Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Studi in Ingegneria Elettronica

Analisi e programmazione del sistema multi-sensore Adafruit Feather Sense

Adafruit Feather Sense multi-sensor system analysis and programming

Relatore:
Prof.ssa Giada Giorgi

Laureando:
Matteo Massaro

Matricola:
2000109

ANNO ACCADEMICO 2023–2024

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

L'avvento dell'Internet of Things (IoT) ha accelerato lo sviluppo di dispositivi compatti e intelligenti, spingendo all'integrazione di sensori avanzati in molte schede di sviluppo. In questo contesto, la scheda Adafruit Feather Sense nRF52840 emerge come un esempio significativo di soluzione versatile e pronta all'uso per progetti IoT. La ricerca si concentra sull'approfondimento della scheda Adafruit Feather Sense nRF52840, esaminandone i suoi componenti hardware e gli ambienti di programmazione compatibili, Arduino e CircuitPython. Vengono approfonditi i sensori integrati presentando, per ognuno di essi, i codici in Circuitpython utilizzati per effettuarne il test. In aggiunta, viene esaminata l'interfaccia I2C dei sensori, concentrandosi sulla lettura e scrittura dei registri, anch'essa accompagnata da codici di test in CircuitPython. La conclusione offre un sunto delle analisi svolte e una riflessione su potenziali sviluppi futuri.

Introduzione

In questa tesi si discute l'analisi componentistica e la programmazione della scheda Adafruit Feather Sense.

Adafruit Feather Sense è una scheda di sviluppo contenente il processore Cortex-M4F.

Le schede di sviluppo sono circuiti stampati su cui è installato un microcontrollore o un microprocessore insieme a pochi altri componenti hardware, quali memorie e un interfaccia input/output.

Il microcontrollore o microprocessore può essere facilmente programmato utilizzando un PC, attraverso una porta seriale, così da permettere all'utente di interagire con i diversi sensori e attuatori posti sulla scheda stessa.

Inizialmente tali schede di sviluppo furono introdotte sul mercato solo per utilizzi didattici, mentre ultimamente sono sempre più utilizzate nell'ambito dell'IoT (Internet of Things).

Nel primo capitolo di questa tesi è presentata la scheda tramite una sommaria analisi delle componenti hardware, focalizzando l'attenzione ai molteplici sensori installati nella scheda.

Nel secondo capitolo si discute, invece, la scelta dell'ambiente di programmazione andando ad analizzare pregi e difetti di ciascuna proposta.

Successivamente, nel terzo capitolo, si discutono alcuni programmi scritti con lo scopo di effettuare il test di tutti i sensori a disposizione.

Nel quarto capitolo si discute l'interconnessione, tramite protocollo I2C, tra i sensori e il microprocessore, andando ad analizzare un programma che consente la scrittura e la lettura dei registri di ogni sensore.

Infine, nel capitolo conclusivo, vengono illustrate le possibili analisi e gli ulteriori sviluppi futuri da effettuare sulla scheda Adafruit Feather Sense.

Capitolo 1

Adafruit Feather Sense nRF52840

1.1 Nascita della scheda

La nascita della scheda Adafruit Feather Sense nRF52840 è stata una risposta alla crescente domanda di dispositivi compatti e potenti per l'IoT e, il suo sviluppo è stato influenzato da diversi fattori, tra cui l'avvento di nuove tecnologie wireless come il Bluetooth Low Energy (BLE).

Adafruit ha integrato il microcontrollore nRF52840 in una scheda compatta seguendo il design modulare della linea *Feather*, questo ha permesso agli sviluppatori di combinare facilmente la scheda insieme ad altri moduli e accessori Feather esistenti. La modularità di queste schede ha così permesso di ridurre e semplificare il tempo di sviluppo e il processo di prototipazione.

La scheda Feather Sense, raffigurata in Fig. 1.1, è stata lanciata sul mercato nel 2020, suscitando un forte interesse per il fatto che, oltre alle funzionalità standard del microcontrollore nRF52840, offre una serie di sensori integrati, tra cui accelerometro, giroscopio, magnetometro, sensore di temperatura e umidità, sensore di pressione, un rilevatore di gesti e luce ambientale e un microfono.

Questi sensori permettono agli sviluppatori di creare in modo rapido dispositivi reattivi e intelligenti per l'IoT, adatti ad una vasta gamma di applicazioni dalla domotica all'industra.

1.2 Struttura e componenti

La struttura su cui sono saldate tutte le componenti è una classica scheda della linea *Feather* di Adafruit.

Le dimensioni sono di 2,3x5,1 cm, con fori di 0,25cm di diametro posti su ogni angolo. Sul lato inferiore della scheda si trova una striscia di 16 pin, mentre sul lato superiore è presente, una di 12 pin. In Fig. 1.2 è rappresentata la struttura appena descritta.

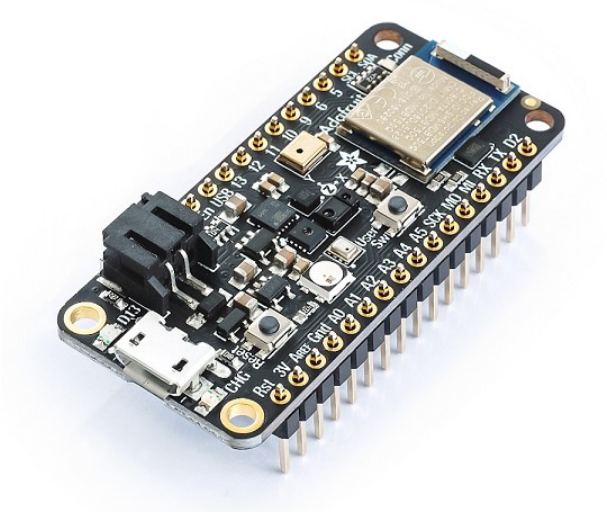

Figura 1.1: Adafruit Feather Sense nRF52840.

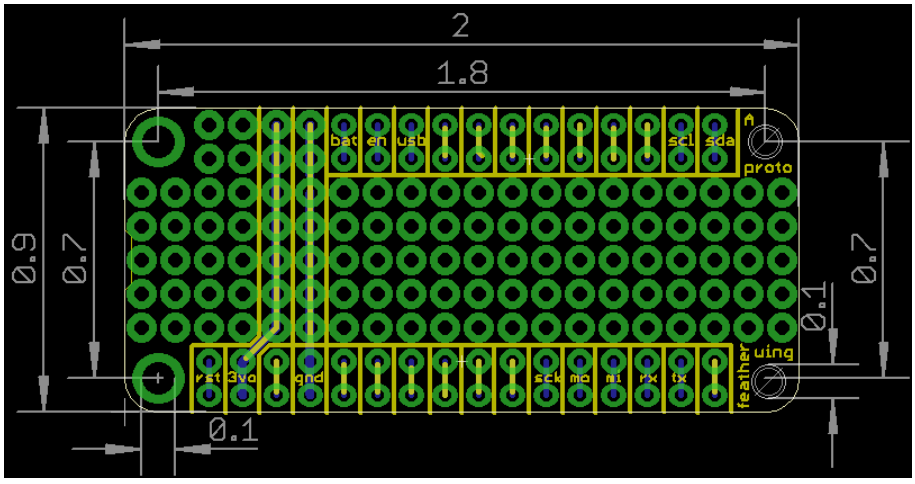

Figura 1.2: Classica struttura di una scheda Feather. Le misure sono raffigurate in pollici.

1.2.1 Piedinatura

Il primo pin a sinistra, della striscia inferiore, è il pin Rst. Esso rappresenta la linea principale per il reset della scheda, il quale avviene nel caso in cui questo pin venga collegato a massa. Nella scheda che stiamo analizzando è anche presente un pulsante di reset che permette di effettuare l'operazione in modo più rapido.

Procedendo in senso antiorario si incontra il pin 3V, il quale insieme ai pin GND, Bat, En e USB rappresenta i pin adibiti alla gestione dell'alimentazione. In particolar modo, il pin 3V è collegato all'uscita dal regolatore da 3.3V integrato e permette di fornire alimentazione a 3.3V a sensori o ad altri moduli esterni, mentre il pin GND rappresenta la massa. Il pin USB offre in uscita la tensione di alimentazione della porta USB, nominalmente 4.5-5.2V, solitamente viene usato per alimentare componenti più dispendiosi come servomotori o molti LED, mentre il pin Bat fornisce in uscita la tensione di alimentazione della batteria LIPO opzionale, nominalmente 3.5-4.2V. Infine il pin En, questo pin può essere impostato su GND per disabilitare l'uscita del pin 3V, per impostazione predefinita è impostato a livello logico alto tramite un resistore di pullup.

