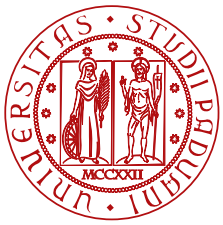


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di laurea in Ingegneria Elettronica

Sistema di monitoraggio remoto basato su BLE Adafruit FeatherSense e Google Sheets

Relatore
Prof.ssa Giorgi Giada

Laureando
Facco Guido

Matricola
2000112

ANNO ACCADEMICO 2023-2024

Data di laurea 23/09/2024

Sommario

In questo elaborato si sviluppa un sistema di monitoraggio remoto che tramite Bluetooth Low Energy invia i dati ricevuti da un qualsiasi sensore ad uno smartphone Android, che a sua volta grazie ad una connessione internet li tabula su un foglio di calcolo online. Il problema nasce dall'esigenza di interfacciare la scheda *Adafruit FeatherSense nRF52480* ad un Computer in modalità wireless, infatti nella forma definitiva il dispositivo deve essere wearable, che raccoglie dati tramite un pulsossimetro esterno che dialoga con la scheda principale tramite protocollo *I2C*. Successivamente in modalità wireless (infatti la board può essere alimentata da una batteria esterna) invia dati con uno streaming costante ad un elaboratore. Nella prima parte di questo elaborato si esamina la scheda con tutte le sue potenzialità e risorse. Un capitolo è dedicato alla spiegazione generica del *Bluetooth LE*, verrà spiegata la differenza dal *Bluetooth* tradizionale e in particolare si svilupperà la trasmissione *Broadcast*, ovvero una connessione *One-Way* tra due dispositivi. Successivamente un breve codice in *CircuitPython* che permette di testare i comandi base del *BLE* e poi di eseguire un semplice check della board e della connessione. Dopo questo passaggio viene proposto uno sketch intermedio per collegare e verificare che il sensore esterno sia a tutti gli effetti funzionante ed in grado di inviare i dati alla scheda. Nel capitolo successivo si fondono assieme le capacità sviluppate nel *Capitolo 4* e nel *Capitolo 5*, ovvero si inviano i dati tramite BLE ad uno smartphone *Android*, ed infine quest'ultimo li carica grazie ad una connessione ad internet su Google Fogli.

Indice

1 Introduzione **1**

2 Materiale utilizzato e funzionamento della scheda con *Circuitpyton* e *C Arduino* **3**

2.1 Utilizzo con *Circuitpyton* 6

2.2 Utilizzo con *C Arduino* 6

3 Funzionamento Bluetooth Low Energy (BLE) **9**

4 Prima prova di collegamento Bluetooth Low Energy **13**

4.1 Codice di prova in *Circuitpyton* 13

4.2 App *Bluefruit Connect* 14

5 Collegamento tramite *I2C* del sensore esterno **17**

5.1 Lettura datasheet e caratteristiche del sensore 17

5.2 Collegamento fisico del sensore 20

5.3 Collegamento software e verifica dati ricevuti con *Arduino IDE* 22

6 Invio dati ricevuti dal sensore a *Google Fogli* **25**

6.1 Setup attrezzatura e problemi riscontrati 25

6.2 Codice *C Arduino* 26

6.3 Setup Google Fogli 29

6.4 Sviluppo App per *Android* 30

6.5 Funzionamento del sistema completo 31

7 Conclusioni e Possibili migliorie **35**

Bibliografia e Sitografia **37**

Capitolo 1

Introduzione

In questo elaborato si propone una soluzione sviluppata nel periodo di tirocinio presso l'Università degli Studi di Padova per produrre un prototipo di pulsossimetro completamente wireless e modificabile sia nel software che nell'hardware. Verranno spiegati i problemi e le limitazioni incontrate nello sviluppo del progetto, e le soluzioni trovate; alcune non sono state percorse perché richiedevano hardware supplementare che non sarebbe arrivato in tempo per la fine del tirocinio, ma non per questo sono meno valide di quella proposta. Il tirocinio, come l'elaborato si sviluppa in due parti:

- nella prima si presenta la scheda *Adafruit FeatherSense* e il BLE per creare degli sketch sia con *Circuitpyton* che con *C Arduino*. [*Capitolo 2*; *Capitolo 3*; *Capitolo 4*]
- nella seconda si sviluppa la parte di progetto vera e propria, usando un sensore di temperatura integrato nella scheda (BMP280) per raccogliere i dati della temperatura ambientale, ed infine per standardizzare la comunicazione tra scheda e sensore si usa sensore digitale di temperatura esterno. [*Capitolo 5*; *Capitolo 6*]

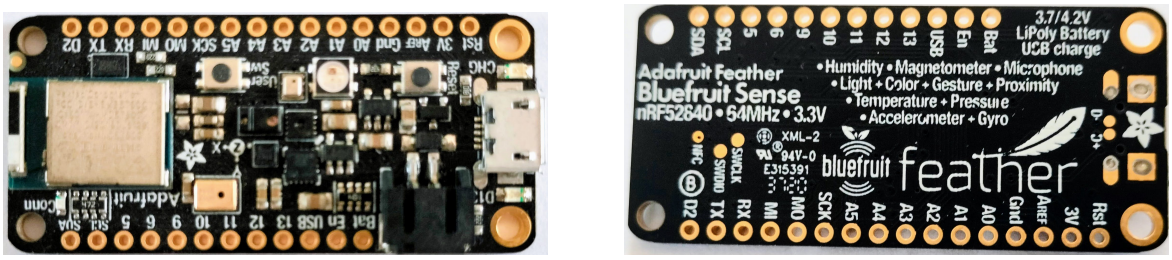
La soluzione non è esente da limitazioni, anche particolarmente complesse. Al *Capitolo 7* proporrò alcune soluzioni alternative o migliorie che possono essere prese in considerazione per migliorare il tutto.

Capitolo 2

Materiale utilizzato e funzionamento della scheda con *Circuitpyton* e *C Arduino*

Per questo progetto abbiamo utilizzato:

1. La scheda *Adafruit FeatherSense nRF52480*:



(a) Fronte. (b) Retro.

Figura 2.1: Scheda *Adafruit FeatherSense nRF52480*.

Una scheda che pur essendo molto contenuta (51 x 23 x 8 mm) possiede moltissimi sensori e caratteristiche interessanti:

- Accelerometro e Giroscopio (LSM6DS3TR-C).
- Magnetometro (LIS3MDL).
- Sensore di prossimità, luminosità, colore e *gesture* (APDS9960).
- Microfono.
- Sensore di umidità.
- Sensore di temperatura e pressione barometrica.
- USB nativa open Source.

- Arm Cortex M4F (64MHz).
- Compatibilità con protocolli SPI, UART, I2C e I2S.

Ovviamente ve ne sono moltissime altre, ad ogni modo lo scopo di questo elaborato non è quello di scoprire ed analizzare tutte le funzionalità che mette a disposizione la *board*. Per questo scopo rimando all'elaborato precedente a questo [1] ed alla pagina ufficiale *Adafruit* [2]. Inoltre la scheda può essere utilizzata in 2 diversi linguaggi come vedremo nelle sezioni 2.1 e 2.2. Anticipo già che solo nel primo collegamento della scheda tramite BLE ho usato *CircuitPyton*, successivamente ho eseguito tutto in *C Arduino* per la maggior reperibiltà di informazioni su codice e librerie.

