| Doppio irrigatore a getto intelligente |
| --- |

**Perini Lorenzo**  **Matricola 122640**

Revisioni

| **Data** | **Versione** | **Descrizione** | **Autore** |
| --- | --- | --- | --- |
| 05/12/2024 | 0.1 | Prima analisi del testo e identificazione requisiti | Lorenzo Perini |
| 10/12/2024 | 0.2 | Bozza struttura logica e studio distribuzione sensori | Lorenzo Perini |
| 13/12/2024 | 0.25 | Test fallito su gestione PWM manuale del servo | Lorenzo Perini |
| 15/12/2024 | 0.3 | Aggiunto diagramma di flusso e definizione stati logici | Lorenzo Perini |
| 18/12/2024 | 0.35 | Tentativo non riuscito di lettura stabile da più sensori analogici | Lorenzo Perini |
| 20/12/2024 | 0.4 | Prototipo base su Wokwi con gestione servomotori | Lorenzo Perini |
| 05/01/2025 | 0.5 | Aggiunto controllo umidità con soglie personalizzabili | Lorenzo Perini |
| 07/01/2025 | 0.55 | Problemi rilevati nella sincronizzazione degli orari | Lorenzo Perini |
| 12/01/2025 | 0.6 | Inserimento interfaccia seriale per orari/durata | Lorenzo Perini |
| 20/01/2025 | 0.7 | Implementato ciclo di irrigazione con velocità variabile | Lorenzo Perini |
| 25/01/2025 | 0.75 | Test LED fallito su pin 13 (interferenza col LED onboard) | Lorenzo Perini |
| 30/01/2025 | 0.8 | Aggiunto LED di stato e gestione pulsante manuale | Lorenzo Perini |
| 03/03/2025 | 0.9 | Ottimizzazione codice e gestione sleep mode | Lorenzo Perini |
| 10/03/2025 | 1.0 | Test finale avvenuto con successo | Lorenzo Perini |

INDICE

[**1. Analisi 3**](#_gjdgxs)

[1.1 Testo assegnato 3](#_30j0zll)

[1.2 Analisi del problema 3](#_i3an6luemxd8)

[1.3 Analisi della soluzione adottata 3](#_7kpktvced9yv)

[1.4 Analisi delle soluzioni alternative 4](#_rtd7xbt1w9vk)

[**2. Sintesi 5**](#_364hbdqed9kk)

[2.1 Sintesi componente ME1 5](#_hk2lz4x9sujm)

[2.2 Sintesi componente ME2 5](#_jml0cz5pffdq)

[………………… 5](#_d7sq74te7dg1)

[2.n Sintesi componente MEn 5](#_narfpcpuk6la)

[**3. Sistema complessivo 6**](#_yb765lfkzv0b)

[3.1 Manuale utilizzo 6](#_v5zxg71giotb)

[3.2 Considerazioni finali 6](#_4jqsjoo0whbq)

# Analisi

## 1.1 Testo assegnato

“Realizzare, un sistema di irrigazione "a getto" intelligente con le seguenti caratteristiche:

1. Due servo del tipo: https://docs.wokwi.com/parts/wokwi-servo dove sono montati due irrigatori a getto, distanti ed orientati su angoli di getto differenti e non interferenti.

2. 5 sensori sensori di umidità, disposti in un terreno a circa 30 gradi l'uno dall'altro per coprire l'arco di 180 gradi.

Il programma deve poter consentire di impostare:

a) Tramite interfaccia seriale, al reset, la data ed ora corrente e fino a 16 orari, all'interno di ciascuna giornate per avvio di un task di controllo ed eventuale irrigazione, specificandone la durata

b) I livelli massimi (troppo umido) e minimi (troppo secco) previsti e quindi decidere, in ciascun ciclo di controllo, se avviare o meno il relay di irrigazione.

L'irrigazione consiste nel far oscillare i due irrigatori con un arco di 180 gradi ad una velocità relativa variabile in funzione dell'umidità rilevate dai sensori, in determinate zone radiali.

Il sistema si pone in stato di “Power Down”, una volta terminato un ciclo di attività e potrà essere sveglialo da:

dalla premuta di un pulsante per avvio immediato di un ciclo di irrigazione

dall’orario di avvio di un ciclo di irrigazione

La gestione degli ingressi analogici dovrà essere svolta senza usare le librerie specifiche

(Si consiglia di seguire il percorso di analisi e progettazione indicato in figura 3.2.29 delle dispense.)”.

