Relazione DMDAS (Gruppo 6)

Gianluca Blè, Stefano Di Lena, Giuseppe Manzo, Nicolas Nicoletti 01/04/2025

Introduzione

Lo scopo di questo elaborato è quello di gestire l'impianto di riscaldamento di una casa domotica, sfruttando dei moduli Wi-Fi (ESP32).

Abbiamo utilizzato quattro circuiti differenti, tre sono dedicati alla raccolta di dati, mentre il quarto gestisce il controllo degli attuatori.

Il primo circuito offre due modalità operative:

1. taratura del sensore di temperatura (DHT11) rispetto ad un riferimento (dallas 18b20, sensore di temperatura digitale);
2. trasmissione WiFi della temperatura misurata, dopo aver effettuato la taratura.

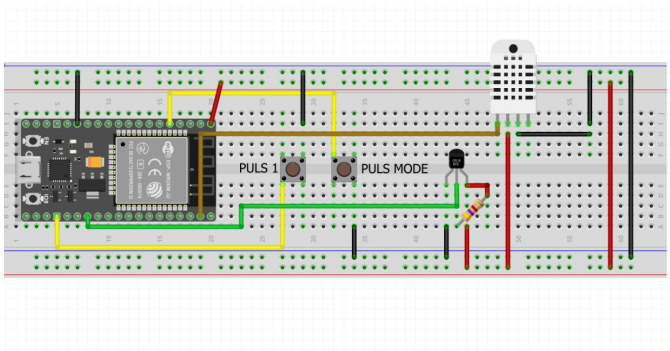


Figure 1: Primo circuito. Un bottone serve per cambiare tra le due modalità, un altro è usato per prendere la misura della temperatura in punti specifici, sono necessari almeno tre punti con temperature differenti per ottenere una buona stima.

Il secondo circuito è dotato di una fotoresistenza, la quale rileva le variazioni di luce ambientale per simulare condizioni diurne o notturne.

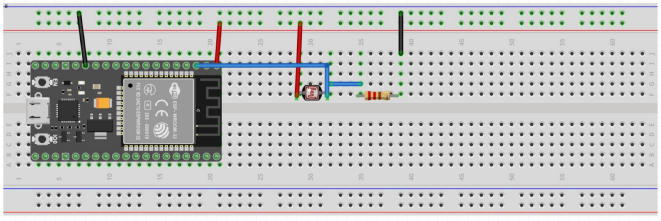


Figure 2: Secondo circuito.

Il terzo circuito fornisce ulteriori misurazioni di temperatura.

Infine, il quarto comprende due relè, uno simula l'attivazione del climatizzatore a pompa di calore e l'altro dei termosifoni. Per evidenziare l'attivazione o meno di questi, li abbiamo sostituiti a scopo didattico con dei diodi led.

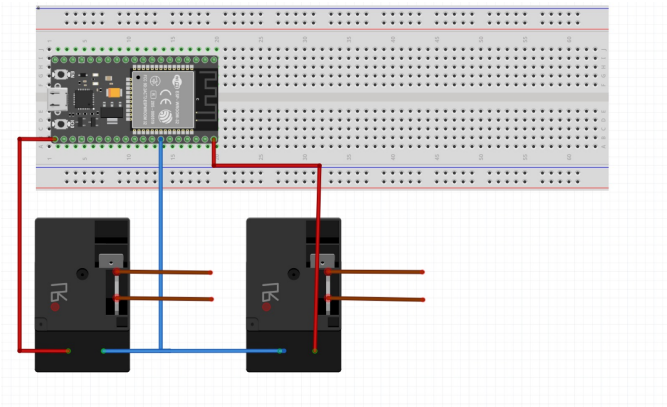


Figure 3: Quarto circuito. Il relè a sinistra simula l'attivazione dei termosifoni, quello a destra il climatizzatore.

Materiale Impiegato

ESP32

Modulo WiFi e Bluetooth a 2.4 GHz, il quale integra un set di periferiche che quali possono essere: I2C, I2S, UART, Ethernet, SPI ad alta velocità.

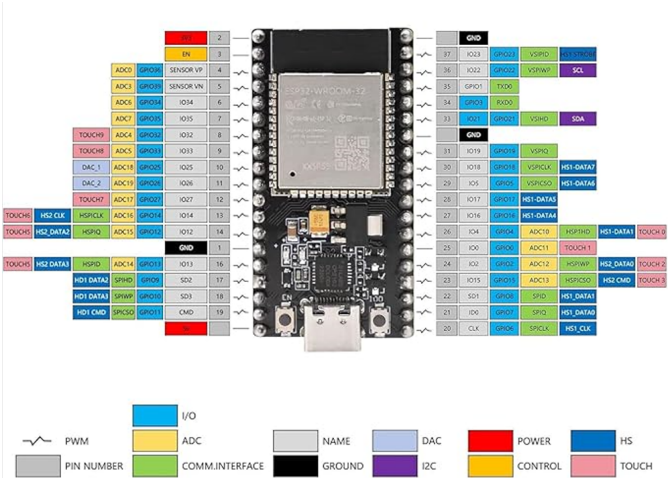


Figure 4: ESP32 Module Pins.

Symbol	Parameter	Min	Typical	Max	Unit
$VDD33$	Power supply voltage	3.0	3.3	3.6	V
I_{VDD}	Current delivered by external power supply	0.5	-	-	A
T	Operating ambient temperature	-40	-	85	°C

Table 1: Condizioni operative raccomandate.

DS18B20

É un sensore di temperatura digitale, in grado di misurare temperature che variano da -55°C a +125°C. La sua risoluzione è configurabile dall'utente a 9, 10, 11 o 12 bits, che corrisponde rispettivamente ad incrementi di 0.5°C, 0.25°C, 0.1245°C e 0.0625°C.

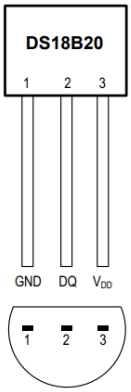


Figure 5: DS18B20 Pin Configuration.

La tensione di alimentazione del DS18B20 deve essere minimo 3.0V e massimo 5.5V.

Il DS18B20 comunica attraverso un bus 1-Wire, che per definizione richiede una sola linea dati per la comunicazione. Questo tipo di comunicazione è costituita da un unico master e da uno o più slave (il DS18B20 si comporta da slave).

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	**BIT 14**	**BIT 13**	**BIT 12**	**BIT 11**	**BIT 10**	**BIT 9**	**BIT 8**
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Figure 6: Temperature Register Format.

I dati di temperatura sono memorizzati in formato complemento a due esteso su 16 bit. Per temperature positive S = 0, per quelle negative S = 1.

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

Figure 7: Relazione Temperatura/Dati

Il DS18B20 campiona la temperatura e la converte in una rappresentazione digitale memorizzata in una memoria interna (scratchpad). Il tempo di conversione varia in base alla risoluzione impostata.

DHT11

É un sensore di temperatura ed umidità digitale. La sua alimentazione può variare tra 3V e 5.5V (DC). Al suo interno, il DHT11 integra un termistore per la misurazione della temperatura e un elemento capacitivo per rilevare l'umidità relativa.

Il formato di comunicazione utilizzato per la trasmissione e sincronizzazione dei dati con l'MCU è single-bus.

