Alma Mater Studiorum - Università di Bologna
Campus di Cesena

Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione
"Guglielmo Marconi"

Corso di laurea in Ingegneria Elettronica per l'Energia e l'Informazione

Progetto di un sistema IoT a microcontrollore per il telerilevamento di impianti industriali

Elaborato in: Elettronica dei Sistemi Digitali

Relatore:
Aldo Romani

Presentato da:
Osimi Silvio

Correlatore:
Claudio Caldarigi

A.A. 2019/2020

Sessione II – Appello II

Abstract

La rapida evoluzione delle tecnologie elettroniche, la concorrenza e le nuove esigenze che continuamente si presentano sul mercato hanno reso indispensabile un sempre maggiore sviluppo di approcci innovativi nell'attività economica odierna. Per questo il tirocinio curricolare svolto presso l'attività dell'Ing. Claudio Caldarigi si presta a rappresentare i tempi odierni, dove anche le piccole imprese del territorio provinciale iniziano a valutare seriamente la possibilità di migliorare la loro competitività sul mercato ed il loro approccio verso il cliente tramite tecnologie innovative e creative che fino a qualche tempo fa non sarebbero state prese seriamente in considerazione.

L'attività dell'Ing. Caldarigi è uno studio di ingegneria che si occupa soprattutto di installazione di impianti industriali e civili di produzione di energia fotovoltaica, nasce di per sé come realtà che guarda con grande fiducia all'innovazione e scommette sul fatto che il progresso tecnologico possa virtualmente e realmente migliorare la società in cui ci troviamo a vivere. Grazie a questa fiducia nel futuro e nelle potenzialità dell'ingegneria, ed ispirato dai valori e le prospettive poco sopra elencate, l'Ing. Caldarigi ha deciso di dedicare una parte dei suoi sforzi alla realizzazione di un progetto che con grande aspettativa lo aiuterà nell'attività di consulenza e gli permetterà di proporre ai suoi clienti la miglior soluzione a fronte della minima spesa.

Sulla base di quanto detto, il progetto consiste nello sviluppo di un sistema di telerilevamento capace di monitorare soprattutto i consumi energetici degli impianti e la produzione di energia dai sistemi di generazione fotovoltaica che consentirà la raccolta, l'organizzazione e l'elaborazione di dati di interesse per comprendere al meglio le esigenze del cliente e da poter proporre la miglior soluzione una volta individuate le principali debolezze dell'impianto sotto esame.

Ovviamente tale sistema non è specificatamente innovativo in quanto inedito, dato che questi sistemi si trovano facilmente in commercio da quando è stato introdotto il concetto di industria 4.0 presso i maggiori produttori di dispositivi e impianti industriali sotto il nome di "Data logger", per esempio il *ComX 510* prodotto da Schneider Electric. [1] [2]

L’innovazione che questo lavoro intende portare con sé, risiede nell’abbattere delle barriere fondamentali per i processi di aggiornamento tecnologico delle imprese italiane, quali:

1) Elevato costo in termini di acquisto ed installazione;
2) Complessità del sistema che richiede l’addestramento di un addetto al suo utilizzo.

La soluzione che invece si intende proporre al cliente è quella di un sistema ad utilizzo in comodato, incluso nell’attività di consulenza e quindi a costo zero, che verrà installato e gestito direttamente dallo studio di Ingegneria del Ing. Caldarigi, assicurando al cliente un servizio efficiente ed innovativo ad un costo molto contenuto rispetto alla concorrenza.

Sommario

1 INTRODUZIONE AL PROGETTO

L'attività consiste nella progettazione, come già accennato, di un sistema di telerilevamento IoT per impianti industriali, il quale non è stato immaginato per essere adibito alla gestione di una singola applicazione, in quanto i problemi che possono presentarsi in ambito industriale sono innumerevoli e dipendono da numerosi fattori e dal tipo di impianto con cui ci si andrà ad interfacciare.

Sulla base di questo assunto, si è deciso insieme allo studio che il progetto presentato consisterà nella realizzazione di un sistema a microcontrollore che ovviamente non sarà una soluzione efficiente ed ottimizzata come il progetto di un ASIC, ma che soddisferà dei parametri che per la fase di prototipizzazione sono fondamentali, ossia:

- **Riusabilità**;
- **Semplicità**;
- **Affidabilità**.

Garantendo la persistenza di queste qualità (a cui si farà riferimento numerose volte durante la trattazione sotto la dicitura di "*princìpi")*, dalla scelta e connessione dei componenti alla stesura del firmware, si renderanno eventuali modifiche ed ampliamenti il più veloci possibile con la minima probabilità di imbattersi in problemi di incompatibilità e rendendo agevole la comprensione globale del progetto.

Sulla base dei *princìpi* elencati, il sistema dovrà essere in grado di ripetere in loop i seguenti passi, da quando verrà collegato all'alimentazione fino a quando verrà scollegato o spento:

1) Interfacciarsi alla rete elettrica dell'impianto che si intende studiare;
2) Connettersi alla rete internet;
3) Leggere i segnali su un periodo stabilito dal progettista;
4) Collezionare i dati raccolti;
5) Inviare i dati letti su una piattaforma di Cloud.

Si noti a riguardo che, nonostante spesso i sistemi IoT integrino al loro interno anche un sistema di analisi dei dati (edge computing) e machine learning, in questo progetto si è deciso

di non implementarli per privilegiare i princìpi del progetto e rimandarli ad una futura implementazione, dato che ora non rientrano negli obiettivi prefissati.

La progettazione di un sistema simile a quello descritto consente una infinità di applicazioni che lasciano ampio spazio all'immaginazione, consentendo di far fronte in maniera ingegnosa a numerosi problemi che si possono presentare in ambito industriale. [3] [4]

1.1 Esempio: Monitoraggio energetico (Smart Metering)

Una possibile applicazione per cui potrebbe essere utilizzato un apparato di telerilevamento come quello appena descritto potrebbe essere una analisi dettagliata dei consumi di una industria.

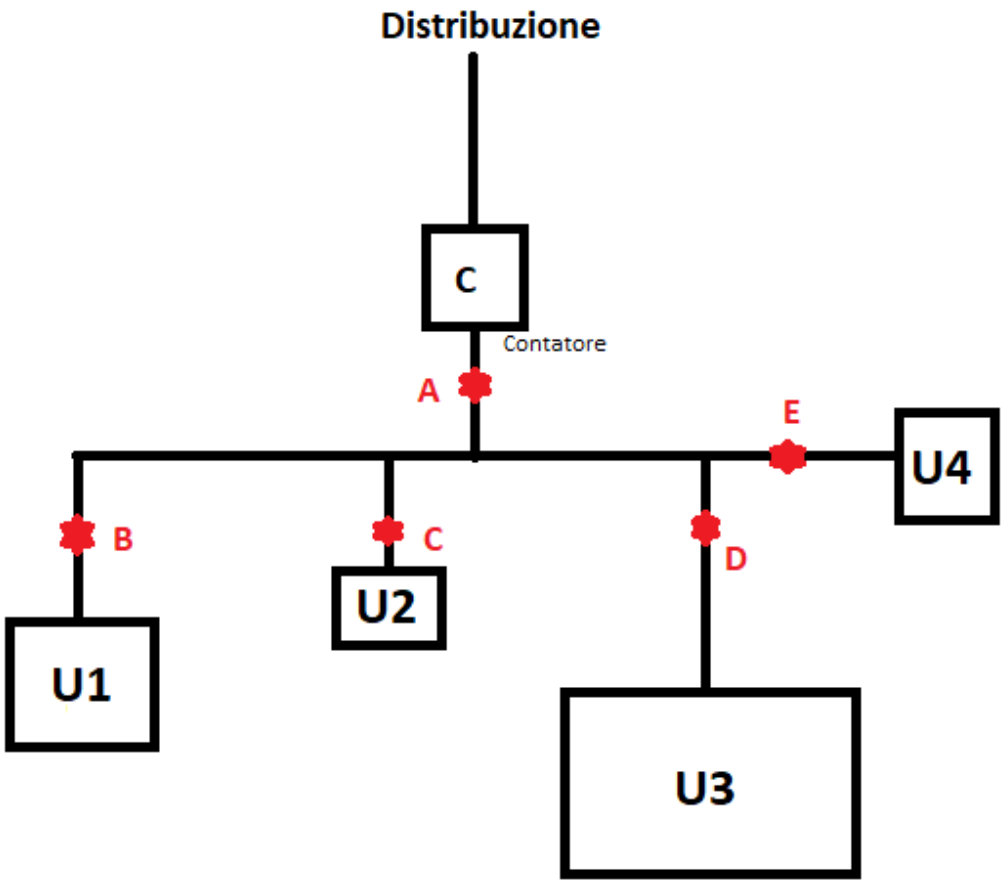

Fig. 1.1: Schema a blocchi di un ipotetico impianto industriale accessibile dalla rete attraverso un contatore, con quattro utilizzatori. Riportati con A, B, C, D, E sono i possibili punti di collegamento del sistema a microcontrollore in progetto.

Molto spesso l'unico strumento che un'azienda possiede per valutare il consumo di energia elettrica è la bolletta messa a disposizione a fine ciclo dal proprio distributore di energia, ma questa non consente uno studio in tempo reale degli impianti e tanto meno consente di isolare i consumi di un singolo reparto o di un singolo macchinario, dato che in essa vengono riportati unicamente dei valori medi misurati nel punto A visto che al distributore non interessa un monitoraggio capillare. In realtà non è raro che un'azienda per qualsiasi motivo si interessi ad uno studio capillare dei singoli impianti per identificare, ad esempio, i momenti di maggiore consumo della giornata, che può tornare utile per rendere più efficiente il comparto macchine implementando, ad esempio, specifici sistemi di generazione, immagazzinamento e gestione dell'energia.

1.2 ESEMPIO: ANALISI ARMONICA

Un problema molto importante negli impianti industriali è quello delle armoniche. In un sistema trifase a stella equilibrato, nel conduttore neutro non dovrebbe scorrere corrente per le proprietà delle reti equilibrate stesse, ma talvolta possono essere generate a causa di alcuni carichi non lineari che implementano alimentatori switching, delle armoniche importanti in quanto tali macchinari (ad esempio una saldatrice) prelevano dalla rete degli impulsi di corrente. Normalmente nei sistemi equilibrati, le armoniche si annullano essendo sfasate tra loro, ma alcune, come le armoniche omopolari (ossia di ordine dispari multiple di 3), invece di annullarsi si sommano nel neutro producendo un problema di sovraccarico e altri numerosi effetti indesiderati, quali: aumento delle perdite nei trasformatori e nei motori; malfunzionamento degli interruttori magnetotermici e differenziali (quindi gravi problemi di sicurezza su lavoro); ulteriore surriscaldamento dei conduttori; problemi sugli impianti di rifasamento; etc.

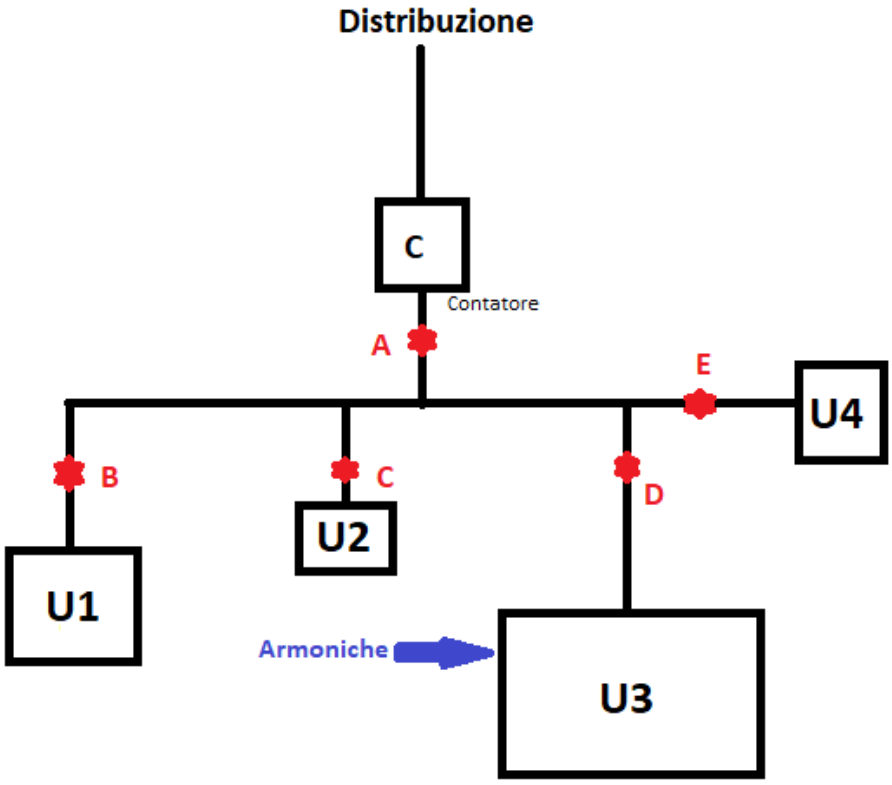

Fig. 1.2: Rilevazione attraverso il sistema di telerilevamento di un utilizzatore responsabile della produzione di armoniche

Dato che le macchine responsabili di tale effetto possono essere numerose e non facilmente rintracciabili, una possibile applicazione per il sistema che si intende progettare può essere quella di collegarlo alla rete dell'impianto per monitorare la presenza di armoniche nell' impianto elettrico così da consentire uno studio per il rintracciamento dei responsabili della generazione armonica ed una conseguente correzione del problema. [5] [6]

2 SCELTA DEI COMPONENTI

Tenendo conto dei principi esposti fino ad ora e delle funzionalità che si intende implementare, sarà di fondamentale importanza un'attenta scelta dei protocolli di comunicazione e dei componenti.

2.1 SYSTEM ON CHIP (SOC)

Per la creazione di un prototipo che segua i princìpi del progetto, è necessario dotarsi prima di tutto di un sistema a microprocessore capace di sostenere gli oneri computazionali e che allo stesso tempo segua la filosofia dei sistemi IoT andando a minimizzare la sua richiesta energetica. Una delle scelte più comuni in tal senso consiste nell' affidarsi ad un MCU basato su architettura ARM

2.1.1 Architettura ARM

Arm, abbreviazione di *"Advanced RISC Machine"*, è una famiglia di architetture di microprocessori che ricoprono un ampio spettro di utilizzi, che vanno dalla progettazione di MCU per smartphone di ultima generazione ad elevate prestazioni fino al design di sistemi embedded e SoC a basso consumo e prestazioni ridotte. [7]

Quando si parla di architettura ARM non si fa riferimento ad una specifica implementazione di un microprocessore, in quanto la costruzione ed il design di un processore rientrano in quella che viene detta *"micro architettura"*, bensì si fa riferimento alle specifiche funzionali, ossia al comportamento ed il set di istruzioni che il processore dovrà seguire. Si può vedere l'architettura come quel ponte che collega hardware e software: l'architettura descrive quali funzionalità possono essere espresse in software facendo riferimento alle reali potenzialità dell'hardware, in particolare un'architettura ARM specifica:

- set di istruzioni: viene descritta funzione e rappresentazione in memoria di ogni azione che può compiere il processore;
- set di registri: si stabilisce il numero, la dimensione, la funzione e lo stato iniziale per ogni registro;

- modelli per le eccezioni: vengono stabiliti i livelli di privilegio del processore, le tipologie di eccezioni e come quest'ultime vengono gestite;
- modelli di memoria: si stabilisce come avviene l'accesso alla memoria e come viene utilizzata la cache;
- profili di debug: viene stabilito come vengono settati i breakpoint e quali informazioni debbano mostrare gli strumenti di trace.

Per rendere più chiaro il suo catalogo, Arm ha deciso di inquadrare ogni architettura progettata all'interno di una classificazione secondo i cosiddetti profili, i quali suggeriscono l'ambito di utilizzo auspicabile del processore progettato con tale architettura. I profili definiti dall'azienda sono 3:

- **A-profile**, dove A sta per *"Applications"*, infatti le architetture di profilo A sono pensate per fornire elevate prestazioni e per eseguire programmi complessi come i sistemi operativi (ad esempio lo smartphone OnePlus Nord contiene al suo interno un MCU Qualcomm Snapdragon 765G, basato su architettura ARM Cortex-A76 [8]);
- **R-profile**, dove R sta per *"Real-time"*, il che indica una particolare predisposizione di questi sistemi ad essere impiegati nella progettazione di sistemi real-time, ossia dove il requisito più importante è una risposta veloce ed affidabile, come ad esempio un sistema di frenata assistita di una smart car;
- **M-profile**, dove M sta per *"Microcontroller"*, ossia consigliata per la creazione di processori a basso ingombro, basso consumo, elevata efficienza e prestazioni contenute, tutte caratteristiche che ben si adattano alla maggior parte dei sistemi IoT.

2.1.2 ARMv6-M

Appare evidente dai dati brevemente riportati, dando per assodata la scelta di un MCU basato su un processore ad architettura ARM, che la scelta debba ricadere su un processore di profilo M.

Ciò che distingue le architetture del profilo M dalle altre 2 in maniera predominante è l'utilizzo unicamente di un set di istruzioni ridotto; infatti Arm per le sue architetture ha sviluppato diversi set di istruzioni che fanno rifermento essenzialmente a 2 tipologie pioniere:

- A32

  L'**Arm Instruction Set** a 32bit è utilizzato da processori a profilo A ed R dalla versione ARMv7 e precedenti, dove il numero 32 sta ad indicare la lunghezza di ogni istruzione eseguita di un numero di bit pari a 32. Nei modelli ARMv8 e superiori viene utilizzato il set di istruzione A64, con codifica delle istruzioni a 64bit;

- T

  Il **Thumb instruction set** è implementato su tutti i profili ARMv7 e inferiori ed è l'unico disponibile nei profili M, dove ogni istruzione e lunga 16bit (le versione ARMv8 e superiore utilizzano il T32, dove vengono combinate istruzioni a 32 e 16 bit).

Le differenze tra i due set di istruzioni sono innumerevoli, in quanto il primo set di istruzioni può disporre in generale di una gamma più ampia di stati di eccezione ed ha un più agile accesso ai registri general purpose presenti nell'architettura, ma la principale differenza risiede nella possibilità di scrittura diretta di alcuni registri speciali riservati esclusivamente al set di istruzioni A32: tali registri (**cpsr** e **spsr**) consentono al programmatore di sfruttare a pieno le potenzialità del processore in quanto tramite la loro scrittura è possibile rendere molto più veloci delle istruzioni che solitamente nelle macchine RISC operano da collo di bottiglia, come gli interrupt e la gestione delle eccezioni. Invece un programma scritto in Thumb avrà come caratteristiche un'elevata efficienza e performance elevate con una notevole densità di codice. [9] [10] [11]

2.1.3 Cortex M0+

Il campo per la scelta dell'MCU a questo punto si è molto ristretto, quindi seguendo sempre i princìpi elencati poco sopra, si può iniziare ad esplorare in rete e valutare alcune schede a microcontrollore con profilo M.

A titolo di esempio, STM®, multinazionale italo-francese di componenti elettroniche, offre sul suo sito un'ampia gamma di MCU per applicazioni embedded, in particolare la serie STM32 presenta una gamma di processori con tecnologia proprietaria "Ultra low power", i quali implementano proprio i processori con architettura di cui si è discusso fin qui. Dato che il sistema da progettare consiste di un prototipo semplice ed efficiente, la scelta più logica ricadrebbe sull'acquisto di un processore della gamma STM32L0, i quali hanno

prestazioni molto contenute a fronte di un elevato risparmio energetico ed un'elevata efficienza dell'MCU.

La gamma STM32L0 implementa una gamma di processori altamente efficienti dal punto di vista energetico basati su architettura **Arm Cortex M0+**, tale efficienza è dovuta principalmente al bassissimo numero di gate CMOS implementati rispetto ad altri processori a profilo M e quindi sono caldamente raccomandati per applicazioni deeply embedded che richiedono ottimizzazione di spazio e consumo. [12] [13]

2.1.4 Arduino MKR1010 WiFi

Dopo un'analisi dei vari modelli di board disponibili sul mercato per l'implementazione embedded di sistemi IoT, la scelta è definitivamente caduta su un modello di **Arduino**.

Arduino è un'azienda italiana nata ad Ivrea, in Piemonte, che si occupa della progettazione di schede a microcontrollore capaci di elaborare dei dati in input e restituire un output adeguato come tutte le schede ad MCU, ciò che però ha distinto la società è la volontà di voler rendere il processo di creazione aperto e il più semplice possibile, permettendo all'utente di interagire con hardware e software attraverso un ambiente user-friendly che faciliti l'apprendimento e quindi l'implementazione delle idee degli sviluppatori in tempi brevi, soprattutto se alle prime armi. L'esperienza Arduino risulta piacevole grazie alla abbondante e sempre attiva community mondiale di sviluppatori che condividono volentieri la loro esperienza ed aiutano i meno esperti a identificare e risolvere i propri problemi.

La scelta di questo marchio è stata quasi naturale proprio in quanto condivide gli stessi princìpi che vengono seguiti in questo progetto, in particolare Arduino offre:

- Abbondante catalogo di schede a costo contenuto o comunque in linea con quello delle altre schede a microcontrollore presenti sul mercato;
- Un ambiente di lavoro (Arduino IDE) semplice e intuitivo che facilita l'apprendimento consentendo di iniziare a progettare molto in fretta;
- Ampia disponibilità di codice open source distribuito dagli utenti più esperti della community che, seguendo la filosofia dell'azienda, rendono pubbliche le loro librerie ed il loro codice con **licenza Creative Commons** rendendolo quindi riutilizzabile da chiunque in forma parziale o per intero;

- Riusabilità elevata del codice facilmente adattabile a molte altre schede MCU per applicazioni embedded.

Questa soluzione si rivela quindi ottima per la progettazione del sistema di telerilevamento in quanto consentirebbe di minimizzare il tempo dedito all'apprendimento ed alla stesura del firmware mantenendo comunque i costi in linea con le altre soluzioni di mercato. [14]

Come ultimo passo non rimane che scegliere la board più adatta al caso: tenendo conto delle specifiche di progetto già presentate in questo documento, appare evidente la necessità di interfacciare il sistema alla rete internet, perciò sarebbe opportuno dotarlo di un modulo di comunicazione wireless per renderlo il più possibile indipendente. Per fortuna Arduino, da qualche anno a questa parte, mette a disposizione una famiglia di schede che puntano proprio a creare un primo approccio al mondo IoT, essendo facili da usare ma che al tempo stesso, se usate da mano esperta, possono essere ottimizzate per l'implementazione di sistemi embedded estremamente efficienti: la famiglia **Arduino MKR**.