Procedendo ulteriormente verso destra si evidenzia il pin Aref, il quale insieme ai pin A0,…,A5 costituisce i pin analogici della scheda. I pin A0,…,A5 sono sei ingressi analogici che permettono di acquisire dati a 8, 10 o 12 bit (o 14 con sovracampionamento), a velocita fino ai 200kHz. Il pin Aref può, invece, essere utilizato come riferimento analogico per il comparatore interno; il riferimento analogico interno di defoult è di 0.6V.

Si trovano poi i pin SCK, MO e MI, i quali permettono di utilizzare il bus SPI.

Riguardanti il bus ci sono ulteriormente i pin SDA e SCL, i quali, invece, permettono di utilizzare il protocollo I2C e, infine, i pin RX e TX, che permettono l'utilizzo del protocollo UART.

Come ultimi, ci sono sette pin di tipo *GPIO* (General purpose input/output) che corrispondono ai pin 5, 6, 9, 10, 11, 12, 13.

1.2.2 Microcontrollore

Il microcontrollore installato nell'Adafruit Feather Sense è il Nordic nrF52840. Un SoC (System-on-chip) multiprotocollo che rappresenta una soluzione a chip singolo altamente flessibile e a bassissimo consumo, ideale per applicazioni wireless a corto raggio. L'nrF52840 consente simultaneamente la connettività wireless Bluetooth 5.2 e Thread. L'nrF52840 è costruito attorno ad un processore Arm Cortex M4F a 32 bit e 64 Mhz con 1 MB di memoria flash e 256 kB di RAM sul chip.

1.2.3 Sensori

La caratteristica principale, per la quale l'Adafruit Feather Sense ha suscitato maggiore interesse, è la presenza di sei sensori integrati nella scheda. In particolar modo, sono presenti accelerometro, giroscopio, magnetometro, sensore di temperatura e umidità, sensore di pressione, un rilevatore di gesti, luce ambientale e prossimità e un microfono.

L'accelerometro e il giroscopio sono contenuti in un unico modulo, il sensore LSM6DS33, formato da un accelerometro a 3 assi che restituisce una misura di accelerazione lineare lungo i tre assi cartesiani del sensore e da un giroscopio, anch'esso a 3 assi, in grado di misurare rotazione e torsione della scheda.

Il magnetometro è presente con il modulo LIS3MDL, un sensore a 3 assi che permette di rilevare la direzione da cui proviene la forza magnetica più forte. Generalmente è utilizzato per rilevare il nord magnetico oppure per misurare campi magnetici.

Il sensore SHT30 permette di misurare l'umidità relativa dell'ambiente con una precisione del ±2% e, inoltre, la temperatura con una precisione di ±0.5°C.

Il sensore BMP280 è un modulo integrato che permette di misurare la pressione barometrica e la temperatura rispettivamente con una precisione di ±1 hPa e ±1°C.

È possibile, inoltre, rilevare gesti, prossimità e luce ambientale grazie all'utilizzo del sensore APDS9960. Il sensore restituisce la quantità di rosso, blu, verde e luce chiara, la distanza di un oggetto dalla parte anteriore del sensore e rileva semplici gesti come per esempio lo scorrere del dito da sinistra a destra, dall'alto al basso, ecc.

Infine si può usufruire del sensore MP34DT01-M, un microfono PDM.

1.2.4 Led di stato

Nella scheda sono presenti quattro led di stato, NeoPixel, D13, Conn e CHG.

Il NeoPixel è un led RGB, situato circa al centro della scheda, utilizzato come led di stato per il bootloader e per CircuitPython, controllabile anche tramite il codice.

Il led rosso D13 è un led di stato del bootloader, mentre il led blu Conn è utilizzato come led di stato per la connettività con Arduino.

Infine, il led CHG, posto sotto il connettore USB, è il led dello stato di carica. Quando viene connessa una batteria il led è giallo fisso, altrimenti il led lampeggia.

Capitolo 2

Ambienti di programmazione per lo sviluppo firmware

Nel contesto dello sviluppo di applicazioni per la scheda Adafruit Feather Sense nRF52840 è essenziale comprendere gli ambienti di programmazione compatibili. Questo capitolo si focalizzerà sull'analisi e la comparazione dei due principali ambienti supportati: CircuitPython e Arduino IDE.

2.1 Arduino IDE

Arduino IDE è un software utilizzato per la creazione di progetti con schede di sviluppo e microcontrollori. È un ambiente di programmazione integrato e multipiattaforma che permette di scrivere, compilare e caricare vari programmi sulla propria scheda. Il linguaggio di programmazione utilizzato per la stesura del codice deriva dal C e C++.

2.1.1 Guida all'installazione

Nella sezione *Learn* del sito Adafruit è presente una guida dettagliata all'installazione di Arduino IDE per la scheda Adafruit Feathe Sense [1], tuttavia se non si ha molta dimestichezza con l'utilizzo di questo software, la guida può risultare poco chiara, per questo motivo, di seguito è riportata una guida più concisa e semplificata.

Il primo passaggio consiste nello scaricare il software adeguato al sistema operativo del proprio computer dal sito *Arduino.cc* [2]. Una volta terminato il download estrarre l'applicazione Arduino IDE dal file .zip appena scaricato e, successivamente, seguire le istruzioni che compariranno a schermo.

Una volta terminata l'installazione si devono aggiungere i pacchetti di supporto per le schede Adafruit. Per fare ciò, selezionare "Preferences" dal menu "File" e aggiungere il seguente URL *https://adafruit.github.io/arduino-board-index/package_adafruit_index.json* nella sezione "Additional Board Manager URL". Terminato ciò, non resta che aggiungere i pacchetti di supporto specifici per le schede Adafruit nRF52. Selezionare "Boards Manager" dal menu "Tools" e cercare tramite la barra di ricerca "Adafruit nRF52" ed installare il pacchetto.

Come ultimo passaggio, rimangono da installare le librerie che permettono di utilizzare i sensori e i vari componenti della scheda. Selezionando "Include libraries" e successivamente "Manage libreries" dal menu "Sketch", è possibile cercare ed installare le librerie di ogni componente semplicemente digitandone il nome nell'apposita barra di ricerca.

2.2 CircuitPython

«CircuitPython è un linguaggio di programmazione progettato per semplificare la sperimentazione e l'apprendimento della codifica di schede con microcontrollori a basso costo». [3] Il linguaggio nasce nel 2017, partendo da MicroPython, un progetto open-source nato per rendere Python utilizzabile anche nella programmazione dei microcontrollori e Adafruit ne fa da promotore per la programmazione delle sue schede.

2.2.1 Guida all'installazione

Anche per questo linguaggio di programmazione, nella sezione *Learn* di Adafruit sono riportati tutti i dettagli per l'installazione e il corretto funzionamento [4]. Di seguito, invece, è riportata una guida semplificata.

Come prima cosa, per utilizzare CircuitPython, è necessario installare un editor di testo, è possibile utilizzare l'editor Mu. Una volta effettuato il download dell'applicazione dal sito *codewith.mu* [5] non rimane che seguire le istruzioni di intallazione per ottenere l'editor pronto all'uso.

Successivamente, si dovrà scaricare il software aggiornato di CircuitPython specifico per la scheda che si utilizza. Il download è possibile al seguente link *https://circuitpython.org/board/feather_bluefruit_sense/* [6] e, una volta terminato, non resta che installare il software nella scheda.

Per eseguire ciò, dopo aver connesso la scheda al computer tramite cavo USB, la si deve portare in modalità "bootloader" premendo due volte consecutive il pulsante RST. Verrà così riconosciuta dal computer una nuova unità rimovibile nella quale si deve trascinare il file contenente il software. A questo punto l'unità rimovibile scomparirà e, dopo alcuni istanti, il computer rileverà una nuova chiamata CIRCUITPY.

All'interno della nuova unità si trovano il file "code.py", in cui deve essere salvato il codice che la scheda eseguirà, e una cartella chiamata "lib" vuota.

