Laboratorio TECNOLOGIE PER IOT – HW PARTE 2

Gatti Gabriele s245636

Iobbi Amedeo s228856

Parte 1

```
#define TEMP_PIN A1
#define LED_PIN 5 //Pin PWN della scheda
#define FAN_PIN 6

typedef struct
{
  int tmin;
  int tmax;
  int tminP;
  int tmaxP;
} Temp;

Temp acTemp;
Temp htTemp;

float temp;
int fanSpeed;
int fanLCD;
int ledIntensity;
int ledLCD;

int acTmin;
int acTmax;
int htTmin;
int htTmax;

// Inizializzo i sensori con le temperature massime e minime
void setupTemp(Temp* temp, int tmin, int tmax, int tminP, int tmaxP) {
  temp->tmin = tmin;
  temp->tmax = tmax;
  temp->tminP = tminP;
  temp->tmaxP = tmaxP;
}

//Funzione per leggere i valori di temperatura dal sensore
float readTemp() {

  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

  return T;
}
void setup() {
  Serial.begin(9600);
  while (!Serial);

  //Inizializzo il motore e pongo la velocità a 0
  pinMode(FAN_PIN, OUTPUT);
  analogWrite(FAN_PIN, 0);

  //Inizializzo il sensore di temperatura
  pinMode(TEMP_PIN, INPUT);
```

```
  //Inizializzo il LED
  pinMode(LED_PIN, OUTPUT);
  analogWrite(LED_PIN, 0);
  }

  Serial.println("Starting Lab HW 2");
  Serial.println("Che valori di temperatura vuoi utilizzare? (Scrivere il numero
corrispondente nel monitor serial, è possibile cambiare opzione in qualsiasi momento
dell' esecuzione)");
  Serial.println("\tA) Valori impostati in automatico dal programma per i due casi
(stanza vuota e stanza con persone)");
  Serial.println("\tB) Impostare manualmente i valori");

}
void loop(){
//Leggo la temperatura
    temp = readTemp();
//Le temperature sono state scelte per risparmiare energia, se non c'è nessuno in casa
abbiamo un range più alto per evitare che il condizionatore
        //stia sempre acceso anche quando non ci siamo.
        setupTemp(&acTemp, 27, 35, 20, 26);

/*====================================== Punto 1
=========================================*/
    if (temp < acTmin) {
      //Siamo al di sotto della temperatura minima quindi la ventola non fa nulla
      fanSpeed = 0;
      fanLCD = 0;
      analogWrite(FAN_PIN, fanSpeed);
    }
    else if (temp >= acTmin && temp <= acTmax) {

      //Valore di temperatura all'interno del range per il condizionatore
      //Calcolo la velocità della ventola
      fanSpeed = map(temp, acTmin, acTmax, 0, 255);
      fanLCD = map(temp, acTmin, acTmax, 0, 100);
      analogWrite(FAN_PIN, fanSpeed);

#if DEBUG
      Serial.print("Speed: ");
      Serial.println(fanSpeed);
#endif
    }
    else if ( temp > acTmax) {
      //Siamo al di sopra della temperatura massima quindi la ventola la lascio al
massimo
      fanSpeed = 255;
      fanLCD = 100;
      analogWrite(FAN_PIN, fanSpeed);
    }

/*=============================================================================
=====*/
delay(1000);
}
```

Nel primo punto del laboratorio bisogna misurare la temperatura presente nell' ambiente e in base alla temperatura rilevata bisogna far attivare una ventola. Il valore di temperatura viene confrontato in un range di valori massimo e minimo di temperatura. Se la temperatura misurata è dentro al range attraverso la funzione map che riceve come parametri i due valori minimo e massimo della temperatura e i valori minimo e massimo a cui il motore può funzionare ci mappa la temperatura corrente con un valore da inviare al motore. Stessa cosa facciamo per calcolare la percentuale alla quale sta girando il motore+ventola che ci servirà in un punto successivo. Se la temperatura misurata è minore o

maggiore del set point minimo o del set point massimo mettiamo il motore rispettivamente a velocità minima (0) o a velocità massima (255).

I valori passati sono quattro, due in caso che la stanza sia vuota, due in caso di presenza di persone nella stanza. Questa situazione verrà analizzata in un punto successivo dell' esercizio.

Per memorizzare il range di temperature utilizziamo una struttura.

