



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA TRIENNALE IN INGEGNERIA
DELL'INFORMAZIONE

Sviluppo di un sistema integrato per la misura di parametri ambientali eterogenei

RELATORE:

Prof. Alessandro Pozzebon

LAUREANDO:

Michele Corbioli

2032528

Anno Accademico 2024/2025

Data di Laurea 22/07/2025

INDICE

INTRODUZIONE	v
1 Tipologia di parametri ambientali misurati e sensori utilizzati	1
1.1 Temperatura, umidità e pressione	1
1.2 Polveri Sottili	2
1.3 Anidride carbonica (CO ₂)	2
1.4 Velocità e direzione del vento	3
1.5 Pioggia	3
2 Algoritmi e tipologia di segnali utilizzati	5
2.1 I2C	5
2.2 UART	6
2.3 SPI	6
3 PROGETTAZIONE SCHEDA ACQUISIZIONE DATI	9
3.1 Microcontrollore	9
3.2 Alimentazione del sistema	10
3.3 Tensioni necessarie per il corretto funzionamento del sistema	10
3.3.1 Convertitori DC-DC	13
3.3.2 Convertitori lineari	14
3.4 Ottimizzazioni software per la gestione dell'energia	14
3.4.1 Intervalli di trasmissione dati	15
3.4.2 Disattivazione Sensori non necessari	16
3.4.3 Sleep Mode	17
3.5 Stato attuale di progetto	18
4 Controllo e Gestione Errori e Guasti	21
4.1 Verifica presenza sensori in fase di avvio	22
4.2 Verifica Valori misurati	22
4.2.1 Pressione	23
4.2.2 Temperatura	23
4.2.3 Pioggia, velocità e direzione del vento	25
4.3 Trasmissione dati di Backup	25

5 Limiti Ricontrati ed Implementazioni future	27
Bibliografia	29
Ringraziamenti	33

INTRODUZIONE

Al giorno d'oggi siamo circondati da onde radio, basti pensare per esempio al wifi di casa, o alla radio dei mezzi di trasporto, al 4G o al 5G che si sta diffondendo sempre più nel mondo.

Ma quanto effettivamente crea danno o beneficio l'essere immersi in queste onde radio?

e c'è un modo per ottimizzare la trasmissione di segnali radio?

Personalmente penso che a queste risposte non potremo dare una risposta nel breve periodo, ma quello che possiamo fare è cominciare da una di esse e provare a cercare delle correlazioni che ci aiutino a capire e comprendere il potenziale di quello che ci circonda.

Quasi per caso sono venuto a conoscenza di un altro progetto in fase di realizzazione che cerca di rispondere alla seguente domanda: in che modo la trasmissione di segnali viene ridotta o interrotta a causa di eventi atmosferici e come possiamo ottimizzarne la trasmissione?

Con lo sviluppo tecnologico e la nascita dell'intelligenza artificiale abbiamo la possibilità di stimare con più certezza e precisione, basandoci su grandi moli di dati, la variazione di segnale ricevuto in diverse condizioni meteo.

L'altro progetto di cui siamo venuti a conoscenza infatti si basa sullo studio di grandi moli di dati di vario tipo e genere che vengono date in input ad un'intelligenza artificiale che li studia e ne cerca le correlazioni.

L'unica cosa che manca all'altro progetto, di fatti, si tratta della raccolta di questa grande mole di dati.

Il progetto quindi nasce con l'obiettivo di raccogliere dei dati meteorologici eterogenei quali temperatura, umidità, pressione, quantitativo di polveri sottili, quantità di pioggia, direzione e velocità del vento.

I dati meteorologici che verranno raccolti saranno poi trasmessi tramite la rete LoRaWAN ad un gateway dell'università di Padova e saranno successivamente dati in input agli algoritmi di intelligenza artificiale in modo da studiare la correlazione tra qualità del segnale ricevuto e le varie condizioni meteo.

L'obiettivo secondario che ci siamo posti è stato quello di alimentare il sistema con

dei pannelli fotovoltaici e quindi dovendo andare a gestire tutta la parte energetica per ottimizzare al meglio le risorse a nostra disposizione.

1 | TIPOLOGIA DI PARAMETRI AMBIENTALI MISURATI E SENSORI UTILIZZATI

Come abbiamo avuto modo di leggere nell'introduzione andremo a misurare diversi parametri ambientali di origini differenti e dovremo farlo in maniera più precisa e più efficiente possibile.

Andiamo a vedere nel dettaglio i parametri ambientali che andremo a raccogliere ed i sensori che useremo.

1.1 Temperatura, umidità e pressione

La temperatura è un parametro ampiamente utilizzato in vari ambiti, alcuni esempi potrebbe essere il frigo in cucina, la lavatrice in bagno o, per cambiare contesto, la carburazione dei motori a scoppio e molti altri. Per la misurazione della temperatura ci sono infatti diversi sensori con range di misura differenti.

Nel nostro caso abbiamo deciso di optare per l'AHT20, un sensore abbastanza economico e preciso, che misura da -40°C a 80°C con una risoluzione di 0.01°C ed un'accuratezza di $\pm 0.3^{\circ}\text{C}$.

L'AHT20 misura inoltre l'umidità presente nell'aria in un range da 0 %RH a 100 %RH con una risoluzione di 0.024 %RH ed una accuratezza di ± 2 %RH [1].