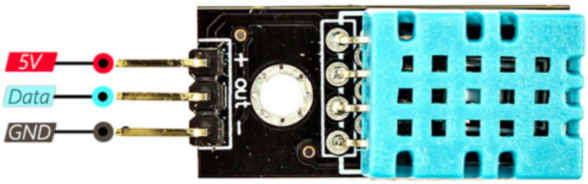


Figure 8: DHT11 Pin Configuration.

Quando il MCU invia un segnale di avvio, il DHT11 passa dalla modalità basso consumo energetico alla modalità operativa, in attesa che il microcontrollore completi l'invio del segnale.

A questo punto, il sensore invia una risposta contenente 40 bit di dati, che includono le informazioni relative a umidità relativa e temperatura.

Dopo che i dati sono stati raccolti, il DHT11 torna in modalità basso consumo energetico finché non riceve un nuovo segnale di avvio dal MCU.

LM35

É un sensore di temperatura analogico, progettato per funzionare in un intervallo di temperatura compreso tra -55°C e 150°C. É possibile alimentarlo in un range di tensione che varia da 4V a 30V.

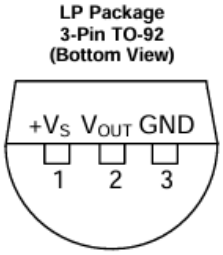
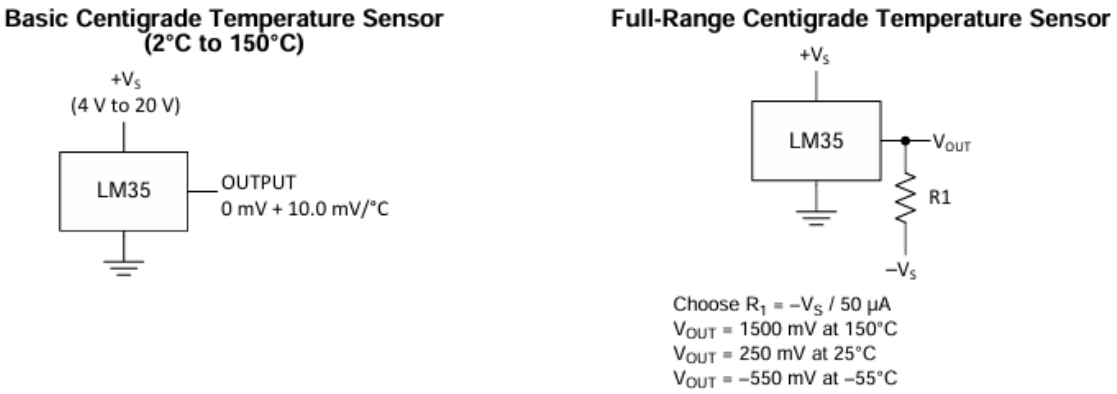


Figure 9: LM35 Pin Configuration.

La tensione in uscita dall'LM35 è direttamente proporzionale alla temperatura in gradi Celsius, il sensore fornisce 10mV per ogni grado (ad esempio, a 25°C l'uscita sarà 250mV).

Per implementare la lettura in termini di temperatura è necessario utilizzare un ADC (convertitore analogico-digitale), perché il segnale prodotto dal sensore è analogico.



Fotoresistore

É una resistenza realizzata con materiale semiconduttore, la cui conduttanza varia in base alla quantità di luce ricevuta.

In condizioni di luminosità bassa, la resistenza del componente è elevata. Con l'aumentare della luce la resistenza diminuisce, permettendo il passaggio di una maggiore corrente.

Per convertire la variazione di resistenza in un segnale utilizzabile, la fotoresistenza viene impiegata in un partitore di tensione. In questo modo, si ottiene un'uscita di tensione che varia in funzione dell'intensità luminosa.

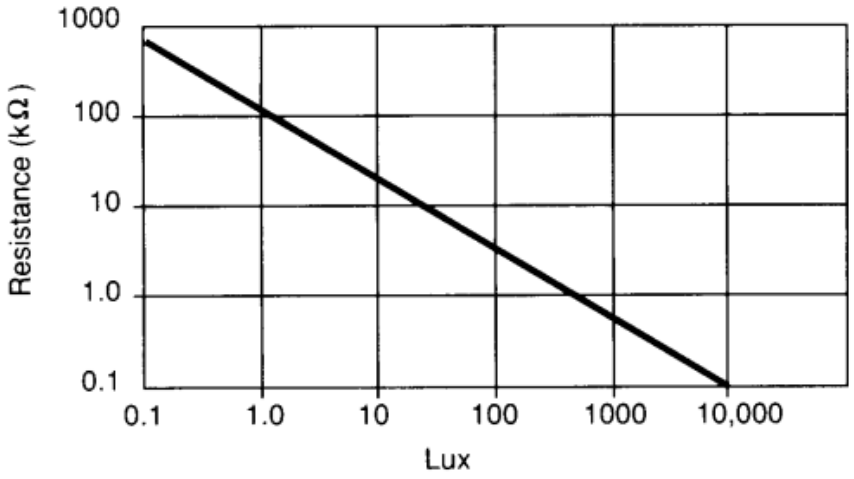


Figure 10: Resistenza in funzione dell'illuminazione.

Calibrazione dei Sensori

La calibrazione è un processo di correzioni degli errori di misura. Avviene in due fasi:

1. confrontiamo un sensore con uno strumento di riferimento (nel nostro caso il dallas 18b20) per determinare l'errore tra il valore vero ed il valore misurato;
2. usiamo questa informazione per correggere le future misure effettuate dal sensore.

Nel nostro caso:

- X è la temperatura reale della stanza;
- Y è il valore misurato dal DHT11;
- $\hat{X}$ è la stima corretta della temperatura (ottenuta dopo la calibrazione).

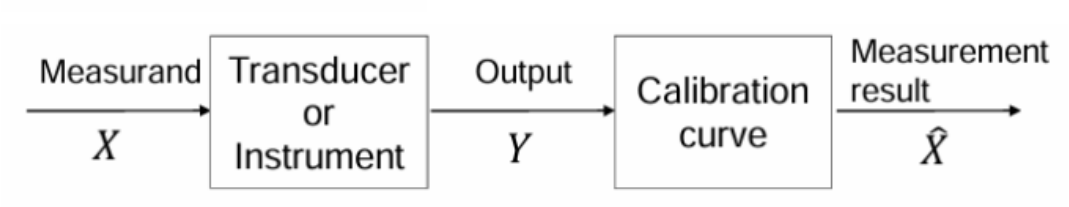


Figure 11: Calibrazione

Per effettuare la calibrazione dobbiamo prelevare almeno N campioni di misura a temperature differenti (nel nostro caso $N = 3$). Otteniamo N punti (x, y) che permettono di identificare un modello matematico del trasduttore:

$$Y = g(X) + E,$$

dove E rappresenta un errore a media nulla.

L'equazione può anche essere riscritta come:

$$Y = \beta_0 + \beta_1 + E.$$

Per la calibrazione classica (regressione di y su x) basata sul Least Square method, dobbiamo stimare i coefficienti β_0 e β_1.

Le stime sono ottenute minimizzando la Residual Sum of Squares:

$$RSS = \sum_{i=1}^{N} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_n)^2.$$

Ottenendo, quindi, le soluzioni:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2},$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

dove $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ e $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ sono le *sample means*.