2. Kit di sensori *KEYSIGHT*: Questo kit è stato fornito per sostituire il pulsossimetro e completare l'elaborato.

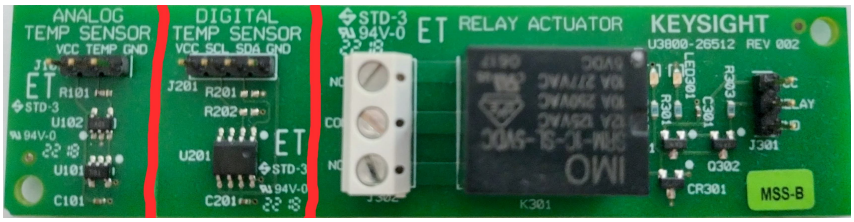


Figura 2.2: Kit sensori

È una scheda che contiene 2 sensori di temperatura ed un relè:

- Il sensore di temperatura analogico, a sx, non contiene registri e nemmeno utilizza un protocollo di scambio dati.
- Il relè, la parte più a dx, è un dispositivo di *output*.
- Il sensore di temperatura digitale(*LM75A*), la parte più centrale, elabora i dati di temperatura in locale e li salva in un registro, quindi tramite il protocollo *I2C* si possono accedere e leggere i dati presenti nel registro. In questo modo si avranno già i dati corretti e pronti all'uso senza doverli elaborare con il controllore.

Da quanto appena riportato l'unico sensore utile per lo scambio di informazioni digitali è il sensore di temperatura digitale.

3. Cavo USB:

Figura 2.3: USB TipoA - micro USB.

Il cavo sarà necessario per configurare la scheda (vedi sezioni 2.1 e 2.2), per caricare il codice e per scambiare dati con il pc.

4. Jumpers e Breadboard:

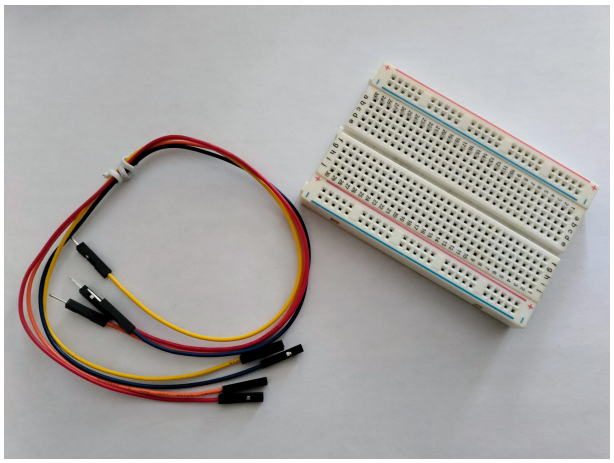


Figura 2.4: Jumpers e breadboard.

Servono per collegare il sensore esterno alla scheda con il protocollo *I2C* che utilizza 4 connettori diversi.

5. Smartphone Android, la prova è stata fatta utilizzando un creatore di app online *Mit App Inventor* che produce applicazioni in formato apk.

6. PC per programmare e per elaborare i dati.

2.1 Utilizzo con *Circuitpyton*

Per utilizzare la scheda con *Circuitpyton* bisogna seguire dei semplici passaggi:

1. Scaricare sul PC l'editor di testo *Mu* e una volta lanciato impostare *CircuitPyton* come linguaggio.

2. Scaricare dal sito di *CircuitPyton* il firmware per la scheda.

3. Collegare la scheda al PC e portarla in modalità bootloader premendo 2 volte il tasto RESET. Potrebbe non funzionare al primo tentativo, nel caso riprovare. Una volta completata l'operazione il led di stato lampeggerà di rosso per poi rimanere verde.

4. Verrà visualizzata una nuova unità dal pc, trascinare il software scaricato precedentemente nell'unità. Il led di stato dovrebbe lampeggiare nuovamente, l'unità scomparirà e ne verrà visualizzata una nuova chiamata CIRCUITPY. Aprendo l'unità CIRCUITPY si troverà un file "code.py" contenente il primo programma "Hello World!" e una cartella "lib" vuota.

5. Installazione delle librerie: scaricare il pacchetto di librerie dal seguente link: https://circuitpython.org/libraries, estrarre il file zip e aprire la cartella. Copiare La cartella *'lib'* dentro all'unità CIRCUITPY.

6. Testare la scheda copiando ed incollando questo script di prova su il file *'code.py'* dentro all'unità CIRCUITPY. Servirsi di *Mu Editor* per salvare il file così modificato. Una volta salvato dovrebbe eseguire un blink del led.

```
import board
import digitalio
import time
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
while True:
    led.value = True
    time.sleep(0.1)
    led.value = False
    time.sleep(0.5)
```

Listing 2.1: Blink Led

2.2 Utilizzo con *C Arduino*

Per utilizzare la scheda con *C Arduino* bisogna seguire dei semplici passaggi (spiegato da questo link):

1. Installare Arduino Ide.

2. Aprire l'applicazione ed entrare nel menù File > Preferenze ed incollare sulla voce *Additional Board Manager URLs* il seguente link: Link Board.

3. Riavviare Arduino Ide, una volta riaperto entrare in Strumenti > Schede e cercare *'Adafruit nRF52 by Adafruit'*, una volta trovato, scaricare il pacchetto. Se non si era mai usato *CircuitPyton* allora provare ad aprire File > Examples > Basic > Blink e provare la scheda.

4. Se invece si era usato *CircuitPyton* allora bisogna reinstallare il bootloader come spiegato in questo link e successivamente riprovare il punto 3.

Capitolo 3

Funzionamento Bluetooth Low Energy (BLE)

In questo capitolo vengono esposte le principali carratteristiche e differenze tra il *Bluetooth* ed il *Bluetooth Low Energy*, verranno messi a confronto diretto e successivamente si parlerà meglio dei limiti del BLE. Per eseguire meglio il confronto si espongono le differenze in Tab.3.1 .

Caratteristica	**Bluetooth Classico (BR/EDR)**	**Bluetooth Low Energy (BLE)**
Velocità di Trasferimento Dati	Fino a 3 Mbps	Fino a 1 Mbps
Consumo Energetico	Alto	Basso
Range di Copertura	10-100 metri	10-100 metri
Tempo di Connessione	Secondi	Millisecondi
Modalità di Connessione	Continua	Intermittente
Applicazioni Tipiche	Audio, giochi, trasferimento file	IoT, sensori, dispositivi indossabili
Compatibilità	Ampia con dispositivi esistenti	Migliore nei nuovi dispositivi IoT
Versione Standard	1.0 - 3.0 + HS	4.0 e successive
Utilizzo di Profilo	A2DP, HFP per audio e chiamate	GATT per scambio dati limitati
Architettura	Connessioni punto a punto	Supporto per mesh networking

Tabella 3.1: Differenze tra Bluetooth Classico e Bluetooth Low Energy (BLE)

Come possiamo vedere il BLE è una versione più semplificata e meno energivora rispetto al più usato Bluetooth. Infatti nasce con l'esigenza di connettere dispositivi senza intaccare particolarmente la batteria, permettendo uno scambio dati intermittente e limitato ma che trova uso in molteplici dispositivi quali: dispositivi medici wearable (Pulsossimetri, glucometri, saturimetri e molti altri), sensori di temperatura, stazioni meteo, termostati, smart Tags, tastiere e mouse ma anche altri sensori wireless. Queste caratteristiche lo rendono perfetto per tutti quegli strumenti che non necessitano di uno stream continuo di dati ma di una velocità di connessione rapida e saltuaria che permette uno scambio di dati anche di pochi Kbyte. BLE offre solo 300 Kbps e i dati vengono trasmessi usando pacchetti di piccolissime dimensioni (20 byte) ma il raggio di copertura può in questo caso arrivare fino a 100 metri (tipicamente i dispositivi Bluetooth classici arrivano fino a qualche decina di metri). Tutte queste caratteristiche permettono ai dispositivi che lo usano di durare con una batteria a "bottone" mesi se non adirittura anni.

