

ALPHA

Acquisizione Locale di Parametri con
Hardware Avanzato

Alessio Tommasi

2 maggio 2025

Indice

1	Introduzione	3
1.1	Dipendenze	3
1.2	Configurazione dell'Arduino IDE	4
2	Hardware	5
2.1	Funzionamento	5
2.1.1	Alimentazione	5
2.1.2	Ingressi Digitali	7
2.1.3	Uscite	8
2.1.4	Multiplex	10
2.1.5	hardware esterno posizionabile sulla board	12
2.2	Hardware esterno	14
2.2.1	Winzet W5500 Ethernet module	14
2.2.2	MAX485 Modbus RTU	16
2.2.3	ADS1115 ADC	18
2.2.4	ESP32 38 Pin	19
3	Software	23
3.1	Diagramma UML	23
3.1.1	Pattern	23
3.2	Performace	24
3.3	WebServer	25
3.3.1	Componenti HTML Dinamici	26
3.3.2	Vantaggi del Framework	28
3.3.3	Conclusioni	28
3.4	Modbus	28
3.5	Sfide	28
4	Attività	29

5 Conclusioni	30
5.1 Demo	30
5.2 Sviluppi futuri	30

Capitolo 1

Introduzione

Il progetto *ALPHA* è stato sviluppato nel corso di IoT del Master in Informatica presso SUPSI. Il focus principale è sull'ESP32 e il protocollo Modbus.

La documentazione ufficiale del progetto disponibile al seguente link:
[ALPHA](#).

1.1 Dipendenze

Driver Per gli utenti Windows, è necessario installare [CP210xDriver](#)

Compiler Per compilare tale progetto è stato utilizzato Arduino IDE 2.3.3. disponibile al seguente link: [Arduino IDE](#).

Pubblic library

AsyncTCP ulteriori informazioni sono disponibili a questo link [AsyncTCP](#)

ESPAsyncTCP ulteriori informazioni sono disponibili a questo link [ESPAsyncTCP](#)

ESPAsyncWebServer ulteriori informazioni sono disponibili a questo link [ESPAsyncWebServer](#)

UIPEthernet ulteriori informazioni sono disponibili a questo link [UIPEthernet](#)

Modbus Sono state testate diverse librerie, ma nessuna di esse ha funzionato in modo ottimale.

1.2 Configurazione dell'Arduino IDE

Link repo ufficiale: [iotProject](#).

Per compilare i file nelle sottocartelle, è necessario aggiungerli come librerie (.zip) all'Arduino IDE. Ho creato una cartella specifica per le librerie dove posizionare o sostituire i file zip. Per una corretta compilazione, importa tutte le cartelle zip presenti in **/Library**.

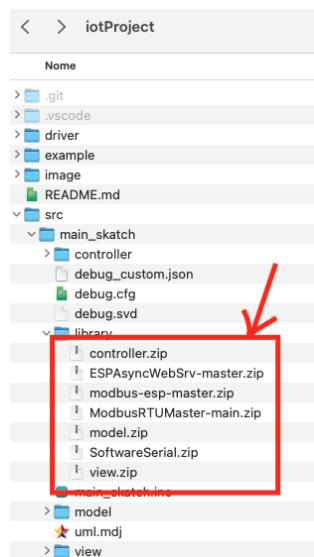


Figura 1.1: Importazione delle librerie nell'Arduino IDE

Altrimenti clonare la versione Portable del progetto disponibile al seguente link: [iotProject-portable](#).

Compilatore per la compilazione utilizzare la board **ESP32 DEV Module**.

Board espresif per una corretta compilazione si necessita dell'utilizzo della versione 3.0.7 delle board fornite da Espressif. Maggiori dettagli a questo link.

Capitolo 2

Hardware

2.1 Funzionamento

2.1.1 Alimentazione

E' possibile alimentare la board tramite un alimentatore esterno da 24V DC che fornisca almeno 0.5A.

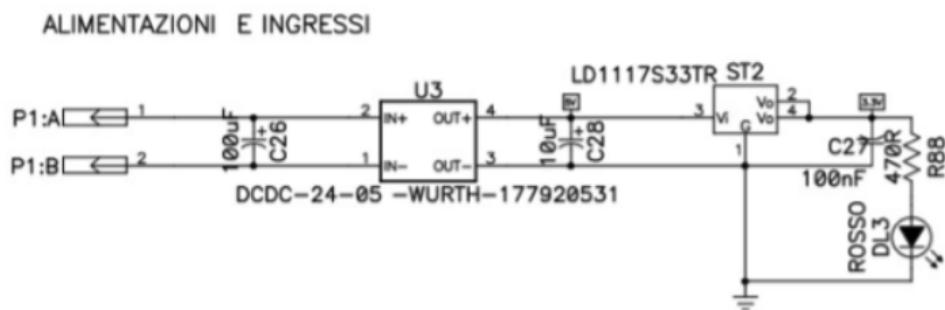


Figura 2.1:

Descrizione Lo schema in Figura 2.1 rappresenta il sistema di alimentazione con regolatori di tensione e relativi condensatori di filtraggio. I connettori P1:A e P1:B forniscono l'ingresso principale della tensione al circuito.

Conversione DC-DC Il primo stadio dell'alimentazione è rappresentato dal modulo DC-DC Wurth 177920531, (Data Sheet Here) che si occupa della conversione della tensione di ingresso a un livello adeguato per il regolatore successivo.

La stabilizzazione della tensione è garantita dai condensatori C26 e C28 entrambi da 100 μ F, i quali riducono eventuali ripple presenti nella tensione di alimentazione.

Regolazione di tensione Dopo la conversione DC-DC, il regolatore lineare LD1117S33TR (U3 sulla board) fornisce una tensione stabile di 3.3V, necessaria per l'alimentazione dei componenti successivi. Questo regolatore include un condensatore di ingresso C28 di 10uF e un condensatore di uscita C27 di 100nF per migliorare la stabilità del segnale. Un LED rosso (D3) con la resistenza R88 fornisce un'indicazione visiva della corretta alimentazione del sistema.

Regolatore Switching LM7660 Un ulteriore regolatore di tensione, l'LM7660 (U10), viene utilizzato per generare una tensione negativa o fornire una conversione di tensione specifica.

Questo componente opera con i condensatori C29 e C30 (entrambi da 10uF), i quali servono per stabilizzare la tensione e ridurre le oscillazioni indesiderate.

Uscite Le alimentazioni ottenute distribuiti tramite i connettori P20:1, P20:2, P21:1, P21:2, P22:1 e P22:2, che permettono l'integrazione del sistema con altri moduli elettronici.

