EQUA SMART BOX

- **UNIVERSITA:**

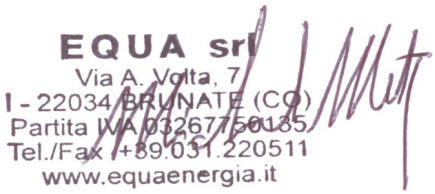
  POLITECNICO DI MILANO

- **Studente:**

Erick Bryan Julon Ramirez

- **Tutore Aziendale:**

Ing. Michael Metzger

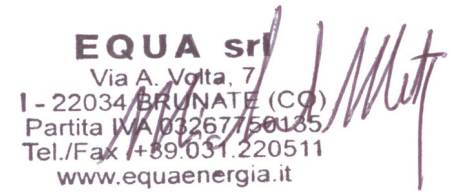


- **Tutor Universitario:**

Ing. Luca Ferrarini

- **Azienda:**

EQUA srl



- **Progetto:**

Equa Smart Box

INDICE Progetto EQUA SMART BOX

FUNZIONAMENTO

Obiettivo del progetto:

Il progetto prevede un sistema di **monitoraggio e controllo intelligente** di un impianto di riscaldamento tipicamente alimentato da una pompa di calore, che a sua volta può essere alimentata da pannelli solari.
Il nome del progetto è **EQUA SMART BOX**.

Pompa di calore:

Il funzionamento della pompa di calore è a relativamente semplice e si basa sul prelievo di energia termica dall'ambiente esterno, tipicamente dall'aria o da acqua di falda, l'utilizzo di detta energia termica è per portare a temperatura utile un fluido, tipicamente acqua, e **trasferire il calore nelle unità immobiliari.**

Il vantaggio rispetto alle tecnologie tradizionali è di avere disponibile un serbatoio di energia termica a bassa temperatura, l'ambiente esterno, praticamente illimitato e gratuito permettendo di abbassare in modo significativi sia costi che impatto ambientale.

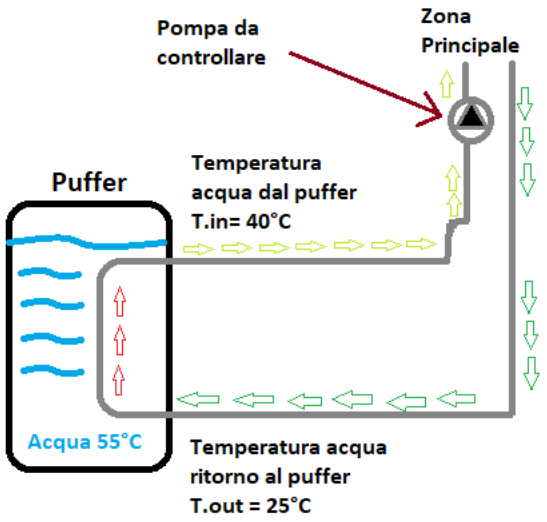
il riscaldamento viene effettuato attraverso la circolazione dell'acqua riscaldata che a partire dal serbatoio a temperature più alta, viene distribuita nei vari ambienti dove opportuni sistemi radianti, dai classici termosifoni ai più attuali sistemi a pavimento, provvedono a trasferire il calore negli ambienti.

Abbiamo due casi, un caso con un'unica pompa di circolazione e uno con un collettore.

Primo caso:

Nel caso più semplice un'unica pompa di circolazione porta l'acqua in tutto l'edificio e di norma richiede un unico controllo di temperatura per tutta la casa.

Immagine Indicativa e sintetica del funzionamento: Senza Collettore

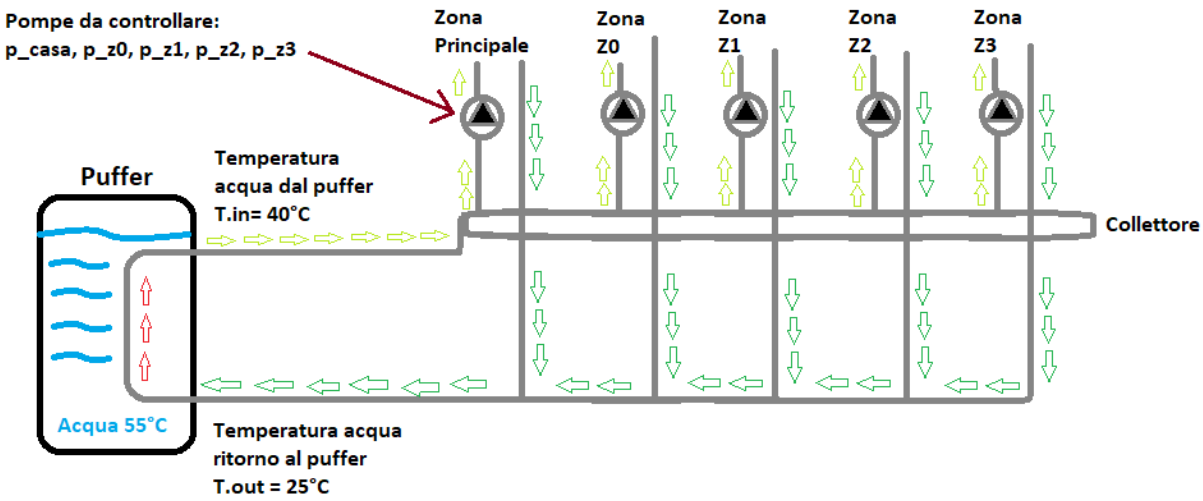


Secondo caso:

Nel caso in esame si preferisce di riscaldare in modo controllato diversi ambienti e quindi risulta necessario gestire un insieme di sensori e sistemi di controllo multipli. In questo caso si usa un collettore collegato ad un attuatore (pompa dedicata o elettrovalvola) che permette di avere un controllo indipendente per ogni ambiente.

La soluzione implementata prevede il controllo fino a 5 diverse zone.

Immagine Indicativa e sintetica del funzionamento: Con Collettore



Cosa può fare adesso il progetto

L'implementazione attuale, è collaudata a mezzo di breadboard in un ambiente con attuatori simulati.

Prevede un **monitoraggio delle temperature e umidita** continuo. Le misure sono visualizzate su una interfaccia utente basata su un display touch screen che permette di **controllare il funzionamento dei singoli attuatori (pompe di circolazione).**

Ogni pompa alimenta una zona, e risulta possibile attivare le singole zone e programmare la temperatura desiderata **(temperatura di setpoint**) per ogni zona.
I loop di controllo prevedono la modulazione della richiesta di riscaldamento nel momento in cui la temperatura misurata raggiunge il relativo set point.

Schema Sintetico ed Indicativo del progetto attuale:

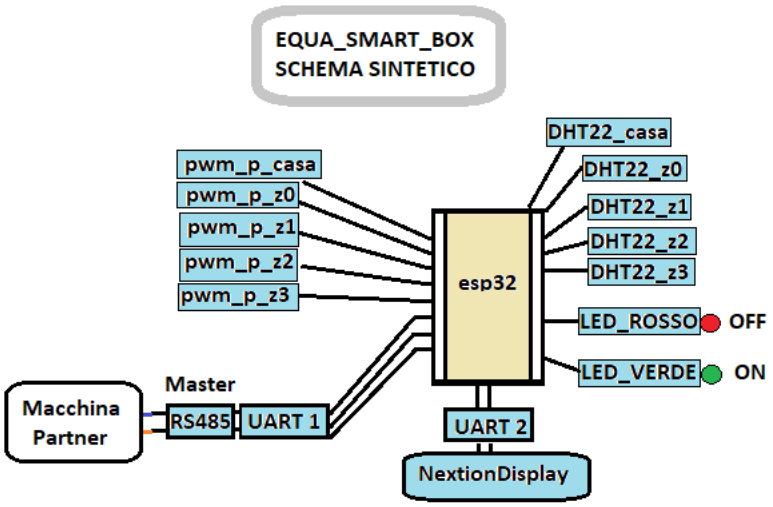
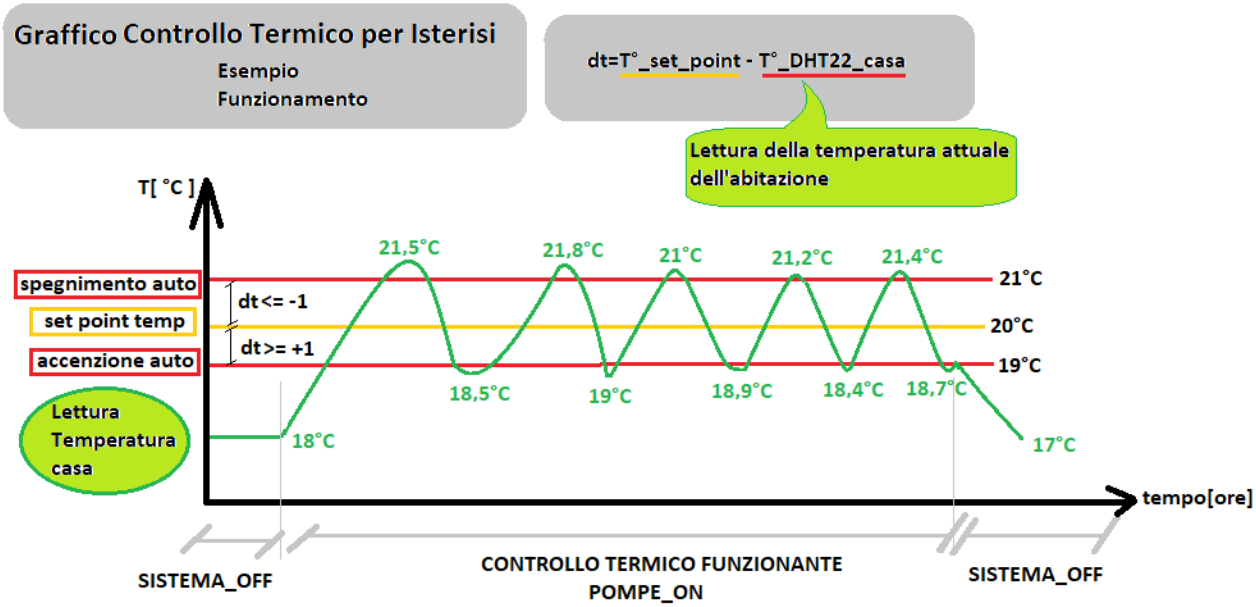


Grafico del controllo termico che effettua l'EQUA SMART BOX



Upgrade & +

Per garantire l'espandibilità del sistema e la possibilità di interfacciarsi con diversi sistemi commerciali è stata prevista l'integrazione di un interfaccia RS485 da utilizzare in ambiente Modbus, ambiente industriale largamente utilizzato nelle implementazioni di sistemi di controllo distribuiti.

Dal momento che questo deve essere un prodotto finale si deve tener conto dell'impatto economico. Non si possono aggiungere troppe board (altre schede) a meno che non siano proprio necessarie, questo comporta più lavoro di ricerca e fantasia per trovare delle soluzioni alternative.

Nel processo tra ricerca di soluzioni software alternative e comparse di nuove problematiche, sono arrivato a progettare una versione **più flessibile** del progetto. Lasciando di lato l'aspetto economico.

Perché molto flessibile? Ho notato che il mondo industriale ha molti protocolli di comunicazioni. Per evitare di modificare l'hardware o software per ogni nuovo protocollo di comunicazione che ci troveremmo in futuro ho pensato a sfruttare la gran versatilità che ha il microcontrollore in questione. L'**esp32** è in grado di comunicare quasi con tutti.

PROBLEMATICHE TROVATE NEL PROGETTO ATTUALE: (UN UNICO CORPO OVVERO UN SOLO ESP32)

- **Numero di PIN limitato da utilizzare**
- **Numero limitato di interfacce di comunicazione seriale nel caso si renda necessario implementare più funzioni (Modbus Master, Modbus Slave , Mbus)** per mettere in comunicazione sistemi d produttori diversi. L'esp32 ha solo 3 porte. Chiamati UART.

Porte di comunicazione seriale : le interfacce Modbus e Mbus utilizzate per la comunicazione con sistemi industriali e sistemi di misura utilizzano protocolli di comunicazione seriale
ESP32 ha 3 UART hardware :

UART 0: Utilizzata di massima **SOLO** per la comunicazione con dall'ambiente Arduino per il caricamento dei programmi e per il monitoraggio di debug.

UART 1: Pin digitali, quindi utilizzare altri 2 pin fisici **(USATO PER IL RS485) che può esser utilizzata previa opportuna mappatura dei pin per una interfaccia seriale ad esempio per comunicazione verso Mbus.**

UART 2: Utilizzata con i pin di default 17 e 16 (TX e RX rispettivamente) **(USATO PER IL DISPLAY) per la comunicazione con il touch screen display NEXTION**

- **Memoria utilizzata**, un unico microcontrollore elabora e contiene tutti i dati e variabili che servono per il controllo termico.
- **Elasticità del dispositivo**, dal momento che tutto è concentrato in un unico corpo ogni volta che si deve aggiungere, togliere o modificare qualcosa si perde molto tempo perché ci sono tanti parametri e considerazioni da fare.
- **Distanza limitata** dei sensori DHT22 **a 20 metri,** altrimenti si perde il segnale o peggio ancora ci dà letture errate.

SOLUZIONE

Dal momento che **ci sono tanti parametri da cui tenere conto e allo stesso tempo la necessità di tenere il controllo in unico corpo**. Ho pensato a questa configurazione. Grazie alla comunicazione MODBUS RS485 si può acquisire dati da tutti gli SLAVE che abbiamo con solo 2 fili. **Allo stesso tempo ogni slave ha una propria autonomia quanto il corpo principale**. E soprattutto i moduli RS485 possono trasmettere il segnale senza perdita fino a 1500 metri. Quindi, invece di lavorare con un unico corpo, si fa una delega dei compiti agli slavi e il corpo principale si dedica solo a ricevere, inviare, monitorare e controllare l'intero sistema.

Pro:

- **NON c'è più il problema dei PIN**, il numero di PIN aumenta di 6 volte. E abbiamo l'opzione di creare una mini domotica locale **indipendente** per ogni zona, quindi senza complicare il codice principale.
- L'**UART 2** viene utilizzato per la comunicazione MODBUS RS485 ma abbiamo l'**UART 1** di ogni slave pronto all'uso. In questo caso 6 UART a disposizione per altri protocolli di comunicazione tali come **MODBUS**, **M-BUS**, **CAN-BUS** ecc.
- Dal momento che dividiamo i compiti, **la memoria utilizzata viene spartita tra tutti i microcontrollori**. Non avremmo più il problema della memoria.
- **L'elasticità sarà il suo punto forte**. Con questa configurazione, abbiamo molti più microcontrollori da programmare ma dal momento che ognuno deve fare un compito specifico è molto più semplice la codifica in confronto a programmare **molte azioni in un unico corpo**. L'aggiunta di nuove macchine con cui comunicare o creare una mini domotica su richiesta dal cliente, diventa più semplice. Non importa dove si trova la macchina con cui comunicare, avremmo per ogni slave 1500 metri da utilizzare per arrivarci. E non importa quanto sia complessa la domotica richiesta, questa **sarà indipendente dal funzionamento principale** (Dal Master) che si occupa del controllo della pompa di calore.
- La distanza dei sensori di temperatura non saranno più un problema, **avremmo 1500 metri da utilizzare dal master verso ogni slave, e altri 20 metri da utilizzare dagli slave verso i sensori dht22**.