MKR offre un ampio ventaglio di prodotti con modulo di comunicazione wireless integrato ed ognuno di essi può essere utilizzato per lo svolgimento del progetto presentato. Quelli scelti per questa implementazione sono due:

- **MKR-WiFi-1010**

  Descritta dallo stesso produttore come "*il più semplice punto di accesso verso la programmazione IoT*", tale scheda implementa, oltre al classico circuito per la programmazione, un modulo di comunicazione wireless **u-blox NINA-W102** [15], un circuito stand alone dotato di un MCU ad architettura Harvard ottimizzato per applicazioni a bassissimo consumo che gestisce un modulo WiFi 802.11b/g/n 2.4GHz ISM ed un modulo Bluetooth v4.2. La scheda implementa come cuore il microcontrollore **Microchip ATSAMD 21**G18A – 48QFN. [16] La scheda è dotata, ovviamente, di un PINOUT che consente l'interfacciamento del microcontrollore a componenti esterni; in particolare la scheda è dotata di 8 Pin I/O digitali, 13 Pin PWM, 7 di input ed 1 di output analogici (grazie all'implementazione di ADC e DAC nel SAMD21), oltre ai classici pin di distribuzione dell'alimentazione, di massa e di reset. Da notare è anche la presenza di una presa per il collegamento di una batteria al litio che può risultare molto utile per l'installazione del dispositivo senza doverlo collegare all'alimentazione. [17]

Fig. 2.1: Scheda MCU Arduino MKR 1010 WiFi [17]

- **MKR-GSM1400**

  Scheda elettronica molto simile a quella appena descritta in quanto riporta lo stesso MCU e lo stesso pinout in una scheda delle stesse dimensioni. L'unica differenza sostanziale è la presenza di un modulo wireless u-blox SARA-U201 che permette l'inserimento di una microSIM per sfruttare la comunicazione GSM, come già suggerito dal nome, anziché il protocollo WiFI. [18]

Fig. 2.2: Scheda MCU Arduino MKR GSM 1400 [18]

Alla pagina seguente è possibile apprezzare lo schema a blocchi dell'architettura MCU SAMD21, la quale si presta particolarmente alla sua implementazione in schede come quelle appena descritte in quanto in un unico chip di dimensioni molto contenute si implementano un elevatissimo numero di periferiche, connesse tramite **AHB** (Advanced Microcontroller Bus Architecture, progettato anch'esso da Arm ed attualmente uno dei modelli di BUS per microcontrollori più veloci ed efficienti), ad un processore basato sulla già discussa architettura ARM Cortex M0+.

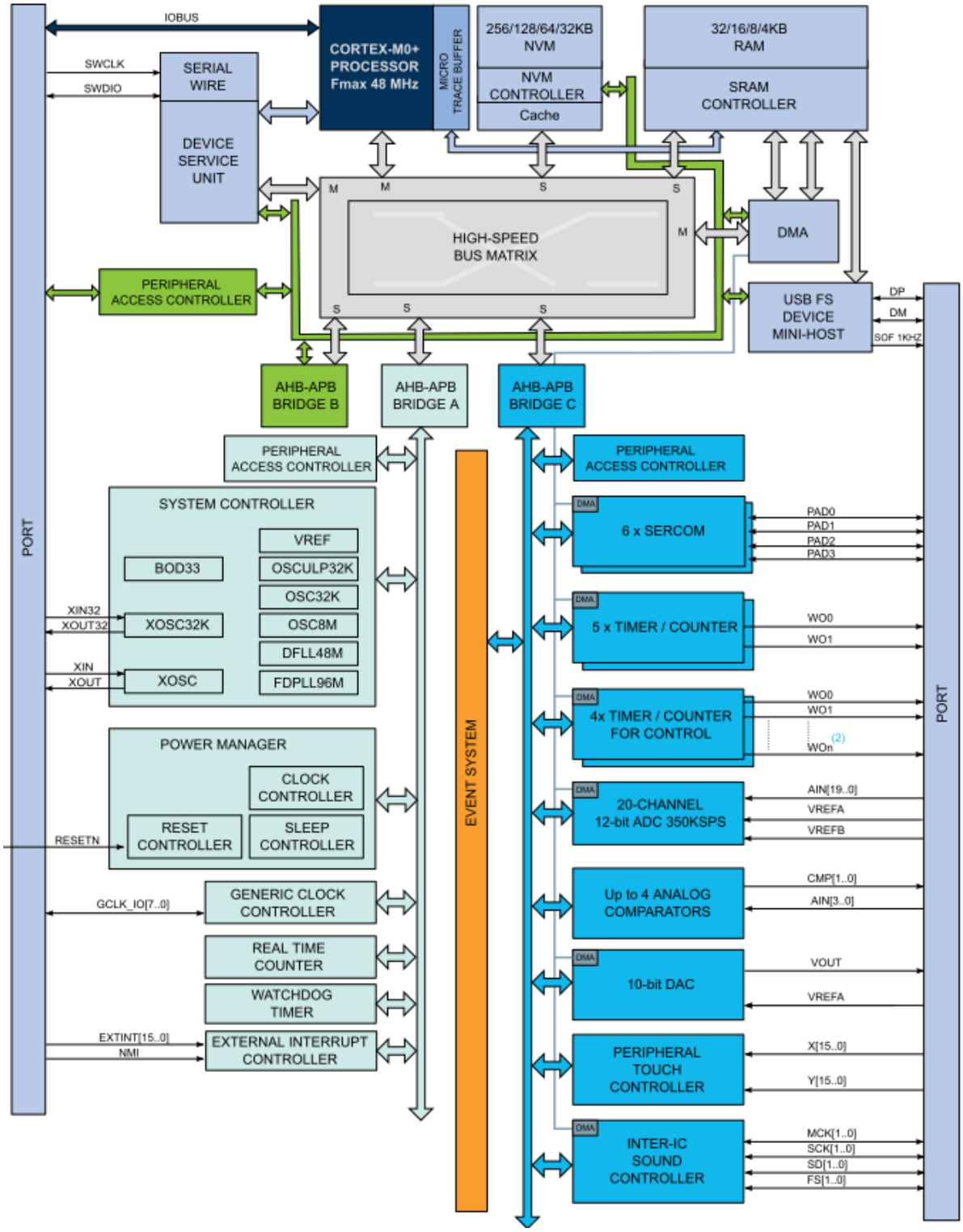

Fig. 2.3: Schema a blocchi architettura Microchip SAMD21 MCU Family [16]

Dato che per l'utilizzo della soluzione con modulo GSM si sarebbe dovuto disporre di una microSIM e quindi di un contratto telefonico dedicato, essendo questo solamente un prototipo, si è optato per la prima alternativa e quindi per l'utilizzo di una comunicazione WiFi.

2.2 SENSORE AMBIENTALE

Come già accennato, la piedinatura della scheda permette di collegare in maniera intuitiva dei dispositivi esterni per implementare funzionalità aggiuntive di cui la scheda è sprovvista. Come primo elemento esterno da implementare si è optato per un sensore di temperatura, il quale può sempre tornare utile per raccogliere informazioni ambientali riguardo al sito industriale che si intende analizzare, come ad esempio un impianto di generazione fotovoltaica dove l'energia viene raccolta proprio dall'ambiente. Il sensore dovrà essere progettato per il basso consumo ed una elevata accuratezza, e dovrà essere capace di misurare le principali variabili ambientali quali temperatura, pressione ed umidità. La soluzione utilizzata per questo progetto è ricaduta sulla scheda **Grove BME280 by Seeed** [19], la quale implementa un sensore BME280 prodotto da Bosch [20] altamente versatile ed accurato e che offre la possibilità di una sua configurazione in 3 modalità di funzionamento per l'ottimizzazione energetica: sleep; normal; forced. Il PCB scelto consente la sua implementazione tramite due protocolli di comunicazione molto utilizzati in ambito embedded di cui si discuterà successivamente nell'elaborato: SPI ed I^2C.

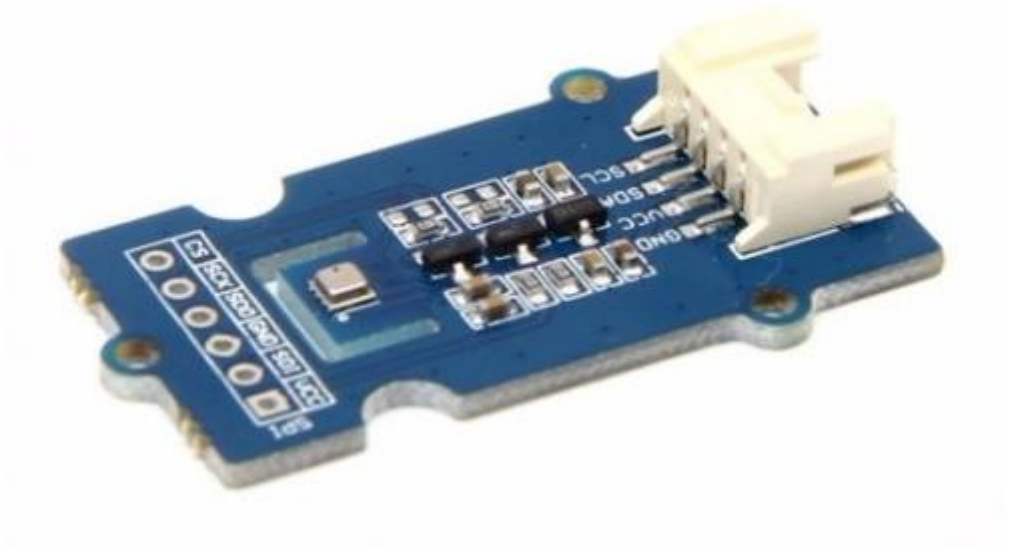

Fig. 2.4: Sensore ambientale Grove BME280. [21]

2.3 AZDELIVERY: SD ED RTC

Il sistema, come già delineato, dovrà occuparsi anche della registrazione in loco dei dati che verranno estratti dall'apparato che si sta progettando; ciò verrà fatto affinché si disponga di un quadro di backup nel caso in cui la connessione attraverso la rete dovesse essere indisponibile per un guasto o per qualsiasi altra evenienza. Per poter quindi registrare i dati in questa maniera si dovrà disporre di quel che in gergo tecnico viene chiamato "datalogger".

Per l'implementazione di un datalogger sono necessari due componenti da collegare alla scheda Arduino:

- Modulo-RTC

  RTC sta per *Real Time* Clock; tale circuito ha come scopo quello di contare lo scorrere del tempo. Esso viene implementato come modulo esterno, nella maggior parte dei casi, in quanto si tende a sollevare l'MCU da questo incarico e si preferisce utilizzare un oscillatore indipendente per alcuni motivi, quali l'ottimizzazione energetica, una maggiore precisione e l'indipendenza dall'alimentazione centrale. È infatti di fondamentale importanza che lo scorrere del tempo venga effettuato anche quando la scheda MCU entra in modalità di risparmio energetico o quando viene spento, perciò spesso si aggiunge una batteria a basso ingombro come quella degli orologi. Una possibile soluzione commerciale è il DS1307 RTC breakout board kit by Adafruit, un semplice PCB con comunicazione I^2C che integra una batteria e l'RTC DS1307 by Maxim Semiconductors [22]

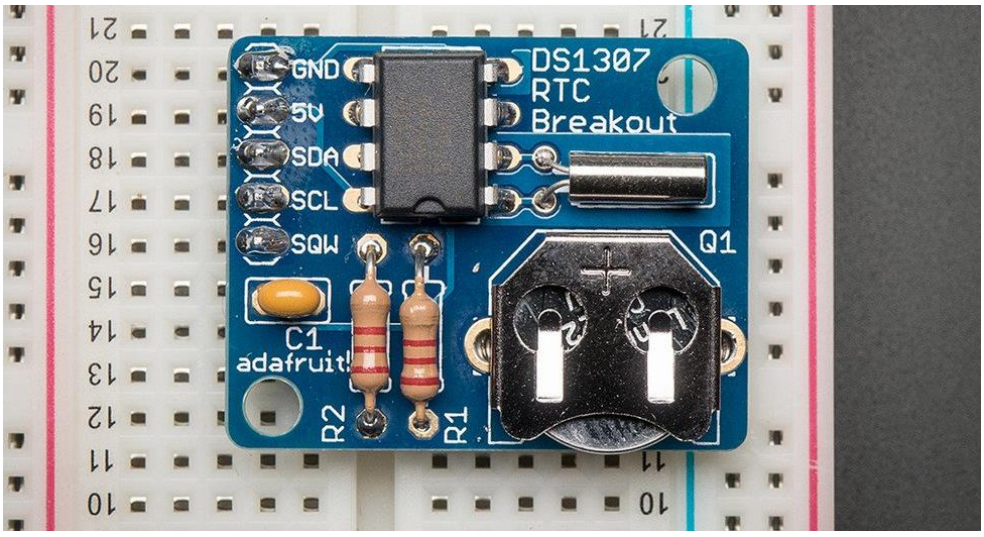

Fig. 2.5 DS1307 RTC breakout board kit by Adafruit [23]

- Modulo-SD

  Come dispositivo di archiviazione esterno, utilizzare un comunissimo **modulo SD** è l'ideale in quanto una scheda di archiviazione è facilmente reperibile, ha memoria sufficiente per i dati che verranno trattati, è altamente portabile ed è scrivibile attraverso protocolli di comunicazione standard come l'SPI e l'SDIO. Una possibile soluzione è Adafruit Micro SD breakout board, la quale implementa uno slot per microSD e comunicazione seriale su un piccolo PCB molto economico.

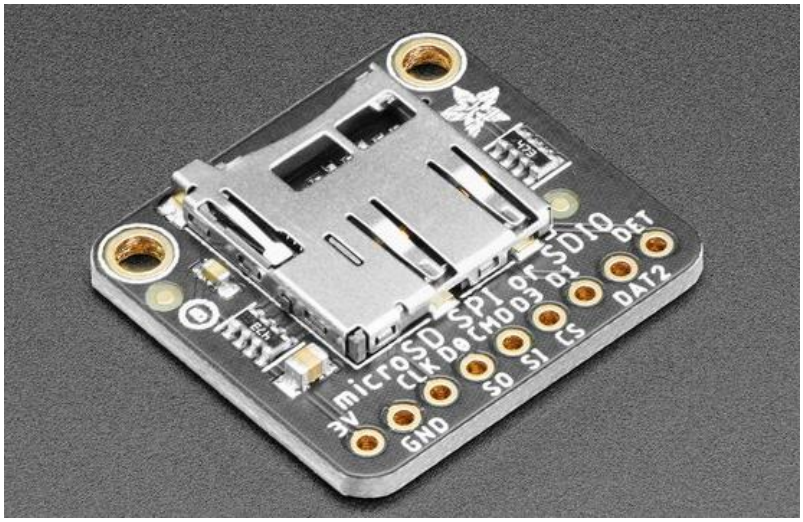

Fig. 2.6: Adafruit Micro SD breakout board [24]

Nel caso corrente, lo studio dell'Ing. disponeva già di un modulo di estensione **Datalogger by AZDelivery** per schede Arduino UNO R3, che implementa proprio i componenti appena descritti, in particolare il DS1307.

Fig. 2.7: AZDelivery modulo Datalogger [25]

2.4 RS485 SHIELD

Visto che il progetto prevede il collegamento del sistema ad un impianto industriale, è necessario implementare un protocollo che consenta la corretta comunicazione tra i due in assoluta sicurezza, dato che una scheda MCU lavora a tensioni molto più basse di un impianto industriale e se qualche componente fosse difettoso, si rischierebbe di danneggiare l'intero sistema durante la fase di collaudo o peggio durante l'utilizzo dal cliente. Per questo è necessario, oltre a scegliere il protocollo, trovare un componente esterno che garantisca l'interconnessione tra i due ambienti di lavoro in maniera affidabile. Proprio per questo si è deciso, sempre in linea con i princìpi elencati all'inizio della trattazione, di scegliere il **MKR 485 Shield by Arduino**: tale componente implementa una comunicazione seriale tra le più longeve ed utilizzate in ambito industriale (RS-485) ed essendo un componente di marca Arduino si avrà la garanzia che tale componente non tradisca le aspettative in termini di affidabilità e sicurezza, anche se questa scelta comporta un costo più elevato rispetto ai suoi competitori. [26]

Il MKR 485 Shield è un circuito di estensione dedicato per la famiglia di schede MKR, perciò, dato che il collegamento di protocolli industriali a schede MCU talvolta non è banale, si avrà auspicabilmente una maggiore semplicità di implementazione assicurata dalla comune provenienza di scheda ed estensione.

Fig. 2.8: Arduino MKR 485 Shield [26]

Lo shield in esame implementa come componente principale il **MAXIM MAX3157** [27], un transceiver RS485 con isolamento a ±50V ed alto CMRR molto affidabile e con la possibilità di essere utilizzato in modalità sia half-duplex che full-duplex (riferimento al capitolo dell'RS485). I piedini più interessanti di tale componente, ai fini del progetto sono:

Numero	Nome	Funzione
1	RO	Receiver Output
2	!RE	Receiver Output Enable
3	DE	Driver Output Enable
4	DI	Driver Input
9	ISOVCC	Internally generated power-supply voltage
10	H/F	Half-Duplex / Full-duplex selector pin Se collegato ad ISOVCC si opera in Half-duplex Se collegato ad ISOGND si oper in Full-duplex
14	ISOGND	Cable Ground
15	Z	Inverting Driver Output (and Inverting Reciver Input in Half-Duplex Mode)
16	Y	Noninverting Driver Output (and Noninverting Reciver Input in Half-Duplex Mode)
17	B	Inverting Receiver Input in Full-Duplex Mode
18	A	Noniverting Receiver Input in Full-Duplex Mode

Tabella 2.1: Principali Pin MAXIM MAX3157 [27]

Appare quindi evidente che i segnali A e B vengano utilizzati sono in modalità Full-Duplex mentre Y e Z vengono utilizzati sempre.

Per poter collegare gli ingressi allo shield, il circuito implementa un connettore a vite a sei ingressi attraverso cui si possono collegare ISOVCC, ISOGND, A, B, Y e Z.

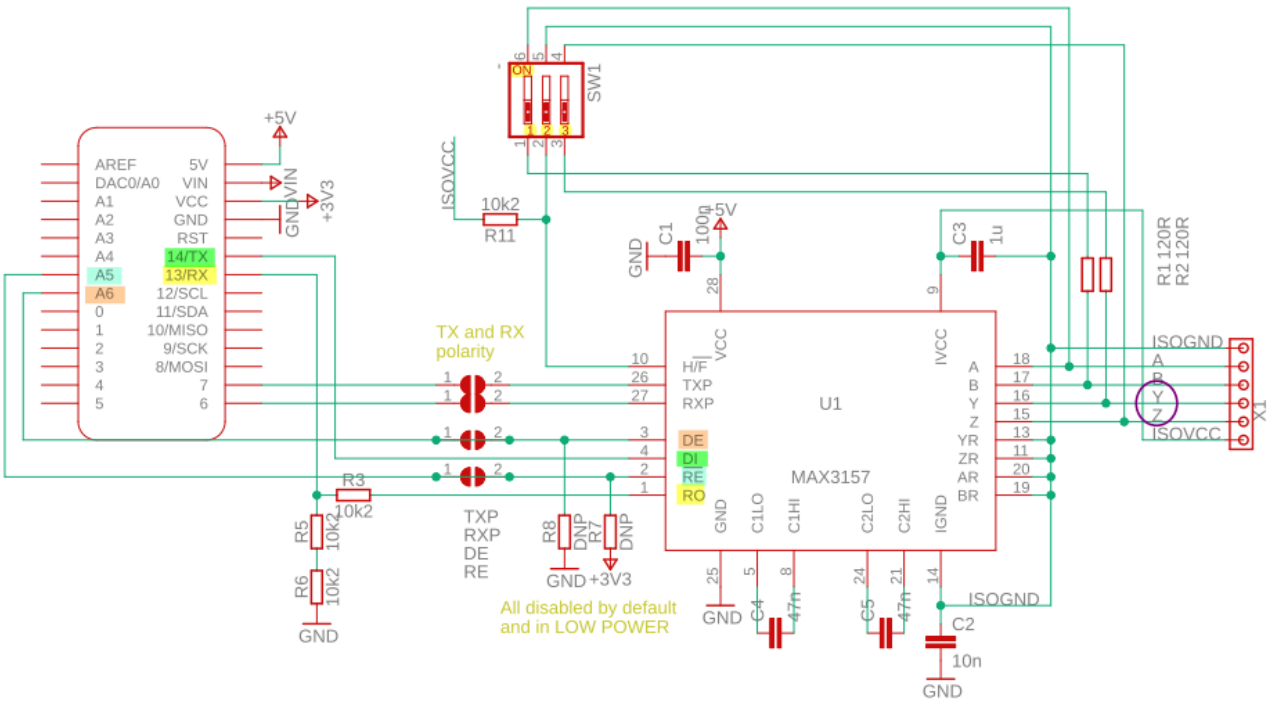

Fig. 2.9: Schema elettrico MKR 485 Shield [28]

Si può inoltre notare che il transceiver è fisicamente configurabile attraverso lo switch multiplo SW1 attraverso il quale è possibile attivare o disattivare 3 interruttori:

- il primo interruttore dello switch consente di aggiungere una resistenza in parallelo tra le piste con i segnali A e B e viene attivato qualora si intenda usare la comunicazione full-duplex;
- il secondo crea un cortocircuito tra ISOGND ed il selettore H/F del MAXIM, quindi posizionandolo su ON si attiverà la modalità Full-Duplex;
- il terzo interruttore consente l'aggiunta di una resistenza in parallelo tra Y e Z.

Nella seguente trattazione si vedrà un esempio di impostazione dello switch per l'utilizzo della comunicazione in half-duplex.

3 PROTOCOLLI DI COMUNICAZIONE

Come già anticipato, è necessario stabilire quali protocolli di comunicazione utilizzare per l'interconnessione dei vari componenti tra loro, ma mentre questa soluzione appare banale quando si parla dei protocolli SPI ed I2C, il cui utilizzo è assodato nella comunità di Arduino, un po' meno scontato è l'utilizzo del protocollo ModBus per interfacciare l'apparato di telerilevamento all'impianto sotto esame su linea RS485.

3.1 RS485

Nel 1983, EIA (Electronics Industries Association) approvò l'RS485, un nuovo standard di trasmissione che ha trovato molto spazio nell'utilizzo industriale e biomedicale.