Nel nostro caso dobbiamo eseguire una **stima inversa**:

$$\hat{x}_0 = \frac{y_0 - \hat{\beta}_0}{\hat{\beta}_1}.$$

Potrebbero verificarsi delle incertezze nella stima dopo la calibrazione, dovute ad errori di incertezza in input e/o output durante le fasi di misura.

LabVIEW

Per garantire un'efficace comunicazione tra i moduli ESP32 e il software LabVIEW, abbiamo utilizzato un'architettura basata su un router Wi-Fi (in questo caso, l'hotspot del telefono) per connettere i dispositivi. Gli ESP32 funzionano come server, rimanendo costantemente in attesa di messaggi da parte di LabVIEW, che agisce come client.

La comunicazione tra LabVIEW e i moduli ESP32 avviene tramite il protocollo UDP (User Datagram Protocol). Questa scelta è stata dettata dalla necessità di un sistema di trasmissione veloce e senza handshake, poiché UDP non richiede un meccanismo di acknowledgment (ACK), riducendo così la latenza di trasmissione dei pacchetti.

Tuttavia, affinché i dispositivi possano comunicare, è necessario conoscere il *socket* di destinazione, ovvero la coppia (porta, indirizzo IP del ricevitore). Poiché gli indirizzi IP degli ESP32 possono variare a ogni connessione, il sistema implementa una procedura di discovery automatica tramite messaggi broadcast.

Block Diagram

Figure 12: Block Diagram View

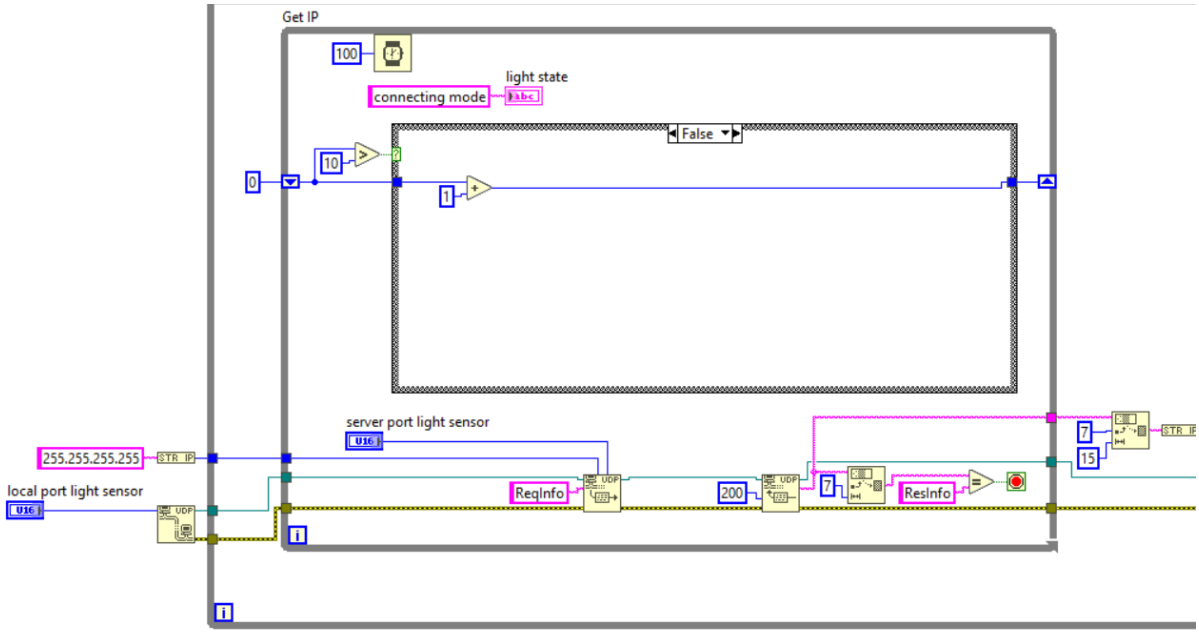
Il sistema è progettato utilizzando quattro cicli While, ognuno dedicato alla comunicazione con un singolo ESP32. Questa suddivisione garantisce che, in caso di disconnessione di uno dei moduli, il programma possa tentare automaticamente la riconnessione senza dover riavviare l'intero sistema.

Connessione

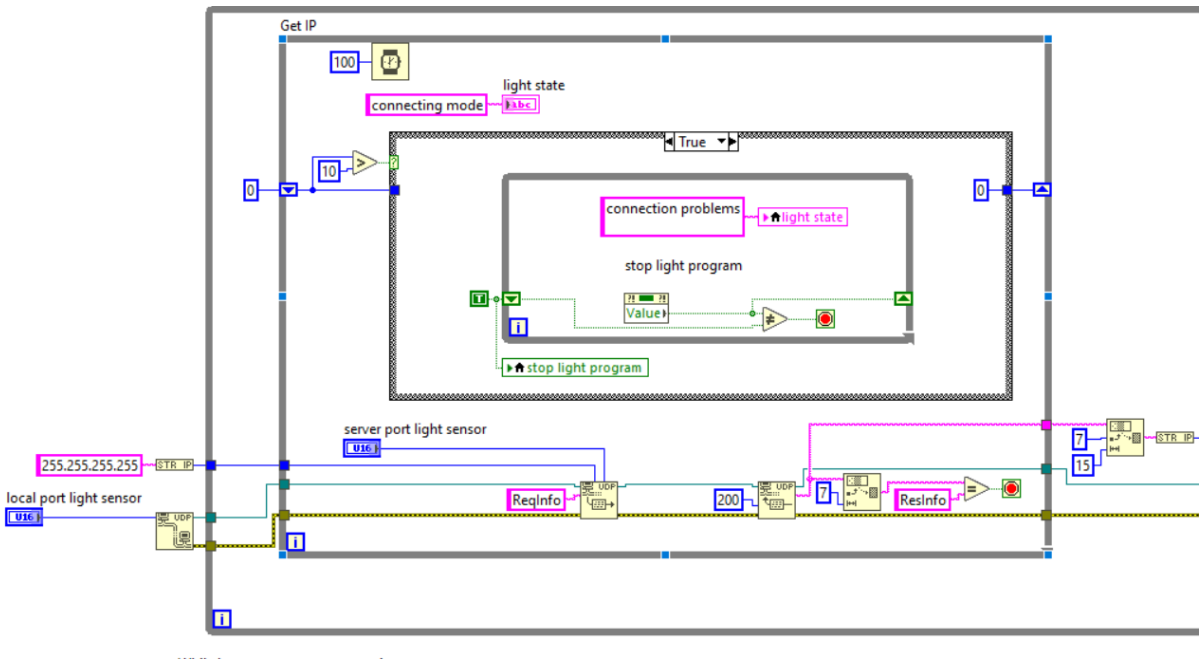
La fase di connessione iniziale è identica per tutti gli ESP32.

All'avvio, LabVIEW utilizza il blocco OPEN, che consente di assegnare un numero di porta alla rete locale del PC, e invia una richiesta di connessione trasmessa a tutti i dispositivi sulla rete attraverso l'indirizzo 255.255.255.255. Questa richiesta viene ricevuta da tutti gli ESP32, ma solo quello configurato per rispondere alla specifica porta di comunicazione invierà un messaggio di conferma contenente il proprio indirizzo IP.