Parte 2

```
  /*==================================== Punto 2
=============================================*/
    if (temp < htTmin) {
      //Siamo al di sotto della temperatura minima quindi led è a potenza massima
      ledIntensity = 255;
      ledLCD = 255;
      analogWrite(LED_PIN, ledIntensity);
    }
    else if (temp >= htTmin && temp <= htTmax) {

      //Valore di temperatura all'interno del range per il riscaldatore
      //Calcolo la intensità del led

      ledIntensity = map(temp, htTmax, htTmin, 0, 255);
      ledLCD = map(temp, htTmax, htTmin, 0, 100);
      analogWrite(LED_PIN, ledIntensity);

#if DEBUG
      Serial.print("Led: ");
      Serial.println(ledIntensity);
#endif
    }
    else if ( temp > htTmax) {
      //Siamo al di sopra della temperatura massima quindi led è a potenza minima
      ledIntensity = 0;
      ledLCD = 0;
      analogWrite(LED_PIN, ledIntensity);
    }
```

Nel secondo punto del laboratorio dobbiamo sempre misurare la temperatura della stanza e confrontarlo con il nostro range di temperature, in questo caso simuliamo un dispositivo per riscaldare la stanza e per farlo utilizziamo un LED che in base alla temperatura aumenta o diminuisce la sua intensità, più è bassa la temperatura più alta sarà l'intensità del LED e viceversa. Per regolare l'intensità del LED lo colleghiamo ad un pin PWN della scheda arduino.

Il codice per calcolare il valore del LED è praticamente identico a quello del punto precedente, con la differenza che se la temperatura è al di sotto del valore minimo setteremo il LED alla potenza massima (255) se invece e oltre alla soglia massimo lo terremo spento. Anche in questo caso calcoliamo il valore percentuale che utilizzeremo in un punto successivo.

Parte 3

```
bool pir_presence = false;
unsigned int pir_timeout = 300; //300s --> 5 minuti
unsigned int pir_time = 0;
const unsigned int PIR_PIN = 8;
```

```
void setup() {
  pinMode(PIR_PIN, INPUT);

  ...

  while(!Serial);
  Serial.println("Connessione seriale avviata!");
}

bool checkPIR(){
  if(digitalRead(PIR_PIN)){
    pir_presence = true;
    pir_time = millis()/1000;
    //Serial.println("Individuata persona");
  }else{
    if(millis()/1000 - pir_time > pir_timeout){
      pir_presence = false;
      //Serial.println("Nessuno presente");
    }
  }
  return pir_presence;
}
```

In questa sezione era richiesta l'implementazione del sensore di movimento PIR e la gestione del rilevamento concorde con eventuali tempistiche, configurabili mediante variabili.

La gestione del sensore avviene in modo sincrono, pertanto senza l'utilizzo di interrupt, date le limitazioni della scheda (un solo pin configurabile come input rimanente D7, in quanto D0 e D1 sono in parallelo alle linee di comunicazione con il sottosistema Linux, mentre D2 e D3 sono utilizzate dal display I2C) è stato deciso di utilizzare questa funzionalità nell'implementazione del sensore di rumore, richiesta nelle sezioni successive. Pertanto la funzione `checkPIR` verrà richiamata all'interno del loop (non direttamente ma tramite una funzione wrapper, come vedremo in seguito).

La suddetta funzione si occupa di leggere il valore di tensione presente sul pin a cui il sensore è collegato, in caso di potenziale alto alla variabile booleana `pir_presence`, viene assegnato il valore logico `true`, inoltre viene memorizzato il tempo della rilevazione (i millisecondi trascorsi dall'avvio della scheda Arduino) mediante la funzione `millis`, valore che viene convertito in secondi. Nel caso in cui la lettura risulti in un valore di bassa tensione invece si calcola il tempo trascorso dall'ultima rilevazione (memorizzato nella variabile `pir_time`) come differenza `millis/1000 - pir_time`, se questa è maggiore di `pir_timeout` significa che non è stata percepita nessuna presenza in un lasso di tempo superiore al timeout impostato e pertanto `pir_presence` viene impostata a `false`.

Il valore della variabile booleana viene infine ritornato per essere utilizzato dalla funzione wrapper della sezione 5.