Come ultimo passaggio, non resta che l'installazione delle librerie. È possibile scaricare l'intero pacchetto dalla sezione *Libraries* di CircuitPython [7] e, per l'installazione, è necessario solamente copiare il contenuto della cartella appena scaricata nella cartella "lib" dell'unità CIRCUITPY.

2.3 Confronto tra i due ambienti

La differenza sostanziale, tra le due opzioni sopra illustrate, consiste nella tipologia di linguaggio. Il C e C++, linguaggi utilizzati da Arduino IDE, sono linguaggi compilati, mentre CircuitPython è un linguaggio interpretato.

La differenza tra un linguaggio compilato ed uno interpretato risiede nel modo in cui essi interagiscono con la scheda.

Nei linguaggi come il C e il C++, il codice che viene scritto nell'editor di testo deve prima essere compilato, attraverso il compilatore, e trasformato in linguaggio macchina, solo successivamente all'esecuzione di questo passaggio. A causa di questo processo, ad ogni modifica del codice sorgente dovrà avvenire la ricompilazione del programma.

Al contrario, invece, i linguaggi come CircuitPython traducono ed eseguono ogni singola istruzione del programma e in caso di modifica del codice sorgente non richiede la ricompilazione.

Lo svantaggio dei linguaggi interpretati è invece la velocità di esecuzione, in quanto il processo di traduzione del codice durante l'esecuzione può sovraccaricare il sistema, rallentando il programma. Il progresso in velocità e memoria dei microcontrollori moderni sta però riducendo sempre di più il divario rendendolo ormai non più un problema.

Nel proseguio della tesi, i codici e gli esempi saranno presentati con il linguaggio interpretato CircuitPython.

Capitolo 3

Analisi e test dei sensori

3.1 Accelerometro e giroscopio LSM6DS33

La scheda Adafruit Feather Sense nrF52840 predispone on chip il sensore LSM6DS33.

Esso è compreso di un accelerometro e di un giroscopio, entrambi di tipo capacitivo MEMS, ed è considerato un'unità di misura inerziale a sei gradi di libertà di movimento (su-giù, destra-sinistra, avanti-indietro, rollio, imbardata e beccheggio), il che permette la misurazione e l'analisi dell'accelerazione lineare ed angolare.

Dal datasheet [8] ,le misure vengono rilevate sui tre assi X, Y, Z, il quale orientamento è raffigurato, nella scheda Feather Sense, di fianco al microcontrollore.

L'accelerazione lineare è misurata in m/s^2 mentre quella angolare in dps (gradi su secondo).

Il sensore LSM6DS33 può comunicare con il microcontrollore sia tramite bus I²C, sia con bus SPI e può operare a diverse frequenze di output (Output Data Rate, ODR): 12.5Hz 26Hz 52Hz 104Hz 208Hz 416Hz 833Hz 1666Hz 3332Hz 6667Hz.

Sia per l'accelerometro che per il giroscopio possono essere impostati diversi intervalli di fondo scala, raffigurati in Tab. 3.1 e Tab. 3.2, ad ognuno dei quali corrisponde una determinata sensibilità dello strumento.

Di seguito è riportato il codice in CircuitPython per il test completo del sensore LSM6DS33.

FS Range	Sensibilità
±2g	0.061mg/LSB
±4g	0.122mg/LSB
±8g	0.244mg/LSB
±16g	0.488mg/LSB

Tabella 3.1: Intervalli di fondo scala e rispettive sensibilità dell'accelerometro

FS Range	Sensibilità
±125dps	4.375mdps/LSB
±250dps	8.75mdps/LSB
±500dps	17.5mdps/LSB
±1000dps	35mdps/LSB
±2000dps	70mdps/LSB

Tabella 3.2: Intervalli di fondo scala e rispettive sensibilità del giroscopio

Il programma utilizza inizialmente il comando **accelerometer_range**, il quale va a modificare il valore di fondo scala dell'accelerometro. Le opzioni impostabili sono RANGE_2G, RANGE_4G, RANGE_8G e RANGE_16G. A titolo di esempio si è scelto di impostare il fondo scala a ±4G.

Per la modifica del fondo scala del giroscopio si utilizza invece il comando **gyro_range**. Per questo, i valori predisposti sono RANGE_125DPS, RANGE_250DPS, RANGE_500DPS, RANGE_1000DPS e RANGE_2000DPS. In questo caso si è scelto di utilizzare ±500dps.

Successivamente, con l'utilizzo del comando **accelerometer_data_rate** si va a variare il valore di frequenza di output a cui l'accelerometro opera. Le scelte possibili sono RATE_12_5_HZ, RATE_26_HZ, RATE_52_HZ, RATE_104_HZ, RATE_208_HZ, RATE_416_HZ, RATE_833_HZ, RATE_1666_HZ, RATE_3332_HZ e RATE_6667_HZ.

Il valore di frequenza di output a cui lavora il giroscopio viene corretto tramite l'utilizzo del metodo **gyro_data_rate** e i valori impostabili sono i medesimi del comando riguardante l'accelerometro. Per entrambi i rilevatori si è scelto di utilizzare una frequenza 104Hz.

Infine, tramite un ciclo while si stampano le misure effettuate.

```
"""
Test del sensore accelerometro e giroscopio LSM6DS33 6-DoF

Il programma si occupa di testare in tutti i suoi parametri il sensore,
il valore impostato di essi è a titolo d'esempio.
"""

import board
import time
import adafruit_lsm6ds.lsm6ds33
from adafruit_lsm6ds import Rate, AccelRange, GyroRange

i2c = board.I2C()

```

```
print("\n\nAdafruit␣LSM6DS33␣test!")
print("-------------------------␣\n")

sensor = adafruit_lsm6ds.lsm6ds33.LSM6DS33(i2c)

"""
Modifica del valore di fondo scala dell'accelerometro.
"""
sensor.accelerometer_range = AccelRange.RANGE_4G
accel_range = sensor.accelerometer_range
print("Accelerometer␣range␣set␣to:␣4G")

"""
Modifica del valore di fondo scala dell'accelerometro.
"""
sensor.gyro_range = GyroRange.RANGE_500_DPS
gyroscope_range = sensor.gyro_range
print("Gyroscope␣range␣set␣to:␣500DPS")

"""
Modifica del valore di frequenza di output a cui
l'accelerometro opera.
"""
sensor.accelerometer_data_rate = Rate.RATE_104_HZ
accel_rate = sensor.accelerometer_data_rate
print("Accelerometer␣data␣rate␣set␣to:␣104HZ")

"""
Modifica del valore di frequenza di output a cui
lil giroscopio opera.
"""
sensor.gyro_data_rate = Rate.RATE_104_HZ
print("Gyroscope␣data␣rate␣set␣to:␣104HZ")

while True:
    ac_x, ac_y, magc_z = sensor.acceleration
```

```
51      print("\nACCELERATION:␣")
52      print("X:{:.2f}␣Y:{:.2f}␣Z:{:.2f}m/s^2".format(ac_x, ac_y,ac_z))
53      gy_x, gy_y, gy_z = sensor.gyro
54      print("GYRO:␣")
55      print("X:{:.2f}␣Y:{:.2f}␣Z:{:.2f}dps".format(gy_x, gy_y, gy_z))
56      time.sleep(0.009)
```

3.2 Magnetometro LIS3MDL

Un ulteriore sensore on chip è il LIS3MDL, magnetometro a 3 assi (X, Y, Z) ad alte prestazioni e bassissimo consumo di potenza. L'orientamento degli assi è il medesimo del sensore LSM6DS33.

La direzione e l'intensità del campo magnetico misurato dal sensore è espresso in Gauss.

Come consultabile dal datasheet [9], il sensore LIS3MDL permette di scegliere tra due modalità di rilevazione della misura, in particolare tra *Single-measurement* e *Continuous-measurement*. In modalità *Single-measurement*, il dispositivo acquisisce una singola misura, ne salva il risultato nel registro "data output" e successivamente torna in modalità inattiva. In modalità *Continuous-measurement*, invece, il dispositivo acquisisce in maniera continua le misure.