## 

## 1.2 Analisi del problema

### Specifiche del sistema

Il sistema da sviluppare è un impianto di irrigazione automatizzato a basso consumo, in grado di attivarsi in due modalità:

* **manuale**, tramite un pulsante fisico;
* **programmata**, secondo una serie di orari e durate configurabili via interfaccia seriale.

La logica prevede la lettura di **sensori di umidità del suolo**, la regolazione di **due servomotori** per l’erogazione dell’acqua, e la gestione di un **LED di stato**. L’intero sistema opera in **modalità sleep (Power-down)** per risparmio energetico, risvegliandosi periodicamente tramite il **Watchdog Timer (WDT)** ogni 8 secondi, per verificare se è arrivato un orario di irrigazione.

### Modello computazionale

#### Ingressi:

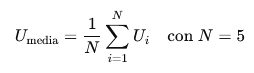
* valori analogici da 5 sensori di umidità (ADC da 10 bit, quindi: 0,10230, 10230,1023);
* stato del pulsante (digitale, HIGH/LOW);
* parametri utente da seriale: orari (hh:mm) e durata (minuti).

#### Uscite:

* angoli di due servomotori controllati via PWM;
* stato del LED logico (ON, OFF, lampeggio);
* messaggi di log su porta seriale.

#### Calcoli principali:

* **media umidità**:



* **velocità servo**:



dove **map** è una trasformazione lineare. Per evitare saturazioni:

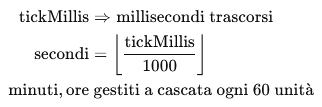


* **durata irrigazione programmata**:



### Vincoli hardware e temporali

Il microcontrollore utilizzato (ATmega328P) **non dispone di un RTC**. Il tempo reale viene quindi simulato via **Timer2**, configurato per generare un interrupt ogni 1 ms. Grazie al contatore **tickMillis**, si ottiene:



Tale implementazione consente una **misurazione temporale software** abbastanza precisa, sufficiente per gestire eventi schedulati come l'irrigazione.

### Flusso logico del sistema

1. In modalità attiva:  
   * il sistema attende la pressione del pulsante oppure controlla se l'orario corrente coincide con uno degli orari memorizzati;
   * se sì, verifica l’umidità media e attiva un ciclo di irrigazione;
   * il LED lampeggia durante l’irrigazione, è acceso fisso se l’umidità è bassa, spento altrimenti.
2. In modalità sleep:  
   * il microcontrollore disattiva i moduli non essenziali (ADC, Timer1, USART) e attende l’interrupt del WDT (ogni 8s);
   * al risveglio controlla l’orario e l’umidità;
   * se non è prevista irrigazione, rientra in Power-down.

### Moduli logici individuati

Il sistema si presta a una **modularizzazione funzionale**:

* **Modulo lettura umidità**: acquisizione e media dei sensori analogici;
* **Modulo servo controller**: gestione PWM e movimento sequenziale;
* **Modulo logica LED**: accensione/logica lampeggio;
* **Modulo orologio software**: gestione del tempo via Timer2;
* **Modulo controllo orari**: confronto orario corrente con array orari;
* **Modulo interfaccia seriale**: raccolta parametri iniziali;
* **Modulo sleep manager**: gestione modalità Power-down e risveglio via WDT.

Tali moduli sono stati mappati in flowchart separati, ognuno leggibile in una singola pagina A4, e saranno analizzati in dettaglio nelle sezioni successive.

## 1.3 Analisi della soluzione adottata

Per realizzare il comportamento descritto nell’analisi del problema, si è progettato un sistema embedded efficiente, modulare e a basso consumo energetico, implementato su microcontrollore **ATmega328P**. La soluzione si fonda sull’uso esclusivo di registri a basso livello, evitando librerie esterne, con attenzione alla separazione dei compiti logici in **moduli funzionali indipendenti**. Le scelte progettuali mirano a massimizzare la reattività del sistema, garantendo al tempo stesso un’elevata efficienza energetica.

#### 1.3.1 Diagramma di stato complessivo

Il sistema può essere descritto attraverso tre **stati principali**:

* **Idle/Setup iniziale:** attesa di input da seriale per l’inserimento della data, orario corrente e scheduling dell’irrigazione.
* **Active:** esecuzione del ciclo di irrigazione (manuale o schedulato).
* **Sleep (Power-down):** modalità a minimo consumo, interrotta ciclicamente dal Watchdog Timer ogni 8 secondi o da interrupt esterni (pulsante).