Le variabili potrebbero essere dichiarate localmente nello scope della funzione `checkPIR` ma si è deciso di dichiararle globalmente, secrificando memoria all'interno della scheda, per renderle accessibili ad altre funzioni in caso fosse necessario interagire con altri sistemi in laboratori futuri

Parte 4

```
#include <TimerOne.h>

volatile bool noise_presence = false;
volatile unsigned int noise_timeout = 1200; //1200s --> 20 minuti
volatile unsigned int noise_time = 0;
```

```
volatile unsigned int noise_interval = 600; //600s --> 10 minuti
volatile unsigned int noise_count = 0;
volatile unsigned int noise_events = 100;

volatile unsigned int timer_counter = 0;

const unsigned int NOISE_PIN = 7;

void setup() {
  pinMode(NOISE_PIN, INPUT);

  attachInterrupt(digitalPinToInterrupt(NOISE_PIN), checkNOISE, FALLING);
  Timer1.initialize(8e6);
  Timer1.attachInterrupt(noiseTimeout);

  while(!Serial);
  Serial.println("Connessione seriale avviata!");
}

void checkNOISE(){
  unsigned int t = millis()/1000;

  if(t - noise_time < noise_interval){
    noise_count++;
    //Serial.print("Evento: ");
    //Serial.println(noise_count);
  }else{
    noise_count = 0;
    noise_time = t;
  }

  if(noise_count > noise_events){
    noise_count = 0;
    noise_time = t;
    noise_presence = true;
    //Serial.println("Rilevata presenza");
    timer_counter = 0;
    Timer1.restart();
  }
}

//timer1 può essere inizializzato a max 8 secondi,
//il problema si aggira utilizzando il timer più volte
void noiseTimeout(){
  timer_counter++;

  if(timer_counter > noise_timeout/8){
    //Serial.println("Reset noise presence");
    noise_presence = false;
    timer_counter = 0;
    Timer1.stop();
  }
}
```

In questa sezione è richiesta l'implementazione del sensore di rumore per la rilevazione della presenza nei punti ciechi del sensore PIR. Come menzionato in precedenza le rilevazioni di questo sensore avvengono in modo asincrono, configurando un interrupt sul pin 7 in modo che si scateni in corrispondenza dei fronti di discesa del segnale (poiché quando il sensore percepisce un suono il suo output è a basso potenziale); esso viene gestito dalla ISR *checkNOISE*.

La funzione di callback dell'interrupt si occupa innanzitutto di memorizzare il tempo in cui viene rilevato il rumore, successivamente determina se la rilevazione avviene all'interno dell'intervallo di validità (è infatti necessario rilevare un numero di suoni pari a `noise_events` all'interno di un intervallo di `noise_interval` secondi affinchè venga determinata una presenza). In caso affemativo viene incrementato il contatore di eventi `noise_count`, altrimenti quest'ultimo viene azzerato e il tempo di ultima rilevazione `noise_time` viene inizializzato al tempo corrente, in modo da avviare un nuovo intervallo di rilevazione.

Al termine della funzione se il numero di rilevazioni supera il valore di soglia, la variabile flag `noise_presence` viene impostata a `true` e contatore e tempo di rilevazione vengono inizializzati nuovamente per dare inizio ad un nuovo conteggio di presenza.

È necessario però gestire anche il timeout nel caso in cui nessuna presenza sia rilevata dopo `noise_timeout` secondi. Non è infatti possibile farlo mediante la ISR poiché essa viene eseguita solo quando un rumore viene rilevato. Si utilizza pertanto un timer fornito dalla libreria `TimerOne`, esso si occuperà di scatenare un ulteriore interrupt dopo un tempo prestabilito. Il timer viene inizializzato nel setup ad un periodo di 8 secondi (massimo valore a cui è possibile inizializzare tale oggetto) e gli viene legata la funzione *noiseTimeout*. Quest'ultima incrementa un contatore, `timer_counter`, ogni volta che viene richiamata, questo avverrà per un totale di `noise_timeout/8` volte e, poiché tra una chiamate e l'altra trascorrono 8 secondi per via del timer, dopo che saranno trascorsi `noise_timeout` secondi la variabile `noise_presence` verrà impostata a `false`, determinando che non sono state rilevate presenze in un intervallo di tempo pari al timeout, dopo di ciò il timer verrà arrestato per essere poi riavviato alla successiva rilevazione di presenza (fino a quel momento non sarà necessario utilizzare risorse per modificare lo stato della rilevazione in quanto sarà sempre `false`).

La sensibilità del sensore viene impostata ad un valore intermedio tale da non percepire il rumore della ventola delle sezioni precedenti ma comunque in grado di percepire voci, movimenti di porte, schiocchi di dita e battiti di mani.

Parte 5

```
void loop() {
    Serial.println(checkPresence());
    delay(10000);
}

bool checkPresence(){
  return checkPIR() || noise_presence;
}
```

I risultati delle due sezioni precedenti vengono elaborati dalla funzione *checkPresence*, essa viene eseguita in modo sincrono tramite il loop e si occupa di verificare ogni 10 secondi se almeno uno dei due segnali ha rilevato una presenza restituendo il risultato dell'or logico tra i valori booleani ottenuti nelle sezioni precedenti. In particolare il primo, corrispondente alla rilevazione del sensore PIR viene ottenuto richiamando la funzione *checkPIR*, cosa non necessaria con il secondo sensore in quanto gestito mediante interrupt.

Parte 6 – 8

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

  Serial.println("Starting Lab HW 2");
  Serial.println("Che valori di temperatura vuoi utilizzare? (Scrivere il numero
corrispondente nel monitor serial, è possibile cambiare opzione in qualsiasi momento
dell' esecuzione)");
  Serial.println("\tA) Valori impostati in automatico dal programma per i due casi
(stanza vuota e stanza con persone)");
  Serial.println("\tB) Impostare manualmente i valori");

}

void loop() {

  /* Punto 8 */
  if (Serial.available() > 0)
  {
    switch (Serial.read())
    {
      case 'A':
        Serial.println("Valori di default attivi");
        //Le temperature sono state scelte per risparmiare energia, se non c'è nessuno
in casa abbiamo un range più alto per evitare che il condizionatore
        //stia sempre acceso anche quando non ci siamo.
        setupTemp(&acTemp, 27, 35, 20, 26);
         //Le temperature sono state scelte per risparmiare energia, se non c'è nessuno
in casa abbiamo un range più basso per evitare che il riscaldamento
        //stia sempre acceso anche quando non ci siamo.
        setupTemp(&htTemp, 10, 15, 16, 21);
        running = true;
        break;

      case 'B':
        Serial.println("Inserimento valori manuali attivo");
        Serial.println("Inserire tutti i valori in un unica riga sia per l'AC che per
l'HT separati da una spazio");
        Serial.println("Hai 40 secondi per inserire i valori");
        Serial.println("acTminS acTmaxS acTminP acTmaxP htTminS htTmaxS htTminP
htTmaxP");

        running = false;
        delay(40000);

        int i = 0;
        while (Serial.available() > 0) {

          // read the incoming byte:
          if (i < 8) {
            newData[i] = Serial.parseInt();
            i++;
          } else {
            i = 0;
          }
        }
        setupTemp(&acTemp, newData[0], newData[1], newData[2], newData[3]);
        setupTemp(&htTemp, newData[4], newData[5], newData[6], newData[7]);
        running = true;
        Serial.println("B");
        break;
    }
  }
}
```

In questa parte del laboratorio dobbiamo considerare se nella stanza c'è qualcuno, in base della presenza o meno di qualcuno dobbiamo modificare i 4 valori dei range di temperatura. I valori del range di temperatura sono stati scelti in modo da minimizzare i consumi, nel senso che se nella stanza non c'è nessuno è inutile tenere acceso il condizionatore o il riscaldamento, viceversa se c'è qualcuno nella stanza dobbiamo attivare il condizionatore o il riscaldamento per avere un temperatura media nella stanza piacevole.

Per verificare se c'è qualcuno utilizziamo la funzione ***checkPresence***() che restituisce un valore booleano, se c'è qualcuno settiamo delle variabili temporanee con i valori minimi e massimi del range.

Successivamente ci viene chiesto di effettuare l'aggiornamento dei quattro set point in maniera manuale attraverso monitor seriale. All' inizio dell' esecuzione del programma attraverso monitor seriale ci viene chiesto se vogliamo utilizzare come valori di range quelli di default, memorizzati da noi in fase di progetto, o se vuole inserire lui manualmente dei set point diversi. In qualsiasi momento dell ' esecuzione del programma è possibile passare da una configurazione all' altra inviando sempre attraverso monitor seriale la lettera corrispondente alla configurazione desiderata.

- A -> Valori di default
- B -> Valori settati manualmente

I nuovi valori vanno inviati tutti in un'unica riga separati da uno spazio. Prima i valori minimo e massimo in caso di stanza vuota, poi i valori minimo e massimo in caso di presenza della stanza per il condizionatore e successivamente sempre con lo stesso ordine quelli per il riscaldatore.

I valori ricevuti via monitor seriale sono memorizzati in un vettore e i valori salvati nel vettore sono assegnati alla struttura dati corrispondente attraverso la funzione ***setupTemp().***

Durante la fase di acquisizione dati blocchiamo temporaneamente con l'uso della variabile ***running*** l' esecuzione della parte di programma in cui si rileva la temperatura, analisi valore di temperatura ottenuta e messa in moto del motorino con la ventola o del riscaldatore.

Parte 7

```
void lcdDisplay() {

  switch (page_counter) {

    case 1: {    //Design of home page 1
        lcd.home();
        lcd.print("T:");
        lcd.print(temp);
        lcd.setCursor(8, 0);
        lcd.print("Pres:");
        if (isPresence == true) {
          lcd.print("YES");
        }
        else {
          lcd.print("NO");
        }

        lcd.setCursor(0, 1);
        lcd.print("AC:");
        lcd.print(fanLCD);
        lcd.print("%");
        lcd.setCursor(8, 1);
        lcd.print("HT:");
        lcd.print(ledLCD);
        lcd.print("%");

      }
      break;
```

```
    case 2: { //Design of page 2
        lcd.home();
        lcd.print("AC ");
        lcd.print("m:");
        lcd.print(acTmin);
        lcd.print(" M:");
        lcd.print(acTmax);
        lcd.setCursor(0, 1);
        lcd.print("HT ");
        lcd.print("m:");
        lcd.print(htTmin);
        lcd.print(" M:");
        lcd.print(htTmax);
      }
      break;
  }
```

In questo punto ci viene chiesto di mostrare sul display LCD tutte le informazioni. Siccome non ci stanno tutte in un'unica schermata creiamo più pagine, in questo caso due, e le alterniamo ogni cinque secondi. Per cambiare pagina utilizziamo un contatore ***page_counter*** che viene incrementato o decrementato ogni cinque secondi e attraverso un ***switch*** che confronta il valore presente nel contatore e il numero della pagina, pulisce lo schermo attraverso la chiamata della funzione ***lcd.clear(),*** riposiziona il cursore ad inizio schermo attraverso ***lcd.home()*** e poi attraverso delle ***lcd.print()*** stampa i valori desiderati.

Parte 9 (Bonus)

```
const unsigned int LED_PIN = 5;
const unsigned int NOISE_PIN = 7;

unsigned int currentClap = 0;
unsigned int previousClap = 0;

bool ledState = false;
bool justSwitched = false;

void setup() {
  pinMode(NOISE_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  while(!Serial);
  Serial.println("Connessione seriale avviata!");
}
void loop() {
    bonusNOISE();
}

void bonusNOISE(){
  if(digitalRead(NOISE_PIN) == LOW){
    if(currentClap == previousClap){
      previousClap = millis();
      return;
    }
    if(justSwitched){
      delay(600);
      justSwitched = false;
```

```
    }else{
      currentClap = millis();
      if(currentClap > previousClap + 200 && currentClap < previousClap + 600){
        ledState = !ledState;
        analoglWrite(LED_PIN, ledState*255);
        justSwitched = true;
      }
    }
  }
  previousClap = currentClap-1;
}
```

La sezione bonus richiedeva di implementare l'accensione di un led mediante il battito di mani, grazie all'utilizzo del sensore di rumore.

Per fare ciò viene realizzata la funzione *bonusNOISE* che, eseguita periodicamente nel loop rileva la presenza di un suono leggendo la tensione bassa sul pin di output del sensore. Durante la prima rilevazione i valori di `currentClap` e `previousClap` sono entrambi 0, pertanto il secondo viene inizializzato all'istante corrente. Durante la seconda rilevazione a `currentClap` verrà attribuito il nuovo istante, a questo punto si confronteranno i due valori: se essi differiscono per più di 200 ms ma meno di 600 ms (valori ottenuti empiricamente verificando la funzionalità) lo stato del led verrà invertito determinandone l'accensione o lo spegnimento. Inoltre viene abilitato il flag `justSwitched` che impedirà, mediante *delay*`(600)`, alla rilevazione successiva di variare nuovamente lo stato del led nel caso in cui avvenisse entro poco tempo dalla precedente accensione/spegnimento. Al termine della funzione il valore di `previousClap` viene impostato a `currentClap-1`, questa differenza non è significativa ma permette di non cadere nel primo if, utilizzato per l'inizializzazione all'avvio.