A questo punto, LabVIEW attende la risposta del modulo ESP32 utilizzando il blocco READ, impostato su un timeout di 200 ms. Se il messaggio ricevuto contiene l'informazione attesa, la connessione viene stabilita correttamente e il sistema può proseguire con la trasmissione dei dati. Se invece il tempo massimo di attesa scade senza ricevere alcuna risposta, il programma rileva un errore di connessione e ripete il processo di ricerca dell'indirizzo IP, incrementando un contatore interno per tenere traccia dei tentativi.



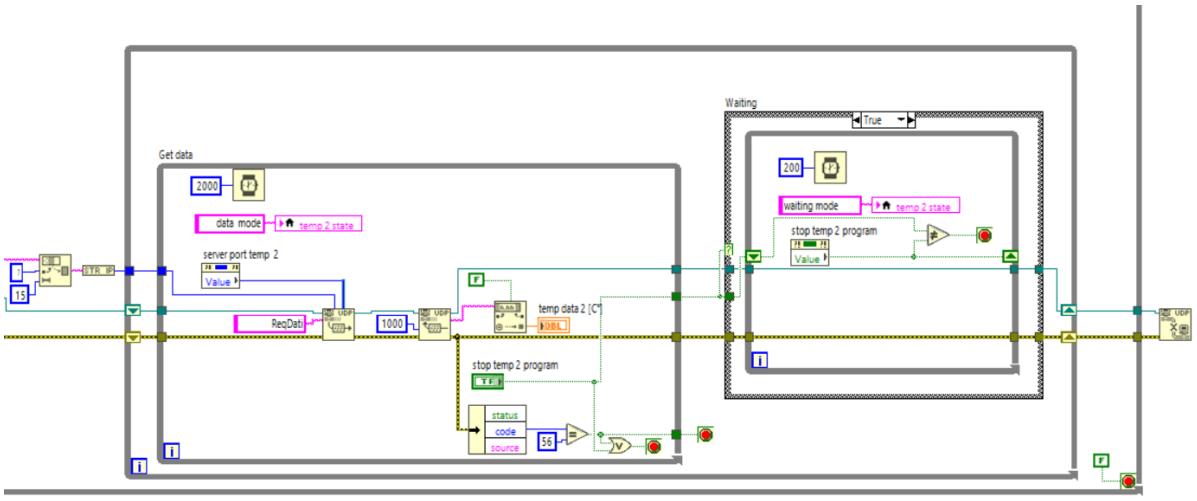
Se, dopo 10 tentativi consecutivi, la connessione non viene stabilita, il sistema entra in una modalità di errore in cui interrompe automaticamente i tentativi di riconnessione e attiva un pulsante di stop. A questo punto, l'utente può decidere di riavviare il tentativo di connessione premendo il pulsante, consentendo così al programma di riprendere il processo di discovery e tentare nuovamente il collegamento con gli ESP32.



Dati di Temperatura

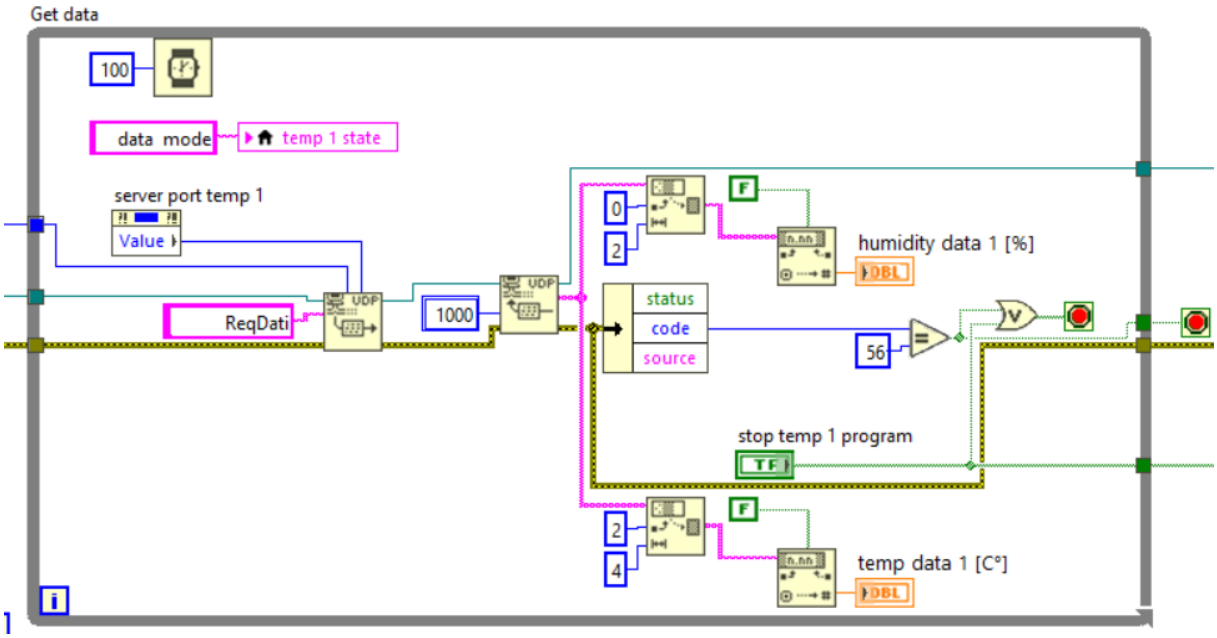
Inviamo periodicamente un messaggio di richiesta dati al server dell'ESP32 (ReqData), al quale questo risponde inviando il valore della temperatura. Parallelamente è implementato un meccanismo di controllo della connessione, per garantire la stabilità del sistema. Se viene rilevato un errore di comunicazione (codice di errore 56), il sistema avvisa l'utente, il quale può avviare un'operazione di riconnessione.

in un secondo ciclo while interno, è implementata una modalità di attesa che permette di mettere in pausa l'acquisizione dati se necessario.

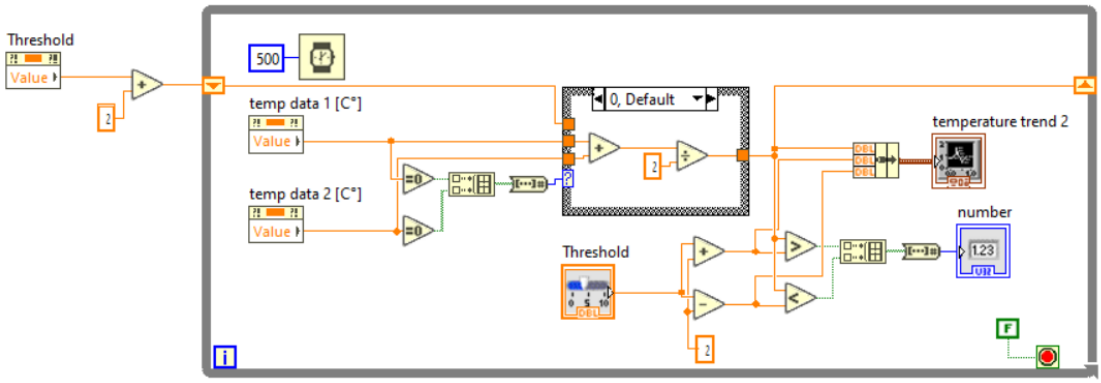


Questa stessa logica di acquisizione è implementata anche per il secondo sensore di temperatura, che, essendo il DHT11, in aggiunta fornisce anche la misura dell'umidità relativa.

