata con `tone(pin, frequency, duration);`
- L'orecchio umano può percepire frequenze nell'intervallo 20Hz - 20KHz
- Per ragioni tecniche Arduino non può generare frequenze al di sotto di 31Hz

tone

- Possiamo generare onde quadre a frequenza variabile con la funzione `tone(pin, frequency);`
  - Solo un'onda quadra alla volta può essere generata
  - L'onda quadra continuerà fino a quando non si chiama la funzione `noTone(pin);`
- In alternativa si può anche specificare la durata con `tone(pin, frequency, duration);`
- L'orecchio umano può percepire frequenze nell'intervallo 20Hz - 20KHz
- Per ragioni tecniche Arduino non può generare frequenze al di sotto di 31Hz

tone

- Possiamo generare onde quadre a frequenza variabile con la funzione `tone(pin, frequency);`
  - Solo un'onda quadra alla volta può essere generata
  - L'onda quadra continuerà fino a quando non si chiama la funzione `noTone(pin);`
- In alternativa si può anche specificare la durata con `tone(pin, frequency, duration);`
- L'orecchio umano può percepire frequenze nell'intervallo 20Hz - 20KHz
- Per ragioni tecniche Arduino non può generare frequenze al di sotto di 31Hz

tone

- Possiamo generare onde quadre a frequenza variabile con la funzione `tone(pin, frequency);`
  - Solo un'onda quadra alla volta può essere generata
  - L'onda quadra continuerà fino a quando non si chiama la funzione `noTone(pin);`
- In alternativa si può anche specificare la durata con `tone(pin, frequency, duration);`
- L'orecchio umano può percepire frequenze nell'intervallo 20Hz - 20KHz
- Per ragioni tecniche Arduino non può generare frequenze al di sotto di 31Hz

tone

- Possiamo generare onde quadre a frequenza variabile con la funzione `tone(pin, frequency);`
  - Solo un'onda quadra alla volta può essere generata
  - L'onda quadra continuerà fino a quando non si chiama la funzione `noTone(pin);`
- In alternativa si può anche specificare la durata con `tone(pin, frequency, duration);`
- L'orecchio umano può percepire frequenze nell'intervallo 20Hz - 20KHz
- Per ragioni tecniche Arduino non può generare frequenze al di sotto di 31Hz