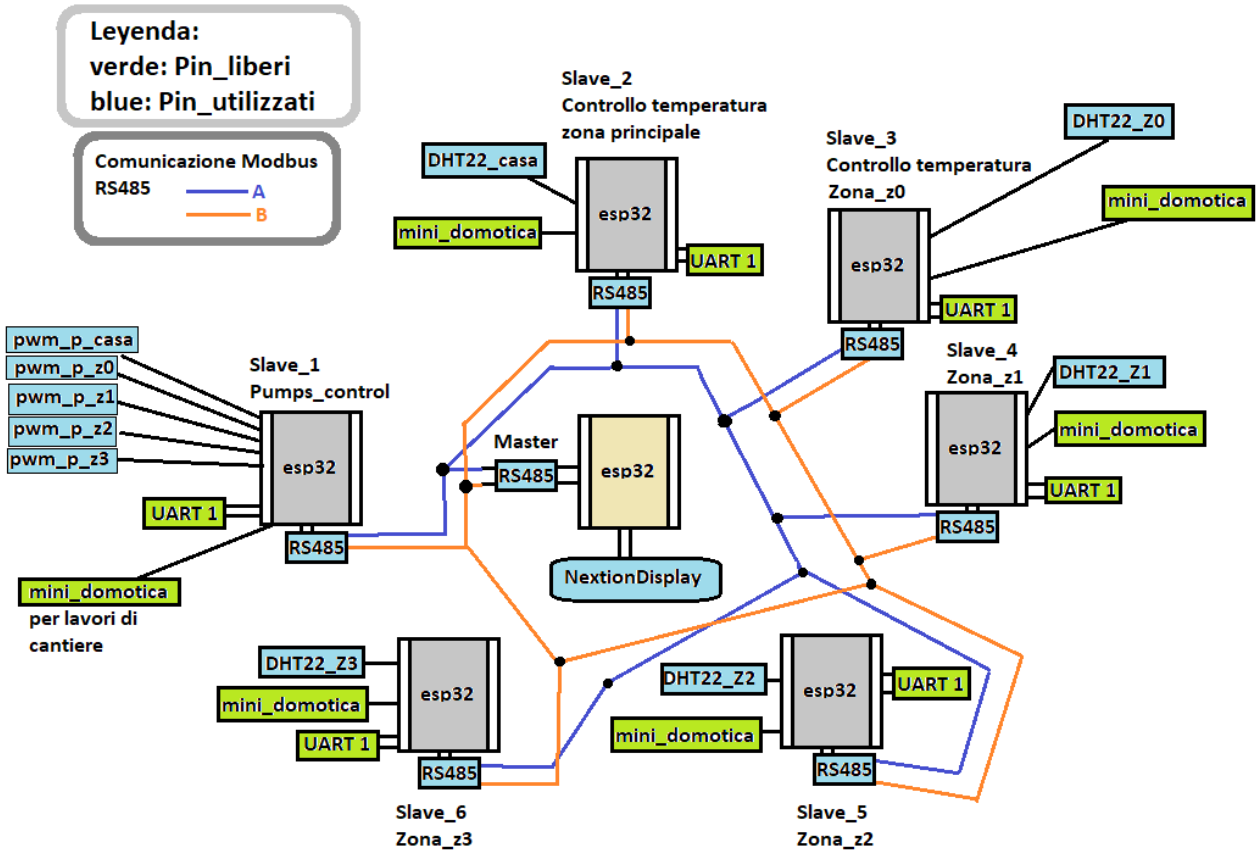
Contro:

- **Il prezzo** del prodotto finale **aumenta almeno di 6 volte**, dal momento che dovremmo prendere altri 6 ESP32 e altri 6 moduli RS485.
- Essendo molto decentrato lo svolgimento dei compiti, questo facilita la programmazione ma **aumenta enormemente la probabilità di avere dei guasti e quindi dei malfunzionamenti del controllo** della pompa di calore.
- **In caso di guasto fisico**, sarà semplice trovare il componente che non funziona. Ma in caso di **errore "digitale" , ovvero di codice o malfunzionamento a causa di tempeste o situazioni impreviste** sarà un po' più complesso arrivare alla radice del problema.

Conclusione:

Come visto, questa configurazione ha dei vantaggi e degli svantaggi. Dal mio punto di vista, prenderei questa strada. Ci permette di avere stabilita (dal punto di vista hardware) a lungo termine e di migliorare il software continuamente senza troppe complicazioni. Ci sarà il rischio che possa avere degli errori (come con qualunque altra configurazione), l'obiettivo è di migliorare e renderlo sempre più affidabile.

Schema Upgrade



Upgrade +

Puramente teorico:

Un paso più avanti, per abbassare i costi e ridurre i cablaggi a quasi zero. Invece di usare una **comunicazione ModBus rs485 master-slave** come cannale di comunicazione, si può utilizzare una rete **wireless Master-slave**. La probabilità dei guasti e malfunzionamenti diventano ancora più alti però. E quindi ancora meno realizzabile.

HARDWARE

DISPOSITIVI UTILIZZATI + MOTIVAZIONI + CARATTERISTICHE E DATI TECNICI

Sono stati utilizzati:

- Un NextionDisplay (5v) , modello: NX4827K043 da 4.3”
- Un microcontrollore ESP32, modello: ESPRESSIF ESP-WROOM-32D
- 8 RELAY MODULE, modello: SRD-05VDC-SL-C
- STARTER KIT ARDUINO
- 5 Sensori di temperatura DHT22, modello: AM2302
- 3 Breadboards
- 1 Modulo ModBus TTL RS485 (5v), modello: MAX485
- 1 TTL Bidirectional Logic Level Converter 3v3 - 5v

Il NextionDisplay è stato scelto grazie alle caratteristiche smart e flessibili che ha. Possiede un microcontrollore, una memoria flash, uno slot per l'inserimento di una SD card contenente il firmware, una interfaccia TTL a 4 pin e un software gratuito (Nextion Editor) realizzato appositamente per disegnare la parte grafica (pulsanti, barre, immagini, testo ecc). E' ideale per creare applicazioni di **monitoraggio e controllo di processi**, che è esattamente l'obiettivo della Equa Smart Box. Soprattutto è in grado di comunicare con altri microcontrollori. Alimentazione 5v.

L'ESP32 è stato scelto grazie all'enorme capacità che ha di adeguarsi a tanti protocolli di comunicazioni e la versatilità dei suoi pin. Ogni pin è in grado di svolgere diversi compiti, hanno diverse proprietà. Può essere programmato con l'ambiente di arduino (**IDE Arduino**) **io ho utilizzato linguaggio c**, oppure con l'ambiente Eclipse (IDE Eclipse) che usa MicroPython.

8 RELAY MODULE, sono stati scelti per il controllo di accezione e spegnimento di dispositivi a 220v , come ad esempio le pompe circolatore o dei fan coil. Alimentazione 5v.

Sensori di Temperature e umidità DHT22, hanno un margine di errore molto basso +/- 0,5°C. Usano un solo filo per la lettura ed invio di datti. Alimentazione 3v3 o 5v.

Starter Kit Arduino, abbiamo preso resistenze, fili, led e altri strumenti presenti per le prove sulle breadboard.

Modulo ModBus RS485, l'abbiamo scelto perché è molto diffuso nel mondo industriale. Al progetto serve per acquisire i dati da altri apparecchi e macchine con cui dovrà comunicare. Alimentazione 5v.

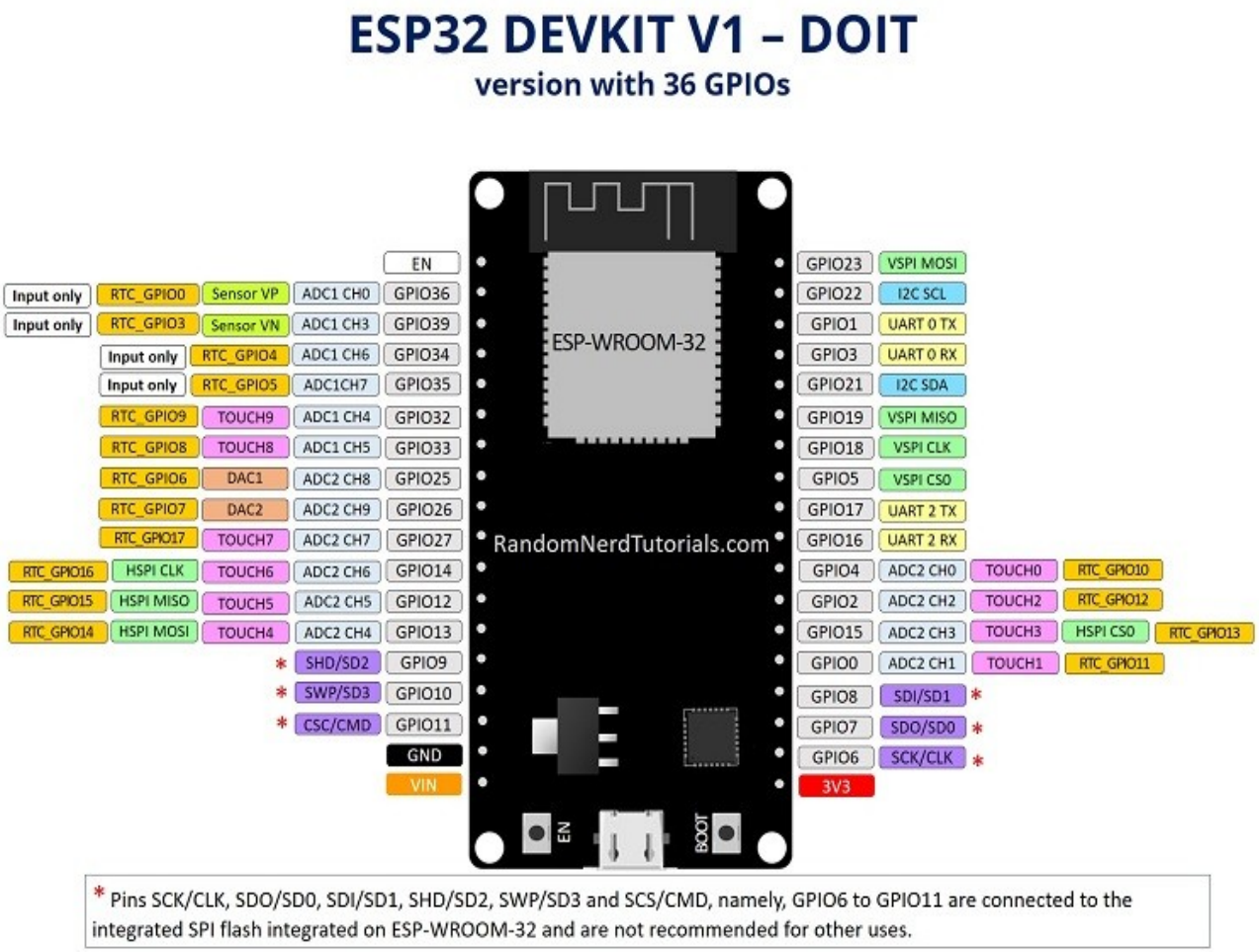
Modulo TTL Bidirectional Logic Level Converter 3v3 - 5v, serve per convertire l'alimentazione 3v3 dell'esp32 a 5v. Serve per l'alimentazione del modulo MODBUS RS485 a 5v

PINSET_ESP32

Oss:

Ogni pin è in grado di fare più di una cosa, in base all'esigenze e alle problematiche da risolvere si decide l'utilizzo di ogni pin (chiamati anche GPIO)

Immagine presa da internet



PIN UTILIZZATI

Oss:

tutti i GPIO dove si trova un asterisco e il fondo di colore della descrizione è bianco, sono i pin non utilizzati per diversi motivi. Alcuni non possono essere proprio utilizzati come i pin collegati alla memoria flash del microcontrollore. Oppure i **pin 39_36_34_35** che possono essere utilizzati **solo come INPUT**. Ci sono diverse considerazioni al momento di scegliere i pin. Infatti in una prima prova ho usato questi pin **(39,36,34,35)** per i sensori dht22 di temperatura e non riuscivo a capire l'errore finché non ho cambiato i pin. Quindi non vanno messi a caso.

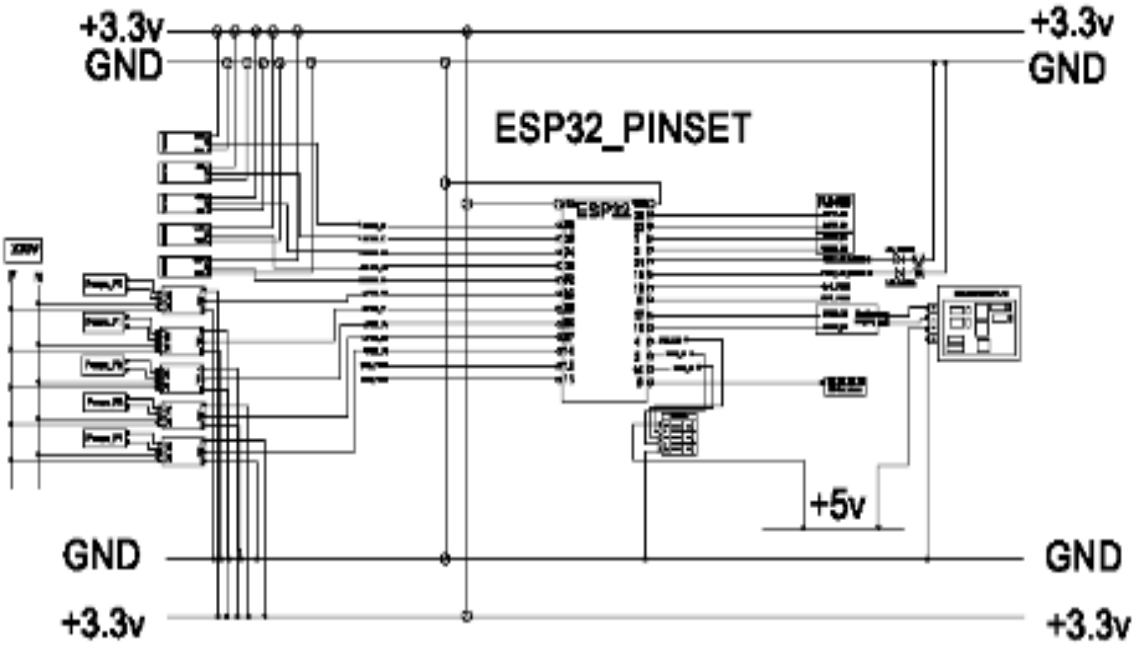
PIN USATI EQUA SMART BOX_V0.0.2			ven, 8 gen2020	
DESCRIZIONE	PIN	ESP32	PIN	DESCRIZIONE
*ONLY_INPUT	GPIO_36		GPIO_23	UART 1 TX (MB_RX_RS485)
*ONLY_INPUT	GPIO_39		GPIO_22	UART 1 RX (MB_TX_RS485)
*ONLY_INPUT	GPIO_34		GPIO_1	UART 0 TX (ONLY_PROG)
*ONLY_INPUT	GPIO_35		GPIO_3	UART 0 RX (ONLY_PROG)
DHT22_Z1	GPIO_32		GPIO_21	(MB_EN_TX_RX_RS485)
PWM_CASA	GPIO_33		GPIO_19	PWM_LED_VERDE
PWM_Z0	GPIO_25		GPIO_18	PWM_LED_ROSSO
PWM_Z1	GPIO_26		GPIO_5	DHT22_Z2
PWM_Z2	GPIO_27		GPIO_17	UART 2 TX (NEXTIONDISPLAY)
PWM_Z3	GPIO_14		GPIO_16	UART 2 RX (NEXTIONDISPLAY)
DHT22_Z0	GPIO_12		GPIO_4	DHT22_Z3
DHT22_CASA	GPIO_13		GPIO_2	VUOTO
*SD (NO UTIL)	GPIO_9		GPIO_15	VUOTO
*SD (NO UTIL)	GPIO_10		GPIO_0	*CLOCK OUT 1
*SD (NO UTIL)	GPIO_11		GPIO_8	*SD (NON UTILIZABILE)
	GND		GPIO_7	*SD (NON UTILIZABILE)
	VIN		GPIO_6	*SD (NON UTILIZABILE)
		USB_PORT	3V3	

SCHEMA_ELETTRICO_INTERO_SISTEMA: ESP32 + NEXTIONDISPLAY + BOARDS

L'esp32 eroga un'alimentazione di 3v3 , se possibile non utilizzare il Vin perché può danneggiare il microcontrollore con tensioni superiori a 3v3. Per le alimentazioni del display e alcuni board di 5v si usa un'alimentazione esterna all'esp32 o un convertitore da 3v3 a 5v.