Anche il sensore LIS3MDL può comunicare con il microcontrollore sia tramite bus I²C, che con bus SPI e può operare a diverse frequenze di output (Output Data Rate, ODR): 0.625Hz 1.25Hz 2.5Hz 5Hz 10Hz 20Hz 40Hz 80Hz; è possibile impostare frequenze più alte, quali 155Hz, 300Hz, 560Hz e 1000Hz, andando a modificare la modalità operativa del sensore. Le quattro modalità operative sono rispettivamente *Ultra high performance mode*, *High performance mode*, *Medium performance mode* e *Low power mode*. L'aumento delle prestazioni comporta una rilevazione della misura con meno rumore ma richiede tempi di avvio più lunghi.

È possibile scegliere tra diversi intervalli di fondo scala, ad ognuno dei quali corrisponde una determinata sensibilità dello strumento, come raffigurato in Tab. 3.3.

Di seguito è riportato il codice in CircuitPython per il test completo del sensore LIS3MDL.

Il programma si occupa di effettuare inizialmente alcune rilevazioni in modalità single-measurement e successivamente in modalità continuous-measurement.

Attraverso il comando **operation_mode**, si va a modificare la modalità di rilevazione della misura. Le tre possibili opzioni sono OperationMode.SINGLE, OperationMode.CONTINUOUS e OperationMode.POWER_DOWN.

FS Range	Sensibilità
±4Gauss	0.146mGauss/LSB
±8Gauss	0.292mGauss/LSB
±12Gauss	0.438mGauss/LSB
±16Gauss	0.584mGauss/LSB

Tabella 3.3: Intervalli di fondo scala e rispettive sensibilità dell magnetometro

Per variare il valore di fondo scala si modifica invece il parametro **range**. I valori impostabili sono Range.RANGE_4_GAUSS, Range.RANGE_8_GAUSS, Range.RANGE_12_GAUSS e Range.RANGE_16_GAUSS. A titolo di esempio è stato scelto ±8Gauss.

Si procede il programma tramite un ciclo for che effettua il test delle varie modalità di prestazione. Tali, vengono selezionate attraverso il comando **performance_mode** ed è possibile scegliere tra PerformanceMode.MODE_LOW_POWER, PerformanceMode.MODE_MEDIUM, PerformanceMode.MODE_HIGH e PerformanceMode.MODE_ULTRA.

La modalità continuous-measurement consente di scegliere il valore di frequenza di output a cui il sensore opera tra le seguenti opzioni, RATE_0_625_HZ, RATE_1_25_HZ, RATE_2_5_HZ, RATE_5_HZ, RATE_10_HZ, RATE_20_HZ, RATE_40_HZ e RATE_80_HZ. La selezione viene effettuata tramite la modifica al parametro **data_rate** e in questo caso è stato scelto 2.5Hz.

Infine, con un ulteriore ciclo for si stampano le misure rilevate in modalità continuous-measurement. Tra una stampa e l'altra si sceglie un tempo d'attesa in funzione della frequenza di output selezionata nei passaggi precedenti.

```
"""
Test del sensore accelerometro e giroscopio Adafruit LIS3MDL

Il programma si occupa di testare in tutti i suoi parametri il sensore,
il valore impostato di essi è a titolo d'esempio.
"""

import time
import board
from adafruit_lis3mdl import LIS3MDL,Rate,PerformanceMode
from adafruit_lis3mdl import OperationMode,Range

i2c = board.I2C()
sensor = LIS3MDL(i2c)

```

```
print("\nLIS3MDL test!")
print("-----------------------\n")

"""
Modifica della modalità di rilevazione della misura in single-measurement.
"""
print("SINGLE MEASUREMENT MODE\n")
sensor.operation_mode = OperationMode.SINGLE

"""
Variazione dell'intervallo di fondo scala del sensore.
"""
sensor.range = Range.RANGE_8_GAUSS

"""
Test dell funzionamento nelle varie modalità di prestazioni.
"""
for i in range(4):
    if i == 0:
        sensor.performance_mode = PerformanceMode.MODE_LOW_POWER
        print("Performance mode: LOW POWER")
    if i == 1:
        sensor.performance_mode = PerformanceMode.MODE_MEDIUM
        print("Performance mode: MEDIUM")
    if i == 2:
        sensor.performance_mode = PerformanceMode.MODE_HIGH
        print("Performance mode: HIGH")
    if i == 3:
        sensor.performance_mode = PerformanceMode.MODE_ULTRA
        print("Performance mode: ULTRA")

    for iter in range(1):
        mg_x, mg_y, mg_z = sensor.magnetic
        print("X:{:.2f},Y:{:.2f},Z:{:.2f}uT\n".format(mg_x, mg_y, mg_z))
```

```
"""
Modifica della modalità di rilevazione della misura in single-measurement.
"""
print("------------------------\n")
print("CONTINUOUS␣MODE\n")
sensor.operation_mode = OperationMode.CONTINUOUS

"""
Modifica dell valore di frequenza di output a cui il sensore opera.
"""
sensor.data_rate = Rate.RATE_2_5_HZ

for i in range(100):
    mg_x, mg_y, mg_z = sensor.magnetic
    print('X:{:.2f},Y:{:.2f},Z:{:.2f}uT'.format(mg_x, mg_y, mg_z))
    print('')
    time.sleep(0.4)

"""
Modifica della modalità di rilevazione della misura, questa volta
si sceglie OperationMode.POWER_DOWN in modo da interrompere
l'acquisizione continua delle misure.
"""
sensor.operation_mode = OperationMode.POWER_DOWN
```

3.3 Sensore di temperatura e pressione BMP280

La scheda offre, inoltre, il sensore di temperatura e pressione BMP280.

Esso si basa sul collaudato sensore di pressione piezoresistivo Bosch, tecnologia caratterizzata da elevata robustezza, precisione, stabilità e linearità. Come è possibile consultare dal datasheet [10], il BMP280 riesce a misurare temperature dai -40°C ai +85°C con una precisione assoluta di ±1°C e pressioni dai 300hPa ai 1100hPa con una precisione assoluta di ±1hPa. Siccome la pressione atmosferica cambia con l'altitudine, può anche essere utilizzato come altimetro con una precisione di ±1m.

Anche per questo sensore i protocolli di comunicazione con il microcontrollore sono l'I²C e l'SPI.

Il sensore BMP280 è provvisto di tre modalità operative: *Sleep mode*, *Forced mode* e *Normal mode*.

Nella *Sleep mode* il sensore è in standby, non vengono effettuate misure e il consumo di potenza è minimo.

Nella *Forced mode*, invece, viene eseguita una singola misura e successivamente il sensore torna alla *Sleep mode*, mentre per quanto riguarda il funzionamento in *Normal mode*, il sensore effettua misure continue ad una frequenza modificabile. Il periodo di campionamento delle misure in *Normal mode* viene identificato come periodo di stand-by e può essere scelto tra le seguenti opzioni: 0.5ms, 62.5ms, 125ms, 250ms, 500ms, 1s, 2s e 4s.

Di seguito è riportato il codice in CircuitPython utilizzato per effettuare il test del sensore.

Inzizialmente, tramite il comando **sea_level_pressure** si va ad impostare un numero intero equivalente alla pressione del livello del mare. Mediamente è circa di 1013hPa e verrà utilizzata come valore di riferimento per effettuare le misure.

In seguito, il programma si occupa di effettuare rilevazioni in modalità Forced mode e Normal mode. Tali modalità vengono selezionate tramite il parametro **mode** e le tre opzioni sono MODE_FORCE, MODE_NORMAL e MODE_SLEEP.