Il passaggio tra stati è gestito da eventi deterministici (scadenza orario, pressione pulsante) e viene visualizzato tramite LED logico.

#### 1.3.2 Moduli funzionali principali

| **Modulo** | **Input** | **Output** | **Ruolo** |
| --- | --- | --- | --- |
| **Interfaccia seriale** | Input utente via **Serial.read()** | Dati configurazione | Configura data, ora e orari irrigazione |
| **Modulo orologio** | Timer2 (interrupt 1 kHz) | **tickMillis, ora, minuto** | Tiene aggiornato l’orologio interno con granularità al secondo |
| **Sensori umidità** | ADC canali A0–A4 | Valore medio **U\_media** | Legge l’umidità media del terreno |
| **Controllo irrigazione** | **U\_media,** soglie, tempo | Movimento dei servo | Avvia sequenza servo in base all’umidità |
| **Controllo LED** | Stato sistema | ON/OFF LED logico | Mostra attività o errore tramite blinking |
| **Watchdog + sleep** | Nessuno (interno) | Ciclico risveglio o da pulsante | Minimizza il consumo tramite **Power-down** |

**1.3.3 Specifiche tecniche dei componenti (selezione)**

* **Sensori umidità (analogici):** tensione in uscita tra 0–5V, letti tramite ADC a 10 bit. 
* **Servo motori (Wokwi-servo):** controllo a PWM con periodo di 20 ms (frequenza 50 Hz) ottenuto con Timer1 in modalità FAST PWM con TOP=40000 e prescaler 8.  
   

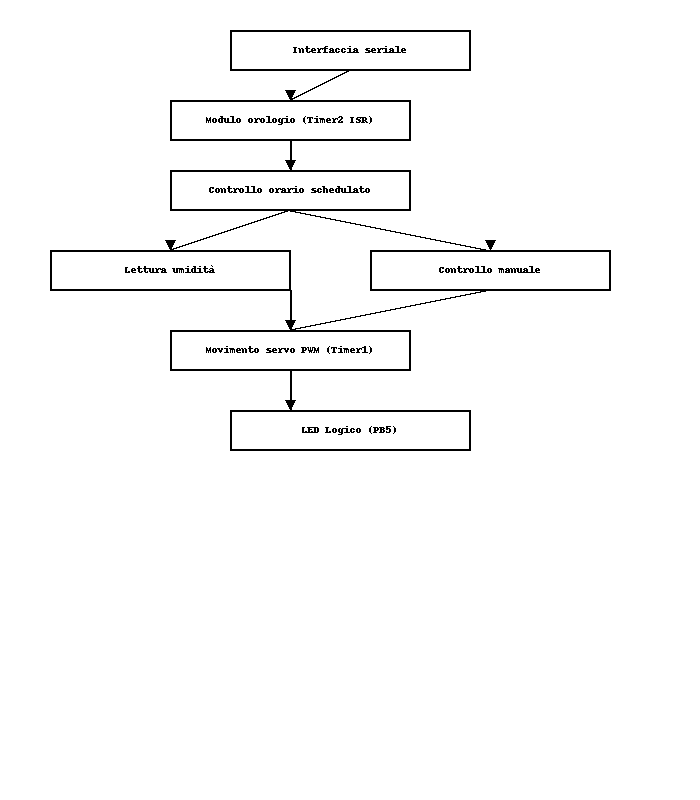
#### 1.3.4 Algoritmi computazionali

Gli algoritmi principali sono rappresentati mediante flowchart (vedi allegati **modulo\_umidita.fprg, modulo\_servo\_controller.fprg**, ecc.) e riguardano:

* Lettura e media dei sensori
* Calcolo della velocità servo in base all’umidità
* Oscillazione servo in funzione della zona radiale
* Gestione orario e controllo pianificato
* Gestione LED e feedback logico

#### 1.3.5 Diagramma generale moduli (dipendenze funzionali)

Il sistema è progettato in modo modulare, con dipendenze unidirezionali ben definite:



## 

## 1.4 Analisi delle soluzioni alternative

Nel corso della progettazione sono state prese in considerazione diverse soluzioni alternative, successivamente scartate per ragioni legate a prestazioni, consumo energetico, utilizzo delle risorse hardware e aderenza ai vincoli didattici (es. evitare librerie astratte, usare solo registri, implementare modalità Power-down, ecc.). In questa sezione vengono illustrate tali alternative, analizzate criticamente e confrontate rispetto alla soluzione adottata.