Questo è lo schema finale (fino dove si è arrivato), è già stato verificato il funzionamento su breadboard di **microcontrollore + sensori + nextionDisplay + rispettivi codici**. Paso successivo, utilizzo di un millefori e attrezzatura per saldare in modo tale di poter avere un dispositivo compatto e cominciare delle prove su impianti veri.

- **Lo schema è stato fatto su AutoCAD**
- **Doppio clic sull'immagine per vederlo meglio.**



MODULO RS485

È stato utilizzato per acquisire informazione via comunicazione Modbus RS485 da altre macchine con cui dovremmo comunicare. È una comunicazione duplex, riesce a raggiungere i 1500metri senza perdere il segnale ed è molto diffuso nel mondo industriale.

La comunicazione ModBus ci permette di avere una **configurazione MASTER-SLAVE**, fino un massimo di 255 slaves.

I pin DI e RO sono collegati ai pin TX e RX dell'esp32, mentre i due DE e RE vanno cortocircuitati e collegati ad un pin del microcontrollore (questo sarebbe il pin EN_enable).

E' stato utilizzato per acquisire informazione via comunicazione Modbus RS485 da altre macchine con cui dovremmo comunicare. E' una comunicazione duplex, riesce a raggiungere i 1500metri senza perdere il segnale ed è molto diffuso nel mondo industriale.

La comunicazione ModBus ci permette di avere una **configurazione MASTER-SLAVE**, fino un massimo di 255 slaves.

I pin DI e RO sono collegati ai pin TX e RX dell'esp32, mentre i due DE e RE vanno cortocircuitati e collegati ad un pin del microcontrollore (questo sarebbe il pin EN_enable).

il Master manda una richiesta che contiene l'indirizzo dello SLAVE che risponde:
Il protocollo Modbus è di tipo Master/Slave e quindi nella rete è presente sempre e solo un dispositivo Master che gestisce la comunicazione nei confronti di uno o più dispositivi Slave. Ogni scambio di informazioni è originato dal Master il quale invia un frame di bytes sul bus di campo contente una particolare richiesta, normalmente un comando di lettura o di scrittura delle informazioni contenute in uno degli Slave. Tutti gli Slave sono normalmente in ricezione ed ascoltano le richieste del Master. Solo lo specifico Slave interrogato cattura le informazioni inviate dal Master, provvede all'esecuzione del comando e risponde al Master inviando a sua volta le proprie informazioni sulla rete.

I pin A e B sono i collegamenti, vanno collegati rispettivamente ai pin A e B dello slavo da controllare o da cui acquisire dati.

- **Lo schema è stato fatto su AutoCAD**
- **Doppio clic sull'immagine per vederlo meglio.**

RS485_PINSET

MODULO 8RELAY_BOARD:

Con questi moduli siamo in grado di creare un controllo ON/OFF con o senza timer , di apparecchi esterni e se si vuole anche a controllare la direzione di un motore c.c./ c.a.

L'inserimento di questi moduli erano inizialmente previsti per gestire l'alimentazione delle pompe e mediante segnali PWM controllare il passaggio di corrente e quindi la velocita. Successivamente abbiamo trovato dei moduli pwm con delle protezioni che mantengono anche loro isolati il microcontrollore dalle tensioni 220v. Abbiamo abbinato comunque 3 relay per utilizzi futuri a richiesta del cliente. (Come ad esempio controllare dall'Equa Smart Box dei fan coil o un piccolo impianto domotico, ecc)

Collegamenti:

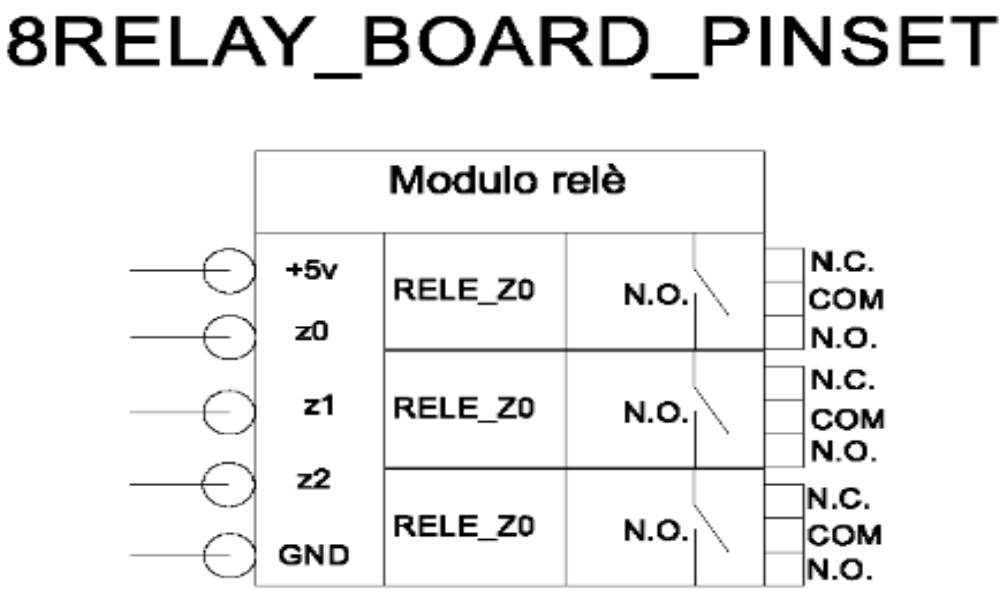
Dal lato sinistro dove si trova l'alimentazione troviamo dei pin z0,z1,z2. Questi sono i pin che azioneranno ognuno il proprio relay.

Dal lato destro troviamo i pin N.C. (Normalmente Chiuso), COM (Comune) , N.O (Normalmente Aperto). Prendono le iniziali dai rispettivi nomi in inglese. Questi sono i pin dove dobbiamo collegare l'alimentazione dei dispositivi che vogliamo controllare.

Nel nostro caso abbiamo utilizzato **N.O.** e **COM**, **quindi un circuito aperto**. Sono come degli interruttori. In questo caso l'alimentazione non avviene finché non mandiamo il segnale di attivazione (**ad esempio al pin z0 mandiamo un segnale 1**) e quindi il circuito si chiude che

permette il passaggio di corrente e quindi l'accezione della macchina di interesse che stiamo controllando.

- **Doppio clic sull'immagine per vederlo meglio.**
- **Oss: In questo esempio ci sono solo 3 relay, il board viene con 8 relay.**



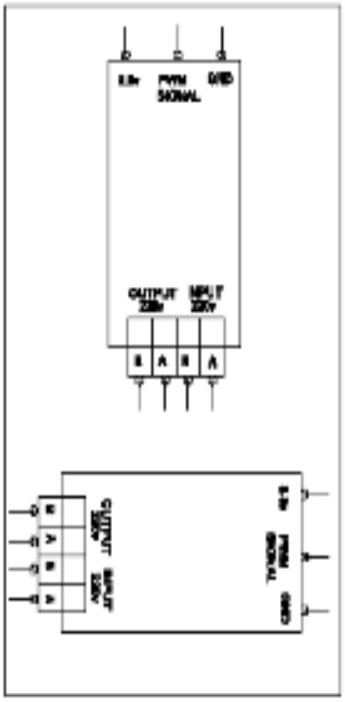
MODULO_PWM

Questi moduli sono stati utilizzate per il controllo dei segnali pwm che vanno collegati alle pompe circolatore da controllare. L'utilizzo di questo board ci dà una protezione aggiuntiva isolando le elevate tensioni dall'esp32.

In alternativa si possono costruire con un semplice kit di arduino, dove ti trovi un mosfet e un relè.

- **Lo schema è stato fatto su AutoCAD**
- **Doppio clic sull'immagine per vederlo meglio.**

PWM_BOARD_PINSET

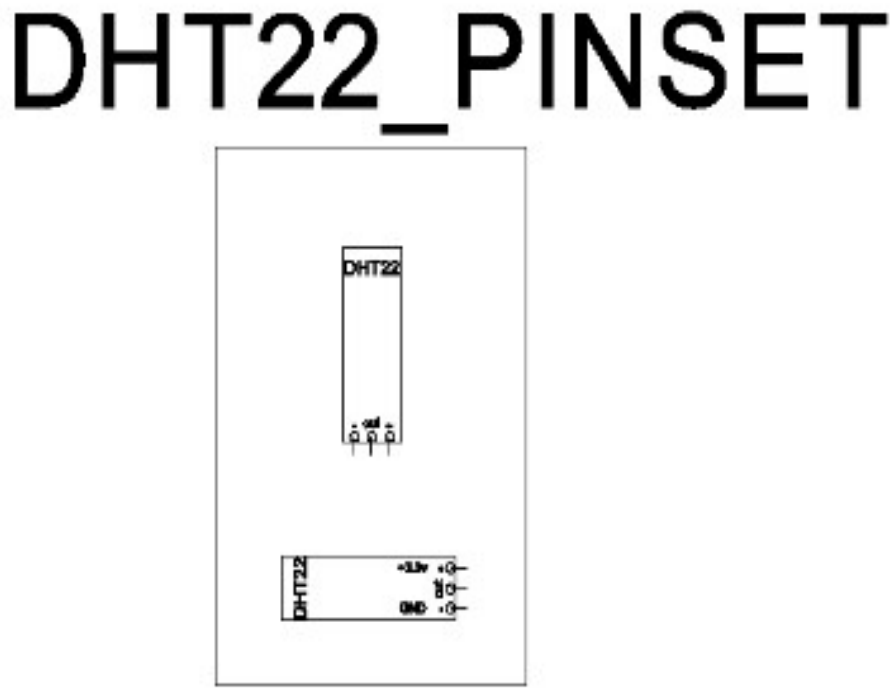


SENSORE DHT22

Sensore utilizzato per la lettura della temperatura e umidità. Questo sensore ci permette una lettura senza predita del segnale fino a 20 metri di distanza, con un margine di errore di +/- 0,5°C.

Collegamenti: Usa un unico pin per la lettura ed invio dei dati. Tenere conto che nel momento della programmazione per un corretto funzionamento dei sensori, questi devono avere il tempo per fare la lettura e inviare i dati al microcontrollore. Altrimenti ti troverai errori di lettura.

- **Lo schema è stato fatto su AutoCAD**
- **Doppio clic sull'immagine per vederlo meglio.**

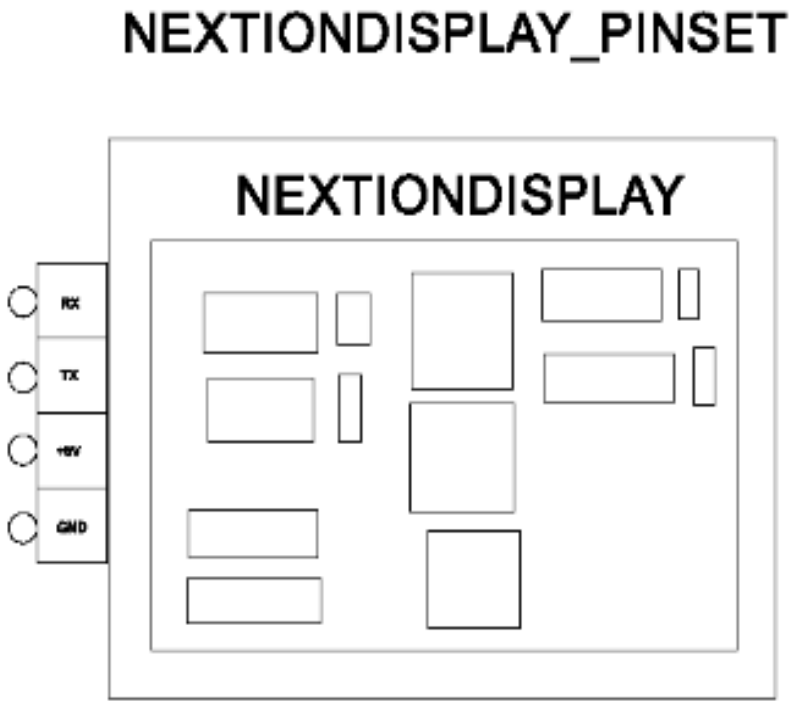


NEXTION_DISPLAY_(SCHEMA INDICATIVO)

Il display scelto è stato il NextionDisplay, molto versatile e diffuso grazie alle sue qualità, elasticità e facilita per la propria programmazione. Segue una programmazione ad oggetti, semplice da imparare. Comunica via seriale, e viene alimentato a 5v.

Nonostante sia possibile programmare sul NextionDisplay ho scelto di limitarmi a programmare solo bottoni e tuto ciò che coinvolge **l'interazione dell'utente con l'impianto**. Ho lasciato tutta l'elaborazione dei dati al microcontrollore per evitare che se per qualche motivo il NextionDisplay si rompe o non funziona più, il sistema in questione da controllare rimarrà funzionante. Per il cliente non si potrà più comunicare con il sistema finché non si cambierà il display. Ma con un computer sul posto o via internet sarà possibile, per i tecnici, comunicare con l'impianto e avere una soluzione temporanea finché non si fa il ricambio del display.

- **Lo schema è stato fatto su AutoCAD**
- **Doppio clic sull'immagine per vederlo meglio.**



INTERFACCIA FINALE NEXTIONDISPLAY + ISTRUZIONI PER L'UTENTE FINALE

Le seguenti immagini è l'interfaccia finale programmata nel NextionDisplay. Il processo per la creazione di ogni oggetto con il NextionEditor è relativamente semplice. Il renderlo però semplice, piacevole alla vista e amichevole per un utente finale inesperto, con il ragionamento dietro che dovrò anche comunicare con il microcontrollore non dipende dal NextionEditor ma del programmatore.

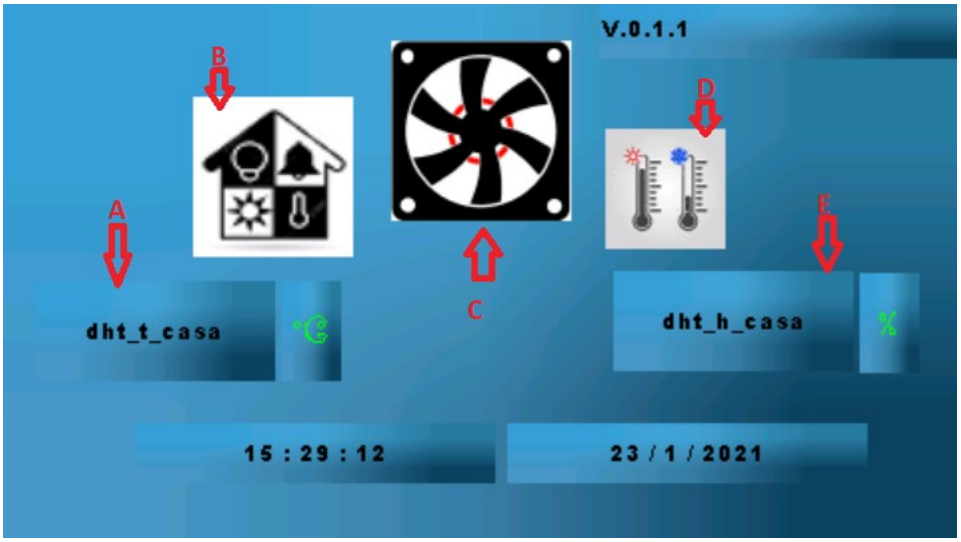
In allegato lascio il programma in formatto **.HMI** del NextionDisplay. E' possibile fare una simulazione scaricando il NextionEditor.