Il **BLE** è suddiviso in vari ruoli:

- **Central**: è un dispositivo che esegue la scansione dei pacchetti di Advertising[1], ed è in grado di connettersi con altre periferiche (dispositivi slave), dopo aver ricevuto i pacchetti di Advertising. Un dispositivo centrale può connettersi con più dispositivi periferiche (slave).

- **Peripheral**: è un dispositivo che esegue Advertising fino a quando qualcuno si connette ad esso.

- **Observer**: è un dispositivo che scansiona l'attività di advertising di altri dispositivi senza mai connettersi a loro. Un esempio può essere un varco elettronico che monitora gli ingressi e le uscite.

- **Broadcaster**: è un dispositivo che può solamente emettere dati in output , ma senza collegarsi ad altri o ricevere altri dati in input. Come riferimento si può pensare ad una radio che emette sempre le informazioni ma senza nessun feedback.

Nel nostro caso quello che utilizzeremo sarà una modalità "periferica" in quanto vogliamo che i dati vengano inviati solo quando ci colleghiamo e non in modo continuativo anche se non siamo collegati, in pratica dovremo ricevere il feedback di connessione per partire con la trasmissione, prima di ciò il dispositivo resta in Advertising mode. Il dispositivo "centrale" invece sarà lo smartphone che invierà i dati al cloud.

Una volta capito come strutturare la connessione, bisogna comprendere come vengono trasmessi i dati. Infatti ogni dispositivo periferica, come si vede in Fig.3.1, può essere diviso in macro aree chiamate *Service* (ad esempio accelerometro) ed ogni servizio può essere suddiviso in varie *Charateristic* (nel caso dell'accelerometro l'accelerazione per ogni asse). Nel nostro caso noi useremo solamente un *Service* composto a sua volta da una sola *Charateristic*.

[1]L'Advertising è l'azione di un dispositivo che si prepara ad essere collegato ad un altro, infatti invia continuamente i suoi dati e diventa possibile cercarlo e connettersi.Questa azione solitamente termina quando il collegamento è andato a buon fine.

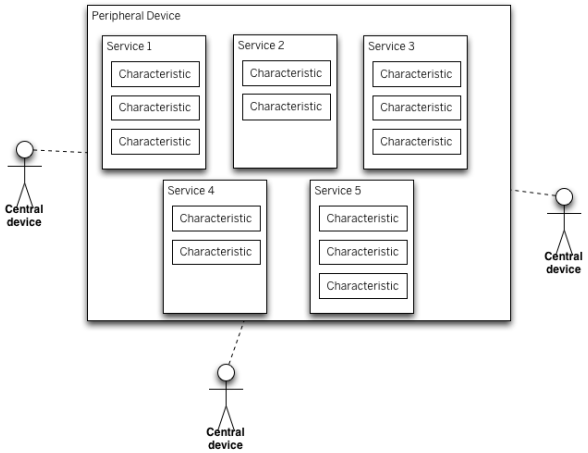


Figura 3.1: BLE charateristic e services.

Un dispositivo periferica viene visto come una lavagna composta da varie voci che contengono a loro volta i dati che vogliamo leggere, non è necessario ”vedere” tutta la tabella, ma conoscendo gli indirizzi possiamo leggere solo l’informazione che ci serve.

Per ulteriori informazioni rimando a questo video [3], ma più in generale alla playlist completa ed inoltre al sito di Arduino [4].

Capitolo 4

Prima prova di collegamento Bluetooth Low Energy

In questo capitolo cercheremo di testare con un semplice codice se il BLE funziona correttamente, non invieremo dati di nessun tipo dal microcontrollore al telefono, sarà infatti il contrario, perchè proveremo ad inviare dei pacchetti di codice colore da riprodurre sulla board grazie ad un led RGB integrato in essa. I comandi saranno trasmessi da uno smartphone grazie all'app *Bluefruit Connect*. Il codice è un tester messo a disposizione da *Adafruit IND.*, ci aiuta a capire le basi dell'Advertising e della ricezione dati.

4.1 Codice di prova in *Circuitpyton*

Questo capitolo sarà l'unico in cui il codice sarà in *CircuitPyton* poichè la board era già configurata per funzionare in questo linguaggio, dal *Cap.5* invece sarà usato *C Arduino* per la maggiore semplicità di scrittura del codice e dalla maggior disponibilità di materiale informativo.

Per caricare questo tester (Listing4.1), bisogna essere sicuri che la board stia lavorando in *CircuitPyton* (in caso contrario vedere *Sez.2.1*). Aprire *Mu Editor* e dal programma aprire il file "code.py" che si trova dentro alla memoria del microcontrollore (dovrebbe vedersi come una memoria di massa con il nome: "CIRCUITPY") se il file contiene qualcosa, va sovrascritto con il tester suscritto, una volta salvato, la board starà eseguendo in loop il codice dentro questo file.

```
# SPDX-FileCopyrightText: 2020 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# CircuitPython NeoPixel Color Picker Example

import board
import neopixel

from adafruit_bluefruit_connect.packet import Packet
from adafruit_bluefruit_connect.color_packet import ColorPacket

from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService

ble = BLERadio()
uart_service = UARTService()
advertisement = ProvideServicesAdvertisement(uart_service)

pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=0.1)

while True:
    # Advertise when not connected.
    ble.start_advertising(advertisement)
    while not ble.connected:
        pass
    ble.stop_advertising()

    while ble.connected:
        if uart_service.in_waiting:
            packet = Packet.from_stream(uart_service)
            if isinstance(packet, ColorPacket):
                print(packet.color)
                pixels.fill(packet.color)
```

Listing 4.1: Neopixel test

Il codice completo si trova in [5] per scaricarlo più agevolmente.

4.2 App *Bluefruit Connect*

Per vedere se tutto funziona correttamente abbiamo bisogno dell'app che riesce a mandare informazioni via BLE. Adafruit ha sviluppato una serie di app che possono aiutare a testare queste carattteristiche, sono molto limitate ma per scopi come questo funzionano bene. Scaricare dall'App Store l'applicazione *Bluefruit Connect*:

Figura 4.1: Bluefruit App

Una volta aperta, ci verranno mostrate tutti i vari dispositivi a cui è possibile connettersi, se abbiamo fatto giusto dovremmo leggere un dispositivo "CIRCUITPYxxxx" dove 'xxxx' è un codice alfanumerico identificativo. Una volta connessi entriamo nella scheda Modules, qui premere Controllers, e successivamente Color Picker. Ora si può selezionare il colore desiderato

e successivamente premendo SELECT si vedrà sul led della scheda lo stesso colore. Inoltre se si apre il terminale seriale sul verranno stampati volta per volta i valori RGB dei colori selezionati.

Capitolo 5

Collegamento tramite *I2C* del sensore esterno

Come detto precedentemente il sensore esterno che si è utilizzato è un sensore di temperatura digitale LM75A di *Texas Instrument*, più precisamente un sensore che converte i dati tramite un ADC di tipo Sigma/Delta ed il codice completo è *88LC LM75 AIM* come si legge dalla Fig.5.1 ed il suo datasheet si trova facilmente in rete. Tramite questo riusciamo a risalire a tutte le informazioni necessarie per il suo corretto funzionamento come vedremo dalle Sezioni successive. Il datasheet che ho scaricato si trova a questo link.