Le caratteristiche del protocollo sono le seguenti:

- Linea bilanciata multipunto;
- Segnali differenziali in valori compresi tra -7 V fino a 12 V;
- Supporto fino a 32 load units (secondo il protocollo una load unit rappresenta una resistenza di carico di circa 12kΩ);
- Velocità di circa 10 Mbps con un cavo di 12 metri, estendibile fino a 1200m circa, ma a velocità ridotta.

Lo standard suggerisce il collegamento dei transceiver ad un bus con struttura daisy chain, la quale consiste nella connessione in serie dei dispositivi dove ogni nodo può propagare al successivo il segnale ricevuto (in questo modo si possono assegnare dei livelli di priorità senza necessità di firmware), ma può essere utilizzata anche una topologia a stella dove il centro è il bus principale

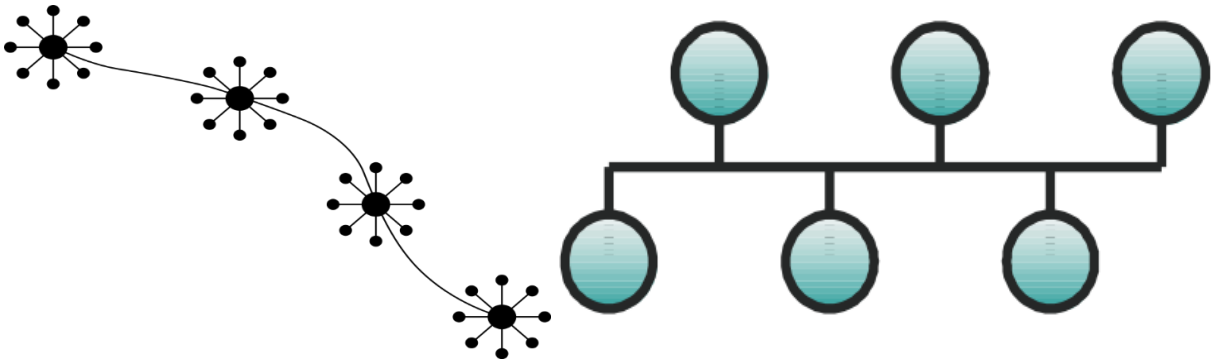

Fig. 3.1: Architettura BUS Daisy chain (sinistra) e BUS a stella (destra) [29] [30]

Le linee guida permettono una duplice implementazione della comunicazione seriale:

1) **Full-Duplex**

   È richiesta la presenza di due linee differenziali (in tutto 4 cavi) così da dividere i domini di trasmissione e ricezione del segnale, prendendo sempre come riferimento il client (o master) così da permettergli di trasmettere e ricevere contemporaneamente.

2) **Half-Duplex**

   Modalità meno dispendiosa dal punto di vista elettrico in quanto si impiega un solo bus per la comunicazione. Essendo presente un solo bus in cui inviare e ricevere le PDU, è necessario stabilire delle regole via software affinché la rete venga pilotata da un nodo alla volta.

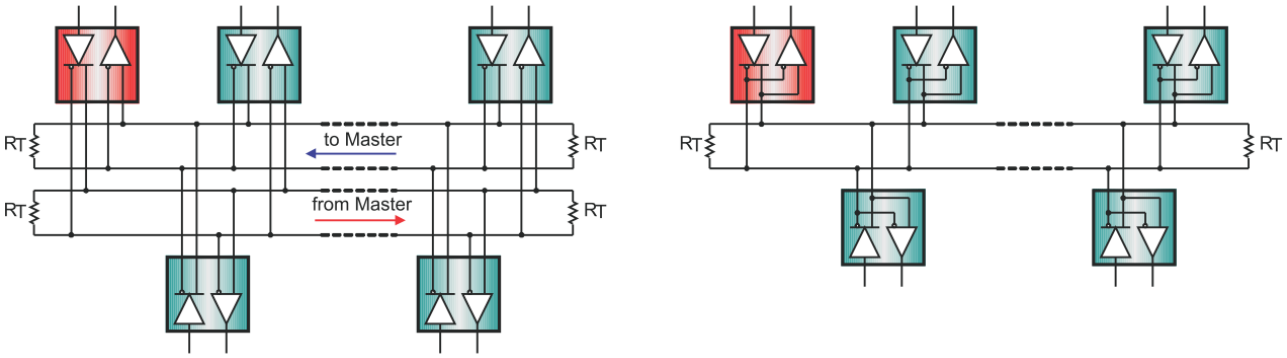

Fig. 3.2: Schema a blocchi BUS RS485 Full-Duplex (sinistra) ed Half-Duplex (destra) [30]

Una delle ragioni del successo di questo protocollo è la sua robustezza, infatti è sufficiente, nonostante il protocollo suggerisca che il driver immetta un segnale differenziale di almeno 1.5V, che al ricevitore arrivi un segnale differenziale di appena 200mV affinché esso sia riconoscibile, rendendo la linea molto resistente a delle condizioni di degradazione del segnale molto spinte.

Fig. 3.3: Schema si un generico Trasmettitore (sinistra) ed un ricevitore (destra) RS85 [30]

Lo standard inoltre raccomanda l'utilizzo di cavi UTP a 4 paia per l'implementazione di applicazioni industriali con una impedenza di 120Ω ed un AVG di 22-24 unità, ma per implementazioni low cost, soprattutto quando si parla del sistema half-duplex, è possibile utilizzare anche cavi con due paia o singolo paio di fili.

È importante ricordare che lo standard definisce unicamente le caratteristiche elettriche della comunicazione, il che lo differenzia dagli altri standard di interfaccia che definiscono in modo completo anche caratteristiche meccaniche e funzionali, infatti l'RS485 è stato progettato per essere accoppiato a degli standard di livello più alto in cui vengono definite le regole per la costruzione del pacchetto informativo, le regole di trasmissione ed il controllo degli errori. Nel caso preso in esame, RS485 verrà utilizzato per supportare il protocollo ModBus già citato. [30]

3.2 MODBUS RTU

ModBus è un protocollo creato nel 1979 da Modicon, oggi parte di Schneider Electric. Inizialmente pensato per far comunicare tra loro delle macchine PLC, successivamente si è affermato nell'ambito industriale grazie alla sua semplicità di implementazione e per la sua efficienza elevata, nonostante sia un protocollo royalty-free.

Tale protocollo agisce al settimo livello dello stack ISO OSI, stabilendo le regole per una comunicazione di tipo **client** – **server** (dalle ultime versioni, mentre anticamente il client veniva indicato come master ed il server come slave), instaurabile sia su bus di tipo Ethernet, sfruttando quindi il protocollo TCP/IP, oppure, come in questa trattazione, tramite trasmissione seriale asincrona su linea EIA/TIA-485-A (anche detto RS485) in versione ASCII o **RTU**. È interessante notare che questo protocollo mantiene un'elevata flessibilità grazie alla possibilità di interconnessione tramite gateway di linee ModBus operanti con diversi protocolli di linea.

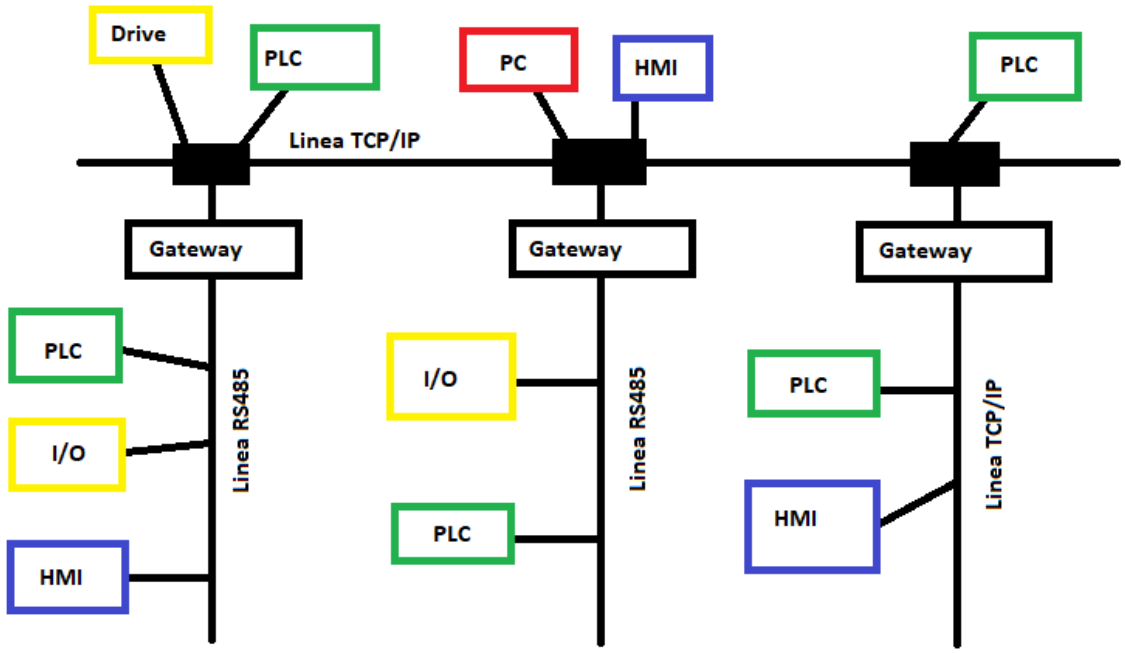

Fig. 3.4: Schema a blocchi di un'ipotetica rete Modbus su più protocolli di linea

Il protocollo in questione definisce una semplice unità dati (**PDU**, ossia protocol data unit), indipendente dal protocollo di linea su cui viene veicolata, composta da un **function code** ed un **data field**; in aggiunta possono essere appesi all'inizio ed alla fine della PDU dei campi aggiuntivi derivanti dal tipo di linea che si intende utilizzare per la trasmissione e l'insieme di questi campi aggiuntivi uniti alla PDU prende il nome di **ADU** (application data unit). Ad esempio, nel Modbus RTU, la PDU viene arricchita con un campo di controllo a ridondanda ciclica (CRC), un indirizzo e dei momenti di silenzio ad inizio e fine trasmissione dell'ADU.

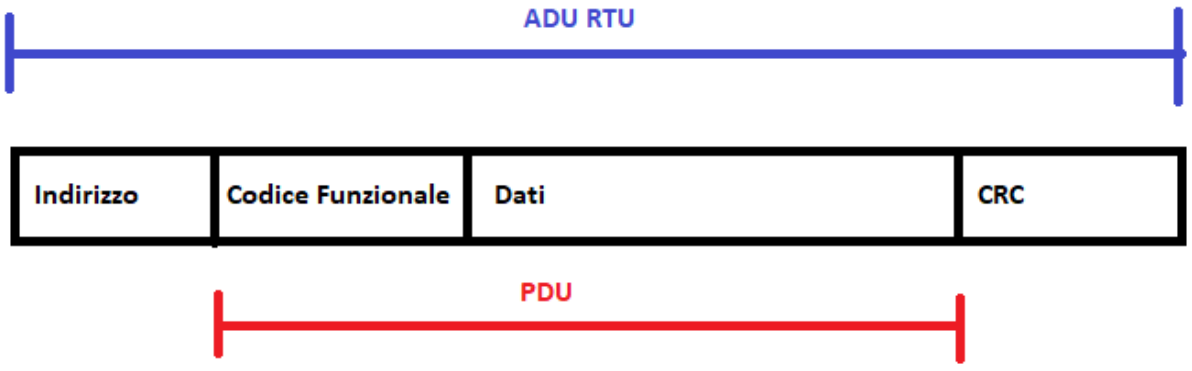

Fig. 3.5: Rappresentazione a blocchi di una generica ADU RTU

La connessione è di tipo **request-response**, perciò inizia dal client, il quale invia sulla seriale l'ADU, che deve essere composto, come già accennato, almeno dal function code ed il data field:

- Il function code contiene un numero che va da 1 a 255 e che specifica l'operazione che il client vuole che il server compia;
- Il data field contiene le informazioni aggiuntive che servono al server per eseguire l'operazione richiesta dal client (ad esempio se il client chiede al server di scrivere un registro, nel data field verrà riportato il valore da scrivere). Interessante notare che il data field non è di lunghezza fissa in quanto dipende dal function code e può essere anche lungo 0 bit nel caso in cui non servano informazioni aggiuntive al server per portare a termine il compito assegnatogli.

Qualora il server riesca a soddisfare la richiesta del client, esso risponderà con un pacchetto composto da un function code identico a quello del pacchetto inviato dal client ed il data field con i dati richiesti eventualmente dal client.

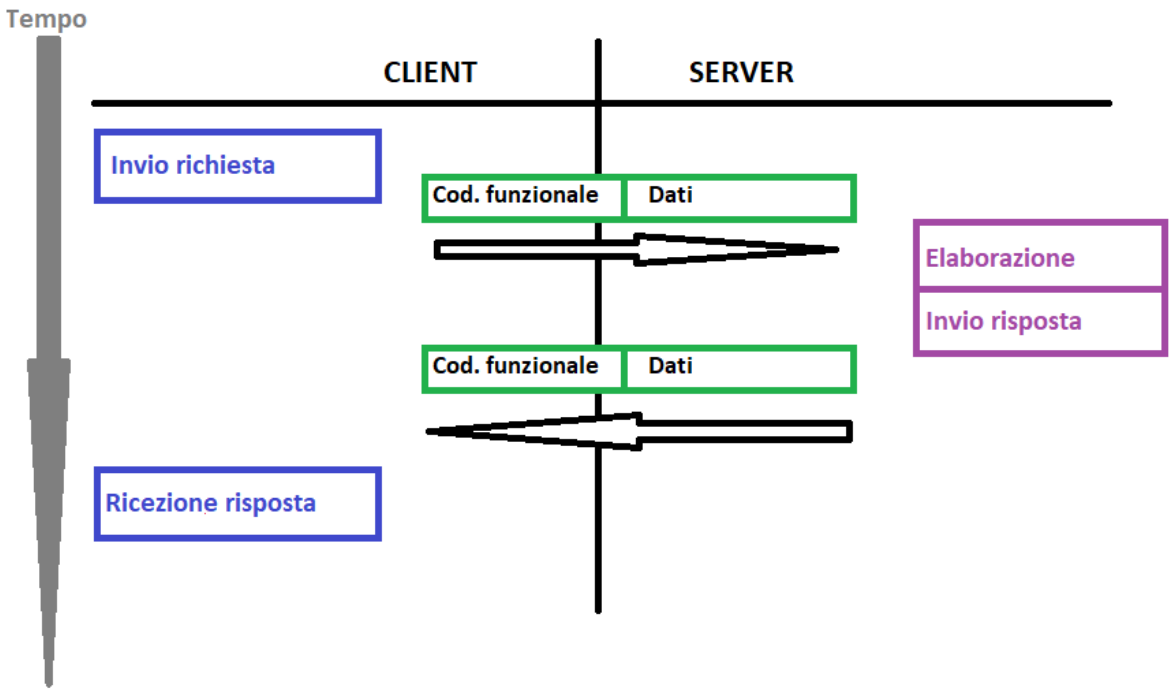

Fig. 3.6: Diagramma di flusso di una comunicazione Modbus portata a termine senza errori

Nel caso di errore verrà inviato un pacchetto simile a quello appena descritto con l'unica differenza che il bit più significativo del function code verrà aggiornato da "zero" ad "uno" riportando quindi un errore per la seguente richiesta e riportando nel data field il codice dell'errore in cui si è incappati.

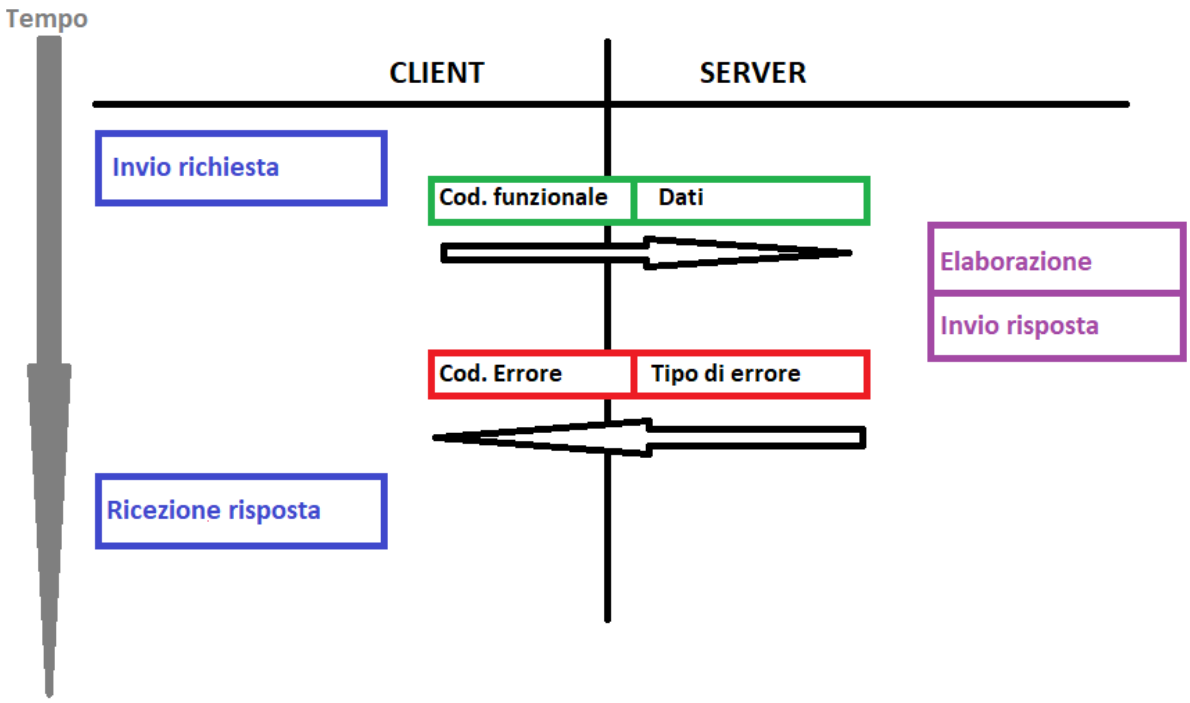

Fig. 3.7: Diagramma di flusso di una comunicazione Modbus portata a termine riportante un errore

I function codes di cui il client dispone si dividono in pubblici, user-defined e riservati. Quelli pubblici, ossia implementati esplicitamente nel protocollo da MODBUS Org. sono numerosi, ma quelli più utilizzati ed importanti sono i seguenti:

Codice funzione	Funzione implementata
0x01	Lettura Coil
0x02	Lettura input discreto
0x03	Lettura Holding register
0x04	Lettura Input register
0x05	Scrittura di un solo Coil
0x06	Scrittura di un solo Holding register
0x07	Leggi lo stato di eccezione
0x0F	Scrittura di più coil (contemporaneamente)
0x10	Scrittura di più Holding registers
0x17	Lettura/Scrittura di più Holding registers

Tabella 3.1: Principali codici funzionali PDU Modbus [31]

Dalla sua creazione, il Modbus riconosce quattro tipi di dato indirizzabili in un dispositivo server, i quali non seguono le attuali convenzioni bensì utilizzano dei tipi di dato derivanti dalla storia del protocollo stesso. I campi leggibili sono i seguenti:

Campo Dati	Tipo di Dato	Tipo di Accesso
Coil	Booleano	Read-Write
Input discreto	Booleano	Read only
Input Register	16-bit u_word	Read only
Holding register	16-bit u_word	Read-Write

Tabella 3.2: Tipi di dato accessibili ad un client Modbus [31]

Spesso nei datasheet degli apparecchi che sfruttano il protocollo Modbus vengono specificati il tipo di dato convenzionali (es: string; int; float; etc.) e l'indirizzo che si intende leggere, in più se si tratta di un registro, verrà specificata la rappresentazione del dato in maniera dettagliata, come si avrà modo di verificare in questa trattazione. [31]

3.3 I^2C

Proposto da Philips nel 1982, questo protocollo sta tornando in auge soprattutto grazie al suo ampio utilizzo nelle applicazioni IoT, come quella che si descrive in questo documento. Esso prevede l'utilizzo di 2 linee oltre la massa, è un protocollo master-slave con possibilità di avere più master sullo stesso bus ed una velocità di 100kbps. Le sue caratteristiche lo rendono perfetto per applicazioni embedded in quanto permette di collegare numerosi dispositivi utilizzando solo tre cavi.

Come accennato, I2C utilizza 2 linee per la comunicazione dette SDA ed SCL (rispettivamente Seraial Data e Serial Clock) dove la prima si occupa della trasmissione sincrona dei dati, mentre la seconda scandisce il tempo di clock attraverso il quale il dispositivo che sta comunicando ed allinea la comunicazione con il device in ricezione. [32]

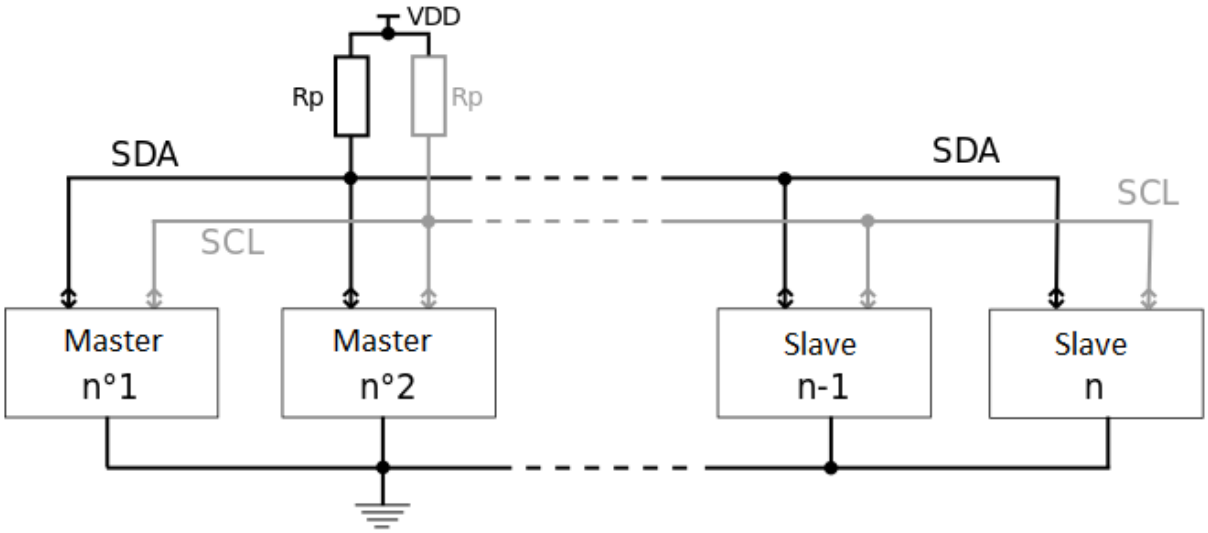

Fig. 3.8: Schema a blocchi di una rete I²C [33]

3.4 SPI

Proposto da Motorola negli anni ’80, il protocollo SPI (Serial Peripheral Interface) è un protocollo di comunicazione sincrono full-duplex, maste-slave a singolo master molto utilizzata oggi nei sistemi embedded. Il suo funzionamento prevede la presenza di almeno 5 cavi: chip_select, massa, SCLK, MISO, MOSI, dove il primo rappresenta il clock di sincronizzazione tra master e slave, il secondo (Master In Slave Out) permette la trasmissione dei dati verso il master ed il terzo (Master Out Slave In) consente la trasmissione verso lo slave.