Breve descrizione dell'interazione **Display-Utente**

HOME_PAGE_pg0 (ESEMPIO PULSANTE START OFF)

- **A: Oggetto_Testo, qua trovi la lettura di temperatura finale elaborata senza errori dal sensore dht22_casa**
- **B: Ogetto_Bottone, pulsante Control_Zone. Ti invia alla pagina di controllo delle 4 zone.**
- **C: Ogetto_DualStateButton, pulsante Start (ROSSO => OFF) per il funzionamento_automatico della zona principale (sala o cucina).**
- **D: Ogetto_Bottone, pulsante SetPoint di Temperatura. Mi porta alla pagina control_Setpoint della zona principale.**

- **E: Oggetto_Testo, qua trovi la lettura di Umidità finale elaborata senza errori dal sensore dht22_casa.**



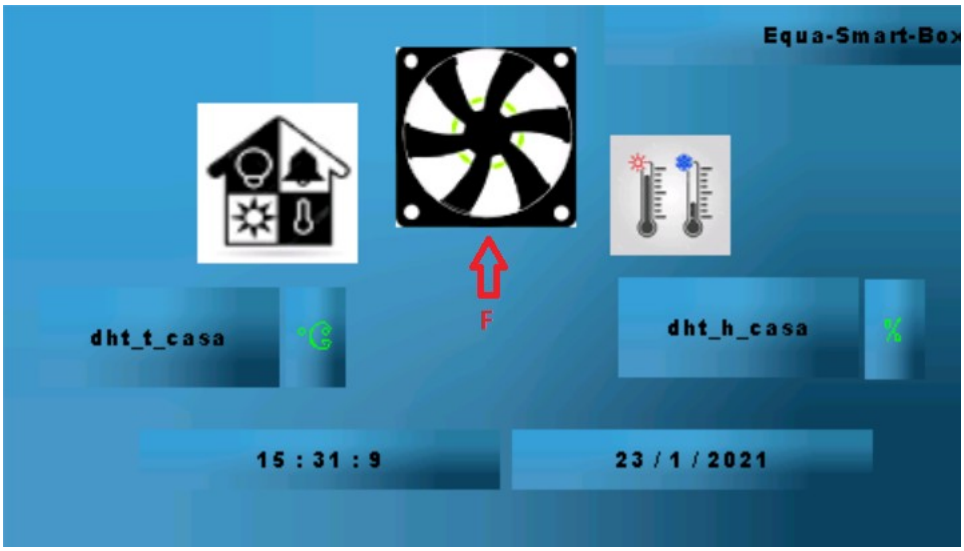
Oss:

Questa è la home_page nel caso il bottone START non è stato premuto. Questo è di colore ROSSO se non è in funzionamento e una volta schiacciato diventa VERDE come indicazione visiva che si è accesso il riscaldamento.

Successivamente se si vuole accendere anche le zone , si schiaccia il bottone che mi porta alla pagina delle zone e accendo le zone di cui ho bisogno che siano riscaldate. Ognuno è indipendente, ognuno ha un controllo termico proprio.

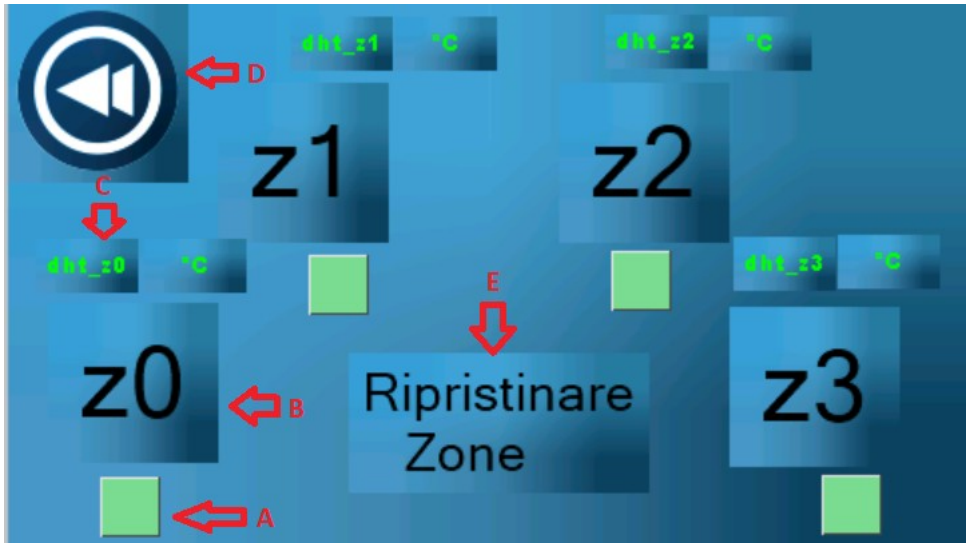
(ESEMPIO PULSANTE START ON)

- **F: Ogetto_Bottone, pulsante Start (VERDE => ON) per il funzionamento_automatico della zona principale (sala o cucina).**



ZONE_PAGE_pg2 (ESEMPIO ZONE TUTTE OFF)

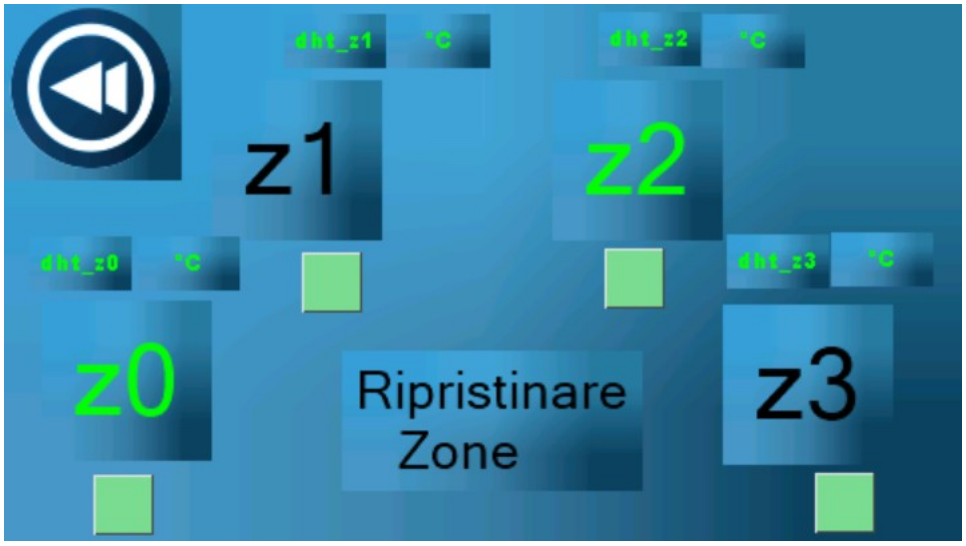
- **A: Ogetto_Bottone, mi porta al SetPoint_Control_Page della zona Z0**
- **B: Ogetto_DualStateButton, pulsante per l'accenzione del riscaldamento automatico della Zona Z0**
- **C: Ogetto_Testo, segnala la Temperatura attuale della zona Z0.**
- **D: Ogetto_Bottone, mi riporta alla HOME_PAGE**
- **E: Ogetto_Bottone, mi spegne tutte le zone.**



Oss:

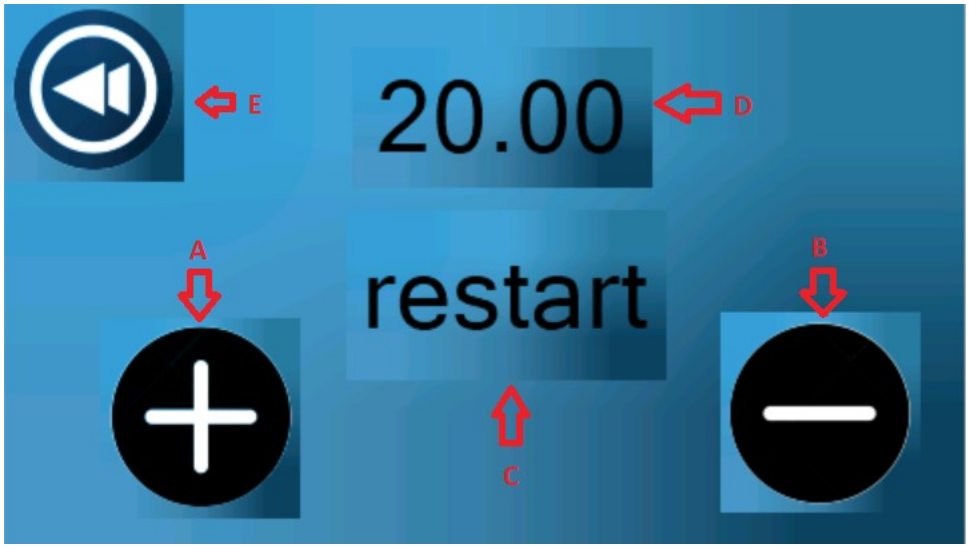
Attenzione che i bottoni Z0,Z1,Z2,Z3 servono per capire quale zone sono accese. Ma le letture di temperature avvengono dal momento in cui si accende l'Equa Smart Box. I bottoni servono per dire al microcontrollore di far partire il funzionamento automatico delle pompe. Il bottone più piccolo di colore verde mi porta alla pagina per il controllo del **SetPoint** (**temperatura desiderata**) di temperatura della zona interessata. Per default tutti i SetPoint di tutte le zone sono inizializzati con una temperatura = 20°C.

(ESEMPIO ZONE Z1 e Z3 -> OFF , Z0 e Z2 -> ON)



CONTROL_SPT_PAGE_pg1 (ESEMPIO CONTROLLO SET_POINT_TEMPERATURA ZONA_PRINCIPALE)

- **A: Ogetto_Bottone, aumenta di 1°C il SetPoint di Temperatura della zona principale.**
- **B: Ogetto_Bottone, diminuisce di 1°C il SetPoint di Temperatura della zona principale.**
- **C: Ogetto_Bottone, riporta il SetPoint di Temperatura a 20.00°C .**
- **D: Ogetto_Testo, mi fa vedere il SetPoint di Temperatura attuale della zona principale.**
- **E: Ogetto_Bottone, mi riporta alla HOME_PAGE**

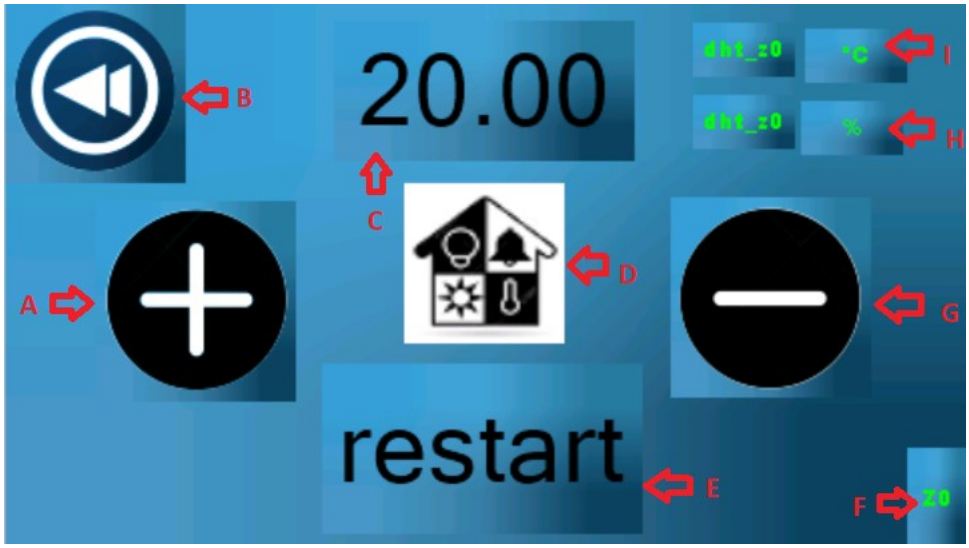


Oss:

Questa è la pagina del controllo del SPT della zona principale. Allo stesso modo tutte le zone hanno una pagina apposta per il controllo del proprio controllo di SPT. Questi valori vengono aggiornati continuamente per il controllo di termico.

CONTROL_SPT_Z0_PAGE_pg3 (ESEMPIO CONTROLLO SET_POINT_TEMPERATURA zona_Z0)

- **A: Ogetto_Bottone, aumenta di 1°C il SetPoint di Temperatura della zona principale.**
- **B: Ogetto_Bottone, mi riporta alla HOME_PAGE**
- **C: Ogetto_Testo, mi fa vedere il SetPoint di Temperatura attuale della zona Z0.**
- **D: Ogetto_Bottone, mi riporta alla ZONE_PAGE, dove si trovano tutte le zone.**
- **E: Ogetto_Bottone, riporta il SetPoint di Temperatura a 20.00°C .**
- **F: Ogetto_Testo, mi fa vedere la zona in cui mi trovo, in questo esempio Z0.**
- **G: Ogetto_Bottone, diminuisce di 1°C il SetPoint di Temperatura della zona Z0.**
- **H: Ogetto_Testo, mi fa vedere l'Umidità attuale nella zona z0.**
- **I: Ogetto_Testo, mi fa vedere la Temperatura attuale nella zona z0.**



Oss:

Allo stesso modo tutte le zone hanno la possibilità di modificare il proprio SetPoint di temperatura, ritornare nella pagina delle zone se si vuole modificare altre zone o tornare direttamente nella Home_Page se non si vuole più modificare niente.

