19/12/2023

- Collegamento I2C sui PIN SCL e SDA (pin 19 e 18) + alimentazione Vcc a 5V e GND
- Scaricato la libreria “rtclib”
- Modificato il parametro “Drift” considerando la calibrazione ad ogni un minuto (invece di ogni 2h)
- Eliminazione *print* di output inutili nel codice esempio di rtclib per la lettura tramite I2C dell’orario da RTC modulo
- Sistemato formattazione data e ora con la funzione pad che aggiunge gli 0 riempitivi su giorno, mese, ore, minuti e secondi

20/12/2023

- Creato file in arduino per ricevere il messaggio “1” usando il codice esempio “serialEvent”
- Creato file in Rpi per inviare il messaggio “1” (tramite seriale) ad arduino ogni 60 secondi utilizzando il protocollo ntp. Il codice ogni circa 45 secondi introduce un ritardo pari a 1 secondo.

21/12/2023

- Presa decisione sul come strutturare l’algoritmo di gestione dei processi in Arduino con FreeRTOS. I task saranno eseguiti tramite semafori che gestiscono variabili globali (temperatura, pressione e velocità ventola). Si distinguono i seguenti task:
    1. Un task con priorità maggiore sugli altri per far sincronizzare il modulo RTC con l’orario della RPI.
    2. Un task che ogni 5 secondi (o millisecondi, poi lo decidiamo) esegue delle funzioni secondo il seguente ordine:
        a. Lettura istante temporale da modulo RTC;
        b. Lettura dati da sensore BMP280;
        c. Check sulla temperatura ed eventuale attivazione/disattivazione della ventola;
        d. Inviare i dati tramite comunicazione seriale alla RPI per memorizzare su database. Questo deve avvenire combinando le variabili globali in una stringa.
    3. Un task che si preoccupa di visualizzare su schermo OLED i dati relativi alle variabili globali.

29/12/2023

- Evitato l’utilizzo dei semafori con FreeRTOS in Arduino implementando semplici task con differenti priorità.
- Implementati i task 1 e 2.a decisi in data 21/12/2023. Per il momento, tramite seriale avvengono le seguenti cose:
    1. Arduino manda a RPI l’orario corrente e alcuni messaggi per segnalare l’effettiva esecuzione dei task (verranno tolti in futuro). Questi messaggi vengono visualizzati in RPI tramite monitor seriale dell’IDE Arduino.
    2. RPI manda ogni X secondi il messaggio “1” ad Arduino per la sincronizzazione e quest’ultima viene effettuata con successo (forse).

- Connesso il sensore BMP280 ad Arduino e scaricata libreria dedicata. I collegamenti tra sensore e Arduino sono stati effettuati mediante cavo STEMMA QT che sfrutta il seguente codice colore:
    1. Nero – GND
    2. Rosso – Vcc
    3. Blu – SDA (pin 18 su Arduino)
    4. Giallo – CLK (pin 19 su Arduino)
- Effettuati primi test sul solo sensore di temperatura mediante sketch di esempio "*bmp280test.ino"* eliminando righe inutili al nostro scopo. Sul codice è presente il comando "bmp.setSampling(.....);" per impostare il tipo di campionamento che deve effettuare il dispositivo: *si è deciso di rimandare l'approfondimento di tali parametri*.

02/01/2024

- Connessa la ventola ad Arduino mediante il seguente schema colore:
    1. Nero – GND
    2. Giallo – 5V
    3. Blu – PWM (pin D6tilde di Arduino)
    4. Verde - velocità in RPM (pin A3 di Arduino)
- Implementata nuova versione "*bmp280_v2.ino*" in cui compare il controllo della ventola. Bisogna solo modificare le soglie e correggere eventualmente i parametri di ingresso nella funzione *map().*
- Connesso schermo OLED sulla linea I2C su cui comunicano Arduino e sensore BMP280.
- Implementata la funzionalità di scrittura su schermo OLED di temperatura e pressione.
- **Sorto un problema:** Arduino non riesce ad erogare abbastanza energia per mantenere nella corretta regione di operatività tutti i dispositivi insieme (ventola+display+sensore+RTC).

03/01/2024

- Risolto il problema di erogazione dell'energia per i dispositivi: il modulo RTC
- e la ventola vengono alimentati da RPI mentre BPM280 e display OLED da Aduino.
- Impostate T_soglia = 24°C e valore del massimo nella funzione map() pari a 38°C.
- **Attenzione:** il pin A0 di Arduino è bruciato!
- Iniziato ad implementare in FreeRTOS tutto il sistema di acquisizione e visualizzazione dei dati su OLED e attuazione della ventola (task 2.a, 2.b, 2.c e 3 decisi in data 21/12/2023).
- **Sorto un problema:** errore di inizializzazione del display OLED nello sketch dedicato a FreeRTOS. Dopo varie ipotesi sollevate si è scoperto alla fine della giornata che molto probabilmente FreeRTOS e il display OLED SSD1306 non sono interfacciabili. Prima di richiedere una riunione al professore si necessitano: approfondimenti sul problema, eventuali possibili soluzioni e/o aggiramenti dell'ostacolo.