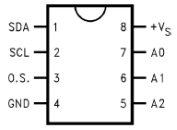
Figura 5.1: Sensore di temperatura digitale LM75A

5.1 Lettura datasheet e caratteristiche del sensore

Per usarlo correttamente dobbiamo innanzitutto conoscere il suo indirizzo per poi richiamarlo nel codice software. Nel datasheet a pg.4 troviamo la piedinatura del sensore:

5 Pin Configuration and Functions

8-Pins
SOIC (D) and VSSOP (DGK) Packages
Top View



Pin Functions

PIN		DESCRIPTION	TYPICAL CONNECTION
NO.	NAME		
1	SDA	I^2C Serial Bi-Directional Data Line, Open Drain	From Controller, tied to a pullup resistor or current source
2	SCL	I^2C Clock Input	From Controller, tied to a pullup resistor or current source
3	O.S.	Overtemperature Shutdown, Open Drain Output	Pull–up Resistor, Controller Interrupt Line
4	GND	Power Supply Ground	Ground
5	A2	User-Set I^2C Address Inputs	Ground (Low, "0") or $+V_S$ (High, "1")
6	A1		
7	A0		
8	$+V_S$	Positive Supply Voltage Input	DC Voltage from 2.7 V to 5.5 V 100-nF bypass capacitor with 10-µF bulk capacitance in the near vicinity

Figura 5.2: Piedinatura LM75A

Da come lo vediamo noi il sensore è girato di 90° per cui il pin SDA lo troveremo in alto a dx invece il pin A2 sarà in basso a sx. Per conoscere l'indirizzo *I2C* a noi interessa conoscere a quale terminale sono connessi i pin A0, A1 ed A2. Osservando attentamente il circuito stampato vediamo chiaramente che i 3 pin sono collegati a GND:

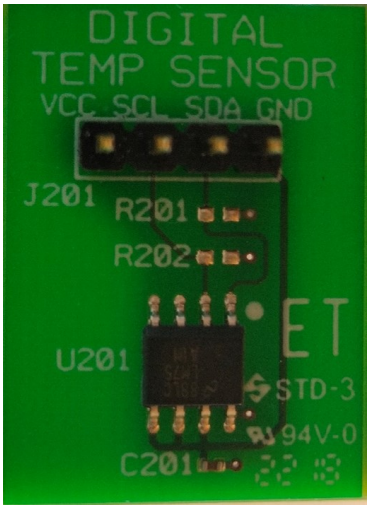


Figura 5.3: Collegamenti dei pin A0, A1 ed A2 che sono i 3 in basso a sx.

A pg.11 vi è l'indirizzo completo:

7.5 Programming

7.5.1 I²C Bus Interface

The LM75A operates as a slave on the I²C bus, so the SCL line is an input (no clock is generated by the LM75A) and the SDA line is a bi-directional serial data path. According to I²C bus specifications, the LM75A has a 7-bit slave address. The four most significant bits of the slave address are hard wired inside the LM75A and are "1001". The three least significant bits of the address are assigned to pins A2–A0, and are set by connecting these pins to ground for a low, (0); or to $+V_S$ for a high, (1).

Therefore, the complete slave address is:

1	0	0	1	A2	A1	A0
MSB						LSB

Figura 5.4: Indirizzo completo *I2C*

Che essendo i 3 LSB a 0 diventa: 1001000 in binario, poi noi lo scriveremo in esadecimale, ovvero 0x48.

Proseguendo con lo studio del datasheet dalla tabella che troviamo a pagina 4 scopriamo che è possibile leggere i dati di temperatura dal sensore in qualsiasi momento accedendo al Registro di Temperatura, e i dati ottenuti saranno il risultato dell'ultima conversione di temperatura completata. Tuttavia, se si accede al LM75A (ad esempio, tramite comunicazione I2C come vogliamo fare noi), il processo di conversione della temperatura in corso verrà interrotto e riprenderà da zero una volta terminata la comunicazione. Quindi per permettere al sensore di aggiornare il Registro di Temperatura con nuovi dati, è necessario attendere almeno un tempo di conversione tra un accesso e l'altro, che come suggerito nella nota è di almeno 300 millisecondi. Se si accede al LM75A continuamente senza rispettare questo tempo di attesa minimo, il Registro di Temperatura non verrà aggiornato con nuovi risultati di conversione. Di conseguenza, si continuerà a leggere lo stesso dato di temperatura.

Invece per leggere i dati, dobbiamo scegliere da quale registro leggere. A pg.12 infatti troviamo gli indirizzi dei registri:

7.6 Register Maps

7.6.1 Pointer Register (Selects Which Registers Will Be Read From or Written to):

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	Register Select		

P0-P1: Register Select:

P2	P1	P0	Register
0	0	0	Temperature (Read-only) (Power-up default)
0	0	1	Configuration (Read/Write)
0	1	0	T_{HYST} (Read/Write)
0	1	1	T_{OS} (Read/Write)
1	1	1	Product ID Register

Figura 5.5: Registri disponobili

Le possibilità sono:

- Temperature: serve per leggere i dati appena convertiti.

- Configuration: serve per configurare alcuni aspetti del sensore come la modalità shutdown, la modalità comparatore o la modalità interrupt(sarebbe stata utile per non usare il polling ma non è stato collegato il pin dedicato) oppure anche per cambiare la polarità dell'interrupt o del comparatore.

- T os: questo registro imposta la soglia di temperatura a cui l'uscita O.S. diventa attiva, segnalando che la temperatura ha superato un limite critico.

- T hyst: questo registro definisce il limite inferiore di temperatura a cui l'uscita O.S. viene disattivata. Questo introduce un'isteresi, evitando che l'uscita O.S. cambi stato rapidamente se la temperatura oscilla intorno alla soglia TOS.

- Product ID register: serve per avere delle informazioni sul prodotto

Per i nostri scopi basta leggere il valore nel registro Temperature che come vediamo dalla Tab.5.5 ha come indirizzo 0x00, inoltre i dati saranno in questo formato:

7.6.2 Temperature Register (Read-Only):

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MSB	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	LSB	X	X	X	X	X	X	X

D0–D6: Undefined. D7–D15: Temperature Data. One LSB = 0.5°C. Two's complement format.

Figura 5.6: Formato dei dati nel registro Temperature

Con queste informazioni abbiamo tutto il necessario per far funzionare il sensore con il protocollo *I2C*.

5.2 Collegamento fisico del sensore

Il collegamento del sensore alla *Adafruit Feather Sense* è molto semplice in realtà, infatti grazie al protocollo *I2C* che scambia i dati in modo sequenziale bastano solamente 4 connettori.

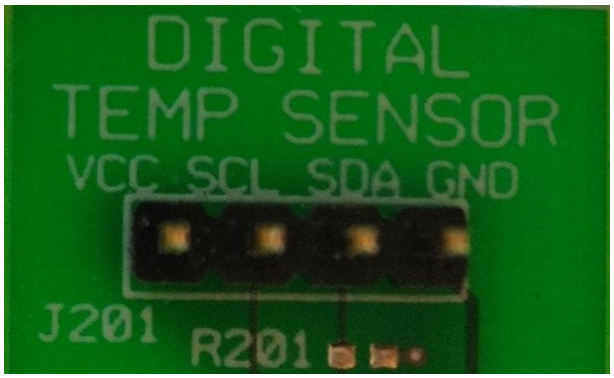


Figura 5.7: I 4 Pin del sensore

Essi corrispondono a:

- VCC: serve per alimentare il sensore, questo va collegato al pin 3V sulla board.(sulla prima pg. del datasheet infatti leggiamo che può operare ad una tensione 2.7÷5.5 V)
- SCL: serve per avere la sincronia tra il microprocessore ed il sensore. Va collegato SCL sulla board.
- SDA: serve per inviare i dati in maniera sequenziale, tutti sincronizzati con il clock del master[1]. Va collegato a SDA sulla board.
- GND: serve per dare il riferimento di massa al sensore, va collegato a GND della board.