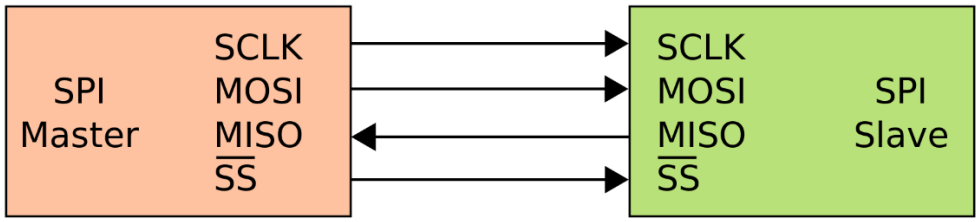

Fig. 3.9: Schema a blocchi comunicazione SPI [34]

Il protocollo consiste in un vero e proprio scambio di dati tra master e slave dove ad ogni colpo di clock, i buffer dei due dispositivi trasmettono un bit verso l’altro dispositivo permettendo di utilizzare un solo registro per la memorizzazione dei dati [34]

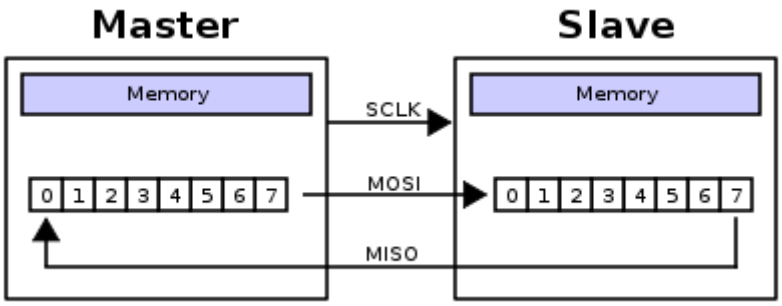

Fig. 3.10: Dettaglio dello scambio di dati tra 2 dispositivi nella comunicazione SPI [34]

4 IMPLEMENTAZIONE E FIRMWARE

Una volta scelti e studiati i componenti ed i protocolli per la comunicazione, si può procedere con il montaggio del sistema e con la programmazione.

Il montaggio del sistema dal punto di vista elettrico, grazie alla trattazione accurata di componenti e protocolli effettuata, appare ora semplice in quanto sarà sufficiente connettere tramite jumper i componenti al pin di alimentazione e massa e successivamente collegare i pin corrispondenti al modulo di comunicazione che il componente utilizza. L'unica eccezione fa riferimento al componente MKR 485 Shield che rappresenta un caso a parte in quanto, come accennato, sarà sufficiente far corrispondere la sua piedinatura al PINOUT della scheda MCU posizionandolo letteralmente sopra di essa.

Il sistema risultante dovrebbe corrispondere a quello presentato nel seguente schema a blocchi:

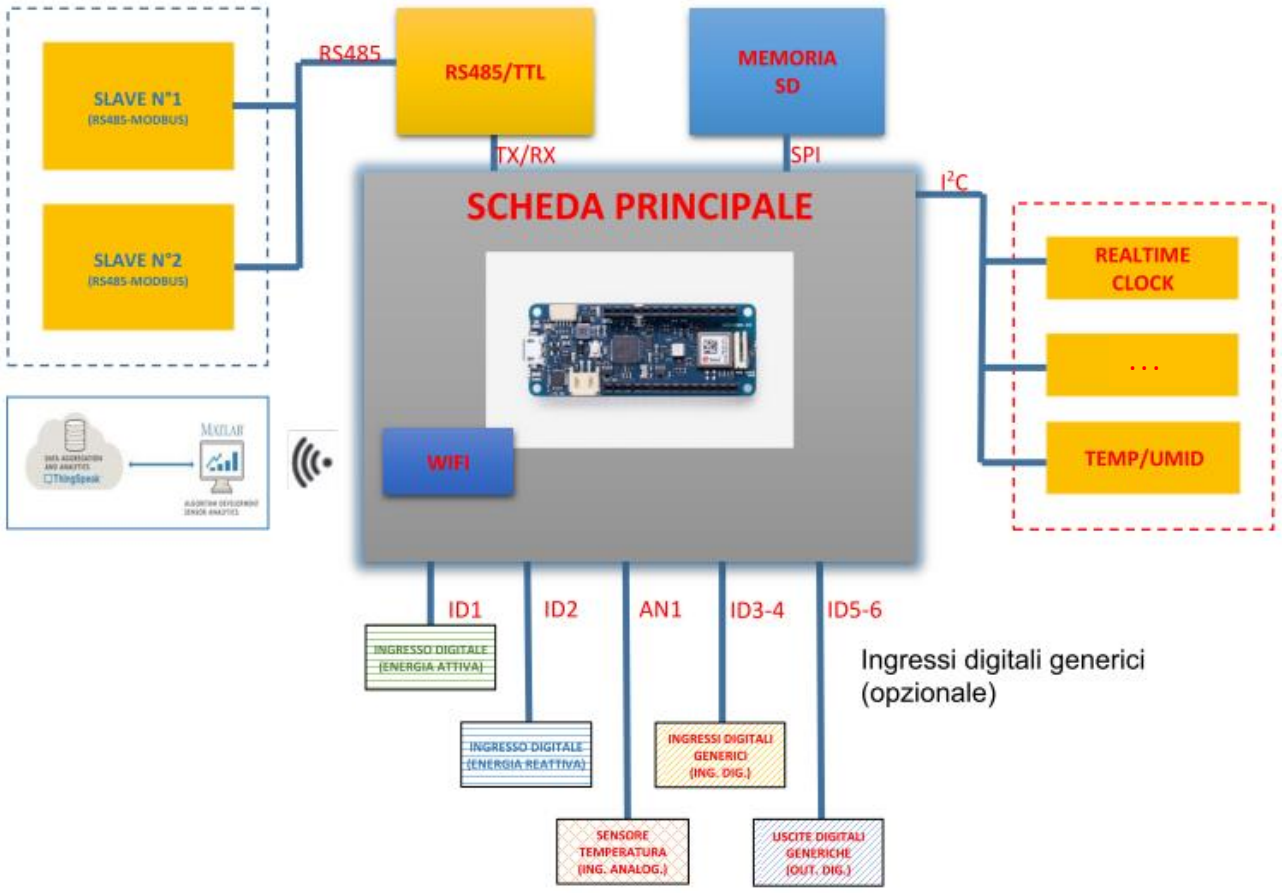

Fig. 4.1: Schema a blocchi dei componenti che compongono il nodo sensore

Riguardo alla programmazione, ipotizzando che il lettore, come è stato per il tirocinante, non si sia mai approcciato alla programmazione di un microcontrollore Arduino, è necessario abbia preliminarmente delle basi di programmazione in **linguaggio C** e similari, in quanto, come accennato in precedenza e come si vedrà in seguito, l'ambiente Arduino è stato progettato per abilitare chiunque alla programmazione MCU in tempi ristretti rispetto agli standard.

4.1 ARDUINO IDE

L'ambiente di sviluppo di Arduino è un software open source basato su Processing che consente la scrittura e la cross compilazione di programmi scritti in C++ (o semplicemente C o in AVR-C per i più esperti) che successivamente possono essere caricati sulla scheda MCU tramite interfacciamento USB, infatti quasi tutte le schede Arduino sono dotate di una interfaccia USB attraverso la quale è possibile caricare il programma cross compilato sul PC verso la board.

Quando si parla di **cross compilazione**, si intende quel processo di generazione di un file eseguibile per una architettura diversa da quella su cui si è scritto il codice.

L'interfaccia grafica di Arduino IDE si presenta appena lo si apre mostrando un editor di testo molto semplice con funzione di highlight per le parole chiave del codice. In alto possiamo trovare le solite barre delle opzioni, dove spicca l'insolita tendina "**Sketch**" nella quale si trovano i comandi utili per la compilazione e l'inclusione delle librerie.

Quando si crea un nuovo sketch, ossia un nuovo foglio di testo per la programmazione, vengono già incluse nel nuovo programma le due funzioni essenziali per la composizione dello script:

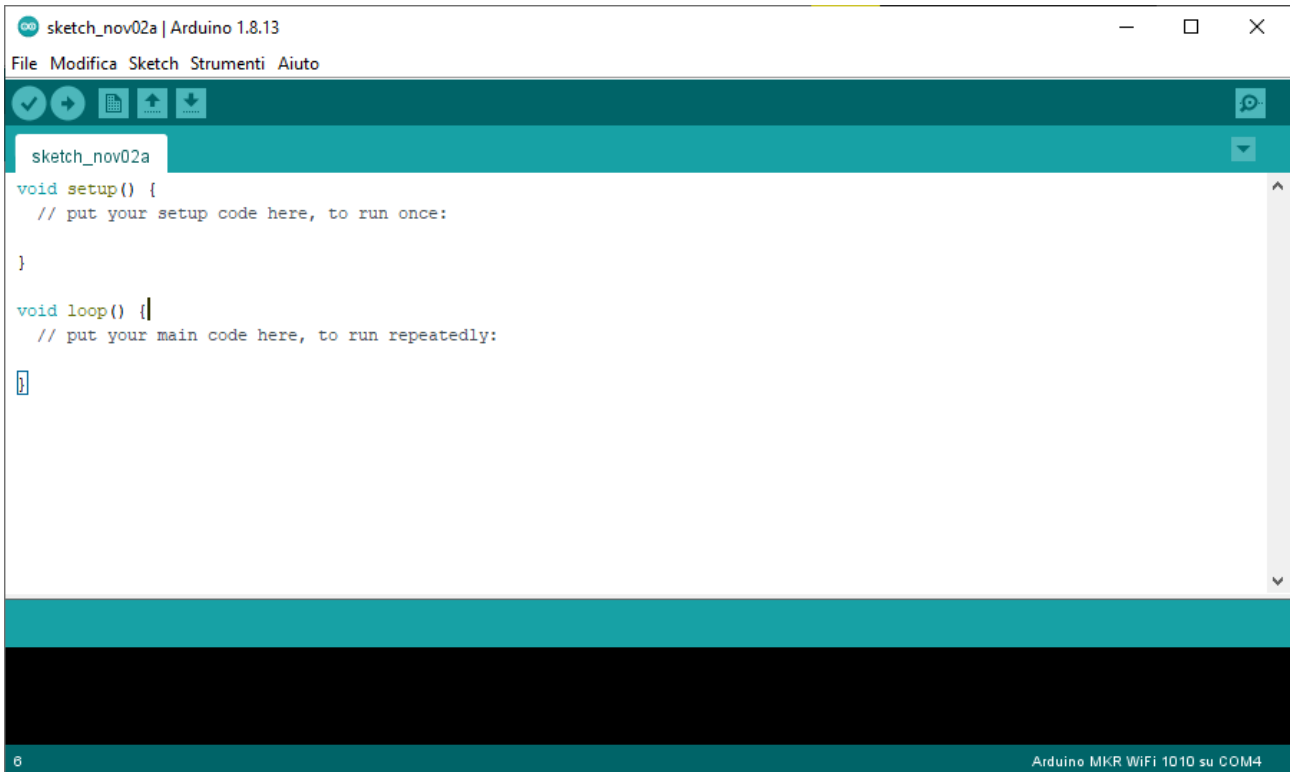

Fig. 4.2: Prima apertura del software Arduino IDE [35]

- **Setup**:

  In questa funzione è possibile inserire i comandi che andranno eseguiti una sola volta al momento del caricamento del programma sulla board e poi non verrà mai più eseguito. Qui si troveranno quindi in modo preponderante comandi di inizializzazione e controllo della connettività.

- **Loop**:

  Funzione eseguita, come suggerisce il nome, in modo ciclico fino a che il dispositivo è alimentato. In essa si troverà la parte principale del codice da eseguire.

Un programma strutturato tramite funzioni setup e loop è detto ad **architettura Super loop** [36], la quale ha trovato molto successo, appunto, nei sistemi embedded per la gestione ordinata ed in un tempo facilmente gestibile dei processi.

L'approccio che si è seguito per l'implementazione dei vari apparati esterni consiste nell'implementare e programmare i vari pezzi uno alla volta in sketch diversi così da poter applicare un approccio ingegneristico di tipo divide et impera, ovvero prima si è deciso di risolvere singolarmente i vari problemi connessi alla programmazione dei singoli dispositivi per poi, infine, unire i risultati in un unico sketch. In questo modo si avrà la sicurezza che i

componenti saranno correttamente configurati ed installati e gli unici problemi con cui ci si dovrà confrontare nello script finale saranno problemi di compatibilità dei codici da unire.

Come accennato, il pregio principale dell'ambiente Arduino è la velocità di apprendimento grazie all'implementazione di funzioni semplici ed intuitive che consentono una rapida interazione con le schede. Arduino mette a disposizione degli esempi già pronti per permettere ai novizi di avere una prima interazione con la programmazione ed il caricamento del software; l'iconico programma di prova di Arduino è il cosiddetto Blink.

4.1.1 Blink

Prima di tutto è necessario selezionare e caricare il core della board su cui si intende caricare il programma, infatti, dato che diverse schede in commercio presentano MCU con architetture differenti, è necessario che il compilatore conosca il calcolatore per cui dovrà scrivere il programma in modo che il codice sia adatto alla microarchitetture, perciò il primo passo è andare su: Strumenti> Scheda: "…"> Gestione Schede e ricercare la famiglia della scheda che si intende programmare. Nel caso di questo progetto, verrà scaricato il **core SAMD**, che come già accennato corrisponde al MCU Microchip Arm Cortex M0+ contenuto nella scheda selezionata per il progetto, la quale è anche menzionata nella descrizione del pacchetto che si andrà ad installare

Fig. 4.3: Dettaglio della sezione "Gestione schede" e download della libreria "Arduino SAMD Boards"

Una volta installato il core, si potrà selezionare navigando su Strumenti> Scheda: "…"> Arduino SAMD> Arduino MKR WiFi1010. Da questo momento in poi, il compilatore genererà codice compatibile con la scheda selezionata.

Cliccando su File> 01.Basics> Blink verrà scritto sull'editor un programma pronto per la compilazione che effettuerà l'accensione e lo spegnimento intermittente del led di sistema della scheda Arduino in questione.

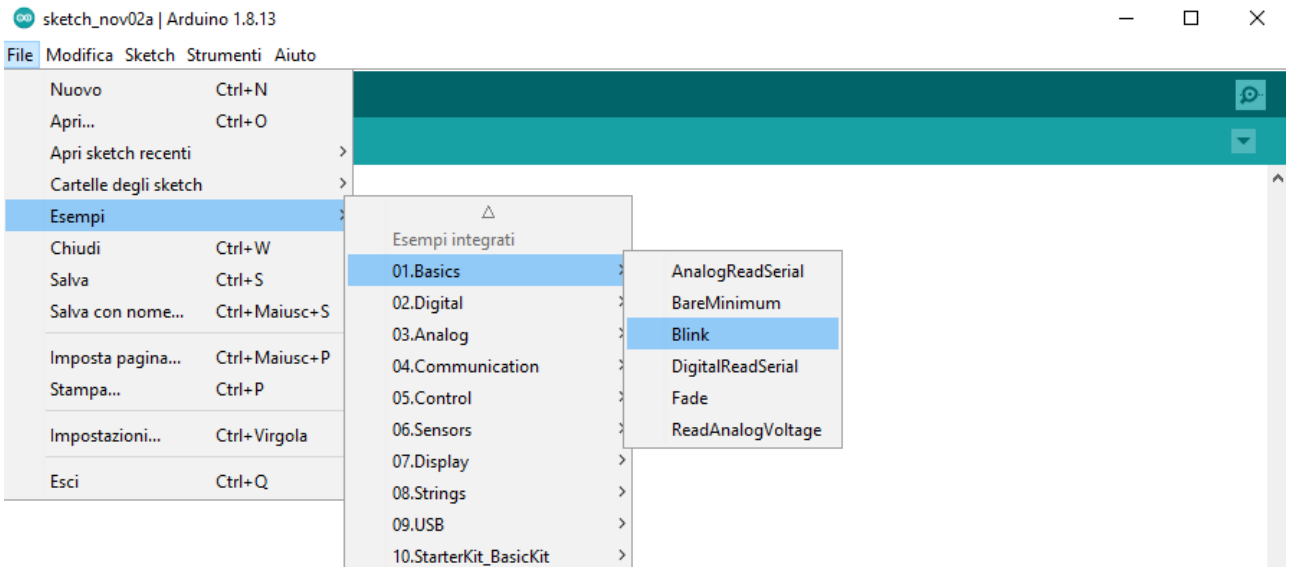

Fig. 4.4: Dettaglio della navigazione attraverso le tendine di Arduino IDE per aprire l'esempio "Blink"

Il programma consta, come accennato, di una funzione di setup dove viene inizializzato il pin a cui è collegato il led di sistema della scheda, che viene indicato con la define "`LED_BUILTIN`", come output tramite il metodo `pinMode()`; nel loop viene invece scritto il programma vero e proprio dove si accende il led di sistema scrivendo sul pin che lo comanda il valore alto per l'accensione (valore intrinseco salvato nella define `HIGH`) e successivamente lo spegnimento (`LOW` è la define del valore basso) con una cadenza di 1000ms tra accensione e spegnimento tramite la funzione `delay()`. Di seguito viene riportato il programma per intero:

