

Michele Maffucci

... my stories, life and work

Arduino – Interrupts – lezione 2

Pubblicato il [Marzo 4, 2021](#) da [admin](#)

Ormai sono passati diversi anni da quando scrissi il primo post sull'uso degli interrupt e durante gli anni a scuola puntualmente svolgo diversi esercizi, pertanto ho deciso di riprendere ed ampliare quanto detto nel precedente post e sicuramente più avanti ci saranno occasioni per aggiungere ulteriori esempi.

La maggior parte dei microcontrollori permette la gestione degli interrupt. Possiamo pensare all'interrupt come a un sistema che ci permette di rispondere agli eventi "esterni" mentre facciamo qualcos'altro.

Riprendo quindi il post: [Appunti su Arduino: interrupts](#) da cui partire per comprendere cosa sono e come usare gli interrupts con Arduino UNO e con questa lezione voglio proporre alcuni esempi in modo che l'argomento spero possa essere più chiaro.

Per comprendere meglio cos'è un interrupt immaginate la seguente situazione:

tornate a casa dopo la scuola e desiderate cucinarvi un piatto di spaghetti. Fate bollire l'acqua ed inserite gli spaghetti in pentola fissando un tempo di 8 minuti, aspettate che gli spaghetti cucinino e poi mangiate. Possiamo paragonare questa azione all'esecuzione del codice nel loop() di Arduino.

In modo diverso potreste avviare un timer che scade ad 8 minuti e in questo intervallo di tempo potreste guardare il notiziario in TV. Quando il timer suona interromperà (interrupt) la vostra visione che vi ricorderà che dovrete porre attenzione alla pasta in pentola. Quindi verrà eseguito un interrupt (il suono del timer) che permetterà di eseguire il programma: "scola la pasta".

In altro modo, riprendendo ciò che avevo scritto in passato:

Supponete di dover rilevare lo stato di alcuni sensori esterni. All'interno del loop() tutte le istruzioni sono eseguite in modo sequenziale e quindi anche la rilevazione del cambiamento di stato dei sensori collegati ad Arduino avviene in modo sequenziale. Supponete di dover eseguire il controllo della variazione di stato di 3 sensori, il vostro sketch eseguirà il controllo sul primo sensore, sul secondo e poi sul terzo.



Supponete che tutti i sensori si trovino al medesimo stato iniziale che chiameremo S1 e che il controllo di Arduino sia in un determinato istante sul secondo sensore, potrebbe capitare nello stesso istante una variazione repentina di stato sul primo sensore che passa da S1 a S2 e poi a S1, Arduino non si accorgerà di nulla perché la sua attenzione è sul secondo sensore; ecco che in questa situazione potrebbe essere utile l'utilizzo degli interrupt.

In ambito elettronico possiamo avere due tipi di interrupt:

- Interrupt hardware: si verificano in risposta ad un evento esterno, come un pin che assume uno stato HIGH o LOW
- Interrupt software: si verificano in risposta a un'istruzione software.

In questo post ci concentreremo sugli interrupt hardware.

Sulla scheda Arduino UNO, Nano, Mini e altre schede basate sul microcontrollore ATmega328 due sono i pin su cui realizzare un interrupt hardware:

pin digitale 2: interrupt 0

pin digitale 3: interrupt 1

Mentre per altre schede Arduino:

- Uno WiFi Rev.2, Nano Every
tutti i pin sono utilizzabili per l'interrupt
- Mega, Mega2560, MegaADK
2, 3, 18, 19, 20, 21
- Micro, Leonardo e schede basate sull'ATmega32u4
0, 1, 2, 3, 7
- Zero
tutti i pin eccetto il 4
- Tutta la famiglia MKR
0, 1, 4, 5, 6, 7, 8, 9, A1, A2
- Nano 33 IoT
2, 3, 9, 10, 11, 13, 15, A5, A7
- Nano 33 BLE, Nano 33 BLE Sense
tutti i pin
- Due
tutti i pin
- 101
tutti i pin e solo i pin 2, 5, 7, 8, 10, 11, 12, 13 lavorano in modalità CHANGE

Ma come funziona un interrupt?