P20:1 e P20:2 forniscono la tensione di alimentazione principale a 5V, mentre P21:1 e P21:2 forniscono la tensione negativa a -5V. P22:1 e P22:2 forniscono la tensione di alimentazione a 3.3V.

2.1.2 Ingressi Digitali

Questo sezione descrive l'implementazione degli ingressi digitali nel circuito, illustrando il principio di funzionamento, la protezione e il filtraggio dei segnali.

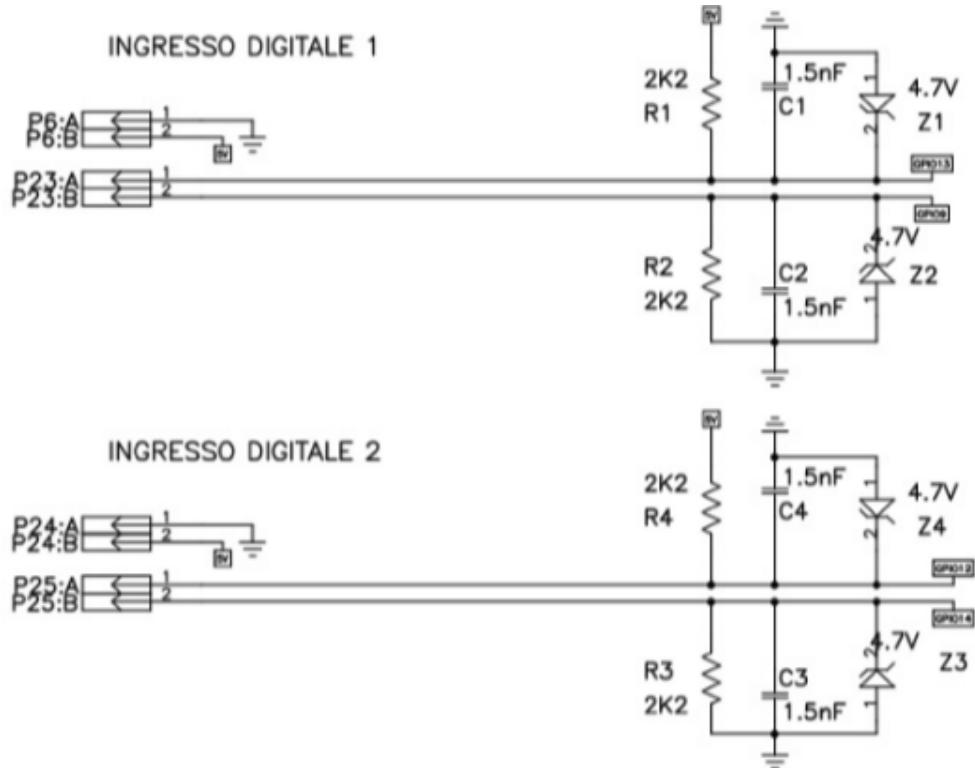


Figura 2.2:

2.1.3 Uscite

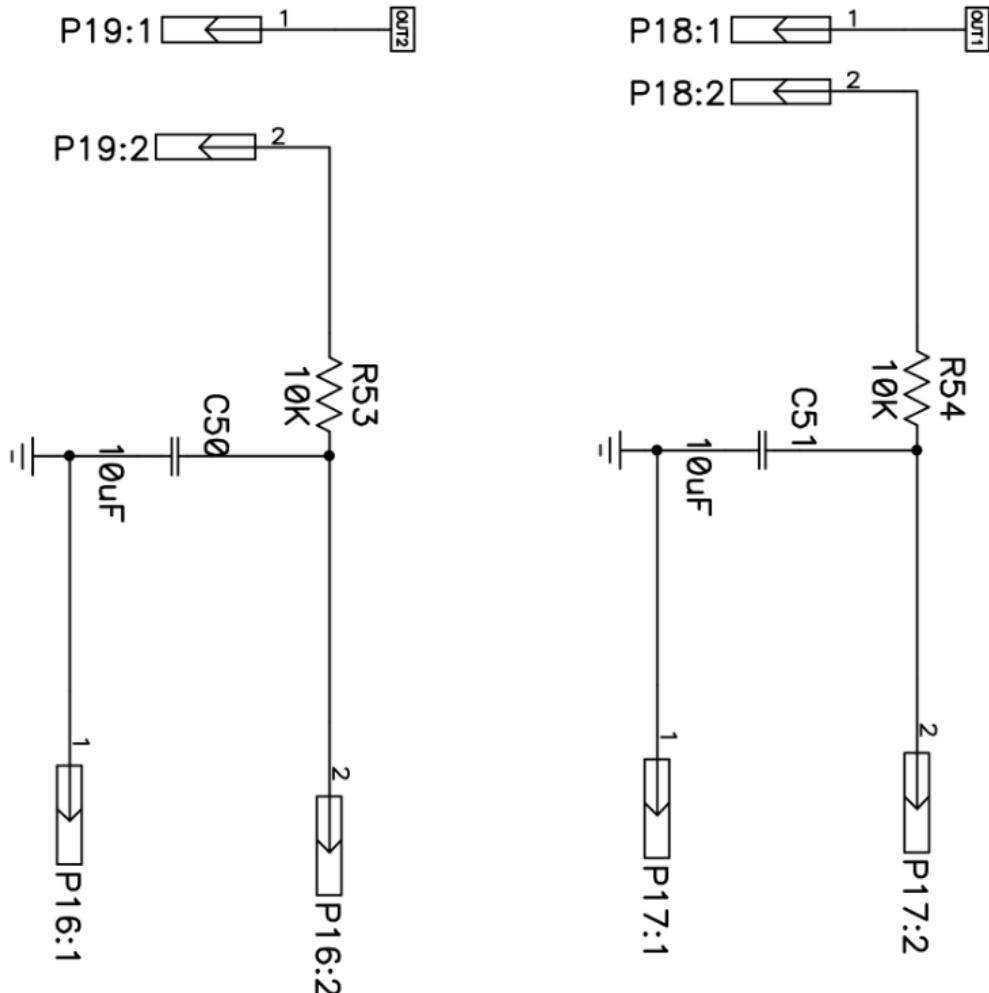


Figura 2.3:

Descrizione Il circuito degli ingressi digitali è progettato per accettare segnali in ingresso e condizionarli adeguatamente prima di inviarli alla logica di elaborazione.

Ogni ingresso è dotato di:

- **Resistenze** da $2.2\text{ k}\Omega$ rispettivamente (R_1, R_2) per ingresso digitale 1 e (R_3, R_4) per ingresso digitale 2: il loro scopo è limitare la corrente di ingresso e formare un partitore resistivo.