Il principio di funzionamento rimane invariato: il programma invia una richiesta dati all'ESP32 tramite UDP, il quale risponde con i valori di temperatura e umidità. Una volta ricevuti i dati, questi vengono elaborati e aggiornati nelle rispettive variabili, consentendo la loro visualizzazione in tempo reale.

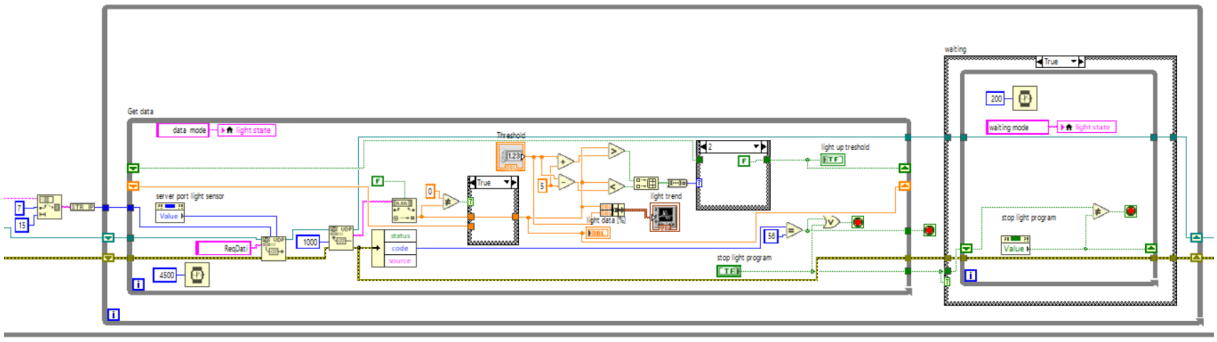


In un ciclo while a parte, la media delle due temperature misurate dai due sensori viene calcolata e rappresenta graficamente. Questo permette di ottenere una stima della temperatura complessiva.



Dati di Luminosità

In questa sezione del programma LabVIEW è gestita l'acquisizione dei dati da un sensore di luce e l'analisi del valore ricevuto rispetto a una soglia predefinita.

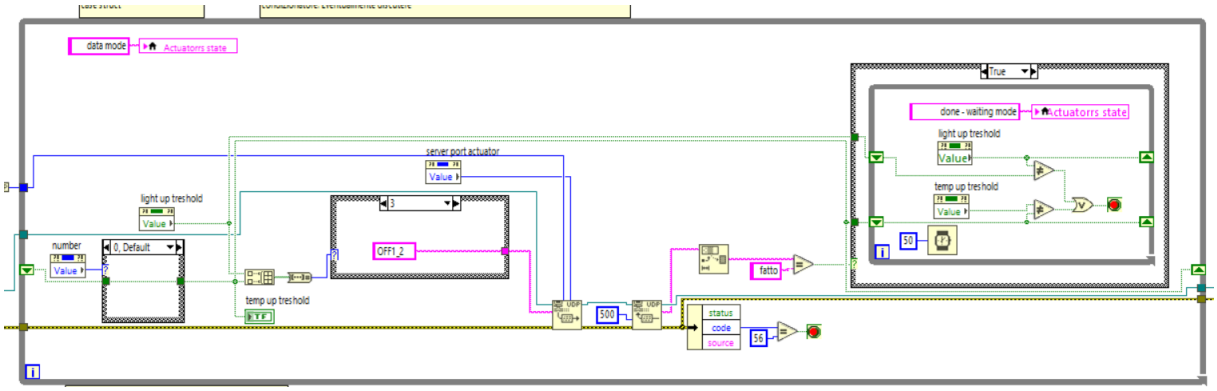


Una volta ottenuto il dato dal sensore, questo è confrontato con una soglia prestabilita per determinare se si è in condizioni di luce o buio. Questo confronto viene eseguito tramite un case structure.

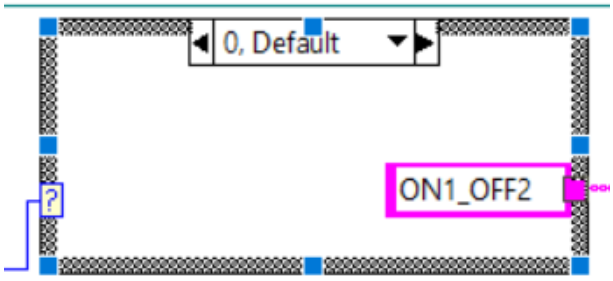
Anche in questo caso è presente un ciclo di attesa che consente di mettere il sistema in uno stato di standby qualora si verifichi un errore di connessione, impedendo tentativi di lettura ripetuti fino a quando non venga premuto il pulsante di ripristino.

Azionamento Attuatori

Il controllo degli attuatori (termosifoni e climatizzatore) è gestito in base alle condizioni ambientali rilevate, ovvero la temperatura e la luminosità.

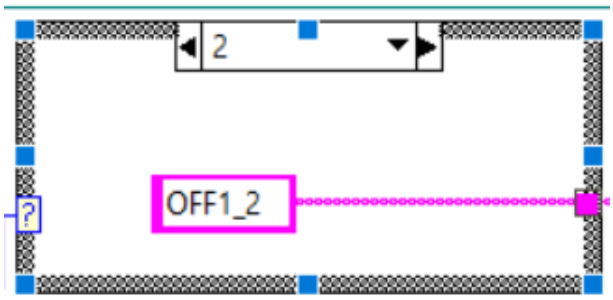


Il sistema riceve i valori di soglia per la luce (light up threshold) e per la temperatura (temp up threshold) e, in base a questi parametri, prende decisioni sullo stato degli attuatori.



Quando il livello di luce indica che è notte e la temperatura è inferiore alla soglia desiderata, allora il sistema attiva i termosifoni per riscaldare l'ambiente.

Se invece è giorno e la temperatura è bassa, il sistema attiva il climatizzatore.



Nel caso in cui la temperatura sia già alla soglia desiderata, sia di giorno che di notte, il sistema non attiva nessun attuatore e mantiene tutto spento per risparmiare energia.

Front Panel

Abbiamo un'interfaccia interattiva per monitorare e controllare il sistema in tempo reale.

É possibile modificare direttamente i valori di soglia per la temperatura e la luminosità attraverso due controlli dedicati. Questo permette di adattare il funzionamento degli attuatori senza dover intervenire sul codice, rendendo il sistema più flessibile.