In estrema sintesi quando si verifica l'interrupt (interruzione), il microcontrollore salva il suo stato di esecuzione attuale ed esegue una piccola porzione di codice che l'utente desidera venga eseguita in presenza di un interrupt, questa porzione di codice prende anche il nome di: **interrupt handler** o **interrupt service routine** (in italiano gestore di interrupt o routine di servizio interrupt).

Il programmatore quindi definisce il codice del gestore di interrupt che deve essere eseguito quando si verifica un interrupt. Per collegare l'interrupt all'interno del programma stesso e per fare ciò in Arduino, come indicato nella mia precedente lezione, utilizziamo una funzione chiamata `attachInterrupt()`:



```
1 | attachInterrupt(digitalPinToInterrupt(PIN), ISR, modo);
```

▪ **digitalPinToInterrupt(PIN)**

è la funzione che consente di convertire il numero del pin su cui effettuare l'interrupt con il numero dell'interrupt quindi nel caso di Arduino UNO si avrà:

`digitalPinToInterrupt (2) > 0`

`digitalPinToInterrupt (3) > 1`

PIN è quindi il pin abilitato all'interrupt, che indica al microcontrollore qual è il PIN da monitorare e come indicato nell'elenco sopra Il PIN dipende dal microcontrollore utilizzato.

▪ **ISR** è la porzione di codice che deve essere eseguito se l'interrupt viene attivato

▪ **modo**

è il tipo di trigger che attiva l'interrupt che, come indicato nella precedente lezione, può essere di 4 tipi:

- **LOW** l'interrupt viene eseguito quando il livello del segnale è basso
- **CHANGE** l'interrupt viene eseguito quando avviene un cambiamento di stato sul pin
- **RISING** l'interrupt viene eseguito quando si passa da un livello LOW ad un livello HIGH
- **FALLING** l'interrupt viene eseguito quando si passa da un livello HIGH ad un livello LOW

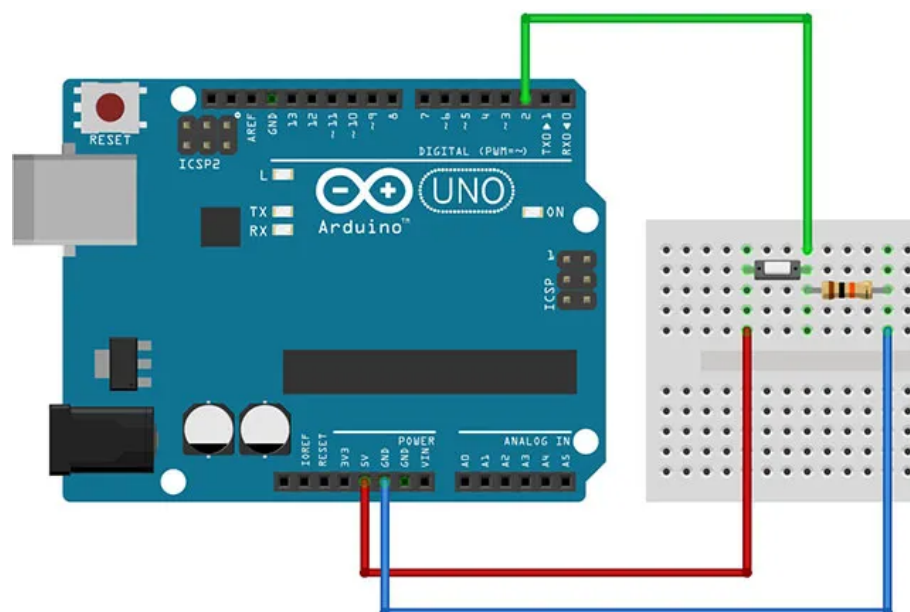
Nota: all'interno della funzione utilizzata in `attachInterrupt`:

- `delay()` non funziona;
- il valore restituito dalla funzione `millis()` non verrà incrementato.
- i dati seriali ricevuti durante l'esecuzione della funzione di interrupt possono essere sbagliati.
- qualsiasi variabile modificabile all'interno della funzione `attached` (chiamata all'interno `attachInterrupt`) devono essere dichiarare come volatili.
- le funzione non può avere parametri di ingresso