Blink.ino

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage
level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage
LOW
  delay(1000);                       // wait for a second
}
```

Successivamente, per implementare il programma sulla scheda, sarà necessario collegare la scheda Arduino al calcolatore tramite interfaccia USB e selezionare Strumenti> Porta: "…" e selezionare la porta seriale su cui viene rilevata la presenza della scheda, che nell'esempio riportato è la porta COM4.

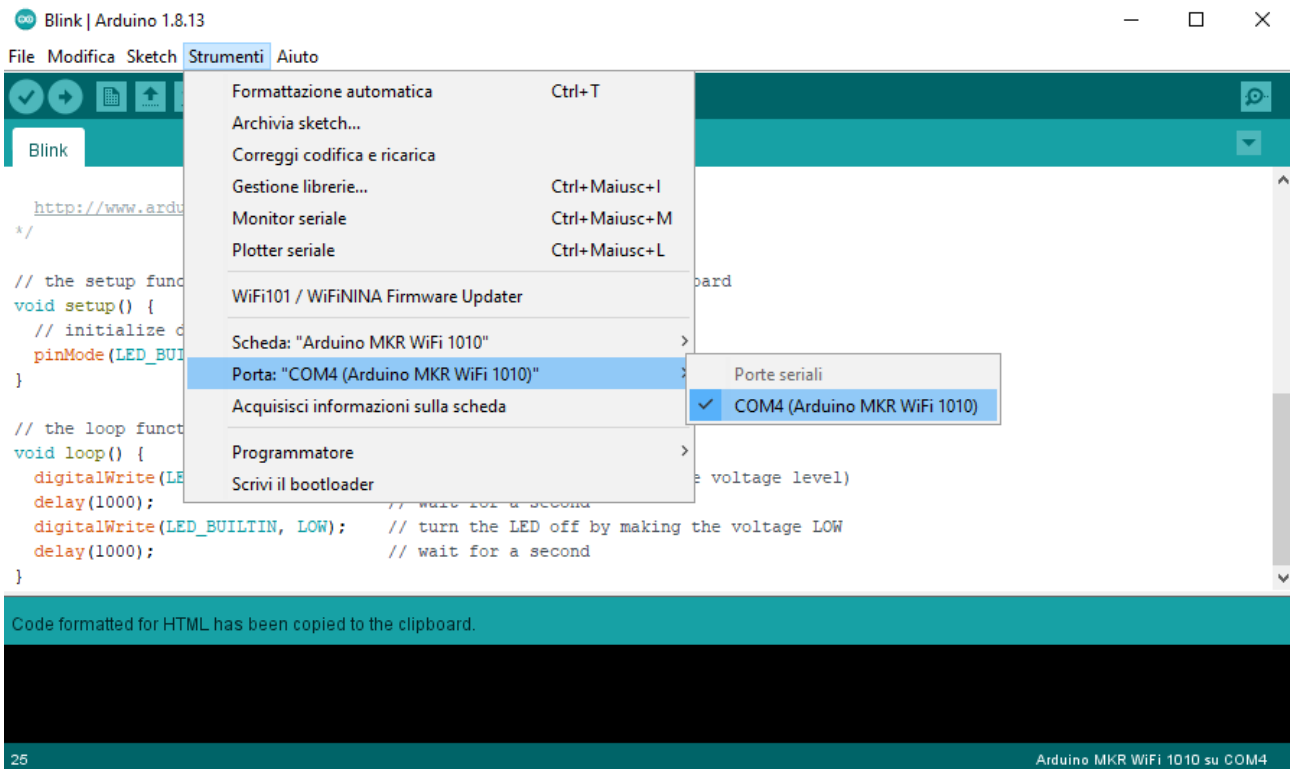

Fig. 4.5: Dettaglio della navigazione sul menu a tendina per selezionare la porta seriale corretta su cui è interfacciata la scheda a microcontrollore che si intende utilizzare

Infine per caricare il programma, ipotizzando di non riscontrare problemi con questo programma di prova, sarà sufficiente selezionare: Sketch> Carica, così che il programma

verrà compilato ed in seguito caricato sulla scheda (è possibile utilizzare anche lo shortcut con il cerchio che contiene una freccia verso destra, cerchiato nella prossima immagine).

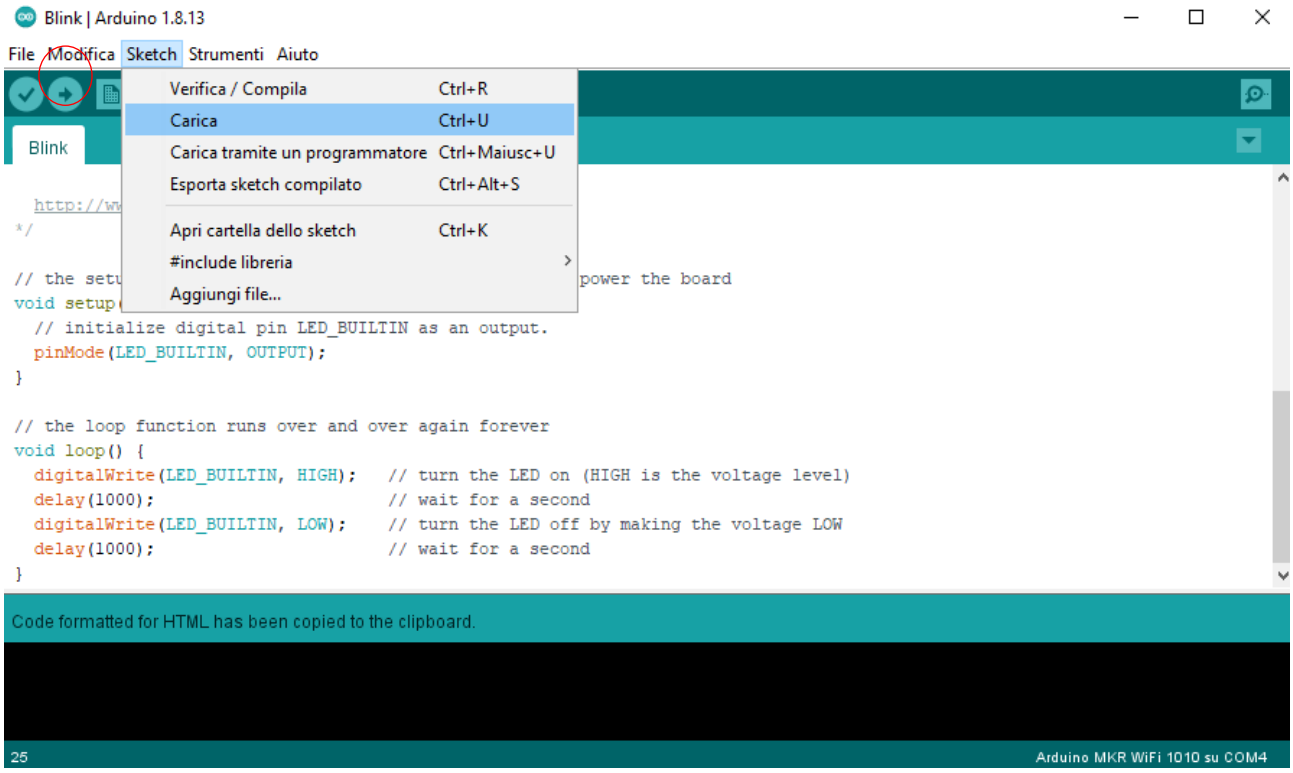

Fig. 4.6: Navigazione sul menu a tendina per caricare lo sketch sulla scheda Arduino. In alternativa si può usare la scorciatoia cerchiata in rosso

A questo punto il compilatore riporterà le operazioni ed eventuali errori nel blocco nero nella parte bassa dell'interfaccia utente ed a fine compilazione scriverà se il caricamento è andato a buon fine e la memoria occupata dal programma.

Qualora il caricamento andasse a buon fine, si dovrebbe apprezzare l'accensione intermittente del led arancione sulla scheda vicino al pin 5V.

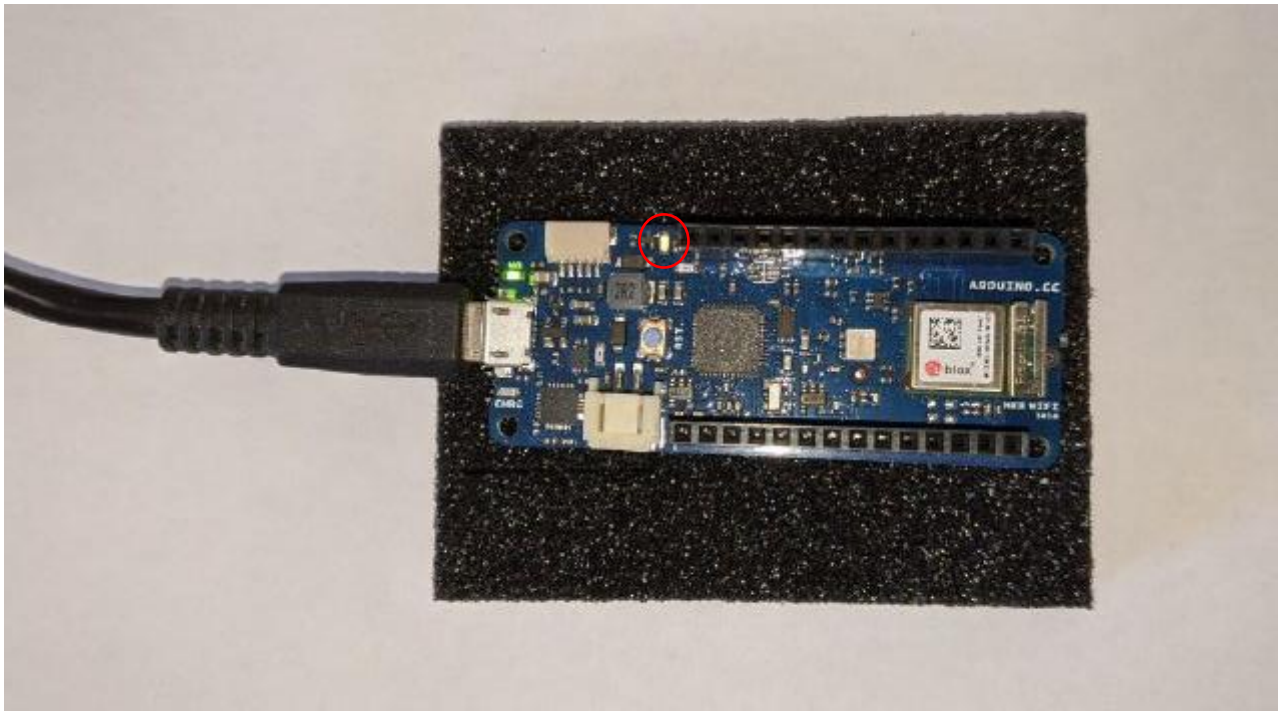

Fig. 4.7: Foto della scheda MKR 1010 WiFi personale. Si noti il colore giallo del LED acceso (cerchiato in rosso)

Una volta presa dimestichezza con l'ambiente di sviluppo ed apprese le basi per la programmazione del microcontrollore, si può procedere con il collegamento dei componenti esterni già descritti ed iniziare a comporre il sistema di telerilevamento.

Si è scelto di partire dal componente più semplice per poi crescere via via con la complessità dei componenti da implementare per poter permettere al programmatore, su ipotesi che egli non abbia mai programmato per schede Arduino, di apprendere il linguaggio alzando gradualmente la difficoltà.

4.2 IMPLEMENTAZIONE TRAMITE I2C & SPI

Come si può vedere dalla piedinatura della scheda MKR, essa prevede dei pin dedicati alla configurazione di un bus I2C ed un bus SPI, perciò il collegamento elettrico dei componenti che sfruttano questo protocollo è intuitivo dato che sarà sufficiente collegare i pin corrispondenti della scheda Arduino con quelli del componente da mettere in comunicazione.

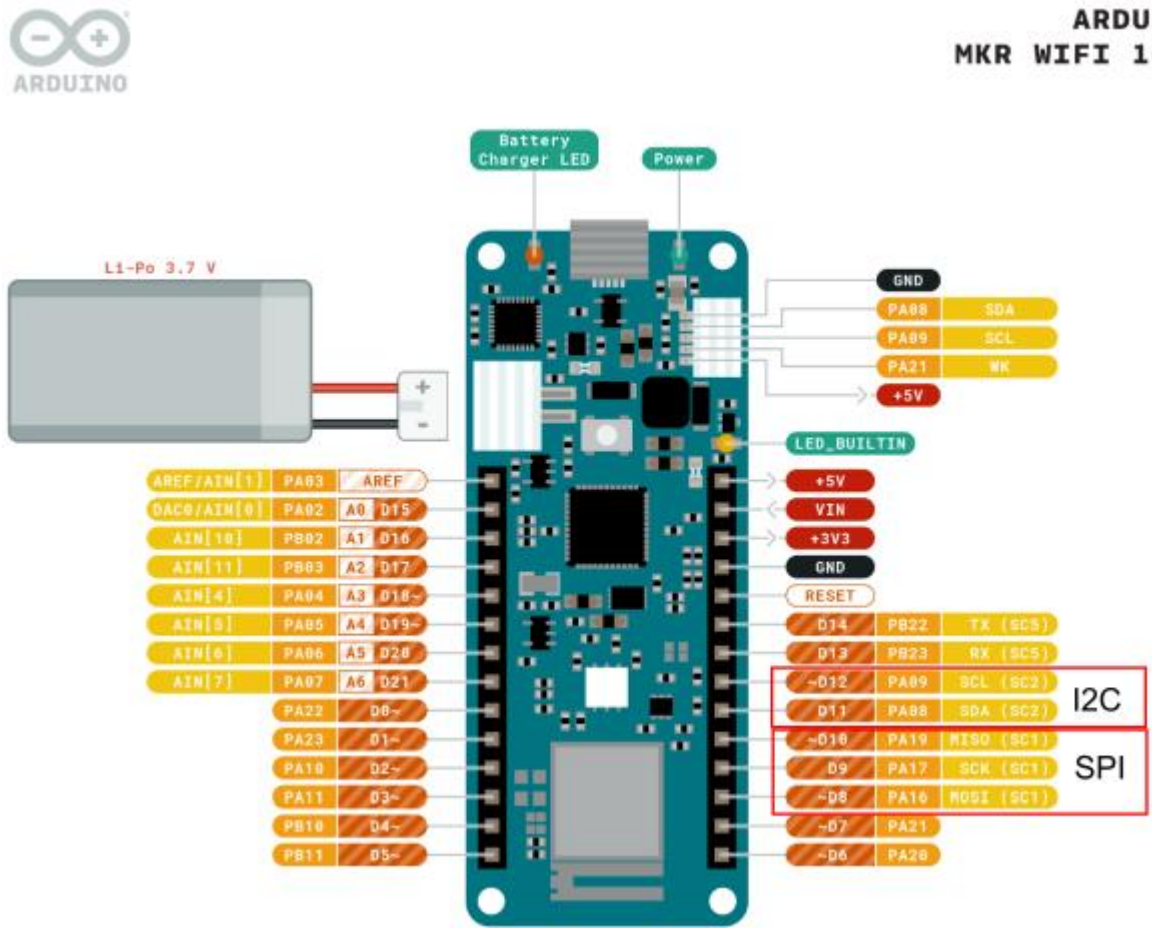

Fig. 4.8: Pinout della scheda Arduino MKR 1010 WiFi [37]

4.2.1 Sensore di temperatura

Il sensore BME280, come accennato, permette la connettività tramite modulo I2C attraverso un pratico connettore integrato collegabile tramite jumper messi a disposizione dallo stesso venditore, perciò per connetterlo sarà sufficiente collegare i giusti pin del microcontrollore ai pin I2C del sensore.

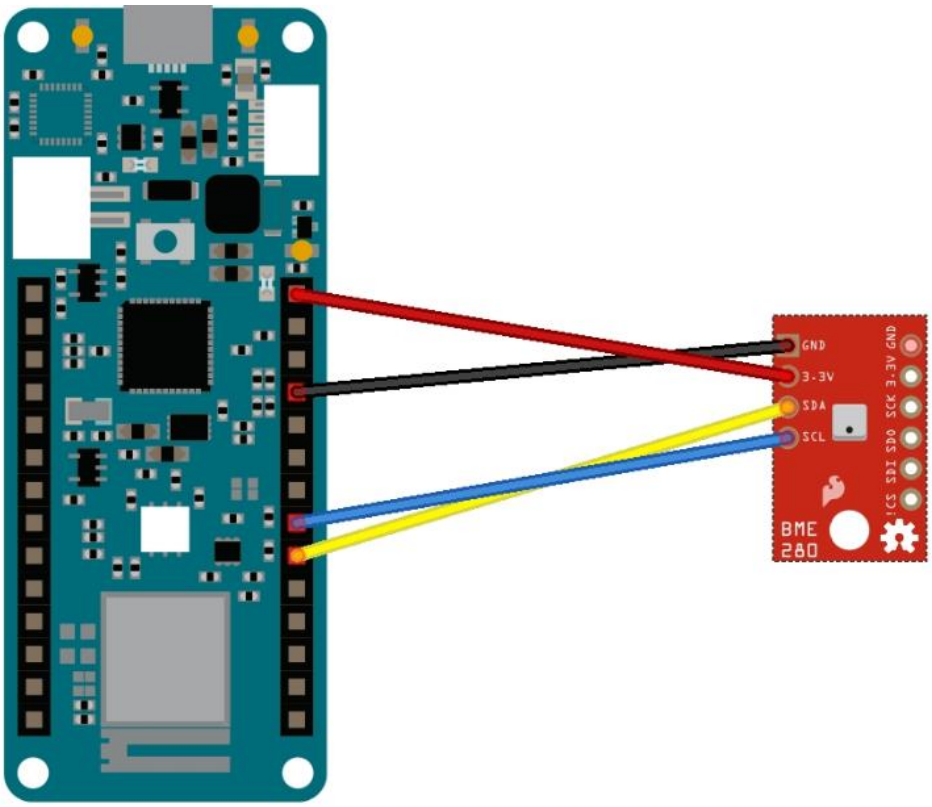

Fig. 4.9: Dettaglio del collegamento tramite del sensore Grove BME280 alla scheda Arduino tramite jumpers

Una volta stabilito il collegamento, si può passare alla fase di programmazione del firmware in cui si cercherà di utilizzare il sensore per fare delle letture di temperatura, pressione ed umidità.

Consultando la pagina del sensore sul sito web Arduino, viene esplicitata la presenza di una libreria standard messa a disposizione da Seeed [38] per una veloce configurazione del sensore. Quando si parla di librerie si intende un pacchetto di codice contenente file scritti in C++ in cui vengono definite delle classi che configurano il funzionamento dei componenti a basso livello e che implementano dei metodi attraverso i quali si può facilmente pilotare il componente senza dover scrivere codice aggiuntivo per la configurazione di ogni singolo pin e senza dover specificare come deve avvenire la comunicazione e la gestione dei dati, in quanto qualcuno lo ha già fatto per noi. Si ricorda infatti che la community di Arduino si distingue per l'elevato numero di utenti che decidono di condividere il loro codice per rendere più facile il percorso agli utenti meno esperti, detto ciò, sarà sufficiente andare su Sketch> #include libreria> Gestione librerie

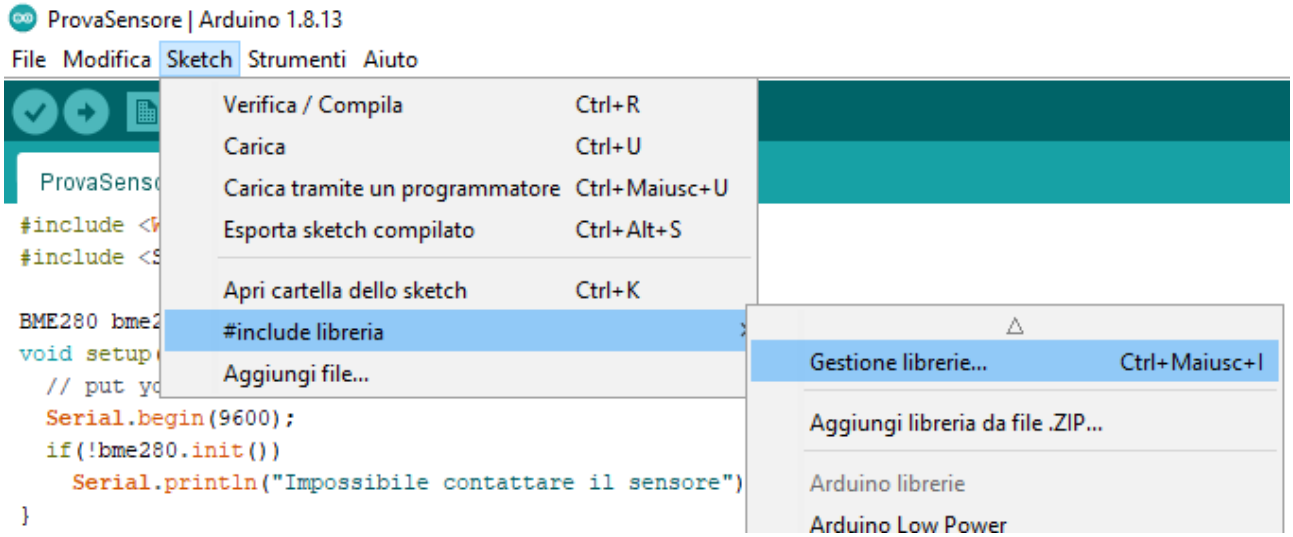

Fig. 4.10: Dettaglio della navigazione sul menu a tendina per arrivare alla sezione "Gestione librerie"

successivamente si può cercare dal motore di ricerca la libreria del componente scrivendone il nome e la marca, e successivamente installarla

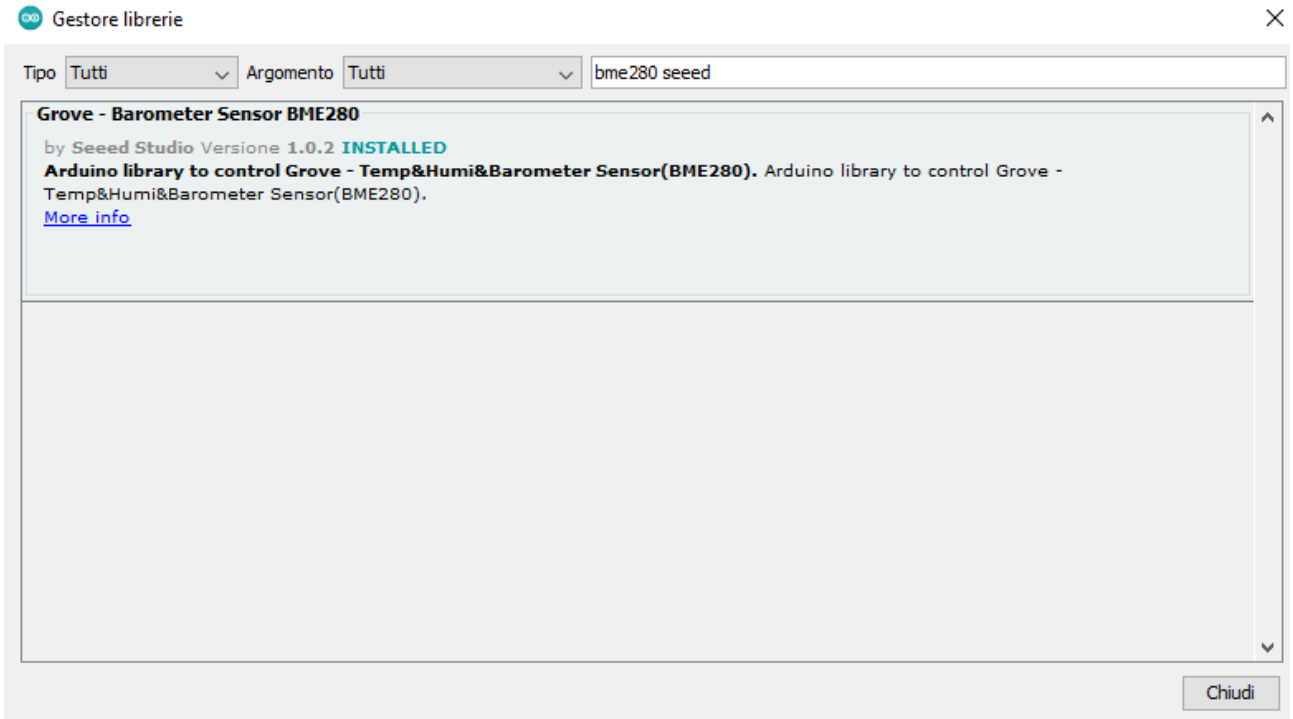

Fig. 4.11: Dettaglio della sezione "Gestione librerie", ricerca della libreria desiderata tramite parole chiave ed installazione

Fatto questo basterà tornare su Sketch> #include libreria e selezionare nel menu a tendina la libreria appena scaricata. Una volta fatto questo, apparirà in cima al programma un comando `#include` che includerà la libreria in questione

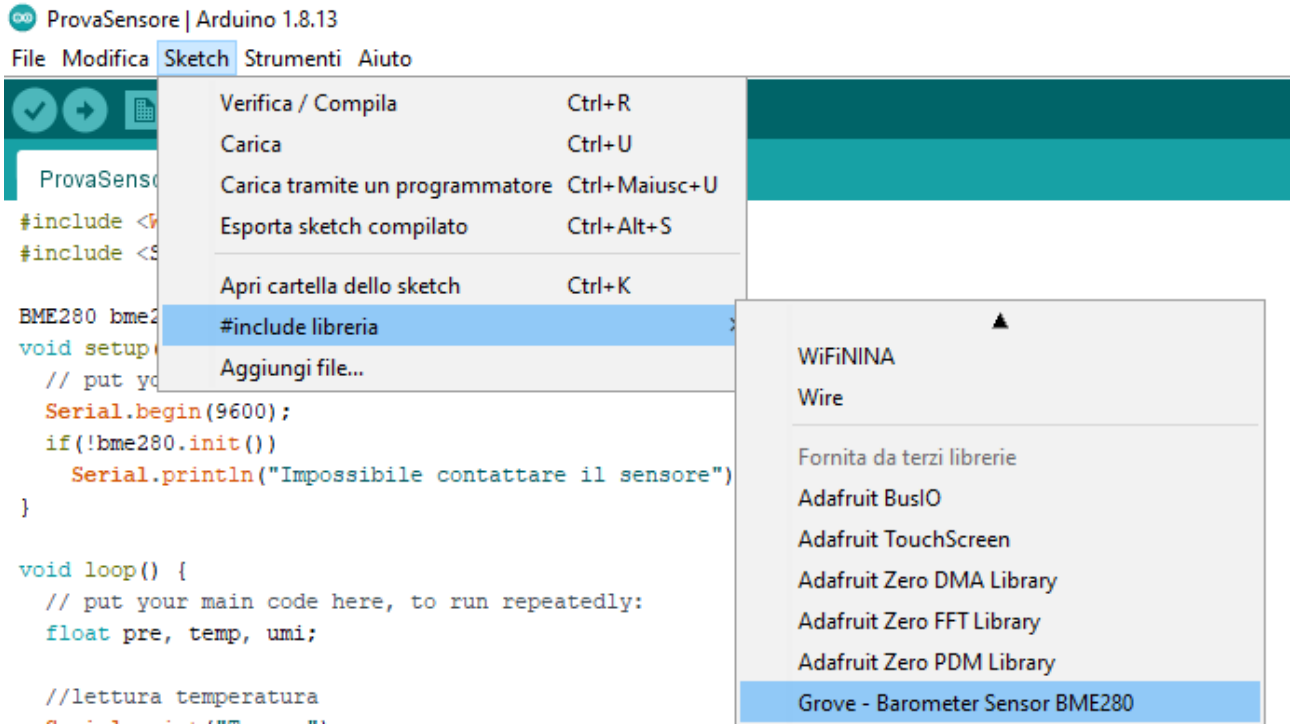

Fig. 4.12: Dettaglio della navigazione sul menu a tendina per selezionare la libreria del sensore appena scaricata

Inclusa la libreria, andando sulla documentazione rilasciata dai programmatori di tale libreria facilmente rintracciabile con una ricerca sul motore Google (la maggior parte della community rende disponibili i suoi progetti su GitHub, portale di condivisione per programmatori) sarà possibile aprire un esempio e capire come funziona la libreria e come possa essere sfruttata. Una possibile implementazione del codice è la seguente:

Prova_sensore.ino