SOFTWARE

Piattaforma utilizzata per la programmazione del microcontrollore esp32 (IDE Arduino)

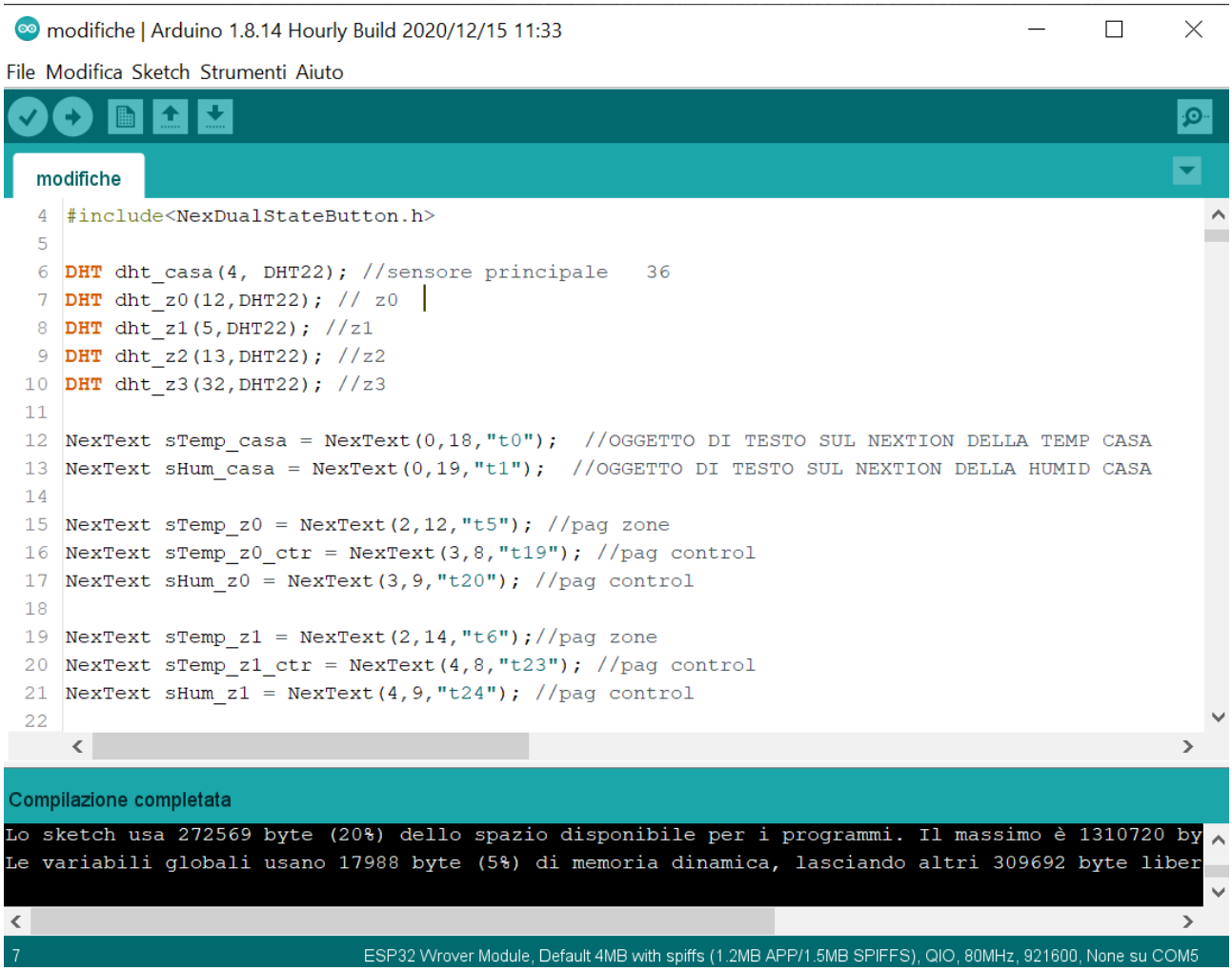
Si ha lavorato con **IDE Arduino** tutto il tempo. Principalmente perché era una piattaforma che ho già utilizzato in passato per progetti personali e lo conoscevo più o meno bene. Il linguaggio di programmazione scelto è stato **linguaggio C** , anche questo scelto perché mi è stato insegnato all'università in Fondamenti di Informatica. Dico **SCELTO** perché ci sono **altre piattaforme e linguaggi alternativi** con cui si poteva lavorare. Ho scelto di lavorare con gli strumenti già acquisiti e rinforzarli prima di imparare altri linguaggi e piattaforme dove lavorare.

In un inizio IDE Arduino andava bene per l'edizione del codice finché il codice non è diventato troppo grande, in quel momento ho cercato un editor esterno per poter ridurre i tempi morti nel

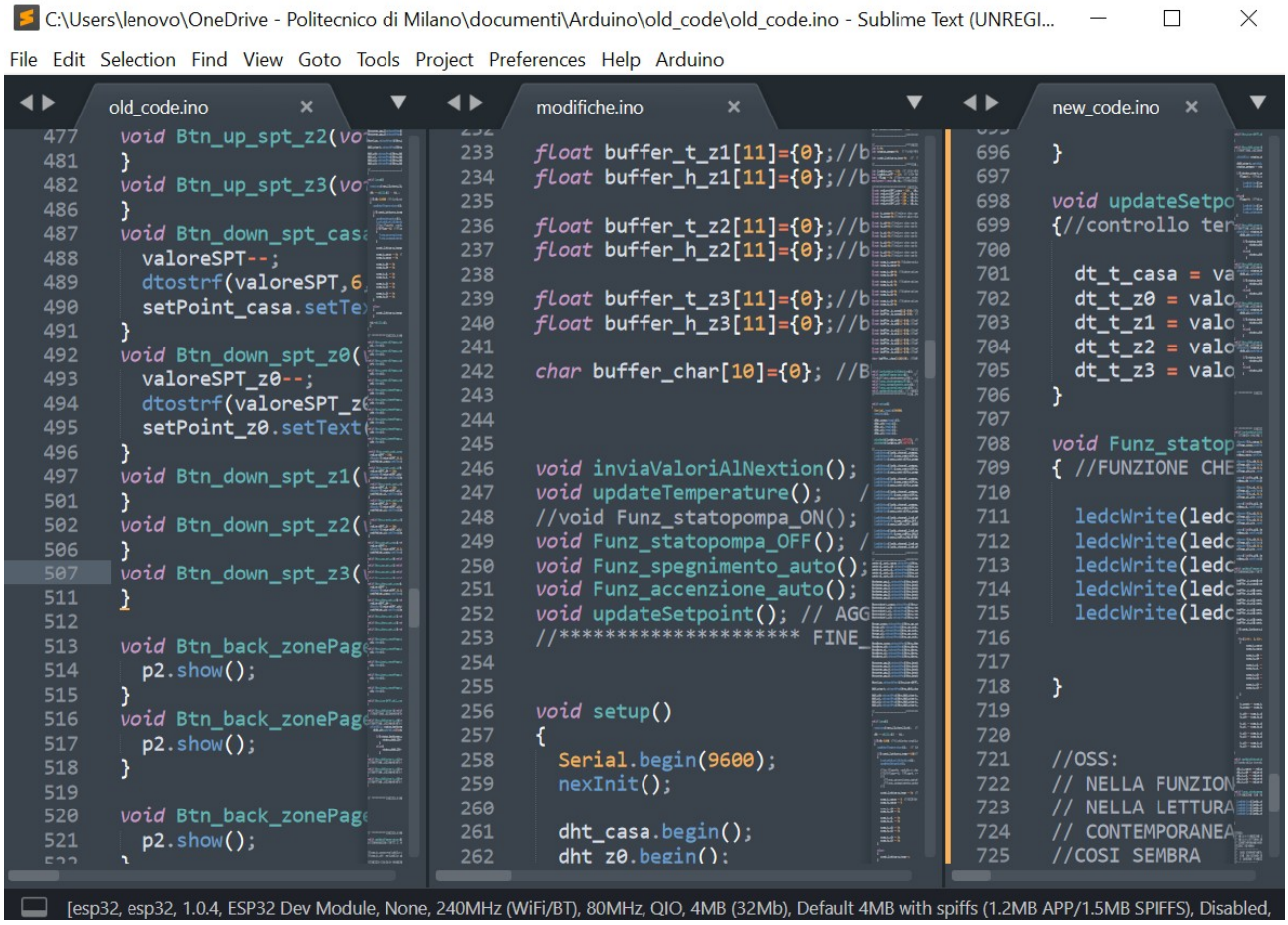
cercare errori o fare modifiche. Mi serviva spazio, e migliori strumenti. Così ho fatto una ricerca e tra le varie opzioni ho scelto di usare **SublimeText_Editor**. È un editor molto versatile, e potente che mi ha aiutato a mantenere in ordine e seguire meglio il codice mammano che cresceva e si complicava.

Di seguito immagini delle due piattaforme citate:

IDE ARDUINO



SUBLIME_TEX



Piattaforma utilizzata per la programmazione del NextiodDisplay (NextionEditor)

Un vantaggio del NextionDisplay è che ha **molto supporto** da parte dall'azienda fabbricante. Nell'apposita pagina web si può tranquillamente trovare un enorme file di istruzioni su come programmare il proprio display e un **Editor gratuito** dove farlo. Grazie anche al fatto che il display è molto diffuso si possono trovare tanti tutorial.

Basicamente segue una g, ogni oggetto ha delle caratteristiche e possono essere modificate dipendendo di cosa si vuole fare.

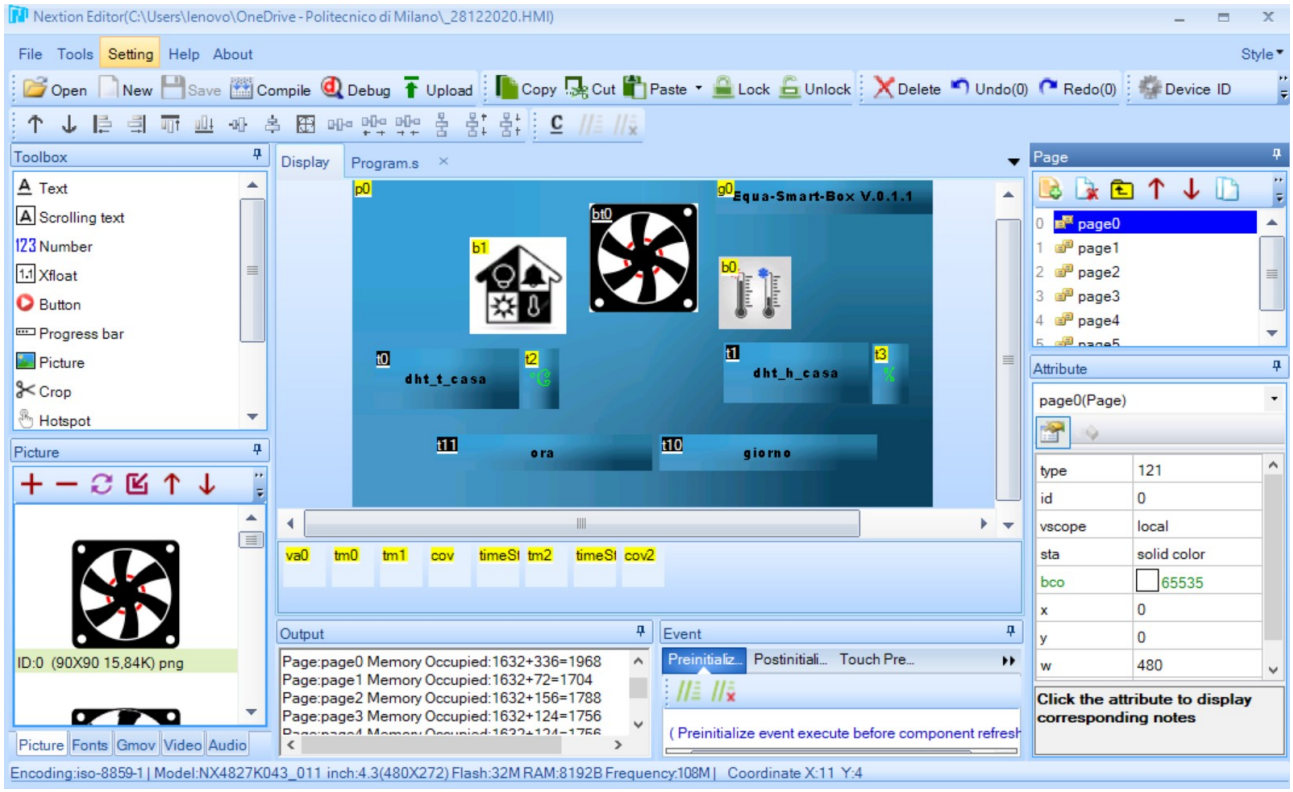
Si possono creare anche dei **"video"** nel display, aggiungendo una serie enorme di fotogrammi della stessa immagine. Procedura fatta per la simulazione del bottone start.

Ogni oggetto può possedere un **programma sia al tocco che al rilascio del bottone**. Si possono aggiungere timer e variabili per creare dei cicli.

Si può fare delle **simulazioni** per verificare che i bottoni e le pagine funzionino correttamente. Non posso descrivere l'itero Editor perché è enorme la quantità di cose che si può fare. Ho dovuto scegliere tra le cose da imparare e che cose lasciare per questioni di tempo.

Di seguito un'immagine dell'editor citato.

NEXTIONEDITOR



Linguaggi Di Programmazione Utilizzati

Ho usato prevalentemente linguaggio C per il microcontrollore.

- Linguaggio C,
- Funzioni di IDE Arduino,
- Programmazione a oggetti nel NextionDisplay,
- Basi di C++
- Basi di Java, programmazione orientata ad oggetti

PEZZI DI CODICI IMPORTANTI E FUNZIONI UTILIZZATE NELL'ESP32

CONTROL_PWM_SIGNAL (ESP32)

Oss:

E' importante la specifica dell'esp32, perché queste funzioni non vanno bene né per Arduino né per il esp8266. dove si usano **" ANALOGWRITE(..) "**

Questa funzione era usata inizialmente per i led, per questo il nome " ledc ", in questo caso si usa per il controllo di motori c.a.(pompa circolatore)

ESEMPIO BREVE:

```
// Numero del Pin_LED
const int ledPin = 16;  // 16 corrisponde al GPIO16
// proprietà PWM
const int freq = 5000;  //scelta frequenza
const int ledChannel = 0; //scelta canale di trasmissione del segnale
const int resolution = 8; // scelta risoluzione, è il numero di bit da usare
void setup(){
// configurazione propietà PWM
ledcSetup ( ledChannel, freq, resolution );
// attacca il cannale al GPIO da controllare
ledcAttachPin ( ledPin, ledChannel );
}
void loop(){
// aumenta la luminosità del LED
for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
// cambia la luminosità del LED con il PWM
ledcWrite(ledChannel, dutyCycle);
delay(15);
}
// diminuisce la luminosità del LED con il PWM
for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
// cambia la luminosità del LED con il PWM
ledcWrite ( ledChannel, dutyCycle );
delay(15);
}
}
```

LETTURA_TEMPERATURA_DHT22 < DHT.H >

Oss:

Si deve usare insieme a un **delay** o un **timer non bloccante** per dare il tempo necessario al sensore di leggere ed inviare i dati.

Si elabora una serie di 10 letture, si fa la media e il risultato si prende come lettura da utilizzare. Il motivo è' che questi sensori sono molto sensibili e negli impianti avvolte capita che un disturbo inaspettato possa creare una lettura erronea. Per minimizzare tale errore si ha pensato a questa soluzione. 10 letture, si fa la media e in caso di errore, l'errore sarà minimizzato.

```
1. #include<DHT.h> //LIBRERIA PER I SENSORI DHT
2. DHT dht_casa(22,DHT22); //CREO L'OGGETTO
3. void setup(){
4. dht_casa.begin(); //LO INIZIALIZO
5. }
6. void loop(){
7. if(dt>100){
8. updateTemperature(); // AGGIORNAMENTO LETTURA DHT22 OGNI 100ms
9. t1=millis();
10. }
11. }
12. void updateTemperature(){  //FUNZIONE CREATA PER L'AGGIORNAMENTO VALORI DI
    TEMPERATURA
13. bufferT[cont_lettura_temp] = dht_casa.readTemperature();  // LETTURA TEMPERATURA
14. bufferH[cont_lettura_temp] = dht_casa.readHumidity();  //LETTURA HUMIDITA'
15. if(cont_lettura_temp==10){ //SOLO QUANDO ARRIVO A 10 LETTURE ENTRO QUI E LE ELABORO
16. for(i=0 ; i<10 ; i++){
17. sum_T=sum_T + bufferT[i];
18. sum_H=sum_H + bufferH[i];
19. }
20. t_casa=sum_T/11; //FACCIO 11 PER MINIMIZARE GLI ERRORI PER DEFAULT DEL SENSORE IN SE
21. h_casa=sum_H/11;
22. }
```

TIMER_NON_BLOCCANTE

OSS:

Serve per evitare di usare i delay, non è possibile usare dei delay in impianti reali fuori dalle prove teoriche perché' un delay comporta rimanere cechi il tempo del delay. **Il delay mette in attesa l'intero codice**, non va bene se devo tenere conto di tanti sensori, elaborazione di dati ecc.