Esempio 01

Realizziamo uno sketch che al verificarsi di un interrupt sul pin 2 cambia lo stato in cui si trova il LED sulla scheda di Arduino

Circuito



```

2   Prof. Michele Maffucci
3   Data 03.03.2021
4
5   Oggetto: utilizzo degli interrupt
6
7   Esempio 01
8
9   */
10
11  int pinLed = 13;
12  volatile int stato = LOW;
13  int pulsante = 2;
14
15  /* dichiariamo volatile la variabile
16     state usata nella funzione usata all'interno di attachInterrupt */
17
18  void setup()
19  {
20    pinMode(pinLed, OUTPUT);      // definiamo pin output
21    pinMode(pulsante, INPUT);     // pulsante collegato al pin 2
22    attachInterrupt(digitalPinToInterrupt(2), cambiaStato, FALLING);
23
24    // usiamo l'interrupt 0 che è associato al pin digitale 2
25    // attachInterrupt chiamerà la funzione collegata, cambiaStato
26    // il modo per la rilevazione del cambiamento di stato
27    // sarà di tipo: FALLING
28    // cioè l'interrupt viene eseguito quando si passa
29    // da un livello HIGH ad un livello LOW
30
31  }
32
33  void loop()
34  {
35    digitalWrite(pinLed, stato);
36    // il pin digitale 13 viene impostato a "state"
37    // che può essere LOW o HIGH
38    // all'avvio di Arduino il LED sarà spento
39  }
40
41  void cambiaStato()
42  // la funzione cambiaStato() esegue la funzione NOT di "stato" cioè
43  // se stato = LOW viene cambiato in HIGH, se stato = HIGH viene cambiato in LOW
44
45  {
46    stato = !stato;
47  }

```

quindi il cambiamento di stato avviene solo e soltanto se cambiamo lo stato sul pin 2.

Non si confonda l'azione del cambiamento di stato del LED con l'azione che si potrebbe avere con un semplice controllo sul pin 2 fatto con un'istruzione IF: “se il pulsante è premuto allora cambia stato al LED”.

In questo caso il funzionamento è totalmente diverso, non è presente un'istruzione di controllo di flusso, ma solamente il cambiamento di stato su un pin di interrupt, quando presente questo cambiamento viene invocata la funzione cambiaStato().

Si noti che le variabili utilizzate nella routine di servizio interrupt devono sempre essere globali e volatili. Nel nostro caso la variabile “stato” utilizzata nella routine di servizio interrupt che abbiamo chiamato cambiaStato() è una variabile di tipo globale e volatile.

Ma cosa vuol dire dichiarare una variabile volatile? Come si può notare la variabile “stato” non è presente all'interno del loop(). Se il compilatore si accorge che sono presenti variabili non usate nel setup() e nel loop(), allora in fase di compilazione, per risparmiare memoria queste variabili vengono cancellate.



Dichiarare quindi “stato” volatile garantisce che il compilatore non elimini questa variabile perchè ci servirà

all'interno della routine di servizio interrupt che abbiamo chiamato `cambiaStato()`.

Una variabile volatile indicherà al compilatore di non memorizzare il contenuto della variabile in uno dei registri del microcontrollore, ma di leggerlo quando necessario dalla memoria. Attenzione però che questa modalità di azione rallenterà l'elaborazione del programma, pertanto non bisogna mai rendere volatile ogni variabile, ma l'operazione è da fare solamente quando necessario.

In generale una variabile dovrebbe essere dichiarata come volatile solamente se è utilizzata sia all'interno dell'ISR (interrupt service routine, in italiano gestore di interrupt) che all'esterno dell'ISR.

