POLLING ED INTERRUPT CON ARDUINO UNO

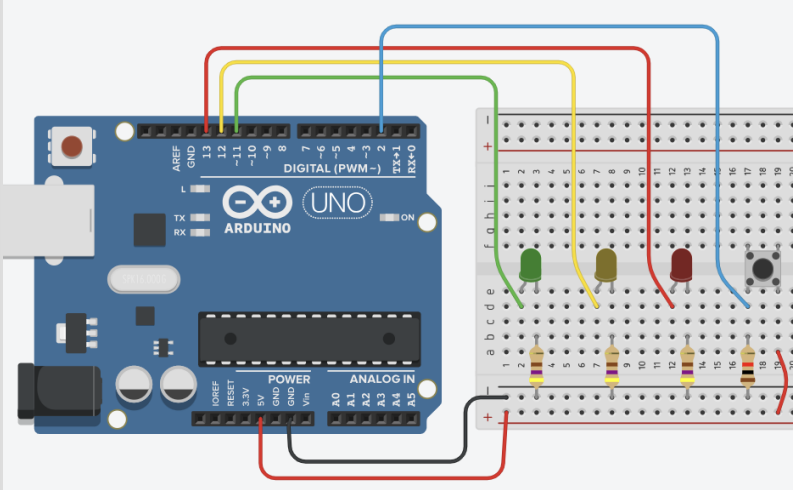
Il **POLLING** è un processo di test ciclico che la CPU esegue, in genere, su un ingresso, per testarne li valore. Il test è un evento **sincrono** che avviene perciò ad istanti ben definiti. Per meglio comprendere il meccanismo facciamo un semplice esempio.

Esempio.

Supponiamo di dover gestire con la nostra scheda Arduino UNO, l'automatismo delle luci di un semaforo ed un pulsante per la chiamata del Rosso che consente l'attraversamento pedonale.

In questo caso la scheda Arduino gestirà 3 uscite (i 3 colori Verde, Giallo e Rosso) ed un ingresso, il pulsante di chiamata pedonale, che farà scattare il Rosso alle autovetture.

Possiamo simulare il sistema con il seguente circuito.



Rosso *= pin* ***13***
Giallo *= pin* ***12***
Verde *= pin* ***11***
Pulsante *= pin* ***2*** *(se il pulsante viene premuto pin2=1)*

Scriviamo innanzitutto un programma che accenda in sequenza le luci senza il controllo del pulsante.

Le prime 4 righe, non sono istruzioni per il microcontrollore, ma ***DIRETTIVE*** *per il compilatore.*

```
#define ROSSO 13
#define GIALLO 12
#define VERDE 11
#define PULSANTE 2
```

Ricordiamo che il compilatore è il software che convertirà il nostro programma in codice macchina.

In questo caso diremo di sostituire tutte le parole ***ROSSO*** *col numero* ***13****, tutte le parole* ***GIALLO*** *col numero* ***12****, tutte le parole* ***VERDE*** *col numero 11 e tutte le parole* ***PULSANTE*** *col numero 2.*

```
void setup() {
  pinMode(ROSSO,OUTPUT);
  pinMode(GIALLO,OUTPUT);
  pinMode(VERDE,OUTPUT);
  pinMode(PULSANTE,INPUT);
}
```

Successivamente definiremo nel SETUP gli ingressi e le uscite.

Nel loop le 3 luci verranno accese in sequenza continuamente. Ma la domanda che dobbiamo porci è: ***"quando dobbiamo leggere il pulsante per fare in modo che in qualsiasi momento esso venga premuto il microcontrollore se ne possa accorgere?"***

```
void loop() {
  digitalWrite(VERDE,HIGH);   //VERDE acceso
  digitalWrite(GIALLO,LOW);   //GIALLO spento
  digitalWrite(ROSSO,LOW);    //ROSSO spento
  delay(30000);               //attesa 30 secondi
  digitalWrite(VERDE,LOW);    //VERDE spento
  digitalWrite(GIALLO,HIGH);  //GIALLO acceso
  digitalWrite(ROSSO,LOW);    //ROSSO spento
  delay(30000);               //attesa 30 secondi
  digitalWrite(VERDE,LOW);    //VERDE spento
  digitalWrite(GIALLO,LOW);   //GIALLO spento
  digitalWrite(ROSSO,HIGH);   //ROSSO acceso
  delay(30000);               //attesa 30 secondi
}
```