Esempio **delay**:

esempio di un led lampeggiante

```
1. void setup(){
2. pinMode(led_PIN_12,OUTPUT); //dichiaro il pin 12 come OUTPUT
3. }
4. Void loop(){
5. digitalWrite(led_PIN_12 , HIGH); //si accende il led
6. delay(100); // si aspetta 100 milisecondi
7. digitalWrite(led_PIN_12 , LOW); //si spegne il led
8. }
```

Esempio **Timer non bloccante**:

Utilizzato nel progetto per dare ai sensori dht22 il tempo necessario per la lettura e l'invio dei dati

Oss: **millis()** e' una funzione che mi dal tempo reale di funzionamento dell'esp32. Ovvero mammano che l'esp32 rimane accesso, millis() continua ad aumentare. La differenza tra delay ed il timer non bloccante è che il primo mi blocca l'intero ciclo del codice per il tempo che inseriamo, mentre il secondo (il timer non bloccante) non mi ferma niente. Semplicemente crea una condizione insieme a millis() dove dico di non azionare ciò che si trova all'interno finché non si arriva al valore dt>100, ovvero il tempo che mi serve per la lettura dei sensori .

Pezzo di codice utilizzato nel progetto:

```
1. Unsigned long t1=0 ;  // variabile che serve per azzerare dt
2. Unsigned long dt ;  //conterrà la differenza tra il tempo di intervallo che mi serve(100ms) e il
   tempo corrente del microcontrollore( millis() )
3. void loop(){
4. dt = millis() – t1;  //finché non arrivo a 100 ms non entro nel if, questo mi dal tempo neccessario
   da dare ai sensori per lavorare
5. if(dt>100){  //pasato il valore 100ms si entra
6. updateTemperature(); // aggiornamento lettura dht22 ogni 100ms
7. t1=millis();  //si azera dt, perche stiamo dando a t1 lo stesso valore di millis()
8. }
9. }
```

ESEMPIO PRATTICO timer non bloccante, CON dt > 1000 ms

loop(250)	300	0	300	dt>1000? NO
loop(250)	550	0	550	dt>1000? NO
loop(250)	800	0	800	dt>1000? NO
loop(250)	1050	0	1050	dt>1000? SI update! e t1=millis()=1050
loop(250)	1300	1050	250	dt>1000? NO
loop(250)	1550	1050	500	dt>1000? NO
loop(250)	1800	1050	750	dt>1000? NO
loop(250)	2050	1050	1000	dt>1000? SI update! e t1=millis()=2050
loop(250)	2300	2050	250	dt>1000? NO
loop(250)	2550	2050	500	dt>1000? NO
loop(250)	2800	2050	750	dt>1000? NO
loop(250)	3050	2050	1000	dt>1000? SI update! e t1=millis()=3050
loop(250)	3300	3050	250	dt>1000? NO
loop(250)	3550	3050	500	dt>1000? NO
loop(250)	3800	3050	750	dt>1000? NO
loop(250)	4050	3050	1000	dt>1000? SI update! e t1=millis()=4050

TROVA_&_CANCELLA_ERRORI

Oss:

Nell'elaborazione dei datti letti, ci siamo resi conto che a causa di **disturbi imprevisti** o errori di fabbrica i sensori possono generare letture sbagliate. Sbalzi di temperatura che generano malfunzionamenti. Come ad esempio l'accezione o spegnimento delle pompe in caso di letture di temperature troppo basse o troppo elevate rispettivamente.

Si è pensato a un filtro digitale, un codice che prima di elaborare ed inviare al NextionDisplay cercano tra tutte le letture ed eliminano quelle che non passano il filtro.

In sintesi: **Ogni 10 letture istantanee, se non ci sono errori i valori si elaborano** (viene fatta la media ogni 10 letture) **e il valore finale viene inviato e utilizzato**. **Se trova uno o più errori di lettura, questi vengono eliminati e si elaborano solo i valori accettabili.**

Ad esempio, se delle 10 letture 2 sono letture troppo sballate rispetto la media, queste vengono eliminate e si fa la media delle 8 letture rimaste.

PEZZO DI CODICE UTILIZZATO NEL PROGETTO:

```
1.  void updateTemperature()
2.  {//INMAGAZINA TUTTI I VALORI DA ELABORARE DOPO
3.
4.    buffer_t_casa[cont_lettura_temp] = dht_casa.readTemperature();  // sesnsore DHT22
    dentro CASA
5.    buffer_h_casa[cont_lettura_temp] = dht_casa.readHumidity();  //10 valori
6.
7.    if(cont_lettura_temp==10) //arrivo ale 10 letture
8.    {
9.
10. //sum_t_casa variabile globale per inmagazzinare t_Casa
11. //sum_t_z0  variabile globale per inmagazzinare t_z0
12.
13. //INIZIO CALCOLO ERRORE
14.
15.  //CALCOLO LA SOMMA DELLE LETTURE DI OGNI SENSORE
16.
17.    for (i=0; i<10; i++)
18.    {
19.        sum_t_casa = sum_t_casa + buffer_t_casa[i];
```

```
20.       sum_h_casa = sum_h_casa + buffer_h_casa[i];
21.
22.    }
23. //CALCOLO LA MEDIA DELLE 10 LETTURE
24.
25. MEDIA_sum_t_casa = sum_t_casa / 10;
26. MEDIA_sum_h_casa = sum_h_casa / 10;
27. //ACQUISIZIONE VALORI PER LA MEDIA DEL MARGINE DI ERRORE
28. // float margine_di_errore_t_casa_%_MEDIA ; per tutti
29.
30. margine_di_errore_10%_MEDIA_t_casa = MEDIA_sum_t_casa / 10;
31. margine_di_errore_10%_MEDIA_h_Casa = MEDIA_sum_h_casa / 10;
32.
33. // float varianza_temperatura = +- 1,5;
34. // float varianza_humidita = +- 5;
35. // int error_count_t_casa=0;
36. // int error_count_h_casa=0; creare per tutte le zone
37.
38. for(i=0 ; i<10 ; i++){ //RIPETERE STESSAPROCEDURA PER UMIDITA E TUTTE LE ZONE
39.
40.     if((buffer_t_casa[i] - MEDIA_sum_t_casa)>=0){ //se errore positivo
41.     if(buffer_t_casa[i] - MEDIA_sum_t_casa > margine_di_errore_10%_MEDIA_t_casa +
    varianza_temperatura)
42.                 error_count_t_casa++;
43.                 buffer_t_casa[i]=0;
44.     }
45.     else if((buffer_t_casa[i] - MEDIA_sum_t_casa)<0){//se errore negativo
46.
47.         if( (buffer_t_casa[i] - MEDIA_sum_t_casa) < (-
    margine_di_errore_10%_MEDIA_t_casa - varianza_temperatura) ){
48.                 error_count_t_casa++;
49.                 buffer_t_casa[i]= 0;
```

```
                    }
    }

    buffer_margine_errore_t_z0 = buffer_t_z0[i] - MEDIA_sum_t_z0;
    buffer_margine_errore_h_casa = buffer_h_casa[i] - MEDIA_sum_h_casa;
  }
  //azero la variabile che contendrà la somma

//rifaccio la somma questa volta senza i valori che non hanno superato il filtro

  for(i=0 ; i<10 ; i++)
  {
     MEDIA_sum_t_casa = MEDIA_sum_t_casa + buffer_t_casa[i];
     MEDIA_sum_h_casa = MEDIA_sum_h_casa + buffer_h_casa[i];
  }

  //ricalcolo la somma e la divido per il numero di letture valide che
  //hanno superato il filtro

MEDIA_sum_t_casa = MEDIA_sum_t_casa / ( 10 - error_count_t_casa);
MEDIA_sum_h_casa = MEDIA_sum_h_casa / ( 10 - error_count_h_casa);

//inmagazino i valori finali di ogni sensore nell'apposita variabile
//che sarà poi inviata al nextion

  t_casa = MEDIA_sum_t_casa / (10-pos_t_casa) ;
  h_casa = MEDIA_sum_h_casa / (10-pos_t_casa) ;
```

Esempio su come funziona il filtro

Temperature[i](°C)		Media	M.E.P.U.L.	SOGLIA	CONTROLO
t1	18	-18,95	-0,95	-3,3	OK
t2	19	-18,95	0,05	3,3	OK
t3	18	-18,95	-0,95	-3,3	OK
t4	18.5	-18,95	-0,45	-3,3	OK
t5	18	-18,95	-0,95	-3,3	OK
t6	19	-18,95	0,05	3,3	OK
t7	19.5	-18,95	0,55	3,3	OK
t8	20	-18,95	1,05	3,3	OK
t9	20	-18,95	1,05	3,3	OK
t10	19.5	-18,95	0,55	3,3	OK
M.T.-->	18.95				

M.T. = Media_Temperature = 18.95 °C
M.E.P.U.L. = Margine_di_Errore_Per_Unita_di_Lettura = Temp[i] - Media_Temp
M.E.M. = Margine di Errore% (10%*Temp_Media) = 1,8°C
V.S. = Varianza_Scelta = +/- 1,5 °C
CONTROLLO SE LE LETTURE SONO OK
SOGLIA DI ERRORE DELLA LETTURA (-V.S. - M.E.M) <= M.E.P.U.L. <= (V.S. + M.E.M)
UGUALE= -3,3 UGUALE = 3,3

Esempio_2) Dieci Letture, con 1 ERRORE

Temperature[i](°C)		Media_T	M.E.P.U.L.	SOGLIA	CONTROLO
t1	18	-21	-3	-3,6	OK
t2	18.5	-21	-2,5	-3,6	OK
t3	19	-21	-2	-3,6	OK
t4	40	-21	19	3,6	ERROR
t5	19.5	-21	-1,5	-3,6	OK
t6	19	-21	-2	-3,6	OK
t7	19	-21	-2	-3,6	OK
t8	19.5	-21	-1,5	-3,6	OK
t9	19	-21	-2	-3,6	OK
t10	18,5	-21	-2,5	-3,6	OK
M.T.-->	21				

M.T. = Media_Temperature = 21 °C
M.E.P.U.L. = Margine_di_Errore_Per_Unita_di_Lettura = Temp[i] - Media_Temp
M.E.. = Margine di Errore% (10% *Temp_Media) = 2,1°C
V.S. = Varianza_Scelta = +/- 1,5 °C
CONTROLLO SE LE LETTURE SONO OK
SOGLIA DI ERRORE DELLA LETTURA (-V.S. - M.E.) <= M.E.P.U.L. <= (V.S. + M.E.)
UGUALE= -3,6 UGUALE = 3,6

NEXTIONDISPLAY_LIBRERIA <Nextion.h> & <nexDualStateButton.h>

Oss:

La programmazione del display è orientata ad oggetti. Per la creazione di oggetti nel display si crea l'apposito oggetto che fa da collegamento nel esp32.

Nella dichiarazione delle funzioni dei bottoni e il loro nome, si vedrà **attachPop** e **attachPush**.Il primo, **attachPop**, vuol dire che la funzione del bottone deve iniziare non appena il dito rilascia il bottone. Mentre **attachPush** vuol dire che la funzione del bottone deve iniziare non appena il dito tocca il bottone.

Sintassi degli oggetti → **< pagina , numero_ID , "nome_oggeto_nel_NextionEditor" >**

ESEMPIO DI ALCUNI OGGETTI UTILIZZATI NEL PROGETTO:

```
#include<Nextion.h>  //libreria nextiondisplay
#include<NexDualStateButton.h>  //libreria per I bottoni dual_state

NexText sTemp_casa = NexText(0,18,"t0"); //oggetto di testo
NexPage p0 = NexPage(0,0,"page0");   //oggetto di pagina
NexButton control_spt_casa = NexButton(0,2,"b0");  //oggetto di bottone_normale
NexDSButton dsB_start = NexDSButton(0,6,"bt0"); //oggetto di bottone dual_state

NexTouch*nex_listen_list[]={  //questa funzione è un loop del display che mette I ascolto tutti I bottoni. Qua dentro infatti devo mettere solo bottoni. Tutti i bottoni avranno le loro funzioni (compiti) elaborati poi alla fine al di fuori del setup e del loop principale
& control_spt_casa,  //bottone per il controllo del setpoint di temperatura della casa
& dsB_start,  //bottone che aziona l'inizio del funzionamento
NULL
};
Void setup(){
control_spt_casa.attachPop(Btn_controlPage_spt_casa,&control_spt_casa); //dichiarazione dei bottoni e do un nome alle rispettive funzioni
dsB_start.attachPop(Btn_dsB_start,&dsB_start);//start button
}
void loop()
{
 nexLoop(nex_listen_list);  // LOOP DEI BOTTONI DEL NEXTION
}
Void control_spt_casa(void *ptr){
…codice, cosa deve fare il bottone..}
Void dsB_start (void *ptr){
…codice, cosa deve fare il bottone.. }
```

LIBRERIA_UTILIZATE

Oss:

Le librerie sono molto importanti, **alcuni sensori sono supportati dalla piattaforma IDE Arduino** e ti trovi all'interno le librerie pronto all'uso**, altre invece non si trovano e vanno scaricare e installate** per poter compilare il codice.

Per ogni pezzo aggiunto si ha fatto una ricerca per poter usare al meglio le librerie offerte dalla comunità di Arduino. Grazie a che questa piattaforma è aperta, si può chiedere dubbi su varie cose o vedere il lavoro di altre persone o usare librerie create da altre persone. Questo è uno strumento molto potente. Ci permette di risparmiare tempo ed evitare sbagli fatti da altre persone.

```
#include<DHT.h>   //sensore di temperatura

#include<Nextion.h>  //NextionDisplay utilizzato

#include<NexDualStateButton.h>  //usato per un oggetto specifico dell'NextionDisplay
```

Comunicazione ESP32 e NextionDisplay

Fino adesso ho parlato delle librerie del NextionDisplay e l'elaborazione di dati del microcontrollore senza specificare molto riguardo il come si è stato fatto.

Per stabilire il collegamento degli oggetti del NextionDisplay e il microcontrollore ESP32 bisogna creare per ogni **oggetto_NextionDisplay** un rispettivo **oggetto nell'ESP32**. **Gli oggetti del NextionDisplay hanno delle caratteristiche**, dei parametri **che fanno da collegamento con gli oggetti creati nel esp32**.