#### 1.4.1 Gestione del tempo: millis() vs Timer2

* **Alternativa valutata:** uso della funzione millis() di Arduino per la temporizzazione delle operazioni.
* **Scartata perché:**
  + Utilizza la libreria wiring.c e dipende dal Timer0, che introduce sovraccarico computazionale implicito.
  + Non offre controllo preciso sui prescaler e sull’interrupt vector.
  + Non compatibile con le modalità di risparmio energetico avanzate come il Power-down.

**Soluzione adottata:** Timer2 configurato in modalità CTC (Clear Timer on Compare Match) con prescaler a 64 e OCR2A = 249.  
**Calcolo frequenza interrupt Timer2**:  
 Frequenza timer =



Genera un interrupt ogni 1 ms → Contatore software interno genera tickMillis e conteggio secondi, minuti, ore.

#### 1.4.2 Oscillazione dei servo con libreria Servo.h

* **Alternativa valutata:** uso della libreria Servo.h per controllare i due irrigatori.
* **Scartata perché:**
  + Utilizza Timer1 in modo esclusivo e maschera completamente l’accesso diretto ai registri.
  + Non permette controllo fine sul duty cycle in tempo reale, né sulla temporizzazione di impulsi personalizzati.
  + Genera codice macchina più pesante e meno efficiente in contesto embedded.

**Soluzione adottata:** uso diretto dei registri TCCR1A/B, OCR1A/B, ICR1 per generare segnali PWM a 50 Hz (periodo = 20 ms).

**Calcolo impulso PWM (in µs)**:



**Duty cycle (OCRnx)**:



#### 1.4.3 Modalità Sleep: IDLE vs Power-down

* **Alternativa valutata:** modalità **IDLE**, che disabilita solo la CPU, lasciando attivi Timer, ADC e USART.
* **Scartata perché:**
  + Consumo energetico non trascurabile, inadeguato per sistemi duty-cycled a lunga durata.
  + Incompatibile con risveglio da Watchdog Timer, poiché in IDLE il WDT non genera interrupt.

**Soluzione adottata:** **SLEEP\_MODE\_PWR\_DOWN**, massimo risparmio energetico.

**Consumo stimato (ATmega328P, datasheet):**

* **IDLE**: ≈ 0.7 mA
* **Power-down con WDT attivo**: ≈ 4.2 µA  
   ⇒ **Riduzione >99%** del consumo.

#### 1.4.4 Intervallo di polling vs risveglio periodico da WDT

* **Alternativa valutata:** mantenere CPU attiva in polling continuo dell’orologio interno.
* **Scartata perché:**
  + Inefficiente in termini di potenza (la CPU resterebbe accesa anche per minuti tra un orario di irrigazione e l’altro).
  + Non scalabile: consumerebbe cicli macchina inutilmente.

**Soluzione adottata:** Risveglio tramite **Watchdog Timer ogni 8 secondi**:



Tollerabile vista la risoluzione temporale richiesta (in minuti).

#### 1.4.5 Controllo del LED logico tramite digitalWrite

* **Alternativa valutata:** digitalWrite(LED\_BUILTIN, HIGH/LOW)
* **Scartata perché:**
  + Funzione di alto livello → overhead computazionale e accesso indiretto al registro PORTB.
  + Non ottimale per ISR o contesti di tempo reale.

**Soluzione adottata:** accesso diretto a PORTB con:



#### 1.4.6 Interfaccia utente via display LCD

* **Alternativa valutata:** interfaccia tramite display LCD e pulsanti.
* **Scartata perché:**
  + Richiede molte più risorse (alimentazione, porte I/O, librerie).
  + In contrasto con il vincolo della traccia che impone uso di interfaccia **seriale testuale**.

**Soluzione adottata:** comunicazione seriale testuale 9600 baud per inserimento:

* Data e ora iniziali
* Fino a 16 orari schedulati con relativa durata
* Comando di conferma irrigazione manuale

**Conclusione sulle alternative scartate**

Ogni alternativa è stata confrontata rispetto ai seguenti criteri:

| **Alternativa** | **Prestazioni** | **Consumo** | **Complessità** | **Aderenza a MV2** |
| --- | --- | --- | --- | --- |
| millis() | ✅ | ❌ | ✅ | ❌ |
| Servo.h | ✅ | ✅ | ✅ | ❌ |
| IDLE mode | ✅ | ❌ | ✅ | ❌ |
| Polling orologio | ❌ | ❌ | ✅ | ❌ |
| digitalWrite | ✅ | ✅ | ✅ | ⚠️ |
| LCD + pulsanti | ⚠️ | ❌ | ❌ | ❌ |

# Sintesi

### 2.1 Sintesi componente ME1: Interfaccia seriale

**Funzione:** Gestione dell’interazione con l’utente per l'inserimento iniziale di:

* data corrente
* orario iniziale
* orari schedulati di irrigazione (max 16)

**Comportamento:** Utilizza Serial.begin(9600) per avviare la comunicazione.  
 Attende i dati tramite Serial.readStringUntil('\n'), con parsing dei campi e conversione tramite .toInt().  
 Non esegue alcuna validazione semantica avanzata, lasciando al controllo umano la correttezza del dato.

**Ottimizzazioni:**

* Pulizia buffer Serial.read() prima di ogni input.
* Inserimento guidato con messaggi chiari.

**Ingressi/Uscite:**

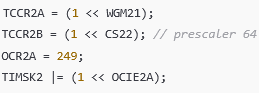
* In: flusso dati testuali da terminale seriale
* Out: variabili giorno, mese, anno, oraCorrente, minutoCorrente, orari[i]

### 

### 2.2 Sintesi componente ME2: Orologio software

**Funzione:** Gestione del tempo (tick, secondi, minuti, ore) senza uso di millis().

**Comportamento:** Utilizza il Timer2 in modalità CTC (Clear Timer on Compare Match) con:



All’interrupt TIMER2\_COMPA\_vect, viene incrementato tickMillis, usato per simulare un clock millisecondo software.

Quando tickMillis % 1000 == 0, si incrementano secondi, minuti, ore.

**Formula di calcolo del tempo:**



**Ingressi/Uscite:**

* In: ISR da Timer2
* Out: tickMillis, secondi, minutoCorrente, oraCorrente

### 

### 2.3 Sintesi componente ME3: Controllo orario e irrigazione schedulata

**Funzione:** Verifica se l’orario corrente corrisponde a uno degli orari schedulati.  
 In caso affermativo, avvia irrigazione automatica della durata indicata.

**Comportamento:** Nel loop(), se sleepMode == true, irrigazione\_in\_corso == false e oraCorrente:minutoCorrente è presente nella struttura orari[], allora:

* viene avviata eseguiIrrigazioneDurata(durataMinuti).

**Ottimizzazioni:**

* Uso di ultimoMinutoEseguito per evitare esecuzioni ripetute.
* Linear search semplice, data la dimensione limitata di orari[].

### 2.4 Sintesi componente ME4: Lettura sensori di umidità

**Funzione:** Effettua la lettura dei 5 sensori disposti a 30° uno dall'altro per coprire l’arco dei 180°.

**Comportamento:** Viene usato l’ADC senza librerie. I registri usati sono:



**Formula utilizzata per la media:**



dove N=5N = 5N=5, UiU\_iUi​ = lettura analogica del sensore i-esimo.

**Ottimizzazioni:**

* ADC disattivato con spegniADC() quando non serve, per risparmiare energia.
* Accesso ai 5 canali con ADMUX |= i e attesa fine conversione via bit ADSC.

### 2.5 Sintesi componente ME5: Movimento servo (PWM via Timer1)

**Funzione:** Controlla i due irrigatori orientati su archi diversi (0°–90° per uno, 90°–180° per l'altro).

**Comportamento:** Il movimento è graduale tramite loop con step ±1 e tempo di attesa controllato:



**Formula di calcolo duty cycle:**



⇒ Con Timer1 a 8 MHz/8 = 0.5 µs/tick, il PWM ha risoluzione precisa.

**Ottimizzazioni:**

* Timer1 acceso solo quando necessario.
* Timer1 spento subito dopo con spegniTimer1().