Il tempo di stand-by per la modalità Normal mode si seleziona variando il parametro **standby_period** e i possibili valori impostabili sono STANDBY_TC_0_5, STANDBY_TC_62_5, STANDBY_TC_125, STANDBY_TC_250, STANDBY_TC_500, STANDBY_TC_1000, STANDBY_TC_2000 e STANDBY_TC_4000. A titolo d'esempio nel seguente codice è selezionato STANDBY_TC_250 equivalente a 250ms.

```
"""
Test del sensore di temperatura e pressione BMP280

Il programma si occupa di testare in tutti i suoi parametri il sensore,
il valore impostato di essi è a titolo d'esempio.
"""

import board
import time
import adafruit_bmp280

i2c = board.I2C()
sensor = adafruit_bmp280.Adafruit_BMP280_I2C(i2c)

print("\n\nBMP280 test!")
```

```
16 print("-------------------------\n")
17
18 """
19 Si imposta il valore di pressione al livello del mare.
20 """
21 sensor.sea_level_pressure = 1013
22
23 """
24 Modifica della modalità di rilevazione della misura in MODE_FORCE.
25 """
26 sensor.mode = adafruit_bmp280.MODE_FORCE
27 print("FORCED␣MODE\n")
28
29 print("Temperature:␣%0.1f␣C␣" % sensor.temperature)
30 print("Pressure:␣%0.1f␣hPa␣" % sensor.pressure)
31 print("Altitude␣=␣%0.2f␣meters␣" % sensor.altitude)
32
33 """
34 Si ripete il comando mode, questa volta si imposta MODE_NORMAL in modo
35 tale da rilevare misure in maniera continua.
36 """
37 print("-------------------------")
38 print("\nNORMAL␣MODE")
39 sensor.mode = adafruit_bmp280.MODE_NORMAL
40
41 """
42 Modifica del periodo di standby utilizzato per la rilevazione delle
43 misure in Normal mode.
44 """
45 sensor.standby_period = adafruit_bmp280.STANDBY_TC_250
46
47 """
48 Il sensore ha bisogno di un istante per raccogliere la prima misurazione
49 """
50 time.sleep(1)
51
```

```
for i in range(100):
    print("\nTemperature: %0.1f C " % sensor.temperature)
    print("Pressure: %0.1f hPa " % sensor.pressure)
    print("Altitude = %0.2f meters " % sensor.altitude)
    time.sleep(0.250)

"""
Si ripete il comando mode. Si sceglie MODE_SLEEP in modo da
interrompere l'acquisizione continua delle misure.
"""
sensor.mode = adafruit_bmp280.MODE_SLEEP
```

3.4 Sensore di umidità e temperatura SHT31

Installato nella scheda, si trova anche il sensore di umidità e temperatura SHT31, prodotto da Sensirion, il quale permette di misurare con precisione e stabilità la temperatura e l'umidità dell'ambiente in cui si trova.

Come dichiarato dal datasheet [11], i protocolli di comunicazione con il microcontrollore, per questo sensore, sono l'I²C e l'SPI.

Questo misura temperature in un range tra i -40°C e i 125°C, ciò nonostante, per ottenere le massime prestazioni da specifiche, è consigliato l'utilizzo per un intervallo di temperature tra i 5°C e i 60°C e di umidità tra il 20%RH e l'80%RH (*Relative Humidity*).

Il sensore è in grado di misurare l'umidità con una risoluzione di 0.01%RH e con un accuratezza sulla misura di ±2%RH, per quanto riguarda le misure di temperatura, la risoluzione è di 0.01°C e l'accuratezza sulla misura è ±0.2°C.

Il modulo SHT31 permette di poter scegliere tra tre diverse modalità, le quali vanno a modificare la ripetibilità e il tempo di acquisizione delle misure. La ripetibilità è definita come il grado di concordanza che si riscontra sulle misure, quando lo strumento misura uno stesso misurando, alle medesime condizioni. Tali modalità sono definite come *low repeatability*, *medium repeatability* e *high repeatability*. Le caratteristiche di queste modalità sono visibili in Tab. 3.4.

In aggiunta, anche per questo sensore, è possibile selezionare la modalità di rilevazione delle misure tra *Single shot mode* e *Periodic data acquisition mode*.

In *Single shot mode*, quando viene emesso un comando di misurazione, il sensore avvia l'acquisizione di una coppia di dati, in particolare, viene acquisita una misura di umidità e una misura di temperatura.

Mode	Humidity Rep.	Temperature Rep.	Measurement duration
Low repeatability	0.21%RH	0.15°C	2.5ms - 4.5ms
Medium repeatability	0.15%RH	0.08°C	4.5ms - 6ms
High repeatability	0.08%RH	0.04°C	12.5ms - 15ms

Tabella 3.4: Caratteristiche delle opzioni di ripetibilità impostabili.

In *Periodic data acquisition mode*, invece, il sensore avvia l'acquisizione di un flusso di coppie di dati. Per questa modalità è possibile impostare una frequenza di acquisizione delle misure tra i seguenti valori: 0.5, 1, 2, 4 e 10 misurazioni per secondo. Il flusso di misurazioni viene salvato sottoforma di array in una cache interna.

Di seguito è riportato il codice in CircuitPython utilizzato per effettuare il test del sensore SHT31.

Anche per questo modulo il programma si occupa di effettuare rilevazioni con entrambe le modalità operative, Single shot mode e Periodic data acquisition mode. Tali modalità vengono selezionate tramite il parametro **mode** e i valori impostabili sono MODE_SINGLE e MODE_PERIODIC.

In modalità Single shot mode, tramite un ciclo for si verifica il funzionamento nelle varie impostazioni di ripetibilità. Mediante il comando **repeatability** si modifica proprio tale caratteristica e le opzioni impostabili sono REP_LOW, REP_MED e REP_HIGH, corrispondenti rispettivamente a low repeatability, medium repeatability e high repeatability.

Con il comando **frequency** si va a modificare la frequenza di acquisizione delle misure in modalità Periodic data acquisition mode. I valori che è possibile selezionare sono FREQUENCY_0_5, FREQUENCY_1, FREQUENCY_2, FREQUENCY_4 e FREQUENCY_10, corrispondenti rispettivamente a 0.5, 1, 2, 4 e 10 misurazioni al secondo. Nel seguente codice si è selezionato ad esempio FREQUENCY_2.

```
"""
Test del sensore accelerometro e giroscopio Adafruit LSM6DS33 6-DoF

Il programma si occupa di testare in tutti i suoi parametri il sensore,
il valore impostato di essi è a titolo d'esempio.
"""

import board
import time
import adafruit_sht31d

```

```
i2c = board.I2C()
sensor = adafruit_sht31d.SHT31D(i2c)

print("SHT31D test!\n")
print("------------------------\n")

"""
Modifica della modalità di rilevazione della misura in MODE_SINGLE.
"""
print("Modalità di funzionamento single shot\n")
sensor.mode = adafruit_sht31d.MODE_SINGLE

"""
Test delle varie impostazioni di ripetibilità.
"""
for i in range(3):
    if i == 0:
        sensor.repeatability = adafruit_sht31d.REP_LOW
        print("REPEATABILITY MODE: LOW")
    if i == 1:
        sensor.repeatability = adafruit_sht31d.REP_MED
        print("REPEATABILITY MODE: MEDIUM")
    if i == 2:
        sensor.repeatability = adafruit_sht31d.REP_HIGH
        print("REPEATABILITY MODE: HIGH")

    for iter in range(1):
        print("Temperature: {:.3f} °C".format(sensor.temperature))
        print("Humidity: {:.2f} %\n".format(sensor.relative_humidity))

"""
Si ripete il comando mode, questa volta si seleziona MODE_PERIODIC.
"""
print("Modalità di funzionamento periodic\n")
sensor.mode = adafruit_sht31d.MODE_PERIODIC

```

```
"""
Modificare della frequenza di acquisizione delle misure.
"""
sensor.frequency = adafruit_sht31d.FREQUENCY_2

"""
Si ripete il comando repeatability, in questo caso si utilizza REP_MED,
a titolo d'esempio.
"""
sensor.repeatability = adafruit_sht31d.REP_MED

"""
Tempo d'attesa di 5 secondi in modo da attonere 10 misurazioni, tale
numero è stato scelto come esempio.
"""
time.sleep(5)

print("Temperature:␣", sensor.temperature)
print("Humidity:␣", sensor.relative_humidity)

"""
Si ripete il comando mode e si imposta MODE_SINGLE in modo tale da
interrompere l'acquisizione continua delle misure.
"""
sensor.mode = adafruit_sht31d.MODE_SINGLE
```

3.5 Rilevatore di luce, prossimita e gesti APDS9960

La scheda Adafruit Feather Sense nRF52840 predispone on chip anche il modulo APDS9960, i cui dati tecnici sono consultabili dal rispettivo datasheet [12].

Tale sensore è dotato di rilevamento avanzato dei gesti, rilevamento di prossimità e rilevamento della luce ambientale digitale (ALS) e Sensore colore (RGBC).

Anche per questo sensore, i protocolli di comunicazione con il microcontrollore sono l'I²C e l'SPI.