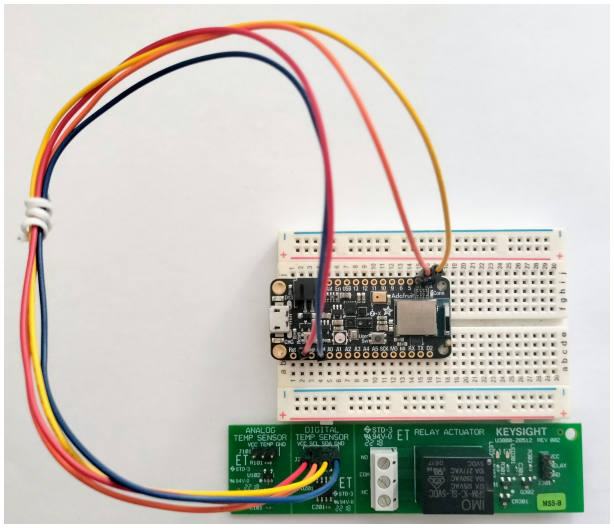


Figura 5.8: Collegamento su breadboard.

[1]Per ulteriori informazioni su questo argomento consultare [6] cap.13 sez.4.

5.3 Collegamento software e verifica dati ricevuti con *Arduino IDE*

Una volta che abbiamo trovato tutte le informazioni ed abbiamo collegato correttamente la scheda al sensore, bisogna interfacciare il tutto lato software: Arduino ha una libreria apposita per gestire il bus *I2C*, la libreria *Wire*. La documentazione della libreria si trova sul sito Arduino (link).

Il codice per provare il sensore è questo e per scaricarlo più agevolmente entrare in [5]:

```
#include <Arduino.h>
#include <Wire.h> //I2C

#define LM75A_ADDRESS 0x48 // Indirizzo I2C del sensore LM75A
#define TEMPERATURE_REGISTER 0x00 //Indirizzo del registro temperatura dove ci sono i dati

//dichiaro le variabili dei led
int LEDR=13;
int LEDB=4;

void setup()
{
    Wire.begin(); //attivo I2C
    Serial.begin(115200);

    /////////COMMENTARE SE NON CONNESSO A PC questo comando aspetta che sia aperto il monitor seriale
    //se non è connesso al pc non si connetterà mai
    while (!Serial);
    ////////////

    Serial.println("");
    Serial.println("------------------------------------------------------------");
    Serial.println("Test I2C keysight\n");


    pinMode(LEDR, OUTPUT); // onboard led red
    digitalWrite(LEDR, LOW); // led red off

    Wire.beginTransmission(LM75A_ADDRESS); //prepara la comunicazione con il sensore mandando l'indirizzo del sensore seguito
    //da uno 0(write)
    Wire.write(TEMPERATURE_REGISTER); //scrive nel sensore l'indirizzo del registro da leggere
    Wire.endTransmission();//chiude questa comunicazione
}

void loop()
{
    LEDR=1;
    int16_t temperatureRaw = readTemperature(); //non è una funzione di libreria , ma è definita sotto
    float temperatureC = temperatureRaw / 256.0; // Conversione dei dati grezzi in gradi Celsius

    Serial.print("Temperature: ");
    Serial.print(temperatureC);
    Serial.println(" C");
    delay(350);//la max velocità è di 300ms ma mi tengo un pò di margine
}

int16_t readTemperature()
{
    Wire.requestFrom(LM75A_ADDRESS, 2);//prepara la comunicazione con il sensore mandando l'indirizzo del sensore seguito da
    //un 1(read) e si aspetta 2 byte di trasmissione, dopo i quali manda uno stop
    uint8_t msb = Wire.read();
    uint8_t lsb = Wire.read();

    int16_t temperature = (msb << 8) | lsb; //messa a punto dei dati come poi vengono letti
    return temperature;
}
```

Listing 5.1: Test sensore esterno

Con questo codice leggiamo i valori di temperatura contenuti nel registro corretto, successivamente li dovremo veder stampati sul monitor seriale di Arduino IDE. Una volta lanciato il codice, se siamo collegati alla scheda con il cavo USB , bisogna aprire il serial monitor che si trova in alto a dx (assomiglia ad una lente di ingrandimento) oppure premendo i tasti Ctrl+Maiusc+M. Ora vengono stampati i valori corretti.

Alcune spiegazioni sul codice:

- A riga 16 e 17 di il ***while(!Serial)*** serve per assicurarsi che il codice non cominci ad inviare dati se prima non si è aperto sull'IDE il monitor seriale, se si alimenta la board con una batteria o con un sistema noin dotato di scambio dati, bisogna commentare riga 18 così il codice salta questa verifica. Non ha utilità in questo codice che deve solo inviare i dati al pc ma come vedremo in List.6.1 del Cap.6 serve rimuoverlo altrimenti non si inizia con l'Advertising del BLE.

- Da riga 47 fino a 56 viene definita la funzione di lettura dei dati nel sensore. Nello specifico vengono richiesti 2 byte dal sensore e questi vengono salvati in 2 variabili senza segno a 8 bit (una per byte). A riga 54 invece sistemiamo i bit come specificato in Fig.5.6 ovvero con uno shift a sx di 8 posizioni spostiamo i MSB a sx di una nuova variabile a 16 bit a cui affianchiamo i LSB, inoltre la variabile è con segno dato che i valori sono in complemento a 2.

Capitolo 6

Invio dati ricevuti dal sensore a *Google Fogli*

6.1 Setup attrezzatura e problemi riscontrati

L'attrezzatura che serve per il corretto funzionamento del progetto è tutta quella indicata nel Cap.2, montata come specificato in 5.2, nei prossimi paragrafi verrà indicato il codice da caricare sulla board ed il procedimento di settaggio dell'App Android e google fogli.

Inizialmente il progetto richiedeva di trasferire i dati via BLE ad un PC e di salvarli in un qualsiasi tipo di file, questo però dopo alcune ricerche si è svelato molto complesso in quanto come riportato nel sito ufficiale di Adafruit(dalla lista mancano Win10 o superiori), Windows non è in grado di usare il BLE correttamente, anche se il dispositivo possiede il modulo Bluetooth. Molte altre fonti online riscontrano lo stesso problema anche con altri dispositivi.(link Adafruit)

Per ovviare a questo problema viene consigliato di sviluppare un'app apposita per le proprie esigenze e di usare un dongle USB esterno (link uso dongle) oppure di scaricare il codice dell'app *Bluefruit Connect* da Github e di compilarlo sul proprio Pc per avere l'app desktop. (link Github, leggere su questa pagina la spiegazione dei problemi di compatibilità di Windows)

Tutte queste soluzioni richiedono hardware esterno di cui non disponevo oppure di sviluppare app, cosa che esula dalle mie competenze.