Esempio 02

Per comprendere meglio quanto scritto nell'esempio precedente facciamo alcune modifiche allo sketch precedente, il circuito rimane invariato.

```
1  /*
2   Prof. Michele Maffucci
3   Data 03.03.2021
4
5   Oggetto: utilizzo degli interrupt
6
7   Esempio 02
8
9  */
10
11 int pin = 13;
12 volatile int stato = LOW;
13 int pulsante = 2;
14
15 /* dichiariamo volatile la variabile
16    state usata nella funzione usata all'interno di attachInterrupt */
17
18 void setup()
19 {
20     pinMode(pin, OUTPUT);    // definiamo pin output
21     pinMode(pulsante, INPUT); // pulsante collegato al pin 2
22     attachInterrupt(digitalPinToInterrupt(2), cambiaStato, FALLING);
23
24     // usiamo l'interrupt 0 che è associato al pin digitale 2
25     // attachInterrupt chiamerà la funzione collegata, cambiaStato()
26     // il modo per la rilevazione del cambiamento di stato
27     // sarà di tipo: FALLING
28     // cioè l'interrupt viene eseguito quando si passa
29     // da un livello HIGH ad un livello LOW
30
31 }
32
33 void loop()
34 {
35     delay(5000); // intervallo di 5 secondi
36 }
37
38 void cambiaStato()
39 // la funzione cambiaStato() esegue la funzione NOT di "stato" cioè
40 // se stato = LOW viene cambiato in HIGH, se stato = HIGH viene cambiato in LOW
41
42 {
43     stato = !stato;
44     digitalWrite(pin, stato);
45     // il pin digitale 13 viene impostato a "stato"
46     // che può essere LOW o HIGH
47     // all'avvio di Arduino il LED sarà spento
48 }
```



L'unica variazione al codice è stato l'inserimento di:

```
1 | delay(5000); // intervallo di 5 secondi
```

che interrompe l'esecuzione del loop per 5 secondi.

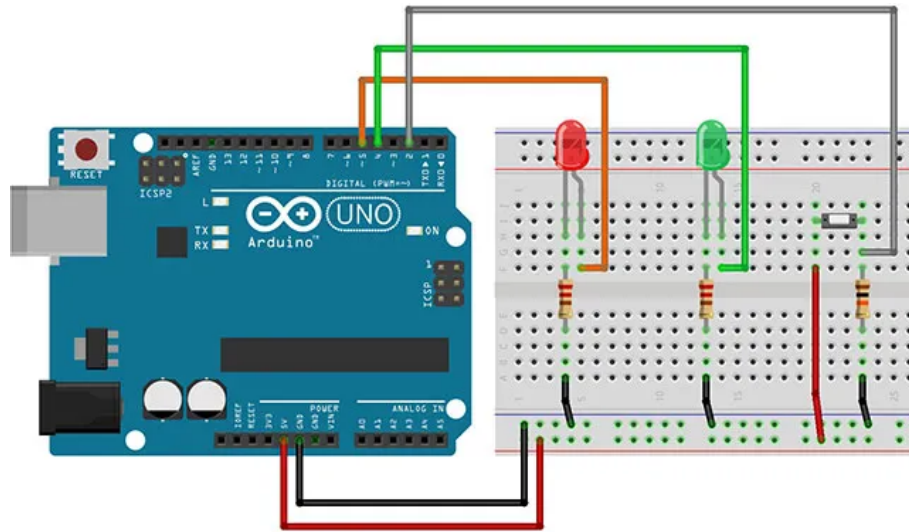
Si noti però che non appena premiamo il pulsante avviene una modifica istantanea del codice indipendentemente dal delay di 5 secondi

Attenzione che il LED rosso poteva essere controllato internamente al loop, ma in questo caso il codice, eseguito sequenzialmente, avrebbe prima eseguito il blink del LED verde e successivamente controlla se il pulsante è premuto per poter accendere il LED verde, ciò non accade con l'uso dell'interrupt.

Esempio 03

Nell'esempio che segue potrete notare il ritardo con cui si manifesta la variazione di stato se il controllo viene effettuato all'interno del loop().