```
#include <Wire.h>          //Libreria per la comunicazione I2C
#include <Seeed_BME280.h> //Libreria per il sensore

BME280 bme280;
void setup() {
  Serial.begin(9600);
  if(!bme280.init())
    Serial.println("Impossibile contattare il sensore");
}

void loop() {
  float pre, temp, umi;

  Serial.print("Temp: ");
  Serial.print(temp = bme280.getTemperature());  //lettura temperatura
  Serial.println("C");

  Serial.print("Pressure: ");
  Serial.print(pre = bme280.getPressure());  //lettura pressione
  Serial.println("Pa");

  Serial.print("Humidity: ");
  Serial.print(umi = bme280.getHumidity());  //lettura umidità
  Serial.println("%\n");

  delay(10000);
}
```

Da notare la linea di comando

```
Serial.begin(9600);
```

La classe `Serial` definisce la comunicazione seriale (ad una baud rate di 9600bps) del microcontrollore con il PC connesso tramite interfaccia USB e offre la possibilità, tramite appositi metodi dedicati, come `.print()`, di interagire in tempo reale con l'MCU con la possibilità di visualizzare stringhe ed il valore delle variabili in quell'istante. Questo strumento è fondamentale per il debug dei programmi, ad esempio, dopo aver compilato e caricato il codice appena visto, cliccando il riquadro in alto a destra è possibile aprire il monitor seriale e verificare il corretto funzionamento del sensore.

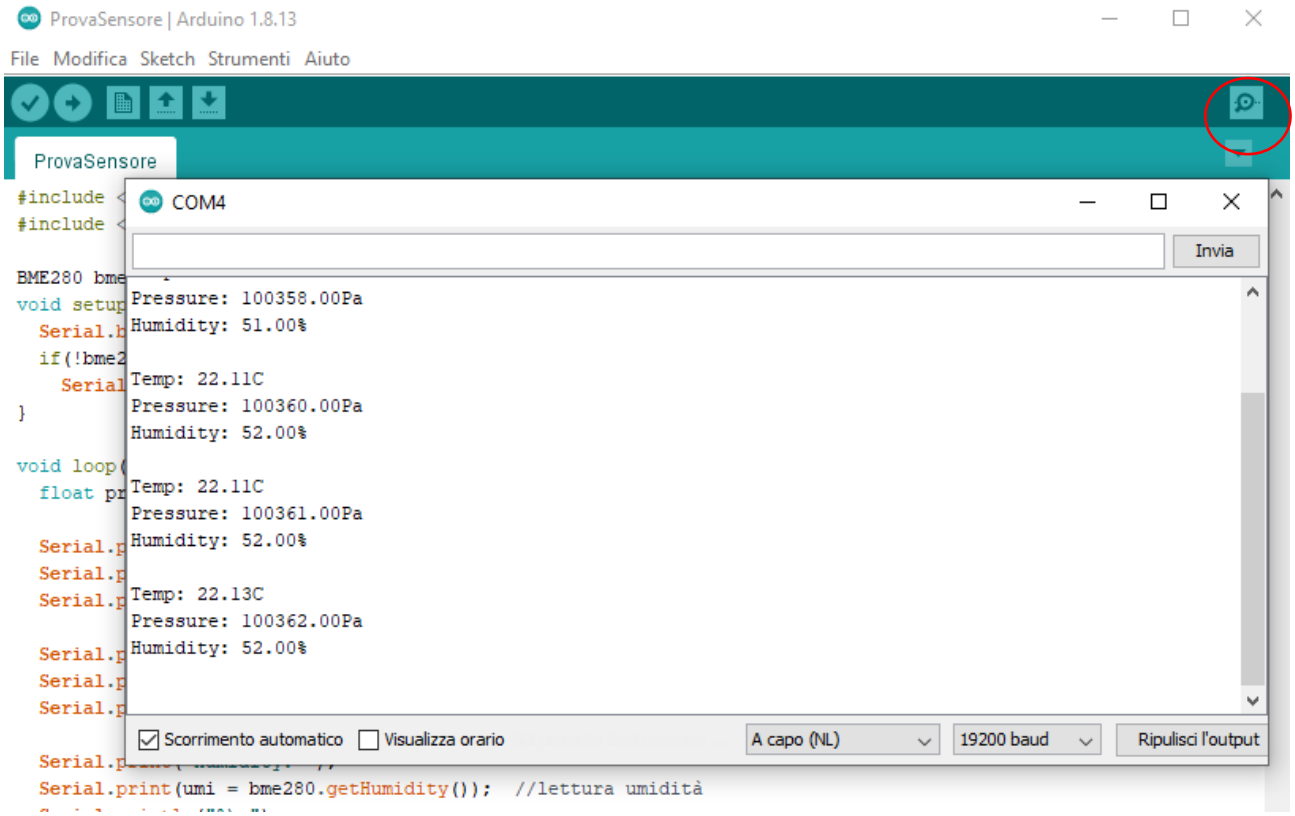

Fig. 4.13: Dettaglio della seriale COM4 aperta tramite l'apposita visual box accessibile dallo shortcut cerchiato in rosso in alta a destra RTC ed SD

4.2.2 RTC & SD

Il collegamento del modulo Datalogger AZDelivery risulta meno intuitivo della configurazione appena descritta in quanto i pin per l'utilizzo dei protocolli I2C ed SPI non sono esplicitamente riportati sulla scheda fisica. Essendo questo componente non un modulo ufficiale Arduino, bensì un componente low cost di seconda fascia, sarebbe potuto essere necessario programmare da zero i singoli bit per la comunicazione seriale, invece ricercando la documentazione sul sito del produttore [39], si può verificare che la piedinatura presenta dei pin prestabiliti per la comunicazione tramite i 2 protocolli SPI ed I2C utilizzati secondo la seguente corrispondenza dei pin:

- A4 →RTC_SDA
- A5 →RTC_SCL
- D10 →SD_CS
- D11 → MOSI
- D12 → MISO
- D13 → SCK

Dato che non ci sono più dubbi a riguardo, sarà possibile operare il collegamento elettrico utilizzando dei jumper che andranno ad interfacciare gli omonimi pin della scheda MKR con i pin appena presentati nell'elenco, aggiungendo ovviamente due jumper per l'alimentazione a 5V e la massa.

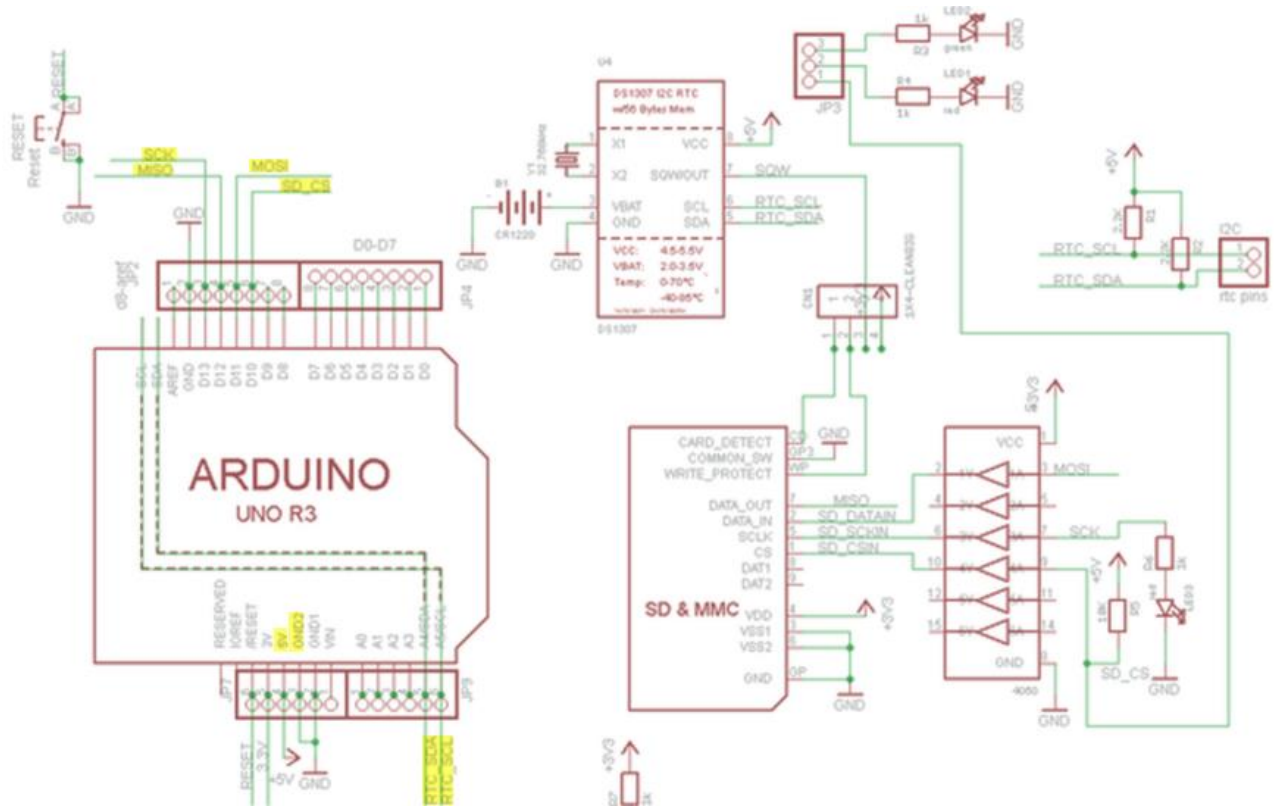

Fig. 4.14: Schematico del datalogger AZDelivery [39]

Completato il collegamento elettrico, si può verificare il corretto funzionamento programmando un semplice datalogger che memorizzi su una scheda SD l'ora letta dal circuito RTC. Aggiungendo le opportune librerie (con lo stesso procedimento descritto poco sopra) per l'utilizzo del modulo SD, della comunicazione SPI ed I2C, disponibili nella documentazione ufficiale Arduino, e una delle tante librerie open source per l'utilizzo dell'RTC DS1307, è possibile implementare una soluzione software come quella presentata nelle prossime righe:

Prova_SD_RTC.ino

```
#include <SD.h>
#include <SPI.h>
#include <RTClib.h>
#include <Wire.h>

RTC_DS1307 rtc;
File logFile;

void setup() {
  Serial.begin(9600);
  Wire.begin();
  rtc.begin();
  rtc.adjust(DateTime(__DATE__, __TIME__));    //Aggironamento dell'RTC
                                                 tramite data ed ora del PC
  if(!SD.begin(7)){                  //Si utilizza il pin 7 della scheda MKR
                                       come CS del modulo SD
    Serial.println("SD non trovata.");
    while(1);
  }
}
void loop() {
  DateTime orario = rtc.now();
  String ora =
    String(orario.hour())
    +":"
    +String(orario.minute())
    +":"
    +String(orario.second());
  Serial.println(ora);
  logFile = SD.open("Orario.txt", FILE_WRITE);
  if(logFile){
    logFile.println(ora);
    Serial.println("Caricamento effettuato");
  }
  logFile.close();
  delay(3000);
}
```

Si noti che quando il programma viene caricato sul microcontrollore, l'RTC viene ricalibrato tramite la riga di codice

```
rtc.adjust(DateTime(__DATE__, __TIME__));
```

così da assicurarsi di riallineare l'orario presente sull'RTC a quello corretto nel caso in cui il circuito non venga usato da molto tempo oppure sia stato scollegato dall'alimentazione.

Per testare il corretto funzionamento del codice, si può aggiungere una scheda SDHC formattata FAT16 al modulo AZDelivery collegato opportunamente al microcontrollore e caricare il programma. Dopo qualche secondo, si troverà al suo interno un file "Orario.txt" con un contenuto simile al seguente:

```
10:57:32
```

```
10:57:35
10:57:38
                                                    10:57:41
                                                    10:57:44
                                                    10:57:47
```

4.3 IMPLEMENTAZIONE WIFI E THINGSPEAK

Oltre alla memorizzazione in loco dei dati raccolti, può risultare molto utile l'impiego di una piattaforma Cloud per la collezione dei dati da remoto così da poter visionare in tempo reale le informazioni rilevate e per poter implementare, in futuro, delle funzioni aggiuntive sempre real time.

La piattaforma Cloud scelta per i rilievi è ThingSpeak [40], la quale permette di raccogliere i dati come ogni altra piattaforma, ma che in più, essendo proprietà MathWorks, permette l'analisi remota dei dati tramite codice MatLab e la visualizzazione dei risultati delle analisi direttamente dall'interfaccia web, rendendo l'operazione di analisi dei dati estremamente semplice, senza dove utilizzare un server proprietario. È comunque richiesto un account ThingSpeak, che per gli studenti universitari è accessibile gratuitamente a fini didattici. In questo documento, la comunicazione wireless implementata per l'invio dei dati alla piattaforma è il WiFi, perciò sarà necessario scrivere software per consentire alla scheda prima di tutto di collegarsi alla rete e successivamente si dovranno inviare i dati. Il seguente esempio permette l'invio di ipotetiche rilevazioni ambientali sulla piattaforma Cloud.

4.3.1 Esempio: Invio dati ambientali su cloud

Come prima operazione è necessario predisporre sulla piattaforma il database personale per la raccolta dei dati. Accedendo tramite credenziali al sito web *ThingSpeak.com* è possibile accedere tramite il menu a tendina alla sezione "*My channels"*, dove il cosiddetto channel, canale, è lo spazio che viene riservato all'utente per il caricamento dei dati e la loro analisi. Cliccando su *New channel* verrà aperto un form in cui inserire i parametri del database che vogliamo generare, dove i parametri più importanti sono i cosiddetti *Fields* ovvero i campi, lo spazio in cui verrà memorizzato un tipo di dato che vogliamo memorizzare. A titolo di esempio sarà creato un nuovo canale con tre campi: Temperatura, Pressione ed Umidità.

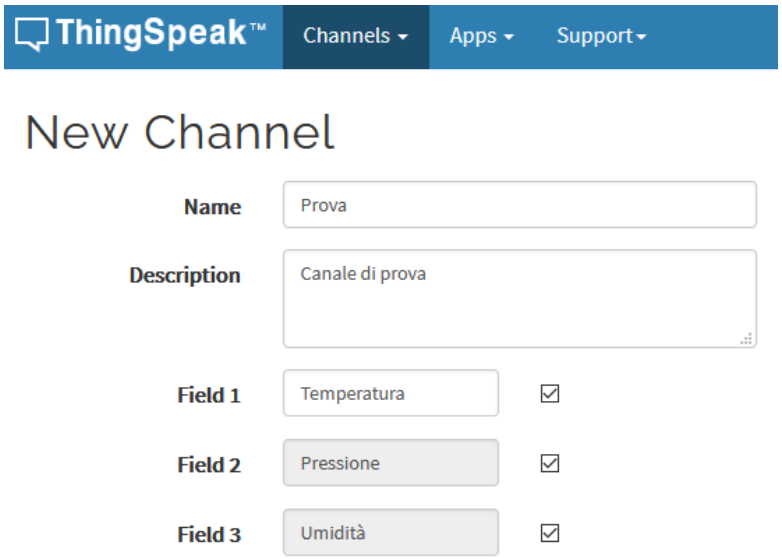

Fig. 4.15: Dettaglio della pagina di compilazione dei campi per la creazione del canale ThingSpeak

Salvando le impostazioni, il nuovo canale verrà finalmente creato e sarà aperta una sezione di gestione del canale dove si potranno vedere i grafici (inizialmente vuoti) dei tre campi definiti e le informazioni del canale. Di particolare interesse sono le informazioni *ChannelID* e *WriteAPIKey*.

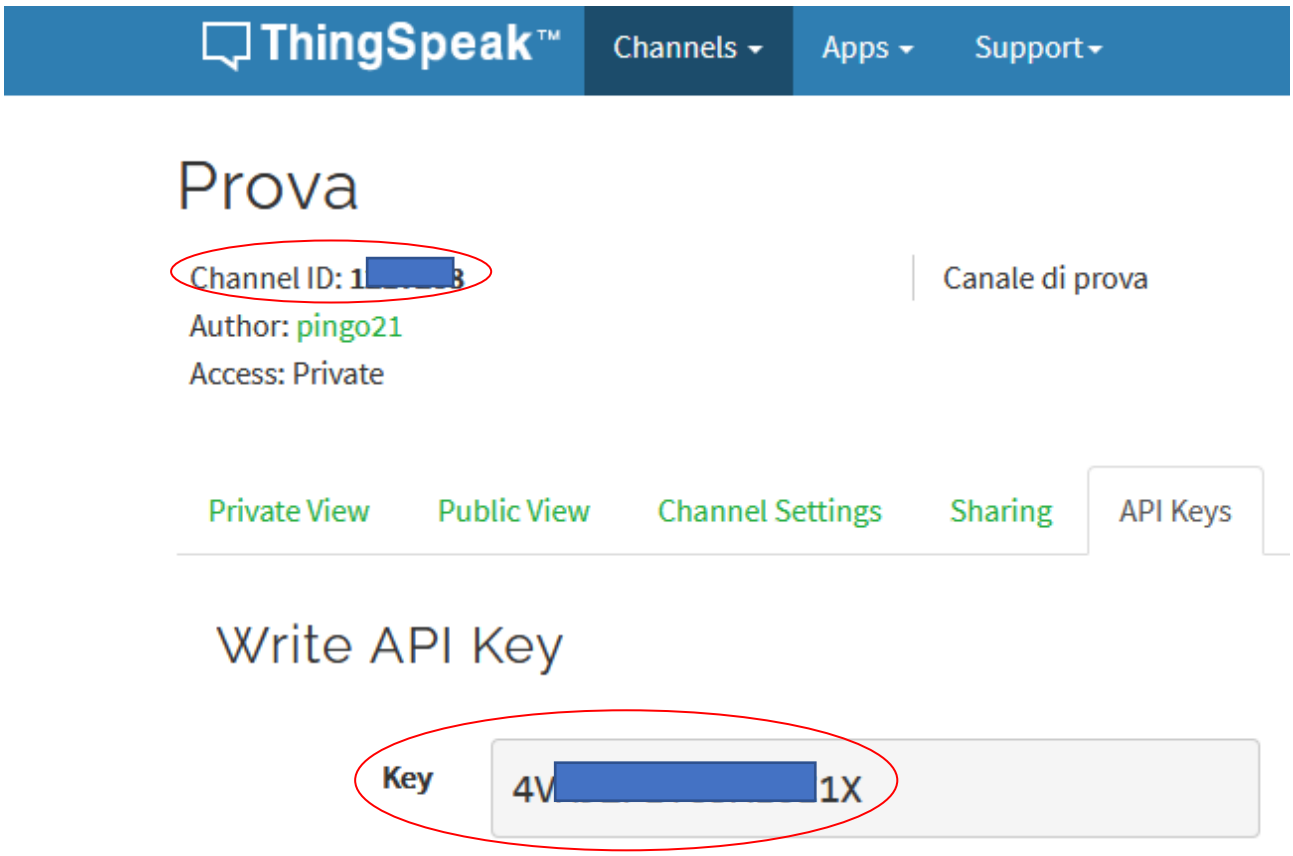

Fig. 4.16: Esplorazione della pagina ThingSpeak per trovare il channel ID e le API Key

Completata la creazione del canale, sarà possibile passare alla programmazione del firmware che, fusa opportunamente con la parte già vista sull'utilizzo del sensore di temperatura, dovrà contenere una funzione che consenta la connessione alla rete (`void connect()`) ed una routine nel loop che consenta il caricamento dei dati su Cloud. Consultando la documentazione delle librerie WiFiNINA by Arduino [41] e ThingSpeak by MathWorks [42] sarà possibile sviluppare una implementazione come quella seguente:

Prova_TS.ino

```
#include <Wire.h>
#include <Seeed_BME280.h>
#include <ThingSpeak.h>//Libreria per interazione con ThingSpeak
#include <WiFiNINA.h>  //Libreria per la comunicazione WiFi

//-------IMPOSTAZIONI DI RETE (WPA/WPA2)----------
char ssid[]="Nome della rete WiFi";
char pass[]="password";
WiFiClient client;
//-------IMPOSTAZIONI TEAMSPEAK-------------------
unsigned long myChannelNumber =1234567;
const char * myWriteAPIKey = "MPXXXXXXXXXXXXO5";

//Funzione per la verifica della connessione e della sua instaurazione
//qualora la connettività non sia presente
void connect(){
  if(WiFi.status() != WL_CONNECTED){
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    while(WiFi.status() != WL_CONNECTED){
      WiFi.begin(ssid, pass);
      Serial.print(".");
      delay(5000);
    }
    Serial.println("\nConnected.");
  }
}

BME280 bme280;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  if(!bme280.init())
    Serial.println("Impossibile contattare il sensore");
  ThingSpeak.begin(client);
}

void loop() {
  connect();

  float pre = bme280.getTemperature();
  float temp = bme280.getPressure();
  float umi = bme280.getHumidity();