04/01/2024

- Nessun aggiornamento rilevante da segnalare

05/01/2024

- Nessun aggiornamento rilevante da segnalare

10/01/2024

- ESP32 + BMP sui pin di default I2C -> FreeRTOS con code (/ESP32/esp32_code_v1.ino) [FUNZIONA]

14/01/23

- Nei primi minuti della chiamata si è fatto il punto della situazione con il nuovo dispositivo ESP32 impiegato;
- A fronte della modifica HW si è ritenuto opportuno modificare la distinzione dei task. Si distinguono i seguenti:
    1. SINCRONIZZAZIONE RTC: Un task con priorità maggiore sugli altri per far sincronizzare il modulo RTC con l'orario della RPI.
    2. LETTURA SENSORI: Un task che ogni 5 secondi (o millisecondi, poi lo decidiamo) esegue delle funzioni secondo il seguente ordine:
        a. Lettura istante temporale da modulo RTC *(datetime*)*;*
        b. Lettura dati da sensore BMP280 *(temperatura* e *pressione*)*;*
        c. Check sulla temperatura ed eventuale attivazione/disattivazione della ventola;
    3. INVIO DATI: Un task che si preoccupa di visualizzare su schermo OLED (*temperatura, pressione e* RPM) i dati relativi alla struttura e inviare questi dati (*+datetime*) tramite comunicazione seriale alla RPI per memorizzare su database.

    I task 2 e 3 comunicano tra loro tramite le code. In particolare, il task 2 acquisisce i dati dai vari sensori (e ventola) inviandoli al task 3.

- Generato nuovo file "*esp_code_v2.ino*" che integra la lettura di tutti i moduli (RTC+BMP280) stampando su seriale e display OLED i valori rilevati (task 2.a, 2.b decisi in data 14/01/2023).
- Generato nuovo file "*esp_code_v3.ino*": rispetto alla versione v2 questo implementa il task dedicato alla SINCRONIZZAZIONE RTC (task 1 deciso in data 14/01/2023).

15/01/2024

- Aggiunti resistori per la lettura dei giri al minuto (generato "*esp_code_v4.ino*" ancora non funzionante)

16/01/2024

- Provato a risolvere problema sulla lettura del tacho senza successo. Si è preso in considerazione di utilizzare il pin SQW del modulo RTC al fine di sfruttare il contatore interno di quest'ultimo.
- Provata la connessione seriale tra RPI ed ESP32: tutto funziona.
- Iniziato a vedere la parte successiva del progetto riguardante database e web server.
- Cose da studiare: mariaDB su linux, HTML, CSS, SQL, Flask e Apache.

19/01/2024

- Aggiornamenti su: RTC, TACHO, sito web e database;
- Terminato il file che gira su ESP32. Adesso RTC emette anche degli interrupt ogni secondo per andare a leggere la velocità della ventola ("*esp_code_v5.ino*").

- Modificato il circuito elettrico per la lettura degli RPM della ventola sostituendo il Condenzatore+Resistore con un solo Resistore da 330 Ohm.
- Creato database “SensorData” con tabella “misure”.
- Aggiunti sulla RPI i codici necessari per la lettura dei dati sulla seriale e successivo aggiornamento del dataset “*mariadb_database.py*”.
- Aggiunto codice necessario per la realizzazione del sito web in locale (IP: *http://xxx.xxx.x.xxx:5000*).

20/01/2024

- Creato “esp_code_v6.ino” per andare a selezionare i vari campi da visualizzare nel database. Adesso in seriale si manda l’informazione secondo la struttura: dataora#temperatura#pressione#rpm
- Modificato lettura da seriale (““*mariadb_database.py*”).Adesso si ricevono i valori da pubblicare all’interno di un’unica stringa, la quale contiene i dati separati da un #. In ricezione si utilizza la funzione *split()* per selezionare i campi del database.

22/01/2024

- In attesa della chiamata con il professore si è discusso sulle varie possibilità di implementazione del gestore dell’alert.

24/01/2024

- Dalla chiamata con il professore si procederà utilizzando un semplice controllo sulla temperatura ogni volta che un valore di questa viene ricevuto sulla seriale mediante condizione “if”, prima dell’inserimento del valore stesso nel database.
- Iniziato lo studio e l’implementazione del chatbot Telegram in script “*chatbot.py*”.

25/01/2024

- Implementata funzione di alert per il superamento della temperatura critica. Il chatbot invia con successo un messaggio di allerta a tutti gli utenti registrati appena la temperatura supera la soglia critica.
- Iniziata implementazione della registrazione dell’ID dei nuovi utenti che avviano il bot.
- Iniziata implementazione della selezione dei valori all’interno del database secondo la data richiesta dall’utente.

26/01/2024

- Implementata funzione che manda una immagine con grafico dal bot all’utente. La funzionalità si connette al database prelevando 1 minuto di valori registrati e poi ne crea un grafico.

- Implementata funzionalità per l'aggiunta del chat_id di nuovi utenti al chatbot in un semplice file txt (*bot_users.txt*) (da rivedere: ci sono problemi nel salvataggio dell'id nel file)

31/01/2024

- Uniti i codici del chatbot: file con id e grafico. Lo script del chatbot adesso rimane in attesa dei comandi "/start" e "/grafico" eseguendo le funzionalità correlate:
    - */start* : dopo che l'utente invia tale messaggio al bot quest'ultimo effettua un controllo degli user già connessi ad esso; se l'id dell'utente è nuovo allora aggiorna il file "bot_users.txt" altrimenti viene inviato un messaggio di risposta del tipo "Utente già registrato".
    - */grafico* : alla ricezione di questo comando il bot genera un grafico prendendo i valori presenti del db risalenti agli ultimi 60 secondi registrati.
- Aggiornata la funzionalità del chatbot "/grafico": oltre all'invio del grafico temporale adesso vengono inviati i valori statistici (media, valore minimo, valore massimo) di pressione, temperatura e velocità ventola.
- **ATTENZIONE**: trovata anomalia nell'invio del grafico da parte del chatbot, errore da analizzare per bene.

02/02/2024

- Implementati i bottoni che permettono al client di richiedere i dati relativi a un intervallo di tempo passato, per cui è stata aggiornata la procedura relativa al comando "/grafico":
    1. alla ricezione di questo comando, il bot, tramite la funzione "*interval_options()*", invia un messaggio con pulsanti per selezionare l'intervallo temporale;
    2. la funzione *"button()"* estrae l'oggetto **'CallbackQuery'**, il quale rappresenta la risposta dell'utente a un pulsante
    3. chiama la funzione "*funz_grafico*" passando il valore **'query.data'** come argomento per rappresentare il numero di minuti selezionato dall'utente
    4. Utilizza "*reply_photo*" per inviare l'immagine del grafico all'utente.
- ATTENZIONE: Risolvere anomalia nell'invio del grafico (l'ora sull'asse delle x)

05/02/2024

- Nessun aggiornamento rilevante da segnalare

09/02/2024

- Risolti i problemi relativi alla comunicazione seriale tra ESP32 e RPI:
    - Il messaggio inviato da ESP32 contiene un carattere speciale che permette il riconoscimento dell'inizio stringa; in questo modo vengono scartati tutti i messaggi letti non correttamente.
    - Dal lato dell'ESP32 è stato introdotto un ritardo per permetterne la corretta inizializzazione della comunicazione seriale evitando eventuali blocchi in lettura (in RPI).

    - Siccome sono stati rilevati problemi relativi alla sincronizzazione del modulo RTC allora si è proceduto immettendo una prima sincronizzazione appena la inizializzazione della seriale su RPI.
- Risolti problemi nella sincronizzazione dell'orario del modulo RTC:
    - Il modulo non effettuava una vera sincronizzazione all'orario attuale. Si è risolto l'errore inviando una stringa (del tipo: #sync#YYYY#MM#DD#hh#mm#ss#)
- Risolti problemi del chatbot relativi all'insufficienza di dati disponibili nel DB:
    - Quando vengono richiesti più minuti registrati di quelli effettivamente disponibili il chatbot aggiunge una nota al messaggio per avvisare l'utente dell'indisponibilità.
    - Quando sono presenti dei "buchi" all'interno dell'intervallo richiesto il chatbot aggiunge la stessa nota menzionata nel punto precedente.

01/03/2024

- Implementato codice CSS per lo stile della pagina web. La pagina visualizza al centro un riquadro dinamico al cui interno viene aggiornata automaticamente una tabella contenente tutti i dati del database.

05/03/2024

- Modifica della struttura del sito web: all'accesso della pagina viene visualizzata sulla sinistra la tabella contenente i dati del database; sulla destra vengono mostrati tre riquadri ciascuno dei quali dovrà contenere la media di temperatura, pressione e rpm dell'ultima ora.
- Modifica dello stile della pagina web.

06/03/2024

- Implementato codice per acquisire e visualizzare nella pagina web i valori di temperatura, pressione e rpm nell'ora passata;
- Effettuato primo test su tutto il sistema lanciando tutti i codici a mano (senza routine di servizio) e risolti alcuni piccoli problemi.

08/03/2024

- Commentati i codici su RPI.

11/03/2024

- Commentato il codice su esp32;
- Implementati i demoni per l'avvio automatico di tutti gli script per ciascuno di essi e verifica dell'effettivo funzionamento del sistema.

14/03/2024

- Fatto punto della situazione;
- Controllato ultimi dettagli sul funzionamento del sistema e CONCLUSIONE IMPLEMENTAZIONE;
- Scritta bozza dell’indice e divisione degli argomenti per la scrittura della relazione.