In un inizio è stato molto difficile capire questa comunicazione e riuscire a crearne vari senza errori di compilazione, mammano che si andava avanti diventavo sempre più versatile e per aumentare l'efficienza dovevo mantenere l'ordine. In un inizio erano pochi oggetti, il codice finale ha quasi 100 oggetti e ce ne sono molti altri da creare per migliorare la comunicazione **display-utente.**

Di seguito una guida esatta di tutti gli oggetti creati nel NextionDisplay per pagine e per parametri

Nextion_Display Ogetti_PAGINA_a_PAGINA

EQUA_SMART_BOX___NEXTIONDISPLAY_OGGETTI /BOTTONI

Page	Button	Scrolling_Testo
P0_home_page	b0 -> Control_Temperatura_dht_Casa	txt_Equa_smart_box_v0.0.2
	b1 -> Control_Zone_dht_zone	
P1_controllo_temp	b2_ (+)_SPT_dht_Casa	
_dht_casa	b3_ (-)_SPT_dht_Casa	
	b4 -> Page0	
	b5_Restart = 20_SPT_dht_Casa	
P2_Control_Zone	b6 -> Pg0	
	b7_ripristina_zone	
	b8 -> z0_Control_SetPointTemperatura_z0	
	b9 -> z1_Control_SetPointTemperatura_z1	
	b10 -> z2_Control_SetPointTemperatura_z2	
	b11 -> z3_Control_SetPointTemperatura_z3	
P3_Controllo_SPT_Z0	b12 -> Pg0	
	b13_(+)_SPT_dht_z0	
	b14_(-)_SPT_dht_z0	
	b15_Restart = 20_SPT_dht_z0	
	b16 -> Pg2	
P4_Controllo_SPT_Z1	b17 -> Pg0	
	b18_(+)_SPT_dht_z1	
	b19_(-)_SPT_dht_z1	
	b20_Restart = 20_SPT_dht_z1	
	b21 -> Pg2	
P5_Controllo_SPT_Z2	b22 -> Pg0	
	b23_(+)_SPT_dht_z2	
	b24_(-)_SPT_dht_z2	
	b25_Restart = 20_SPT_dht_z2	
	b26 -> Pg2	
P6_Controllo_SPT_Z3	b27 -> Pg0	
	b28_(+)_SPT_dht_z3	
	b29_(-)_SPT_dht_z3	
	b30_Restart = 20_SPT_dht_z3	
	b31 -> Pg2	
Page	**Dual_State_Button**	**Testo**
P0_home_page	bt0_ON/OFF_dht_Casa	t0_t_casa
		t1_h_casa

		t2_°C (testo)
		t3_% (testo)
		t10_giorno(da_modificare)
		t11_ora(da_modificare)
P1_controllo_temp		t4_SetPointTempeartura_dht_casa
_dht_casa		
P2_Control_Zone	bt4_ON/OFF_zona_z0	t5_temperatura_t_z0
	bt5_ON/OFF_zona_z1	t6_temperatura_t_z1
	bt6_ON/OFF_zona_z2	t7_temperatura_t_z2
	bt7_ON/OFF_zona_z3	t8_temperatura_t_z3
		t9_°C (testo)
		t12_°C (testo)
		t13_°C (testo)
		t14_°C (testo)
P3_Controllo_SPT_Z0		t15_SPT_z0
		t19_t_z0
		t20_h_z0
		t21_°C (testo)
		t22_% (testo)
		t35_Z0 (testo)
P4_Controllo_SPT_Z1		t16_SPT_z1
		t23_t_z1
		t24_h_z1
		t25_°C (testo)
		t26_% (testo)
		t36_Z1 (testo)
P5_Controllo_SPT_Z2		t17_SPT_z2
		t27_t_z2
		t28_h_z2
		t29_°C (testo)
		t30_% (testo)
		t37_Z2 (testo)
P6_Controllo_SPT_Z3		t18_SPT_z3
		t31_t_z3
		t32_h_z3
		t33_°C (testo)
		t34_% (testo)
		t38_Z3 (testo)

NEXTIONDISPLAY_OGGETTI_PARAMETRI

EQUA_SMART_BOX_PARAMETRI_NEXTIONDISPLAY_OGGETTI /BOTTONI

PARAMETRI	(< page > , < ID > , < nome_oggetto >)		
TESTO	DESCRIZIONE	BOTTONI	DESCRIZIONE
(0 , 18 , "t0")	DHT_T_CASA	(0 , 2 , "b0")	CONTROL DHT_CASA
(0 , 19 , "t1")	DHT_H_CASA	(0 , 2 , "b1")	CONTROL DHT_ZONE
(0 , 4 , "t2")	°C	(1 , 2 , "b2")	(+) DHT_SPT_CASA
(0 , 5 , "t3")	%	(1 , 3 , "b3")	(-) DHT_SPT_CASA
(0 , 14 , "t11")	ORA	(1 , 4 , "b4")	(<<) TORNA ALLA HOME
(0 , 10 , "t10")	GIORNO	(1 , 6 , "b5")	RESTART SPT_DHT_CASA
(1 , 5 , "t4")	SPT_CASA	(2 , 7 , "b6")	(<<) TORNA ALLA HOME
(2 , 12 , "t5")	T°_Z0	(2 , 6 , "b7")	RIPRISTINA ZONE_ALL OFF
(2 , 14 , "t6")	T°_Z1	(2 , 8 , "b8")	CONTROL_SPT_Z0
(2 , 15 , "t7")	T°_Z2	(2 , 9 , "b9")	CONTROL_SPT_Z1
(2 , 16 , "t8")	T°_Z3	(2 , 10 , "b10")	CONTROL_SPT_Z2
(2 , 13 , "t9")	°C	(2 , 11 , "b11")	CONTROL_SPT_Z3
(2 , 17 , "t12")	°C	(3 , 5 , "b12")	(<<) TORNA ALLA HOME
(2 , 18 , "t13")	°C	(3 , 2 , "b13")	(+) SPT_Z0
(2 , 19 , "t14")	°C	(3 , 3 , "b14")	(-) SPT_Z0
(3 , 4 , "t15")	SPT_Z0	(3 , 6 , "b15")	RESTART SPT_DHT_Z0
(3 , 8 , "t19")	T°_Z0	(3 , 7 , "b16")	TORNA ALLA ZONE PAGE
(3 , 9 , "t20")	H%_Z0	(4 , 3 , "b17")	(<<) TORNA ALLA HOME
(3 , 10 , "t21")	°C	(4 , 4 , "b18")	(+) SPT_Z1
(3 , 11 , "t22")	%	(4 , 5 , "b19")	(-) SPT_Z1
(4 , 2 , "t16")	SPT_Z1	(4 , 6 , "b20")	RESTART_SPT_Z1
(4 , 8 , "t23")	T°_Z1	(4 , 7 , "b21")	TORNA ALLA ZONE PAGE
(4 , 9 , "t24")	H%_Z1	(5, 3 , "b22")	(<<) TORNA ALLA HOME
(4 , 10 , "t25")	°C	(5 , 4 , "b23")	(+) SPT_Z2
(4 , 11 , "t26")	%	(5 , 5 , "b24")	(-) SPT_Z2
(5 , 2 , "t17")	SPT_Z2	(5 , 6 , "b25")	RESTART_SPT_Z2
(5 , 8 , "t27")	T°_Z2	(5 , 7 , "b26")	TORNA ALLA ZONE PAGE
(5 , 9 , "t28")	H%_Z2	(6 , 2 , "b27")	(<<) TORNA ALLA HOME
(5 , 10 , "t29")	°C	(6 , 4 , "b28")	(+) SPT_Z3
(5 , 11 , "t30")	%	(6 , 5 , "b29")	(-) SPT_Z3
(6 , 3 , "t18")	SPT_Z3	(6 , 7 , "b30")	RESTART_SPT_Z3
(6 , 8 , "t31")	T°_Z3	(6 , 6 , "b31")	TORNA ALLA ZONE PAGE
(6 , 9 , "t32")	H%_Z3		
(6 , 10 , "t33")	°C		
(6 , 11 , "t34")	%		
(3 , 12 , "t35")	txt_Z0		
(4 , 12 , "t36")	txt_Z1		
(5 , 12 , "t37")	txt_Z2		
(6 , 12 , "t38")	txt_Z3		

PAGES	DESCRIZIONE
(0 , 0 , " page 0")	PAGE 0
(1 , 0 , " page 1")	PAGE 1
(2 , 0 , " page 2")	PAGE 2
(3 , 0 , " page 3")	PAGE 3

(4 , 0 , " page 4")	PAGE 4
(5 , 0 , " page 5")	PAGE 5
(6 , 0 , " page 6")	PAGE 6

DUAL_STATE_BUTTONS	DESCRIZIONE
(0 , 6 , "bt0")	BOTTONE ON/OFF AUTO
(2 , 2 , "bt4")	Z0_START ON/OFF
(2 , 3 , "bt5")	Z1_START ON/OFF
(2 , 4 , "bt6")	Z2_START ON/OFF
(2 , 5 , "bt7")	Z3_START ON/OFF

SCROLLING_TEXT	DESCRIZIONE
(0 , 9 ,"g0")	logo EQUA_S.B.

Oss:

Senza questa tabella è molto difficile seguire e fare modifiche. Sono molto importanti soprattutto per il NextionDisplay che ha un linguaggio molto rigido.

Software + Hardware, codice completo, compilato e provato allegato

In allegato il codice completo in formato **PDF** e in formato **FILE INO** (si può verificare con la piattaforma di IDE arduino).

Oss: Per la compilazione del codice bisogna scaricare le librerie e impostare la scheda per poter caricare il codice sull'esp32.

Immagine reale, su BreadBoard, del progetto finale

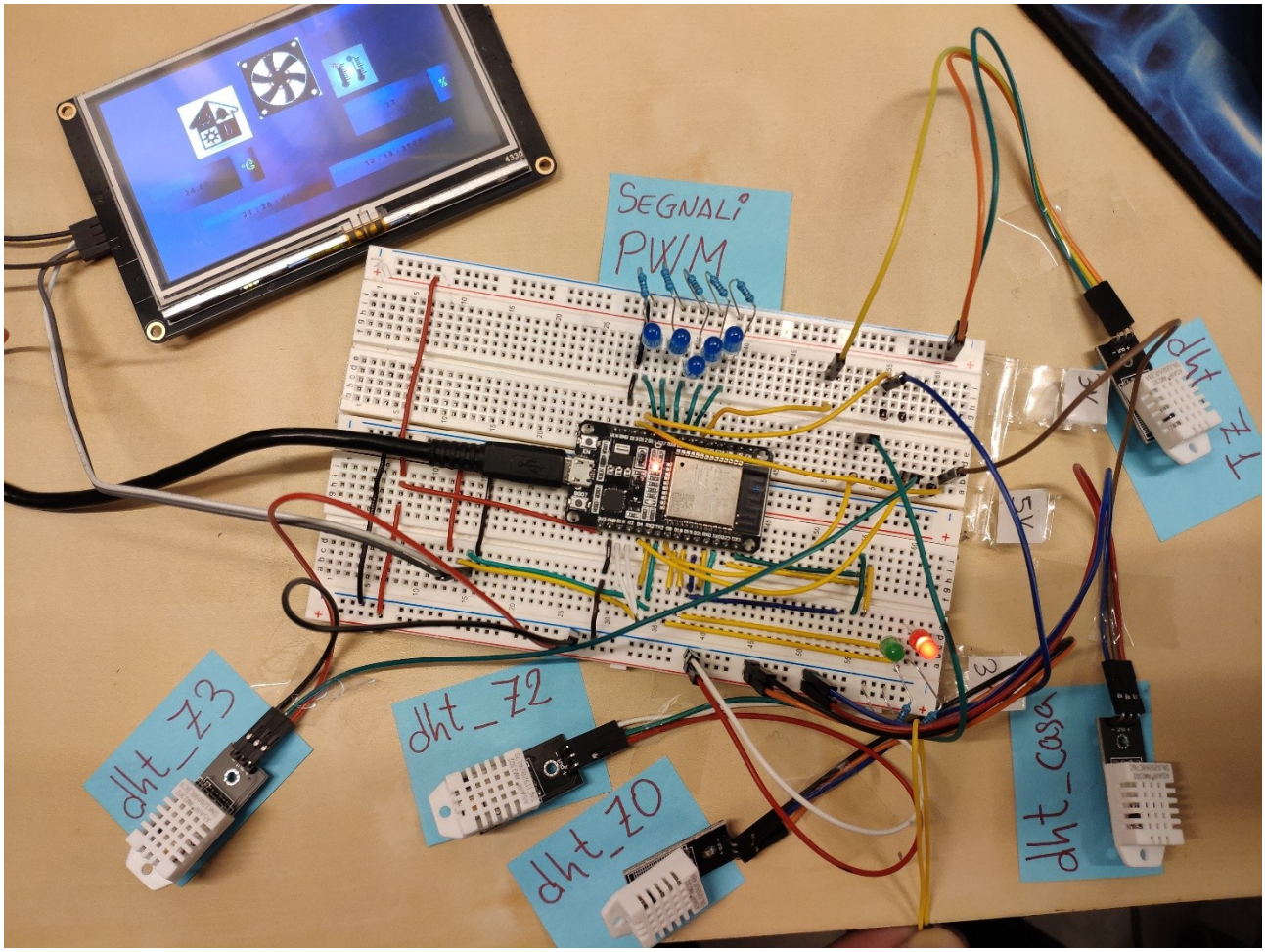


Diagramma di Flusso Del Codice del Progetto

Con questo diagramma è piu facile seguire la lettura del codice completo.