L'unica alternativa è quella di usare uno smartphone con l'app ufficiale (Bluefruit Connect) che supporta nativamente il BLE e di salvare in locale i dati. Anche questa via però non si è dimostrata percorribile in quanto non esistono modi di salvare i dati ricevuti dall'app sul dispositivo.(link FAQ salvataggio dati su Android) Come su PC, infatti richiede un'app apposita. Fortunatamente in rete esiste un progetto molto simile effettuato però con una Board ***Seeed Studio Seeed XIAO BLE nRF52840 Sense*** che usa lo stesso microcontrollore, ma ha hardware

diverso. Questo progetto sviluppato da *MJRoBot (Marcelo Rovai)* usa l'accelerometro interno alla board ed invia i dati dei 3 assi allo smartphone ed infine per evitare il salvataggio in locale viene sfruttata la connessione ad internet di quest'ultimo per caricare i dati su un foglio di calcolo online.(link progetto; link github progetto per i codici). Questo è l'unico modo funzionante trovato.

N.B. Testato e sviluppato con uno smartphone Android 14, non provato su dispositivi con altri sistemi operativi.

6.2 Codice *C Arduino*

Per il funzionamento della scheda è stato necessario modificare completamente il codice fornito da [7], in quanto non usava un sensore I2C e al posto di inviare 3 dati diversi per i 3 assi, necessitiamo solo di inviarne 1, quello di temperatura. Il codice completo si trova in [5] per scaricarlo più agevolmente.

```
/* nrf52840 - Bluetooth Data Logger inspired by MJRoBot (Marcelo Rovai)
 * https://www.hackster.io/mjrobot/sensor-datalogger-50e44d
 * Generic external data via I2C (in this case temperature sensor LM75A by Texas Instruments)
 *
 * Guido Facco
 * 9/7/2024
*/

#include <bluefruit.h> // adafruit ble
#include "Wire.h" //Libreria arduino per gestione I2C

#define LM75A_ADDRESS 0x48 // Indirizzo I2C del sensore LM75A
#define TEMPERATURE_REGISTER 0x00 //Indirizzo del registro temperatura dove ci sono i dati

int minPeriod=350;  /*definisco il periodo minimo in millisecondi di lettura dei dati. Dipende dal sensore che si è collegato.
            Nel mio caso LM75A ha un periodo minimo di 300ms. Lo metto all'inizio in modo da permettere facili modifiche */

/* Online GUID / UUID Generator:
https://www.guidgenerator.com/online-guid-generator.aspx
64cf715d-f89e-4ec0-b5c5-d10ad9b53bf2
*/

// UUid for Service (Universal Unique IDentifier)
const char* UUID_serv = "64cf715d-f89e-4ec0-b5c5-d10ad9b53bf2";

// UUids for temperature data
const char* UUID_temp   = "64cf715e-f89e-4ec0-b5c5-d10ad9b53bf2";

// BLE identificazione
BLEDis bledis;
// BLE Service instanziazione
BLEService tempServ = BLEService(UUID_serv);

// BLE Characteristics instanziazione
BLECharacteristic  tempSens = BLECharacteristic(UUID_temp);

//dichiaro le variabili dei led
int LEDR=13;
int LEDB=4;
int connect = 0;  //indica lo stato di connessione 0-non_conn  1-conn

void setup()
{
    Wire.begin(); //attivo I2C
    Serial.begin(115200);
    setReadRegister();  //funzione definita e scritta a fine pagina , serve per indicare quale registro voglio leggere
```

```
47
48      ////////RIMUOVERE SE NON CONNESSO A PC
49      while (!Serial);
50      ////////////
51
52      Serial.println("");
53      Serial.println("-----------------------------------------------------------------");
54      Serial.println("BLE - Data Logger(Setup)");
55
56      bool err=false;     //flag di errore
57
58      pinMode(LEDR, OUTPUT);      // onboard led red set out
59      pinMode(LEDB, OUTPUT);      // onboard led blue (connection status led) set out
60      digitalWrite(LEDR, LOW);     // led red off
61      digitalWrite(LEDB, LOW);     // led blue off
62
63      // init BLE
64      if (!Bluefruit.begin())
65      {
66          Serial.println("BLE: failed");
67          err=true;
68      }
69      Bluefruit.setName("Datalogger nRF52840");
70      Bluefruit.setTxPower(4);    // Check bluefruit.h for supported values
71      Bluefruit.Periph.setConnectCallback(connect_callback);
72      Serial.println("BLE: ok");
73
74      // error: flash led forever
75      if (err)
76      {
77          Serial.println("Init error. System halted (non è stato possibile inizializzare ble)");
78          while(1)
79          {
80              digitalWrite(LEDR, LOW);  //led off
81              delay(500);
82              digitalWrite(LEDR, HIGH); // led on
83              delay(500);
84          }
85      }
86
87      // BLE service
88      // correct sequence:
89      // set BLE name > advertised service > add characteristics > add service > set initial values > advertise
90
91      // Configurazione e Start Device Information Service
92      bledis.setManufacturer("Unipd(GF)");
93      bledis.setModel("Bluefruit Feather Sense nRF52840");
94      bledis.begin();
95
96      // attivare advertising Service
97      tempServ.begin();
98
99      // attivare characteristics to the Service
100     tempSens.setProperties(CHR_PROPS_NOTIFY);    //non può ricevere dati dall'esterno
101     tempSens.setPermission(SECMODE_OPEN, SECMODE_NO_ACCESS);  //sola lettura da un dispositivo esterno
102     tempSens.setFixedLen(4);    //dati di 4 byte (long)
103     tempSens.begin();
104
105     // start advertising
106     // Set up and start advertising
107     startAdv();
108     Serial.println("Advertising started");
109     Serial.println("Bluetooth device active, waiting for connections...");
110 }
111
112
113
114 int check=0;  //variabile ausiliaria usata per mostrare solo una volta(==0) l'avviso di riconnessione sul serial monitor
115 int i=0;
116 float t=0;
117
118
119 void loop()
120 {
121   //questo if mostra 1 sola volta il messaggio di riconnessione sul serial monitor quando la connessione è persa
```

```
  if(!Bluefruit.connected() && connect>=1 && check==0)
  {
    // central disconnected:
    Serial.println("Disconnected from the device...  Reconnecting...");
    check=1;
    connect=0;
  }

  long preMillis = millis();  //tempo di riferimento va dichiarato fuori dal while

  //questo invece esegue il necessario per leggere il dato dal registro del sensore e inviarlo tramite BLE
  while (Bluefruit.connected())
  {
    if(connect==0)
    {
      connect=1;
      delay(1000);//per evitare conflitti su connessione ble
    }
    check=0;

    //eseguo la lettura ogni minPeriod ma con una funzione non bloccante

    if(millis()>=preMillis+minPeriod) //se il tempo letto ora è maggiore a quello precedente+il periodo indicato all'inizio,
                                //entro nel corpo dell'if
    {
        preMillis=millis(); //aggiorno il valore del tempo salvato
        float t= readTemperature(); //non è una funzione di libreria , ma è definita e scritta sotto
        Serial.print(i);
        Serial.print("  Temperature: ");
        Serial.print(t);
        Serial.println(" °C");
        tempSens.notify32(t);
        i++;
    }

  } //resta nel while quando è connesso
}

////////////////////////////////////////////////////////////////////////////////////////////////////
// Legge la temperatura e trasforma i bit nel valore della temperatura in °C
float readTemperature()
{
  Wire.requestFrom(LM75A_ADDRESS, 2);//prepara la comunicazione con il sensore mandando l'indirizzo del sensore seguito da
                                  //un 1(read) e si aspetta 2 byte di trasmissione, dopo i quali manda uno stop
  uint8_t msb = Wire.read();
  uint8_t lsb = Wire.read();

  int16_t temperature = (msb << 8) | lsb;
  float temperatureC = temperature / 256.0; // Conversione dei dati grezzi in gradi Celsius
  return temperatureC;
}

////////////////////////////////////////////////////////////////////////////////////////////////////
// Setta il registro in cui voglio leggere i dati della temperatura
void setReadRegister()
{
  Wire.beginTransmission(LM75A_ADDRESS); //prepara la comunicazione con il sensore
  Wire.write(TEMPERATURE_REGISTER); //scrive nel sensore l'indirizzo del registro da leggere
  Wire.endTransmission();//chiude questa comunicazione
}

////////////////////////////////////////////////////////////////////////////////////////////////////
// Setta e fa partire l'advertising del BLE
void startAdv(void)
{
  // Advertising packet
  Bluefruit.Advertising.addFlags(BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE);
  Bluefruit.Advertising.addTxPower();

  // Secondary Scan Response packet (optional)
  // Since there is no room for 'Name' in Advertising packet
  Bluefruit.ScanResponse.addName();

  /* Start Advertising
  * - Enable auto advertising if disconnected
```

```
  * - Interval:  fast mode = 20 ms, slow mode = 152.5 ms
  * - Timeout for fast mode is 30 seconds
  * - Start(timeout) with timeout = 0 will advertise forever (until connected)
  *
  * For recommended advertising interval
  * https://developer.apple.com/library/content/qa/qa1931/_index.html
  */
  Bluefruit.Advertising.restartOnDisconnect(true);
  Bluefruit.Advertising.setInterval(32, 244);    // in unit of 0.625 ms
  Bluefruit.Advertising.setFastTimeout(30);      // number of seconds in fast mode
  Bluefruit.Advertising.start(0);                // 0 = Don't stop advertising after n seconds
}

////////////////////////////////////////////////////////////////////////////////////////////////
// Lettura e scrittura nel serial plotter se connesso del nome del dispositivo a cui ci si è connessi
void connect_callback(uint16_t conn_handle)
{
  // Get the reference to current connection
  BLEConnection* connection = Bluefruit.Connection(conn_handle);

  char central_name[32] = { 0 };
  connection->getPeerName(central_name, sizeof(central_name));

  Serial.print("Connected to ");
  Serial.println(central_name);
}
```