La lettura del pulsante all'interno di una struttura condizionale IF, avviene nel seguente modo:

al posto dei puntini dovremo scrivere le istruzioni che vorremmo eseguire quando il pulsante viene premuto.

```
if(digitalRead(PULSANTE)==1){
  ...
}
```

In un ciclo loop come quello scritto sopra, dovunque dovessimo inserire il controllo del pulsante, non riusciremmo ad accorgerci della sua pressione.

Il problema è infatti nel ciclo delay, tutte le altre istruzioni vengono eseguite rapidamente con tempistiche dell'ordine dei microsecondi, ma durante il delay il microcontrollore non svolge altre funzioni, pertanto non potrebbe nemmeno leggere l'ingresso del pulsante.

Volendo fare in modo ad esempio che durante il verde venga controllato il pulsante potremmo applicare la seguente soluzione: suddividere il tempo di 30 secondi in 30000 millesimi di secondo e per ogni piccolo intervallo di 1 msec, testare il pulsante. Questo è un controllo ciclico dell'ingresso che avviene ad istanti ben definiti (sincrono) cioè è un processo di POLLING.

Per gestire questo processo dichiariamo due variabili

Durante l'attesa del VERDE, invece di inserire un unico ritardo di 30 secondi, si esegue un ciclo while con 30000 millesimi di secondo, e tra ogni ritardo si testa il pulsante. Se questo è premuto mettendo cont=0 si esce dal ciclo e si passa al giallo.
Oltre a questo, sempre in caso di pressione del pulsante, viene valorizzata la variabile mem=1 in modo da far durare di più il ROSSO successivamente.

Se la variabile mem=1 si raddoppia la durata del ROSSO.

```
#define ROSSO 13
#define GIALLO 12
#define VERDE 11
#define PULSANTE 2

int cont;
int mem;

void setup() {
  pinMode(ROSSO,OUTPUT);
  pinMode(GIALLO,OUTPUT);
  pinMode(VERDE,OUTPUT);
  pinMode(PULSANTE,INPUT);
}
void loop() {
  digitalWrite(VERDE,HIGH);  //VERDE acceso
  digitalWrite(GIALLO,LOW);  //GIALLO spento
  digitalWrite(ROSSO,LOW);   //ROSSO spento
  //ritardo di 30000 msec = 30sec
  cont=30000;
  mem=0;
  while(cont>0){
    delay(1);
    //test del pulsante se premuto esce dal while
    if(digitalRead(PULSANTE)==1){
      cont=0;
      mem=1;
    }
    else cont--;
  }
  digitalWrite(VERDE,LOW);   //VERDE spento
  digitalWrite(GIALLO,HIGH); //GIALLO acceso
  digitalWrite(ROSSO,LOW);   //ROSSO spento
  delay(30000);              //attesa 30 secondi
  digitalWrite(VERDE,LOW);   //VERDE spento
  digitalWrite(GIALLO,LOW);  //GIALLO spento
  digitalWrite(ROSSO,HIGH);  //ROSSO acceso
  delay(30000);              //attesa 30 secondi
  //prolungo il rosso se è stato premuto il pulsante
  if(mem==1) delay(30000);
}
```

Quanto visto sopra è un classico esempio di **POLLING**, dove il microcontrollore esegue un controllo ciclico ad un tempo ben definito, questa non sempre è una soluzione attuabile e comunque sia la CPU è impegnata continuamente nel test dell'ingresso.
Una soluzione più efficiente ed elegante è quella dell'**INTERRUPT**.