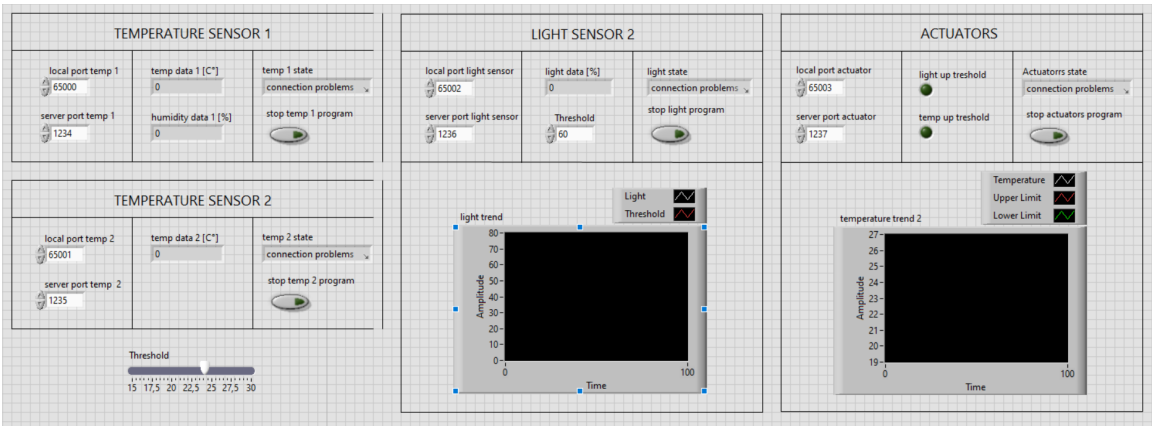


Figure 13: Front Panel View

Grazie a questa interfaccia, l'utente può gestire il sistema in modo intuitivo e immediato, avendo sempre sotto controllo i parametri principali e la possibilità di modificarli in tempo reale.

Arduino IDE

Codice primo circuito

```
#include <WiFi.h>
#include <WiFiUdp.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include "DHT.h"


#define DHTPIN 23      // Pin digitale a cui è collegato il sensore
#define DHTTYPE DHT11  // Tipo di sensore: DHT11 o DHT22
#define PULMOD 12
#define BOTTONE 15
#define MAX_VALORI 3


bool buttonPressed = false;          // Indica se il BOTTONE è attualmente premuto
unsigned long buttonPressStart = 0;  // Quando abbiamo premuto il pulsante
const unsigned long PRESS_THRESHOLD = 3000; // 3 secondi per il reset
unsigned long debounceTime = 0;      // Per evitare rimbalzi
const unsigned long DEBOUNCE_DELAY = 200;
bool connesso = 0;
const int ONE_WIRE_BUS = 4;  // provare a passarlo a define
const char* ssid = "Galaxy 22";         // Nome della rete WiFi
const char* password = "giuseppe";     // Password WiFi
const int udpPort = 1234; // Porta UDP su cui il server ascolta
char packetBuffer[255];   // Buffer per memorizzare i dati ricevuti
unsigned long previousMillis = 0;
float valdallas[MAX_VALORI] = { 0 };  // Vettore per salvare le temperature
float valldht11[MAX_VALORI] = { 0 };
int indice = 0;  // Indice per il vettore
bool stato = 0;
bool lastState = 1;

WiFiUDP udp;
DHT dht(DHTPIN, DHTTYPE);
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensore18(&oneWire);



void setup() {
    Serial.begin(115200);
    delay(1000);

    pinMode(PULMOD,INPUT_PULLUP); // PIN DA MODIFICARE PRIMA DI ACCENDERE ESP
    BOTTONE PER LA MODALITA TARATORE O SERVER
    sensore18.begin();
    Serial.println("Inizializzazione DS18B20 completata");
```

```
47      pinMode(BOTTONE, INPUT_PULLUP);
        pinMode(2, OUTPUT);     // led interno all'esp
49      dht.begin();
}
51


53
void loop() {
55
bool tarato = false;
57   float humidity = dht.readHumidity();
     float temperature = dht.readTemperature();
59   sensore18.requestTemperatures();
     float tempC = sensore18.getTempCByIndex(0);
61   float risultaticoeff[2];
     float valoretarato;
63   float media[10];

65   bool currentState = digitalRead(PULMOD);
     if (lastState == 1 && currentState == 0) {
67     stato = !stato;
       delay(50);
69   }
     lastState = currentState;
71
     // Siamo in modalità taratore
73   if (stato == 0) {
       Serial.println("Mod Taratore");
75
       Serial.print("Temperatura attuale (dallas18): ");
77     Serial.print(tempC);
       Serial.print(" °C     ");
79     Serial.print("Temperatura attuale (dht11): ");
       Serial.print(temperature);
81     Serial.println(" °C");

83     bool bottone = digitalRead(BOTTONE);

85     if (bottone == HIGH && !buttonPressed && (millis() - debounceTime >
       DEBOUNCE_DELAY)) {
         buttonPressed = true;
87       buttonPressStart = millis();
         debounceTime = millis();
89     }
       if (bottone == LOW && buttonPressed && (millis() - debounceTime > DEBOUNCE_DELAY
       )) {
91       buttonPressed = false;
         debounceTime = millis();
93
         unsigned long pressDuration = millis() - buttonPressStart;
95
```

```
    if (pressDuration >= PRESS_THRESHOLD) {
      // ---- RESET ----
      Serial.println("Reset dei valori...");
      for (int i = 0; i < MAX_VALORI; i++) {
        valdallas[i] = 0;
        valldht11[i] = 0;
      }
      indice = 0;

      lampeggia();
      lampeggia();
      lampeggia();
    }else{
      if (indice < MAX_VALORI) {
        valdallas[indice] = tempC;
        valldht11[indice] = temperature;
        indice++;
        Serial.println("Valore salvato!");
        lampeggia();
      }else{
        Serial.println("Memoria piena, impossibile salvare altri valori.");
        lampeggia();
        lampeggia();
      }
    }
  }

  Serial.print("Valori salvati dallas18: ");
  for (int i = 0; i < MAX_VALORI; i++) {
    Serial.print(valdallas[i]);
    Serial.print(" ");
  }
  Serial.println();

  Serial.print("Valori salvati dht11: ");
  for (int i = 0; i < MAX_VALORI; i++) {
    Serial.print(valldht11[i]);
    Serial.print(" ");
  }
  Serial.println();

  // Calcolo stima solo se abbiamo riempito il vettore
  if (indice == MAX_VALORI) {
    stimacoeff(valdallas, valldht11, risultaticoeff);
    if (risultaticoeff[1] != 0) {
      valoretarato = (temperature - risultaticoeff[0]) / risultaticoeff[1];
      Serial.print("Valore STIMATO DHT11 : ");
      Serial.println(valoretarato);
      tarato = true;
    }else{
      Serial.println("Errore: b1 = 0, impossibile calcolare valore tarato.");
```

```
    }
  }
}else{
Serial.println("MODALITA SERVER:");

if(connesso == false){

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
      delay(1000);
      Serial.println("Connessione WiFi...");
  }

  Serial.println("Connesso al WiFi!");
  Serial.print("IP del server: ");
  Serial.println(WiFi.localIP());
  udp.begin(udpPort); // Avvia il server UDP sulla porta specificata
  Serial.println("Server UDP in ascolto...");
  connesso = true;
}else if(WiFi.status() != WL_CONNECTED){
  Serial.println("Connessione persa! Tentativo di riconnessione...");
  connesso = false;
}else{
 unsigned long currentMillis = millis();
 if (currentMillis - previousMillis >= 1000) { // Ogni secondo
    previousMillis = currentMillis;
  }
int packetSize = udp.parsePacket(); // Controlla se ci sono dati UDP ricevuti
if (packetSize) {
  Serial.print("Ricevuto pacchetto di ");
  Serial.print(packetSize);
  Serial.println(" byte");

  // Legge il pacchetto ricevuto
  int len = udp.read(packetBuffer, 255);
  if (len > 0) {
     packetBuffer[len] = '\0'; // Termina la stringa ricevuta
      }

   Serial.print("Messaggio ricevuto: ");
   Serial.println(packetBuffer);

   String richiesta = String(packetBuffer);

   Serial.println(richiesta);

   if (richiesta.indexOf("ReqDati") != -1) {
       // Invia la risposta al client
       float somma = 0;
       udp.beginPacket(udp.remoteIP(), udp.remotePort());
```

```
197         //aggiungere controllo se è stato tarato allmeno una volta fai cosi
    altrimenti mandagli il valore del dallas18
          Serial.print("Temperatura tarata media del DHT11: ");
199       for(int i =0;i<10;i++){
           valoretarato = (temperature - risultaticoeff[0]) / risultaticoeff[1];
201        media[i]= valoretarato;
           delay(10);
203       }
          for(int i =0;i<10;i++){
205        somma = media[i] + somma;
          }
207       Serial.print(somma/10);
          Serial.print(" °C  ");
209       Serial.print("Umidita : " );
          Serial.println(humidity);
211       Serial.print("Stringa inviata: ");
          String output = String(humidity) + String(somma/10, 2);
213       Serial.println(output);
          delay(random(0, 50));  // Aspetta tra 100 e 500 ms prima di trasmettere
215       udp.print(output); //se non va rimettere somma/10
          udp.endPacket();
217      }else if(richiesta.indexOf("Info") != -1 ){
           // Invia la risposta al client
219          String res = "ResInfo" + WiFi.localIP().toString();
             udp.beginPacket(udp.remoteIP(), udp.remotePort());
221          Serial.println(res);
             udp.print(res);
223          udp.endPacket();
        }else{
225       String mes = "non va";
             udp.beginPacket(udp.remoteIP(), udp.remotePort());
227          udp.print(mes);
             udp.endPacket();
229     }
        Serial.println("Risposta inviata!");
231   }
  }
233 }
}
235


237

void lampeggia() {
239 digitalWrite(2, HIGH);
  delay(75);
241 digitalWrite(2, LOW);
  delay(25);
243 }


245
```

```
void stimacoeff(float *puntdallas18, float *puntlm35, float *risultati) {
  float mediay = 0, mediax = 0;
  float sommaNum = 0, sommaDen = 0;
  float b1 = 0, b0 = 0;

  // Calcolo delle medie
  for (int i = 0; i < MAX_VALORI; i++) {
    mediay += puntlm35[i];
    mediax += puntdallas18[i];
  }
  mediay /= MAX_VALORI;
  mediax /= MAX_VALORI;

  // Calcolo del numeratore e denominatore
  for (int i = 0; i < MAX_VALORI; i++) {
    float diffx = puntdallas18[i] - mediax;
    float diffy = puntlm35[i] - mediay;
    sommaNum += diffx * diffy;    // Numeratore
    sommaDen += diffx * diffx;    // Denominatore
  }

  if (sommaDen != 0) {
    b1 = sommaNum / sommaDen;  // Coefficiente angolare
    b0 = mediay - b1 * mediax; // Intercetta
  } else {
    Serial.println("Errore: divisione per zero.");
    b1 = 0;
    b0 = mediay;
  }

  risultati[0] = b0;
  risultati[1] = b1;

  Serial.print("b0: ");
  Serial.println(b0);
  Serial.print("b1: ");
  Serial.println(b1);
}
```