- Condensatori** da 1.5 nF (C1, C2, C3, C4): implementati per ridurre il rumore ad alta frequenza e migliorare l'integrità del segnale.
- Diodi Zener** da 4.7V (Z1, Z2, Z3, Z4): impiegati per proteggere l'ingresso da sovratensioni accidentali che potrebbero danneggiare i componenti a valle.

Conclusioni Grazie all'uso combinato di resistenze, condensatori e diodi di protezione, il circuito è in grado di accettare segnali digitali provenienti da diversi dispositivi, mantenendo una buona immunità al rumore. Inoltre, la configurazione utilizzata permette di garantire livelli logici stabili e ben definiti.

Dunque il sistema di ingressi digitali descritto rappresenta una soluzione efficace per la gestione di segnali binari, garantendo protezione, stabilità e robustezza. Questa configurazione assicura un'interfaccia affidabile tra il mondo fisico e il sistema di elaborazione.

2.1.4 Multiplex

Il dispositivo di multiplexing riulta essere il **CD405xB**. sviluppato da Texas Insreumet, link alla documentazione ufficiale:
CD405xB.

Da cui provengono le seguenti immagini e informazioni:

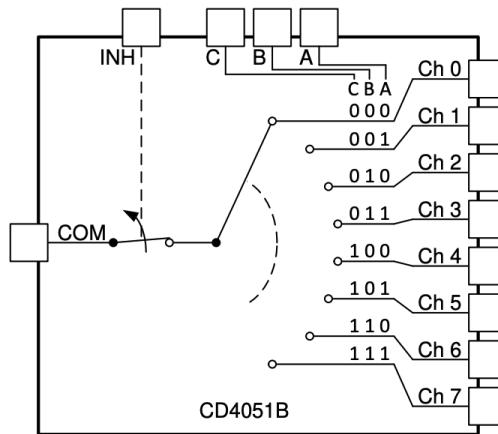


Figura 2.4: Tabella di verità del CD405xB

Collegamenti Multiplexer

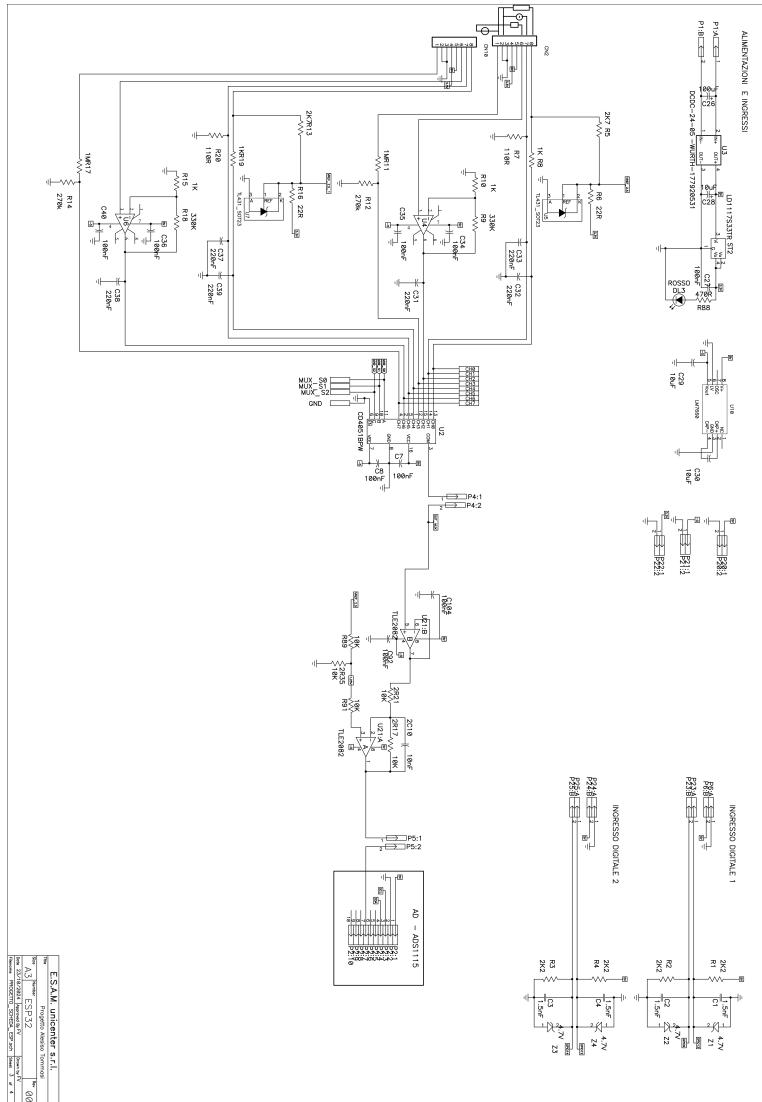
I pin di ingresso A, B, C del multiplexer **CD405xB** sono collegati rispettivamente ai pin GPIO 12, 13, 14 dell'ESP32. La selezione dei canali del multiplexer avviene impostando i pin A, B, C come segue:

Canale	Pin A	Pin B	Pin C
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1
7	1	1	1

Tabella 2.1: Configurazione dei pin per la selezione dei canali del multiplexer

i canali del multiplexer possono essere impostati nella apposita pagina web, la documentazione è disponibile più avanti in questo documento.

Dettaglio canali multiplexer Di seguito sono riportati come i canali si connettono ai segnali esterni, le operazioni effettuate su tali segnali verranno spiegate più nel dettaglio nella sezioni successive.