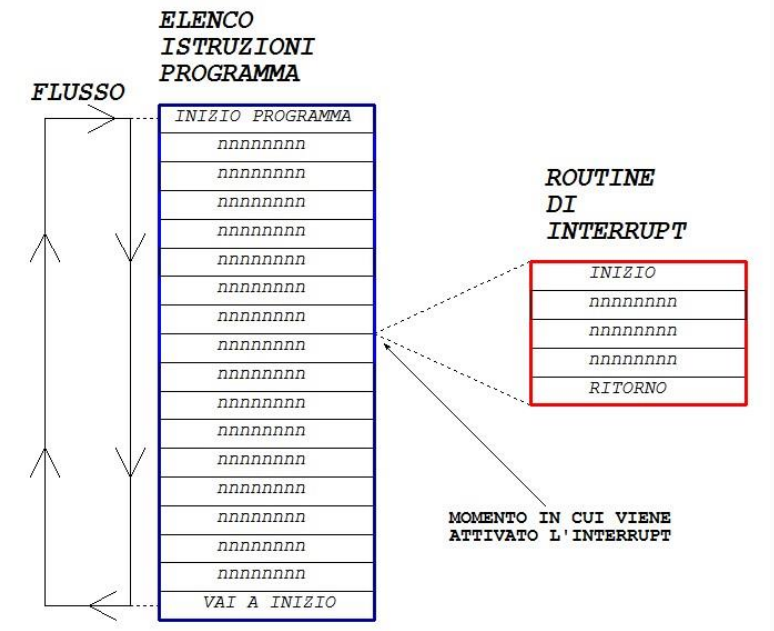
Supponiamo di stare comodamente seduti sul divano a leggere un libro, e nello stesso tempo siamo in attesa di un pacco di un corriere. La lettura del libro richiede la nostra attenzione, e sequenzialmente sfogliamo ogni pagina del libro. Anche l'arrivo del corriere richiederà la nostra attenzione, ed al suo arrivo non potremo continuare a leggere il libro ma dovremo aprire la porta firmare la ricevuta del pacco e tornare alla nostra attività.

Abbiamo essenzialmente i due seguenti modi di procedere:

- Ogni intervallo da me definito, potrebbe essere ad esempio ogni pagina del libro, mi alzo e vado alla porta a controllare l'arrivo del corriere, ed in caso del suo arrivo firmo la ricevuta e torno alla mia attività di lettura. Questa è una modalità "sincrona" richiede cioè un controllo periodico della porta che avviene in tempi ben definiti. **POLLING**.
- Continuo a leggere il libro fino a quando non sento suonare il campanello, a quel punto metto un segnalibro sulla pagina che sto leggendo, mi alzo per firmare la ricevuta e torno come prima alla lettura del libro. In questo caso la modalità non è sincrona, ma asincrona in quanto non c'è nessun intervento periodico da fare, ma un'operazione da effettuare quando viene segnalata la presenza di un corriere alla porta tramite il campanello, che richiede la nostra attenzione. **INTERRUPT**.

L'INTERRUPT pertanto è un controllo ASINCRONO del microcontrollore, che avviene pertanto quando l'evento avviene, nel nostro caso l'evento è la pressione del pulsante.

Immaginando il programma come una sequenza di istruzioni, quando viene generato l'INTERRUPT, il flusso del programma viene interrotto per andare ad eseguire una parte di codice chiamata ROUTINE DI INTERRUPT. Al termine di questa il programma proseguirà il suo flusso dal punto in cui è stato interrotto.



Sulla scheda Arduino UNO, è possibile attivare l'INTERRUPT solamente sui pin 2 e 3, pertanto il pulsante andrà collegato su uno di questi due pin.
L'INTERRUPT può essere generato anche da altri eventi come ad esempio timers, o dati ricevuti sui canali di comunicazione.