  ThingSpeak.setField(1, temp);
```

```
    ThingSpeak.setField(2, pre);
    ThingSpeak.setField(3, umi);
    int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    if(x == 200){
      Serial.println("Dati caricati con successo.");
    }
    else{
      Serial.println("Problema nel caricamento. Codice di errore: " +
                     String(x));
    }
    delay(15000);
  }
```

Si noti che quando si cerca di inviare i dati su Cloud tramite metodo `ThingSpeak.writeFields()`, esso ritorna un valore intero al quale corrisponde un codice di ritorno che viene salvato sulla variabile *x*, il che rende semplice controllare che il caricamento sia avvenuto con successo (x=200: caricamento riuscito con successo).

Una nota aggiuntiva riguarda il delay: la piattaforma ThingSpeak permette caricamenti verso lo stesso canale ogni minimo quindici secondi, perciò appare opportuno caricare più campi insieme come nell'esempio.

Si ricordi inoltre di aggiornare il valore delle variabili `ssid[]` e `pass[]`, che indicano rispettivamente il nome della rete wifi e la sua password, e di aggiornare anche i valori delle variabili `myChannelNumber e myWriteAPIKey` con i valori presenti sulla pagina web del proprio canale.

Caricando il programma, che si suppone funzionante, e lasciando il sistema a lavoro per qualche tempo, sarà possibile notare sulla seriale della scheda (che dovrà ovviamente essere collegata al sensore ambientale come già descritto in questa trattazione), dei risultati analoghi ai seguenti:

Fig. 4.17: Risposta seriale su COM4 durante l'esecuzione dello sketch Prova_TS.ino

Si può inoltre riscontrare il successo del caricamento accedendo al canale ThingSpeak ed aggiornando i grafici automaticamente costruiti dalla piattaforma cloud.

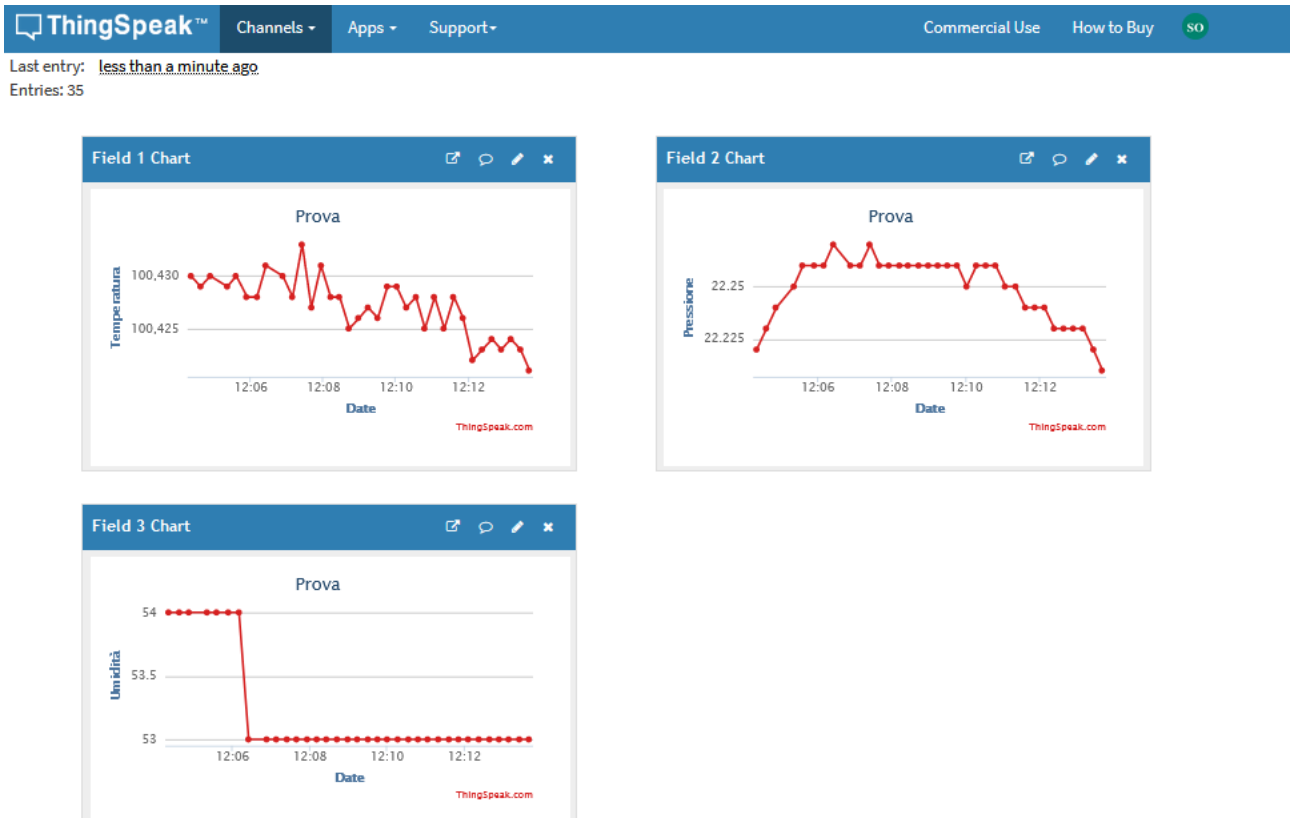

Fig. 4.18: Dettaglio dei grafici derivati dai caricamenti dei dati caricati su ThingSpeak con successo

4.4 IMPLEMENTAZIONE RS485 E MODBUS

Il collegamento elettrico della scheda MKR Shield 485 appare immediato dato che, come già accennato, esse condividono la stessa piedinatura, qualora però si preferisca non collegarli sovrapponendoli fisicamente, si possono collegare tramite jumper semplicemente i seguenti piedini: A5, A6, 6, 7, TX, RX, GND, VCC, VIN, 5V (come si può verificare nella sezione in cui si è parlato dello shield).

Ultimato il collegamento, l'implementazione software sarà possibile attraverso l'utilizzo delle librerie RS485.h [43] e ModBus.h [44] messe a disposizione da Arduino per l'utilizzo dei protocolli omonimi. Come esercizio, si ipotizzi di dover leggere ogni 10 secondi la tensione salvata come float in un holding register di indirizzo 0x00 nel seguente formato

`Volt: xxx.x-V` (ossia tre cifre intere ed una decimale)

su un dispositivo indirizzato da Modbus a 0x01. Una possibile implementazione potrebbe assomigliare alla seguente:

Prova_Mudbus.ino

```
#include <ArduinoRS485.h>
#include <ArduinoModbus.h>
#include <SPI.h>

void setup() {
  Serial.begin(19200);
  while(!Serial);
  if(!ModbusRTUClient.begin(9600)){
    Serial.println("Inizializzazione del MasterRTU fallita.");
    while(1);
  }
}

void loop() {
  if(!ModbusRTUClient.requestFrom(0x01,HOLDING_REGISTERS,0x00,1)){
    Serial.print("failed to read voltage! ");
    Serial.println(ModbusRTUClient.lastError());
  }
  else{
    float let = ModbusRTUClient.read();  //read data from the buffer
    //volt = let;
    Serial.println("Volt: "+String(let/10));
  }
  delay(10000);
}
```

Dove il metodo `ModbusRTUClient.requestFrom()` richiede al server Modbus di indirizzo 0x01 di inviare al client il valore presente nell'holding register di indirizzo 0x00. Il dato richiesto viene poi letto dal buffer di ingresso e salvato nella variabile let.

Si noti che il dato prima di essere stampato viene diviso per dieci per poterlo adattare alla rappresentazione in cui era salvato nel server (si ricorda infatti che gli holding registers contengono una unsigned word).

5 CASE STUDY: CONSUMI ENERGETICI

Per il collaudo del sistema si può provare ad impiegarlo per uno dei possibili usi descritti nell'introduzione, dove il più semplice è una **analisi dei consumi energetici**.

Nel seguente esempio si monitoreranno le informazioni ambientali ed i consumi di tensione, corrente e fattore di potenza di un semplice impianto elettrico, e si effettuerà uno studio dei dati tramite il canale ThingSpeak dedicato, oltre ad essere salvati in loco tramite memoria SD. Per soddisfare tutte le richieste elencate sarà necessario unire tutte le singole implementazioni fin qui presentate così da poterle utilizzare tutte assieme tramite compilazione di un unico firmware.

Per poter operare in sicurezza, senza danneggiare qualche impianto industriale o civile e soprattutto la salute, si prenderà come impianto da analizzare un semplice quadro elettrico autocostruito munito di:

- Interruttore salvavita differenziale magnetotermico ABB DS-941-AC;
- Contatore IME Conto D1 [45];
- Presa di corrente P40 (due unità).

Fig. 5.1: Foto del quadro elettrico utilizzato per il collaudo

5.1 CONTATORE IME CONTO D1

IMEItaly offre nel suo catalogo una gamma di energy meters modulari con integrato un modulo di comunicazione RS485 ed il Conto D1 rientra tra questi, perciò sarà proprio quest'ultimo l'interfaccia tra l'apparato di telerilevamento e l'impianto energetico.

Conto D1 è un contatore monofase di energia che consente di misurare diversi parametri di rete quali: tensione di fase; corrente; energia attiva e reattiva; potenza attiva, reattiva e apparente; fattore di potenza. E' dotato di un display attraverso il quale si possono leggere in tempo reale i dati rilevati e, cosa molto importante, si autoalimenta dalla linea che sta misurando, consentendo una installazione relativamente semplice nella maggior parte degli impianti.

Come accennato, il sistema è stato munito di modulo di comunicazioni RS485 a livello di linea tramite impiego di **3 fili**, ovvero il positivo (non invertente), il negativo(invertente) e GND per l'implementazione half-duplex. A livello applicativo, il protocollo utilizzato è Modbus RTU, dove il bus RS485, isolato galvanicamente dalla linea sotto esame, permette la gestione di un massimo di 32 dispositivi estendibili fino a 247 e prevede che la comunicazione avvenga in modalità **SERIAL_8N1**, ovvero verrà instaurata una connessione seriale asincrona con 8 bit di dati, nessun bit di parità ed un bit di stop. Nel datasheet del componente sono riportate tutte queste informazioni ed anche le possibili velocità di trasmissione (in questo esempio si trasmetterà a 9600bit/s) ed il tempo di risposta massimo (<51ms).

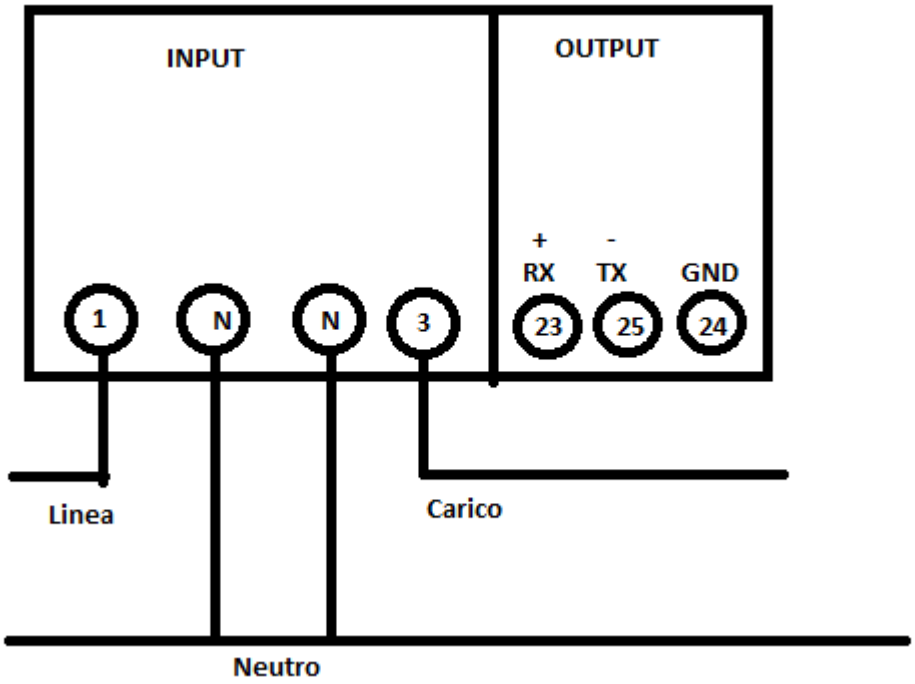

Fig. 5.2: Schema elettrico dell'energy meter IME Conto D1 [45]

Per il collegamento elettrico si dovrà collegare il contatore al pin 1 e 3 rispettivamente la linea monofase in ingresso e verso il carico, oltre a collegare sui pin N il neutro, in questo modo il contatore sarà attraversato dalla linea e potrà alimentarsi e compiere le misurazioni. Per implementare la comunicazione sarà necessario munirsi di un cavo UTP e collegare i 3 fili per la comunicazione half-duplex da portare fino al dispositivo MKR 485 Shield, precisamente agli ingressi ISOGND, Y e Z.

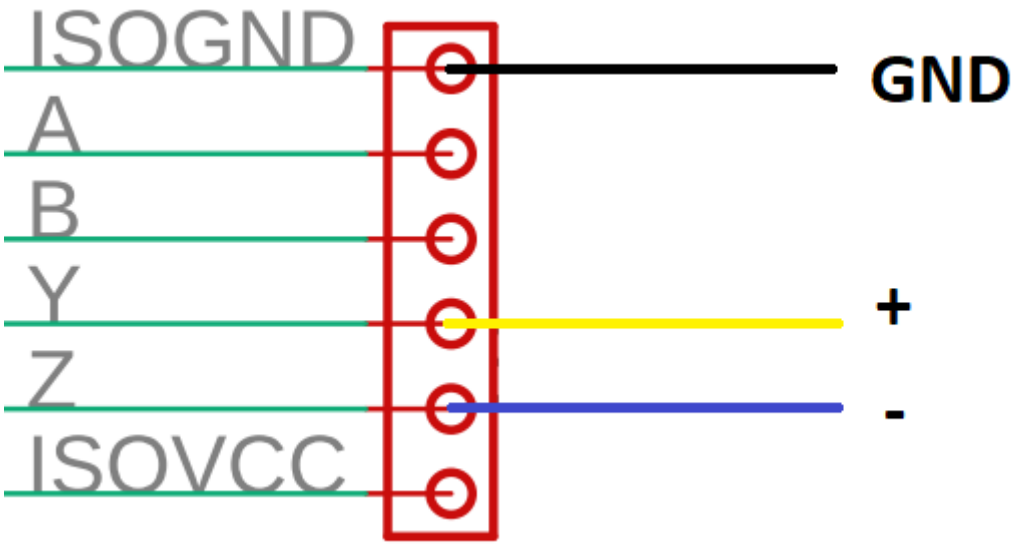

Fig. 5.3: Schema elettrico del interfacciamento del MKR 485 Shield con l'energy meter

5.2 FIRMWARE E LETTURA DEI DATI

Una volta concluso il plugging del sistema alla rete da monitorare, si può passare alla stesura del firmware, che, come per la parte fisica, potrà essere stilato relativamente in poco tempo in quanto sono già stati descritti degli script che eseguono le singole azioni che si intendono compiere in cascata in questo esempio, perciò il grosso del lavoro, come accennato, sarà quello di implementare tutto in un unico script, oltre a creare un nuovo canale ThingSpeak dove collezionare le rilevazioni ed ottenere correttamente i dati dal contatore.

Dal sito IMEItaly si può accedere alla documentazione che descrive come il protocollo Modbus [46] è stato implementato per il dispositivo, e da qui si può verificare che i dati letti dal contatore che devono essere estratti dal MCU vengono registrati in degli holding registers, secondo le seguenti regole di indirizzamento e rappresentazione:

Indirizzo	Formato	Dato	Rappresentazione
0x00	u_word	Tensione	xxx,x V
0x01	u_word	Corrente	xxx,x A
0x06	u_word	Fattore di potenza	-

Tabella 5.1: Caratteristiche degli spazi di memoria di interesse accessibili da client Modbus su Conto D1 [46]

Sarà perciò necessario eseguire tre operazioni di richiesta e di seguente lettura al contatore tramite i comandi visti per l'implementazione software di Modbus. La soluzione software si presenterà in modo analogo alla seguente:

Collaudo.ino

```
#include <SD.h>
#include <SPI.h>
#include <RTClib.h>
#include <Wire.h>
#include <Seeed_BME280.h>
#include <ThingSpeak.h>
#include <WiFiNINA.h>
#include <ArduinoRS485.h>
#include <ArduinoModbus.h>
#include <ArduinoLowPower.h>

//-------IMPOSTAZIONI DI RETE (WPA/WPA2)----------
char ssid[]="--------";
char pass[]="---------";
WiFiClient client;
//-------IMPOSTAZIONI TEAMSPEAK-------------------
unsigned long myChannelNumber =--------;
const char * myWriteAPIKey = "--------------";

//Inizializzazione oggetti: Sensore, Orologio, File di testo.
BME280 bme280;
RTC_DS1307 rtc;
File logFile;

//Funzione richiamata ogni qualvolta il sistema si trovi disconnesso
//dalla rete WiFi
void connect(){
  if(WiFi.status() != WL_CONNECTED){
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    while(WiFi.status() != WL_CONNECTED){
      WiFi.begin(ssid, pass);
      Serial.print(".");
      delay(3000);
    }
    Serial.println("\nConnected.");
  }
}

void setup() {
  //Inizilizzazione delle periferiche e degli oggetti
  Serial.begin(19200);
  Wire.begin();
  rtc.begin();
  rtc.adjust(DateTime(__DATE__, __TIME__));
  if(!SD.begin(7)){ //Si assegna al lettore SD il CS nel pin 7
    Serial.println("SD non trovata.");
    while(1);
  }
  if(!bme280.init())
    Serial.println("Impossibile contattare il sensore");
  ThingSpeak.begin(client);
  if(!ModbusRTUClient.begin(9600)){
    Serial.println("Inizializzazione del MasterRTU fallita.");
    while(1);
  }
}

void loop() {
  connect();
```

```
float pre, temp, umi;
float ten, cor, fat;
DateTime dataOra = rtc.now();

//lettura temperatura
Serial.print("Temp: ");
Serial.print(temp = bme280.getTemperature());
Serial.println("C");

//lettura ed acquisizione pressione
Serial.println("Pressure: ");
Serial.print(pre = bme280.getPressure());
Serial.println("Pa");

//lettura umidità
Serial.print("Humidity: ");
Serial.print(umi = bme280.getHumidity());
Serial.println("%\n");
if(!ModbusRTUClient.requestFrom(0x01,HOLDING_REGISTERS,0x00,1)){
  Serial.print("failed to read voltage! ");
  Serial.println(ModbusRTUClient.lastError());
}
else{
  ten = ModbusRTUClient.read()/10;
  Serial.println("Tensione letta");
}
delay(100);

if(!ModbusRTUClient.requestFrom(0x01,HOLDING_REGISTERS,0x01,1)){
  Serial.print("failed to read current! ");
  Serial.println(ModbusRTUClient.lastError());
}
else{
  cor = ModbusRTUClient.read()/10;
  Serial.println("Corrente letta");
}
delay(100);

if(!ModbusRTUClient.requestFrom(0x01,HOLDING_REGISTERS,0x06,1)){
  Serial.print("failed to read power factor! ");
  Serial.println(ModbusRTUClient.lastError());
}
else{
  fat = ModbusRTUClient.read()/1000;
  Serial.println("cosphi letto");
}

ThingSpeak.setField(1,temp);
ThingSpeak.setField(2,pre);
ThingSpeak.setField(3,umi);
ThingSpeak.setField(4,ten);
ThingSpeak.setField(5,cor);
ThingSpeak.setField(6,fat);
int x=ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
if(x==200)
  Serial.println("Caricamento su TS effettuato");
else
  Serial.println("Problema caricamento su TS. Codice di errore:
                  "+String(x));