2.1.5 hardware esterno posizionabile sulla board

MAX31865

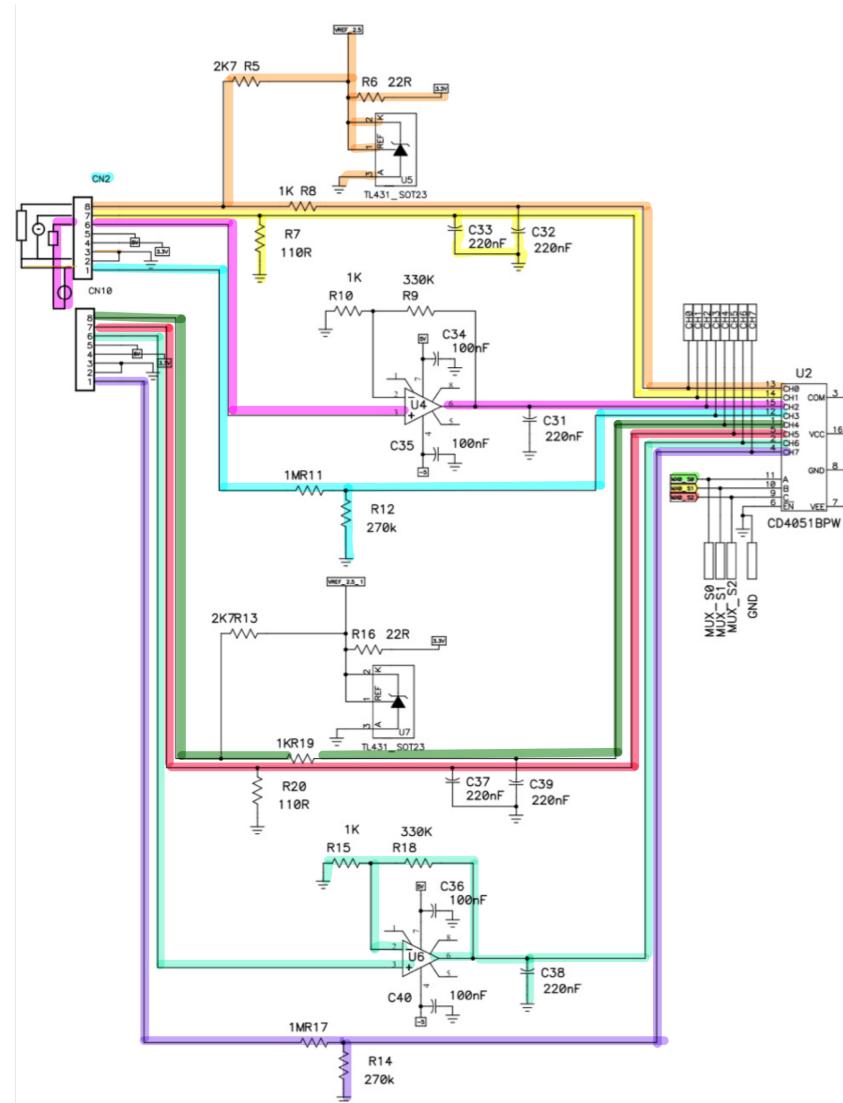


Figura 2.6: Collegamenti dei canali del multiplexer

Ch0 Selezionando Ch0 dall apposita pagina dedicata segue il percorso evidenziato in arancione nella Figura 5.2. E' collegato a pin 8 della morsettiera CN2.

Il segnale in ingresso proviene da una resistenza esterna collegata al connettore (tra i morsetti 8 e 2 e 1).

Una corrente di circa 9 μ A viene iniettata nel morsetto 8.

Questa corrente, attraversando la resistenza esterna, genera una caduta di tensione che viene selezionata tramite il canale Ch0 del multiplexer.

Successivamente, il segnale passa attraverso uno stadio amplificatore per essere portato a un livello adeguato per l'ADC dell'ESP32.

Il componente TL431 funge da riferimento di tensione, ovvero un generatore di tensione costante e calibrata.

Ch1 percorso evidenziato in giallo nella Figura 5.2.

Possibili funzioni del segnale sul segnale in ingresso al pin 7 della morsettiera CN2.

- **Filtro Passa Basso** Se il segnale applicato su R7 è una tensione alternata (AC), il circuito attenua le alte frequenze, lasciando passare solo le frequenze più basse

- **Stabilizzazione del segnale**

Ch2 NON giusto Chiedi !!!

TODO

Ch3 percorso evidenziato in azzurro nella Figura 5.2.

Leggeremo il segnale in ingresso al pin 1 della morsettiera CN2. tra il segnale sul pin 1 ed ESP e' presente un partitore di tensione composto da $R_{11} = 1M$ e $R_{12} = 270K$ dunque

$$V_{esp} = V_{pin1} * \frac{R_{12}}{R_{11} + R_{12}} = V_{pin1} * \frac{270}{1270} = V_{pin1} * 0.2126.$$

Il segnale viene utilizzato internamente per il calcolo della resistenza collegata sul pin 8 della morsettiera CN2.
tramite la seguente formula:

$$R = \frac{V_{ch0} * 4.7 * 1000}{V_{ch2}}$$

Ch4 Connessione equivalente a CH0

Ch5 Connessione equivalente a CH1

Ch6 Connessione equivalente a CH3

Tutte le connessioni equivalenti hanno un circuito hw separato per garantire la massima indipendenza tra i canali.

2.2 Hardware esterno

2.2.1 Winzet W5500 Ethernet module

tale dispositivo è opzionale e deve essere collocato sulla board in alternativa alla scheda SD.

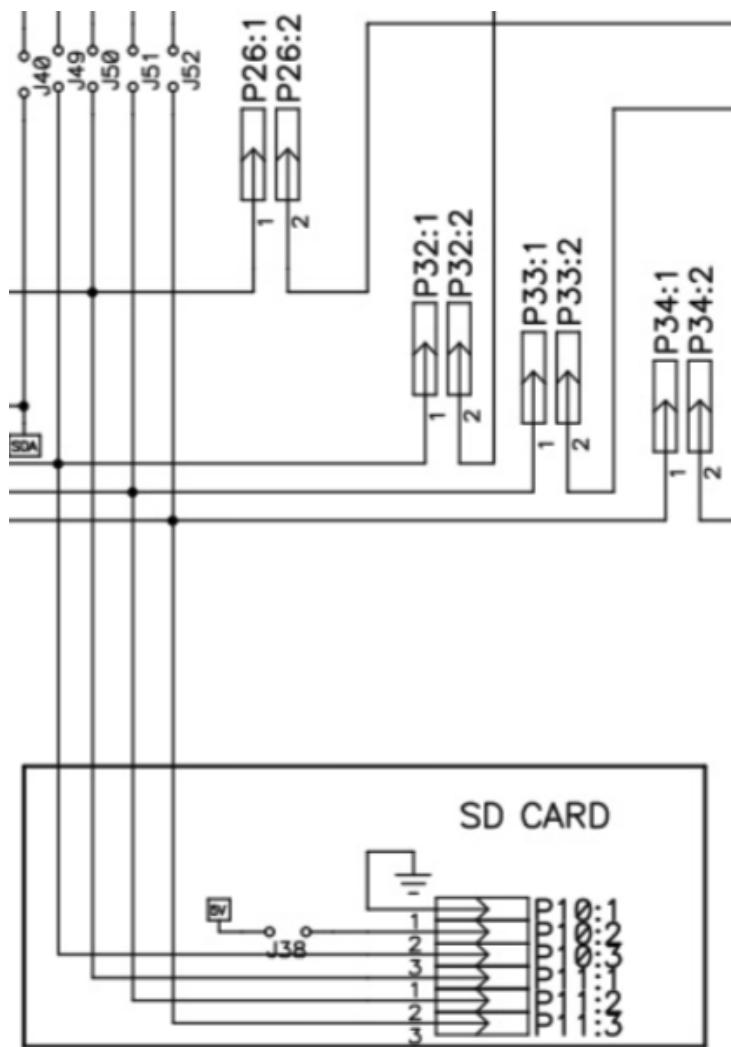


Figura 2.7: Winzet W5500 Ethernet module

per il corretto funzionamento si devono collocare manualmente sullla board seguenti jumper: J49, J50, J51, J52 e J38 visibili in figuras 2.5.