Per attivare l'interrupt sulla scheda Arduino UNO, occorre eseguire le seguenti operazioni:

1. Dichiarare le variabili utilizzate nell'INTERRUPT come **volatili**. In pratica le variabili volatili vengono sempre lette dal microcontrollore nella RAM. Normalmente per velocizzare il codice, il compilatore considera le variabili "**non volatili**", cioè immutate se non ci sono righe di codice che le cambiano. Questo potrebbe essere un problema se si utilizza gli INTERRUPT, perché il microcontrollore non adrebbe a leggere l'effettivo valore della variabile in quanto viene ritenuta non modificata, quando invece potrebbe essere modificata nella routine di INTERRUPT.
2. Inserire nel SETUP la funzione attachInterrupt per indicare il pin usato, la routine richiamata ed il tipo di evento, che può essere:
    a. **LOW**, si attiva quando il pin è basso.
    b. **HIGH**, si attiva quando il pin è alto
    c. **CHANGE**, si attiva ogni volta che il pin cambia stato (da alto a basso o viceversa).
    d. **RISING**, si attiva quando il pin passa da basso ad alto (fronte di salita).
    e. **FALLING**, si attiva quando il pin passa da alto a basso (fronte di discesa).
3. Creare la routine di INTERRUPT (Interrupt Service Routine)
   Nella routine di INTERRUPT è buona norma non inserire tante istruzioni.

```
void setup() {
  ...
  attachInterrupt(digitalPinToInterrupt(numero_pin), nomeRoutine, RISING);
  ...
}

// Questa è la ISR (Interrupt Service Routine)
void nomeRoutine() {
  ...
}
```

Sui microcontrollori, gli INTERRUPT si possono associare anche ad altri eventi, non solo allo stato di un pin di ingresso.

Ad esempio nella scheda Aruino UNO, si possono generare INTERRUPT nei seguenti casi:

- **Timer Interrupts:** si può programmare Arduino affinché esegua una funzione ogni tot millisecondi o microsecondi, nella scheda Arduino UNO ci sono 3 Timer indipendenti, Timer0, Timer1 e Timer2
- **ADC Interrupt:** si può generare un Interrupt quando una conversione analogico-digitale è completata.
- **Serial (UART) Interrupt:** si può generare un Interrupt quando arrivano o vengono inviati dati sulla porta seriale.

Per gestire questi eventi differenti non si utilizza la funzione **attachInterrupt**(), ma la gestione avviene scrivendo direttamente nei registri del microcontrollore o utilizzando altre funzioni.

Vediamo ora come applicare il concetto di interrupt nel nostro caso.

Una prima proposta è quella di lasciare il delay normale ed attivare l'INTERRUPT.

Con la funzione ***attachInterrupt****, viene attivato l'interrupt sul pin 2 (PULSANTE) sul fronte di discesa (RISING) e viene dichiarata la routine ad esso associata cioè* ***gestisciPulsante****.*

Se in ogni momento verrà premuto il pulsante, anche durante un delay, verrà eseguita la routine di Interrupt dove verrà solamente assegnto il valore 1 alla variabile mem.

Al termine dopo aver acceso il ROSSO si controllerà la variabile mem e se il suo valore è 1 verrà prolungato il ritardo.

```
#define ROSSO 13
#define GIALLO 12
#define VERDE 11
#define PULSANTE 2

volatile int mem = 0;

void setup() {
  pinMode(ROSSO, OUTPUT);
  pinMode(GIALLO, OUTPUT);
  pinMode(VERDE, OUTPUT);
  pinMode(PULSANTE, INPUT);
  attachInterrupt(digitalPinToInterrupt(PULSANTE), gestisciPulsante, RISING);
}

// Questa è la ISR (Interrupt Service Routine)
void gestisciPulsante() {
  mem = 1; // "Ricorda" che il pulsante è stato premuto
}
void loop() {
  mem=0;
  digitalWrite(VERDE,HIGH);  //VERDE acceso
  digitalWrite(GIALLO,LOW);  //GIALLO spento
  digitalWrite(ROSSO,LOW);   //ROSSO spento
  delay(30000);
  digitalWrite(VERDE,LOW);   //VERDE spento
  digitalWrite(GIALLO,HIGH); //GIALLO acceso
  digitalWrite(ROSSO,LOW);   //ROSSO spento
  delay(30000);              //attesa 30 secondi
  digitalWrite(VERDE,LOW);   //VERDE spento
  digitalWrite(GIALLO,LOW);  //GIALLO spento
  digitalWrite(ROSSO,HIGH);  //ROSSO acceso
  delay(30000);              //attesa 30 secondi
  //prolungo il rosso se è stato premuto il pulsante
  if(mem==1) delay(30000);
}
```

Questa soluzione semplifica notevolmente il programma, ma in caso di pressione del pulsante, il verde avrà sempre la stessa durata.