La pressione invece è un parametro spesso sottovalutato, ma nella pratica non sono nient'altro che le variazioni di pressione che creano i venti. Per la misura della pressione siamo ricaduti sul BMP280 che ha un range di valori da 300 a 1100 hPa (che equivalgono rispettivamente ad un range da +9000 m fino a -500 m sul livello del mare) un'accuratezza di $\pm 0,12$ hPa che equivale a ± 1 m ed una risoluzione di 0,16 Pa [2].

Con questi 2 sensori che spesso vengono venduti insieme nelle schede di sviluppo (breakout board) raccogliamo già 3 valori tipici per le stazioni meteorologiche.

1.2 Polveri Sottili

Le polveri sottili sono chiamate in questo modo appunto perché sono molto fini (sotto i 10 micrometri) e non vengono filtrate dal nostro organismo quando respiriamo, ma finiscono direttamente nei bronchi causando infezioni, in alcuni casi anche gravi. Per questo motivo abbiamo deciso di misurarle nel nostro progetto.

Le polveri sottili vengono classificate in base al diametro delle stesse solitamente in 4 gruppi:

1. PM10: le polveri con diametro inferiore a 10 μm
2. PM4: le polveri con diametro inferiore a 4 μm
3. PM2.5: le polveri con diametro inferiore a 2,5 μm
4. PM1: le polveri con diametro inferiore ad 1 μm

Come sensore per le polveri sottili abbiamo scelto il sensore SPS30 della Sensirion [3]. Il sensore in questione riesce a misurare fino a 1000 $\mu\text{g}/\text{m}^3$ di quantitativo di polveri sottili nei 4 gruppi sopra indicati, rispettivamente con un'accuratezza di:

- PM1 e PM2.5 da 0 a 100 $\mu\text{g}/\text{m}^3$ accuratezza di $\pm 10 \mu\text{g}/\text{m}^3$
- PM4 e PM10 da 0 a 100 $\mu\text{g}/\text{m}^3$ accuratezza di $\pm 25 \mu\text{g}/\text{m}^3$

1.3 Anidride carbonica (CO2)

L'anidride carbonica è un gas naturale presente nell'atmosfera ed abbiamo deciso di misurarlo per vedere se con elevati livelli di CO2 si verifica una diminuzione della qualità della trasmissione.

Per l'anidride carbonica abbiamo scelto il sensore SCD41 che misura da 0 a 40.000 ppm (particelle per metro cubo) con un'accuratezza di $\pm (40 \text{ ppm} + 5\%)$ [4].

Il sensore SCD41 comunica in I2C con il microcontrollore, dando possibilità di ricevere i valori misurati senza dover andare nelle specifiche tecnico-fisiche della modalità di misura.

1.4 Velocità e direzione del vento

La direzione e la velocità del vento possono essere utili anche per capire le possibili variazioni meteorologiche cicliche. Per misurare la velocità (tramite un anemometro) e la direzione (tramite un anemoscopio) ci siamo basati sui due strumenti di misura già presenti in laboratorio.

L'anemometro ha un funzionamento molto semplice, sulla parte mobile è presente un magnete che fa aprire un contatto reed (un tipo di sensore utilizzato anche per gli allarmi dove il magnete spinge la lamella che forma il contatto, andando così ad chiudere il contatto nel momento della presenza del magnete). Il vento facendo girare la parte mobile dell'anemometro fa aprire e chiudere il contatto reed e conteggiando il tempo di apertura e chiusura di questo contatto riusciamo a calcolarne la velocità.

L'anemoscopio invece è basato sempre sui contatti reed ed un magnete sulla parte mobile dello strumento. La parte fissa è divisa in 8 sezioni, ad ogni sezione è presente un contatto reed ed una resistenza ben definita. L'asse della parte mobile a seconda della posizione sul piano andrà tramite il magnete a chiudere al più due contatti reed (due nel caso in cui si trovasse nel mezzo tra 2 sezioni) ed analizzando il valore della resistenza in maniera indiretta (tramite un partitore resistivo e andando a leggere la tensione ai capi dello strumento) riusciamo ad individuare le 16 zone che si vengono a creare (8 delle quali grazie alle singole resistenze dei contatti reed ed altre 8 con le combinazioni in parallelo della chiusura di 2 contatti reed) [5].

Ovviamente l'anemoscopio andrà installato prendendo un riferimento per il nord (in base alla tensione che avremo ai capi dello strumento) ma quest'ultimo verrà impostato in fase di installazione.

1.5 Pioggia

Per misurare la Quantità di pioggia è necessario l'utilizzo di un pluviometro e dopo una ricerca in letteratura abbiamo trovato il SEN0575 della DFRobot [6].

Il sensore sopra citato è composto dalla scheda e dalla vasca con delle misure ben precise che convoglia l'acqua ad una leva basculante forata, tappata sul fondo (che grazie al peso dell'acqua al suo interno oscilla verso il basso e torna in posizione tramite un contrappeso).

La leva presente all'interno dello strumento fa chiudere un contatto reed. Questa variazione di stato viene misurata dal microcontrollore presente sulla scheda dello strumento che comunica tramite I2C i dati raccolti di quantitativo di pioggia in mm e quantitativo orario di pioggia in mm/h in 1 m² di superficie.

2 | ALGORITMI E TIPOLOGIA DI SEGNALI UTILIZZATI

Nel nostro progetto utilizziamo diversi protocolli di comunicazione, dei quali qui di seguito ho riportato delle brevi descrizioni.