Per una documentazione dettagliata sulla ricerca di tale modulo e sul codice e librerie utilizzate da esso consultare il seguente link: W5500.

Si e optato per l'utilizzo di un modulo Ethernet W5500 per la comunicazione tramite Modbus TCP/IP.

- documentazione dettagliata modbus tcp/ip Modbus TCP/IP

funzionamento modbus TCP/IP

Modbus TCP è stato sviluppato per sfruttare le infrastrutture di rete LAN esistenti, permettendo la comunicazione attraverso reti Ethernet.

Attualmente viene utilizzato per qualsiasi connessione tra dispositivi connessi a internet.

Questo protocollo incapsula i messaggi Modbus RTU in pacchetti TCP, il che facilita la loro trasmissione su reti Ethernet standard. Uno dei principali vantaggi del Modbus TCP è la sua capacità di connettere un numero illimitato di dispositivi, grazie all'uso di indirizzi IP invece delle limitazioni di indirizzamento dei protocolli seriali.

In questo contesto, Modbus TCP ridefinisce la relazione master-slave in termini di client-server, permettendo una comunicazione più flessibile e scalabile. I dispositivi possono agire come client o server, facilitando l'integrazione di più sistemi e migliorando l'efficienza delle comunicazioni nelle reti industriali.

link bibiografia Modbus TCP/IP

Vantaggi W5500

- **Alta velocità e prestazioni:** Supporta fino a 80 Mbps grazie al buffer hardware dedicato.
- **Stack TCP/IP hardware integrato:** Libera risorse sul microcontrollore ESP32.
- **Consumo energetico ridotto:** Più efficiente rispetto all'ENC28J60, ideale per applicazioni a basso consumo.
- **Compatibilità ampia:** Supportato dalla libreria Ethernet ufficiale di Arduino.

Per visualizzare una ricerca più dettagliata sul confronto di tale modulo con altri moduli presenti sul mercato consultare il seguente link:
[ResearchEthModule](#).

2.2.2 MAX485 Modbus RTU

Tale moduli sono utilizzati per la comunicazione seriale tra ESP32 e altri dispositivi che supportano il protocollo Modbus RTU.
il dispositivo deve essere alimentato con 5V e collegato ai pin RX e TX dell'ESP32. per far sì che la comunicazione funzioni correttamente collegare cavo trasporto dati dell'usb e collegare al dispositivo Master o slave.

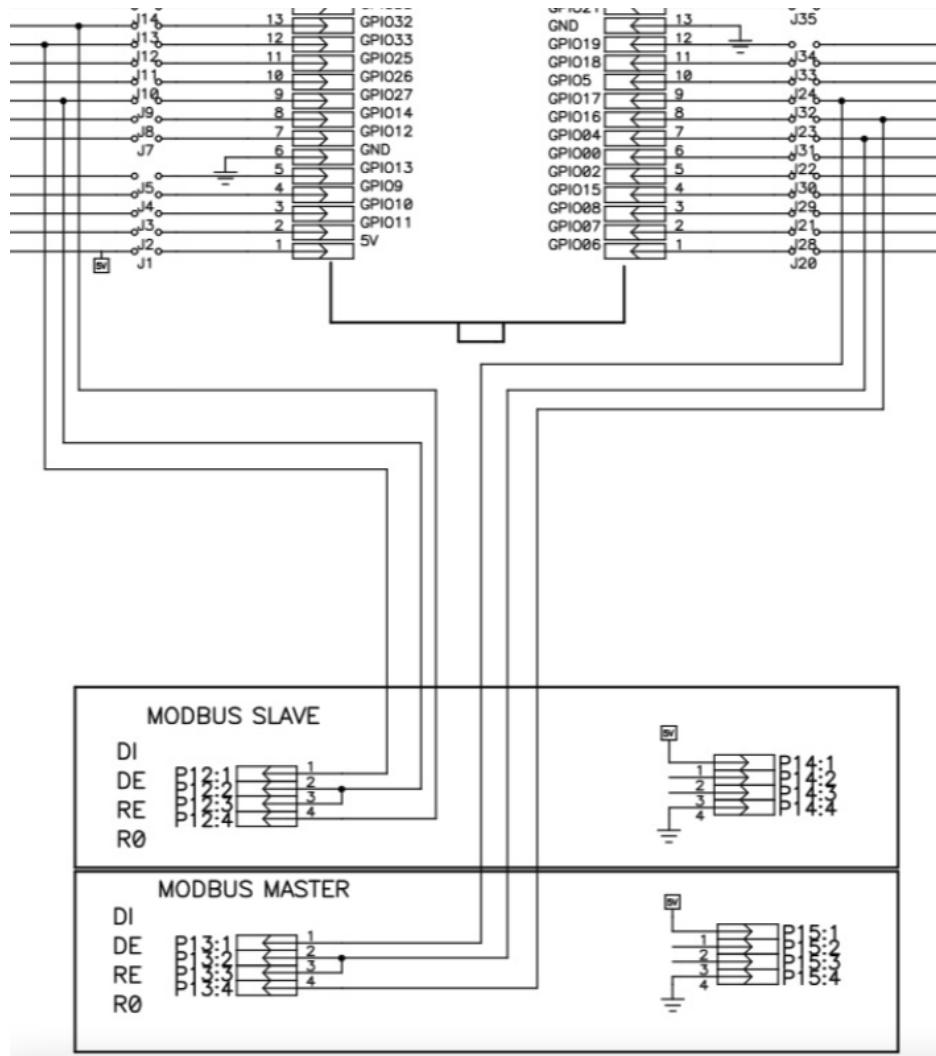


Figura 2.8: MAX485 Modbus RTU

Descrizione Protocollo Modbus RTU

Modbus RTU è basato su una comunicazione seriale asincrona, che consente la trasmissione di dati su cavi.

Per una corretta comunicazione il dispositivo MAX485 SLAVE deve essere collegato su P12 mentre il dispositivo MASTER su P13.
i collegamenti ai pin dell'esp sono rappresentati in figura 2.6.

- **Caratteristiche**

- Formato di messaggio compatto: I messaggi sono compatti, il che consente una trasmissione più rapida ed efficiente.
- Modalità binaria: I dati vengono trasmessi in un formato binario, il che significa che si usano bit (0 e 1).

- **Vantaggi del Modbus RTU**

- Ideale per distanze brevi e medie.
- Semplice ed economico per installazioni piccole o medie.