String logStr =
    String(dataOra.day())+
    "/"+String(dataOra.month())+
```

```
            "/"+String(dataOra.year())+
            ","+String(dataOra.hour())+
            ":"+String(dataOra.minute())+
            ":"+String(dataOra.second())+
            ","+String(temp)+
            ","+String(pre)+
            ","+String(umi)+
            ","+String(ten)+
            ","+String(cor)+
            ","+String(fat);
        Serial.println(logStr);
        if(!SD.exists("data.txt")){
          logFile = SD.open("data.txt", FILE_WRITE);
          logFile.println("aaaa/mm/gg,hh:mm:ss,Temperatura(C°),Pressione(Pa),Umidi
                           tà(%),Tensione(V),Corrente(A),Fattore di potenza");
        }
        else
          logFile = SD.open("data.txt", FILE_WRITE);
        if(logFile){
          logFile.println(logStr);
          Serial.println("Caricamento su SD effettuato");
        }

        logFile.close();
        delay(15000);
    }
```

Le rilevazioni verranno effettuate, come suggerito dal delay all'ultima riga di codice, con un periodo di quindici secondi.

Per fare delle rilevazioni di prova si collegheranno alle prese una semplice **stufa elettrica** ed un **caricabatterie** per smartphone, ed il sensore di temperatura verrà posizionato in prossimità della stufa. Il sistema di collaudo nel suo complesso si presenterà quindi come il seguente:

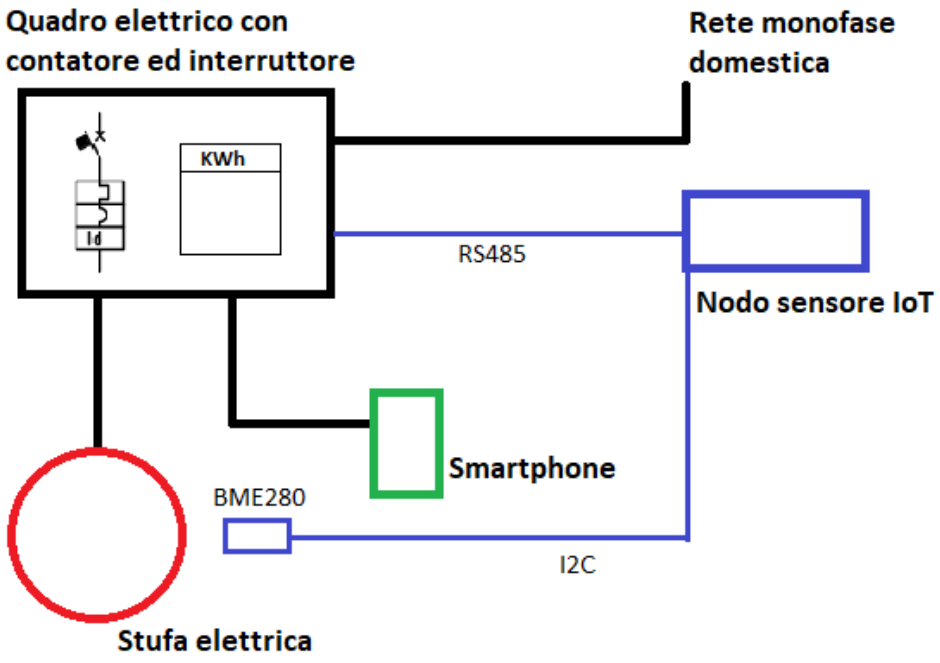

Fig. 5.4: Schema a blocchi del sistema di collaudo nodo IoT

Una volta caricato il firmware, il sistema inizierà la routine di telerilevamento, il cui corretto funzionamento è verificabile attraverso la seriale. Ogni ciclo della funzione loop dovrebbe restituire un risultato analogo al seguente:

Fig. 5.5: Risposta sulla seriale COM4 durante l'esecuzione dello sketch Collaudo.ino

5.2.1 Studio dei risultati su ThingSpeak tramite MatLab

Lasciando il sistema acceso per un certo lasso di tempo trenta minuti circa, si otterrà un campione di dati sufficiente per concludere la prova. Aprendo il canale ThingSpeak creato appositamente per questo collaudo e modificando opportunamente i grafici presenti sulla sezione "Private view", si otterranno dei risultati visivi simili ai seguenti:

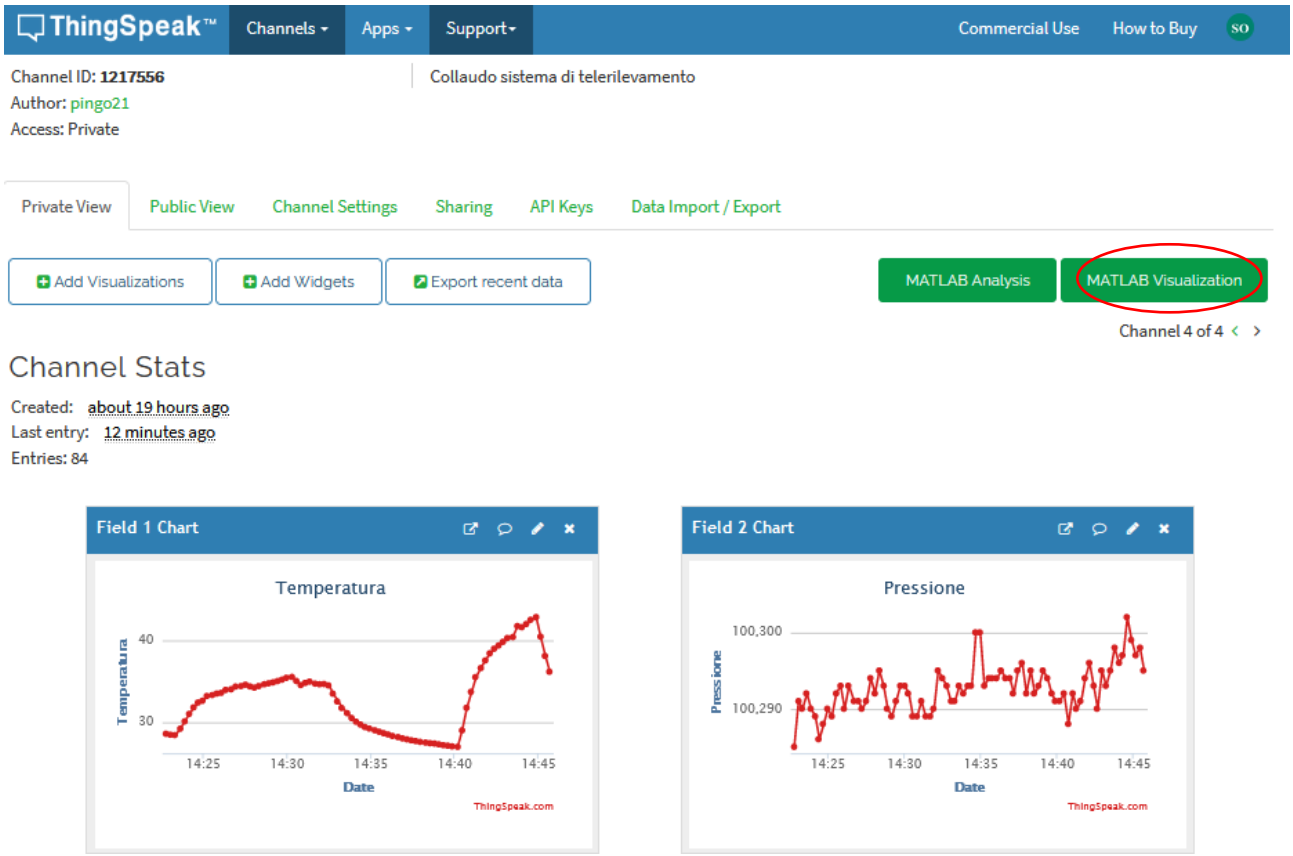

Fig. 5.6: Grafici riportanti i dati caricati con successo su ThingSpeak

Da tali dati si intuisce come il consumo maggioritario della potenza è da arrogare alla stufa elettrica, dato che nei momenti in cui esso viene acceso a potenza intermedia, il consumo di corrente passa da quasi nullo a 3/4A, quando è a potenza massima il consumo di corrente passa a 7A e quando è spento il consumo ritorna presso lo zero. Tali ipotesi sono supportate dal fatto che quando il consumo passa da zero a 3A, i dati ambientali cominciano a variare, in particolare la temperatura sale secondo una curva monotona. Quando la corrente torna verso 0A, si può notare che nello stesso momento la curva della temperatura diventa decrescente, mentre quando la corrente passa a 7A la temperatura sale ancor più velocemente di prima, se ne deduce perciò con sicurezza che il consumo di corrente è monopolizzato dallo scaldino, che ha perfettamente senso in quanto le bobine al suo interno si scaldano per effetto Joule, dissipando notevoli quantità di energia elettrica in calore.

Per effettuare uno studio dei dati tramite piattaforma cloud, sarà sufficiente recarsi sulla sezione “MATLAB Visualizzation” e selezionare uno degli script parzialmente precompilati che ThingSpeak propone come spunto iniziale. Modificando opportunamente lo script a piacimento sarà possible implementare seguendo la documentazione, dei grafici come I seguenti:

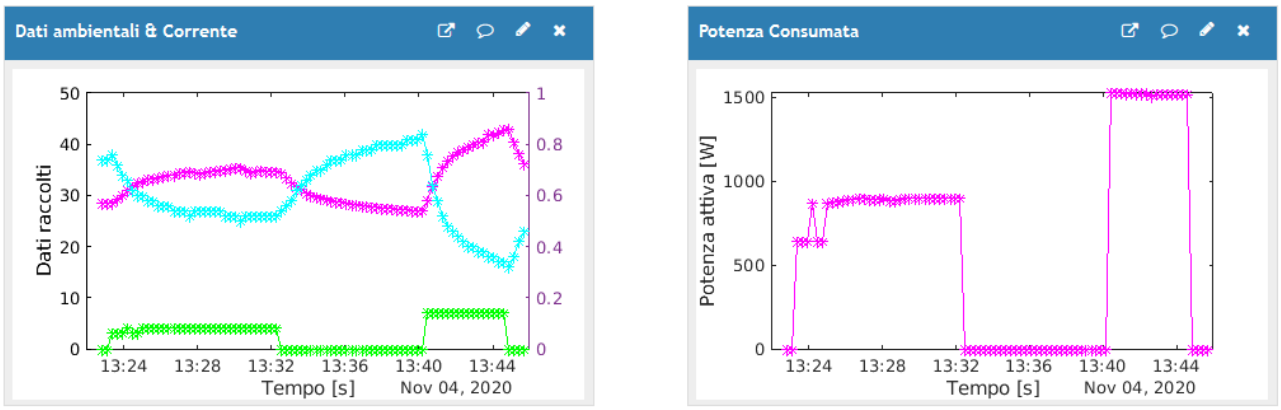

Fig. 5.7: Grafici custom per la visualizzazione dei dati creati tramite MatLab su ThingSpeak. Nel grafico a sinistra si segue la seguente legenda: Magenta = Temperatura; Azzurro = Umidità; Verde = Corrente.

Si noti il grafico a sinistra, descrive in un'unica immagine ciò che è stato spiegato poco sopra allargando la trattazione alle variazioni di umidità: esso ci consente di apprezzare la correlazione dei dati ambientali ed il consumo di corrente, che in questo caso sottolineano come quando lo scaldino viene attivato, si ha effettivamente un riscaldamento dell'ambiente con annessa diminuzione dell'umidità per ovvi effetti termodinamici.

Per finire si può notare come il grafico a destra rappresenti il calcolo della potenza attiva consumata tramite una semplice moltiplicazione tra tensione e corrente (questo dato è veritiero dal momento che il fattore di potenza resta sempre al valore unitario).

Seguendo la stessa logica appena utilizzata, sarà possibile studiare sistemi molto più complessi e mettere in correlazione eventi che senza un sistema di telerilevamento non sarebbero mai stati collegati tra loro, rivelando le potenzialità del prototipo con questo semplice esempio sui consumi energetici.

6 SVILUPPI FUTURI

Grazie alla filosofia seguita per la progettazione di un simile prototipo, si è riusciti a creare un sistema pronto all'uso che manca di efficienza, ma prepara il terreno per l'implementazione di numerosissime applicazioni e permette di implementare altrettanti ampliamenti molto rapidamente, dato che sono state privilegiate caratteristiche già citate di flessibilità, riusabilità e semplicità.

6.1 ARDUINO LOWPOWER.H

Un primo upgrade che si potrebbe subito attuare nel sistema a microcontrollore progettato è l'implementazione della libreria <ArduinoLowPower.h> [47]: la scheda MKR 1010 WiFi, grazie all' MCU che implementa, consente l'attivazione di una modalità a bassissimo consumo che manda in ibernazione le funzioni non essenziali per la scheda, lasciando attiva solamente una piccola parte del processore per consentire l'elaborazione di un segnale di risveglio e quindi la ripartenza del microcontrollore ad un normale funzionamento. Per implementare questa miglioria basterà modificare il programma così da sostituire tutte le funzioni:

```
delay();
```

con il metodo (contenuto nella libreria già citata):

```
LowPower.deepSleep();
```

6.2 IMPLEMENTAZIONE TRAMITE FOTODIODO

Come prima idea per avanzare con la progettazione, lo studio dell'Ing. Caldarigi pensa alla possibilità di implementare, al posto della comunicazione RS485, un fotodiodo con l'intento di risolvere il problema del consumo energetico, dato che, come già accennato, non è progettato per essere efficiente, ma è predisposto ad essere modificato a favore dell'efficienza.

Per l'installazione dell'sistema progettato è necessaria un'alimentazione costante che perduri per un periodo di tempo medio lungo, ma dato che l'installazione tramite connessione alla rete elettrica prevederebbe dei costi per l'installazione, l'ideale sarebbe l'utilizzo di una

alimentazione con batterie al litio, ma l'apparato per effettuare le misurazioni con i protocolli implementati in questa trattazione dovrà rimanere sveglio per un periodo di tempo non banale e quindi non è scontato che esso possa rimanere funzionante per un periodo di tempo sufficiente per eseguire le rilevazioni senza dover cambiare la batteria, cosa che avrebbe un costo non irrisorio visto che sarebbe necessario portare un addetto sul posto. Per questo una possibile soluzione al problema potrebbe essere l'utilizzo di un fotodiodo che, leggendo i lampeggi dal contatore, permetterà al sistema di mantenere un regime di bassissimo consumo (in quanto basterà leggere solamente un pin digitale della scheda), la quale auspicabilmente potrà essere mantenuta in una modalità di ibernazione in modo permanente e di essere svegliato unicamente quando il contatore inizierà a rilevare un consumo di energia, consentendo la durata della batteria per un tempo sufficientemente lungo per eseguire tutte le misure necessarie allo studio.

6.3 MKR 1400 GSM

Una ulteriore misura di ampliamento per rendere il sistema il più indipendente possibile durante la fase di installazione e raccolta dei dati, può essere l'utilizzo della già citata scheda MKR 1400 GSM, la quale richiederebbe una modifica unicamente alla parte dove si tratta del protocollo di comunicazione WiFi. L'implementazione del protocollo GSM sarebbe molto semplice in quanto, come è stato per il WiFi, Arduino mette già a disposizione delle librerie verificate ed efficienti molto facili da programmare.

6.4 PRODUZIONE IN SERIE

Grazie a questo progetto sarebbe anche possibile la produzione in serie di schede PCB che portino a bordo tutti i componenti fino ad ora utilizzati in unica soluzione ottimizzata a livello di spazio, portabilità, affidabilità e consumi. L'idea è quelli di poter offrire al cliente una scheda dedicata che permetta un monitoraggio permanente dell'impianto che si intende controllare a fronte di un prezzo irrisorio rispetto alle soluzioni della concorrenza, con la possibilità in più di poter implementare delle funzioni personalizzate in base alle esigenze del cliente, come delle funzioni di analisi predittive e proattive in modo da poter individuare una criticità prima che essa si presenti.

6.5 MACHINE LEARING & EDGE COMPUTING

Come già accennato nell'introduzione, si possono implementare degli algoritmi edge computing e di machine learning che permettano al sistema di implementare funzioni sempre più intelligenti e perfettamente adatte alle richieste del cliente, come appunto le già citate analisi predittive e proattive che permettono tramite appositi algoritmi la predizione dei guasti.

Appare evidente che i possibili sviluppi di un progetto come quello presentato non sono né pochi né banali e lasciano ampio spazio alla creatività del progettista, d'altronde, come diceva Einstein, "*L'immaginazione è più importante della conoscenza*".

7 BIBLIOGRAFIA

[1] Wikipedia, «Industria 4.0,» [Online]. Available: https://it.wikipedia.org/wiki/Industria_4.0.

[2] Electric, Shneider, [Online]. Available: https://www.se.com/ww/en/product/EBX510/comx-510---energy-server---ul---sdram4-go/.

[3] Wikipedia, «Edge Computing,» [Online]. Available: https://it.wikipedia.org/wiki/Edge_computing.

[4] Oracle, «What is IoT,» [Online]. Available: https://www.oracle.com/it/internet-of-things/what-is-iot/.

[5] G. Ceresini, «Voltimum Italia,» 2006. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiMtK2omensAhUJzqQKHRaCA_gQFjAAegQIAhAC&url=https%3A%2F%2Fwww.voltimum.it%2Fsites%2Fwww.voltimum.it%2Ffiles%2Fit%2Fattachments%2Fpdi%2Ff.

[6] G. Ceresini, «Voltimum Italia,» 2006. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiMtK2omensAhUJzqQKHRaCA_gQFjABegQIARAC&url=https%3A%2F%2Fwww.voltimum.it%2Fsites%2Fwww.voltimum.it%2Ffiles%2Fit%2Fattachments%2Fpdi%2Ff%2F061023_guida_armoniche2.pdf&usg=AOvVaw0Bwh9.

[7] Arm, «Introduction to Arm Architecture,» 2019. [Online]. Available: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/Introducing%20the%20Arm%20architecture.pdf?revision=0b6b67ea-edd7-4975-b23c-d4904ac0cb35.

[8] notebookcheck.net, «Qualcomm-Snapdragon-765G,» 2020. [Online]. Available: https://www.notebookcheck.net/Qualcomm-Snapdragon-765G-Processor-Benchmarks-and-Specs.454066.0.html.

[9] Arm, «ArmV6-M,» 2007. [Online]. Available: https://documentation-service.arm.com/static/5f8ff05ef86e16515cdbf826?token=.

[10] Arm, «A32 Instruction Set,» [Online]. Available: https://developer.arm.com/architectures/instruction-sets/base-isas/a32.

[11] Wikipedia, «Arm Architecture,» [Online]. Available: https://en.wikipedia.org/wiki/ARM_architecture#Thumb.

[12] Arm, «Arm Cortex M0+,» 2012. [Online]. Available: https://documentation-service.arm.com/static/5f044dd9cafe527e86f5dc7f?token=.

[13] STM, «STM 32-bit Arm Cortex MCUs,» 2020. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.

[14] Arduino, «Introduction,» 2018. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction.

[15] u-blox, «NINA-W102,» 2019. [Online]. Available: https://content.arduino.cc/assets/Arduino_NINA-W10_DataSheet_%28UBX-17065507%29.pdf.

[16] Microchip, «SAMD 21 Family,» 2018. [Online]. Available: https://content.arduino.cc/assets/mkr-microchip_samd21_family_full_datasheet-ds40001882d.pdf.

[17] Arduino, «MKR 1010 WiFi,» 2020. [Online]. Available: https://store.arduino.cc/arduino-mkr-wifi-1010.

[18] Arduino, «MKR GSM 1400,» 2020. [Online]. Available: https://store.arduino.cc/arduino-mkr-gsm-1400-1415.

[19] Seeed, «Grove BME280,» 2020. [Online]. Available: https://wiki.seeedstudio.com/Grove-Barometer_Sensor-BME280/.

[20] Bosch, «BME280,» 2015. [Online]. Available: https://files.seeedstudio.com/wiki/Grove-Barometer_Sensor-BME280/res/Grove-Barometer_Sensor-BME280-.pdf.

[21] Arduino, «Grove BME 280,» 2020. [Online]. Available: Fonte: https://store.arduino.cc/grove-temp-humi-barometer-sensor-bme280?queryID=undefined.

[22] Maxim Integrated, «DS1307,» 2015. [Online]. Available: https://www.maximintegrated.com/en/products/analog/real-time-clocks/DS1307.html.

[23] Adafruit, «DS1307 RTC breakout board,» 2020. [Online]. Available: Fonte: https://www.adafruit.com/product/264.

[24] Adafruit, «MicroSD breakout board,» 2020. [Online]. Available: Fonte: https://www.adafruit.com/product/4682.

[25] AZDelivery, «Datalogger,» 2020. [Online]. Available: Fonte: https://www.az-delivery.de/en/collections/arduino-zubehor/products/datenlogger-modul.

[26] Arduino, «MKR 485 Shield,» 2020. [Online]. Available: https://store.arduino.cc/arduino-mkr-485-shield.

[27] Maxim Integrated, «MAX3157,» 2020. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX3157.pdf.

[28] Arduino, «Schema elettrico MKR 485 Shield,» 2018. [Online]. Available: https://content.arduino.cc/assets/MKRWiFi1010V2.0_sch.pdf.

[29] Wikipedia, «Daisy chain,» 2020. [Online]. Available: https://en.wikipedia.org/wiki/Daisy_chain_(electrical_engineering).

[30] Courtesy of Texas Instruments, «The RS485 Design Guide,» 2008. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwi56af909zsAhUC_KQKHU3gD_kQFjACegQIAhAC&url=https%3A%2F%2Fwww.ti.com%2Flit%2Fpdf%2Fslla272&usg=AOvVaw0_szJp9VneGxxWDHLMYHSf.

[31] Modbus.org, «Modbus Application Protocol Specification,» 2012. [Online]. Available: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

[32] Philips, «I2C manual,» 2003. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwimocjz7OHsAhVIDewKHVAxB2IQFjABegQIAhAC&url=https%3A%2F%2Fwww.nxp.com%2Fdocs%2Fen%2Fapplication-note%2FAN10216.pdf&usg=AOvVaw19uKAJx3Y4Q2a2etn6O8v0.

[33] Wikipedia, «I2C protocol,» 2020. [Online]. Available: https://fr.wikipedia.org/wiki/I2C#/media/Fichier:I2C_Architecture_2.svg.

[34] Wikipedia, «Serial Peripheral Interface,» 2020. [Online]. Available: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.

[35] Arduino, «Arduino Software Download,» 2020. [Online]. Available: https://www.arduino.cc/en/software.

[36] Wikipedia, «Super Loop Architecture,» 2020. [Online]. Available: https://en.wikibooks.org/wiki/Embedded_Systems/Super_Loop_Architecture.

[37] Arduino, «MKR 1010 WiFi Schematics,» [Online]. Available: https://content.arduino.cc/assets/MKRWiFi1010V2.0_sch.pdf.

[38] Seeed, «GitHub - GroveBME280 Library,» 2020. [Online]. Available: https://github.com/Seeed-Studio/Grove_BME280.

[39] AZDelivery, «Datalogger Schematics,» [Online]. Available: https://cdn.shopify.com/s/files/1/1509/1638/files/DatenLogger_Modul_Logging_Shield_Schematics.rar?4377813253950635206.

[40] ThingSpeak, «Introduction Page,» 2020. [Online]. Available: https://thingspeak.com/.

[41] Arduino, «WiFi NINA Library,» 2020. [Online]. Available: https://www.arduino.cc/reference/en/libraries/wifinina/.

[42] MathWorks, «GitHub ThingSpeak Library for Arduino,» 2020. [Online]. Available: https://github.com/mathworks/thingspeak-arduino.

[43] Arduino, «RS485 Library,» 2020. [Online]. Available: https://www.arduino.cc/en/Reference/ArduinoRS485.

[44] Arduino, «Modbus Library,» 2020. [Online]. Available: https://www.arduino.cc/en/ArduinoModbus/ArduinoModbus.

[45] IME Italy, «Conto D1 Introduction & Documentation,» 2020. [Online]. Available: https://www.imeitaly.com/prodotto/conto-d1-ce11165a4/.

[46] IME Italy, «Conto D1 Technical-sheet,» 2016. [Online]. Available: https://www.imeitaly.com/wp-content/uploads/technical-sheets/PR133.pdf.

[47] Arduino, «LowPower Library,» 2020. [Online]. Available: https://www.arduino.cc/en/Reference/ArduinoLowPower.

8 RINGRAZIAMENTI

Ringrazio di cuore l'Ing. Caldarigi, che da subito ha riposto in me la sua totale fiducia dimostrandosi sempre molto disponibile, per l'occasione che mi ha offerto, permettendomi di conoscere la sua realtà e quindi le possibilità che il mondo del lavoro ha da offrire ad un ingegnere elettronico anche in una piccola azienda di provincia.

Ringrazio il professor Aldo Romani per la sua sincerità e la pazienza, sempre tempestivo nell'aiutarmi e disponibile. Non da meno è la mia gratitudine per i suoi preziosi insegnamenti, che mi hanno indubbiamente aiutato a trovare la mia strada.

Un particolare ringraziamento lo devo alla mia famiglia, che mi ha supportato nel percorso sia dal punto di vista economico, ma soprattutto da quello emotivo, credendo in me quando anche io stentavo a farlo. Questo risultato lo dedico a voi.

Ringrazio i miei cari amici, fondamentale punto di riferimento della mia vita, sempre presenti in tutti i momenti, sia piacevoli che meno.

Un grazie speciale lo rivolgo ai miei coinquilini, compagni di avventure insostituibili che hanno reso il mio percorso universitario una esperienza di vita insostituibile. Nonostante le nostre strade si siano divise, resterete per sempre nel mio cuore. In bocca al lupo.