I gesti vengono rilevati tramite quattro fotodiodi direzionali che rilevano l'energia infrarossa riflessa proveniente da un LED integrato nel modulo, e convertono le informazioni sul

movimento fisico ovvero velocità, direzione e distanza, in informazioni digitali. Il rilevatore di gesti riconosce i principali movimenti dei dispositivi mobili, come per esempio: semplici gesti su-giù-destra-sinistra.

Tramite lo stesso principio di funzionamento, ovvero la rilevazione da parte di un fotodiodo dell'energia infrarossa riflessa, vengono rilevate anche le misure di prossimità. Tali misurazioni sono rappresentate da un numero compreso tra 0 e 255, il quale rappresenta l'intensità, della luce IR riflessa, rilevata dai fotodiodi interni del sensore. Un valore 0 indica che non è stata ricevuta luce infrarossa riflessa, cioè che non si trovavano oggetti nel campo visivo del sensore. Un valore 255, invece, indica che è stata ricevuta la quantità massima rilevabile di luce IR riflessa, cioè che è stato rilevato un oggetto molto vicino al sensore.

La funzione di rilevamento della luce fornisce informazioni sull'intensità delle componenti, rossa, verde, blu, e chiara. Per ogni componente il risultato della rilevazione è espresso con un valore tra 0 e 65535.

Di seguito viene presentato il codice in Circuitpython utilizzato per effettuare il test del sensore.

Con il comando **enable_proximity** si abilita o disabilita la rilevazione del sensore di prossimità. Nel caso del programma sotto riportato si imposta inizialmente il parametro come True, effettuando la misura e successivamente si ripete il comando impostandolo come False per disabilitare la rilevazione.

La rilevazione della luce si abilita e disabilita con il parametro **enable_color**. Impostandolo come True si abilità la rilevazione, mentre impostandolo come False si disabilita.

Tramite un ciclo while si controlla il parametro **color_data_ready** richiamandolo ogni 5ms. Il parametro restituisce 0 se non è stata rilevata nessuna misura, restituisce 1, invece, se una nuova misura è pronta per essere letta.

In seguito si effettua la misura e si stampano i risultati. Tali, vengono poi utilizzati nelle funzioni **calculate_color_temperature** e **calculate_lux** [13]. La prima è una funzione che calcola la temperatura del colore in gradi Kelvin, mentre la seconda calcola il livello di luce ambientale in lux.

Infine, tramite il parametro **enable_gesture** si abilita e disabilita la rilevazione dei gesti, si imposta True per abilitare e False per disabilitare. Il ciclo while finale permette di rilevare e stampare i vari gesti.

```
"""
Test del sensore rilevatore di prossimità, gesti e luce APDS9960.
Il programma si occupa di testare in tutti i suoi parametri il sensore,
il valore impostato di essi è a titolo d'esempio.
"""
```

```
import time
import board
from adafruit_apds9960.apds9960 import APDS9960
from adafruit_apds9960 import colorutility

i2c = board.I2C()
sensor = APDS9960(i2c)

print("\n\nAPDS9960␣test!")
print("------------------------\n")

"""
Si abilita la rilevazione di misure di prossimità, si effettua la
misura e successivamente, si disabilita la rilevazione.
"""
sensor.enable_proximity = True
print("PROXIMITY␣LEVEL:␣\n")
print(sensor.proximity)
sensor.enable_proximity = False


"""
Si abilita la rilevazione della luce.
"""
sensor.enable_color = True
print("\n------------------------\n")
print("DATA␣COLOUR:␣")

"""
Controllo del parametro color_data_ready
"""
while not sensor.color_data_ready:
    time.sleep(0.005)

r, g, b, c = sensor.color_data
print("\nRed:␣", r)
```

```
print("\nGreen: ", g)
print("\nBlue: ", b)
print("\nClear: ", c)

"""
Calcolo della temperatura del colore in gradi Kelvin.
"""
temp = colorutility.calculate_color_temperature(r, g, b)
print("Color temperature {}".format(temp))

"""
Calcolo del livello di luce ambientale in lux.
"""
ambient = colorutility.calculate_lux(r, g, b)
print("Ambient light {}".format(ambient))

"""
Si abilita la rilevazione dei gesti.
"""
sensor.enable_proximity = True
sensor.enable_gesture = True
print("\n------------------------\n")
print("GESTURE: \n")

"""
Si utilizza un ciclo infinito che permette di rilevare i vari gesti
"""
while True:
    gesture = sensor.gesture()

    if gesture == 0x01:
        print("Swipe up \n")
    elif gesture == 0x02:
        print("Swipe down \n")
    elif gesture == 0x03:
        print("Swipe left \n")
```

```
78      elif gesture == 0x04:
79          print("Swipe right \n")
```

3.6 Microfono PDM MP34DT01-M

Infine, l'ultimo sensore presente nelle scheda Adafruit Feather Sense è il microfono PDM MP34DT01-M.

Come descritto nel datasheet [14],il modulo MP34DT01-M è un microfono MEMS (*Micro Electro Mechanical Systems*) digitale ultracompatto, a bassa potenza e omnidirezionale, costruito con un elemento sensibile capacitivo e interfaccia di comunicazione I²S, *Inter-IC Sound*.

L'elemento sensibile, in grado di rilevare onde acustiche, è prodotto utilizzando un processo specializzato di microlavorazione del silicio, dedicato alla produzione di sensori audio.

Il sensore ha buone caratteristiche acustiche, in particolare, un punto di sovraccarico acustico di 120dBSPL con un rapporto segnale-rumore di 61dB e sensibilità –26dBFS.

Di seguito è riportato il codice in CircuitPython utilizzato per effettuare il test del sensore.

Si utilizza il comando **audiobusio.PDMIn(clock_pin, data_pin, sample_rate, bit_depth)**, della libreria audiobusio, per creare un oggetto di tipo PDMIn associato al microfono PDM MP34DT01-M installato nella scheda. I quattro parametri impostati sono rispettivamente, il pin di clock del sensore, il pin da cui leggere i dati del sensore, il sample rate e il numero di bit per campione.

Successivamente si definisce una funzione chiamata **normalized_rms**, la quale restituisce una media normalizzata di valori efficaci. Si utilizza questa funzione per rilevare i campioni sonori molto rapidamente e restituirne una media così da ottenere una lettura più accurata.

Infine, si utilizza un ciclo while infinito per acquisire e stampare i dati. In tale ciclo, per prima cosa si crea un array di 160 interi senza segno inizializzati a 0 e, successivamente, con il comando **record(destination, destination_length)** si acquisiscono 160 dati e si salvano sull'array appena creato. Si stampa la media normalizzata dei 160 valori acquisiti così da avere una lettura più precisa. Il tutto viene ripetuto ogni 2 secondi.

```
1  """
2  Test del sensore microfono PDM MP34DT01-M.
3
4  Il programma si occupa di testare in tutti i suoi parametri il sensore,
5  il valore impostato di essi è a titolo d'esempio.
6  """
7
```

```
import board
import time
import math
import array
import audiobusio

"""
Si crea un oggetto di tipo PDMIn associato al microfono PDM MP34DT01-M
installato nella scheda.
"""
microphone = audiobusio.PDMIn(board.MICROPHONE_CLOCK, board.MICROPHONE_DATA,
                              sample_rate=16000, bit_depth=16)

"""
Definizione della funzione normalized_rms.
"""
def normalized_rms(values):
    minbuf = int(sum(values)/len(values))
    return int(math.sqrt(sum(float(sample-minbuf)*(sample-minbuf)
                             for sample in values)/len(values)))

"""
Ciclo while che esegue le rilevazioni, processa e stampa i risultati.
"""
while True:
    samples = array.array('H', [0] * 160)
    microphone.record(samples, len(samples))
    print("Sound␣level:", normalized_rms(samples))
    time.sleep(2)
```

Capitolo 4

Analisi dell'interfacciamento dei sensori tramite bus I²C

4.1 Introduzione al funzionamento del protocollo I²C

Il protocollo I²C è stato sviluppato dalla *Philips Semiconductors* nel 1982 e la sigla sta per *Inter-integrated Circuit*. Questo protocollo consente la comunicazione di dati tra due o più dispositivi sfruttando l'utilizzo di un singolo bus a due fili.