- **Limitazioni del Modbus RTU**

- Limitazioni di distanza e velocità.
- Numero limitato di dispositivi che possono essere collegati sulla stessa linea di comunicazione, ovvero 32 dispositivi.

2.2.3 ADS1115 ADC

Il modulo ADS1115 è un convertitore analogico-digitale (ADC) a 16 bit con quattro canali di ingresso.

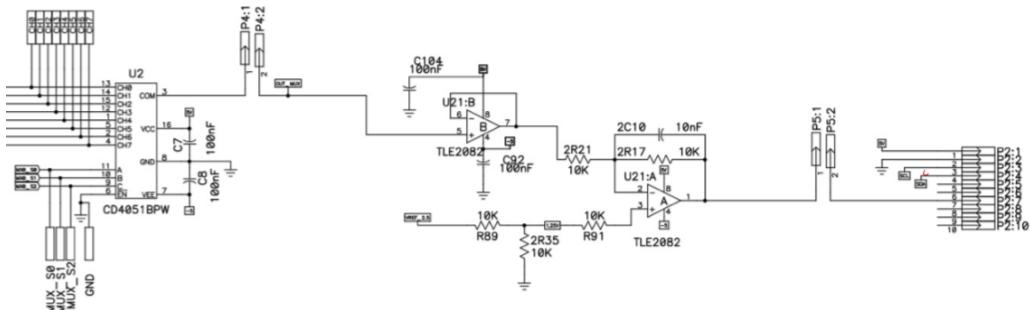


Figura 2.9: ADS1115 ADC

Il dispositivo utilizza un multiplexer descritto nella sezione 2.1.3 per selezionare uno dei canali di ingresso, amplifica il segnale tramite il PGA e poi lo converte in un valore digitale.

Il risultato della conversione viene inviato al microcontrollore tramite l'interfaccia I2C.

il datasheet dettagliato di tale dispositivo e' disponibile al seguente link:
[ADS1115](#).

Pin per trasferimento dati:

Il collegamento SCL dell ADS una volta collegato il jumper J39 si collega a PIN39 ovvero GPIO 22

Il collegamento SDA dell ADS una volta collegati il jumper J40 si collega al pin GPIO 21

2.2.4 ESP32 38 Pin

Descrizione

L'ESP32 è un potente microcontrollore prodotto da Espressif Systems che integra funzionalità Wi-Fi e Bluetooth, rendendolo ideale per applicazioni IoT (Internet of Things), automazione domestica, sistemi indossabili, monitoraggio remoto e molti altri progetti.

La versione a 38 pin offre una vasta gamma di GPIO (General Purpose Input/Output) utilizzabili per diversi scopi.

Caratteristiche principali:

- **Dual-core Tensilica LX6:** Processore dual-core con velocità di clock fino a 240 MHz.
- **Wi-Fi e Bluetooth:** Supporto per Wi-Fi 802.11b/g/n e Bluetooth v4.2 BR/EDR e BLE (Bluetooth Low Energy).
- **Interfacce Multiple:** Include SPI, I2C, I2S, UART, ADC, DAC, PWM, e touch sensor.
- **Basso Consumo Energetico:** Modalità di basso consumo per applicazioni a batteria.
- **Memoria Integrata:** SRAM da 520 KB e supporto per memoria esterna.

Link alla documentazione ufficiale: [ESP32](#).

