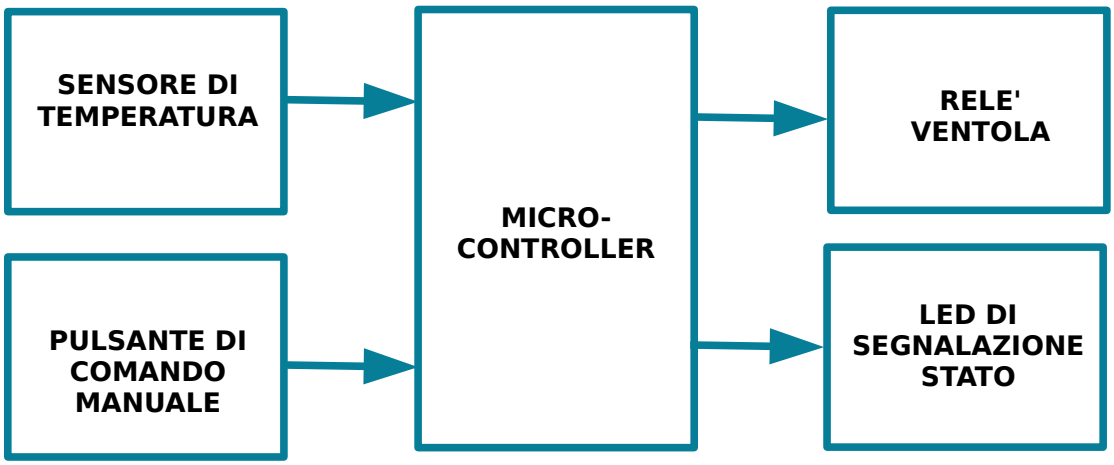
SENSORI, ATTUATORI, ELABORAZIONE E CONTROLLO

In questo documento viene proposto un semplice ma completo esempio di attività locale di un microcontroller.

In genere un microcontroller ha la responsabilità di acquisire informazioni dall'ambiente in cui è inserito attraverso i sensori, deve elaborare i dati grezzi ricevuti per ottenere informazioni significative e comunicarle in uscita eventualmente attuando un controllo automatico o manuale.

ESEMPIO DI STRUTTURA DI ACQUISIZIONE, ELABORAZIONE E CONTROLLO

L'esempio molto semplice prevede un controllo automatico e manuale di temperatura.