Le informazioni vengono trasmesse in modo seriale attraverso due linee: SDA (linea adibita alla trasmissione dei dati) e SCL (linea adibita alla trasmissione del segnale di clock). In relazione a questo, a volte, tale standard di trasmissione viene chiamato anche Two-Wire Serial Interface (TWI).

In realtà, sono necessarie anche una terza e quarta linea, corrispondenti rispettivamente alla massa comune e alla tensione di alimentazione, condivise da tutti i dispositivi.

Tale protocollo di comunicazione prevede l'interazione tra due tipologie di dispositivi: il master e gli slave.

Il master è il dispositivo che controlla il bus gestendone il segnale di clock e generando i segnali di start e stop. In particolare, il segnale di clock deve rispettare alcuni standard, i quali prevedono una frequenza massima di 100KHz (*Standard mode*), 400KHz (*Fast mode*) e 3.4MHz (*High speed mode*). Nella praticità, viene normalmente utilizzata solo la prima, nonostante molti dispositivi tra cui la scheda Adafruit Feather Sense, supportino anche la seconda.

Gli slave, invece, si limitano ad ascoltare il bus, ricevendo dati dal master o inviandoli su richiesta di quest'ultimo.

Una semplice sequenza di lettura o scrittura di dati tra un master ed uno slave segue precise istruzioni.

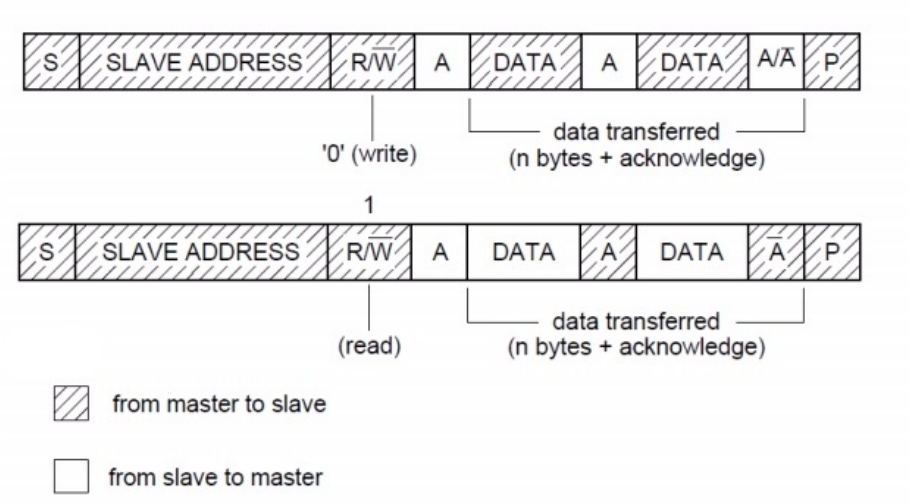

Figura 4.1: Sequenza di scrittura e lettura di dati tramite bus I²C. (Protocollo-di-comunicazione-I2C, Samuele, 2017, p.8)

La comunicazione inizia con l'invio, da parte del master, del bit di start (S), a cui segue l'invio dei bit corrispondenti all'indirizzo dello slave (ADDR) e, a catena, l'invio del bit che manifesta se effettuare la lettura (R) o la scrittura (W) di dati. Gli indirizzi degli slave sono solitamente a 7 o 10 bit.

In seguito a ciò, avviene un passaggio fondamentale per la comunicazione seriale, vale a dire l'invio o l'attesa del bit di riconoscimento (ACK). Tale bit ha il compito di comunicare la corretta ricezione dei dati.

Infine, in successione, avviene l'invio o la ricezione del byte di dati (DATA), l'attesa o l'invio di un ulteriore bit di riconoscimento e, in ultimo, l'invio del bit di stop (P) da parte del master.

Il processo di scrittura e lettura è illustarto in Fig. 4.1.

È possibile, tramite un metodo chiamato *repeated start*, effettuare sia operazioni di scrittura che di lettura, il tutto generando, in un'unica sequenza, una singola condizione di stop. Tale metodo prevede che, al termine di un operazione, invece di generare la condizione di stop venga prodotto un nuovo comando di start. Il tutto seguito nuovamente da un indirizzo e dal bit di lettura o scrittura.

4.2 Protocollo I²C nella scheda Adafruit Feather Sense

La scheda Adafruit Feather Sense dispone di un protocollo di comunicazione I²C a singolo master, rappresentato in Fig. 4.2, il quale è compatibile con le modalità di funzionamento a 100KHz e 400KHz. Tutti i dettagli tecnici sono consultabili dal datasheet [15].

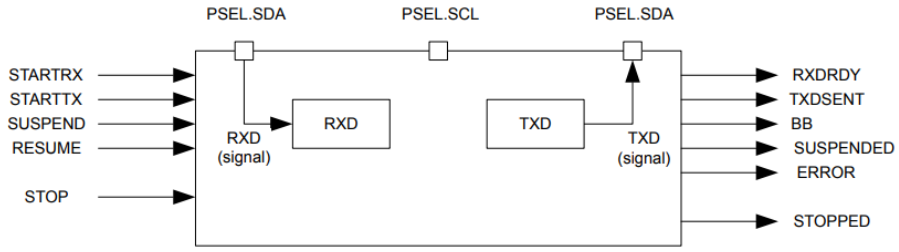

Figura 4.2: TWI master presente nella scheda Adafruit Feather Sense.

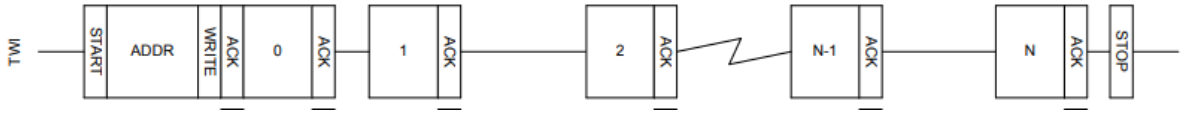

Figura 4.3: Sequenza di scrittura su uno slave da parte del master.

4.2.1 Sequenza di scrittura da parte del master

La sequenza di scrittura comincia mediante l'invioo del comando di start (*start transmission* STARTTX).

Successivamente a tale comando, il master invia al bus la condizione di start, seguita a catena da un indirizzo e da un bit, il quale indica se effettuare la scrittura o la lettura dei dati dallo slave (scrittura = 0 e lettura = 1). L'indirizzo inviato al bus deve coincidere con l'indirizzo dello slave su cui il master vuole effettuare l'operazione di scrittura.

Il bit di scrittura o lettura è seguito poi da un bit di riconoscimento o non riconoscimento, rispettivamente ACK e NACK (ACK = 0 e NACK = 1), generato dallo slave. In seguito alla ricezione di tale bit, il master spedisce sul bus, in sincronia con il clock, i byte di dati che sono scritti nel registro TXD. Ogni byte generato dal master sarà, anch'esso, seguito da un bit di riconoscimento generato dallo slave.

Il trasmettitore del master è a singolo buffer, perciò un secondo byte può essere scritto nel registro TXD solo dopo che il byte precedente è stato spedito e il bit di riconoscimento è stato generato.

La sequenza di scrittura termina quando viene attivato il comando di stop. In seguito a tale attivazione, il master invia la condizione di stop nel bus e interrompe la comunicazione. Tale sequenza è raffigurata in Fig. 4.3.

4.2.2 Sequenza di lettura da parte del master

La sequenza di lettura comincia anch'essa, mediante l'invio del comando di start (*start receiving* STARTRX).

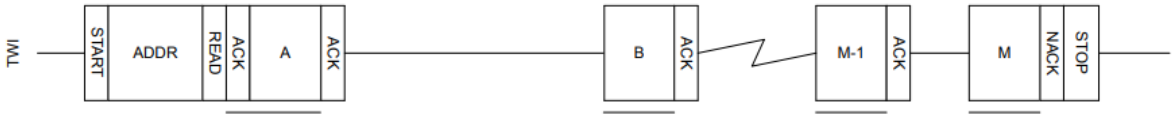

Figura 4.4: Sequenza di lettura dello slave da parte del master.

Successivamente, sino all'invio del primo bit di riconoscimento, la sequenza procede in maniera identica a quella descritta per l'operazione di scrittura. In seguito alla ricezione di tale bit, lo slave invia i dati al master utilizzando il segnale di clock generato dal master.