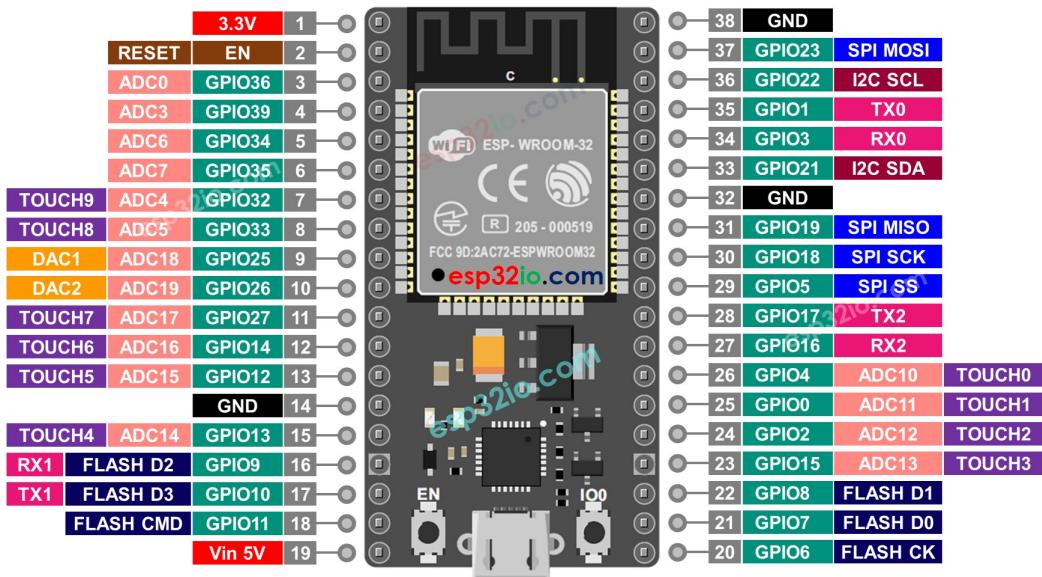


Figura 2.10: Pinout dell'ESP32-DOIT-DEV-KIT v1

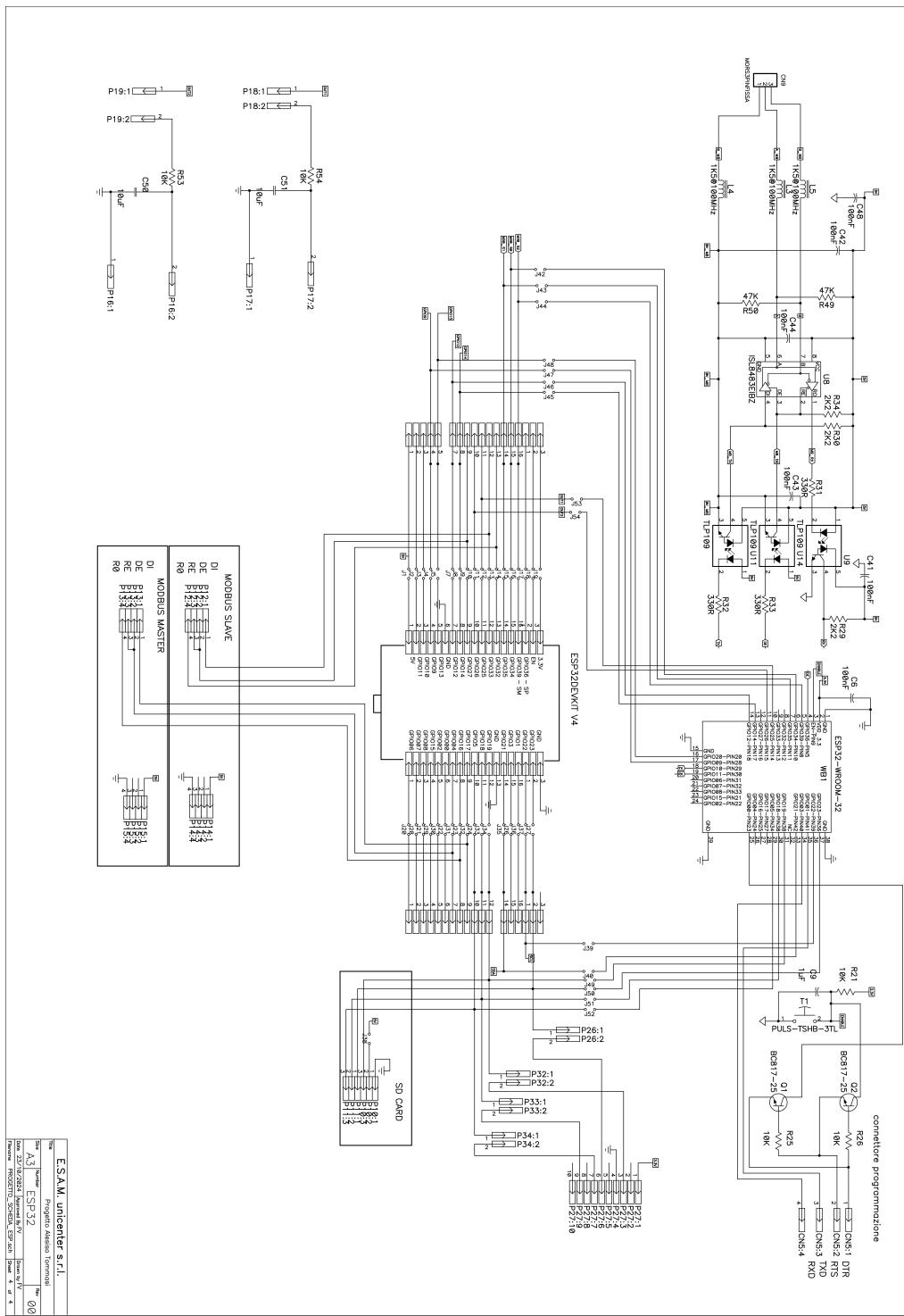


Figura 2.11: collegamenti all'ESP32-DOIT-DEV-KIT v1

Tabella 2.2: Pinout dell'ESP32 (Versione a 38 pin)

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot, must be LOW to enter flashing mode
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED, must be left floating or LOW to enter flashing mode
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot, strapping pin
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fails if pulled high, strapping pin
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot, strapping pin
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

Capitolo 3

Software

3.1 Diagramma UML

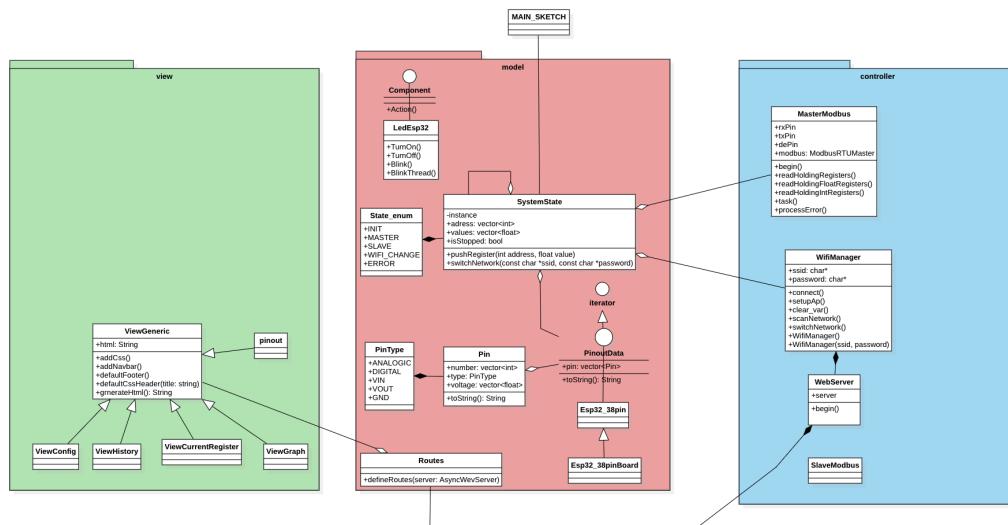


Figura 3.1: Diagramma UML del sistema

3.1.1 Pattern

Model-View-Controller (MVC)

Model Contiene la struttura dei dati e le funzioni per accedere e modificarli. Le classi coinvolte sono **SystemState**, **Pin**, **PinoutData**, **Routes**.

View Si occupa di visualizzare i dati e di interagire con l'utente. principalmente sono classi che si occupano della generazione dei componenti delle pagine web visualizzate dall'utente.

Controller Gestisce le richieste dell'utente e aggiorna il modello di conseguenza.

Singleton

utilizzato per garantire unicità e atomicità dei dati **SystemState** è la classe che implementa tale pattern. Sarà particolarmente utile in futuro per l'implementazione di un sistema di datalogging e persistenza dei dati su scheda SD.

Dallo sketch principale è inoltre possibile definirsi una Board custom estendendo la classe pinout e un wifi di default sulla quale connettersi senza dover passare ogni volta dall'AP dell'ESP per garantirne una migliore usabilità.

3.2 Performance

Il programma caricato ottiene i seguenti risultati dal compilatore integrato ad Arduino IDE:

```
Sketch uses 1291045 bytes (98%) of program storage space.  
Maximum is 1310720 bytes.  
Global variables use 50448 bytes (15%) of dynamic memory,  
leaving 277232 bytes for local variables.  
Maximum is 327680 bytes.
```

Considerazioni: Analizzando tale output si ne deduce che il programma utilizza il 98% dello spazio di memorizzazione disponibile.

Questo elevato utilizzo della memoria flash implica un margine limitato per l'aggiunta di nuove funzionalità o librerie.

L'utilizzo delle variabili globali è pari al 15% della memoria disponibile, lasciando un ampio margine per le variabili locali e altre strutture di dati dinamici.

Questa situazione è favorevole, in quanto garantisce la flessibilità necessaria per gestire strutture dati complesse e consentire allocazioni dinamiche. Dunque, per garantire la stabilità e l'espandibilità del progetto, potrebbe essere utile considerare l'acquisto di una versione dell'ESP32 con una maggiore capacità di memoria flash disponibile in commercio.