Circuito



```
1  /*
2   Prof. Michele Maffucci
3   Data 03.03.2021
4
5   Oggetto: controllo di stato subordinato dai delay
6
7   Esempio 03
8
9  */
10
11 int pinRosso = 5; // pin a cui è connesso il LED rosso
12 int pinVerde = 4; // pin a cui è connesso il LED verde
13 int pulsante = 2;
14 int stato = LOW;
15
16 void setup()
17 {
18   pinMode(pinRosso, OUTPUT);
19   pinMode(pinVerde, OUTPUT);
20   pinMode(pulsante, INPUT); // pulsante collegato al pin 2
21 }
22
23 void loop()
24 {
25   digitalWrite(pinVerde, HIGH);
26   delay(3000);
27   digitalWrite(pinVerde, LOW);
```



```

28   delay(3000);
29
30   if (digitalRead(pulsante)) {
31       digitalWrite(pinRosso, !stato);
32       stato = !stato;
33   }
34 }
35
36 // Il controllo sul LED rosso potrà avvenire solamente dopo 6 secondi
37 // La pressione ripetuta del pulsante nei primi 6 secondi non modifica
38 // lo stato del LED fino a quando il controllo non giunge all'IF.

```

Esempio 04

Realizziamo ora un quarto sketch in cui abbiamo un LED verde che lampeggia controllato dal codice all'interno del loop ed un secondo LED rosso che cambia stato se premiamo il pulsante connesso al pin 2. Noterete che la pressione del pulsante, che varia lo stato del LED rosso, non influirà sulla normale esecuzione del loop().

Il circuito è il medesimo dell'esercizio precedente.

```

1  /*
2   Prof. Michele Maffucci
3   Data 03.03.2021
4
5   Oggetto: utilizzo degli interrupt
6
7   Esempio 04
8
9  */
10
11 int pinRosso = 5;
12 int pinVerde = 4;
13 int pulsante = 2;
14 volatile int stato = LOW;
15
16 /* dichiariamo volatile la variabile
17    state usata nella funzione usata all'interno di attachInterrupt */
18
19 void setup()
20 {
21     pinMode(pinRosso, OUTPUT);
22     pinMode(pinVerde, OUTPUT);
23     pinMode(pulsante, INPUT);    // pulsante collegato al pin 2
24
25     attachInterrupt(digitalPinToInterrupt(2), cambiaStato, FALLING);
26
27     // usiamo l'interrupt 0 che è associato al pin digitale 2
28     // attachInterrupt chiamerà la funzione collegata, cambiaStato()
29     // il modo per la rilevazione del cambiamento di stato
30     // sarà di tipo: FALLING
31     // cioè l'interrupt viene eseguito quando si passa
32     // da un livello HIGH ad un livello LOW
33
34 }
35
36 void loop()
37 {
38     digitalWrite(pinVerde, HIGH);
39     delay(3000);
40     digitalWrite(pinVerde, LOW);
41     delay(3000);
42 }
43
44 // Durante i 6 secondi di accensione e spegnimento del LED verde
45 // non saremo bloccati dai delay presenti nel loop, potremo modificare
46 // lo stato del LED rosso ogni volta nei 6 secondi di pausa del loop
47

```




```

48 void cambiaStato()
49 // la funzione cambiaStato() esegue la funzione NOT di "stato" cioè
50 // se stato = LOW viene cambiato in HIGH, se stato = HIGH viene cambiato in LOW
51
52 {
53     stato = !stato;
54     digitalWrite(pinRosso, stato);
55     // il pin digitale 5 viene impostato a "stato"
56     // che può essere LOW o HIGH
57     // all'avvio di Arduino il LED sarà spento
58 }

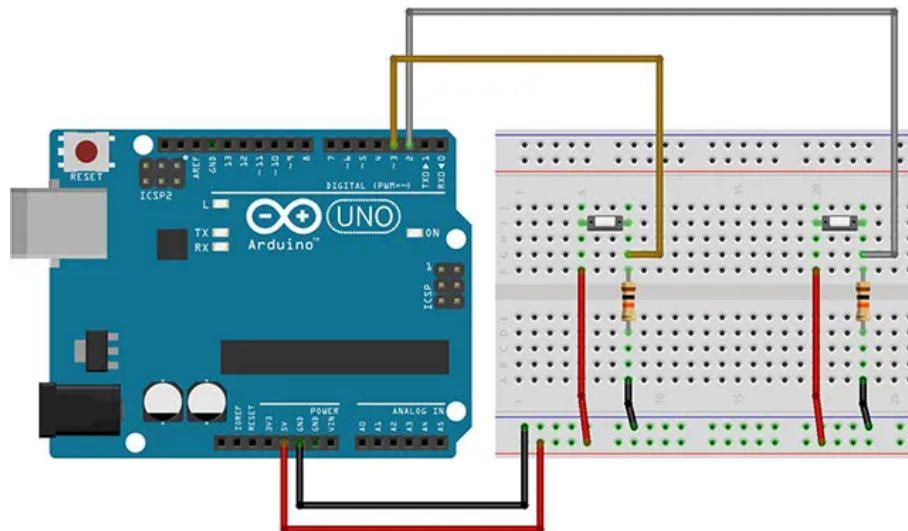
```