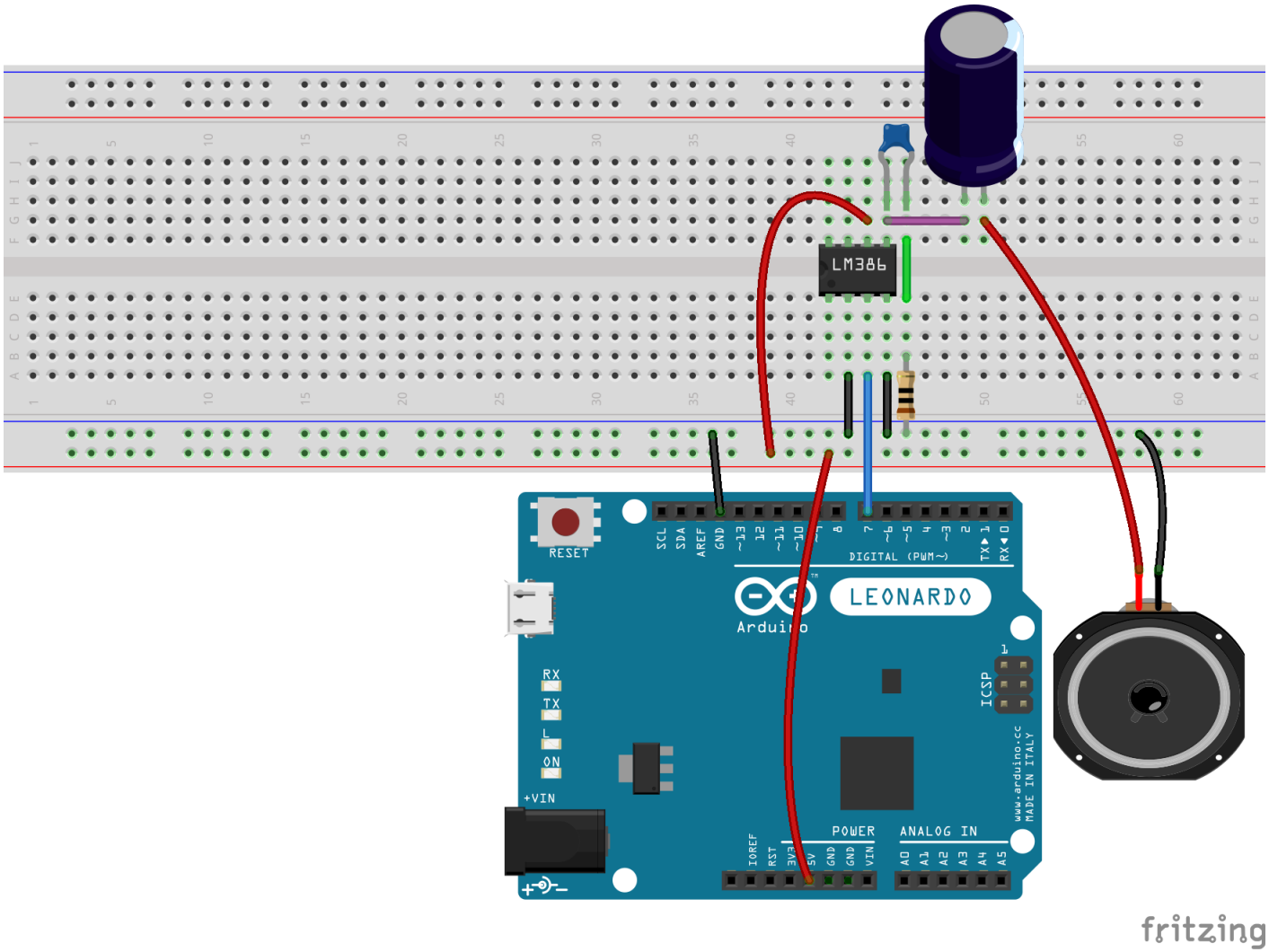
tone

- Possiamo generare onde quadre a frequenza variabile con la funzione `tone(pin, frequency);`
  - Solo un'onda quadra alla volta può essere generata
  - L'onda quadra continuerà fino a quando non si chiama la funzione `noTone(pin);`
- In alternativa si può anche specificare la durata con `tone(pin, frequency, duration);`
- L'orecchio umano può percepire frequenze nell'intervallo 20Hz - 20KHz
- Per ragioni tecniche Arduino non può generare frequenze al di sotto di 31Hz

Esempio altoparlante

Scrivete un programma che faccia suonare l'altoparlante ad una frequenza fissata.
Bonus: fate in modo che ogni secondo la frequenza cambi, con una pausa di un secondo ogni due secondi

Circuito altoparlante



Resistenza: 10 Ω, condensatore piccolo: 50 nF, condensatore grande: 250 μF

Soluzione

```
int pinAltoparlante = 7;
void setup(){
  tone(pinAltoparlante, 100);
}
void loop(){
}
```

Soluzione bonus

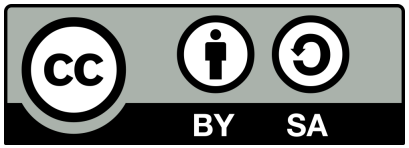
```
int pinAltoparlante = 7;
void setup(){
}
void loop(){
  tone(pinAltoparlante, 100);
  delay(1000);
  tone(pinAltoparlante, 200);
  delay(1000);
  noTone(pinAltoparlante);
  delay(1000);
}
```

Domande?

Se vi vengono in mente più tardi, fatele sul gruppo

Per oggi è tutto

Ci vediamo settimana prossima



`https://www.poul.org`