Oppure in alternativa, si può optare per la modifica lo schema di partizionamento dell'ESP32 per allocare più memoria al codice del programma, riducendo lo spazio destinato alla memoria Dinamica. Questa personalizzazione permette di adattare la memoria disponibile alle specifiche esigenze del progetto senza introdurre costi aggiuntivi.

Monitor Per un controllo più granulare delle metriche come stack e heap del dispositivo, è stata creata una pagina dedicata che permette il monitoraggio in tempo reale senza impattare sulle prestazioni.

3.3 WebServer

È stato sviluppato un framework estendendo la libreria `AsynkWebsrwerver` per la gestione delle pagine web e delle richieste HTTP. tale framework consente di aggiungere dinamicamente componenti HTML alle pagine in base alle richieste del client.

Questo approccio garantisce una struttura uniforme e un tema coerente che si adatta a tutti i dispositivi.

Il framework è progettato per essere estensibile e modulare. Per aggiungere una nuova pagina, è sufficiente estendere la classe `ViewGeneric` e aggiungere i componenti desiderati. I nuovi componenti erediteranno automaticamente il tema dell'applicativo, assicurando un aspetto e una sensazione coerenti in tutto il sistema.

3.3.1 Componenti HTML Dinamici

Il framework consente di definire componenti HTML come stringhe e di aggiungerli dinamicamente alle pagine web. Questo approccio permette di creare interfacce utente flessibili e reattive.

Esempio di Aggiunta di una Pagina

Per aggiungere una nuova pagina al WebServer, è necessario seguire questi passaggi:

1. Estendere la classe `ViewGeneric`.
2. Definire i componenti HTML desiderati come stringhe.
3. Aggiungere i componenti alla pagina utilizzando i metodi forniti dal framework.

```
class ViewConfig : public ViewGeneric
{
public:
    static String generateHTML(std::vector<std::string> ssidList);
    static String generateHTML();

    static String generateComponents(std::vector<std::string> &networks); ////
};

#include "viewConfig.h"

String viewConfig::html = "";

String viewConfig::generateListComponents( std::vector<std::string> &networks)
{
    /*Example of a component*/
    String html = "<div class='form-container'>";
    html += "<form action='/switch_wifi' method='get'>";
    html += "<label for='network'>Select Network:</label>";
    html += "<select id='ssid' name='ssid' required>";

    for (const std::string &network : networks)
    {
        html += "<option value='" + String(network.c_str()) + "'>" + String(
    }
```

```

        html += "</select>";
        html += "<label for='password'>Password:</label>";
        html += "<input type='password' id='password' name='password' required>";
        html += "<button type='submit'>Connect</button>";
        html += "</form>";
        html += "</div>";

        return html;
    }

String viewConfig::generateHTML(std::vector<std::string> ssidList)
{
    String html = viewGeneric::defaultCssHeader("Config Page");

    html += viewConfig::generateListComponents(ssidList);

    html += viewComponent2::generate();

    ...

    html += viewComponentN::generate();

    html += viewGeneric::defaultFooter();

    return html;
}

```

Routes Una volta definito il componente, è necessario aggiungerlo alle route desiderate per poterlo visualizzare.

Questo può essere fatto in due modi: 1. Estendendo il file ‘Routes.h’ e aggiungendo la nuova route. 2. Aggiungendo la route direttamente nel file ‘Routes.cpp’ come segue:

```

void Routes::addRoutes()
{
    server.on("/new_component_routes", HTTP_GET, [] (AsyncWebServerRequest *r)
    {
        String htmlContent = viewConfig::generateHTML(SystemState::getInstance());
        const char *htmlContentPtr = htmlContent.c_str();
    });
}

```

```

    request->send(200, "text/html", htmlContentPtr);
}

...
}

```

3.3.2 Vantaggi del Framework

- **Uniformità:** Garantisce un aspetto e una sensazione coerenti in tutto il sistema.
- **Modularità:** Facilita l'aggiunta di nuove pagine e componenti senza modificare il codice esistente.

3.3.3 Conclusioni

Il framework per il WebServer rappresenta una soluzione robusta e flessibile per la gestione delle interfacce web. La sua architettura modulare e la capacità di aggiungere dinamicamente componenti HTML rendono il sistema facile da mantenere e espandere, garantendo al contempo un'esperienza utente coerente e reattiva.

3.4 Modbus

Questa funzionalità non riulta attualmente perfettamente funzionante quindi sarà rivista nelle attività future. Sono stati fatti diversi tentativi di ognuno di essi è consultabile al seguente link: Modbus.

3.5 Sfide

Una delle attività più complesse è stata la gestione dell'accesso concorrente alle variabili da parte delle richieste HTTP asincrone.

È stato necessario implementare una logica che garantisse un accesso atomico alle variabili e una sincronizzazione in tempo reale di tali variabili su più dispositivi.

Capitolo 4

Attività

Attività	Descrizione
Monitor	creazione di una pagina web dedicata al monitoraggio delle prestazioni e dello stato del sistema.
Lettura segnali analogici	Implementare la lettura di segnali analogici tramite gli ingressi ADC dell'ESP32
Gestione uscite digitali e analogiche	Sviluppare la gestione delle uscite digitali e analogiche tramite l'ESP32.
Comunicazione RS485 (Modbus RTU)	Integrare la comunicazione RS485 utilizzando il protocollo Modbus RTU per interfacciarsi con altri dispositivi.
Server Web (Ethernet TCP/IP)	Sviluppare un server Web basato su Ethernet TCP/IP per il monitoraggio e controllo remoto dei dati acquisiti.
Test e validazione	Testare e validare il sistema attraverso simulazioni e test su hardware reale.

Capitolo 5

Conclusioni

5.1 Demo

mostrare funzionamento delle funzionalita del sistema durante presentazione

5.2 Sviluppi futuri

persistenza e datalogging su scheda SD Si intende implementare un sistema che una volta salvata una configurazione wifi e una board personalizzata e possibile garantirne la persistenza su scheda SD.

backup dati su cloud implementazione backend con adeguata scalabilita per garantire una migliore elaborazione ed estrapolazione dei dati poiche la board e limitata in termini di memoria e capacita di calcolo.

personalizzazione implementazione di un sistema di modifica e dello stato dei pin e dei canali multiplexer direttamente da apposite web.

Implementazione di un sistema di notifiche Sviluppare un sistema di notifiche per avvisare l'utente quando i valori registrati sullo stack superano una soglia limite definita dall'utente. Inoltre, notificare la presenza o l'assenza di connessione internet.

Modbus implementazione corretta Modbus TCP/IP attraverso dispositivo Winzet W5500.