Listing 6.1: Codice completo per la trasmissione dati via BLE

6.3 Setup Google Fogli

Per caricare i dati ora dobbiamo preparare il foglio Google a riceverli; sul sito([8]) viene spiegato bene come procedere, tuttavia riporto i passaggi fondamentali per il corretto funzionamento, dato che ci sono delle piccole precisazioni:

1. Entrare tramite google (consigliato un PC, no smartphone) a https://docs.google.com/-spreadsheets.

2. Aprire un file bianco e rinominarlo a piacere, poi entrare in Strumenti e poi in ”Crea un nuovo modulo”

3. Si aprirà una nuova scheda per impostare il modulo, sulla domanda bisogna scrivere Temperatura, come risposta invece inserire ”risposta breve”.

4. Premere sui 3 puntini di opzione in alto a dx vicino alla foto dell’account Google e premere su ”genera link precompilato”.

5. inserire una risposta, ad esempio 30 (non si deve inserire il simbolo di gradi o scrivere altro), poi premere ”genera link”. Il link così ottenuto va salvato momentaneamente su un blocco note, infatti ci servirà nella Sez.6.4.

6. Per controllare se funziona possiamo modificare il link del punto 5 sostituendo ”viewform” con ”formResponse?&submit=Submit”, una volta modificato il link basta incollarlo su una scheda google e premere invio.

7. Ora basta tornare sul foglio di calcolo se andiamo su ”Risposte del modulo 1” in basso a sx troveremo 30 (la risposta che avevamo scritto) sul foglio di lavoro.

6.4 Sviluppo App per *Android*

Per il lato smartphone le cose sono un pò diverse e più complesse in quanto bisogna ”creare” un’app. Sempre dal link [8] si trova un modo abbastanza semplice per crearla: utilizzare MIT App Inventor, dopo opportune modifiche (il progetto originale era fatto per lavorare su un android di qualche versione precedente e le autorizzazioni sull’uso del bluetooth sono cambiate nel corso degli anni, la soluzione per i problemi di autorizzazione è stata trovata su questo link soluzione) si trova sulla mia pagina [5] il progetto corretto e rivisto chiamato ”Datalogger_ext_copy.aia”.

Per quanto riguarda i sistemi operativi diversi da Android, non possedendone non ho potuto eseguire nessuna prova, comunque lascio un link per informazione su come si può usare MIT App Inventor per creare app per IOS.(https://iosbuildservertest.appinventor.mit.edu/reference/other/build-ios-apps.html).

Ecco i passaggi per far funzionare correttamente l’app:

1. Andare su [5] e scaricare il file sopracitato.

2. Collegarsi alla pagina https://appinventor.mit.edu/.

3. Premere in alto a sx su ”Create App” e successivamente registrarsi/effettuare login.

4. Premere su Project in alto a sx e poi su ”*Import project (.aia) from my computer*” e selezionare il file precedentemente scaricato.

5. Dopo qualche secondo dovrebbe aprirsi in automatico una schermata tipo questa:

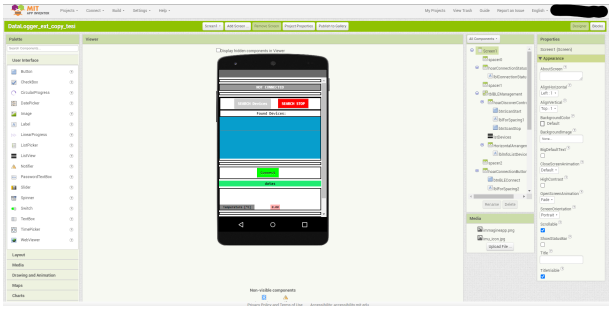


Figura 6.1: MIT App Inventor.

6. In alto a dx premere sul comando Blocks.

7. Bisogna modificare il link precedentemente salvato: da prima si è già fatta la modifica principale, ora basta rimuovere il numero di prova che avevamo inserito (30) che è in coda al link.(ora il link dovrebbe terminare con ”=”)

8. Ora bisogna copiare il link modificato e copiarlo sul blocco ”join” magenta collegato al blocco viola ”URL” (è il penultimo della seconda colonna di blocchi). Se è già presente un link sovrascrivetelo.

9. Salvare il file silla voce ”Project” e poi andare su ”Build” - ”Android App” in alto a sx e seguite le istruzioni.

10. Installate l'app sul dispositivo.

Ora dovrebbe essere tutto pronto per funzionare correttamente.

6.5 Funzionamento del sistema completo

Una volta caricato il codice sulla board (ricordare di commentare il *while* a riga 49 se non si usa collegata al pc) bisogna darle alimentazione, essa può essere data sia da pc subito dopo il caricamento del codice oppure tramite usb , quindi con un powerbank o un trasformatore da muro di quelli da smartphone, bisogna però prestare attenzione perchè la scheda utilizza una tensione interna di 3.3V, quindi non si può collegare con 2 jumpers una tensione di 5V direttamente ai terminali della scheda come si usa fare con le board di Arduino. L'ultimo metodo è quello di usare una batteria LI-PO da 3.7V nominali, infatti la scheda ha integrato un circuito (con tanto di led di stato) apposito per la ricarica e scarica di questo tipo di batterie. Una volta collegata l'alimentazione, il led blu sulla board, dovrebbe lampeggiare perchè non vi è nessun dispositivo connesso tramite bluethoth. Ora dall'app sullo smartphone bisogna premere *SEARCH devices* in alto a sx.