Esempio 05

Vediamo ora un'altro esempio dove con due pulsanti controlliamo l'incremento e il decremento di una variabile che viene stampata sulla Serial Monitor.

Notate all'interno del loop() che viene implementato un blink lento e ll'incremento e il decremento della variabile non è influenzata dai delay() del blink, ma risulta immediata.

Circuito



```

1  /*
2   Prof. Michele Maffucci
3   Data 03.03.2021
4
5   Oggetto: utilizzo degli interrupt
6
7   Esempio 05
8
9  */
10
11 volatile int valore = 10;
12
13 int pulsanteIncrementa = 2;
14 int pulsanteDecrementa = 3;
15
16 /* dichiariamo volatile la variabile
17    state usata nella funzione usata all'interno di attachInterrupt */
18
19 void setup()
20 {
21     Serial.begin(9600);
22
23     pinMode(pulsanteIncrementa, INPUT); // pulsante collegato al pin 2
24     pinMode(pulsanteDecrementa, INPUT); // pulsante collegato al pin 3
25
26     attachInterrupt(digitalPinToInterrupt(2), incrementa, FALLING);

```




```
27 attachInterrupt(digitalPinToInterrupt(3), decrementa, FALLING);
28
29 // usiamo l'interrupt 0 e 1 che sono associati ai pin digitali 2 e 3
30 // attachInterrupt chiamerà la funzione collegata, cambiaStato
31 // il modo per la rilevazione del cambiamento di stato
32 // sarà di tipo: FALLING
33 // cioè l'interrupt viene eseguito quando vi è un passaggio da HIGH a LOW
34
35 }
36
37 void loop()
38 {
39   Serial.println(valore);
40 }
41
42 void incrementa() {
43   valore++;
44 }
45
46 void decrementa() {
47   valore--;
48 }
```

Attenzione che durante le prove potreste notare qualche problema nella gestione dei pulsanti su cui dovrebbe essere effettuato un controllo di antirimbato, in questo caso però potrà essere realizzato solamente in modo hardware (in modo elettronico) e non per via software in cui viene richiesto un controllo del tempo di pressione del pulsante mediante il `delay()` o di calcoli di intervalli di tempo e tutto ciò non è permesso all'interno dell'interrupt service routine .

Buon coding a tutti 😊


Articoli Correlati:

1. [Appunti su Arduino: interrupts](#)
2. [Bloccare Arduino ed altre schede alla breadboard](#)
3. [Esercizio Arduino in 5 minuti – usare un Sensor Tracking](#)
4. [Annikken Andee – estendere in modo semplice il controllo di progetti Arduino utilizzando dispositivi mobili](#)
5. [Installare l'Arduino IDE 1.6.X su Ubuntu – metodo 1/2](#)

Questa voce è stata pubblicata in [arduino](#), [i miei allievi](#) e contrassegnata con [arduino](#), [attachInterrupt](#), [digitalPinToInterrupt](#), [interrupt](#), [interrupts](#). Contrassegna il [permalink](#).

Questo sito usa Akismet per ridurre lo spam. [Scopri come i tuoi dati vengono elaborati](#).

Michele Maffucci

 proudly powered by WordPress.