2.1 I2C

I2C, ovvero Inter-Integrated Circuit, è un'interfaccia di comunicazione seriale sincrona che venne sviluppata da Philips (attualmente NXP) durante gli anni '80. I2C è stata progettata affinché i microcontrollori potessero comunicare con delle periferiche a brevi distanze con l'utilizzo di un sistema a due linee: SDA (Serial Data Line) e SCL (Serial Clock Line) [7].

Quando si tratta degli aspetti specifici dell'I2C, una delle caratteristiche più rilevanti del protocollo è la possibilità di gestire molteplici dispositivi sullo stesso bus tramite un sistema di indirizzamento. Infatti, ciascun dispositivo ha un indirizzo univoco (comunemente 7 bit, ma esistono anche versioni a 10 bit) che consente al Master di comunicare con uno specifico Slave.

Inoltre, è importante sottolineare che il protocollo è compatibile con molteplici master, ma in applicazioni comuni si utilizza un singolo master con molteplici slave.

La comunicazione inizia dal master, che trasmette un segnale di start, quindi l'indirizzo dello slave e un bit che dice se sta effettuando una lettura o una scrittura. Lo slave risponde con un segnale di acknowledgment (ACK). I dati sono spediti un byte per volta, seguiti da un ACK dal ricevente. Infine, la comunicazione viene interrotta attraverso un segnale di stop trasmesso dal master.

Il protocollo I2C ha diverse velocità standard:

- Standard Mode: fino a 100 kbit/s;
- Fast Mode: fino a 400 kbit/s;
- Fast Mode Plus: fino a 1 Mbit/s;
- High-Speed Mode: fino a 3.4 Mbit/s.

Nel nostro progetto viene utilizzata la Fast Mode fino a 400kHz e l'utilizzo di questo protocollo è stato scelto per la sua facilità di collegamento e la sua implementazione in molti sensori presenti in commercio.

2.2 UART

La comunicazione UART (Universal Asynchronous Receiver/Transmitter) è un'interfaccia seriale asincrona che utilizza due linee, TX (trasmissione) e RX (ricezione), per consentire la trasmissione di dati tra due dispositivi. Non richiede un clock condiviso, il che la rende semplice da implementare e affidabile per comunicazioni punto-a-punto [8].

La UART è notevolmente più limitata di altri protocolli per quanto concerne velocità e numero di dispositivi supportati. Essa, tuttavia, è estremamente robusta e semplice da gestire, sia dal punto di vista hardware che software, e difatti nel nostro progetto, La UART viene utilizzata come canale di backup, attivandosi solo nel caso in cui la trasmissione dei valori tramite la rete LoRaWAN non fosse disponibile.

2.3 SPI

La comunicazione SPI (Serial Peripheral Interface) è un protocollo seriale sincrono ad alta velocità, progettato per il trasferimento rapido di dati tra un dispositivo master e uno o più slave [9].

La comunicazione SPI utilizza 4 linee principali:

- MOSI (Master Out Slave In): trasmette i dati dal master allo slave
- MISO (Master In Slave Out): invia i dati dallo slave al master
- SCLK (Serial CLock): clock base per la comunicazione
- SS (Slave Select): seleziona lo slave con cui comunicare

Nel nostro progetto, utilizziamo il protocollo SPI per trasmettere i valori raccolti dai vari sensori alla scheda di comunicazione (RFM95), che si occupa poi di inviarli tramite la rete LoRaWAN.

3 | PROGETTAZIONE SCHEDA ACQUISIZIONE DATI

3.1 Microcontrollore

La parte di ricerca dei componenti è stata molto ricca di soluzioni e durante la fase di test di queste soluzioni abbiamo deciso di utilizzare l'ATMEGA32L come microcontrollore per gestire tutto il funzionamento del progetto [10].

La scelta di questo controllore è dovuta all'elevato numero di ingressi/uscite programmabili (32), i quali ci danno una grande libertà di movimento.

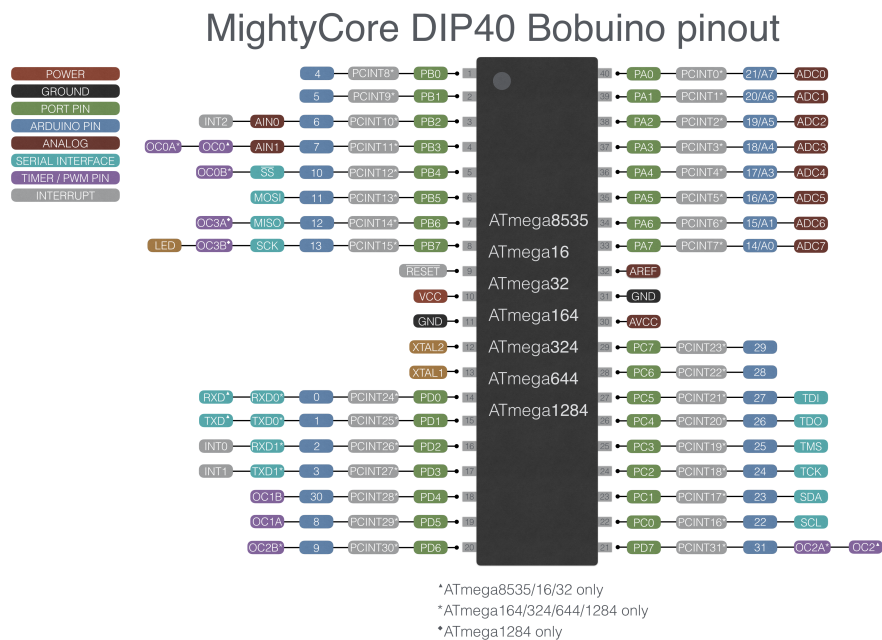


Figura 3.1: Pinout microcontrollore[11]

Avendo scelto questo microcontrollore abbiamo la possibilità di controllare più ingressi e uscite per decidere come gestire dal punto di vista energetico i diversi sensori, andando ad escluderli nel momento del bisogno.