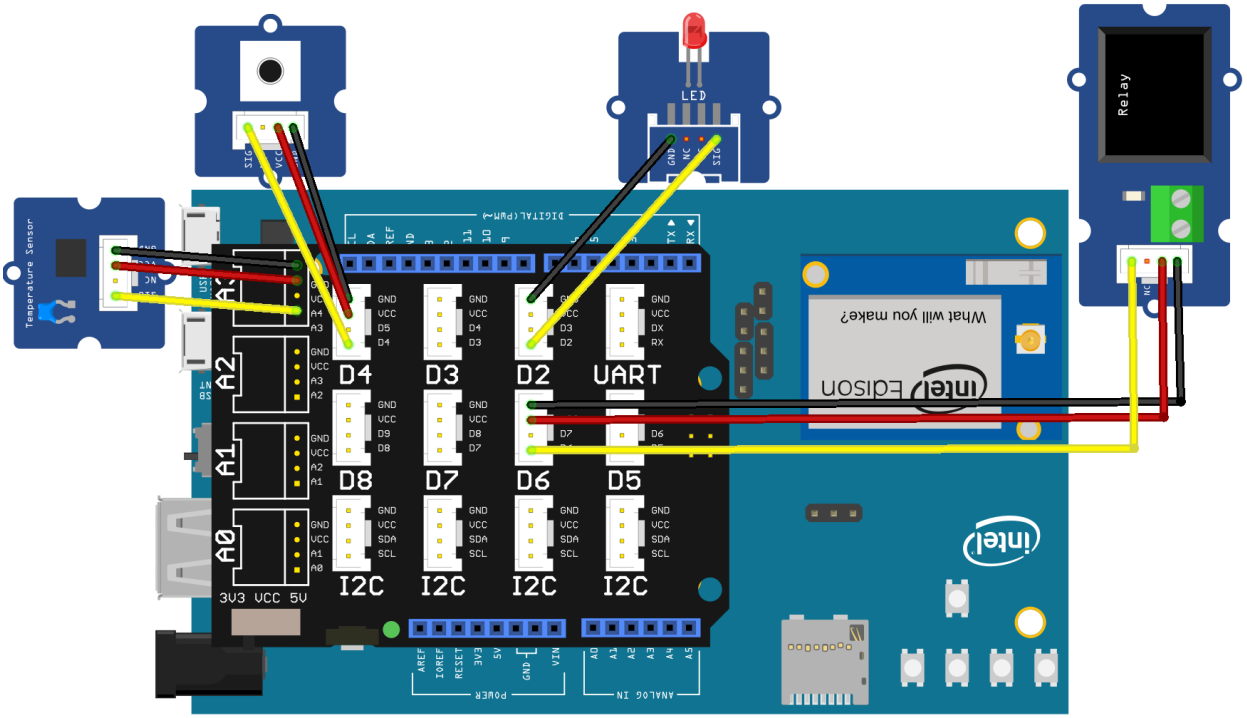


Il microcontroller acquisisce la temperatura dell'ambiente da un sensore di temperatura ed attua la rotazione di una ventola attraverso un rele' se la temperatura sale oltre una soglia predefinita. La ventola viene fermata automaticamente quando la temperatura scende sotto la soglia.

Il microcontroller acquisce anche un pulsante di comando manuale che, operando in modo toggle può accendere incondizionatamente la ventola indipentemente dalla temperatura.

Un led di segnalazione indica se il pulsante è stato premuto.

Nella versione Edison for Arduino + Grove IoT si ottiene il seguente schema di montaggio:



ALGORITMO DI ACQUISIZIONE, ELABORAZIONE E CONTROLLO

Nell'ambiente di sviluppo di Arduino la struttura del programma si compone di due parti:

- **setup()**: funzione eseguita solo una volta all'accensione del dispositivo; qui vanno inserite tutte le operazioni di inizializzazione che vanno eseguite solo una volta come la configurazione dell'hardware degli I/O, l'inizializzazione di eventuali variabili di stato, la connessione alla rete.

- **loop()**: funzione eseguita ripetutamente durante il funzionamento del dispositivo. La frequenza con cui la funzione viene eseguita è variabile e dipende dal tempo impiegato per eseguire tale funzione. L'efficacia della risposta in "tempo reale" del sistema dipende proprio da questa frequenza di ripetizione. E' quindi necessario che nella funzione loop non siano presenti algoritmi che richiedono lunghi tempi di esecuzione o algoritmi "sospensivi" che attendono il verificarsi di un evento esterno per terminare.

Non è quindi possibile inserire nella funzione loop() cicli for con limiti molto elevati, chiamate alla funzione delay() con valori alti, cicli while che attendo il

verificarsi di una condizione che potrebbe verificarsi dopo tempi molto lunghi o anche mai.

Al posto di tali algoritmi si deve inserire una struttura del tipo "automa a stati finiti": se una azione non può essere completata con una breve sequenza va spezzata in più sequenze che vengono eseguite in successivi lanci della funzione loop(); per regolare l'esecuzione delle sequenze successive si definisce una "variabile di stato"; si tratta di una variabile statica (mantiene il suo valore in lanci successivi della funzione loop e può essere una variabile globale oppure un attributo di una classe) che va inizializzata nella funzione setup() e può cambiare il suo valore in successive esecuzioni della funzione loop() influenzando il comportamento dell'automa.

Alcuni esempi di questa tecnica sono:

- **Realizzazione di un ritardo non sospensivo**
  Se si deve realizzare un tempo di attesa molto lungo non è possibile usare la funzione delay() perché questa sospende l'esecuzione della funzione loop() impendendo alle altre parti di programma di funzionare. Si può utilizzare la funzione millis() che restituisce, senza sospensione, il tempo passato a partire dall'ultima accensione espresso in millisecondi. Si crea una variabile di stato statica di tipo long destinata a contenere il valore dell'istante inziale del ritardo:
  ```
  long start;
  ```
  Quando si vuole iniziare il ritardo si inizializza start al valore corrente di millis()
  ```
  start=millis();
  ```
  In ogni loop si verifica, rileggendo millis() se il tempo passato rispetto a "start" supera il tempo di ritardo desiderato:
  ```
  if( millis()- start > RITARDO) {
      //azione per ritardo scaduto
  }
  ```
  N.B.: dopo il termine di un ritardo è necessario reinizializzare la variabile di stato (se si vuole fare un altro ritardo) o disabilitare il controllo (se non si vuole fare un altro ritardo) per evitare che nei loop successivi la condizione continui ad essere verificata.