-Diagramma sintetico del codice

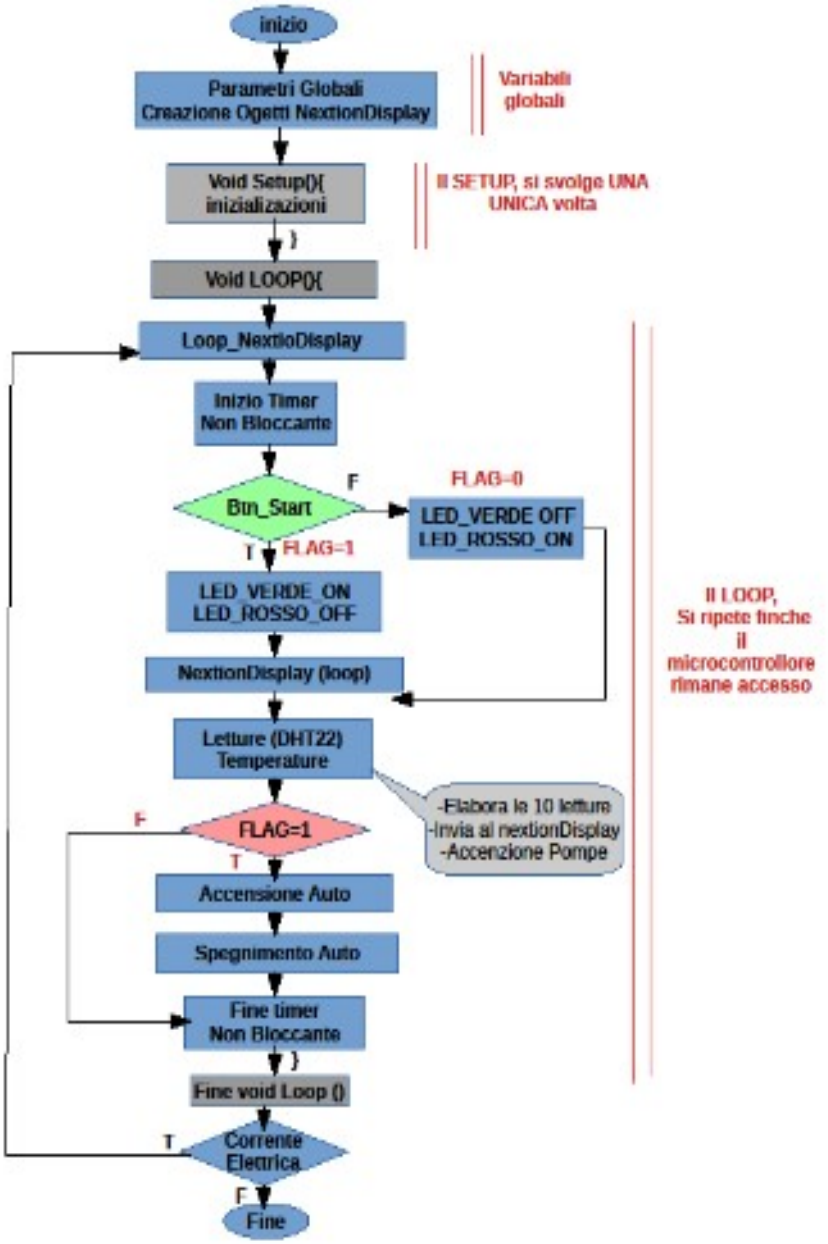
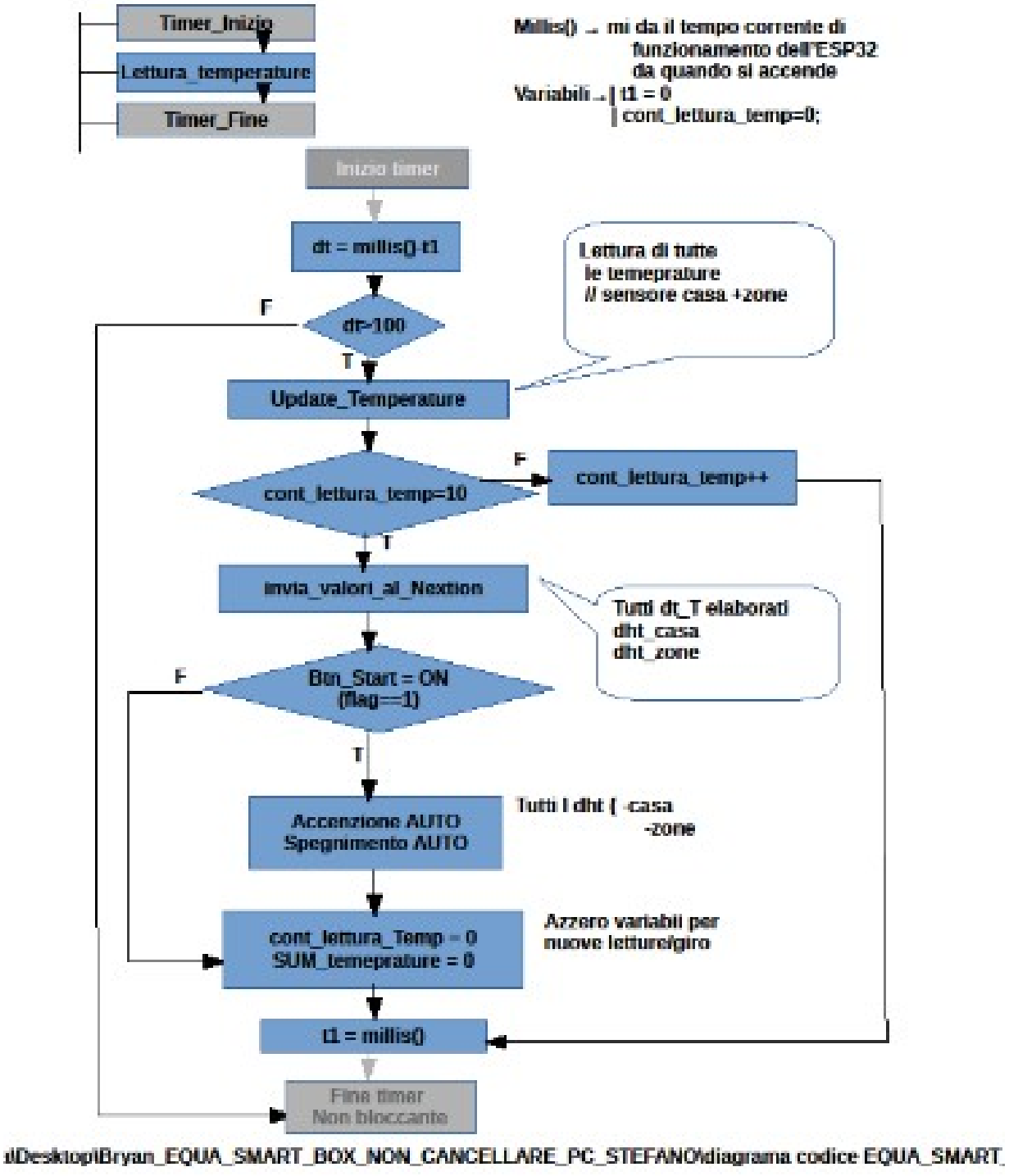
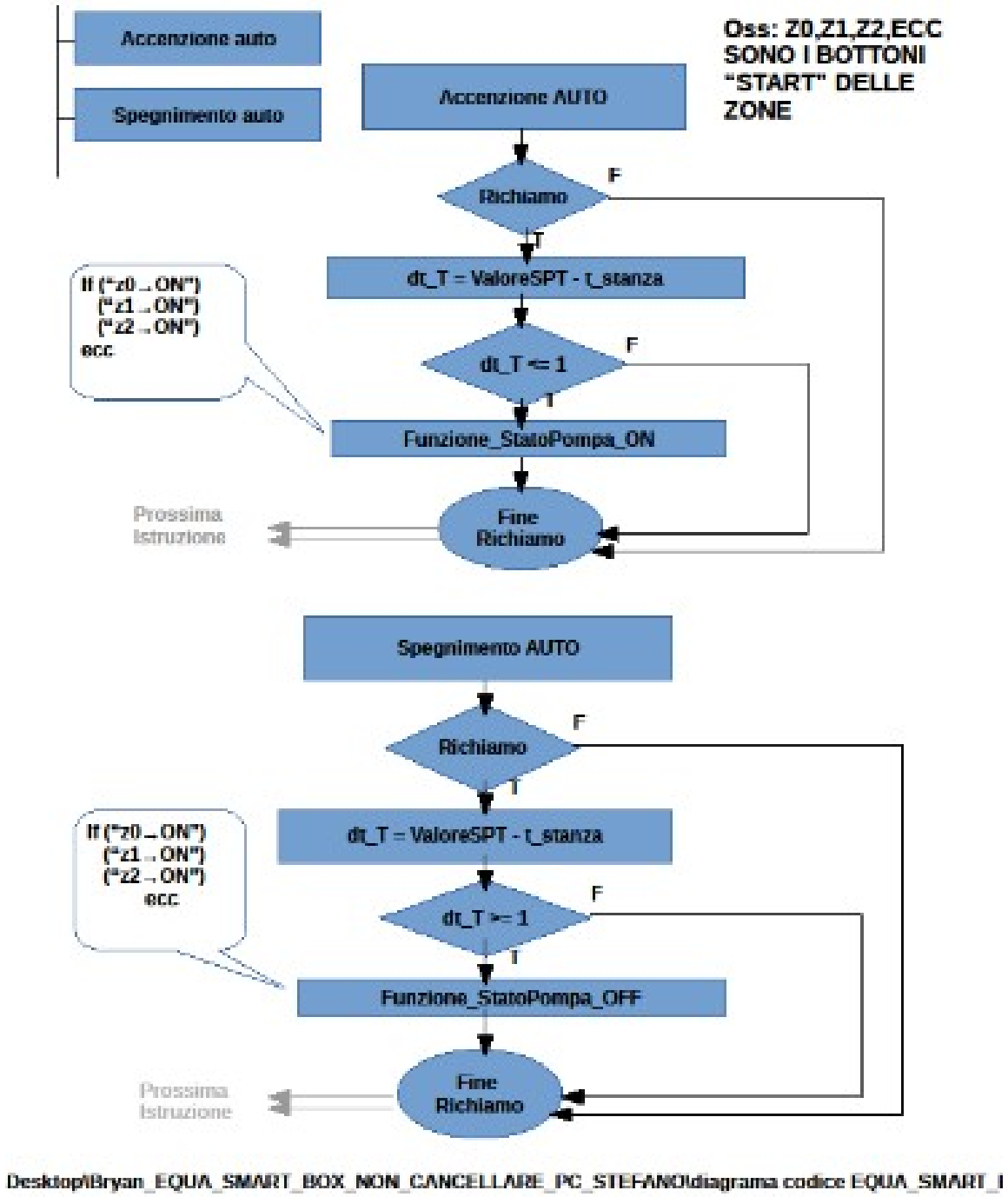


Diagramma di flusso specifico del timer non bloccante



dDesktop\Bryan_EQUA_SMART_BOX_NON_CANCELLARE_PC_STEFANO\diagrama codice EQUA_SMART_

Diagramma di flusso del funzionamento della funzione automatica



Desktop\Bryan_EQUA_SMART_BOX_NON_CANCELLARE_PC_STEFANO\diagrama codice EQUA_SMART_I

PERCORSO FORMATIVO

In un inizio le **problematiche di partenza** erano **come programmare? quale piattaforma?** e **quali sono le capacita del microcontrollore?** . Quindi ho fatto ricerca su internet, libri e forum per avere un'idea migliore sul **come partire e procedere con il progetto**. Ad ogni paso c'era sempre ricerche da fare, sia per nuove librerie da imparare, per comandi nativi dalle piattaforme utilizzate, questioni hardware su come riuscire ad avere protezioni per il microcontrollore e i sensori da elevate tensioni, scelta dei sensori, scelta dei protocolli di comunicazione, ecc. **Per ogni paso si ha fatto una ricerca e una scelta accurata per soddisfare le richieste del progetto.**

Ho trovato enorme gratificazione nel processo della ricerca e formazione, mi sono trovato d'avanti un'innumerevole quantità di argomenti fatti dai vari corsi all'università applicate al mondo reale, a circuiti veri. Tante delle considerazioni di valori di misure presse in classe dati dai professori che in classe non si capiva da subito perché scegliesse tali valori, nel fare delle prove con hardware e software venivano fuori questi valori. Diventava tutto sempre su interessante.

In un inizio costruivo tutto a pezzi seguendo la logica insegnata dai vari corsi, tra misure, mosfet, considerazioni di correnti, velocita di trasmissioni dati ecc. Tutte considerazioni fatte come se fossero esercizi in carta e penna, questa volta però con risultati visivi e pratici. Mammano che presentavo aggiornamenti del progetto il tutor dell'azienda mi dava dei feedback su come migliorare e delle considerazioni da prendere. Mi ha colpito la prima volta che mi hanno menzionato i Board (**schede, moduli che fanno un compito specifico**), non dovevo costruire esattamente tutto da capo, c'erano moduli che potevamo implementare. Da lì in poi tutto diventò un lavoro di continuo studio software, ricerca hardware e trovare una via di mezzo. Non potevo usare solo dei moduli e non potevo solo costruire tutto da capo.

Linguaggi Di Programmazione Imparati

Migliorie nella programmazione in Linguaggio C

Fondamenti basici di linguaggio C++

Comandi nativi dalla piattaforma IDE Arduino

- **pinMode** – per configurare i pin della scheda.
- **digitalWrite** – per accendere un pin.
- **digitalRead** – per leggere lo stato di un pin.
- **analogWrite** – per creare dei segnali variabili.
- **analogRead** – per leggere segnali analogici.

Migliorie nella programmazione in Java

Migliorie nell'uso di ATUTOCAD per gli schemi elettrici ed altri programmi simili per la creazione dell'interfaccia del NextionDisplay.

Libri Letti e Siti Web da cui ho preso Informazioni

Libri

1. **Il Manuale di Arduino guida completa**, Paolo Aliverti
2. **Arduino Trucchi e Segreti**, Paolo Aliverti
3. **Programmare**, Paolo Aliverti

Pagine Web

1. https://forum.arduino.cc/index.php?board=34.0 ,**foro Arduino**
2. https://github.com/espressif/arduino-esp32 , **github** è un potente sito dove trovi ogni genere di librerie e lavoro di altre persone. Anche questo è come un forum. Le persone compartono codici, problemi e soluzioni.
3. https://nextion.tech/ , sito della casa produttrice del **NextionDisplay**. Qua trovi tutto ciò che ti serve riguardo questi display.

ESPERIENZA LAVORATIVA

INIZIO PROGETTO, PRIMI PASI, SVOLGIMENTI, AGGIORNAMENTI, PROGETTO FINALE

Prima dei lavori nei cantieri

Anche se pochi, i lavori nei cantieri sono stati crociali per lo sviluppo del progetto. In un inizio la comunicazione tra quello che loro (l'azienda) avevano in mente e cosa capivo io (senza aver mai fatto prima il lavoro) era un po' sfasata. Sapevo che dovevo sviluppare un controllo termico e quindi in un inizio sapevo che dovevo usare sensori di temperatura e un display. **Infatti il primo prototipo consegnato era un piccolo Arduino MEGA insieme ad un sensore DHT11 e un display LCD 2x8, già con un codice dentro e funzionante**. Ma l'introduzione di "controllo di un motore" e quindi la creazione di un regolatore PID mi era un po' difficile da digerire. Non capivo dove entrava il motore, come costruire il PID, o come era fatto di per sé una pompa di calore. Questo era proprio l'inizio del tirocinio, i primi giorni.

Dopo i lavori nei cantieri

Con il tempo mammano che andavamo avanti, la strada era sempre più chiara su come fare, su cosa fare e alla fine di ogni modifica importante presentavo un aggiornamento del progetto per avere dei feedback, errori e consigli da parte del tutor dell'azienda per in fine arrivare al progetto finale presentato.

LAVORI NEI CANTIERI, ESPERIENZA DI PRIMA PERSONA E FORMAZIONE SUL LAVORO SVOLTO DELL'AZIENDA

Impianti Pompa di Calore

Più che i lavori nei cantieri, è la qualità dell'ingegnere che mi forniva l'informazione. Mi hanno fatto vedere l'intero percorso di una pompa di calore di un cliente, facendomi vedere dall'interno e sul tetto come è fatto e come funziona il ciclo completo. Loro hanno un'azienda Partner, mi hanno fatto vedere come è fatto l'interfaccia utente (Il display di una azienda partner), cosa può fare, cosa monitora, le pagine di controllo, come sono collegati, lo spazio e le distanze che dovrà coprire l'Equa Smart Box da costruire, ecc. La spiegazione nei cantieri mi ha sbloccato e chiarito svariati dubbi su come procedere con la scelta dell'hardware e lo sviluppo del software.

Impianti Pannelli Solari

Di solito gli impianti di pompa di calore visti funzionavano insieme ad un impianto di panelli solari. Anche quest'ultimo mi è stato spiegato e fatto vedere dal momento della sua progettazione, schemi e installazione. Il progetto è mirato a continuare verso il controllo di questi impianti per avere un controllo intelligente del flusso di energia utilizzato.

COLLABORAZIONI E SUPPORTO DA PARTE DALL'AZIENDA

EQUA (Energia di Qualità per l'Uomo e l 'Ambiente), è un team di **Ingegneri e un architetto**, molto professionali ed esperti. Il team va dai fondatori dell'azienda fino arrivare ai più giovani (non per questo meno esperti) e i tirocinanti.

Sin dall'inizio sia il tutor che il resto del team di ingegneri sono stati più che disponibili e pazienti per ogni richiesta o chiarimento. Insieme al tutor mi è stato affiancato un ingegnere che mi aiutava a seguire il progetto, l'ingegnere Renato Manzoni che è stato molto paziente, mi ha seguito, consigliato e offerto la sua esperienza sul campo. Soprattutto nella guida sui protocolli di comunicazione, attualmente il codice ha due parti. Una riguarda i protocolli di comunicazioni (fatto prevalentemente da lui) e un altro il controllo termico. Su questo report si trova soltanto il controllo termico dal momento che è il pezzo su cui ho lavorato.

Ringraziamenti

All’intero Team EQUA per il supporto nello svolgimento del progetto.

Michele ed Ernesto per la disponibilità e pazienza nel processo di apprendimento in officina e nei cantieri.

Renato per la guida e il supporto per la realizzazione del progetto.