3.2 Alimentazione del sistema

Per questioni pratiche per alimentare il sistema abbiamo deciso di utilizzare 2 pannelli fotovoltaici presenti in laboratorio e di utilizzare un regolatore di carica (CMTD-A2420) che ci dà in uscita una tensione di 12 V in corrente continua. Sull'ingresso dei pannelli collegheremo i 2 pannelli fotovoltaici in Serie, sull'ingresso della batteria installeremo la batteria di accumulo da 12 V 4,5 Ah. Sulla terza coppia di terminali invece installeremo il carico che sarà il nostro progetto da alimentare [12].

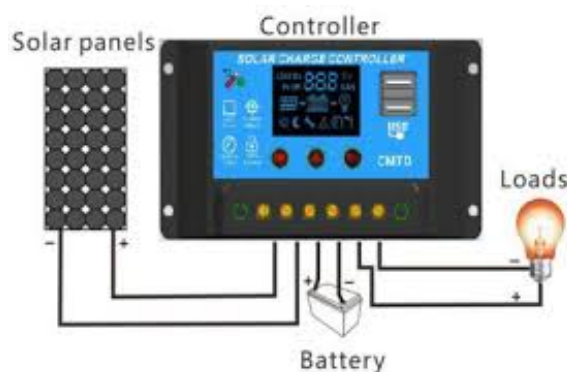


Figura 3.2: collegamento elettrico regolatore di carica

Utilizzando in questo modo il regolatore di carica noi avremo in uscita sul carico una tensione che corrisponderà alla tensione della batteria di accumulo, quindi 12 V.

3.3 Tensioni necessarie per il corretto funzionamento del sistema

Come abbiamo visto nella sezione precedente Il regolatore di carica ci dà a disposizione una tensione di 12 V (nominali) dalla quale dovremo derivare le tensioni di lavoro del microcontrollore e dei singoli sensori.

Il microcontrollore può lavorare in un range di tensione che va da 2,7 V a 5,5 V il che ci dà abbastanza gioco per muoverci con le tensioni.

Il sistema di misura della velocità del vento è un semplice contatto che si apre e chiude, quindi possiamo scegliere noi le tensioni di lavoro.

Per quanto riguarda il modulo LoRaWAN ho predisposto sulla scheda dei pin da

saldare a seconda della tensione di lavoro del modulo RFM95 (essendoci modelli alimentati a 3,3 V ed a 5 V).

L'alimentazione della sensoristica è suddivisa come mostrato in tabella 3.1.

Sensori a 3.3V	Sensori a 5V
• AHT20 (Temperatura e Umidità) • BMP280 (Pressione) • SCD41 (CO2) • SEN0575 (Pluviometro)	• SPS30 (Polveri sottili) • Direzione del vento (per ridurre possibili errori di lettura sul partitore resistivo)

Tabella 3.1: Suddivisione delle alimentazioni dei sensori

Come è possibile notare non ci sono componenti che Funzionino a 12 V, quindi bisogna trovare il modo per abbassare la tensione in maniera indipendente dalla corrente assorbita da 12 V a 5 V ed a 3.3 V, ed inoltre essendo un impianto ad isola bisogna essere il più efficienti possibile.

Per poter raggiungere questo risultato si potevano percorrere 2 strade:

1. Passare da 12 V a 5 V ed a 3.3 V (12 V -> 5 V & 12 V -> 3.3 V)
2. Passare da 12 V a 5 V e passare da 5 V a 3.3 V (12 V -> 5 V & 5 V -> 3.3 V)

Per ridurre i rumori sul bus I2C ed ottimizzare il sistema in generale abbiamo scelto la seconda opzione.

3.3.1 Convertitori DC-DC

I DC-DC converter sono dei sistemi che come fa intuire il nome convertono la tensione continua da un livello all'altro [13].

Questi tipi di convertitori si dividono generalmente in 2 famiglie come mostrato in tabella 3.2.

Buck DC-DC Converter	Boost DC-DC Converter
Buck	Boost
Il buck converter é utilizzato per ottenere in uscita una tensione più bassa di quella in ingresso	Il boost converter è utilizzato per ottenere in uscita una tensione più alta di quella in ingresso

Tabella 3.2: Tipologia di DC-DC converter

Come penso sia ovvio nel nostro caso è necessario utilizzare un buck converter, che ci abbassi la tensione da 12V a 5V. Questo tipo di circuito sfrutta le proprietà fisiche dei componenti elettrici come induttanze e condensatori per aumentare l'efficienza della conversione arrivando anche oltre il 98%.

3.3.2 Convertitori lineari

Per la conversione da 5V a 3.3V invece abbiamo deciso di utilizzare un convertitore lineare. Quest'ultimo non fa nient'altro che dissipare l'energia in più in calore, diventando meno efficiente, ma più stabile ai disturbi.

La scelta di utilizzare un convertitore lineare è dovuta al fatto che il bus I2C è alimentato a 3.3V, e la riduzione di disturbi rende più sicura la comunicazione tra microcontrollore e sensori. Inoltre i sensori alimentati a 3.3V sono quelli che richiedono meno energia per funzionare quindi la perdita di energia in percentuale su tutto il sistema è minima.