Se vogliamo che alla pressione del pulsante il delay del verde si interrompa, dobbiamo comunque realizzare un ciclo While insieme alla gestione dell'INTERRUPT..

La parte iniziale è la stessa come anche la routine di INTERRUPT.

In questo caso è stato sostituito il delay del VERDE con un ciclo While della durata di 30 secondi (300 cicli da 100msec). Se viene premuto il pulsante la variabile mem viene messa ad 1 ed il ciclo si interromperà facendo scattare il GIALLO e facendo comunque durare il ROSSO del doppio del tempo come fatto prima.

```
#define ROSSO 13
#define GIALLO 12
#define VERDE 11
#define PULSANTE 2
volatile int mem = 0;
int cont;

void setup() {
  pinMode(ROSSO, OUTPUT);
  pinMode(GIALLO, OUTPUT);
  pinMode(VERDE, OUTPUT);
  pinMode(PULSANTE, INPUT);
  attachInterrupt(digitalPinToInterrupt(PULSANTE), gestisciPulsante, RISING);
}

void gestisciPulsante() {  // Questa è la ISR (Interrupt Service Routine)
  mem = 1; // "Ricorda" che il pulsante è stato premuto
}
 void loop() {
  digitalWrite(VERDE,HIGH);  //VERDE acceso
  digitalWrite(GIALLO,LOW);  //GIALLO spento
  digitalWrite(ROSSO,LOW);   //ROSSO spento
  cont=300;
  mem=0;
  while(cont>0){
    delay(100);
    cont--;
    if(mem==1) cont=0;
  }
  digitalWrite(VERDE,LOW);   //VERDE spento
  digitalWrite(GIALLO,HIGH); //GIALLO acceso
  digitalWrite(ROSSO,LOW);   //ROSSO spento
  delay(30000);              //attesa 30 secondi
  digitalWrite(VERDE,LOW);   //VERDE spento
  digitalWrite(GIALLO,LOW);  //GIALLO spento
  digitalWrite(ROSSO,HIGH);  //ROSSO acceso
  delay(30000);              //attesa 30 secondi
  //prolungo il rosso se è stato premuto il pulsante
  if(mem==1) delay(30000);
}
```

Come detto in precedenza, la scheda Arduino UNO, ha due pin che possono essere configurati come INTERRUPT, il pin 2 (INT0) ed il pin 3 (INT1).

Nel caso si dovessero configurare entrambi occorre dichiarare due routine di Interrupt come di seguito indicato:

```
void setup() {
  ...
  attachInterrupt(digitalPinToInterrupt(2), routine_pin2, RISING);
  attachInterrupt(digitalPinToInterrupt(3), routine_pin3, RISING);
}

void routine_pin2(){
  ...
}

void routine_pin3(){
  ...
}
```

In caso i due segnali dovessero arrivare allo stesso tempo vale la gerarchia della priorità.

Il segnale sul pin 2 è associato all'interrupt chiamato INT0 ed è quello con la priorità più alta e pertanto in caso di simultaneità con l'altro pin, verrà eseguita per prima la routine associata al pin 2.

Possono pertanto verificarsi due condizioni:

- Arrivano due segnali contemporaneamente sui due pin, entrambi configurati come Interrupt.
  In questo caso viene eseguita prima la routine del pin 2 (INT0) e poi quella del pin 3 (INT1).
- Mentre è in esecuzione una delle due routine arriva un segnale sull'altro pin.
  In questo caso qualsiasi sia la routine in esecuzione, verrà prima terminata per poi eseguire quella associata all'altro interrupt.