project/Calibratore.ino

Codice secondo circuito

```
#include <WiFi.h>
#include <WiFiUdp.h>

const char* ssid = "Galaxy 22";         // Nome della rete WiFi
const char* password = "giuseppe"; // Password WiFi

WiFiUDP udp;
```

```
const int udpPort = 1236; // Porta UDP su cui il server ascolta
char packetBuffer[255];   // Buffer per memorizzare i dati ricevuti

unsigned long previousMillis = 0;
bool connesso = 0;
float luminosita = 0;
float  somma = 0;
float percentuale;


void setup() {
    Serial.begin(115200);
    delay(1000);
}


void loop() {

   if (connesso == 0){
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connessione WiFi...");
    }

    Serial.println("Connesso al WiFi!");
    Serial.print("IP del server: ");
    Serial.println(WiFi.localIP());
    udp.begin(udpPort); // Avvia il server UDP sulla porta specificata
    Serial.println("Server UDP in ascolto...");
    connesso = true;
   }else if(WiFi.status() != WL_CONNECTED){
    Serial.println("Connessione persa! Tentativo di riconnessione...");
    connesso = false;
  }else{
  //incremento tramte timer ogni secondo
   unsigned long currentMillis = millis();
   if (currentMillis - previousMillis >= 1000) { // Ogni secondo
    previousMillis = currentMillis;
    }
    int packetSize = udp.parsePacket(); // Controlla se ci sono dati UDP ricevuti
    if (packetSize) {
        Serial.print("Ricevuto pacchetto di ");
        Serial.print(packetSize);
        Serial.println(" byte");

        // Legge il pacchetto ricevuto
        int len = udp.read(packetBuffer, 255);
        if (len > 0) {
            packetBuffer[len] = '\0'; // Termina la stringa ricevuta
```

```
        }
60      Serial.print("Messaggio ricevuto: ");
        Serial.println(packetBuffer);
62
        // Determina la risposta in base alla richiesta ricevuta
64      String richiesta = String(packetBuffer);

66      Serial.println(richiesta);

68      if (richiesta.indexOf("ReqDati") != -1) {
            // Invia la risposta al client
70          somma = 0;
             udp.beginPacket(udp.remoteIP(), udp.remotePort());
72           for(int i = 0;i <10;i++){
              somma += analogRead(36);
74            delay(20);
             }
76
             luminosita = somma/10.0;
78
             percentuale = (luminosita/4095.0)*100;
80
             Serial.println(percentuale);
82           udp.print(percentuale);
             udp.endPacket();
84      }else if(richiesta.indexOf("Info") != -1 ){
          // Invia la risposta al client
86          String res = "ResInfo" + WiFi.localIP().toString();
            udp.beginPacket(udp.remoteIP(), udp.remotePort());
88          Serial.println(res);
            udp.print(res);
90          udp.endPacket();
        }else{
92        String mes = "non va";
          udp.beginPacket(udp.remoteIP(), udp.remotePort());
94          udp.print(mes);
            udp.endPacket();
96      }
        Serial.println("Risposta inviata!");
98    }
    }
100 }
```