Figura 6.2: App Datalogger Homepage.

Compare la lista di tutti i dispositivi disponibili, selezionare l'unico con il nome *Datalogger nRF52480*.

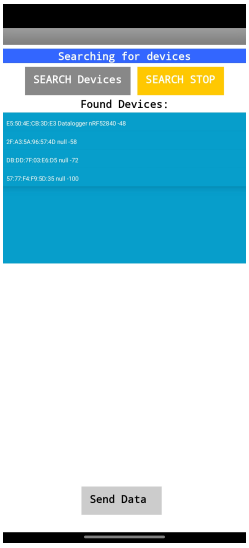


Figura 6.3: App Datalogger searching devices.

Premendo *connect* e si dovrebbe vedere questa schermata.

Figura 6.4: App Datalogger.

Una volta premuto *Send Data* se il telefono è connesso ad internet, i dati verranno caricati sul foglio di calcolo online di Google.

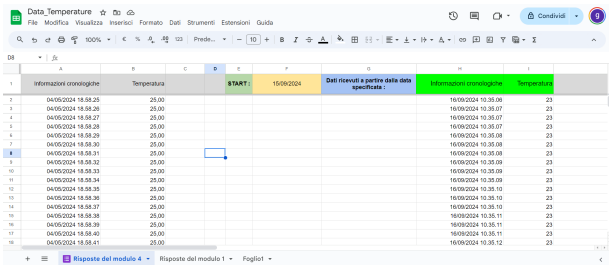


Figura 6.5: Google fogli

In particolare in questo foglio è stato aggiunto un filtro in modo da impostare la data e l'ora dell'inizio dei dati d'interesse, infatti Google fogli non può elliminare i dati vecchi che non ci interessano più ma ne aggiunge di nuovi in coda, filtrando i dati riusciamo ad evitare di scorrere la pagina fino al punto in cui i dati stanno venendo inseriti. Per replicare questo filtro è sufficente copiare questa folmula all'interno della casella H1 ed inserire una data valida nella casella F1.

```
=FILTER((A:B) ; A:A >= F1)
```

Figura 6.6: Formula da inserire in H1 per avere il filtro

Il sistema completo dovrebbe apparire così:

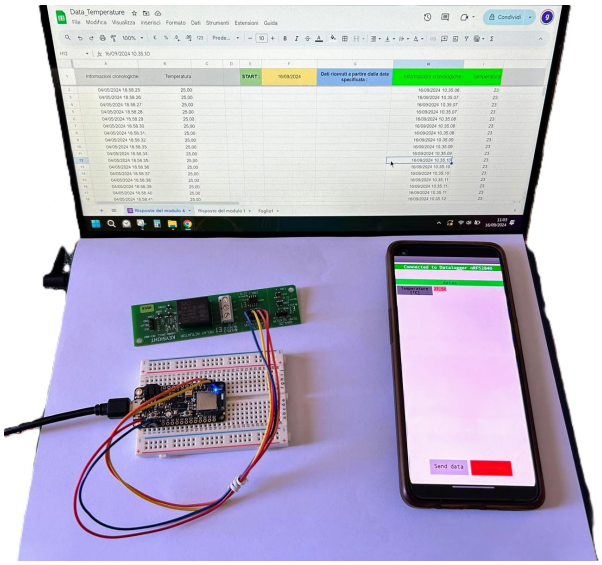


Figura 6.7: Sistema completo

Ovviamente il foglio di calcolo può essere esaminato anche da smartphone mentre l'app di caricamento dati lavora in background.

Capitolo 7

Conclusioni e Possibili migliorie

Il progetto di questo sistema ha richiesto varie soluzioni, alcune molto semplici da eseguire altre hanno richiesto molto più lavoro sia nel progetto che nell'usabilità del sistema stesso.

Usare il sensore esterno con il protocollo *I2C* richiedeva lo studio del datasheet e capire come scrivere il codice ma è stato relativamente semplice ed utile sia per comprendere a pieno la comodità di questo standard che per estendere la flessibilità di raccolta dati considerando che esiste un sensore *I2C* quasi per ogni tipologia, quindi basta modificare l'indirizzo e i dati verranno ricevuti dalla scheda allo stesso modo, inoltre sarà possibile collegarne molti sulla stessa linea dati così da risparmiare cablaggi o piste nell'eventualità di sistemi finiti.

Un passaggio invece complesso e laborioso sia da progettare che da usare è l'utilizzo dello smartphone in quanto richiede uno strumento esterno e un'app dedicata a caricare i dati sul cloud. Molto più facile e diretto a questo punto progettare un'app desktop per pc così da salvare i dati in un file locale e saltare il passaggio in rete. Inoltre sviluppandola si può introdurre un sistema di controllo per assicurarsi che i dati ricevuti siano corretti e non se ne sia perso nessuno, cosa molto complessa da fare sull'app di questa versione. Si potrebbero anche 'impachettare' i dati in gruppi di più elementi e spedirli in una volta sola in modo da usare una frequenza di invio minore; cosa fattibile solo se in ricezione si riescono a dividere e salvare in modo corretto i valori. Infine visto che il dispositivo dovrebbe essere indossabile e quindi di dimensioni il più contenute possibili, consiglio di provare con una board *Seeed Studio XIAO nRF52840 Sense* che usa lo stesso processore ma è grande circa la metà della versione *Adafruit*, usa sempre lo stesso linguaggio quindi il codice è utilizzabile e supporta l'*I2C* permettendo di replicare il progetto.

In conclusione mi ritengo soddisfatto del progetto, anche se molto migliorabile, perchè sono riuscito ad utilizzare delle competenze acquisite in questi anni di studio e metterle in pratica riuscendo ad ottenere dei risultati soddisfacenti.

Bibliografia

[1] M. Massaro, «Analisi e programmazione del sistema multi-sensore Adafruit Feather Sense,» Tesi Triennale, Università degli Studi di Padova, Padova, Italy, 2024.

[2] Adafruit Industries. «Adafruit Feather nRF52840 Sense.» (2024), indirizzo: https://www.adafruit.com/product/4516.

[3] S. E. Engineering. «nRF5 SDK - Tutorial for Beginners Pt 40 - Bluetooth Low Energy an Introduction to basics in BLE.» (2021), indirizzo: https://www.youtube.com/watch?v=AQu80Fdn3T8&t=1301s.

[4] Arduino. «Arduino BLE.» (), indirizzo: https://www.arduino.cc/reference/en/libraries/arduinoble/.

[5] G. Facco. «URL github codice.» (), indirizzo: https://github.com/io-11/Adafruit-FeatherSense-nRF52480-Datalogger?tab=readme-ov-file#adafruit-feathersense-nrf52480-datalogger.

[6] S. Buso, *Introduzione alle applicazioni industriali di MICROCONTROLLORI E DSP*. Società editrice ESCULAPIO, 2020.

[7] Marcelo Rovai. «Sensor DataLogger.» (2022), indirizzo: https://github.com/Mjrovai/Sensor_DataLogger.

[8] Marcelo Rovai. «Sensor DataLogger.» (2022), indirizzo: https://www.hackster.io/mjrobot/sensor-datalogger-50e44d#code.