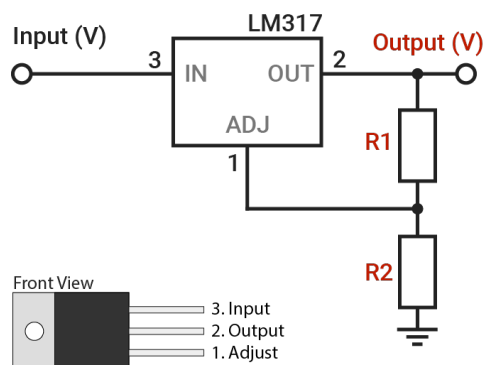


Figura 3.3: Esempio di collegamento convertitore lineare

3.4 Ottimizzazioni software per la gestione dell'energia

Ovviamente essendo un sistema autonomo non ci siamo limitati a ricercare solo i componenti giusti in modo da rendere l'intero sistema più conservativo, ma ci siamo messi ad operare a livello software ed hardware.

Infatti abbiamo implementato l'utilizzo di MOSFET per il controllo dell'alimentazione dei sensori in modo da alimentarli solo nel momento del bisogno ed inoltre abbiamo implementato un controllo della tensione della batteria in modo da adattare il tempo di trasmissione in base alla tensione della batteria.

3.4.1 Intervalli di trasmissione dati

Un intervallo di trasmissione dati variabile è stata la prima implementazione che abbiamo fatto. Per farlo andiamo a misurare la tensione della batteria tramite una misura indiretta su un partitore resistivo.

Il partitore resistivo è composto da una resistenza da 8,2 kΩ ed una resistenza da 3,9 kΩ come nello schema qui sotto.

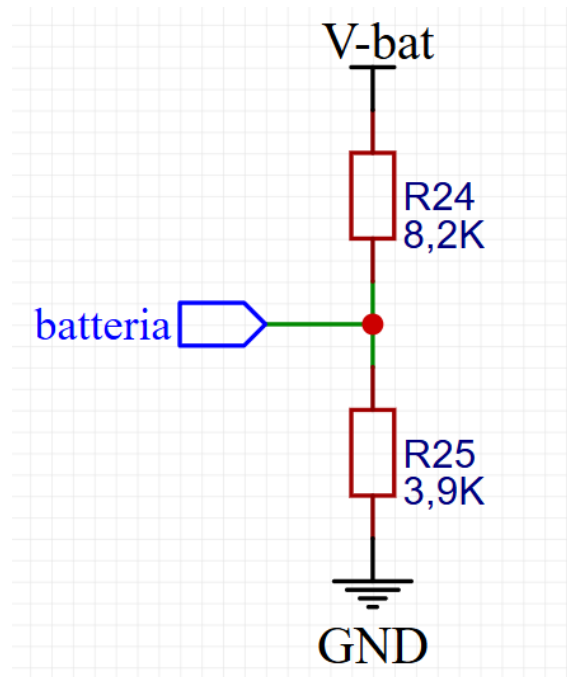


Figura 3.4: Schema elettrico del partitore resistivo per misurare la tensione della batteria

Questo partitore ci dà ai capi della resistenza da 3,9 kΩ una tensione così calcolata:

$$V_{mis} = V_{bat} * 3900 / (8200 + 3900) = V_{bat} * 3,9 / 12,1 \quad (3.1)$$

Dove V_{mis} sta ad indicare la tensione misurata dal partitore resistivo.

Nel caso in cui la batteria sia completamente carica, quindi con una tensione di 14,4 V questo partitore resistivo restituisce un valore di:

$$V_{mis} = V_{bat} * 3,9 / 12,1 = 14,4 * 3,9 / 12,1 = 4,64 \text{ V} \quad (3.2)$$

Questo calcolo ci mostra che anche a batteria completamente carica la tensione misurata non supera i 5 V e che ci lascia un pò di margine per verificare che il controllore dei pannelli solari non carichi troppo la batteria.

Nel caso in cui invece la batteria sia completamente scarica, quindi con una tensione intorno a 10,5 V questo partitore ci permette di misurare una tensione

di:

$$V_{mis} = V_{bat} * 3,9/12,1 = 10,5 * 3,9/12,1 = 3,38 \text{ V} \quad (3.3)$$

Così facendo abbiamo un range di variazione della tensione misurata.

Linearizzando il rapporto tra tensione della batteria e quantità di carica in essa contenuta possiamo decidere il tempo di riposo del nostro controllore. Abbiamo deciso che normalmente le misure verranno prese ogni 15 minuti, ma nel caso in cui la tensione misurata dal partitore resistivo sia sotto i 4 V allora possiamo adattare il tempo tra una misura e l'altra ed impostarlo a 30 minuti, oppure nel caso in cui sia sotto i 3,5 V fare la misura ogni 45 minuti.

Questo metodo di gestione ci garantisce che durante le ore notturne nei periodi più bui la batteria non vada a scaricarsi troppo e continui a funzionare all'interno del range corretto di tensione.

3.4.2 Disattivazione Sensori non necessari

Un'altra implementazione che abbiamo fatto è stata a livello dei sensori: in questo caso andiamo ad accendere e spegnere i singoli sensori solo nel momento della misura in modo da ridurre al minimo il tempo di accensione dei singoli sensori.

Questa implementazione per quanto possa sembrare magari banale per alcuni sensori come l'anemometro o l'anemoscopio in realtà risulta essenziale nei momenti di scarsità di energia a disposizione.