project/Luce.ino

Codice terzo circuito

```
#include <WiFi.h>
2 #include <WiFiUdp.h>
#include <OneWire.h>
```

```
#include <DallasTemperature.h>


const unsigned long DEBOUNCE_DELAY = 200;
const int ONE_WIRE_BUS = 4;  // provare a passarlo a define
const char* ssid = "Galaxy 22";         // Nome della rete WiFi
const char* password = "giuseppe"; // Password WiFi
const int udpPort = 1235; // Porta UDP su cui il server ascolta
char packetBuffer[255];   // Buffer per memorizzare i dati ricevuti
unsigned long previousMillis = 0;
bool connesso = false;

WiFiUDP udp;
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensore18(&oneWire);



void setup() {
    Serial.begin(115200);
    sensore18.begin();
    Serial.println("Inizializzazione DS18B20 completata");
}



void loop() {

  if(connesso == false){

    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connessione WiFi...");
    }

    Serial.println("Connesso al WiFi!");
    Serial.print("IP del server: ");
    Serial.println(WiFi.localIP());
    udp.begin(udpPort); // Avvia il server UDP sulla porta specificata
    Serial.println("Server UDP in ascolto...");
    connesso = true;
  }else if(WiFi.status() != WL_CONNECTED){
    Serial.println("Connessione persa! Tentativo di riconnessione...");
    connesso = 0;
  }else{
   unsigned long currentMillis = millis();
   if (currentMillis - previousMillis >= 1000) { // Ogni secondo
      previousMillis = currentMillis;
    }
  int packetSize = udp.parsePacket(); // Controlla se ci sono dati UDP ricevuti
```

```
  if (packetSize) {
    Serial.print("Ricevuto pacchetto di ");
    Serial.print(packetSize);
    Serial.println(" byte");

    // Legge il pacchetto ricevuto
    int len = udp.read(packetBuffer, 255);
    if (len > 0) {
       packetBuffer[len] = '\0'; // Termina la stringa ricevuta
        }

     Serial.print("Messaggio ricevuto: ");
     Serial.println(packetBuffer);

    // Determina la risposta in base alla richiesta ricevuta
     String richiesta = String(packetBuffer);

     Serial.println(richiesta);

     if (richiesta.indexOf("ReqDati") != -1) {
         // Invia la risposta al client
         udp.beginPacket(udp.remoteIP(), udp.remotePort());
         Serial.print("Temperatura DS18: ");
         sensore18.requestTemperatures();
         float tempC = sensore18.getTempCByIndex(0);
         Serial.print(tempC);
         Serial.println(" °C  ");
         udp.print(tempC);
         udp.endPacket();
        }else if(richiesta.indexOf("Info") != -1 ){
          // Invia la risposta al client
            String res = "ResInfo" + WiFi.localIP().toString();
            udp.beginPacket(udp.remoteIP(), udp.remotePort());
            Serial.println(res);
            udp.print(res);
            udp.endPacket();
        }else{
          String mes = "non va";
            udp.beginPacket(udp.remoteIP(), udp.remotePort());
            udp.print(mes);
            udp.endPacket();
        }
        Serial.println("Risposta inviata!");
    }
  }
}
```

project/Temperatura.ino

Codice quarto circuito

```
#include <WiFi.h>
#include <WiFiUdp.h>
#define caldaia 23
#define condizionatore 22

const char* ssid = "Galaxy 22";         // Nome della rete WiFi
const char* password = "giuseppe"; // Password WiFi
const int udpPort = 1237; // Porta UDP su cui il server ascolta
char packetBuffer[255];   // Buffer per memorizzare i dati ricevuti
unsigned long previousMillis = 0;
bool connesso = false;

WiFiUDP udp;



void setup() {
    Serial.begin(115200);
    pinMode(caldaia,OUTPUT);
    pinMode(condizionatore,OUTPUT);
}



void loop() {

  if(connesso == false){

    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connessione WiFi...");
    }

    Serial.println("Connesso al WiFi!");
    Serial.print("IP del server: ");
    Serial.println(WiFi.localIP());
    udp.begin(udpPort); // Avvia il server UDP sulla porta specificata
    Serial.println("Server UDP in ascolto...");
    connesso = true;
  }else if(WiFi.status() != WL_CONNECTED){
    Serial.println("Connessione persa! Tentativo di riconnessione...");
    connesso = false;
  }else{
   unsigned long currentMillis = millis();
   if (currentMillis - previousMillis >= 1000) { // Ogni secondo
      previousMillis = currentMillis;
    }
  int packetSize = udp.parsePacket(); // Controlla se ci sono dati UDP ricevuti
```

```
50  if (packetSize) {
      Serial.print("Ricevuto pacchetto di ");
52    Serial.print(packetSize);
      Serial.println(" byte");
54
      // Legge il pacchetto ricevuto
56    int len = udp.read(packetBuffer, 255);
      if (len > 0) {
58       packetBuffer[len] = '\0'; // Termina la stringa ricevuta
          }
60     Serial.print("Messaggio ricevuto: ");
       Serial.println(packetBuffer);
62
      // Determina la risposta in base alla richiesta ricevuta
64     String richiesta = String(packetBuffer);

66     Serial.println(richiesta);

68     if (richiesta.indexOf("OFF1_2") != -1) {
           // Invia la risposta al client
70         udp.beginPacket(udp.remoteIP(), udp.remotePort());
           Serial.print("CALDAIA SPENTA, CONDIZIONATORE SPENTO");
72         digitalWrite(condizionatore,LOW);
           digitalWrite(caldaia,LOW);
74         udp.print("fatto");
           udp.endPacket();
76       }else if(richiesta.indexOf("ON1_OFF2") != -1 ){
           // Invia la risposta al client
78           udp.beginPacket(udp.remoteIP(), udp.remotePort());
             Serial.println("CALDAIA ACCESA, CONDIZIONATORE SPENTO");
80           udp.print("fatto");
             digitalWrite(condizionatore,LOW);
82           digitalWrite(caldaia,1);

84           udp.endPacket();
         }else if(richiesta.indexOf("OFF1_ON2") != -1 ){
86         // Invia la risposta al client
             udp.beginPacket(udp.remoteIP(), udp.remotePort());
88           Serial.println("CALDAIA SPENTA, CONDIZIONATORE ACCESO");
             digitalWrite(condizionatore,1);
90           digitalWrite(caldaia,LOW);
             udp.print("fatto");
92           udp.endPacket();
         }else if(richiesta.indexOf("Info") != -1 ){
94         // Invia la risposta al client
             String res = "ResInfo" + WiFi.localIP().toString();
96           udp.beginPacket(udp.remoteIP(), udp.remotePort());
             Serial.println(res);
98           udp.print(res);
             udp.endPacket();
100      }else{
```

```
            String mes = "non va";
102           udp.beginPacket(udp.remoteIP(), udp.remotePort());
              udp.print(mes);
104           udp.endPacket();
          }
106       Serial.println("Risposta inviata!");
      }
108   }
}
```

project/Attuatori.ino

Conclusioni

Il progetto presentato ha dimostrato come l'integrazione di più moduli ESP32, unita all'utilizzo di LabVIEW, possa costituire una soluzione efficace per il monitoraggio e il controllo di un sistema domotico di riscaldamento. Grazie alla comunicazione UDP, i dispositivi riescono a scambiare dati in modo rapido e leggero, consentendo l'aggiornamento in tempo reale dei parametri di temperatura e luminosità.

L'interfaccia grafica realizzata in LabVIEW permette di gestire in modo intuitivo le soglie di temperatura e luce, oltre a visualizzare chiaramente i valori rilevati e lo stato degli attuatori. La struttura del programma, basata su cicli While indipendenti, rende il sistema robusto nei confronti di eventuali disconnessioni, permettendo la riconnessione automatica dei dispositivi senza la necessità di riavviare l'intero software.

La soluzione è un valido esempio di applicazione pratica dei principi di domotica e automazione, con benefici in termini di efficienza energetica e comfort abitativo.

References

[1] Aosong. *DHT11 Humidity and Temperature Digital Sensor*.

[2] Espressif. *ESP32-WROOM-32 Datasheet*, 2023. Version: 3.4.

[3] Maxim Integrated. *DS18B20 Datasheet*, 2021. Rev. 6.

[4] RS Components. *Light Dependent Resistors*, 1997.

[5] Texas Instruments. *LM35 Precision Centigrade Temperature Sensors*, 2017. Rev. H.