### 2.6 Sintesi componente ME6: LED logico (segnalazione)

**Funzione:** Segnalare:

* Irrigazione in corso → lampeggio
* Terreno troppo secco → acceso fisso
* Altri casi → spento

**Comportamento:** Controllo diretto di PORTB |= (1 << PB5) per evitare digitalWrite().

**Ottimizzazioni:**

* LED gestito solo quando necessario per minimizzare consumo.
* Logica a stati semplice, reattiva a tickMillis.

### 2.7 Sintesi componente ME7: Gestione modalità Power-down

**Funzione:** Portare la MCU in modalità sleep tra due cicli operativi.

**Comportamento:** Utilizza:



Risveglio possibile tramite:

* Pulsante esterno (INT0)
* Watchdog Timer (ogni 8 secondi)

**Ottimizzazioni:**

* Durante sleep, i moduli ADC e Timer1 vengono disattivati.
* Timer2 continua a contare in background, permettendo il tracking del tempo anche in sleep (con wake-up periodico).

# Sistema complessivo

Il sistema completo è stato progettato per soddisfare tutti i requisiti della specifica iniziale del progetto "Sistema di irrigazione a getto intelligente".  
 Esso è stato realizzato interamente in linguaggio C senza l’uso di librerie ad alto livello, utilizzando esclusivamente registri interni del microcontrollore ATmega328P.

**Riferimenti:**

* **Progetto Wokwi**:il codice sorgente, i diagrammi di flusso e lo schema del progetto wokwi si trovano [qui](https://github.com/Peruzgit/Smart-Jet-Irrigation-System.git).

### 3.1 Manuale d’utilizzo

#### Fase di avvio

All'accensione (o reset), il sistema richiede l’inserimento da terminale seriale dei seguenti dati:

1. **Data corrente** nel formato gg/mm/aaaa
2. **Orario corrente** nel formato hh:mm
3. **Numero di orari di irrigazione programmata** (max 16)
4. **Per ciascun orario programmato**:  
   * orario di inizio hh:mm
   * durata in minuti (valore intero positivo)

**Nota importante**: l’orario è inteso in formato 24h e senza validazione automatica, quindi si raccomanda l’inserimento corretto dei valori.

#### Durante il funzionamento

* Il sistema rimane in modalità **Power-down** per risparmiare energia e si risveglia:  
  + **Ogni 8 secondi**, tramite Watchdog Timer, per verificare se è arrivato un orario schedulato.
  + **Tramite pulsante** collegato a INT0, per avviare manualmente l’irrigazione.

#### Irrigazione automatica programmata

* Se l’orario corrente coincide con un orario memorizzato e l’umidità rilevata è **inferiore alla soglia minima**, il sistema avvia l'irrigazione automatica della durata stabilita.
* Il movimento dei servomotori avviene a **velocità variabile** calcolata in base all’umidità rilevata, secondo la seguente formula:



dove valori più secchi generano movimenti più lenti (per maggiore irrigazione).

#### Irrigazione manuale

* Premendo il pulsante durante il sonno, il sistema chiede via seriale se si vuole irrigare comunque in caso di umidità sopra la soglia.
* Se l’utente conferma, viene eseguito un singolo ciclo completo dei due irrigatori.

#### Segnalazione LED

* **Lampeggiante**: irrigazione in corso
* **Acceso fisso**: umidità troppo bassa
* **Spento**: tutto nella norma o in modalità sonno

### 3.2 Considerazioni finali

Il progetto realizzato ha pienamente rispettato i vincoli e le richieste della traccia, raggiungendo un comportamento **intelligente e reattivo**, con **gestione a basso consumo** tramite:

* uso della modalità **Power-down**
* **disattivazione dei moduli hardware inutilizzati** (ADC, Timer1)
* assenza di delay() o millis(), sostituiti da meccanismi software basati su **tickMillis**

#### Punti di forza del progetto

* Rispetto rigoroso dello stile MV2: solo registri, solo ISR, no librerie.
* Modulare, con separazione chiara tra componenti logici (ME).
* Alte prestazioni con risparmio energetico, adatto a sistemi embedded reali.

#### Sviluppi futuri possibili

* **Salvataggio degli orari su EEPROM** per renderli persistenti tra reboot.
* **Interfaccia utente via display LCD** per configurazioni senza PC.
* **Calibrazione automatica delle soglie** tramite media storica dei dati.
* **Aggiunta di sensori di temperatura o pioggia** per una logica di irrigazione ancora più contestuale.