- **Riconoscimento della pressione di un pulsante**
  Se si deve riconoscere l'evento di pressione di un pulsante, ad esempio per realizzare un contatore o per avviare una azione, non si può leggere semplicemente lo stato "attivo" del segnale perché lo stato attivo viene riletto più volte dal loop durante la pressione del pulsante che, seppur breve dura molto rispetto alla frequenza di ripetizione del loop.

La soluzione consiste nel confrontare lo stato del segnale in un loop con lo stato nel loop precedente facendo l'azione solo quando si rileva una variazione.
E' necessario definire una variabile di stato statica destinata a contenere il valore del segnale nel loop precedente:

```
int old;
```

La variabile va inizializzata al valore corrente del segnale nella funzione setup():

```
old=digitalRead(INGRESSO);
```

Ad ogni loop si deve confrontare il valore attuale del segnale con con il valore nel loop precedente; se i due segnali sono diversi si è verificata una transizione; in questo caso è possibile verificare se la transizione è positiva (da LOW ad HIGH) o negativa (da HIGH a LOW).
L'operazione può essere fatta in un unico test:

```
int new=digitalRead(INGRESSO);
if ((new!=old) && (new==HIGH) {
   //riconosciuta transizione positiva
}
```

N.B.: ad ogni loop è necessario aggiornare il valore della variabile di stato old con il valore attuale in modo che il sistema sia pronto per riconoscere una successiva transizione:

```
old=new;
```

Molte librerie di supporto di dispositivi sono dotate di strumenti che garantiscono la non sospensione del processo.
Ad esempio:

- Ricezione dal canale seriale
  Sebbene la funzione read() di Serial non sia sospensiva (restituisce -1 se non ci sono caratteri disponibili in ricezione non è conveniente chiamarla incondizionatamente ad ogni loop perché i tempi di comunicazione seriale sono molto lenti rispetti alla frequenza di loop.
  La classe Serial è dotata di un metodo available() che restituisce il numero di caratteri disponibili nel buffer di ricezione e consente di chiamare la read() solo quando ci sono caratteri nel buffer.

```
if(Serial.available()>0) {
    rxChar=Serial.read();
   //azioni sulla ricezione di un carattere
}
```

ESEMPIO DI CODICE

Il seguente esempio mostra una implementazione del controller su hardware Intel Edison for Arduino + Grove IoT con ambiente di sviluppo IDE di Arduino

Costanti simboliche

Le costanti simboliche sono delle definizioni che consentono di parametrizzare le caratteristiche del sistema come posizioni dei pin di I/O, tipologia dell'hardware, valori da utilizzare nell'algoritmo

Definizione degli ingressi

Definisce la posizione del pulsante di comando

```
#define PULSANTE   4
```

Definizione delle uscite

Definisce la posizione del relay e del led di stato

```
#define RELAY      6
#define STATO      2
```

Definizione dell'ingresso analogico

Una compilazione condizionale consente di commutare rapidamente tra un sensore di temperatura di tipo NTC ed un potenziometro per la simulazione
Con la definzione NTC compila la versione operativa con sensore NTC; commentando la definizione NTC si compila la versione di test con un potenziometro al posto del sensore.

```
#define NTC
#ifdef NTC
#define SENSORE    A3          //sensore NTC
#else
#define SENSORE    A0          //simulazione con potenziometro su A0
#endif
```

Definizione delle costanti di elaborazione

Definisce in modo simbolico le costanti usate dall'algoritmo di elaborazione

```
#define SOGLIA 25.0            //soglia di avvio del relay
#define AUT 0                  //modo automatico
#define MAN 1                  //modo manuale
#define INTERVALLO 1000        //campionamento diagnostica

const float B = 4275.0;        //cofficiente moltiplicativo
const float R0 = 100000.0;     //cofficiente adittivo
```

Variabili di stato
Le variabili di stato sono variabili statiche che devono mantenere il loro valore in successivi lanci della funzione loop.
Queste variabili possono essere definite variabili globali come in questo esempio oppure possono essere attributi di classi.

```
int oldPuls;    //stato precendente del pulsante
int stato;      //modo automatico (0) o manuale (1)
long start;     //tempo iniziale dell'intervallo di campionamento
int relay;      //stato del relay
```

Inizializzazione
Nella funzione setup() vengono inserite le azioni da svolgere solo una volta all'avvio del programma come:

- l'inizializzazione del canale seriale per la diagnostica
- la configurazione degli ingressi e delle uscite
- l'inizializzazione delle variabili di stato
- Il prompt di diagnostica

Inizializzazione del canale seriale per la diagnostica
Il canale seriale di default viene aperto per la diagnostica locale

```
Serial.begin(9600);
```

Configurazione degli ingressi e delle uscite
I pin utilizzati dal progetto vengono configurati in ingresos o uscita

```
pinMode(PULSANTE,INPUT_PULLUP);
pinMode(RELAY,OUTPUT);
pinMode(STATO,OUTPUT);
```

Inizializzazione delle variabili di stato
Le variabili di stato vengono inizializzate al loro valore iniziale

```
oldPuls=digitalRead(PULSANTE);      //simula valore prec. Puls.
stato=AUT;                          //modo automatico
start=millis();                     //inizio intervallo di camp.
```

Prompt di diagnostica
Avvia la diagnostica locale su canale seriale

```
Serial.println("Hackathon SMD18");
```

Ciclo
Nella funzione loop() vanno inserite le azioni non sospensive che devono essere ripetute per ottenere l'algoritmo di controllo.
Trattandosi di una struttura ciclica non ha importanza l'ordine in cui le varie azioni sono disposte.

Acquizione della temperatura
Il valore grezzo (da 0 a 1023) viene acquisito dal canale analogico. In base allo switch di compilazione condizionale può essere applicata la formula di Steinerhart-Hart (proporzionalità inversa dell'NTC) o la mappatura di un potenziometro di simulazione per ottenere il valore di temperatura.

Legge di Steinerhart-Hart:

$$T = \frac{1}{\frac{1}{T_0} + \frac{1}{B}\log_e(\frac{R}{R_0})}$$

```
ingAnalog=analogRead(SENSORE);
#ifdef NTC                                          //sensore NTC
float R = 1023.0/((float)ingAnalog)-1.0;            //formula
R = 100000.0*R;                                     //di
temp=1.0/(log(R/R0)/B+1/298.15)-273.15;             //Steinerhart-Hart
#else                                               //potenziomatro
temp=map(ingAnalog,0,1023,-100.0,100.0);            //mappatura prop.
#endif
```

Verifica della soglia e attuazione relay se in automatico
Se lo stato del sistema è in automatico il relay viene pilotato in base al confronto con la soglia. Per evitare indecisioni nello scatto quando la temperatura è intorno alla soglia.
Il comando manuale ha la priorità su quello automatico quindi in caso di modo manuale il controllo di soglia non viene effettuato

```
if(stato==AUT) {    //modo automatico
    if (temp > SOGLIA + 1.0) {     //sopra la soglia
       relay=HIGH;
    }
    else if (temp < SOGLIA - 1.0){ //sotto la soglia
       relay=LOW;
    }
    digitalWrite(RELAY,relay);     //attiva relay
}
```

Comando manuale
Il comando manuale viene attivato/disattivato in commutazione da successive pressione del pulsante. Se lo stato è automatico una pressione porta il sistema in modo manuale (relay sempre acceso) mentre se lo stato è manuale una pressione porta il sistema in modo automatico (relay comandato dalla soglia).
Il riconoscimento della pressione (evento transizione positiva del segnale) viene realizzato confrontando lo stato del segnale nel loop precedente con quello nel loop attuale. Se i due stati sono diversi e se lo stato attuale è HIGH c'è stata una transizione positiva e quindi lo stato viene commutato.

```
int puls=digitalRead(PULSANTE);          //lettura ingresso attuale
  if((puls!=oldPuls)&&(puls==HIGH)){     //fronte positivo
    if(stato==AUT) {                     //commuta in MAN
      stato=MAN;                         //accende
      relay=HIGH;
    }
    else {                               //commuta in AUT
      stato=AUT;                         //spegne
      relay=LOW;
    }
    digitalWrite(RELAY,relay);
    digitalWrite(STATO,stato);
  }
  oldPuls=puls;                          //aggiorna stato prec.
```

Diagnostica
Il loop viene eseguito alla massima velocità possibile ma questo porterebbe ad una presentazione della diagnostica di difficile lettura.
I dati diagnostici vengono mandati sulla seriale solo ad intervalli regolari di tempo definiti dalla costante simbolica INTERVALLO (1 secondo nell'esempio)
All'inizio di ogni intervallo viene inzializzata la variabile start con il valore corrente di millis(). L'azione di diagnostica viene fatta quando è passato un tempo pari o superiore ad INTERVALLO

```
  if(millis()-start> INTERVALLO){
    Serial.print("ingr.grezzo: ");
    Serial.println(ingAnalog);
    Serial.print("temperatura: ");
    Serial.println(temp);
    Serial.print("modo: ");
    Serial.println(stato);
    start=millis();
  }
```