L'anemoscopio è composto da diverse resistenze a seconda della direzione del vento. Questo però implica che almeno 1 resistenza è sempre collegata al circuito di alimentazione. Nel caso in cui ci siano 2 resistenze dell'anemoscopio collegate in parallelo al circuito di alimentazione queste creano una resistenza equivalente secondo la formula:

$$R_{eq} = R1 + R2/(R1 * R2) \quad (3.4)$$

Da questa formula è possibile notare fin da subito che l'assorbimento di corrente aumenterà e quindi andrà a ridurre la durata della batteria.

Implementando un controllo dell'alimentazione con un MOSFET riusciamo ad avviare i sensori nel momento di fare la misura (o prima nel caso del sensore di polveri sottili che deve fare la pulizia dalla polvere depositata all'interno del sensore per poter effettuare una misura corretta).

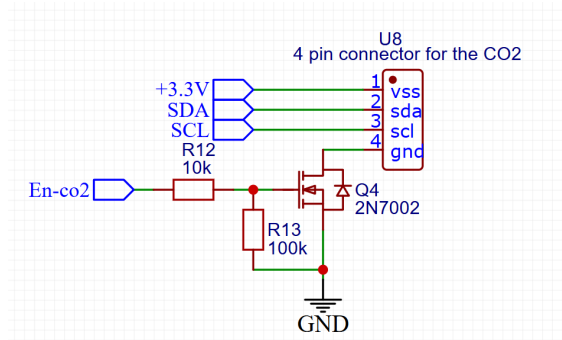


Figura 3.5: Schema di esempio dei collegamenti elettrici per i sensori

3.4.3 Sleep Mode

Un'altra implementazione che abbiamo fatto è stato aggiungere un RTC (real time clock) in modo da impostare un allarme che "svegli" il microcontrollore nel momento in cui ci sia il bisogno di fare una misura.

Con l'implementazione del real time clock possiamo mandare il microcontrollore in sleep profondo in modo che non consumi quasi niente e rimanga in attesa dell'interrupt da parte del RTC.

L'impostazione dell'allarme sul RTC avviene appena prima dell'ingresso in modalità sleep del microcontrollore ed avviene subito dopo la verifica della tensione della batteria. In questo modo è il real time clock a tenere il tempo tra una misura e l'altra e ci garantisce una tempistica perfetta.

3.5 Stato attuale di progetto

Il progetto allo stato attuale è ancora nella fase di sviluppo ed ottimizzazione. Recentemente è stato mandato in stampa un primo prototipo di una scheda sulla quale verranno saldati i fili ed il microcontrollore.

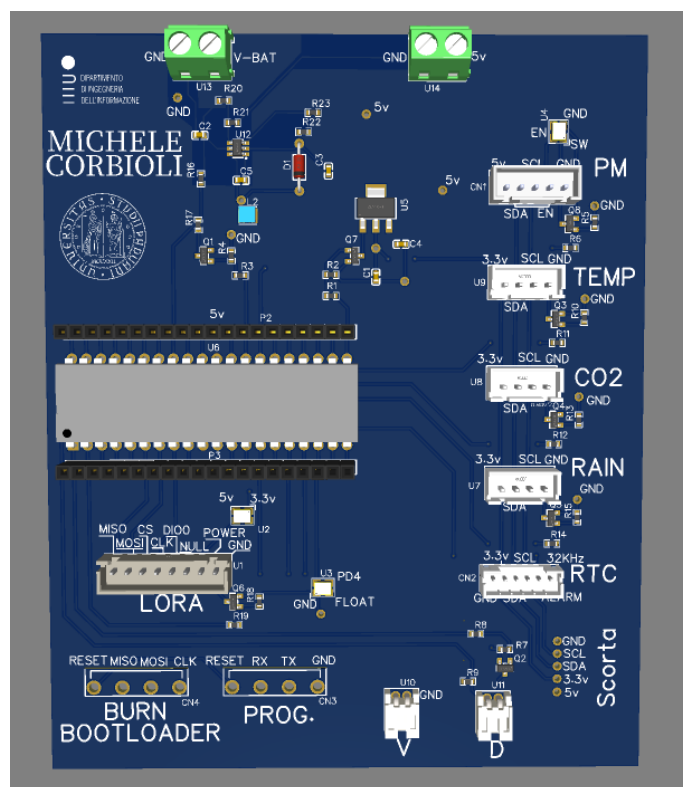


Figura 3.6: Render 3D della scheda

Le dimensioni attuali della scheda sono leggermente grandi per il progetto in maniera tale da poter apportare modifiche se necessario.

Per rimanere nell'ottica delle possibili modifiche che potrebbero venire applicate all'attuale scheda sono stati predisposti dei connettori femmina in maniera tale da poter collegare e visionare il corretto funzionamento di quest'ultima.

Sempre per una questione di predisposizione a possibili modifiche sono stati disposti dei componenti che non verranno mai saldati per dare la possibilità di scelta delle diverse configurazioni che possono essere attuate.

Un esempio di queste predisposizioni è l'alimentazione del modulo RFM95 per la trasmissione dei dati tramite LoRaWAN, infatti quest'ultima a seconda dei modelli necessita di un'alimentazione a 5V o a 3.3V.

Nell'angolo in basso a destra è stata predisposta una connessione sul bus in I2C in maniera tale da poter mantenere la scheda anche nel caso in cui si rovinassero alcuni punti di saldatura.

4 | CONTROLLO E GESTIONE ERRORI E GUASTI

Una parte importante per il progetto alla quale normalmente non si presta attenzione inizialmente è stata la ricerca e la gestione degli errori di comunicazione e dei possibili guasti che potrebbero accadere ai sensori.

A inizio codice, nella fase di setup abbiamo implementato una verifica della presenza e della corretta comunicazione con i vari sensori collegati in I2C.

Per poter effettuare questa verifica dei sensori proviamo ad impostare i valori delle unità di misura che vogliamo che ci restituiscano e se non abbiamo risposta continuiamo a riprovare 1 volta ogni secondo per 30 secondi.

Se in 30 secondi non otteniamo alcun tipo di risposta escludiamo la raccolta dei dati dal sensore in questione.

Facendo in questo modo possiamo escludere i sensori non comunicanti.

Di seguito illustrerò i vari modi per distinguere se un sensore non comunica in base alle librerie che sono state utilizzate per la comunicazione con il sensore stesso.

Se un sensore non comunica impostiamo il microcontrollore a leggere l'indirizzo 0 della cella di memoria, andando a resettare il microcontrollore che, essendo ad 1 solo processore, esegue il codice all'indirizzo 0 della memoria e ricomincia da capo il ciclo di setup.

Impostando il microcontrollore a leggere l'indirizzo 0 della memoria ed a rifare il setup facciamo rifare il ricontrollo della presenza della corretta comunicazione con i sensori.

$$\text{void}(*resetFunc)(\text{void}) = 0; \quad (4.1)$$

Questa è la riga di codice che fa leggere il valore della cella di memoria all'indirizzo 0.

4.1 Verifica presenza sensori in fase di avvio

Come descritto prima nella fase iniziale cerchiamo la corretta comunicazione con i sensori collegati in I2C.

Se il sensore non comunica continuiamo a provare ad impostarlo per 30 secondi.

Se dopo 30 secondi il sensore non risponde andiamo ad escluderlo in maniera tale che non crei problemi nella fase di loop.

Qui di seguito viene mostrato un pezzo di codice come esempio:

```
//status_aht = aht.begin();
status = status | aht.begin();
while ((status & 1)==0)    /* !status_aht
{
    Serial.println(F("non sono riuscito a trovare un sensore aht20"));

    if (check_cable >= 30)
    {
        Serial.println(F("bipasso"));
        break;
    }
    delay(1000);
    status = status | aht.begin();
    check_cable++;
}
```

4.2 Verifica Valori misurati

Nella fase di testing iniziale su breadboard abbiamo scoperto che il microcontrollore una volta risvegliato dalla fase di sleep non eseguiva il setup.

Questo fatto di non eseguire il setup rende l'avvio e l'acquisizione dei dati molto più veloce, però non veniva più eseguita la parte iniziale di ricerca dei sensori non comunicanti.

Non avendo modo di verificare che i sensori comunicassero correttamente e senza far rieseguire in continuazione la parte di setup iniziale che richiede del tempo dove tutti i sensori sono accesi, abbiamo deciso di trovare una soluzione alternativa.

Studiando le singole librerie abbiamo scoperto cosa ci veniva restituito nel momento della mancata comunicazione con il sensore, ed abbiamo deciso di reindirizzare il microcontrollore ad eseguire l'indirizzo 0 della memoria ogni volta che si verificava un problema con la comunicazione.

Facendo in questo modo siamo riusciti a far eseguire la parte di Setup iniziale solamente quando è necessario eseguirla e risparmiare energia ogni qualvolta fosse possibile senza trasmettere valori errati.

4.2.1 Pressione

Per il sensore di pressione (BMP280) abbiamo scoperto che la libreria restituiva un valore di pressione negativo nel momento in cui il sensore non comunicava, quindi la soluzione per capire che il sensore non comunica o risponde correttamente è stata un semplice if.

Andando quindi a controllare il valore che ci viene restituito dalla libreria riusciamo a capire, se negativo, che il sensore non comunica correttamente e resettiamo il microcontrollore in maniera tale da reimpostare il sensore e/o escluderlo se difettoso.

Questa soluzione banale è stata utile nella fase di testing in quanto abbiamo riscontrato che uno dei sensori che avevamo comprato (difettoso) non comunicava più dopo 3 minuti di alimentazione.

4.2.2 Temperatura

Per il sensore della temperatura ed umidità (AHT20) abbiamo scoperto che la libreria ci restituisce l'ultimo valore che aveva ricevuto dal sensore.

Sfruttando questa informazione abbiamo creato un array degli ultimi valori della temperatura per verificare se il sensore comunica correttamente.

L'array è composto da 4 elementi ed eseguendo una lettura ogni 15 minuti si riesce a mantenere uno storico della temperatura di un'ora.

Nell'arco di un'ora è tendenzialmente impossibile che la temperatura rimanga costante al decimo di grado.

Viene riportata qui sotto la sezione di codice che controlla ed aggiorna lo storico dei valori della temperatura.

```
int16_t actual_temperature = (int16_t)(temperatura * 100);

if ((old_temperature[0] == old_temperature[1]) &&
    (old_temperature[1] == old_temperature[2]) &&
    (old_temperature[2] == old_temperature[3]) &&
    (old_temperature[3] == actual_temperature))
{
    resetFunc(); // call reset
}

// aggiorniamo le misure precedenti
old_temperature[0] = old_temperature[1];
```

```
old_temperature[1] = old_temperature[2];  
old_temperature[2] = old_temperature[3];  
old_temperature[3] = actual_temperature;
```

4.2.3 Pioggia, velocità e direzione del vento

Per Il sensore della pioggia non siamo ancora riusciti a trovare un modo per verificare la corretta comunicazione ed il corretto funzionamento dello stesso.

Per l'anemometro non c'è un modo per verificarne il corretto funzionamento in quanto andiamo a contare la quantità di volte che il contatto reed si chiude. Facendo in questo modo noi calcoliamo la velocità del vento basandoci su una stima di circa 30 secondi per rappresentare il valore di 15 minuti.

Per l'anemoscopio invece andiamo a verificare un guasto verificando la tensione presente sul partitore resistivo. Facendo in questo modo riusciamo a capire che si è interrotto il collegamento con il sensore perché ai capi del partitore avremo la tensione di alimentazione.

Quest'ultima soluzione non è ancora stata implementata a livello software ma provvederemo ad integrarla nella prossima versione.

4.3 Trasmissione dati di Backup

E' stata predisposta una trasmissione dei dati raccolti e dei log di lavoro sulla UART del microcontrollore. In questo modo nel momento in cui non riceviamo alcun dato dalla rete LoRaWAN riusciamo a verificare il corretto funzionamento. La comunicazione Tramite UART rende possibile la verifica anche dei sensori e del loro corretto funzionamento.



Figura 4.1: Connettore UART per raccolta dati di backup

5 | LIMITI RISCONTRATI ED IMPLEMENTAZIONI FUTURE

Il primo limite che abbiamo riscontrato è stata la scarsità di memoria a disposizione per il codice, forzandoci a modificare alcune librerie per togliere le parti non necessarie.

Questo limite è ancora presente e probabilmente saremo costretti a cambiare microcontrollore o ad implementarne un secondo.

La sostituzione o l'aggiunta si rende necessaria per poter raggiungere i livelli di efficienza prestabiliti, andando a spegnere i sensori non necessari e per poter implementare algoritmi di controllo e riconoscimento di errori e guasti.

Un altro limite che abbiamo riscontrato lo troviamo a livello hardware, in quanto per alcuni sensori come l'anemometro non è possibile verificarne il corretto funzionamento ed è quindi impossibile riuscire ad implementarne un controllo per possibile guasto.

Come implementazione futura c'è l'idea di installare un controllo sulle tensioni di alimentazione della scheda e predisporre delle linee di backup nel caso in cui un componente SMD si bruciasse.

Un'altra implementazione è quella di un connettore per mettere a terra tramite il palo sul quale verrà installato l'impianto la pista GND della scheda, andando a ridurre possibili guasti causati da scariche elettriche.

BIBLIOGRAFIA

- [1] Datasheet AHT20 (sensore temperatura e umidità)
https://static.maritex.eu/file/display/RNvX5GenZti93oVcmXPk9n_PKbFzX2F0/AHT20.pdf
- [2] Datasheet BMP280 (sensore pressione)
<https://www.alldatasheet.com/datasheet-pdf/view/1132069/BOSCH/BMP280.html>
- [3] Datasheet SPS30 (sensore di polveri sottili)
<https://www.alldatasheet.com/datasheet-pdf/view/1284766/SENSIRION/SPS30.html>
- [4] Datasheet SCD41 (sensore di CO2)
<https://www.alldatasheet.com/datasheet-pdf/view/1425096/SENSIRION/SCD41.html>
- [5] Datasheet Anemoscopio (direzione del vento)
https://cdn.sparkfun.com/assets/d/1/e/0/6/DS-15901-Weather_Meter.pdf
- [6] Datasheet SEN0575 (pluviometro)
https://wiki.dfrobot.com/SKU_SEN0575_Gravity_Rainfall_Sensor
- [7] Protocollo I²C da Wikipedia-en
<https://en.wikipedia.org/wiki/I2C>
- [8] Protocollo UART da Wikipedia-en
https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
- [9] Protocollo SPI da Wikipedia-en
https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
- [10] Datasheet ATMEGA32L (microcontrollore)
<https://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf>
- [11] Programmazione ATMEGA32L e pinout (microcontrollore)
<https://github.com/MCUdude/MightyCore>

- [12] Datasheet Regolatore di carica
<https://www.photovoltaikeforum.com/core/file-download/86613/>
- [13] Funzionamento DC-DC converter
https://en.wikipedia.org/wiki/Buck_converter

RINGRAZIAMENTI

Desidero ringraziare il mio relatore, Prof. Alessandro Pozzebon, che mi ha supportato e guidato durante il progetto; senza di lui questa tesi sulla misura di parametri ambientali eterogenei non sarebbe stata possibile.

Un pensiero a tutti i miei amici che mi hanno accompagnato durante questi anni di università, in particolare mia cugina Gloria che mi ha spinto a dare il massimo. La loro vicinanza mi ha permesso di vivere appieno questo percorso, condividendo i sacrifici sotto sessione e i brindisi dopo gli esami.

Sono grato all'azienda in cui lavoro che mi ha dato la possibilità di concludere gli studi e mi ha permesso di avere dei colleghi che mi hanno aiutato nel momento del bisogno con consigli e aperitivi.

Un bacio alla ragazza che mi è vicina in questo ultimo periodo complesso ma ricco di nuove scoperte.

Un abbraccio ai miei fratelli, sempre presenti e pronti a dare un consiglio o una pacca sulla spalla, a seconda dei casi.

Infine, un grande grazie ai miei genitori, a cui dedico questo lavoro. Senza i loro sacrifici e i loro consigli mai sarei arrivato a questo traguardo.