Laboratorio TECNOLOGIE PER IOT – HW PARTE 1

Gatti Gabriele s245636

Iobbi Amedeo s228856

Esercizio 1.1/1.2

```
#include <TimerOne.h>

const unsigned int RLED_PIN = 11;
const unsigned int GLED_PIN = 12;

const float R_HALF_PERIOD = 1.5;
const float G_HALF_PERIOD = 3.5;

volatile unsigned int greenLedState = LOW;
unsigned int redLedState = LOW;

void setup() {
  pinMode(GLED_PIN, OUTPUT);
  Serial.begin(9600);
  while(!Serial);
  Serial.println("Connessione seriale avviata!");
  Timer1.initialize(G_HALF_PERIOD * 1e06);
  Timer1.attachInterrupt(blinkGreen);
}

void loop() {
  serialPrintStatus();
  redLedState = !redLedState;
  digitalWrite(RLED_PIN, redLedState);
  delay(R_HALF_PERIOD * 1e03);
}

void serialPrintStatus(){
  if(Serial.available() > 0){
    unsigned int input = Serial.read();

    if(input == 'R'){
      Serial.print("Stato led RED: ");
      Serial.println(!redLedState);
    }else if(input == 'G'){
      Serial.print("Stato led GREEN: ");
      Serial.println(greenLedState);
    }else{
      Serial.println("Comando invalido");
    }
    // eventuali caratteri oltre al primo vengono ignorati
    while(Serial.available() > 0)
        Serial.read();
  }
}
```

```
void blinkGreen(){
  greenLedState = !greenLedState;
  digitalWrite(GLED_PIN, greenLedState);
}
```

Lo sketch implementa l'accensione sincrona e asincrona di due LED, il primo, associato al PIN 11 della scheda Arduino viene gestito in maniera sincrona mediante il metodo *loop* nel quale viene acceso per R_HALF_PERIOD secondi e spento dopo lo stesso ammontare di tempo, questa temporizzazione viene regolata dalla funzione *delay.*

Il secondo LED, associato al PIN 12 viene gestito asincronamente mediante Timer1, un timer interno alla scheda Arduino che scatena una ISR (in questo caso la funzione *blinkGreen*) al termine di G_HALF_PERIOD secondi. Questa routine si occupa di cambiare lo stato del LED (acceso/spento) nel valore opposto rispetto al precedente.

La funzione *serialPrintStatus* viene aggiunta dall'esercizio 1.2 all'esercizio 1.1 e si occupa di rispondere ad un eventuale richiesta di informazioni da parte dell'utente sullo stato di uno dei due LED. Siccome uno dei due LED viene gestito in maniera sincrona, è necessario fornire all'utente il valore che esso assumerà subito dopo l'esecuzione della funzione, ovvero l'opposto del precedente, così da fornire una descrizione più accurata del sistema (chiamando la funzione dopo l'aggiornamento dello stato del LED comporterebbe la stessa problematica, essendo in un loop esso verrebbe cambiato nell'istruzione successiva).

Vengono utilizzate variabili *unsigned* dove possibile per contenere (seppure in maniera minima) l'utilizzo di memoria.

Esercizio 1.3

```
volatile unsigned int count = 0;
volatile bool active = false;
const unsigned int LED_PIN = 12;
const unsigned int PIR_PIN = 7;

void setup() {
  pinMode(PIR_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(PIR_PIN), checkPresence, CHANGE);
  while(!Serial);
  Serial.println("Connessione seriale avviata!");
}
void loop() {
  delay(30000);
  Serial.print("Total count: ");
  Serial.println(count);
}
void checkPresence(){
  if(!active){
    count++;
    active = !active;
    digitalWrite(LED_PIN, HIGH);
  }else{
    active = !active;
    digitalWrite(LED_PIN, LOW);
  }
}
```

In questo esercizio è richiesta la gestione di un sensore di movimento di tipo PIR, essendo un sensore digitale, l'individuazione di un movimento corrisponde all'aumento di tensione sul PIN di output del sensore, il PIN 7 della scheda Arduino viene pertanto configurato in modalità di INPUT e successivamente convertito in un INTERRUPT, ovvero ogni cambiamento di tensione rilevato dal suddetto PIN (a cui verrà collegato l'output del sensore) scatenerà un interrupt che verrà gestito dalla ISR *checkPresence*.

Siccome ogni cambiamento di tensione determina un interrupt (sia il fronte di salita che quello di discesa del segnale, pertanto) è necessario utilizzare una variabile che determini lo stato del sensore, la variabile *active*, inizializzata al valore logico `false` viene impostata a `true` dall'interrupt corrispondente al fronte di salita della tensione, sempre su questo fronte la ISR si occupa di incrementare il conteggio di rilevazioni e di attivare il LED di stato, istanziato sul PIN 12 della scheda. Nel successivo interrupt la suddetta variabile verrà invertita e il LED spento trattandosi necessariamente del fronte di discesa.

I potenziometri del sensore vengono impostati al valore minimo per quanto riguarda il *delay* così da poter effettuare rilevazioni in tempo utile, la sensibilità invece viene impostata ad un valore intermedio

Esercizio 1.4

```
unsigned int current_speed = 0;
const unsigned int FAN_PIN = 6;

void setup() {
  Serial.begin(9600);
  while(!Serial);
  Serial.println("Ready!");
  Serial.println("+ to increase speed");
  Serial.println("- to decrease speed");
  pinMode(FAN_PIN, OUTPUT);
  analogWrite(FAN_PIN, current_speed);
}

void loop() {
  changeSpeed();
}

void changeSpeed(){
  if(Serial.available() > 0){
    unsigned int input = Serial.read();

    if(input == '+'){
      if(current_speed == 255){
         Serial.println("Max speed reached");
      }else{
        current_speed += 51;
        analogWrite(FAN_PIN, current_speed);
        Serial.print("Setting duty cycle to: ");
        Serial.print((float)current_speed/255*100);
        Serial.println("%");
      }
    }else if(input == '-'){
      if(current_speed == 0){
         Serial.println("Min speed reached");
```

```
      }else{
        current_speed -= 51;
        analogWrite(FAN_PIN, current_speed);
        Serial.print("Setting duty cycle to: ");
        Serial.print((float)current_speed/255*100);
        Serial.println("%");
      }
    }else{
      Serial.println("Comando invalido");
    }
    // eventuali caratteri oltre al primo vengono ignorati
    while(Serial.available() > 0)
        Serial.read();
  }
}
```

In questo esercizio è richiesto il controllo di un attuatore mediante monitor seriale, più nello specifico è richiesto di aumentare o diminuire la velocità di una ventola a seconda del comando ricevuto.

Viene scelto uno step pari a 51 unità in quanto divisore del valore massimo "scrivibile" in un registro PWM (255), così da semplificare i controlli sul valore massimo e ridurre al minimo le possibilità di comportamenti indesiderati.

Lo sketch è costituito principalmente dalla funzione *changeSpeed* ripetuta all'infinito nella funzione *loop.* Essa si occupa di leggere l'input dalla console seriale, controllare la velocità attuale e incrementarla o decrementarla a seconda del comando ricevuto, assicurandosi che non sia negativa o superiore al valore massimo, per poi "scriverla" sul PIN a cui è collegato l'attuatore mediante la funzione *analogWrite*.

Esercizio 1.5

```
#include <math.h>

const int TEMP_PIN = A1;
const long int R0 = 100000; //Ohm
const int B = 4275; //K
const float Vcc = 1023.0;
float R = 0.0;
float T = 0.0; //Kelvin

void setup() {
  Serial.begin(9600);
  while(!Serial);
  pinMode(TEMP_PIN,INPUT);
  Serial.println("Starting Lab 1.5!");
}

void loop() {

  int sensorValue = analogRead(TEMP_PIN); // = Vsig

  R = ((Vcc/sensorValue) - 1.0)*R0;
  T = (1/((log(R/R0)/B)+(1/298.15))) - 273.15; // Temperatura in gradi celsius

  Serial.print("Temperature: ");
  Serial.println(T);
  delay(5000);
}
```

In questo esercizio è richiesto il monitoraggio della temperatura in un ambiente attraverso il sensore e di stampare il valore trovato sul monitor seriale.

Il sensore in base alla temperatura presente, nel nostro caso nella stanza, produce in output un valore di tensione. Attraverso questo valore di tensione e dai valori del termoresistore e dalla relazione tra resistenza e temperatura, questi ultimi due valori si trovano sul datasheet del sensore, riusciamo a calcolare la temperatura presente nella stanza. Da notare che il valore ottenuto della temperatura è in gradi Kelvin, quindi dobbiamo sottrarre 273.15K per ottenere il valore in gradi Celsius.

Esercizio 1.6

```
#include <LiquidCrystal_PCF8574.h>
#include <math.h>
#include <Wire.h>

LiquidCrystal_PCF8574 lcd(0x27);

const int TEMP_PIN = A1;

const long int R0 = 100000; //Ohm
const int B = 4275; //K
const float Vcc = 1023.0;

float R = 0.0;
float T = 0.0; //Kelvin

void setup() {
  int error;
  Serial.begin(9600);
  while (!Serial);

  Serial.println("Starting Lab 1.5!");
  Serial.println("Check for LCD");

  Wire.begin();
  Wire.beginTransmission(0x27);
  error = Wire.endTransmission();

  if (error == 0) {
    Serial.println("LCD found");

    lcd.begin(16, 2); // initialize the lcd
    lcd.setBacklight(255);
    lcd.home();
    lcd.clear();
    lcd.print("Temperature:");

  } else {
    Serial.println("LCD not found.");
  }
}

void loop() {

  int sensorValue = analogRead(TEMP_PIN); // = Vsig

  R = ((Vcc / sensorValue) - 1.0) * R0;
  T = (1 / ((log(R / R0) / B) + (1 / 298.15))) - 273.15;

  lcd.setCursor(12, 0);
  lcd.print(T);

  delay(5000);
}
```

Questo esercizio è uguale al precedente, ovvero bisogna sempre monitorare i valori di temperatura dell' ambiente attraverso il sensore, ma con la differenza che i valori non bisogna più stamparli sul monitor seriale, ma su un display LCD.

Per interfacciarci e per poter stampare/mostrare i valori sul display LCD utilizziamo la libreria LiquidCrystal_PCF8574.h e anche la libreria Wire.h che ci permette di comunicare con dispositivi I2C

Nel setup effettuiamo dei controlli per verificare che il display sia collegato correttamente e disponibile, in caso affermativo inizializziamo LCD definendo le dimensioni dello schermo, il colore di background, posizioniamo il cursore sulla prima riga e colonna e stampiamo la scritta "temperature" che rimarrà fissa per tutta l'esecuzione del programma.

Nel loop per stampare solo i valori di temperatura sul LCD senza dover scrivere tutte le volte "temperature" utilizziamo la funzione setCursor() che ci permette di posizionare il cursore esattamente alla fine della scritta "temperature".