Dopo aver ricevuto 1 byte, il master ritarderà l'invio del bit ACK/NACK allungando l'impulso di clock fino a che la CPU non avrà estratto il byte ricevuto, cioè finché non avrà letto il registro RXD.

La sequenza di lettura termina quando viene attivato il comando di stop. Questo comando deve essere attivato prima che l'ultimo byte sia estratto dal registro RXD per assicurarsi che il master invii il bit NACK allo slave prima di generare nel bus la condizione di stop. Tale sequenza è raffigurata in Fig. 4.4.

4.3 Esempio di implementazione del protocollo I²C con CircuitPython

Di seguito si presenta un esempio di implementazione ed utilizzo del protocollo I²C con CircuitPython.

Il programma, inizialmente, tramite il comando **try_lock()** blocca il bus I²C in modo tale che possa essere utilizzato esclusivamente da questo codice in esecuzione. Tale comando è inserito in un ciclo while, il quale attende fino a che il bus I²C non risulti effettivamente bloccato.

Successivamente, si occupa di effettuare la scansione di tutti gli indirizzi degli slave tramite il comando **scan()** e si mette in comunicazione con uno di essi in modo tale di poter realizzare delle operazioni di lettura e scrittura.

Nel caso seguente, si è scelto il registro *CTRL_REG2* con indirizzo 0x21 del sensore magnetometro LIS3MDL avente indirizzo 0x1C. Come è possibile osservare in Fig. 4.5, la struttura del registro CTRL_REG2 è formata da un singolo byte in cui si mettono in evidenza quattro bit, FS1, FS0, REBOOT e SOFT_RST.

I bit REBOOT e SOFT_RST vengono tralasciati e mantengono il loro valore predefinito, vale a dire 0.

I bit FS1 e FS0 contengono le informazioni riguardanti il valore di fondo scala impostato nel sensore, il programma, nella seconda parte, si occupa quindi di leggere e modificare il valore di tali bit.

FS1	FS0	Full-Scale
0	0	±4Gauss
0	1	±8Gauss
1	0	±12Gauss
1	1	±16Gauss

Tabella 4.1: Selezione del fondo scala

0(1)	FS1	FS0	0(1)	REBOOT	SOFT_RST	0(1)	0(1)

1. *This bit must be set to 0 for the correct operation of the device.*

Figura 4.5: Struttura del registro CTRL_REG2.

In Tab. 4.1 è possibile osservare la correlazione tra i valori di fondoscala e i valori dei singoli bit.

La selezione del registro *CTRL_REG2* sieffettua con il comando **writeto(address, bytes[register])**, in cui il parametro address rappresenta l'indirizzo dello slave mentre register è il byte che rappresenta l'indirizzo del registro su cui si vuole effettuare, in questo caso, l'operazione di lettura.

Il comando **readfrom(address, result)** si effettua la lettura del registro "register" e si salva il contenuto nell'istanza result.

L'operazione di scrittura viene effettuata tramite il comando **writeto(address, bytes[register, value])**, con il quale si scrive il parametro "value" nel registro "register".

In seguito, si effettua nuovamente la sequenza per la lettura di tale registro come prova che l'operazione di scrittura sia andata a buon fine e, infine, si utilizza il comando **unlock()** per sbloccare il bus I²C.

```
"""
Esempio del funzionamento del bus I²C tramite la libreria busio
"""
import busio
import board
import time

i2c = busio.I2C(board.SCL, board.SDA)

"""
Blocco del bus I²C in modo tale che possa essere utilizzato
esclusivamente da questo programma in esecuzione.
"""
```

```
while not i2c.try_lock():
    pass

"""
Scansione di tutti gli indirizzi dei dispositivi collegati al bus I²C.
"""
lista = [hex(x) for x in i2c.scan()]
print(lista)

"""
Si seleziona e si legge il registro CTRL_REG2
"""
i2c.writeto(0x1C, bytes([0x21]))
result = bytearray(1)
i2c.readfrom(0x1C, result)
print(bin(result)[2:].zfill(8))

"""
Si effettua l'operazione di scrittura
"""
i2c.writeto(0x1C, bytes([0x21, 0x20]))

"""
Si effettua nuovamente la sequenza per la lettura del registro come
prova che l'operazione di scrittura sia andata a buon fine.
"""
i2c.writeto(0x1C, bytes([0x21]))
result = bytearray(1)
i2c.readfrom(0x1C, result)
print(bin(result)[2:].zfill(8))

"""
Si sblocca il bus I²C.
"""
i2c.unlock()
```

Capitolo 5

Conclusioni e possibili studi futuri

Questa tesi ha presentato l'analisi e la programmazione del sistema multi-sensore Adafruit Feather Sense nRF52840.

Gli obiettivi posti all'inizio del lavoro, ossia l'analisi e il test della componentistica hardware, focalizzata soprattutto ai sensori presenti on-board, e l'analisi dell'interconnessione tra sensore e microcontrollore tramite bus I²C sono stati rispettati.

In una futura fase dello studio sarà utile analizzare gli ulteriori protocolli di comunicazione supportati, come ad esempio il protocollo SPI e la tecnologia wireless Bluetooth Low Energy, e le prestazioni energetiche della scheda.

Un ulteriore sviluppo futuro potrà consistere nell'estendere l'analisi all'interazione con altre schede o sensori e attuatori esterni, valutando così le capacità di espansione e flessibilità della scheda.

Bibliografia

[1] Kattni Rembor, Dan Halbert, Thach Ha, Eva Herrada, Carter Nelson, *Arduino Support Setup*, Consultato il 16/04/2024, 2020. indirizzo: `https://learn.adafruit.com/adafruit-feather-sense/arduino-support-setup`.

[2] Arduino, *Arduino Software Download*, Consultato il 16/04/2024. indirizzo: `https://www.arduino.cc/en/software`.

[3] MicroPython & CircuitPython contributors, *CircuitPython*, Consultato il 16/04/2024, 2014. indirizzo: `https://circuitpython.org/`.

[4] lady ada, Kattni Rembor, Tim C, Jay Doscher, Michael Schroeder, *Welcome To CircuitPython*, Consultato il 18/04/2024, 2017. indirizzo: `https://learn.adafruit.com/welcome-to-circuitpython/overview`.

[5] Nicholas H.Tollerve, *Code with Mu: a simple Python editor for beginner programmers.* Consultato il 18/04/2024. indirizzo: `https://codewith.mu/`.

[6] CircuitPython, *Feather Bluefruit Sense*, Consultato il 18/04/2024. indirizzo: `https://circuitpython.org/board/feather_bluefruit_sense/`.

[7] CircuitPython, *CircuitPython Libraries*, Consultato il 18/04/2024. indirizzo: `https://circuitpython.org/libraries`.

[8] *iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope*, STMicroelectronics, 2020. indirizzo: `https://www.st.com/resource/en/datasheet/lsm6dsox.pdf`.

[9] *Digital output magnetic sensor: ultralow-power, high-performance 3-axis magnetometer*, STMicroelectronics, 2023. indirizzo: `https://www.st.com/resource/en/datasheet/lis3mdl.pdf`.

[10] *BMP280 Digital Pressure Sensor*, Bosch Sensortec, 2015. indirizzo: `https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp280/`.

[11] *Datasheet SHT3x-DIS Humidity and Temperature Sensor*, Sensirion, 2022. indirizzo: https://sensirion.com/media/documents/213E6A3B/63A5A569/Datasheet_SHT3x_DIS.pdf.

[12] *APDS-9960 Digital Proximity, Ambient Light, RGB and Gesture Sensor*, Avago Technologies, 2013. indirizzo: https://cdn.sparkfun.com/assets/learn_tutorials/3/2/1/Avago-APDS-9960-datasheet.pdf.

[13] Michael McWethy, *calculate$_c$olor$_t$emperature*, 2017. indirizzo: https://docs.circuitpython.org/projects/apds9960/en/latest/_modules/adafruit_apds9960/colorutility.html#calculate_color_temperature.

[14] *MP34DT01-M MEMS audio sensor omnidirectional digital microphone*, STMicroelectronics, 2014. indirizzo: https://www.st.com/en/audio-ics/mp34dt01-m.html.

[15] *nRF52840 Product Specification*, Nordic Semiconductors, 2019. indirizzo: https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf.