POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

—

Programmazione di sistemi di sensing indossabili per training di riabilitazione clinici

Studente
Bernardo Destito

Relatori
Prof. Giorgio De Pasquale
Prof. Andrea Acquaviva

A.A. 2018-2019

Indice

Indice delle figure

Indice delle tabelle

Introduzione

La mano umana è, fin da sempre, uno degli oggetti di studio più intriganti. Scienziati, medici e ingegneri hanno cercato, e tuttora cercano, di analizzarne la complessità strutturale e i molteplici movimenti che ci consentono di svolgere innumerevoli operazioni. Stringere una penna per scrivere, impugnare una posata per mangiare, afferrare un oggetto per spostarlo, sono tutti esempi di movimenti che il nostro cervello ha assimilato senza neanche rendercene conto; movimenti per i quali il nostro sistema nervoso centrale, insieme a quello periferico, mettono in atto dei meccanismi grazie ai quali riusciamo a muovere tutto l'insieme delle articolazioni delle dita (falangi) e del polso in maniera ottimale ed efficiente al fine di eseguire una determinata azione. Anche un semplice *"girarsi i pollici"* potrebbe apparire un'operazione del tutto banale, ma che in realtà banale non è!
La continua evoluzione della tecnologia riveste, senza dubbio, un ruolo molto importante nella ricerca scientifica e, di conseguenza, anche lo studio dei movimenti della mano ha subito nel tempo grandi cambiamenti; basti pensare come a partire da studi come quelli delle neuroscienze e dell'ingegneria biomedica si sia arrivati ad applicazioni come l'interazione uomo-computer (in inglese *Human-Computer Interaction*, HCI), l'intelligenza artificiale o addirittura repliche robotizzate.
Tra i numerosi studi e applicazioni, di notevole interesse sono quelli nell'ambito della riabilitazione medica e della valutazione fisiologica di pazienti che hanno in parte perso la motilità della mano e che fanno uso di sistemi cosiddetti *Glove-Based.*
Oggetto di questa tesi sarà, pertanto, l'analisi e la programmazione di un sistema *Glove-Based*, ovvero un particolare guanto sensorizzato capace di acquisire grandezze come flessione delle dita, forza esercitata dai polpastrelli, accelerazione e rotazione del polso, temperatura della mano, battito cardiaco e misura dell'impedenza della pelle.
Il progetto, in collaborazione con il gruppo di ricerca SAMBA *(SpAtial, Motor & Bodily Awareness)* del Dipartimento di Psicologia della Neuroscienza dell'Università di Torino, intende fornire uno strumento utile allo studio dei movimenti della mano in pazienti con ridotta motilità dell'arto superiore. Per lo studio di questi movimenti, già predefiniti e appositamente studiati, occorre effettuare dei cosiddetti *training* di riabilitazione clinici. Ciascuno di questi training si pone come obiettivo la misura di determinate grandezze e, di conseguenza, il guanto sensorizzato dovrà provvedere all'abilitazione di determinati sensori, effettuare le misure e organizzare i dati in modo da inviarli al computer per essere analizzati e memorizzati in tempo reale.
La scelta per l'implementazione della comunicazione è ricaduta sull'impiego dello standard wireless Bluetooth, in particolare quello di ultima generazione: il *Bluetooth 5.*

Come si vedrà nel relativo capitolo, esso mette a disposizione nuove funzionalità rispetto allo standard delle precedenti generazioni e, per questo motivo, è stato scelto un microcontrollore che integra tale tipo di comunicazione.
Ciascun capitolo, a partire da quelli introduttivi che forniranno i concetti base sui guanti sensorizzati, tratterà una determinata problematica della progettazione di un sistema *glove-based,* fino ad arrivare allo sviluppo del firmware vero e proprio, oggetto principale di questa tesi, fornendo le varie soluzioni e strategie che sono state messe in atto per l'acquisizione dei sensori e l'implementazione della connettività wireless per l'invio dei dati.
In particolare, il primo capitolo verterà sulla descrizione di vari *wired-glove* realizzati nel corso degli anni da diversi gruppi di ricerca nel mondo e le varie migliorie apportate da ciascuno, fornendo una descrizione generale della mano umana utile a comprendere i tipi di sensori da impiegare, il loro collocamento, nonché i materiali solitamente adoperati per questo genere di applicazione.
Nel secondo capitolo verranno introdotti i *training* che sono stati definiti dal gruppo di ricerca SAMBA, le grandezze di interesse e quindi i sensori impiegati nelle misure. Sarà descritta tutta l'elettronica che è stata realizzata per il guanto sensorizzato e, quindi, tutta la fase di progettazione del PCB di partenza *(Evaluation Board)*, come ad esempio il condizionamento dei sensori, fino ad arrivare alla progettazione della scheda miniaturizzata finale, che andrà opportunamente collocata sul dorso del guanto.
Dal terzo capitolo in poi ci si concentrerà sullo sviluppo del firmware per il microcontrollore, cominciando dalla configurazione dell'ambiente di sviluppo e dei vari tool software, per poi trattare la parte riguardante l'acquisizione dei sensori digitali e analogici.
Il quarto capitolo sarà dedicato alla connettività wireless e quindi, partendo dalle caratteristiche fondamentali del Bluetooth 5, saranno esaminati i vari ruoli dei dispositivi nella comunicazione, la struttura del buffer di trasmissione, ricezione e organizzazione dei dati.
Infine, nell'ultimo capitolo, verranno effettuate varie prove di trasmissione e ricezione dati in base al training richiesto, concludendo con la realizzazione del guanto dimostrativo, calibrazione dei sensori, modelli per l'analisi dei dati e loro visualizzazione grafica in tempo reale.

Capitolo 1

Stato dell'arte: Wired-Gloves

In generale, un *Wired-Glove* è un guanto sensorizzato ovvero un comune guanto al cui interno sono presenti sensori di vario tipo [1]. Sensori per la flessione delle dita, sensori per la forza da esse esercitata, ma anche sensori che rivelano la posizione della mano e come essa è orientata nello spazio, e quindi sensori come magnetometro, accelerometro e giroscopio.
L'impiego del *Wired-Glove* dipende dal campo di applicazione; sia esso un mezzo di interazione con un computer (HCI), sia esso un dispositivo che funge da interfaccia con la realtà virtuale, in ogni caso esso acquisisce varie grandezze di misura dalla mano e le trasmette a un software capace di elaborare le informazioni ricevute.
Qui di seguito verranno trattati esempi di *Wired-Glove* che sono stati impiegati in ambito medico, in particolare nel campo della valutazione funzionale e della riabilitazione motoria.

1.1 Esempi di *wired-glove*

I primi metodi impiegati per comprendere i complessi movimenti della mano erano per lo più di tipo meccanico, poiché prevedevano l'uso di goniometri per la misura della flessione delle dita o della rotazione del polso. Metodi come questo erano caratterizzati da procedimenti di misura molto lunghi e scarsa ripetibilità delle misure. Inoltre, considerando che la mano è costituita da 27 ossa con all'incirca 25 gradi di libertà di movimento, l'adoperare tali strumenti non garantiva certamente un'accuratezza elevata e, talvolta, era difficile, se non addirittura impossibile, andare a studiare particolari tipi di movimento.
Ecco dunque che l'impiego di uno strumento di tipo *glove-based* è di gran lunga più agevole sotto vari punti di vista:

- riduzione dei tempi di misura,
- incremento della ripetibilità delle misure,
- misura contemporanea di varie grandezze,
- strumento più maneggevole.

Un sistema di questo tipo è chiaramente ben più complesso, ma i suoi vantaggi sono ben evidenti. Considerando infatti l'ultimo punto, è importante notare che lo strumento di misura non deve (o comunque in maniera molto limitata) interferire con la misura stessa. Ciò significa che il guanto non deve compromettere i movimenti della mano poiché introdurrebbe un errore di tipo sistematico nelle varie misure. Ecco dunque che la realizzazione di un dispositivo di questo tipo prevede un'accurata scelta dei materiali adoperati, capaci sia di non opporsi in maniera eccessiva al naturale movimento della mano (soprattutto in pazienti con ridotta motilità) ma anche di ospitare sensori come quelli di flessione e di forza che vanno letteralmente cuciti all'interno del guanto stesso. In letteratura si trovano vari esempi di wired-glove che implementano questo tipo di misure e ciascuno di essi usa un metodo specifico.
Il primo guanto che si diffuse subito in tutto il mondo è quello che è stato messo sul mercato nel 1987 negli Stati Uniti, il *"Data Glove"*. [2] Esso impiegava sensori magnetici e ultrasuoni per tracciare il movimento della mano nello spazio. Successivamente, due anni dopo, la Nintendo sviluppò un altro guanto chiamato *"Power Glove"*, introducendo sensori ottici al fine di misurare la flessione delle articolazioni delle dita, in particolare quelle MCP (*Metacarpophalangeal*) e quelle PIP (*Proximal-interphalangeal*), rappresentate in Figura 1.1.

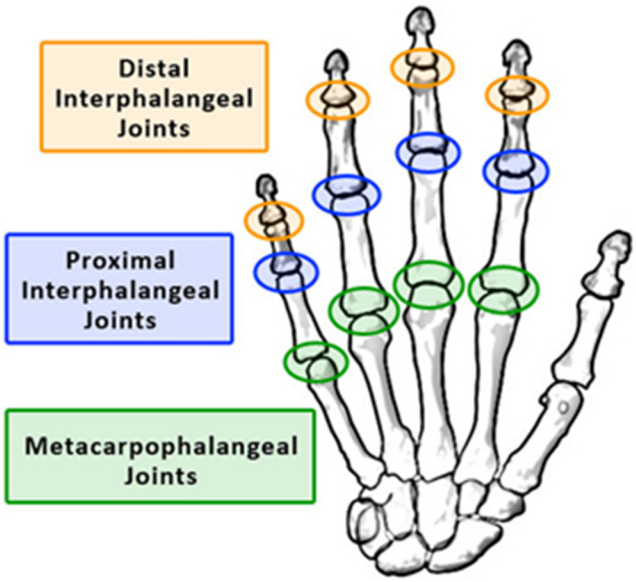

Figura 1.1 Articolazioni delle dita **[3]**

Altri modelli si susseguono, fino ad arrivare allo *Humanglove* prodotto dall'azienda Humanware in collaborazione con la Scuola Superiore Sant'Anna di Pisa [4]. L'impiego principale era quello di misurare gli angoli di flessione delle articolazioni delle dita (con 20 sensori che sfruttavano l'effetto Hall) e i movimenti di abduzione-adduzione tra le varie dita e del pollice da e verso le altre 4 dita. Quasi in contemporanea un altro modello era presente, il modello SIGMA *(Sheffield Instrumental Glove for Manual Assessment)* le cui misure erano gli angoli che riusciva a compiere la mano nei suoi vari movimenti [5, 6].
Il 2007 è il momento dello *Shadow Glove Monitor,* un dispositivo basato su otto sensori di tipo resistivo per la misura degli angoli di flessione delle dita, quindi impiegando un metodo completamente diverso rispetto ai precedenti [7]. A partire da questo guanto

vengono introdotte anche varie modalità di salvataggio dei dati acquisiti selezionabili via software:

- *Sample-and-Send mode:* i dati acquisiti dai sensori non vengono salvati in alcuna memoria interna ma sono direttamente inviati al computer,
- *Sample-and-Save mode:* caso opposto al precedente in cui i dati non vengono inviati al computer ma vengono salvati nella memoria interna per essere successivamente prelevati e analizzati,
- *Sample-and-Dump mode:* modalità di funzionamento intermedia alle due precedenti in quanto invia sia i dati prelevati dai sensori al PC in tempo reale, ma anche quelli presenti in memoria.

Successivamente vengono anche realizzati dei guanti non per l'analisi del movimento della mano ma guanti sperimentali che impiegano soluzioni innovative in modo da ridurre il movimento causato ad esempio dal Parkinson. Ad esempio nel 2009, un guanto basato sull'*harvesting* sfrutta tali oscillazioni della mano trasformando l'energia meccanica in energia elettrica. Mediante appositi attuatori piezoelettrici viene poi applicata una retroazione in modo da opporsi a queste oscillazioni. [8]
Tra i vari wired-glove di maggior successo, uno degno di nota è sicuramente il *Cyber Globe,* il cui primo modello, sviluppato nel *Virtual Space Exploration Laboratory* del Centro di Ricerca Applicata dell'Università di Stanford, risale al 1999 [9]. Esso è stato realizzato per il progetto "Talking Glove", ovvero un guanto che riconosca il linguaggio dei segni. Successivamente messo in commercio dalla *Virtual Technology* ha preso largo piede in ambiti come la realtà virtuale, videogiochi e telerobotica. Esso disponeva di diciotto sensori piezoelettrici per l'analisi dei vari movimenti delle dita, fino ad arrivare alla terza versione (Figura 1.2), *Cyber Glove III*, che monta 22 sensori di bending disposti in prossimità delle articolazioni DIP, MCP e PIP (Figura 1.1) e sul dorso della mano.

Figura 1.2 Cyber Glove III

Un altro guanto simile, realizzato nel 2010, ma con applicazione nell'ambito della valutazione funzionale della mano è il *Neuro-Assess Glove* (Figura 1.3).

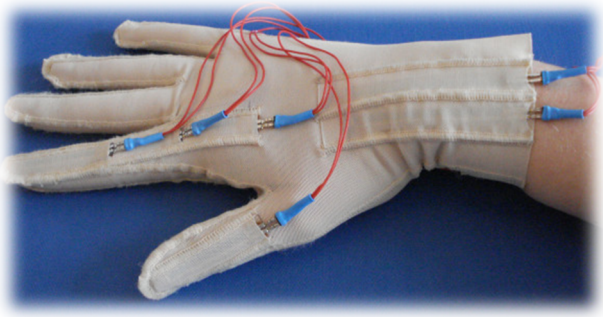

Figura 1.3 Neuro-Assess Glove [**10**]

Durante gli anni si susseguono sempre nuovi modelli, sempre più avnzati e sempre con nuovi stratagemmi per migliorare l'accuratezza delle misure. Un problema infatti comune a tutti i guanti è quello dei sensori di flessione che affinché forniscano una misura corretta dell'angolo di flessione devono essere ben allineati con l'articolazione, motivo per cui in molti casi occorre fare una nuova calibrazione prima dell'uso del guanto.
Di recente, a Dusseldorf in Germania, a novembre del 2017, si è tenuto il Wearable Technologies Show. Durante questo evento sono state presentate nuove tecnologie in ambito riabilitativo e, tra le tante, si annovera il *SEM Glove* (Figura 1.4) realizzato in Svezia da Bioservo Technologies, dove SEM sta per *Soft Extra Muscle.* Lo scopo di tale guanto è infatti quello di aumentare la presa (grip) e la forza alla mano in persone con ridotta motilità, e quindi è un guanto di ausilio che sfrutta tecnologie di tipo non invasivo. Il range di pressioni che ricopre va da 1 kPa a 500 kPa.

Figura 1.4 *SEM Glove*

1.2 Dalla concezione alla realizzazione di un *wired-glove*

Visti i modelli in letteratura e le usuali grandezze di interesse che si intende misurare, questa sezione affronta la realizzazione di un sistema-guanto a partire dalla scelta dei materiali fino ad arrivare all'architettura elettronica per poter acquisire i dati dai sensori.

1.2.1 Materiali ad hoc

Il materiale adoperato dai guanti precedentemente descritti è l'*elastam,* anche conosciuto come *Spandex,* un materiale fatto in fibra sintetica di poliuretano utile per elasticizzare i tessuti [11] e che trova largo utilizzo in questo genere di applicazione perché garantisce un'ottima aderenza con la pelle e quindi un migliore contatto tra la pelle e l'oggetto da afferrare (nel caso di particolari training riabilitativi).
Le continue ricerche nel campo dei materiali, nonché l'evoluzione delle tecnologie produttive di questi, hanno portato a inventare materiali chiamati *electro-textiles*, ovvero dei tessuti in grado di condurre e che quindi facilitano l'interconnessione tra sensori e PCB all'interno del guanto [12]. Le caratteristiche fondamentali che portano all'impiego di questo materiale sono la flessibilità, elasticità, la possibilità di essere lavato e appunto una buona conducibilità elettrica. In questo modo, dato che le parti conduttive sono integrate, si riduce notevolmente il rischio di attorcigliamento e rottura delle parti conduttive.
Un'altra idea molto innovativa è quella di adottare dei PCB flessibili, *flexible-PCB,* come ad esempio quello usato nel progetto *Tyndall* [13] in cui il dorso della mano viene ricoperto da un PCB flessibile (Figura 1.5) a 8 strati e quindi meno invasivo.
Si nota come il PCB stesso costituisce la struttura del guanto stesso, tanto che i sensori di flessione sono di tipo SMD.
Un'altra innovazione più recente è quella sviluppata dal *Neuroinformatics Group, Center of Excellence Cognitive Interaction Technology* (CITEC) [14]. Si tratta di un nuovo tipo di tessuto ancora più flessibile ed elastico che funge anche da sensore di pressione. I rigidi array di sensori tattili sono stati rimpiazzati da una fibra che emula la pelle umana e che è costituita da tante "celle tattili" integrate in un unico pezzo di tessuto. Il range di valori di pressione che è possibile misurare va da 1 kPa a 500 kP, che rappresenta l'intervallo medio di valori di pressione che esercita la mano durante le comuni azioni quotidiane.

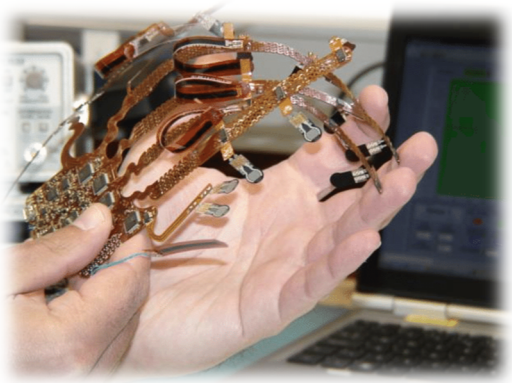

Figura 1.5 PCB flessibile

1.2.2 Tipi di sensori adoperati

Come già accennato, la scelta dei sensori è chiaramente strettamente connessa al tipo di grandezza che si intende misurare. Per misurare una determinata grandezza non esiste un singolo tipo di sensore. Qui si vedranno quelli comunemente impiegati nei wired-glove. Anche il numero e la loro posizione sul guanto è importante ai fini della

misura e un approccio teorico che viene in aiuto è quello proposto da D.J. Sturman e Zelter[1]. Per quanto concerne gli studi della mano in ambito riabilitativo la grandezza più importante è l'**angolo di flessione** di ciascun dito. I sensori di flessione comunemente impiegati possono essere distinti in 4 categorie:

- Sensori ricoperti da un film di materiale **piezoelettrico**: funzionano bene su guanti che usano l'electro-textile, offrono bassi consumi di potenza e una vasta possibilità di scelta perché ne esistono di tutte le dimensioni (vari fattori di forma).
- Sensori che sfruttano la **fibra ottica**: i punti di forza sono il peso ridotto, ridotta sensibilità alle interferenze elettromagnetiche (EMI), flessibilità, elevata sensibilità e affidabilità. Sono delle guide d'onda planari (2D) che vengono disposte sulle dita, ad una estremità si trova la sorgente luminosa (LED), mentre dall'altra il fotorivelatore (fotoresistore o fotodiodo). Nel momento in cui la fibra si piega, a seguito della flessione delle dita, al fotorivelatore giunge meno luce; quindi interpretando la quantità di luce che arriva è possibile risalire all'angolo di flessione.
- Sensori di tipo **resistivo**: sono caratterizzati da una un supporto in carbonio, il cui valore di resistenza varia in base all'angolo di flessione. Il grande vantaggio nell'impiego di questo tipo di sensori è il costo, notevolmente inferiore rispetto ai precedenti due. Se ne trovano di lunghezza variabile. Quelli solitamente usati sono quelli in poliestere laminato prodotti dalla Flexpoint (Figura 1.6).

Figura 1.6 Sensore di flessione Flexpoint

- Sensori ad **effetto Hall**: la tensione di uscita cambia al variare del campo magnetico applicato. Questo effetto è comunemente usato anche in sensori di velocità, prossimità, posizione e corrente.

Nel guanto qui descritto verranno usati sensori di tipo resistivo, ma in letteratura si trovano guanti che, per misurare i movimenti di adduzione e abduzione delle dita, sfruttano sensori ad effetto Hall (lo *Humanglove*) o sensori ottici nel *5DT Data Glove.*

[1] D.J. Sturman and D. Zelter, «A survey of glove input», IEEE Comput. Graph. Appl., Vol 14, n.1, pp. 30-39, Jan 1994

Altra grandezza di fondamentale importanza è la **forza** esercitata dalle dita nel prendere un oggetto, ma anche quella esercitata da un dito su un altro. I sensori solitamente usati sono:

- FSR *(Force Sensing **Resistor**)*: variano la loro resistenza in risposta alla forza applicata, per l'esattezza maggiore è la forza esercitata e minore è la resistenza. Tra i più usati sul mercato vi si trovano quelli della Tekscan (Figura 1.7).

Figura 1.7 Tekscan FlexiForce, sensore FSR

- Sensori di pressione **piezoelettrici**: si viene a generare una differenza di potenziale a fronte di una pressione. Per applicazioni come quelle sistema-guanto vengono usati sensori con un tessuto piezoresistivo elastico costituito per il 72% da nylon e la restante parte da spandex.
- Sensori di pressione di tipo **capacitivo**: per misure che vanno da 4,35 kg a 22,7 kg.
- Sensori **EMG** (*Electromyography*): impiegati per la misura dei segnali elettrici scaturiti dalla contrazione dell'avambraccio e per stimare la forza applicata dalle dita.
- Sensori **MARG** (*Magnetic Angular Rate Gravity*): è un sistema di sensori che determina un modello cinematico dei movimenti della mano combinando sensori magnetici, di velocità angolare e gravimetri. Si sfruttano ad esempio l'accelerometro e il giroscopio per rilevare quando il dito tocca un oggetto; in tal caso infatti si ha una grande variazione di velocità in poco tempo.

Analizzare i movimenti della mano nello spazio è anche importante perché permette di comprendere e quantificare i progressi della riabilitazione. Tali movimenti vengono comparati con quelli di una persona con normali funzionalità motorie al fine di ottenere informazioni riguardo la stabilità della mano nel compiere particolari traiettorie. Esistono vari modi per studiare questo genere di dati e ognuno di essi sfrutta particolari tecnologie. Si trovano ad esempio applicazione che fanno uso di una *Motion Processing Unit (MPU)* ovvero un'unità costituita da un accelerometro 3D e un giroscopio 3D. In questo modo si riesce a misurare l'accelerazione lineare e angolare. Simile alla MPU c'è l'IMU *(Inertial Measurement Unit)* che è quella che è stata usata nel guanto in esame. Essa viene montata sul dorso della mano e quindi permette di studiarne i movimenti di rollio (roll) rispetto all'asse x, beccheggio (pitch) rispetto all'asse y e imbardata (yaw) rispetto all'asse z (Figura 1.8 e Figura 1.9).
La scheda monta anche il magnetometro triassiale, pertanto si hanno 9 misure totali (3 per ogni asse). È importante precisare che l'utilizzo di questi dispositivi presuppone una fase di calibrazione iniziale per stabilire il sistema di riferimento iniziale e rimuovere eventuali offset; motivo per cui molti guanti dispongono di un sistema di posizionamento iniziale per gli avambracci, in modo da garantire sempre le stesse condizioni

di misura iniziali. Esistono tuttavia anche altri metodi di calibrazione, come ad esempio quelli implementati nei moderni smartphone per la calibrazione della bussola.

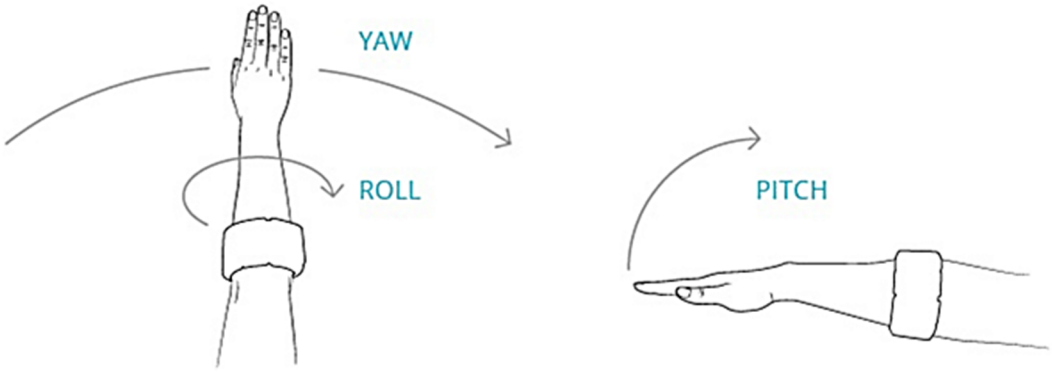

Figura 1.8 Movimenti di roll, pitch e yaw della mano

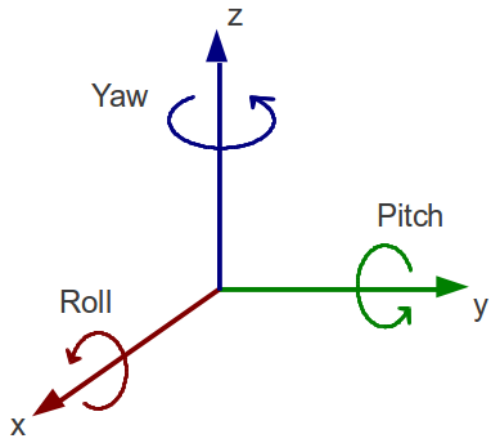

Figura 1.9 Movimenti di roll, pitch e yar con riferimenti cartesiani

Un altro metodo di misura, sempre in riferimento alle traiettorie che la mano può compiere, prevede l'impiego di accelerometri e sensori di flessione i cui dati vengono processati dal cosiddetto filtro di Kalman. Tale algoritmo implementa un filtro ricorsivo per la valutazione di un sistema dinamico a partire da una serie di misure soggette a rumore [15]. Esso è usato in particolare quando le variabili che si intendono misurare possono essere stimate soltanto indirettamente a partire dalla misura di altre; quindi la misura finale è ottenuta dalla combinazione di dati provenienti da più sensori.
Infine altre tecniche più evolute prevedono l'uso di una IMMS *(Inertial and Magnetic Measurement System)* sfruttando l'algoritmo di Kalman Extended (EKF).

1.2.3 Collocamento dei sensori

Come già accennato, la posizione dei sensori per ottenere una misura corretta è di fondamentale importanza.
Molti dei guanti precedentemente descritti hanno delle inserzioni, come delle tasche, in cui andare a collocare i vari sensori. Quindi ad esempio, in prossimità delle articolazioni delle dita, vi si trovano delle fessure in cui inserire i sensori di flessione. In questo modo i sensori possono scorrere sulle dita per adattarsi ai vari movimenti.

Altri guanti invece non adoperano questo escamotage; esistono soluzioni in cui ad esempio i sensori di forza vengono realizzati dal guanto stesso che dispone di molte celle tattili integrate nel tessuto.
Il sistema realizzato in [16] impiega delle strisce di PCB flessibile sulle dita e sul dorso della mano e ogni striscia è dotata di tre accelerometri-giroscopi 3D, mentre sulla punta di ogni dito è presente un magnetometro.

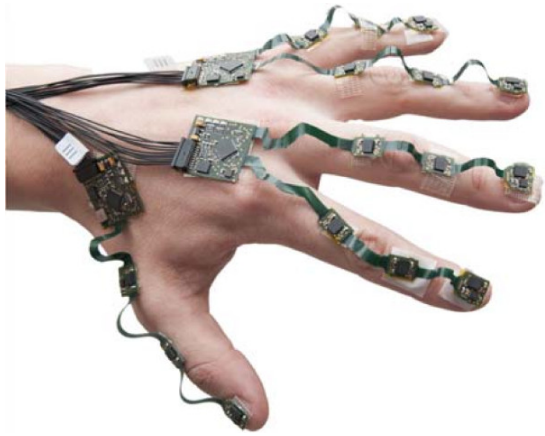

Figura 1.10 Flexible PCB Glove [**16**]

1.2.4 Test di validazione e calibrazione dei sensori

Una volta che è stato definito l'insieme di sensori e la loro collocazione, il guanto sensorizzato non è subito pronto all'uso in quanto bisogna fare alcuni tipi di test che garantiscano la ripetibilità delle misure e il grado di accuratezza desiderato. Il problema è che non esistono delle procedure standard che definiscono dei test di validazione efficaci.
Sicuramente un fattore che influenza la correttezza dei dati acquisiti è il corretto posizionamento dei sensori; ad esempio per quanto riguarda i sensori di flessione bisogna assicurarsi che la flessione avvenga in modo omogeneo. Alcuni studi hanno provato infatti che basta un disallineamento longitudinale di pochi millimetri, rispetto all'articolazione del dito, per far sì che la misura venga considerevolmente alterata, ovvero va a cambiare la forma della curva che mette in relazione la tensione prodotta dal sensore e l'angolo di flessione. Questo è un esempio di scarsa ripetibilità delle misure. Le prestazioni sono di certo influenzate dalla *zero-position*, cioè la posizione iniziale dei sensori prima di eseguire le varie acquisizioni, condizione chiaramente strettamente legata alla posizione di partenza della mano, ma anche a come il guanto stesso si adatta a essa. Altro parametro da non sottovalutare è che non tutte le mani sono uguali e di conseguenza la misura che si va ad effettuare cambia da persona a persona, andando ad inficiare la ripetibilità.
Tutti questi fattori presuppongono che venga eseguita una fase di calibrazione iniziale di tutti i sensori. Questa fase può essere più o meno lunga, a seconda dei sensori e di come viene fatta; talvolta anche fastidiosa perché richiede al paziente di assumere determinate posizioni con la mano e di mantenerla per un certo tempo. Considerando ad esempio i sensori di flessione, alcuni tipi di guanto, come ad esempio quello trattato in [10], prevedono di fare una misura in condizioni di riposo, ovvero mano completamente aperta quindi flessione nulla, e un'altra con la massima flessione, cioè pugno

chiuso. Questo tipo di calibrazione (di 14 sensori) richiedeva all'incirca 10 secondi. Al contrario, nel modello descritto in [7], ci impiegava ben 8 minuti poiché i sensori impiegati erano di tipo non-lineare. È da precisare che queste calibrazioni iniziali vanno effettuate per ogni paziente, per ogni nuova misura.
Altri tipi di calibrazione effettuano varie letture; facendo sempre riferimento ai sensori di flessione, è possibile andare a misurare i valori di uscite per diversi angoli di flessione, quindi ad esempio a 20°, 40°, 60° e 90°, come quanto fatto in [17]. Successivamente, mediante Matlab, le varie letture sono state interpolate in modo da trovare la relazione non lineare e generare una *look-up table* ad intervalli di 0,1. Un procedimento molto simile è stato effettuato con i sensori di forza, usando varie masse note: 10 g, 50 g, 100 g, 150 g, 200 g, 250 g, 300 g e 350 g. Andando ogni volta a misurare le corrispondenti tensioni di uscita prodotte a fronte dei vari carichi, si è ottenuta, mediante interpolazione, una caratteristica pressoché lineare.
Come detto, queste misure sono strettamente dipendenti dalla mano del paziente e andare ad effettuare determinate misure ad angoli precisi può essere poco pratico e poco gestibile. Motivo per cui, nel Cyber Glove III, viene fruttato un *dummy finger,* cioè un dito finto con il quale vengono calibrati automaticamente i sensori di flessione.

1.2.5 Condizionamento dei sensori e interfaccia utente

Una volta che sono state scelte le grandezze di interesse, quindi i sensori utili alla misura, nonché la loro posizione sulla mano, è necessario procedere ai circuiti di condizionamento che convertono la grandezza di interesse in corrispondente valore di tensione, in modo tale da poterlo leggere, processare e memorizzare.
Ogni sensore ha il proprio circuito di condizionamento dedicato e opportunamente progettato tenendo conto delle varie specifiche progettuali e delle interfacce elettroniche adoperate per la loro lettura, come ad esempio gli amplificatori utili al condizionamento, la tensione di alimentazione disponibile o il convertitore analogico-digitale presente all'interno del microcontrollore adoperato.
Chiaramente, essendoci più sensori, bisognerà adottare delle strategie tali da poterli acquisire tutti quanti. Il modo in cui viene fatto dipende da tanti fattori, ad esempio il numero di canali di cui dispone l'ADC adoperato. Un altro metodo spesso impiegato è quello di inserire tra i sensori e il convertitore un multiplexer (Figura 1.11).

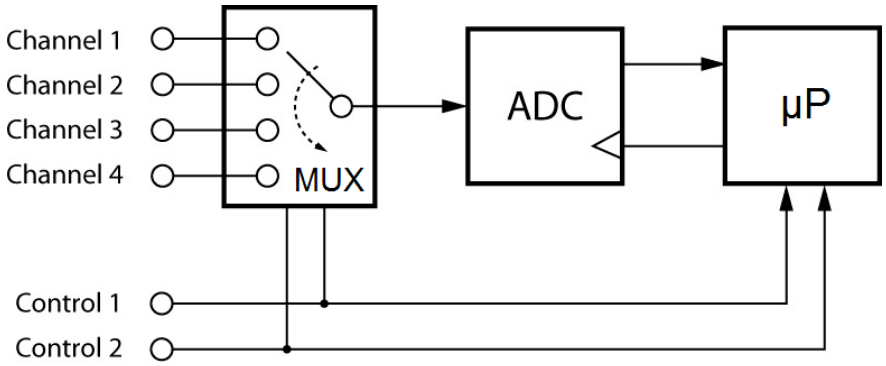

Figura 1.11 Canali, multiplexer, ADC e microcontrollore

In tal caso, pilotando opportunamente il multiplexer tramite il microcontrollore (indicato in Figura 1.11 con "µP"), sarà possibile selezionare di volta in volta il canale da

acquisire, ovvero il sensore che sarà direttamente collegato all'ADC e che pertanto verrà campionato. Tale soluzione però comporta che ogni sensore viene acquisito ad una frequenza minore; se ad esempio i sensori sono N e la frequenza di campionamento dell'ADC è f_c , allora la massima frequenza con cui può essere campionato il sensore i-esimo sarà $f_i = f_c/\mathrm{N}$.
Una volta realizzate tutte le interfacce elettroniche per il condizionamento dei sensori, per le varie comunicazione a bordo della scheda, nonché l'invio dei dati, il passo finale è la realizzazione dell'interfaccia utente, ovvero il software con cui si interfaccerà il medico per poter eseguire i vari training e quindi analizzare i dati acquisiti.
Requisito fondamentale di un'interfaccia grafica è l'essere *user-friendly,* cioè di facile utilizzo. Chi userà il sistema guanto sensorizzato non deve preoccuparsi minimamente di tutto ciò che avviene al suo interno se non dei risultati finali che esso fornisce. Se esso costituisce il prodotto finito per l'ingegnere che l'ha progettato, al tempo stesso esso è la materia prima per il medico o chi di dovere, cioè il primo strumento per verificare l'integrità funzionale della mano e i progressi che il paziente effettua.
Anche tutte le fasi di calibrazione iniziale dovranno essere il più semplice possibile mediante l'interfaccia software.
Tra le varie soluzioni presenti in letteratura per lo sviluppo di un'apposita interfaccia grafica si trova *LabView;* esso è già però uno strumento piuttosto avanzato per strumenti di misura e l'analisi dei dati in generale. Un'altra possibilità è l'utilizzo di uno script Matlab che effettua tutti i vari calcoli e stampa a vide le informazioni utili, ma anch'esso non è la migliore soluzione.
L'approccio migliore da intraprendere è, senza dubbio, l'implementazione di un software proprietario, ovvero un software completamente dedicato al guanto (non un software generico per l'analisi e la lavorazione dei dati). Quindi scrivere un software da zero (in linguaggio C, o C++, o C#) capace di connettersi da remoto, impostare il training da seguire, lanciare i sensori, acquisire i dati, processarli, visualizzare a video modelli 3D, informazioni tecniche e soprattutto informazioni utili a chi usa il guanto per l'interpretazione dei dati.

1.2.6 Power Management

Anche se è stato detto che il passo finale nello sviluppo di un wired-glove è l'interfaccia utente, viene posta qui attenzione alla gestione dell'energia, soprattutto nel caso della scheda qui presa in esame che funzionerà in modalità wireless e quindi con una batteria.
La *Power Management Unit (PMU)* è quell'unità in grado di gestire tutta l'alimentazione della scheda, quindi l'alimentazione dei vari dispositivi in essa presenti, ma anche la gestione della carica della batteria, in modo da fornire indicazioni sul suo stato di carica, nonché provvedere ad un circuito che consente di ricaricarla.
Le batterie che è possibile usare sono svariate. In precedenza venivano usate batterie stilo AA; tuttavia i continui sviluppi tecnologici in questo ambito sono riusciti a produrre batterie al litio o agli ioni di litio di tipo *coin* (Figura 1.12), cioè batterie ricaricabili al litio a forma di gettone, unendo quindi la praticità in termini di ingombro alla funzionalità in quanto possono essere anche ricaricate.
Gli ultimi guanti infatti implementano un circuito di ricarica con porta micro-USB per poter sfruttare un comune caricatore per smartphone.

Figura 1.12 Batteria a gettone

Di recente si trovano anche guanti che implementano tecniche innovative per il recupero dell'energia *(Energy Harvesting)* a partire dai movimenti della mano, ovvero tecniche che a partire dall'energia cinetica producono energia elettrica da immagazzinare nella batteria. Ad esempio esiste un materiale composito in fibra piezoelettrica (PFC) che realizza proprio tale funzione. Chiaramente, nello scegliere il materiale bisogna tenere sempre conto che non deve essere di intralcio ai movimenti della mano. Il PFC è un buon compromesso in quanto unisce le capacità piezoelettriche, seppur minori rispetto ai materiali di tipo ceramico, a quelle di flessibilità.

Capitolo 2

Dal prototipo *"Evaluation"* alla scheda miniaturizzata

Il fine ultimo del guanto, nonché le varie caratteristiche tecniche, sono strettamente connessi alle specifiche di partenza per la realizzazione del PCB che implementa tutte le funzionalità.
I vari training, e quindi i dati che si vogliono monitorare, sono stati definiti dal gruppo di ricerca SAMBA *(SpAtial, Motor & Bodily Awareness)* del Dipartimento di Psicologia della Neuroscienza dell'Università di Torino. Tale gruppo di ricerca si occupa di studiare le basi anatomo-funzionali delle funzioni cognitive umane e quindi aspetti come la consapevolezza spaziale, la consapevolezza motoria e la consapevolezza corporea [18]. Gli allenamenti prevedono esercizi giornalieri per l'arto superiore, come ad esempio esercizi di velocità, di precisione e di forza. A tal proposito sono stati individuati i seguenti training:

- *Pinching:* consiste nel toccare la punta del pollice con la punta delle altre quattro dita con una particolare sequenza predefinita,
- *Grasping:* afferrare un oggetto e rilasciarlo,
- *Wrist training:* allenamento per il polso, cioè l'esecuzione di rotazioni, flessioni e estensioni per migliorare le articolazioni intercarpali,
- *Reaching:* afferrare un oggetto nel minor tempo possibile,
- *Following:* dopo un segnale acustico che indica l'inizio dell'esercizio, il paziente deve seguire con un dito un punto luminoso su uno schermo, nel modo più preciso possibile.

Come detto, ogni training consiste nel monitorare opportuni parametri, motivo per il quale è stato necessario definire innanzitutto le grandezze fisiche da misurare a partire dalle quali si riescono ad ottenere i parametri desiderati; definite le grandezze fisiche il passo successivo è stato quello di stabilire quali sensori impiegare. A tal proposito si riporta la tabella riassuntiva (Tabella 2.1) che indica, per ciascun tipo di training definito, le varie grandezze da misurare e i sensori impiegati a tal proposito.
Si nota come nel caso del *pinching* sono stati impiegati dei sensori di forza sulle dita per la misura del tempo di esecuzione e la correttezza della sequenza da seguire. Per

tempo di esecuzione si intende l'intervallo di tempo che intercorre tra il segnale acustico (emesso da un'opportuna interfaccia software) e il contatto tra la punta del pollice con quella di un altro dito.

Tabella 2.1 Training e relative grandezze di interesse

TRAINING	GRANDEZZA DA MISURARE	SENSORE
Pinching	Tempo di esecuzione Errori di sequenza	Forza
Grasping	Forza delle dita	Forza
	Angolo d'apertura delle dita	Flessione
Wrist	Rotazione del polso	Accelerometro a 3 assi Giroscopio a 3 assi Magnetometro
	Flessione del polso	
Reaching	Tempo di esecuzione	Forza
Following a spot on a screen	Tempo di reazione	Accelerometro
	Tempo di esecuzione	
Physiological measurements	Pulsazione e battito cardiaco	Pulsazione
	Temperatura	Temperatura
	Sudorazione	GSR

Il *grasping* fa uso dei sensori di forza per determinare la forza e la pressione esercitata dalle dita nell'afferrare un oggetto, mentre l'angolo delle articolazioni PIP viene determinato con i sensori di flessione.
I training per il movimento del polso usano sensori come accelerometro, giroscopio e magnetometro triassiali. In particolare, con il giroscopio si quantifica la variazione dell'orientazione spaziale mediante opportuna integrazione delle velocità angolari misurate; mentre l'accelerometro e il magnetometro forniscono informazioni sull'orientazione attraverso la proiezione dell'accelerazione di gravità e del campo magnetico sulle 3 componenti si uno specifico sistema di riferimento. Le informazioni prese singolarmente, così come sono in uscita dai sensori, non garantiscono alcun risultato in termini di precisione e affidabilità, in quanto l'integrazione numerica può portare a errori sui dati. È necessario pertanto combinare i risultati dei 3 sensori per fornire una stima sulla posizione assoluta.
Nel training *following* il tempo di reazione indica il tempo che intercorre tra l'inizio del movimento del punto luminoso e l'inizio del movimento della mano; il tempo di esecuzione è quello tra l'inizio e la fine del movimento della mano. Queste due grandezze possono essere rilevate con un accelerometro che indica l'inizio e la fine del movimento.
Oltre queste misure, vengono fatte altre di carattere fisiologico e quindi il monitoraggio della temperatura e del battito cardiaco. Queste grandezze sono utili per degli studi sulla propriocezione (capacità di percepire e riconoscere la posizione del proprio corpo nello spazio e lo stato di contrazione dei propri muscoli, senza il supporto della vista [19]); ad esempio al paziente si finge di colpire la propria mano con un coltello e in tal caso, oltre a quantificare i tempi di reazione nel muovere la mano per sfuggire al pericolo, è interessante anche determinare come esso reagisce dal punto di vista emotivo, quindi parametri come la sudorazione fredda.

Il battito cardiaco rappresenta i movimenti di contrazione ed espansione del cuore, mentre la frequenza cardiaca è il numero di battiti cardiaci al minuto. Entrambe queste grandezze si ottengono dallo stesso sensore.
La temperatura può essere un buon indicatore per la propriocezione della mano. La sudorazione, anch'esso parametro da tenere in considerazione per le misure di tipo propriocettivo, può essere monitorata mediante il GSR *(Galvanic Skin Response).*

2.1 *Evaluation board*

A partire dalla definizione delle grandezze di interesse è stata progettata una scheda come prototipo di partenza, l'*Evaluation board* (Figura 2.1). Tale scheda, realizzata come attività di tesi da L. Mastrototaro, rappresenta il punto di partenza di tutto lo sviluppo del sistema wired-glove, sia a livello hardware che firmware. Le sue dimensioni sono volutamente esagerate, 190 mm x 127 mm x 1 mm, e in essa sono presenti parti utili soltanto ad un rapido sviluppo e valutazione della scheda stessa. Ad esempio, sono presenti vari interruttori per accendere o spegnere una parte del circuito e relativo LED per indicarne lo stato.

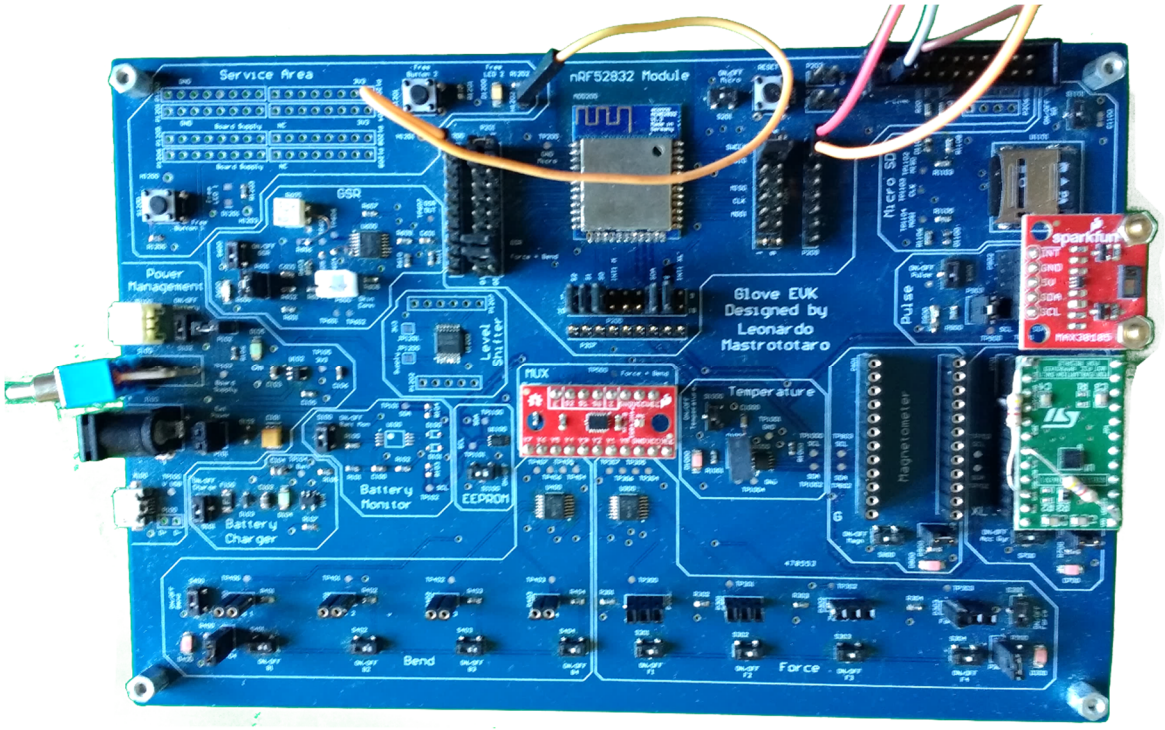

Figura 2.1 Evaluation Board

Come si vede in essa sono presenti varie aree funzionali:

- *Service Area*: in essa sono presenti vari punti di connessione come ad esempio le varie tensioni di alimentazione (3.3 V), punti per poter fare eventuali rapidi test, due bottoni e LED da usare a piacimento per eventuali funzioni di debug, essendo connessi direttamente al microcontrollore;
- *Power Management:* è la parte che fornisce la tensione di alimentazione a tutta la scheda e quindi a tutti gli integrati e sensori. Essendo una scheda di valutazione, si è pensato di installare vari connettori da poter usare per alimentare la

scheda: un connettore micro USB di tipo B, un connettore jack DC (foro di diametro 5,5 mm con un polo centrale di 2,1 mm, è il classico connettore uscente dai trasformatori collegabili ad una presa da parete), un header maschio a 2 pin per connettere una generica tensione di alimentazione esterna (Figura 2.2: da sinistra verso destra: micro-USB, jack barrel DC, connettor per batteria, header maschio a 2 pin);

Figura 2.2 Connettori presenti nella Power Management Area

- *Storage Area*: sono presenti due tipi di memorie, una EEPROM e una mini SD
- *Microcontrollore*: il cuore della scheda è connesso ai sensori analogici (mediante un multiplexer intermedio), ai sensori digitali (accelerometro, giroscopio, magnetometro, termometro, pulse-oximeter) tramite I2C, alle 2 memorie, alla Service Area. Esso contiene anche l'antenna Bluetooth;
- *Sensori:* 8 sensori analogici, di cui 4 di forza e 4 di flessione per le 4 dita escluso il pollice, e i sensori digitali già citati. Nella scheda è possibile connettere i sensori analogici in modo molto semplice in quanto sono presenti degli appositi header; sul guanto la connessione tra i sensori (situati sulle dita) e la scheda verrà fatta mediante i cosiddetti Pogo-Pin.

Di seguito si fornirà una rapida descrizione delle aree fondamentali e degli accorgimenti che sono stati adottati per essere presi poi in considerazione nello sviluppo della scheda miniaturizzata.

2.1.1 Power Management Area

Di tutti i connettori già presentati, chiaramente, quello di cui si farà uso successivamente è quello per la batteria. Tutti i dispositivi presenti sulla scheda, compresi i sensori analogici, sono alimentati a 3,3 V. A tal proposito è stato adoperato un regolatore LDO (*Low Drop Out voltage regulator*) che presenta una caduta tipica ai suoi capi di 250 mV.
Nella Power Management Area sono anche presenti due sezioni per la gestione di un'eventuale batteria: un circuito per la ricarica con annesso integrato e un circuito per il monitoraggio della carica e relativo integrato che misura la tensione, la corrente e la temperatura e tiene traccia in un log che salva sulla EEPROM.

2.1.2 Storage Area

Lo scopo finale del guanto è quello di effettuare le misure precedentemente descritte ma anche tenere traccia delle stesse. Per questo motivo la comunicazione Bluetooth dovrà assicurare la trasmissione completa dei dati verso il PC, che si occuperà di tenere traccia delle misure nonché dell'elaborazione dei dati ricevuti. In questa fase potrebbe

succedere, qualora il quantitativo di dati da inviare sia eccessivo, che il buffer in trasmissione si riempia con conseguente incapacità di inviare tutti i dati e quindi la perdita degli stessi. Per far fronte a questa evenienza sono state installate due memorie: una EEPROM e una micro-SD. Chiaramente sarà il microcontrollore che dovrà eseguire una scrittura in memoria.

- Micro-SD: è la classica scheda di memoria che viene inserita negli smartphone. Comunica con il microcontrollore mediante protocollo SPI;
- EEPROM *(Electrically Erasable Programmable Read-Only Memory)*: memoria di tipo non volatile per immagazzinare una quantità di dati non eccessivamente grande. Comunica con il microcontrollore mediante protocollo I2C.

2.1.3 Force Sensing Area

È l'area in cui è presente tutto il circuito di condizionamento e i connettori per i quattro sensori di forza. Si riporta lo schematico generale in Figura 2.4. Come si nota ogni sensore ha il proprio circuito di condizionamento. I 4 op-amp si trovano nello stesso integrato e l'uscita amplificata va a finire in ingresso al multiplexer e quindi in ingresso al convertitore analogico-digitale. La relazione che intercorre tra la tensione in ingresso all'ADC e la forza applicata è la seguente:

$$V_{ADC} = 3{,}3\frac{31{,}6}{2{,}36 \cdot 10^7 \cdot m^{-2.021} + 31{,}6} \quad [V] \tag{2.1}$$

Nella (2.1) m rappresenta la massa espressa in grammi, cioè a quanti grammi corrisponde la forza esercitata da un determinato dito. In Figura 2.3 se ne riporta l'andamento.

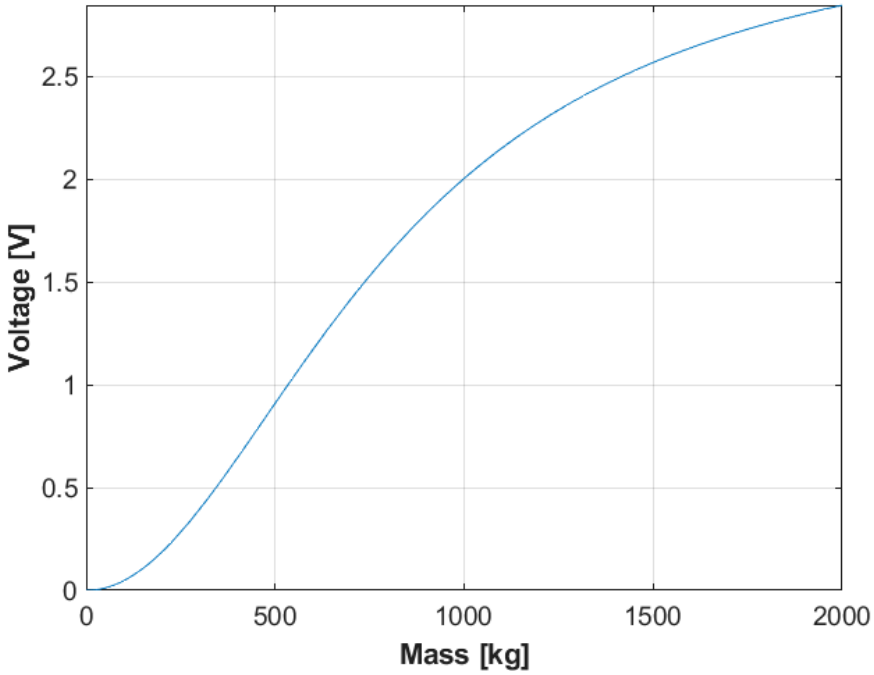

Figura 2.3 Andamento tensione-massa applicata

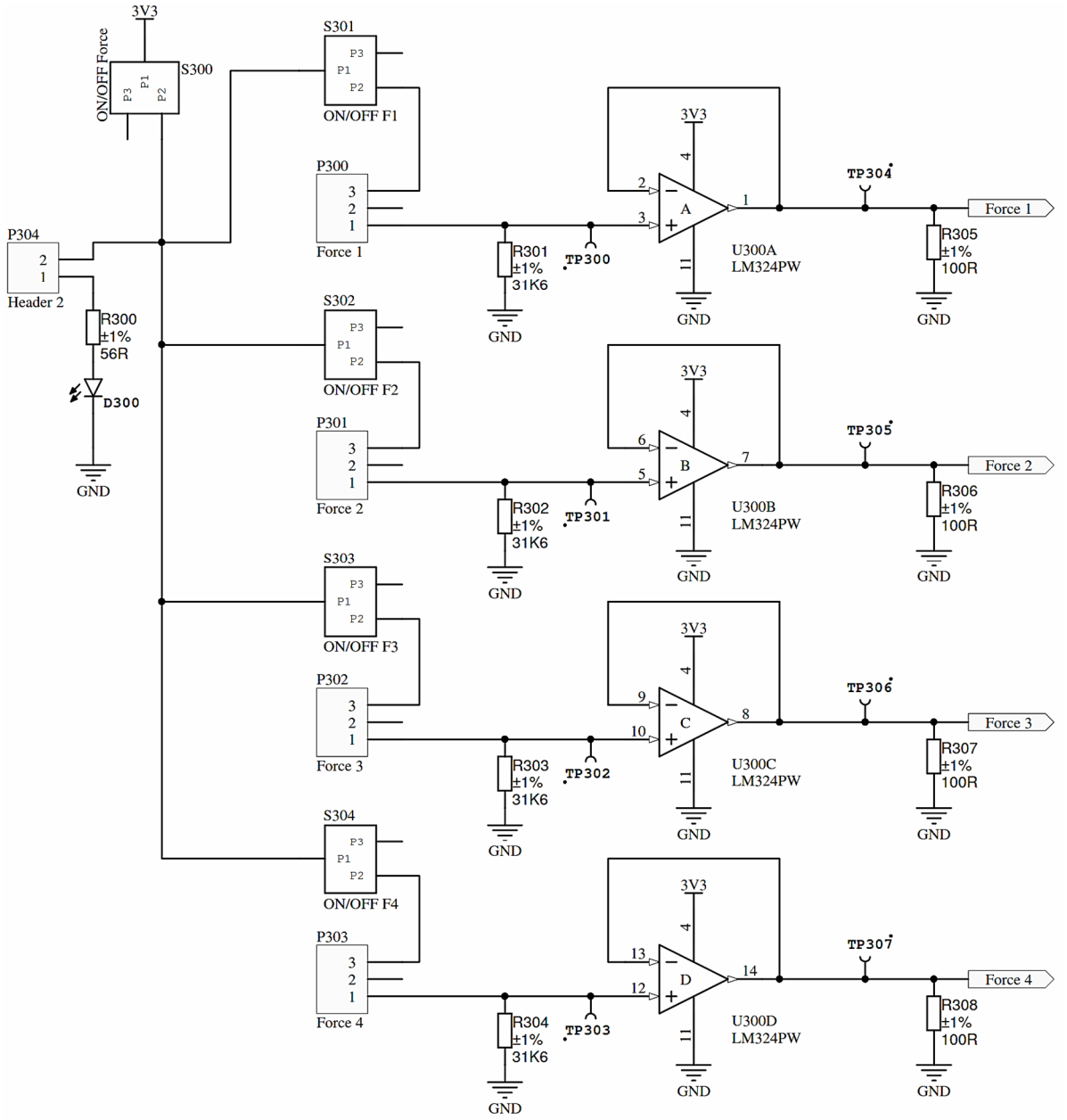

Figura 2.4 Force Sensing Area

La formula corrisponde al partitore di tensione adottato nel condizionamento (Figura 2.4), scelto in modo da avere una tensione di alimentazione unipolare (3.3 V). Si è partiti con la caratterizzazione del sensore di forza scelto, ovvero il sensore "FlexiForce A301 25 lbs" (Figura 1.7); si tratta di un sensore di forza per massimo 25 libbre, cioè circa 11,3 kg, anche se qui serve al massimo usare 2 kg. Una volta caratterizzato il sensore, cioè trovati i valori di resistenza a fronte di alcuni pesi (espressi in grammi) applicati, si è determinata la curva interpolatrice che descrive l'andamento della resistenza del sensore in funzione della massa applicata. Successivamente, per determinare il valore ottimale dell'altra resistenza presente nel partitore è stato studiato l'andamento della V_{ADC} in funzione di questa. Il valore ottimale di resistenza che garantisce un guadagno pressoché costante lungo tutto l'intervallo delle possibili forze applicabili (al massimo 2000 g) e che consente di sfruttare al meglio tutta la dinamica di ingresso del convertitore analogico-digitale (al massimo 3,3 V) è di 31,6 kΩ.

A partire dalla conoscenza di come funziona la conversione di un convertitore analogico-digitale, quindi a partire dalla conoscenza di come far corrispondere il valore numerico ottenuto al valore di tensione corrispondente, è possibile ricavare il valore di forza (o meglio di massa) semplicemente invertendo la relazione (2.1). In particolare, si nota come al crescere della forza applicata, aumenta anche la tensione di ingresso all'ADC. Per avere questo tipo di comportamento, dato che il valore di resistenza del sensore diminuisce al crescere della forza, è stato adottato il circuito in Figura 2.5.

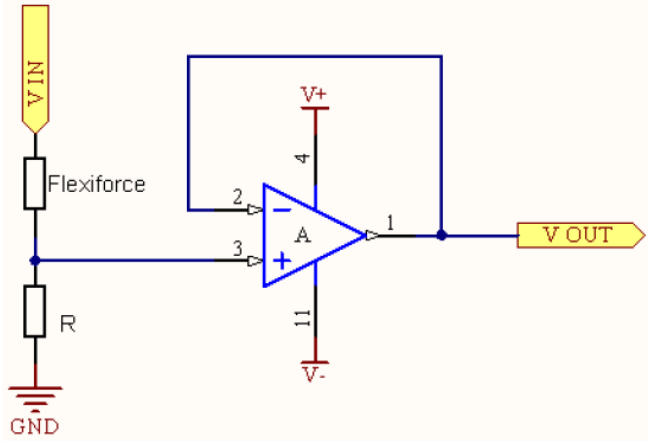

Figura 2.5 Condizionamento per i sensori di forza [**20**]

2.1.4 Bend Sensing Area

Un discorso del tutto analogo è stato affrontato nel progettare l'area dei sensori di flessione, di cui si riporta lo schematico generale in Figura 2.6.
Caratterizzare i sensori di flessione è più difficile dei sensori di forza perché la loro resistenza dipende da due parametri, l'angolo di flessione e la curvatura del sensore attorno al punto di flessione, motivo per il quale non è stata fatta alcuna caratterizzazione in quanto sarebbe stata poco accurata (bisogna garantire angoli e deflessioni angolari costanti).
Tuttavia, nei sensori di flessione adoperati (Flexpoint Rb-Flx-03, ovvero dei sensori di flessione da 3 pollici unidirezionali) la resistenza aumenta all'aumentare della flessione, motivo per cui è stato adottato il circuito di condizionamento in Figura 2.7.
La tensione in ingresso all'ADC aumenta all'aumentare della flessione secondo la relazione:

$$V_{OUT} = V_{ADC} = 3{,}3\frac{R_{SENS}}{R_{SENS} + R} \quad [V] \tag{2.2}$$

R_{SENS} è la resistenza del sensore (il valore nominale misurato quando il sensore è a riposo varia tra i 25 e i 30 kΩ) e varia al variare della flessione; mentre il valore di resistenza R è stato opportunamente scelto di 3 MΩ effettuando gli stessi ragionamenti esposti prima e tenendo conto che il datasheet del sensore consiglia come intervallo di valori da 10 kΩ a 10 MΩ.
I sensori di flessione vengono collocati sull'articolazione PIP e i possibili angoli vanno pertanto da 0° (dito disteso) a 90° (l'articolazione forma un angolo retto).

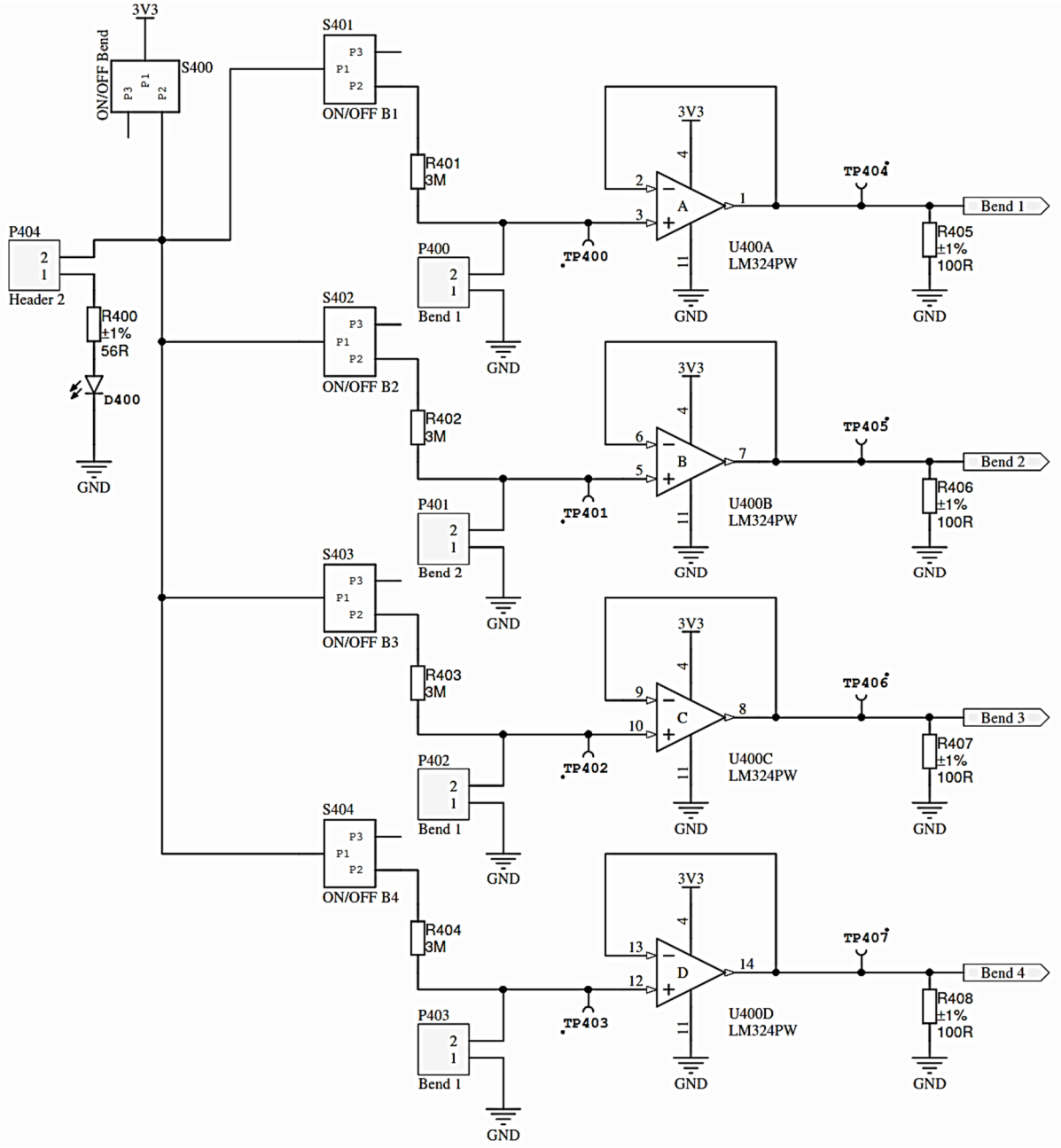

Figura 2.6 Bend Sensing Area

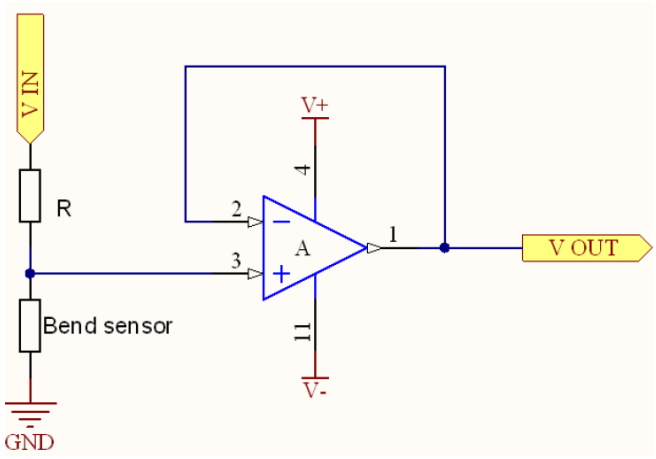

Figura 2.7 Condizionamento per i sensori di flessione

2.1.5 Sensore GSR

Il sensore GSR è il terzo e ultimo tipo di sensore analogico presente sulla scheda. Il nome del sensore stesso indica che il tipo di misura che si va ad effettuare è quella risposta galvanica della pelle, anche conosciuta come "Attività Elettrodermica" (EDA). Essa consiste nell'effettuare misurazioni continue delle caratteristiche elettriche della pelle, come ad esempio variazioni della conduttanza a seguito della sudorazione del corpo umano [21]. È infatti risaputo che la resistenza cutanea dipende dallo stato delle ghiandole sudoripare cutanee e l'attività di queste ultime varia i base alla risposta del sistema nervoso. Quindi la misura della conduttanza della pelle può essere un indice della risposta del Sistema Nervoso (Simpatico), direttamente coinvolto nella regolazione del comportamento emotivo umano. È possibile infatti rilevare variazioni notevoli della conduttanza della pelle a fronte di variazioni emotive ad esempio a seguito di stress, stanchezza e coinvolgimento.
Recenti attività di ricerca hanno ormai integrato tale tipo di sensore all'interno di applicazioni medicali, o nell'ambito dell'*healthcare* in generale, come ad esempio bracciali, orologi e guanti sensorizzati come quelli qui trattati, non ultimo anche l'ambito delle neuroscienze. Il sensore GSR è costituito da due elettrodi che misurano la variazione di corrente (a basso voltaggio). Risalire dai valori misurati allo stato emotivo non è ragionevole in quanto le misure difficilmente sono ripetibili, dato che lo stato emotivo è qualcosa di costantemente variabile e la conduttanza della pelle non è soggetta a controllo umano.
Il circuito di condizionamento adottato per il sensore GSR scelto è quello mostrato in Figura 2.8.

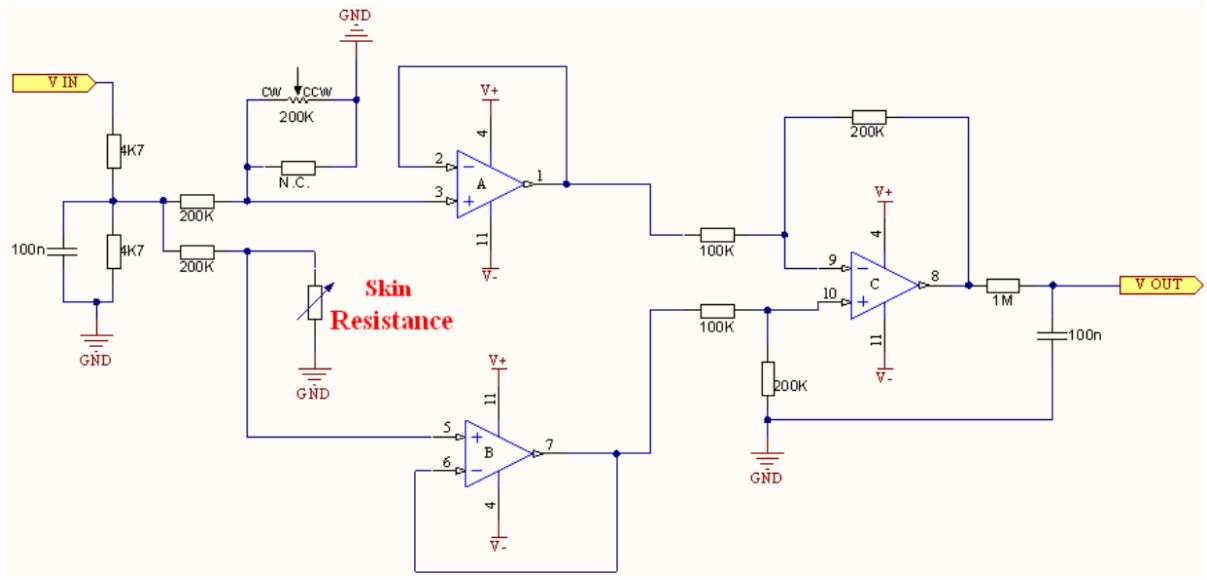

Figura 2.8 Condizionamento GSR [**20**]

La resistenza denominata "Skin Resistance" rappresenta la resistenza della pelle, ovvero quella misurata tra i due elettrodi posti su un dito. Il potenziometro da 200 kΩ serve invece a regolare la sensibilità del sensore in base all'età, al sesso, peso, e così via.

2.1.6 IMU (Inertial Measurement Unit)

L'IMU rappresenta il sensore inerziale, costituito da accelerometro e giroscopio triassiali. Questi vengono adoperati particolarmente nel caso in cui si vogliano analizzare I moviementi del polso, e quindi rotazione e flessione. In genere, le velocità angolari e le accelerazioni lineari non superano rispettivamente i 500 dps (*degree per second*, angoli al secondo) e gli 8g, dove g rappresenta l'accelerazione gravitazionale, pari a 9,8 m/s^2. L'accelerometro che è stato scelto è l'LSM6DS0 prodotto dalla STMicroelectronics. In Figura 2.9 si riportano il suo orientamento e i suoi assi di riferimento per le misure. Bisogna precisare che la Figura 2.9 riporta la rappresentazione della versione successiva rispetto a quello adoperato sulla evaluation board e per cui è stato anche previsto il footprint sulla scheda miniaturizzata. Purtroppo, la versione di partenza è diventata ormai obsoleta. Si è provato a trovare qualche ultimo modello presente sul mercato, o a richiedere qualche adattatore particolare all'azienda produttrice, ma senza successo.

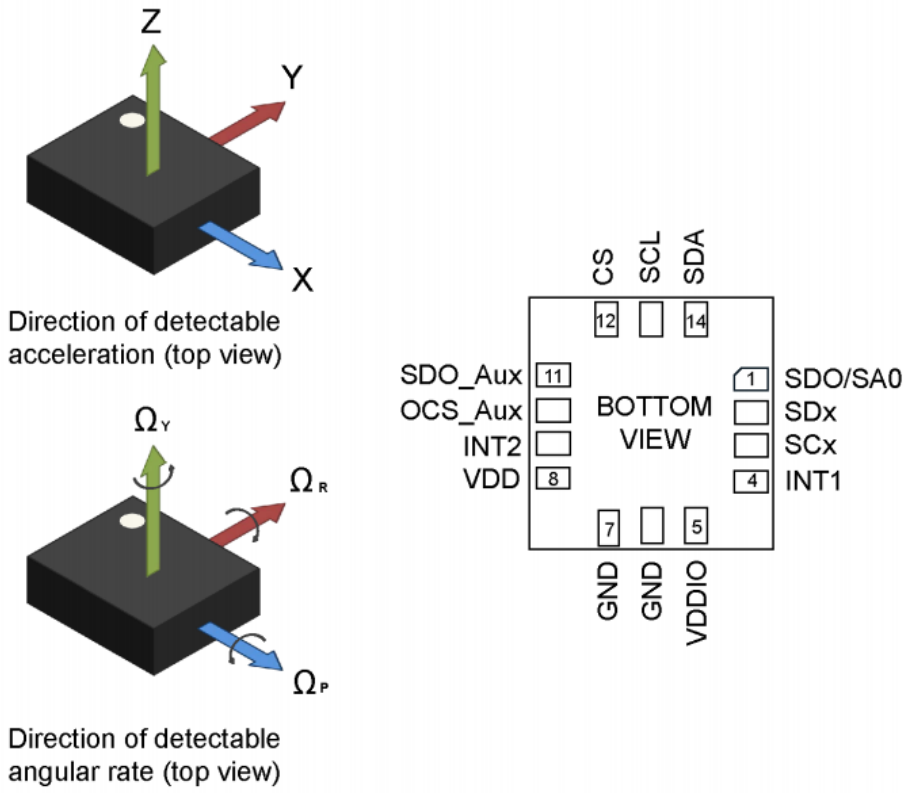

Figura 2.9 Orientamento e assi di riferimento per LSM6DS0 [**22**]

L'integrato è dotato di 6 canali di uscita, ovvero 6 registri di cui 3 per leggere le accelerazioni lineari e 3 per quelle angolari. Ci sono due metodi di funzionamento: uno con accelerometro e giroscopio entrambi attivi e stesso Output Data Rate (ODR), l'altro in cui viene attivato soltanto l'accelerometro. Come per tutti gli altri sensori digitali, per l'interfacciamento con il microcontrollore si è adoperato il bus di comunicazione seriale I2C, nonostante disponga anche di interfaccia SPI.
Una delle caratteristiche maggiormente importanti dell'IMU è la *sensitivity,* cioè la sensibilità, definita in metrologia come il rapporto tra la variazione del valore misurato e la variazione reale della grandezza misurata [23].

La sensibilità dell'accelerometro può essere determinata, applicando lungo uno dei 3 assi di riferimento, un'accelerazione esattamente pari a 1g. Basterà quindi, ad esempio, disporre il chip in maniera piatta, cioè adagiato su un piano perfettamente ortogonale alla perpendicolare al pavimento; successivamente si misura il valore di uscita in LSB. L'operazione va ripetuta capovolgendo il chip e in tal caso si otterrà il valore opposto, perché facendo riferimento al sistema di assi del chip stavolta l'accelerazione applicata sarà -1g. Infine, dividendo l'accelerazione applicata per il risultato ottenuto in LSB si trova quanti m/s^2 si riescono a misurare con 1 LSB, cioè appunto la sensibilità.
Il giroscopio è, invece, un sensore che dà come risultato un valore positivo qualora la rotazione, intorno all'asse preso in considerazione, avvenga in senso orario. La sua sensibilità è pari al guadagno del sensore.

2.1.7 Magnetometro

Oltre all'IMU, utile lo studio del movimento del polso, di fondamentale importanza, poiché i dati vanno integrati l'uno con l'altro, è il magnetometro. Esso fornisce l'angolo di rotazione spaziale rispetto a uno dei 4 punti cardinali (N, S, E, O). L'angolo viene calcolato a partire dalla misura delle 3 componenti del campo magnetico terrestre, ovviamente dipendenti dalla posizione sul globo terrestre.
Il sensore in questione che è stato scelto è l'LIS3MDL dalla STMicroelectronics (Figura 2.10).

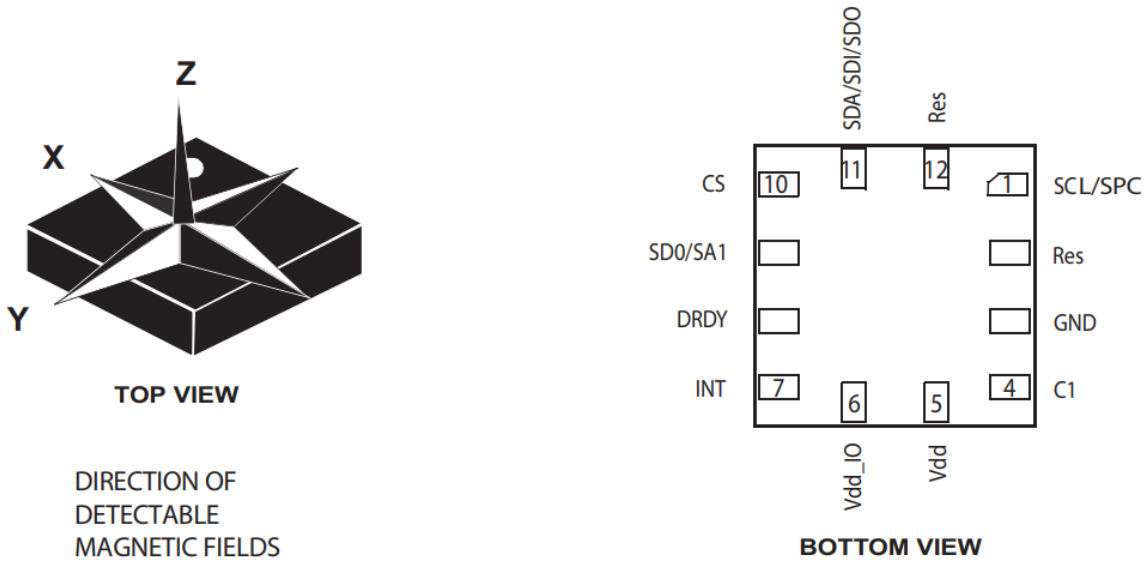

Figura 2.10 Magnetometro LIS3MDL [**24**]

Un'osservazione degna di nota riguarda le possibili interferenze a cui il sensore può essere soggetto. Per la legge di Biot-Savart infatti un filo percorso da corrente genera attorno a sé un campo elettromagnetico. Al fine di minimizzare possibili errori nella misura, dovuti appunto alla somma vettoriale del campo magnetico terrestre e quello generato da fili conduttori adiacenti, è fondamentale collocare il sensore a qualche millimetro di distanza da percorsi in cui scorre una corrente maggiore di 10 mA.

2.1.8 Pulsimetro

Il pulsimetro è il sensore utile al monitoraggio del battito cardiaco. Esistono vari tipi di pulsimetri, ognuno con metodi di misura differenti; quello scelto qui è il MAX30105 prodotto dalla Maxim Integrated (Figura 2.11).

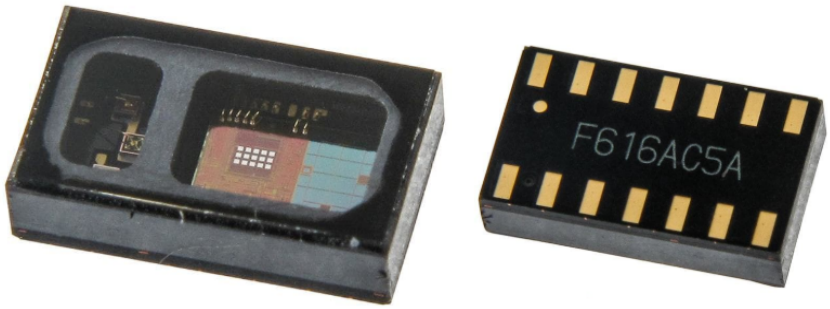

Figura 2.11 MAX30105 **[25]**

Esso è costituito da 3 LED ricoperti da una lamina di vetro, uno verde, uno rosso e uno infrarosso. La corrente che vi scorre può essere programmata e quindi è possibile scegliere l'intensità luminosa di ciascuno. Esso, rimuovendo la componente di luce ambientale, emette la luce dai LED e in base a come essa ritorna riesce a capire il battito cardiaco. Dispone anche di un sensore di temperatura integrato, con una risoluzione di 0,0625 °C.
Per misurare il battito cardiaco il pulsimetro verrà installato su una scheda secondaria a contatto con la parte inferiore del polso.

2.1.9 Sensore di temperatura

Il sensore che è stato scelto è il MAX30205 della Maxim Integrated (Figura 2.12). Esso è progettato per lavorare nell'intervallo di temperatura che va da 0 °C a 50 °C. I valori che si ottengono in uscita sono 2, ciascuno di 1 byte: uno per la parte intera della temperatura, l'altro per la parte decimale, con un'accuratezza di 0,1 °C.

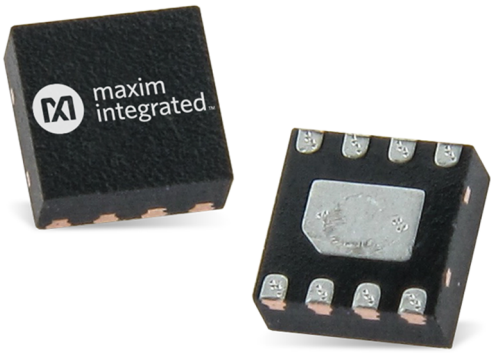

Figura 2.12 Sensore di temperatura MAX30205 **[26]**

Una nota degna di osservazione è la seguente: il valore di temperatura misurato è in particolare quello del die interno dell'integrato, quindi all'interno del package; di conseguenza la correttezza della misura è strettamente legata a come si riesce a trasmettere la temperatura dal corpo umano al package, motivo per cui il sensore di temperatura

deve essere a stretto contatto con la mano. Per questo motivo, sia il sensore di temperatura che il pulsimetro verranno installati non sulla scheda miniaturizzata principale (quella contenente anche il microcontrollore), ma su una scheda miniaturizzata secondaria da collocare a stretto contatto con la parte inferiore del polso.

2.1.10 Microcontrollore

Il microcontrollore è il centro operativo di tutta la scheda, poiché è quello che distribuisce il clock a tutte le periferiche, gestisce tutta la comunicazione con i sensori e l'elaborazione dei dati. Il modulo installato sulla Evaluation Board è il modulo "Aconno ACN52832", che integra il microcontrollore nRF52832 della Nordic Semiconductor e tutta la parte per la connessione Bluetooth, come ad esempio l'antenna in microstriscia, facilmente visibile in Figura 2.13.

Figura 2.13 Aconno ACN52832

Le sue dimensioni sono all'incirca quelle di una moneta da 2 Euro. Di seguito si riportano le caratteristiche fondamentali [27]:

- 32 bit ARM Cortex a 64 MHz
- 512 kB Flash, 64 kB RAM
- Sorgenti di clock a 32 MHz e 32,768 MHz
- 28 GPIO programmabili
- PPI (Programmable Peripheral Interface) per ridurre l'attività della CPU
- SPI, UART, I2C
- ADC a 12 bit (14 in oversampling) con massima frequenza di campionamento di 200 ksps
- Alimentazione da 1,7 V a 3,6 V
- Batterie al litio da 3 V supportate
- Ultra Low Power Consuption
- Intervallo di temperatura: da -40 °C a 85 °C

2.2 Migrazione verso scheda miniaturizzata

La progettazione della scheda miniaturizzata è partita dall'analisi della scheda prototipo *Evaluation* con l'obiettivo principale di ridurre il più possibile le dimensioni al fine di poter essere messa nel guanto, sul dorso della mano. L'obiettivo quindi fondamentale in fase di progetto sono dunque le ridotte dimensioni. La Figura 2.14 illustra il risultato finale. Le dimensioni finali sono di 36 x 33 x 0,9 mm e quindi una riduzione dell'area occupata di oltre il 95% rispetto alla scheda prototipo. In Figura 2.15 si riporta un confronto delle dimensione tra le due schede.

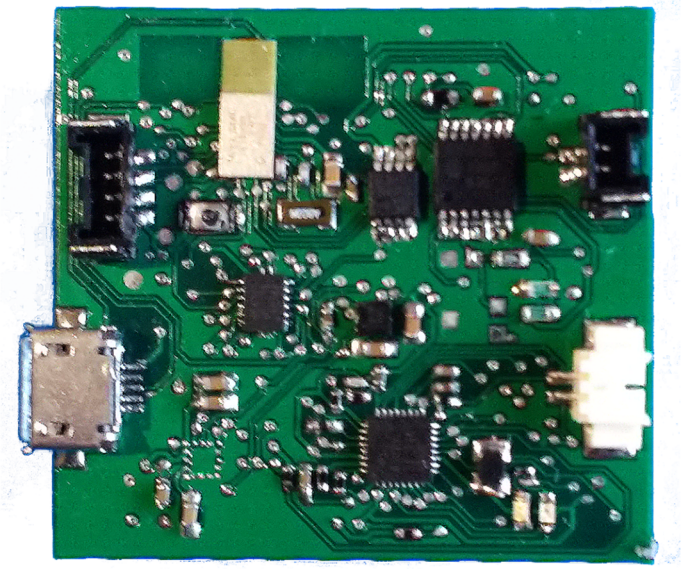

Figura 2.14 Scheda miniaturizzata

Innanzitutto sono state rimosse tutte quelle parti che non servono, quindi ad esempio la *Service Area*, tutti gli header e i connettori utili nella fase di test della scheda prototipo, nonché gli interruttori e i relativi LED per abilitare/disabilitare manualmente i vari sensori e periferiche [28].
Di fondamentale importanza è stata la fase di progettazione del *Power Management:* sono stati rimossi vari connettori e sono stati lasciati soltanto uno per connettere una batteria (3,7 V nominali) e uno micro-USB (5 V) per alimentare la scheda da PC. È stato quindi scelto un integrato che provvede sia al monitoraggio dello stato di carica della batteria, alla sua carica, nonché alla regolazione della tensione di alimentazione su tutta la scheda, usando la batteria o la porta USB quando questa è attaccata. L'integrato in questione è il TPS65721 prodotto dalla Texas Instrument. Esso integra anche un pin per fare un rilevamento della temperatura della batteria.
Ovviamente tutti i componenti discreti sono stati sostituiti con quelli di dimensioni notevolmente più piccoli, così come i sensori e il loro condizionamento è stato fatto tutto a bordo scheda. Nella scheda prototipo, infatti, i sensori come magnetometro e accelerometro erano all'interno di una scheda di valutazione per il sensore stesso. In tal caso è stato preso il relativo integrato e saldato sulla scheda miniaturizzata con tutto il condizionamento necessario. I sensori che sono stati inseriti e saldati sulla nuova scheda sono il magnetometro, l'accelerometro-giroscopio e il sensore GSR. Il sensore di temperatura e il pulsimetro varranno installati in un'altra scheda secondaria da collocare vicino al polso.

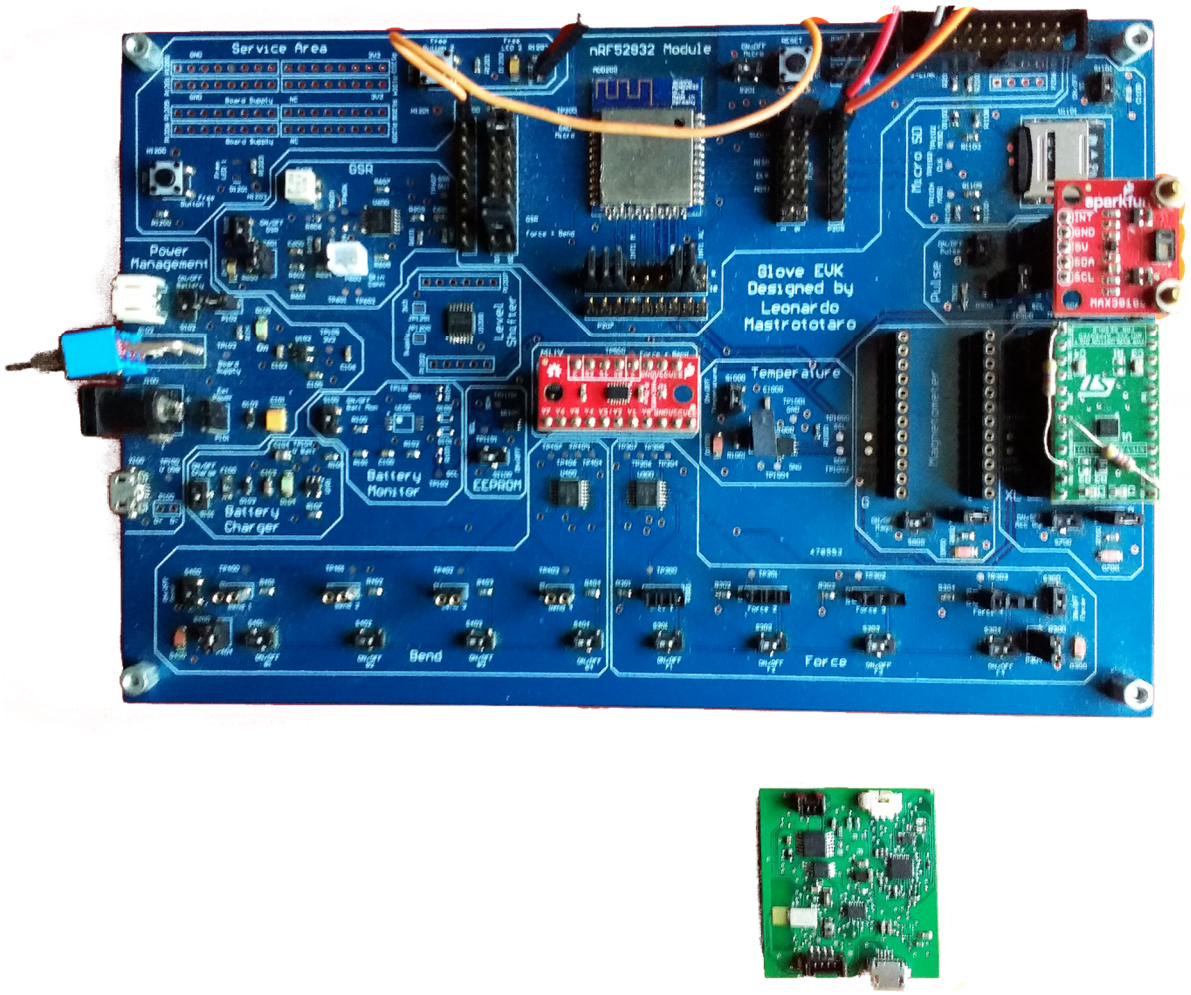

Figura 2.15 Confronto tra le due schede

Per implementare la connessione tra le due e per provvedere anche alla connessione con i sensori di forza e flessione che vanno messi sulle dita, la scheda è stata dotata di appositi connettori chiamati *Pogo Pin* (Figura 2.16); questi usano dei particolari elementi meccanici a molla per connettersi tra loro.

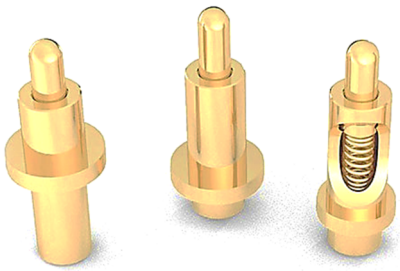

Figura 2.16 Pogo Pin

I Pogo Pin sono stati posizionati sulla parte inferiore della scheda *(Bottom Side)*, così come anche l'integrato contenente i vari operazionali per realizzare il condizionamento dei sensori analogici.

Per l'esattezza, sono presenti 21 Pogo Pin: 16 per connettere gli 8 sensori analogici (forza e flessione) e altri 5 per la comunicazione con la schedina secondaria da posizionare in prossimità del polso.
Mentre nella scheda prototipo era presente il modulo Aconno ACN52832, le cui dimensioni eguagliano quasi quelle della scheda miniaturizzata, in quest'ultima è stato adottato un chip integrato diverso, ma sempre basato sullo stesso microcontrollore (nRF52832) e con Bluetooth 5.0. Il chip in questione è prodotto dalla Taiyo Yuden, per l'esattezza è stato scelto il modello EYSHSNZWZ caratterizzato dalla dicitura *ultra small package* e le cui dimensioni sono 3,25 x 8,55 x 0,85 mm (il primo a sinistra in Figura 2.17).

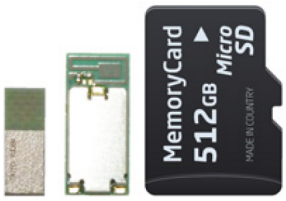

Figura 2.17 Confronto dimensioni del chip

Altra modifica che è stata effettuata per la scheda miniaturizzata è la rimozione della scheda micro-SD, mantenendo soltanto, qualora dovesse servire utilizzarla, la memoria EEPROM. La memoria in questione è la 24LC64 della Microchip Technology, avente a disposizione 64 kBit organizzata come singolo blocco da 8000 righe con una word da 8 bit; ciò significa che è possibile memorizzare fino a 8000 dati da 8 bit.
Per la programmazione della scheda è stato previsto un connettore maschio verticale a 4 pin della Molex. Esso è connesso al microcontrollore mediante interfaccia SWD *(Serial Wire Debug)* e quindi i segnali sono SWDIO, SWDCLK, VDD e GND. Sempre tra i vari connettori saldati sulla scheda vi è quello a 2 pin per connettere il sensore GSR e il connettore femmina micro-USB. In particolare i 5 V vanno in ingresso al pin AC dell'integrato TPS65721, mentre le due linee di dato dell'USB, *Data+* e *Data-*, sono state collegate rispettivamente ai pin GPIO P0.02/AIN0 e P0.03/AIN1 del modulo Bluetooth Taiyo.
Sono stati saldati dei LED per indicare lo stato della batteria: il rosso indica batteria quasi scarica, mentre verde batteria carica.
Per il posizionamento del microcontrollore e dell'antenna Bluetooth sono state seguite delle regole ben precise in modo da garantire il corretto funzionamento della comunicazione. Facendo riferimento alla Figura 2.18, la prima regola consiste nel non collocare alcun componente metallico all'interno del rettangolo con contorno in blu. La seconda regola indica che non devono essere presenti parti in rame all'interno del rettangolo verde; può essere adoperato soltanto il materiale dielettrico FR-4 poiché l'antenna Bluetooth adopera questo tipo di materiale. Infine, terza e ultima regola da seguire, bisogna mettere un piano di ground con una lunghezza minima di 30 mm a partire dal vertice basso del rettangolo verde. Attenendosi a queste regole, il microcontrollore è stato collocato sulla parte superiore della scheda *(Top Side)*, in modo da disporre di un *ground plane* il più lungo possibile, e quasi in prossimità del bordo della scheda.

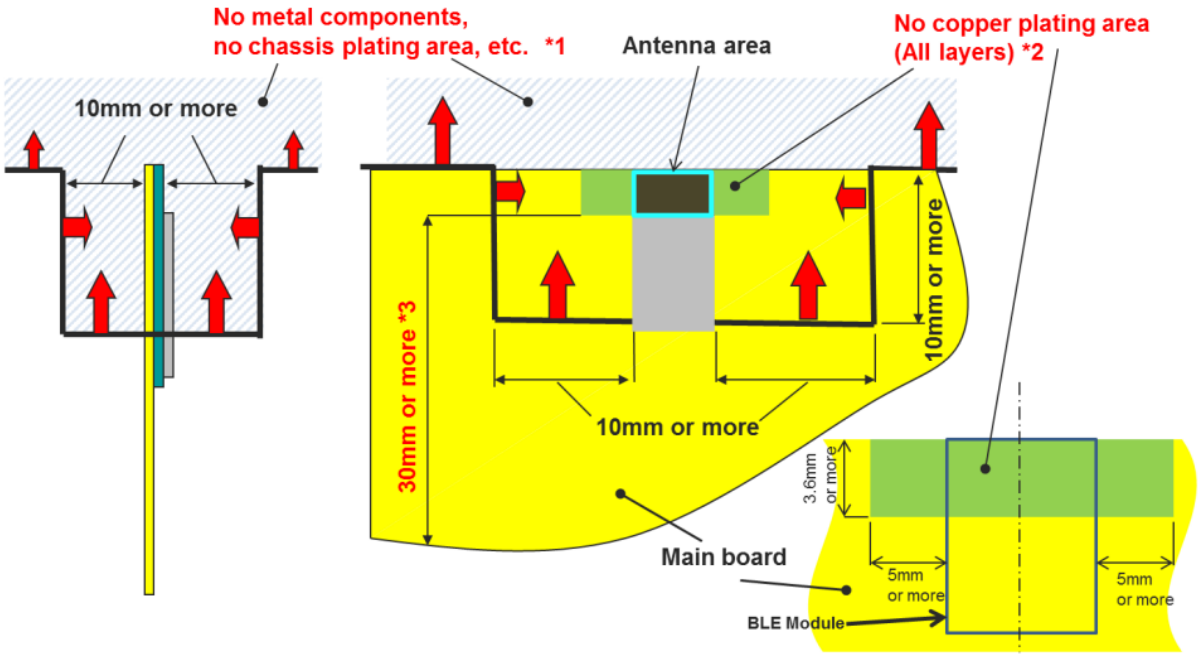

Figura 2.18 Regole per il montaggio del modulo EYSHSNZWZ [**29**]

2.2.1 Packaging

Finito l'assemblaggio dei vari componenti sulla scheda miniaturizzata, è stato realizzato anche l'involucro per poter inserire la scheda all'interno e adagiarla sul dorso del guanto. Nel fare queste operazioni sono stati presi come riferimento i prodotti *wearable* realizzati dall'azienda Athos, specializzata nella realizzazione di dispositivi elettronici per il monitoraggio dell'attività sportiva e che vengono adagiati su appositi indumenti (Figura 2.19).

Figura 2.19 Athos Wearable [**30**]

Il dispositivo a destra si adagia all'interno della tuta con degli appositi magneti e pertanto si è pensato che potesse essere una soluzione molto vantaggiosa per l'applicazione in questione.
Allo stesso tempo, si è preso come riferimento il guanto *Peregrine Glove* realizzato dalla Iron Will, il quale monta sul dorso della mano una scheda che si interfaccia,

tramite appositi connettori quando viene inserita, con i vari sensori presenti anche sulle dita (Figura 2.20).

Figura 2.20 Peregrine Glove [**31**]

Tenendo conto di questi due modelli e delle specifiche di partenza, come ad esempio il fatto che i sensori inerziali devono percepire i vari movimenti compiuti dal polso, si è deciso di realizzare un opportuno packaging da poter adagiare sul dorso della mano e che, una volta inserito, si interfaccia tramite i connettori Pogo Pin alla scheda secondaria (collocata nella parte inferiore del polso) e con i sensori analogici disposti lungo le dita. In Figura 2.21 si può osservare la scheda miniaturizzata disposta dentro al packaging di forma stondata; lo spazio che rimane serve per collocare la batteria al litio che alimenta la scheda.

Figura 2.21 Scheda miniaturizzata dentro al packaging

Infine, in Figura 2.22, si può osservare il confronto di dimensioni tra il guanto e il packaging che verrà attaccato al dorso della mano, quest'ultimo abbastanza piccolo da non impedire i movimenti durante i training.

Figura 2.22 Package appoggiato sul dorso del guanto

Capitolo 3

Sviluppo del firmware

Il capitolo tratta lo sviluppo del firmware del microcontrollore. Gran parte del lavoro svolto è stato testato sulla *Evaluation Board* in modo da avere un controllo maggiore sui sensori, ma anche una visione più grande e di conseguenza una praticità maggiore della scheda stessa.
Al fine di comprendere le scelte progettuali e le varie problematiche, con relativa risoluzione, che sono sorte in fase di sviluppo, l'ordine che verrà seguito è lo stesso di quello adottato in fase di stesura del codice.
È importante inoltre sottolineare che il firmware è stato scritto in maniera modulare; primo motivo fra tutti, poiché quasi tutti se ne dimenticano, è la comprensibilità della lettura del codice. Come uno scrittore suddivide la propria opera in capitoli, paragrafi e argomenti, allo stesso modo un programmatore dovrà fare con il proprio codice, ovvero creare più file (con estensione *.c* e *.h*), all'interno dello stesso progetto, ciascuno con una determinata funzione. Oltre a una questione di semplicità di lettura (che facilita il debug e contribuisce a rendere più agevole il rilascio di nuove versioni) e al fatto che sulla scheda sono presenti sensori di vario tipo (analogici e digitali, i cui dati andranno inviati via Bluetooth), la modularità è dettata dalle specifiche di utilizzo del dispositivo. In altre parole, ogni *training* riabilitativo prevede l'acquisizione di determinate grandezze e non contemporaneamente tutte quelle che i sensori presenti sul guanto sono capaci di acquisire; di conseguenza, è ragionevole attivare soltanto i sensori inerenti allo specifico training riabilitativo, riducendo così il consumo energetico e la quantità di dati da inviare via Bluetooth.

3.1 Configurazione dell'ambiente di sviluppo

Prima di iniziare lo sviluppo del firmware vero e proprio, è necessario configurare l'ambiente di sviluppo comprensivo di SDK (*Software Development Kit*), IDE (*Integrated Development Environment*), driver per interfacciare scheda e PC e altri *tool* secondari che verranno via via descritti.
Il pacchetto di sviluppo, ovvero l'SDK, relativo al microcontrollore utilizzato e messo a disposizione dalla casa produttrice (Nordic Semiconductor) è la versione 15.2, scaricabile dal sito. Si noti, inoltre, che avendo fatto riferimento sempre alla versione 15.2,

e quindi a tutte le *define* e funzioni in essa definite, è consigliabile continuare a lavorare al progetto sempre con questa versione di SDK.
Il modulo montato sulla scheda è l'nRF52832, ovvero un modulo costituito da un'interfaccia Bluetooth e un SoC (*System-on-Chip*) proprietario. In Figura 3.1 viene riportata una rapida descrizione illustrativa.

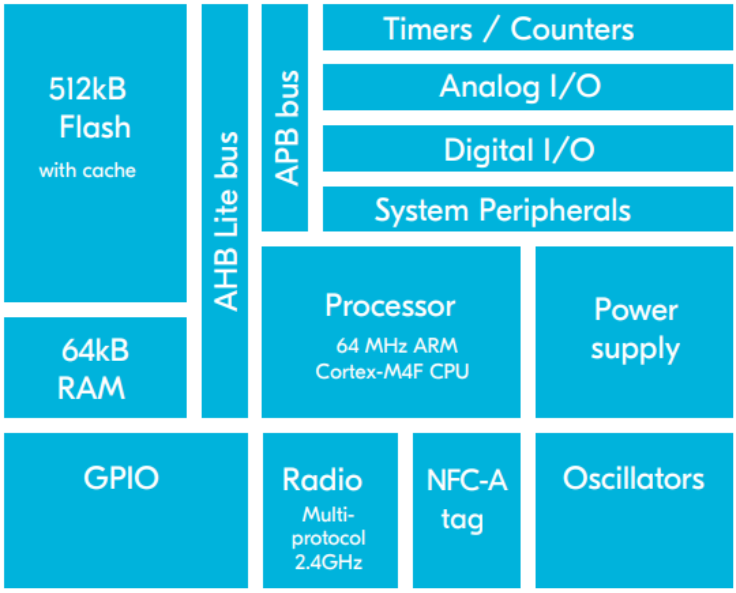

Figura 3.1 Descrizione illustrativa modulo l'nRF52832 **[32]**

Una volta scaricato l'SDK, al suo interno sono presenti le seguenti cartelle di cui si riporta una rapida descrizione:

- *components:* in essa sono presenti tutti i driver per far funzionare le periferiche del microcontrollore, anche per il Bluetooth e il *SoftDevice* (più avanti si vedrà il suo scopo);
- *config:* sono i file di configurazione per la toolchain e i vari moduli, tra questi anche l'nRF52832;
- *documentation:* come dice il nome stesso, è la relativa documentazione dell'SDK, molto importante per lo sviluppo del firmware in quanto fornisce una descrizione dettagliata delle periferiche e le relative funzioni che mette a disposizione l'SDK. Questa qui presente fa riferimento a dei link online, ma è possibile scaricare quella completa anche per poterla consultare offline;
- *examples:* cartella anche questa molto importante poiché fornisce dei progetti di esempio da prendere come riferimento nella creazione del progetto e nella stesura dello stesso;
- *external:* sono le librerie per componenti esterni, quindi anche di terze parti, da poter usare per interfacciare il microcontrollore con *shield* esterne;
- *external_tools:* contiene lo strumento *"CMSIS Configuration Wizard"* utile per l'integrazione e l'abilitazione delle varie librerie relative alle periferiche da usare; se ne farà uso più avanti;
- *integration:* in essa si trovano definite delle nuove librerie: *nrfx*. Queste librerie vanno a costituire un livello aggiuntivo collocandosi sopra a quelle precedenti *nrf* già definite. In questa versione, entrambe le librerie coesistono ed è possibile usare l'una oppure l'altra. La casa produttrice ha già annunciato che presto

verranno abbondonate le vecchie librerie (*nrf*) e che verranno completamente sostituite dalle nuove (*nrfx*). È ragionevole dunque usare le *nrfx* anche perché introducono funzionalità aggiuntive e di più facile utilizzo.
- *modules:* cartella simile alla precedente.

Oltre alle cartelle, all'interno della cartella di root dell'SDK si trovano anche due file di installazione che tuttavia non si sono rilevati utili allo scopo in quanto fanno riferimento a IDE come Keil µVision e IAR. Come IDE è stato adoperato SEGGER Embedded Studio (SES). Per completezza è utile precisare che le versioni SDK compatibili con l'IDE in questione vanno dalla 14.1 in poi. Andando infatti a scegliere un progetto di esempio all'interno della cartella *examples,* si trova la cartella "*ses*" contenente il file di progetto con estensione relativa a SEGGER Embedded Studio.
La scelta dell'IDE è ricaduta su SES per vari motivi, tra cui la possibilità di ottenere una licenza gratuita di tipo non commerciale e il fatto che mette a disposizione i vari driver di interfacciamento. Qualora durante l'installazione di SES non dovrebbero installarsi anche i driver, è possibile scaricarli dal sito singolarmente, cercando i driver *J-Link*. Questi ultimi forniranno anche dei tool molto utili, come ad esempio il tool *J-Link RTT Viewer* che consente di visualizzare vari messaggi su terminale, tra cui anche quelli relativi al debug.
La configurazione dell'ambiente di sviluppo termina qui ma, come si vedrà più avanti, sarà necessario installare altri programmi necessari all'implementazione della connessione Bluetooth, in particolar modo utili alla programmazione del *SoftDevice*, come precedentemente accennato.

3.2 Inizializzazione I2C e rivelazione dei sensori digitali

Per ogni training riabilitativo vanno attivati soltanto alcuni sensori e pertanto il firmware dovrà prevedere la possibilità, da parte dell'utente, quale training scegliere e di conseguenza quali sensori attivare.
Tra i sensori digitali si trovano:

- accelerometro e giroscopio (integrati nello stesso dispositivo che verrà di seguito indicato come IMU (*Inertial Measurement Unit*)
- magnetometro
- termometro
- sensore per il battito cardiaco (*Pulse Oximeter*).

Essi sono collegati e comunicano con il microcontrollore mediante interfaccia seriale I2C, che nel caso dell'nRF52832 viene chiamata bus TWI (*Two-Wire Interface*). Il fatto che si indichino come sensori "digitali" sta a significare che essi presentano una parte digitale al loro interno e, quindi, in essi sono presenti dei registri che opportunamente impostati ne programmano il comportamento del sensore. Non necessitano pertanto di alcun circuito di condizionamento, ma il valore che essi restituiscono è un valore che va soltanto interpretato in base alla grandezza che si sta misurando.
Discorso diverso è invece per i sensori analogici, per i quali è stato impiegato un opportuno circuito di condizionamento, e il cui valore che si otterrà dal convertitore analogico-digitale fa riferimento ad un valore di tensione, ovvero una tensione prodotta dal sensore a fronte di uno stimolo esterno. Tra i sensori analogici adoperati rientrano:

- sensori di forza
- sensori di flessione
- sensore GSR (*Galvanic Skin Response*).

La prima parte dello sviluppo del firmware è consistita nella rivisitazione di un codice precedentemente scritto. In particolar modo, è stata ripresa l'implementazione della comunicazione e l'acquisizione dei dati dei sensori digitali.
L'SDK mette a disposizione molti progetti di esempio, ma anche un *template* vuoto da poter utilizzare. Anziché creare un nuovo progetto da zero sono stati sfruttati due esempi: `twi_scanner` e `twi_sensor`, all'interno della cartella *peripherals* (*examples* → *peripherals*). Il primo si occupa di effettuare una scansione del bus I2C per rilevare eventuali dispositivi connessi, mentre il secondo fornisce un esempio su come comunicare via I2C con un sensore che sfrutta la stessa interfaccia. Prima ancora di lanciare l'esempio è necessario però definire l'impiego dei vari pin del microcontrollore saldato sull'*evaluation board*. È stato pertanto creato il file `custom_board.h` di cui si riportano soltanto alcune parti (omettendo varie righe di codice) per commentare alcuni passaggi importanti.

```
1. #ifndef  CUSTOM_BOARD_H
2. #define  CUSTOM_BOARD_H
3.
4. //SCL SDA (TWI)
5. #define ACONNO_MODULE_SCL   11  //P0.11
6. #define ACONNO_MODULE_SDA   12  //P0.12
7.
8. //MULTIPLEXER
9. #define ACONNO_MODULE_S2    NRF_GPIO_PIN_MAP(0,5)
10.#define ACONNO_MODULE_S1    NRF_GPIO_PIN_MAP(0,6)
11.#define ACONNO_MODULE_S0    NRF_GPIO_PIN_MAP(0,7)
12.#define MUX_OUTPUT          NRF_GPIO_PIN_MAP(0,3)
13.#define MUX_ENABLE          NRF_GPIO_PIN_MAP(0,9)
14.
15.#endif  //CUSTOM_BOARD_H
```

Innanzitutto occorre specificare che un file *.h* nel linguaggio di programmazione C è un cosiddetto *file header* o "file di intestazione". In esso vengono inseriti i prototipi delle funzioni definite nel relativo file *.c*. I prototipi consentono al compilatore di generare un codice oggetto (ovvero la traduzione del codice sorgente in linguaggio macchina binario) che può essere unito (operazione di *linking*) con quello della relativa libreria anche successivamente [33, 34]. In un file header vengono anche inserite le varie *define*. Se il file header `esempio.h` contiene prototipi di funzioni, sarà necessario creare il corrispondente `esempio.c` in cui vengono definite le funzioni e in tal caso richiamare l'header `esempio.h` mediante la direttiva `#include esempio.h`. Seguendo questo ragionamento tutti i file header creati sono stati richiamati in un unico file `include.h`, richiamato nel file `main.c` mediante la direttiva `#include include.h`.
Le direttive `#ifndef`, `#define` e `#endif` servono a dire al compilatore che se il file `custom_board.h` non è stato ancora incluso nella compilazione, al momento dell'inclusione, allora va definito e incluso. La direttiva `#ifndef` termina con la corrispendente `#endif` ed esse vanno utilizzate in tutti i file *.h*, in modo tale che se in qualche parte

del codice ricompare nuovamente la stessa inclusione, essa non viene fatta in quanto il file è già stato incluso, evitando così inclusioni reciproche e ricorsive.
Proseguendo nell'analisi del codice sopra riportato, troviamo le *define* per il bus TWI, cioè SCL e SDA, rispettivamente chiamati `ACONNO_MODULE_SCL` al pin 11 e `ACONNO_MODULE_SDA` al pin 12. Stessa cosa per i pin di controllo del multiplexer che si utilizzerà in seguito.
Una volta definiti i pin per il bus TWI, è possibile procedere alla sua inizializzazione e quindi richiamare la struttura di inizializzazione passando le *define* di cui sopra:

```
1.  void twi_init(void)
2.  {
3.      ret_code_t err_code;
4.
5.      const nrf_drv_twi_config_t twi_config = {
6.         .scl                = ACONNO_MODULE_SCL,
7.         .sda                = ACONNO_MODULE_SDA,
8.         .frequency          = NRF_DRV_TWI_FREQ_100K,
9.         .interrupt_priority = APP_IRQ_PRIORITY_HIGH,
10.        .clear_bus_init     = false
11.     };
12.
13.     err_code = nrf_drv_twi_init(&m_twi, &twi_config, twi_handler, NULL);
14.     APP_ERROR_CHECK(err_code);
15.
16.     nrf_drv_twi_enable(&m_twi);
17.
18.     SEGGER_RTT_printf(0, "TWI initalized\r\n");
19. }
```

Alla riga 18 si nota come sono state introdotte anche le librerie proprietarie SEGGER per poter stampare sul terminale *J-Link RTT Viewer*. Ogni inizializzazione che viene fatta viene notificata sul terminale in modo da tenere sotto controllo il flusso delle istruzioni.
Si osservi come alla riga 13 venga anche abilitato l'interrupt sul TWI, passando l'Interrupt Service Routine (ISR) chiamata `twi_handler`. In seguito verrà spiegato il motivo.
L'operazione successiva consiste nel lanciare la funzione `twi_scanner` per rilevare gli indirizzi dei sensori connessi sul I2C. Lo scanning ha prodotto i seguenti risultati; si riportano gli indirizzi dei sensori:

- IMU: 0x6A
- Magnetometro: 0x48
- Pulse Oximeter: 0x57
- Termometro: 0x48

Dato che gli indirizzi non cambiano, si è scelto di appuntarli e inserirli successivamente nel file header di ogni sensore in modo da non dover effettuare un nuovo scanning ogni qualvolta si accende la scheda. Pertanto la funzione `twi_scanner` pur essendo definita in `twi.c` (e `twi.h`) non verrà richiamata nel codice, ma è utile tenerla a disposizione quando verrà fatta la migrazione del firmware sulla scheda miniaturizzata per trovare i nuovi indirizzi.

3.3 Acquisizione sensori digitali

Dopo la scrittura dei file *twi.c* e *twi.h* che provvedono alla inizializzazione del bus TWI, si è passati alla rielaborazione del codice di ciascun sensore digitale in modo da rendere il tutto il più modulare possibile. Per ogni sensore è stato scritto un file *.c* e *.h*. Nel file header sono stati definiti i registri interni sulla base dei datasheet e ne è stato assegnato un corrispondente valore in modo da stabilire il comportamento di ogni sensore. A titolo di esempio si riporta il file header *temp_sensor.h* contente la configurazione del sensore di temperatura MAX30205.

```
1. /*    Sensore di temperatura MAX30205   */
2.
3. #ifndef TEMP_SENSOR_H_
4. #define TEMP_SENSOR_H_
5.
6. #include "include.h"
7.
8. //Indirizzo sensore
9. #define TEMP_SENS_ADDR       0x48U
10.
11.//definizione dei Pointer bytes dei 4 registri del temp sensor MAX30205
12.#define MAX30205_REG_TEMP    0x00U   //Read-only register -> LETTURA TEMPE-
                                                                    RATURA
13.#define MAX30205_REG_CONF    0x01U
14.#define MAX30205_REG_THYST   0x02U
15.#define MAX30205_REG_TOS     0x03U
16.
17.//Configurazione sensore MAX30205 (si veda datasheet per descrizione regi-
   stri)
18.#define shutdown_modeD0     1   //funzionamento in SHUT DOWN mode
19.#define OScomparator_modeD1 0   //uscita OS in COMPARATOR MODE
20.#define OSpolarityD2        0   //polarità uscita OS => attivo basso
21.#define D3                  1
22.#define D4                  0
23.#define normal_dataD5       0   //formato temperatura => Normal data
24.#define bus_timeoutD6       0   //0: bus time-out ON
25.#define one_shotD7          1   //abilito la ONE-SHOT conversion
26.
27.//definizione di Tmax e Tmin
28.#define TEMP_MAX    0x28U       //40 gradi in °C
29.#define TEMP_MIN    0x01U       //1 gradi in °C
30.
31.#define CONVERSION_TIME_MAX    50
32.
33.void MAX30205_set_mode();
34.void start_new_read();
35.void read_sensor_data();
36.uint8_t MAX30205_read_temp();
37.
38.#endif
```

Si nota come sono presenti le direttive `#ifndef`, `#define` e `#endif`; l'indirizzo trovato tramite la funzione di scanning è stato dichiarato come *define* alla riga 9. Successiva-

mente si trovano gli indirizzi dei quattro registri interni, indicati sul datasheet. Proseguendo ancora sotto vi è tutta la configurazione che è stata scelta. È utile soffermarsi un attimo alla riga 18 in cui si vede che è stata abilitata la *shutdown mode*, ovvero il sensore si trova in modalità basso consumo e quando gli viene richiesto esce da questa modalità ed effettua la conversione (acquisisce la temperatura) [26]. Connessa a questa modalità si trova la *One-Shot conversion* per la quale il sensore non fa un'acquisizione continua ma solo quando il microcontrollore lo interroga. È stata scelta questa configurazione in modo da limitare i consumi e considerando il fatto che non è necessario leggere la temperatura ad una frequenza elevata, come quella ad esempio per l'accelerometro che necessita di una lettura più frequente.
Le stesse operazioni di configurazione dei registri sono state fatte per gli altri sensori, ovviamente tenendo conto del relativo datasheet, e, a tal proposito, sono state scritte delle funzioni di inizializzazione per ciascun sensore. In altre parole, dopo aver correttamente inizializzato il bus I2C tramite la funzione `twi_init()`, è possibile eseguire le funzioni di inizializzazione dei sensori che si necessita abilitare. Quindi, ad esempio, se si necessita del sensore di temperatura basterà eseguire `MAX30205_set_mode()`, che passerà i parametri di configurazione definiti in `temp_sensor.h`. Per passare i parametri occorre usare la funzione dedicata per la trasmissione (TX dal microcontrollore verso il sensore) indicando l'indirizzo del sensore. Nel fare ciò, occorre assicurarsi che la trasmissione del dato sia stata completata (un discorso analogo vale quando si riceve il dato, vale a dire quando si effettua una lettura dei registri del sensore). Questa operazione deve essere fatta per qualsiasi sensore e quindi per tutte le operazioni che avvengono sul bus I2C, motivo per il quale la variabile di tipo `bool` che realizza tale controllo chiamata nel codice `m_xfer_done`, è definita in `twi.c`. L'idea è che finché essa assume il valore `false` allora il bus è occupato; quando invece è `true` l'operazione è andata a buon fine e si può procedere ad un nuovo invio o ricezione. Dunque, prima di mandare o ricevere bisogna settare `m_xfer_done` a `false`, eseguire l'invio/ricezione e aspettare che l'operazione in corso termini. Quando l'operazione termina la variabile viene rimessa a `true`; l'evento che indica che il trasferimento sul bus è stato completato è `NRF_DRV_TWI_EVT_DONE`. Ecco dunque il motivo per cui è stata abilitata la ISR sul TWI. Richiamando l'evento in questione in `twi_handler`, `m_xfer_done` viene commutata su `true`. Si riporta l'handler del TWI:

```
1. void twi_handler(nrf_drv_twi_evt_t const * p_event, void * p_context)
2. {
3.     switch (p_event->type)
4.     {
5.         case NRF_DRV_TWI_EVT_DONE:    //Transfer completed event
6.
7.             if (p_event->xfer_desc.type == NRF_DRV_TWI_XFER_RX)    //RX
8.             {}
9.             if (p_event->xfer_desc.type == NRF_DRV_TWI_XFER_TX)    //TX
10.            {}
11.            m_xfer_done = true;
12.            break;
13.
14.        default:
15.            break;
16.    }
17.}
```

Sia che si tratti di una trasmissione che di una ricezione, una volta che l'operazione è stata completata, alla riga 11 si verifica quanto appena detto.
Per aspettare che `m_xfer_done` diventi `true` basterà usare un `while` che blocca il flusso delle istruzioni, come illustrato nell'inizializzazione del sensore di temperatura, di cui si riporta soltanto il codice relativo all'abilitazione della *shutdown mode*.

```
1.  extern volatile bool m_xfer_done;
2.
3.  void MAX30205_set_mode()
4.  {
5.      ret_code_t err_code;
6.
7.      //configurazione bit D0
8.      m_xfer_done = false;
9.      uint8_t confD0[2] = {MAX30205_REG_CONF, shutdown_modeD0};
10.     err_code = nrf_drv_twi_tx(&m_twi,TEMP_SENS_ADDR,confD0,sizeof(confD0),
        false);
11.     APP_ERROR_CHECK(err_code);
12.     while (m_xfer_done == false);
```

Prima di eseguire l'invio con la funzione `nrf_drv_twi_tx` (riga 10), si imposta a `false` (riga 8) e successivamente si inserisce il `while` (riga 12).
Un'osservazione degna di nota va fatta sulla prima riga del codice sopra. Essendo `m_xfer_done` definita in `twi.c`, occorre richiamarla in tutti i file `.c` dei sensori come variabile `extern`. Ma la cosa sicuramente più importante è il "qualificatore di tipo" `volatile`. Tale parola comunica al compilatore che il valore della corrispondente variabile può essere modificato da eventi esterni al normale flusso sequenziale delle istruzioni. Infatti, in fase di compilazione, il compilatore può ottimizzare una sequenza di accessi in lettura ad una variabile in modo tale che non vi siano scritture intermedie; così facendo, il codice eseguibile preleva dalla memoria il valore della variabile una sola volta e lo colloca in uno dei registri del processore al fine di riusare in seguito tale registro con la certezza che questo continui a mantenere il valore corretto. Il qualificatore di tipo `volatile` elimina tale certezza, indicando al compilatore a produrre un codice che rilegge dalla memoria il valore di tale variabile, non essendovi garanzia che questo sia, nel frattempo, rimasto inalterato [35, 36]. Ciò sta a significare che, in tal caso, se viene omesso `volatile`, il flusso delle istruzioni non va avanti ma si arresta al `while(m_xfer_done == false)`. Il valore di `m_xfer_done` viene commutato, durante la sequenza delle istruzioni, da un'altra parte di codice, ovvero dall'ISR del TWI e dunque bisognerà andare a ricontrollare il suo valore in memoria e non nei registri interni della CPU.

3.4 Acquisizione sensori analogici

Per quanto riguarda l'acquisizione dei sensori analogici, ovvero sensori di forza, flessione e GSR (*Galvanic Skin Response*), si è fatto riferimento all'esempio *saadc* contenuto nella cartella *peripherals*.
IL convertitore analogico-digitale presente nel microcontrollore fa uso di un registro ad approssimazioni successive (SAR) [37]. Le sue caratteristiche fondamentali sono:

- Risoluzione a 8/10/12 bit o 14 bit in modalità di sovracampionamento (*oversampling*)
- Fino a 8 canali di ingresso
- Supporto per il trasferimento diretto in RAM del campione acquisito, ovvero DMA (*Direct Memory Access*)

Le modalità di funzionamento sono:

- *One-shot mode:* la conversione viene "triggherata" a comando, utile ad esempio per acquisire un valore da un determinato canale
- *Continuous mode:* acquisizione continua che viene dettata dal timer interno dell'ADC, oppure da un timer dedicato, facendo anche uso del PPI (*Programmable Peripheral Interconnect*)
- *Oversampling:* modalità di sovracampionamento utile alla riduzione del rumore su un determinato canale; non può essere usata in *Scan mode*
- *Scan mode:* utile quando bisogna fare un'acquisizione di tutti i canali abilitati dell'ADC, ovvero una scansione degli stessi.

I canali che mette a disposizione l'ADC del microcontrollore sono 8, mentre i sensori analogici da acquisire sono 9: 4 sensori di forza, 4 sensori di flessione e il GSR. Per questo motivo, ma anche per evitare di usare tutti i canali a disposizione e lasciare libero qualcuno per eventuali sviluppi futuri, sulla scheda è stato previsto l'utilizzo di un multiplexer (MUX). Gli 8 canali di ingresso del MUX sono i 4 sensori di forza e i 4 sensori di flessione (le 4 dita escluso il pollice). L'uscita del multiplexer è connessa al canale 1 dell'unico convertitore analogico digitale di cui dispone il microcontrollore; pertanto tutti i segnali provenienti dai sensori di forza e di flessione vanno a finire su un unico canale. Sarà quindi necessario effettuare una scansione dei sensori in questione andando a selezionare, ad ogni acquisizione, un ingresso diverso del MUX (S2, S1, S0).

Tabella 3.1 Ingressi di selezione e sensore corrispondente

S2	S1	S0	SENSORE
0	0	0	FORCE_1
0	0	1	FORCE_2
0	1	0	FORCE_3
0	1	1	FORCE_4
1	0	0	BEND_1
1	0	1	BEND_2
1	1	0	BEND_3
1	1	1	BEND_4

FORCE_1 e BEND_1 sono rispettivamente il sensore di forza e flessione dell'indice, così come FORCE_4 e BEND_4 sono quelli del mignolo.
Di seguito viene riportato il codice di inizializzazione dell'ADC:

```
1. void adc_init(void)
2. {
3.     mux_init();
4.     saadc_init();
5.     saadc_sampling_event_init();
```

```
6.      saadc_sampling_event_enable();
7.
8.      SEGGER_RTT_printf(0, "ADC initalized\r\n");
9.  }
```

Alla prima riga si trova l'inizializzazione del MUX; questa è una fase importante perché determina tutto l'ordine della sequenza di acquisizione e quindi la struttura del buffer da trasmettere poi via Bluetooth. Essa consiste nel configurare come output i pin GPIO (*General Purpose Input Output*) del microcontrollore connessi ai pin si selezione e di abilitazione del MUX e una particolare sequenza degli ingressi da cui partire, come si vedrà in seguito.
La funzione `saadc_init()` imposta il canale di acquisizione su *single-ended*, il buffer in cui salvare i campioni e il guadagno interno, per il quale è stato tenuto in considerazione che i sensori sono alimentati a 3.3 V, che è anche il riferimento interno dell'ADC.
La modalità che è stata scelta per il funzionamento dell'ADC è la *Continuous mode*.
La funzione `saadc_sampling_event_init()` effettua proprio questa configurazione impostando un timer (che detta la frequenza di campionamento) e definendo il ruolo del PPI.
Il PPI (*Programmable Peripheral Interconnect*) abilita le periferiche a comunicare in maniera autonoma tra di loro, ovvero senza l'ausilio della CPU, associando un *task* a un *event* [37].

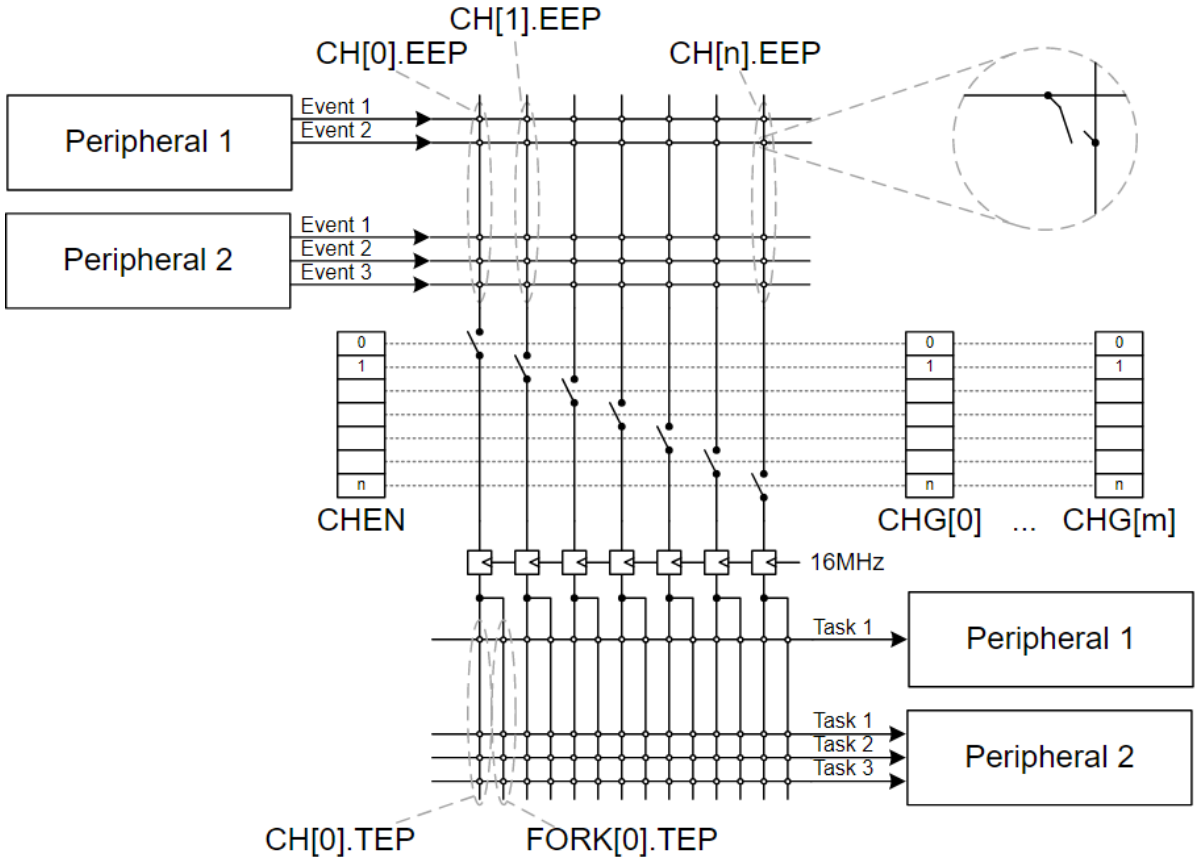

Figura 3.2 Interconnessione tramite PPI [**37**]

Come si vede in

Figura 3.2, il PPI non ha un singolo canale bensì molteplici per mettere in comunicazione tra loro varie periferiche e associare ad un particolare evento di una un compito dell'altra. In tal caso, l'evento scatenante l'interrupt del timer, ovvero il conteggio eguaglia il valore del *counter register*, fa sì che venga triggherato il processo (task) di acquisizione dell'ADC. In questo modo, la CPU non verrà richiamata per gestire l'operazione di acquisizione e, nel caso in cui non stia svolgendo altre operazioni, potrà rimanere in uno stato di *idle*, come si vedrà in seguito quando si introdurrà il Bluetooth.
La riga 6, `saadc_sampling_event_enable()` abilita il canale PPI preconfigurato e quindi dà inizio al funzionamento dell'ADC.
Durante il processo di acquisizione avvengono due ISR:

- L'interrupt dell'ADC, `saadc_callback`, che si verifica al riempimento del buffer contenete gli 8 campioni acquisiti, ovvero un campione per sensore,
- L'interrupt del timer, `timer_handler`, che si verifica alla frequenza di campionamento impostata, e che pertanto commuta l'ingresso del MUX.

In altre parole, ogni `1/SAADC_SAMPLE_RATE` secondi avviene un nuovo campionamento, ogni volta di un sensore diverso e, dunque, l'ISR del timer dovrà contenere la commutazione del MUX per l'acquisizione di un nuovo sensore. Si riporta di seguito il codice relativo alla ISR del timer.

```
1. static void timer_handler(nrf_timer_event_t event_type, void * p_context)
2. {
3.     if(event_type == NRF_TIMER_EVENT_COMPARE0)
4.     {
5.         static uint8_t buffer_index;
6.
7.         switch(buffer_index)
8.         {
9.             case 0:   //FORCE_1 => 000
10.            nrf_gpio_pin_clear(ACONNO_MODULE_S1);
11.            break;
12.
13.            case 1:   //BEND_1 => 100
14.            nrf_gpio_pin_set(ACONNO_MODULE_S2);
15.            break;
16.
17.            case 2:   //BEND_2 => 101
18.            nrf_gpio_pin_set(ACONNO_MODULE_S0);
19.            break;
20.
21.            case 3:   //FORCE_2 => 001
22.            nrf_gpio_pin_clear(ACONNO_MODULE_S2);
23.            break;
24.
25.            case 4:   //FORCE_4 => 011
26.            nrf_gpio_pin_set(ACONNO_MODULE_S1);
27.            break;
28.
29.            case 5:   //BEND_4 => 111
30.            nrf_gpio_pin_set(ACONNO_MODULE_S2);
31.            break;
32.
```

```
33.             case 6:   //BEND_3 => 110
34.             nrf_gpio_pin_clear(ACONNO_MODULE_S0);
35.             break;
36.
37.             case 7:   //FORCE_3 => 010
38.             nrf_gpio_pin_clear(ACONNO_MODULE_S2);
39.             break;
40.         }
41.         buffer_index=(buffer_index+1)%SAMPLES_IN_BUFFER;
42.     }
43. }
```

Nel codice si individua anche la sequenza di acquisizione dei vari sensori e quindi come essi sono collocati all'interno del buffer. Anziché andare a selezionare i sensori in sequenza, cioè andare a leggere in sequenza gli 8 ingressi del MUX (Figura 3.3), si è preferito adottare una codifica di tipo Gray.

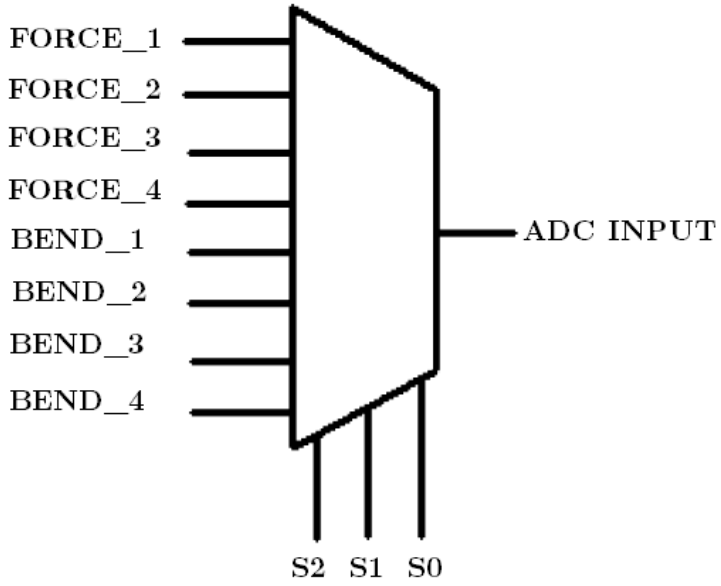

Figura 3.3 MUX e sensori

In questo modo si provocano meno commutazioni interne al MUX e si alleggerisce anche il codice in quanto basta fare un toggle di un solo ingresso di selezione. Questo spiega anche il motivo per cui in `mux_init()` gli ingressi di selezione vengono inizialmente messi a 010, che corrisponde all'ultimo sensore da acquisire (Force_3). Alla prima acquisizione, ovvero alla prima esecuzione della ISR del timer, gli ingressi di selezione vengono portati alla sequenza 000, in modo da rispettare la sequenza riportata in Tabella 3.2.

Tabella 3.2 Struttura del buffer ADC e sequenza con codifica Gray

Sensore	Force1	Bend1	Bend2	Force2	Force4	Bend4	Bend3	Force3
$S_2S_1S_0$	000	100	101	001	011	111	110	010

Nell'ISR dell'ADC, `saadc_callback`, si provvede invece alla lettura del buffer e alla creazione di uno nuovo, contenete informazioni aggiuntive, per l'invio tramite Bluetooth.

3.4.1 Acquisizione del GSR

Il GSR è connesso al canale 0, `NRF_SAADC_INPUT_AIN0`, dell'ADC. Innanzitutto occorre precisare che una volta che l'ADC è stato inizializzato (ad esempio perché bisogna fare uso dei sensori di forza e di flessione), non è possibile eseguire una nuova inizializzazione allo stesso modo, pena errore e blocco del microcontrollore. Di conseguenza, l'inizializzazione dell'ADC va fatta una sola volta.
Inizialmente, era stato pensato di riscrivere il codice per l'ADC in modo da farlo funzionare con due canali, uno proveniente dall'uscita del MUX e uno direttamente connesso al GSR. Tuttavia, sebbene non sia impossibile, il codice si complicherebbe non di poco in quanto occorrerebbe effettuare una scansione del MUX (tramite interrupt del timer) e una commutazione del canale dell'ADC dopo aver acquisito gli 8 sensori connessi al MUX. In tal caso, molto probabilmente, sarebbe più conveniente usare un'acquisizione di tipo *one-shot* in modo da avere il pieno controllo sulla commutazione dei due canali dell'ADC (in *scan-mode* si avrebbe un continuo passaggio da un canale all'altro a ogni nuovo campione).
Al tempo stesso, dalla Tabella 2.1 si è notato che il GSR e gli altri sensori non vengono mai usati contemporaneamente, cioè non vi è alcun training riabilitativo che prevede l'impiego sia del GSR sia dei sensori di forza e/o flessione. Per questo motivo, la scelta è ricaduta sull'adottare lo stesso codice precedentemente scritto per i sensori di forza e flessione, anziché configurare l'ADC per lavorare con due canali. Ciò che cambia è il canale di ingresso diverso, per l'esattezza `NRF_SAADC_INPUT_AIN0`, e il fatto che l'ISR del timer non sarà dovrà eseguire nulla (nel codice precedente cambiavano i pin di selezione del MUX).
Occorre tuttavia precisare che le due funzioni di inizializzazione, `adc_init()` e `gsr_init()`, vanno usate in maniera esclusiva, ovvero l'uso dell'una esclude l'uso dell'altra, altrimenti si incorrerebbe in una duplice inizializzazione dell'ADC arrestando il funzionamento del microcontrollore.

3.4.2 CMSIS Configuration Wizard

Lanciando un progetto di esempio, oltre al `main.c`, si trova anche il file `sdk_config.h`. Questo è il cosiddetto *SDK Configuration Header* che serve a gestire e configurare le varie dichiarazioni (e *define*) dell'applicazione. Nel caso in esame bisogna far riferimento al PCB PCA10040 che usa il microcontrollore nRF52.
A titolo di esempio si riporta una parte di `sdk_config.h` inerente la configurazione del convertitore analogico-digitale:

```
1. // <e> NRFX_SAADC_ENABLED - nrfx_saadc - SAADC peripheral driver
2. //==========================================================
3. #ifndef NRFX_SAADC_ENABLED
4. #define NRFX_SAADC_ENABLED 1
5. #endif
6. // <o> NRFX_SAADC_CONFIG_RESOLUTION  - Resolution
7.
8. // <0=> 8 bit
9. // <1=> 10 bit
10.// <2=> 12 bit
```

```
11. // <3=> 14 bit
12.
13. #ifndef NRFX_SAADC_CONFIG_RESOLUTION
14. #define NRFX_SAADC_CONFIG_RESOLUTION 1
15. #endif
16.
17.
18. // <o> NRFX_SAADC_CONFIG_IRQ_PRIORITY  - Interrupt priority
19.
20. // <0=> 0 (highest)
21. // <1=> 1
22. // <2=> 2
23. // <3=> 3
24. // <4=> 4
25. // <5=> 5
26. // <6=> 6
27. // <7=> 7
28.
29. #ifndef NRFX_SAADC_CONFIG_IRQ_PRIORITY
30. #define NRFX_SAADC_CONFIG_IRQ_PRIORITY 6
31. #endif
32.
33. // <e> NRFX_SAADC_CONFIG_LOG_ENABLED - Enables logging in the module.
34. //==========================================================
35. #ifndef NRFX_SAADC_CONFIG_LOG_ENABLED
36. #define NRFX_SAADC_CONFIG_LOG_ENABLED 0
37. #endif
```

Come si nota, sono presenti le direttive `#ifndef`, `#define` e `#endif`, che vengono usate per indicare se l'ADC è abilitato o meno, nel caso di abilitazione il suo valore di risoluzione o la priorità della ISR, e così via fino anche all'abilitazione della generazione di messaggi di log (dalla riga 33 alla 37), utili per il debug.
Ecco dunque che quando si usa una periferica, ad esempio l'ADC, occorre abilitarla nel file `sdk_config.h`. Il file `sdk_config.h` è stato incluso tra le varie inclusioni nel file `include.h`. Pertanto, in fase di compilazione il compilatore andrà a leggere tutte le periferiche attive e le relative configurazioni nel file `sdk_config.h`.
È possibile configurare tale file manualmente direttamente dall'IDE o usando un semplice editor di testo, ma ciò comporterebbe, oltre a un notevole dispendio di tempo e energia, anche la possibilità di commettere errori in quanto si tratta di oltre 12000 righe di codice.
Il tool *CMSIS Configuration Wizard* serve proprio a configurare il file `sdk_config.h` in maniera semplice e intuitiva mediante un'apposita interfaccia grafica. Prima di poterlo usare è necessario però importarlo nell'IDE, come specificato nella guida "Nordic Semiconductor Infocenter" consultabile online, o dalla documentazione [38] dell'SDK andando in *nRF5 SDK→ Getting Started→ SDK configuration header file.*
Una volta configurato, basterà fare click destro sul file `sdk_config.h` (nella barra laterale *"Project Explorer"* di SES) e schiacciare su *CMSIS Configuration Wizard*; si aprirà una finestra come quella in Figura 3.4. Si vede come sono presenti varie cartelle; esse non sono altro che il contenuto di `sdk_config.h`, ma strutturato in modo che sia facilmente interpretabile. Ritornando all'esempio del codice relativo all'ADC sopracitato e aprendo la cartella "*nRF_Drivers*", in Figura 3.5 si nota come è possibile procedere alla sua abilitazione e alla sua configurazione in maniera molto intuitiva. Occorre anche prestare particolare attenzione al fatto che, come precedentemente detto,

esistono delle librerie *nrfx* e *nrf* e pertanto per ogni periferica si trovano due menù di configurazione, da configurare entrambi.

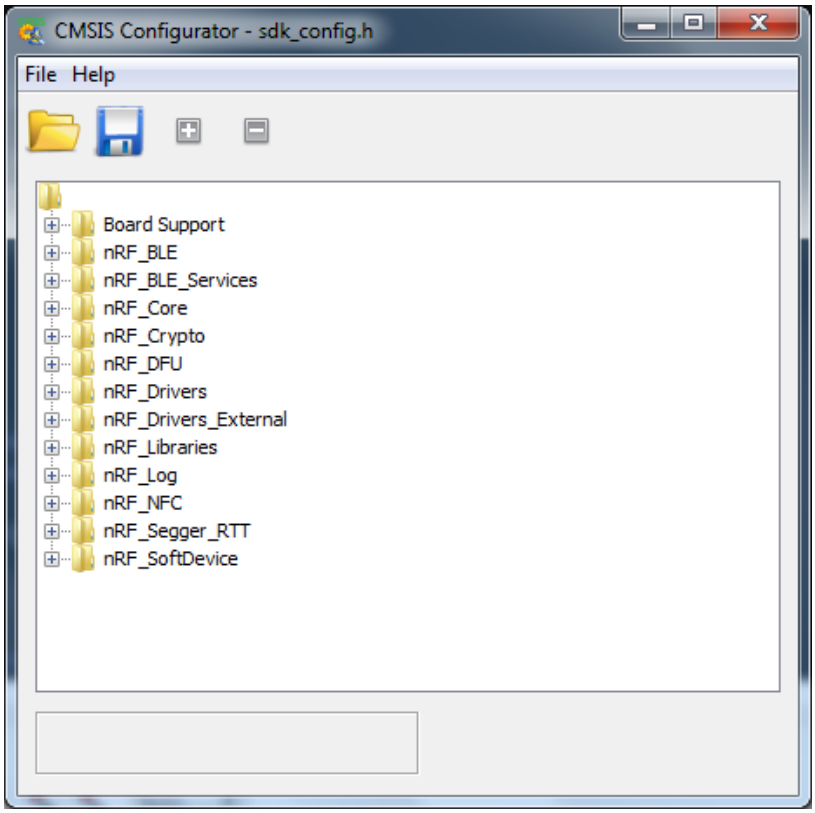

Figura 3.4 *CMSIS Configuration Wizard*

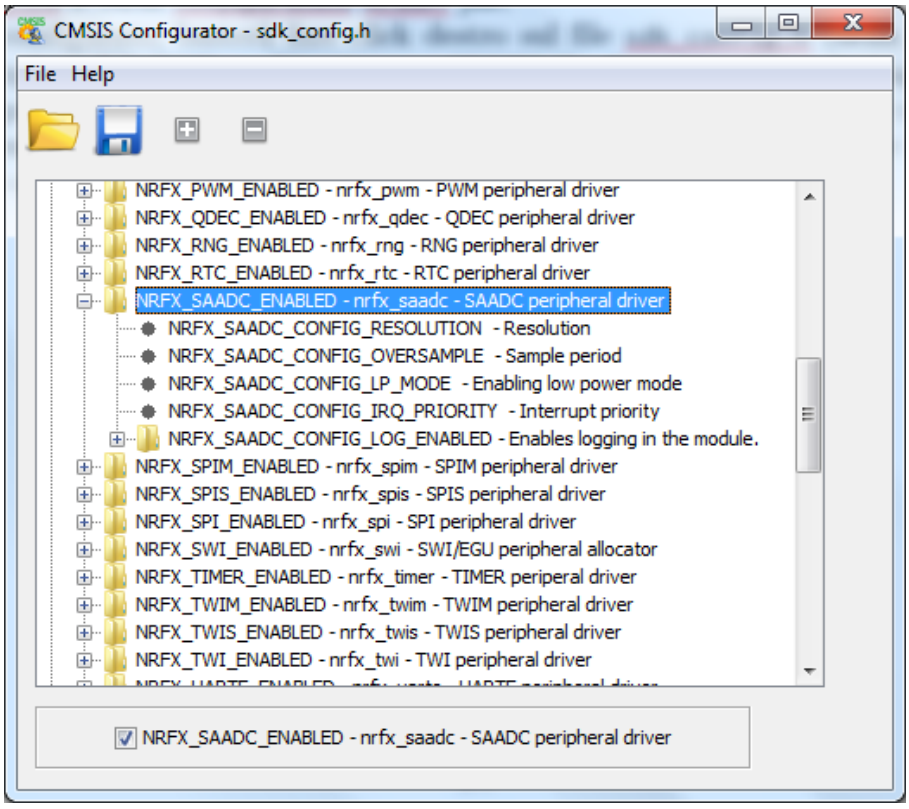

Figura 3.5 Configurazione ADC mediante *CMSIS Configuration Wizard*

L'uso del tool *CMSIS Configuration Wizard* per la configurazione delle periferiche non sostituisce la creazione dell'istanza e la successiva inizializzazione della funzione di *init* dedicata. Tuttavia è importante che la periferica in questione venga abilitata con il segno di spunta (Figura 3.5), in modo tale che vengano caricate le relative librerie. È bene inoltre che vi sia corrispondenza tra il codice di inizializzazione della periferica scritto manualmente e i parametri definiti mediante *CMSIS Configuration Wizard*. Ad esempio, se per l'ADC è stata scelta una risoluzione di 12 bit con la linea di codice `saadc_config.resolution = NRF_SAADC_RESOLUTION_12BIT`, è bene impostare la stessa cosa anche in *CMSIS Configuration Wizard.*

Capitolo 4

Connettività wireless e risparmio energetico

Uno dei tanti motivi per cui è stato scelto il microcontrollore nRF52832 è, senza dubbio, il fatto che esso dispone della connettività Bluetooth, in particolare del Bluetooth 5, e pertanto esso fornisce tutto il supporto software con uno stack dedicato (*SoftDevice*) per le nuove funzionalità che esso implementa.

4.1 Cenni al funzionamento e caratteristiche del Bluetooth 5

Il Bluetooth è uno standard di comunicazione e trasmissione dati. Esso fornisce un metodo economico e sicuro per scambiare informazioni tra dispositivi tra loro vicini e di vario tipo. Cellulari, computer, tablet, fotocamere, stampanti, braccialetti *smart* intelligenti, accessori per lo sport, sono tutti esempi di dispositivi che ormai usano una connessione Bluetooth per comunicare tra loro e lasciano intuire quanto questo tipo di connessione abbia ormai preso il sopravvento.
L'ultima versione di questo standard è il Bluetooth 5, evoluzione della versione 4.2 che già metteva a disposizione la caratteristica LE, ovvero *Low Energy* (basso consumo di energia), anche conosciuta come "*Bluetooth Smart*" [39].
Le caratteristiche più importanti che il Bluetooth 5 introduce sono [40, 41]:

- Un datarate fino a 2 Mbps, 2 volte superiore,
- Un raggio di copertura fino a 4 volte maggiore pur mantenendo lo stesso consumo energetico, ma riducendo la velocità a 500 kbps e 125 kbps,
- Capacità di trasmissione fino a 8 volte superiore in broadcast (*advertising extension*),
- Maggiore efficienza nell'uso dei canali di trasmissione sulla banda 2,4 GHz.

Anche se si ha copertura del funzionamento con le precedenti versioni (retrocompatibilità), il Bluetooth 5 introduce un nuovo e completamente diverso metodo di comunicazione, che prende il nome di *"advertising mode"*. Ciò significa che è comunque possibile instaurare una comunicazione e scambiare dati con un dispositivo che usa una versione precedente, ma chiaramente non si avranno i vantaggi che apporta la versione 5.

Le 2 modalità di funzionamento sono:

- *Connected mode*, che usa il livello *Generic Attribute* (GATT) per instaurare una comunicazione con un atro dispositivo (*one-to-one connection*) e scambiare dati,
- *Advertising mode*, che invece usa il livello *Generic Access Profile* (GAP) per trasmettere i dati in modalità broadcast a tutti i dispositivi che sono in ascolto e abilitati a ricevere i pacchetti provenienti dal dispositivo che li trasmette.

Al fine di comprendere le varie differenze nella comunicazione tra le diverse versioni del Bluetooth, la Figura 4.8 illustra i differenti stack.

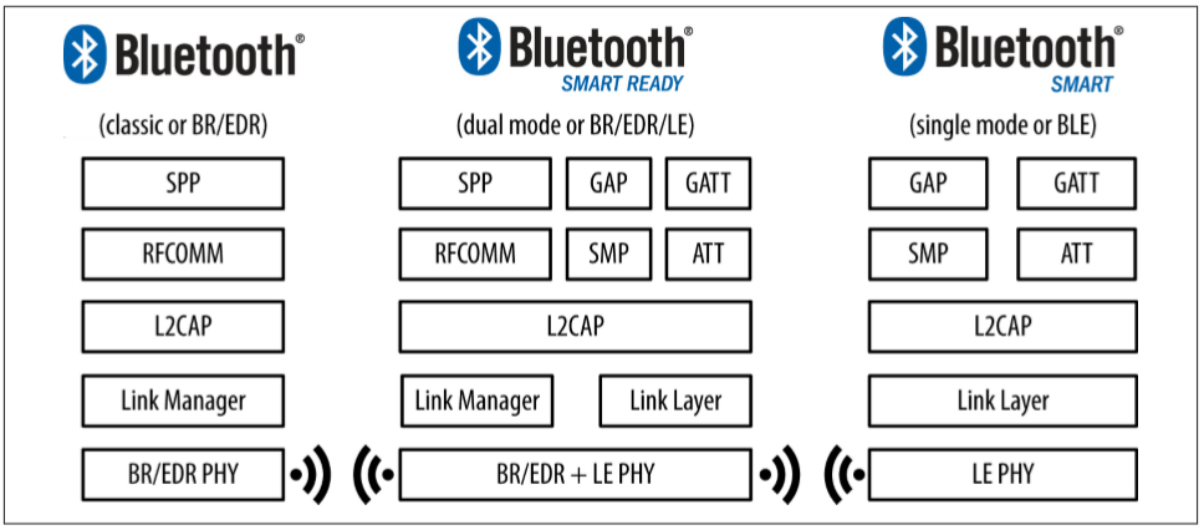

Figura 4.1 Tipi di configurazione e di dispositivi in varie versioni Bluetooth [**42**]

- *BR/EDR (classic Bluetooth):* standard che si è evoluto dalla versione 1.0 (primo da sinistra in Figura 4.1),
- *BLE (Bluetooth Low Energy):* standard a basso consumo introdotto dalla versione 4.0 (secondo e terzo da sinistra in Figura 4.1).

Quest'ultimo si suddivide a sua volta in 2 categorie:

- *Single-mode (BLE, Bluetooth Smart) device:* dispositivo che implementa il BLE e che può comunicare in *single-mode* o *dual-mode* (terzo da sinistra in Figura 4.1), ma non con quelli EDR (*Enhanced Data Rate*),
- *Dual-mode (BR/EDR/LE, Bluetooth Smart Ready) device:* implementa sia BR/EDR che il BLE (al centro in Figura 4.1) e può comunicare con qualsiasi dispositivo Bluetooth

4.1.1 Livelli di protocollo

Ci sono molti livelli di protocollo; qui si riportano i 3 più importanti:

- *Application:* livello di interfaccia utente con tutto il resto dello stack Bluetooth,
- *Host:* i livelli più alti di tutto lo stack Bluetooth e immediatamente sotto il livello *application*,
- *Controller:* livello più basso che include anche la parte radio, quindi a stretto contatto con l'antenna.

In realtà, in molti casi, è presente un ulteriore livello tra quello di *Host* e quello di *Controller,* chiamato per l'appunto *Host Controller Interface* (HCI). Questo viene inserito da alcuni produttori per poter aggiungere particolari funzioni proprietarie e avere un maggiore controllo sullo stack qualora il livello di *Host* e di *Controller* siano realizzati da produttori diversi e quindi occorre farli comunicare in maniera opportuna.
I livelli enunciati possono essere implementati o su un singolo circuito integrato (IC) o su integrati separati connessi tra loro mediante un livello di comunicazione come (UART, USB, SPI, ecc.).
Le tre configurazioni più comuni disponibili in commercio sono:

- *SoC (System on Chip):* un unico IC in cui sono implementati tutti e 3 i livelli *(Application, Host, Controller),*
- *Dual IC over HCI:* un IC esegue il livello applicativo e quello di *host* e comunica con il *Controller* mediante HCI,
- *Dual IC with connectivity device:* un IC esegue il livello applicativo e comunica usando un protocollo proprietario con un secondo IC in cui si trovano sia *Host* che *Controller.* Essendo il protocollo di comunicazione proprietario, l'applicazione va adattata alle particolari specifiche e caratteristiche implementate dal produttore.

La Figura 4.2 riporta quanto appena detto.

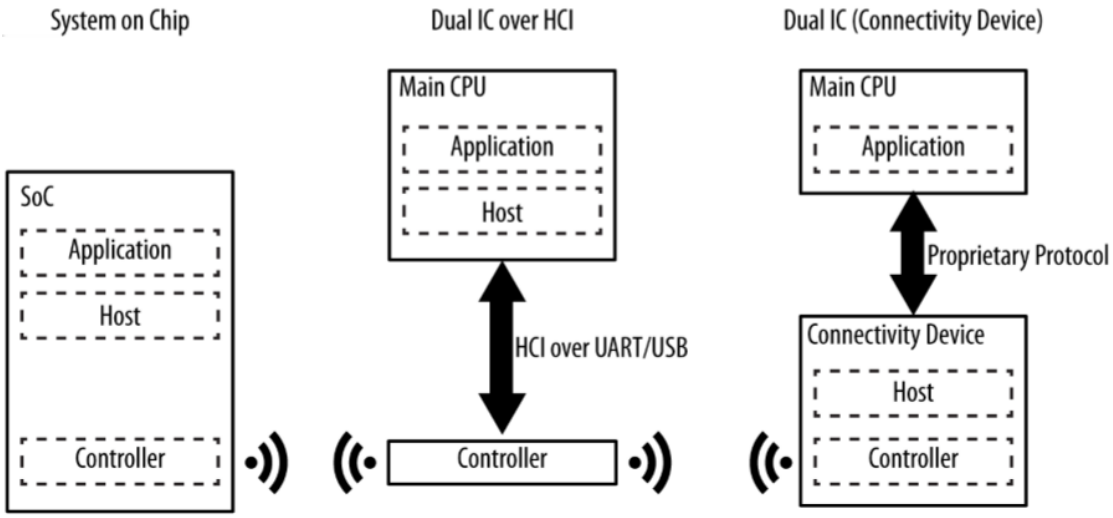

Figura 4.2 Possibili configurazioni Hardware [**42**]

In genere, i sensori (ad esempio per IoT) usano la configurazione *SoC* per mantenere i costi di produzione e le dimensioni del PCB i più bassi possibile; dispositivi come smartphones, tablet e simili impiegano l'architettura di tipo *Dual IC over HCI* in quanto hanno a disposizione una CPU più potente. La configurazione *Dual IC con Connectivity Device* è invece usata in altri tipi di dispositivi, ad esempio un orologio in cui è già presente un microcontrollore che esegue determinate funzionalità, ma si vuole aggiungere quella della connettività Bluetooth, senza stravolgere l'intero progetto. Quindi in generale, in quest'ultimo caso, rientrano quelle tipologie di architetture nelle quali viene inserito un modulo Bluetooth per estenderne la connettività.

4.1.2 Topologie di rete

Un dispositivo Bluetooth Low Energy può comunicare con gli altri dispositivi in 2 modi: *modalità broadcasting* o *modalità connessione*. Ogni meccanismo ha i suoi pro e contro, ma entrambi sono stabiliti dalle linee guida del *Generic Access Profile* (GAP). In modalità ***broadcasting*** le informazioni vengono inviate a coloro che stanno effettuando uno *"scanning"* o che sono in ascolto (Figura 4.3).

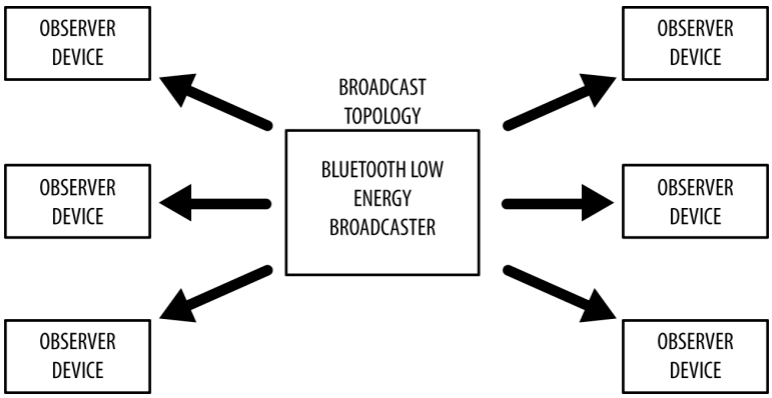

Figura 4.3 Topologia *Broadcast* [**42**]

I dati vengono pertanto inviati solo in una direzione: il *broadcaster* invia periodicamente gli *advertising packets* a tutti gli osservatori abilitati a riceverli, mentre i dispositivi *osservatori* effettuano un continuo scanning (alle frequenze preimpostate) per ricevere tali pacchetti.
La modalità ***connessione*** è invece impiegata quando è necessario trasmettere dati in entrambe le direzioni, oppure quando la mole di dati da trasmettere è maggiore rispetto a quella massima consentita da un pacchetto advertising, caratterizzato da un payload di 31 byte. Inoltre tutti i dati sono privati, vale a dire che essi vengono inviati e ricevuti soltanto dai due dispositivi tra loro connessi. In modalità connessione sono definiti due ruoli: *central* e *peripheral.*
Il dispositivo ***central***, che ha il ruolo di *master*, effettua uno scanning ripetuto delle frequenze in modo da rilevare eventuali *advertising packet* e inizializzare una connessione. Una volta che la connessione è stata inizializzata, il *central* gestisce anche i vari timing e gli scambi di dati.
Il dispositivo ***peripheral***, che ha il ruolo di *slave*, manda periodicamente gli *advertising packet* e accetta connessioni entranti. Una volta che la connessione è attiva, il *peripheral* segue il timing dettato dal *central* e scambia dati con lui.
Per inizializzare una connessione, il central rileva gli advertising packet del peripheral e manda una richiesta per instaurare una connessione privata tra i due. Quando la connessione è instaurata, il peripheral smette di inviare gli advertising packet e i due dispositivi possono iniziare a comunicare tra loro in entrambe le direzioni, come in Figura 4.4.
Va inoltre specificato che, sebbene sia il dispositivo central a provvedere all'instaurazione della connessione, i dati possono essere scambiati in entrambe le direzioni e i ruoli non impongono alcuna restrizione al throughput o alla priorità.

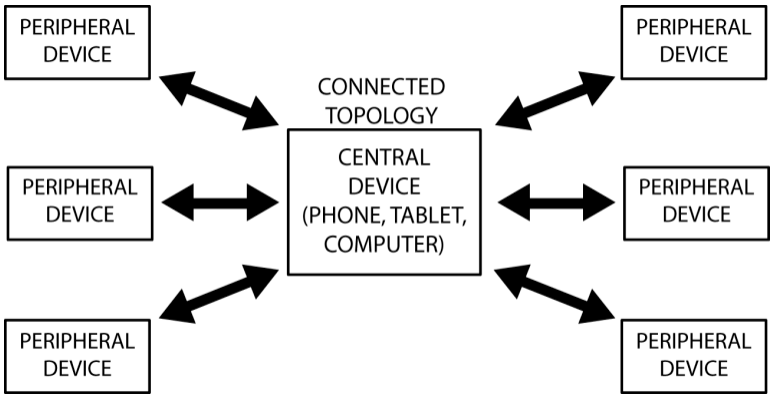

Figura 4.4 Modalità connessione [**42**]

Dalla versione 4.1, ogni restrizione sui ruoli è stata rimossa, ovvero si può avere una combinazione di ruoli:

- Un dispositivo può agire come central o peripheral allo stesso tempo,
- Un central può essere connesso a più peripheral,
- Un peripheral può essere connesso a più central.

4.1.3 Protocolli e profili

Fin dalla prima versione, le specifiche del Bluetooth definiscono, con una netta separazione, due nozioni molto importanti: protocolli e profili (Figura 4.5).
I **protocolli** sono i "mattoni" che servono a dotare un dispositivo di connettività Bluetooth. In altre parole, tutti i dispositivi Bluetooth devono attenersi a certe specifiche costruttive, e quindi essere dotati di tutti i livelli necessari che implementano i differenti formati di pacchetto (ad ogni livello): routing, multiplexing, encoding, decoding.
I **profili** sono invece dei "pilastri" verticali che attraversano tutto lo stack e che coprono o le modalità operative di base, GAP e GATT, o usi specifici come *Proximity Profile* o *Glucose Profile*.

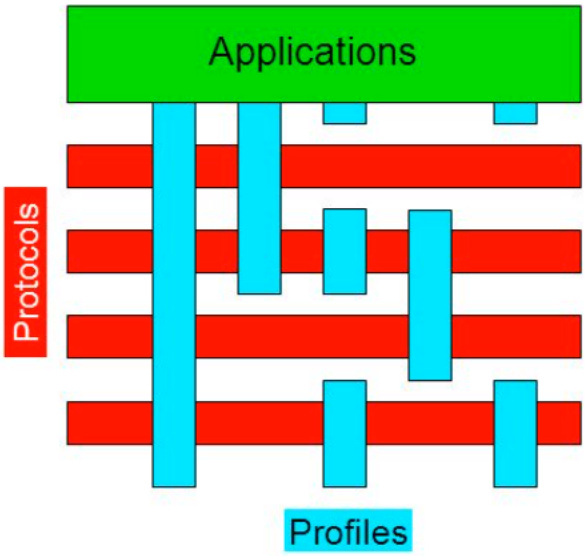

Figura 4.5 Protocolli e profili [**43**]

I profili definiscono, essenzialmente, come i protocolli vanno usati e come devono interagire tra loro per un obiettivo ben preciso. Nella pratica essi corrispondono ai servizi offerti da un dispositivo ed è responsabilità del produttore dotare il proprio dispositivo del profilo Bluetooth necessario a svolgere una determinata operazione. Esempi di profili sono:

- *Generic Access Profile* (GAP)
- *Service Discovery Application Profile*
- *Serial Port Profile* (SPP)
- *Headset Profile*
- *Hands Free Profile*

Senza i profili l'interoperabilità tra dispositivi di diversi produttori sarebbe impossibile; quindi essi rappresentano uno standard di utilizzo per garantire l'interoperabilità.

4.1.4 GAP e GATT

Il Generic Access Profile (GAP) è quel profilo che serve a rendere visibile il dispositivo e a determinare come i dispositivi possono o non possono interagire tra loro. Definisce quindi i ruoli di *central* e *peripheral*.
Esso è inoltre responsabile del processo di *advertising* e di *scanning*. Affinché un dispositivo peripheral sia visibile esso dovrà trasmettere obbligatoriamente i pacchetti di advertising; i pacchetti di scanning *(scan response payload)* sono invece opzionali in quanto sta al dispositivo *central* richiedere o meno la connessione.
Una volta che la connessione tra il *central* e il *peripheral* è instaurata, la fase di advertising viene interrotta, non vengono inviati più pacchetti di advertising e si passa all'uso dei servizi e delle caratteristiche messe a disposizione dal GATT per comunicare in entrambe le direzioni.
Oltre ai ruoli definiti dal GAP (*central* e *peripheral*), esistono anche i ruoli di *server* e *client* definiti invece dal GATT e che sono indipendenti dai primi due. Il dispositivo che mette i dati a disposizione è il *server*, mentre quello che vi accede è il *client*. Il server GATT organizza i dati in ciò che viene detta *attribute table* che altro non è che l'insieme degli **attributi** contenenti i dati.

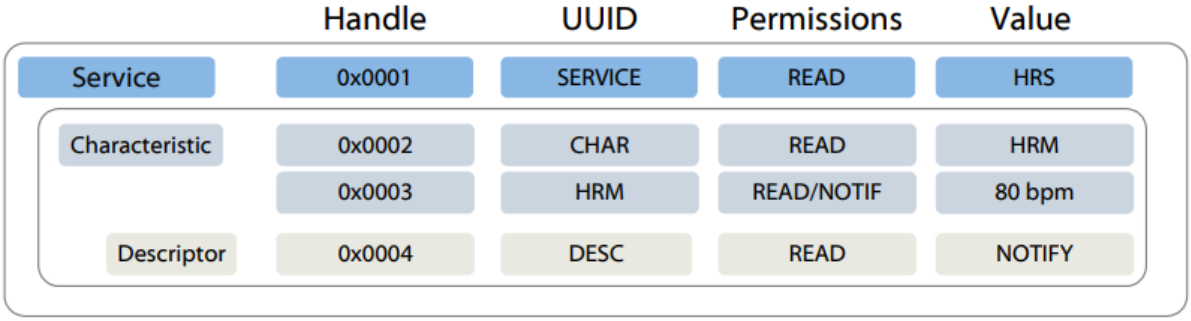

Figura 4.6 GATT Table [**44**]

Servizi, caratteristiche e descrittori sono tutti tipi di attributi. Ogni attributo ha un handle, un UUID e un valore. L'handle è l'indice dell'attributo nella GATT Table (Figura 4.6) ed è univoco; l'UUID contiene le informazioni sul tipo di dato dentro l'attributo e quindi serve a comprendere il valore del dato.

Una **caratteristica** occupa il livello più basso nelle transizioni GATT. Ogni caratteristica incapsula un singolo dato (sebbene possa anche contenere un array facente riferimento a più dati, ad esempio i valori di accelerazione lungo gli assi x, y e z forniti da un accelerometro triassiale). È un concetto molto importante poiché nel Bluetooth Low Energy l'utente interagisce proprio con le caratteristiche. Esse possono essere usate in lettura o scrittura, infatti per comunicare con la periferica BLE, e quindi mandare ad essa i dati, non si fa altro che un'operazione di scrittura sulla caratteristica. Ad esempio, è possibile implementare un servizio di trasmissione seriale dedicato (sfruttando ad esempio l'UART) e associare ad esso una caratteristica per la trasmissione e una per la ricezione: la prima avente i privilegi di sola lettura (da parte del client) e la seconda i privilegi di scrittura [45].
Un **servizio** consiste in una o più caratteristiche. I servizi racchiudono funzionalità tra loro collegate, ad esempio le letture dei valori forniti da un sensore e le relative impostazioni. Un servizio standard è il Heart Rate Service che contiene 3 caratteristiche: *Heart Rate Measurement, Body Sensor Location* e *Heart Rate Control Point.*
Un **profilo** è l'insieme di uno o più servizi. Esso fornisce sia informazioni sui servizi che contiene, ma anche informazioni utili alla comunicazione, come ad esempio il ruolo GAP e GATT che i due dispositivi devono assumere durante lo scambio di dati (informazioni come *advertising*, *connection interval*, ecc.) [44].
La Figura 4.7 fornisce un riepilogo illustrativo sulla gerarchia del GATT.

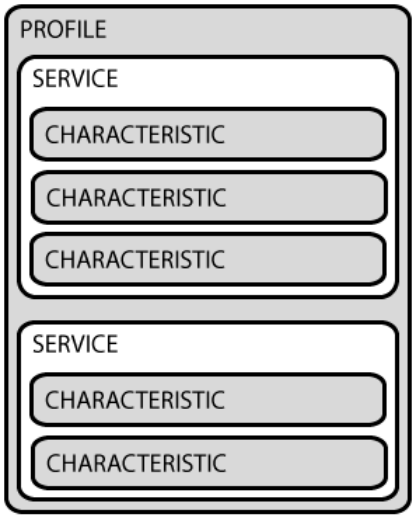

Figura 4.7 Gerarchia GATT **[45]**

4.2 Implementazione della connessione

Come visto, i dati da trasmettere sono quelli provenienti da sensori di vario tipo e di conseguenza l'informazione da trasmettere va opportunamente organizzata per poter poi essere compresa e processata in fase di ricezione.
Uno dei profili di funzionamento definiti dallo standard Bluetooth è il *Serial Port Profile* (SPP). Quest'ultimo nasce dall'dea di rimpiazzare con una connessione Bluetooth una comunicazione che altrimenti sarebbe stata cablata (Figura 3.1). L'SPP definisce dunque l'insieme dei protocolli e delle procedure da impiegare per emulare una connessione con cavo (ad esempio RS232 o UART) e quindi per la realizzazione di una cosiddetta *Virtual Port COM.*

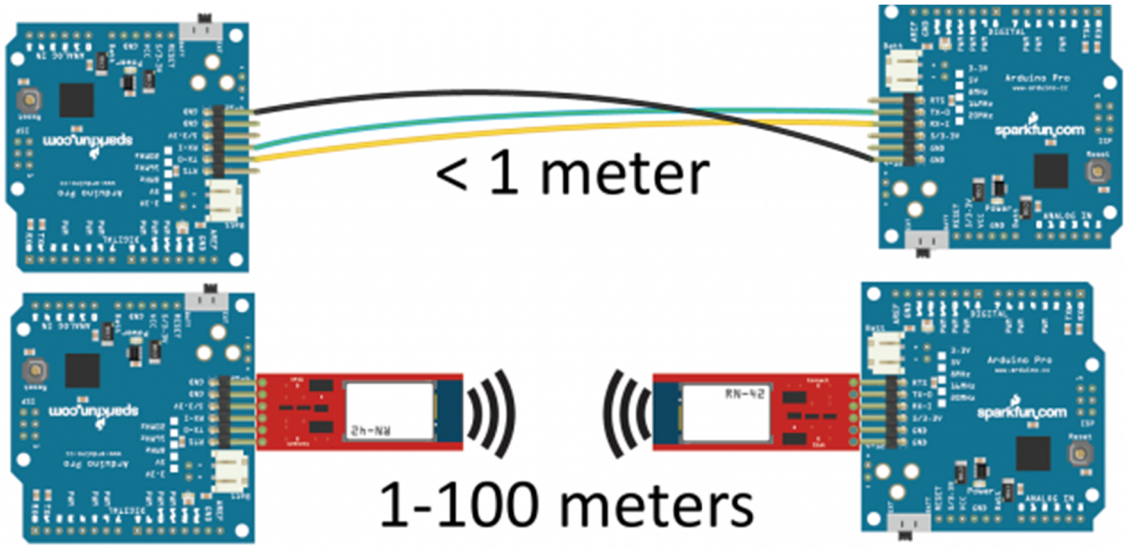

Figura 4.8 Da una connessione seriale cablata a una comunicazione mediante SPP [**46**]

In un microcontrollore (e relativo modulo Bluetooth) in cui viene abilitato il profilo SPP per il trasferimento seriale dei dati ad un altro dispositivo viene inizializzata la periferica UART. Tutta la comunicazione si basa pertanto sulla trasmissione e ricezione dei dati tramite interfaccia UART: i dati da inviare vengono immessi sull'UART che provvederà a passarli al modulo Bluetooth, viceversa i dati ricevuti tramite Bluetooth vengono immessi sull'interfaccia UART per essere prelevati e letti. Con questo meccanismo il modulo Bluetooth fa soltanto da tramite tra le due interfacce seriali UART (una per ogni dispositivo) e quindi ogni microcontrollore che invia e riceve è come se fosse collegato all'altro serialmente.
I vantaggi di questo tipo di comunicazione, oltre a tutti quelli introdotti da una connessione Bluetooth, sono quelli di poter usare un'interfaccia seriale ben nota come l'UART e quindi di usufruire ad esempio dei segnali RTS/CTS (*Request To Send/Clear To Send*) per il controllo di flusso [47].
Tuttavia, il Bluetooth Smart (o Bluetooth LE), e quindi il Bluetooth 5, non implementa l'SPP, come facilmente visibile in Figura 4.1. È importante comprendere infatti che nel BLE tutto il supporto dei profili e dei servizi è a livello applicativo, a differenza dello standard classico in cui il supporto è nativo dello stack adoperato. Un servizio di questo tipo va quindi implementato, ovvero va creato un servizio personalizzato.
La casa produttrice del microcontrollore mette a disposizione un servizio proprietario chiamato "*Nordic UART Service*" (NUS). Per quanto appena detto, questo servizio è incompatibile con qualsiasi altro dispositivo Bluetooth che non lo implementa, motivo per cui, pur avendo provato a usare il Bluetooth già presente nel PC, o un dongle USB, non è possibile ricevere i dati che la scheda invia. In altre parole, dopo che la connessione è stata implementata sulla scheda, tutti i dispositivi compatibili con il Bluetooth Low Energy sono in grado di rilevarla e connettersi, ma non sono in grado di comunicare tra loro. A differenza infatti dell'SPP, non è possibile usare una *Virtual Port COM* dedicata.
Si sottolinea il fatto che per rilevare la scheda occorre avere un dispositivo Bluetooth LE e quindi tutto lo stack che lo implementa, anche a livello di sistema operativo, motivo per cui la scheda non è rilevabile usando Windows 7. Il Bluetooth LE è supportato da Windows 8.1 in poi.

Visti i concetti di dispositivo *central* e *peripheral,* precedentemente esposti al paragrafo 4.1.2, si intuisce che la scheda installata sul guanto svolge il ruolo di *peripheral*, mentre il dispositivo che si connette e riceve i dati da essa svolge il ruolo di *central.*
L'implementazione della connessione sulla scheda prototipo è partita quindi dall'analisi dell'esempio `ble_app_uart` presente nell'SDK dentro la cartella *examples→ble_peripheral.* Il *Nordic UART Service* emula il *Serial Port Profile* usando l'UART come periferica seriale. In particolare le due funzioni che realizzano la ricezione e la trasmissione (anche se in seguito sono state modificate allo scopo) sono `nus_data_handler(ble_nus_evt_t * p_evt)` e `uart_event_handle(app_uart_evt_t * p_event)`. La prima preleva il dato ricevuto via Bluetooth e lo immette sull'UART, mentre la seconda preleva tutto ciò che viene inviato sull'UART e lo manda via Bluetooth tramite la funzione `ble_nus_data_send`, presente all'interno della ISR dell'UART. È importante ricordare che la scheda non deve essere solo capace di inviare i dati acquisiti via Bluetooth, ma anche ricevere i parametri passati da parte dell'utente al momento dell'inizializzazione, ovvero nella fase in cui si seleziona il training da effettuare.
Il NUS è un servizio basato su GATT avente due caratteristiche: una in trasmissione e una in ricezione. Il dato ricevuto dal peer viene passato all'applicazione e quello ricevuto dall'applicazione viene passato al peer. Va inoltre precisato che i caratteri che esso accetta sono i caratteri ASCII.
Per poter usare tutto lo stack Bluetooth, e quindi anche il NUS, è necessario programmare prima il *SoftDevice.*

4.2.1 SoftDevice

Il *SoftDevice* è una libreria precompilata che implementa tutto lo stack (Figura 4.9) per il protocollo di comunicazione wireless, sviluppato dalla Nordic Semiconductor.

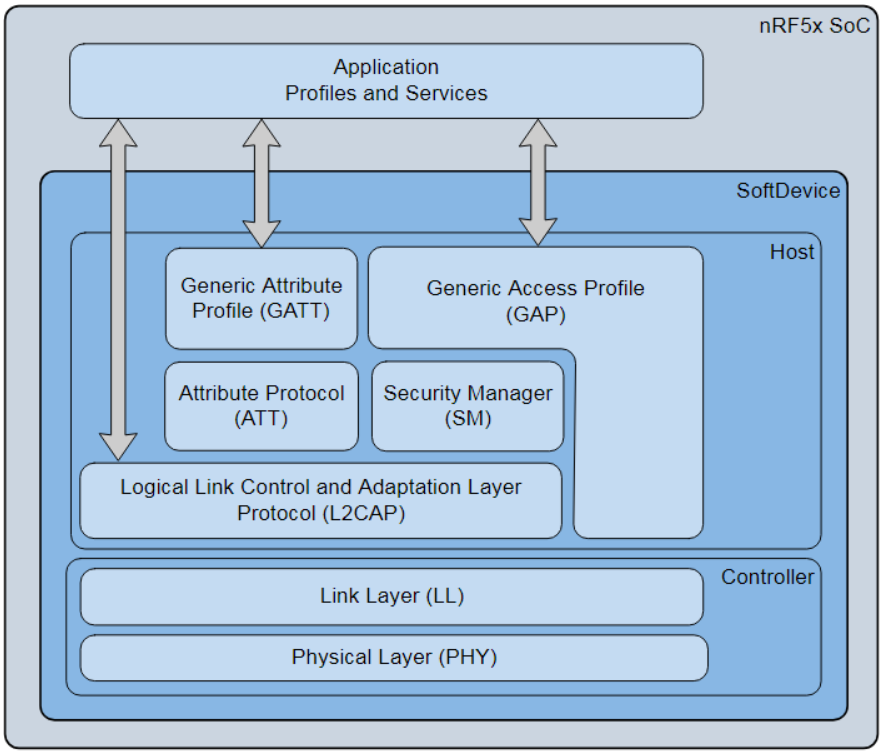

Figura 4.9 *Stack SoftDevice* **[38]**

In particolare il *SoftDevice* s132 contenuto nell'SDK fornisce tutto lo stack per il BLE sia *central* che *peripheral.*
Per inserire il SoftDevice sulla scheda è necessario usare il software nRFgo Studio. Una volta connessa la scheda al Development Kit per la programmazione e quest'ultimo via USB al PC, è possibile aprire il programma in questione.
A sinistra, nella finestra del programma, all'interno della sezione *Device Manager,* selezionare la scheda di sviluppo, come illustrato in Figura 4.10.

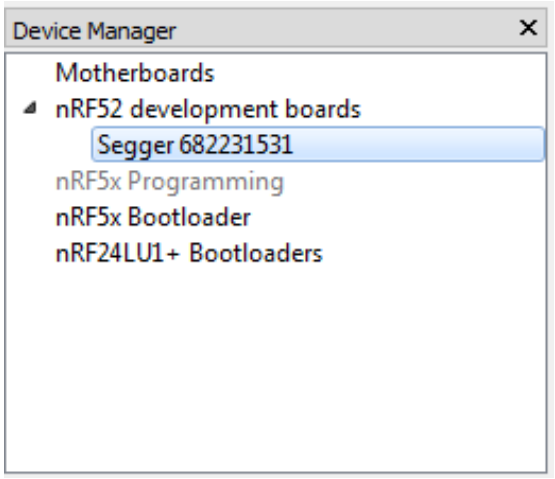

Figura 4.10 Device Manager – nRFgo Studio

Come si nota in Figura 4.11 si apre il riquadro che indica lo stato della memoria interna e l'interfaccia per la programmazione. Selezionare la voce "*Browse*" e indicare il file del SoftDevice s132 con estensione `.hex` fornito con l'SDK, per l'esattezza il file `s132_nrf52_6.1.0_softdevice.hex`, all'interno del percorso:
components→softdevice→s132→hex.

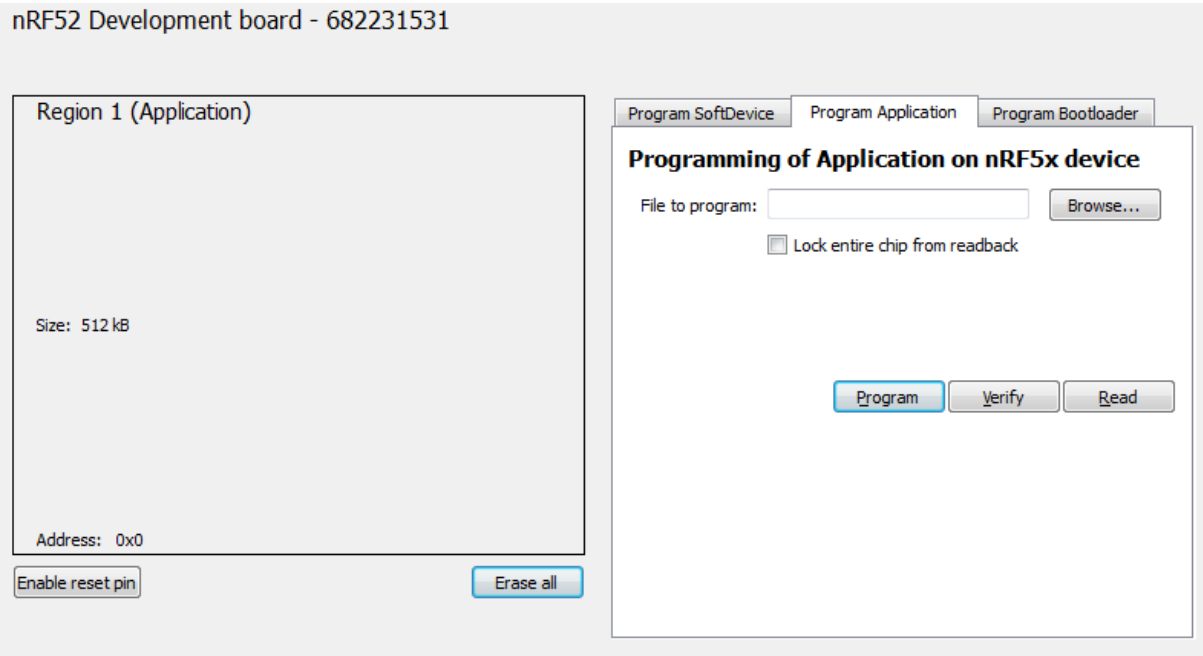

Figura 4.11 Memoria interna e programmazione SoftDevice

Cliccando su "*Program*", la scheda viene programmata con il seguente risultato (Figura 4.12):

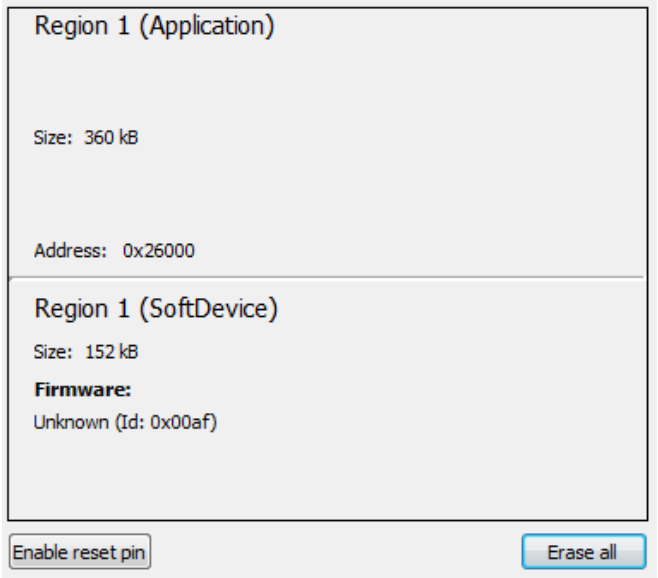

Figura 4.12 SoftDevice programmato in memoria

Come si nota in Figura 4.9, l'applicazione sviluppata dall'utente è indipendente dal SoftDevice e andrà ad occupare la parte sopra in Figura 4.12.
Come riportato in Figura 4.13, utilizzando il SoftDevice alcune periferiche non sono più disponibili perché servono per il corretto funzionamento dello stesso; motivo per il quale il codice dell'ADC, precedentemente scritto, è stato modificato utilizzando il TIMER1 anziché il TIMER0.

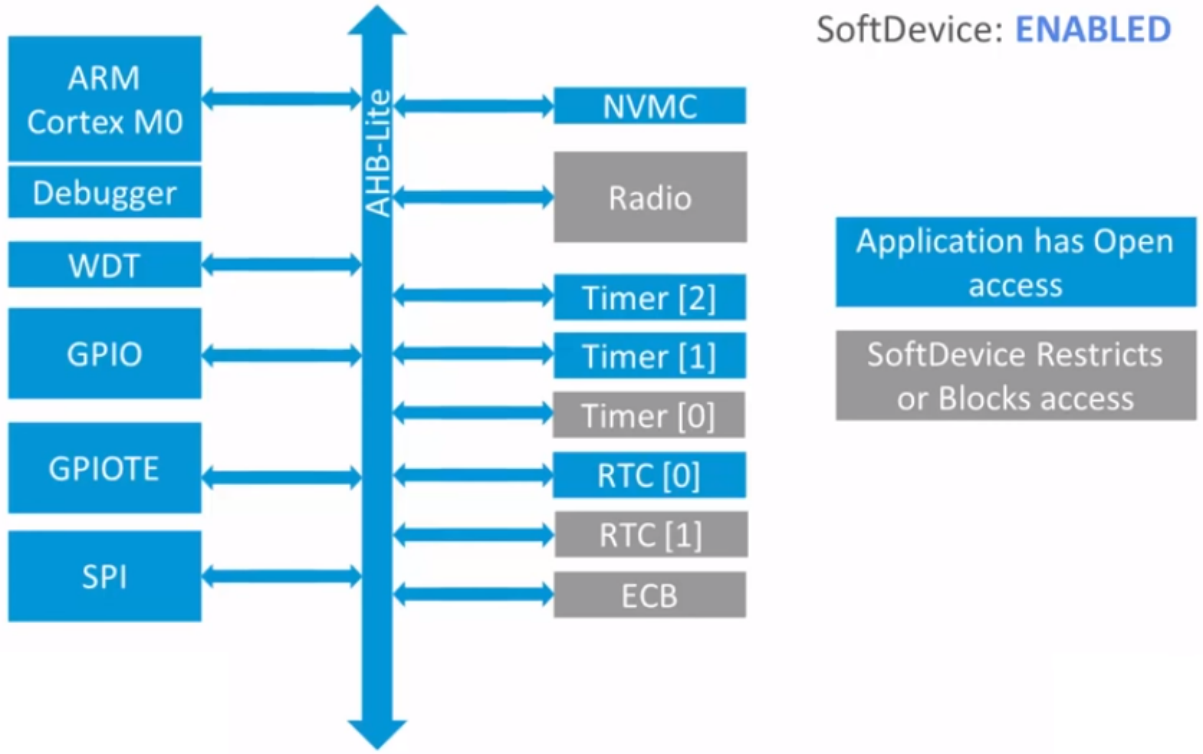

Figura 4.13 SoftDevice HW Block Protection [**48**]

4.3 Buffer di trasmissione e ricezione dati

Come si è già detto in precedenza, la scheda deve sia trasmettere i dati ma anche riceverli in modo da impostare il training e le varie configurazioni, motivo per cui va opportunamente gestita l'informazione da tramettere e l'informazione ricevuta. Il NUS mette a disposizione due caratteristiche, una per trasmettere e una per ricevere:

- *RX Characteristic:* il peer connesso (il PC) può mandare i dati alla peripheral scrivendo su questa caratteristica del servizio; la peripheral li trova scritti e li legge come dati ricevuti;
- *TX Characteristic:* se il peer ha abilitato le notifiche per questa caratteristica, allora l'applicazione può mandare i dati (che la peripheral scrive su questa caratteristica) sotto forma di notifiche; quindi il peer preleva da questa caratteristica i dati che la peripheral manda.

Il dato ricevuto sulla scheda viene processato nell'handler `nus_data_handler,` in particolare quando scatta l'evento `BLE_NUS_EVT_RX_DATA`. In tal caso il buffer che contiene il dato appena ricevuto è `p_evt->params.rx_data.p_data`.
Per quanto riguarda invece il buffer da inviare, esso va di volta in volta scritto in opportune parti di codice dopo aver fatto la lettura del sensore. L'idea di base è quella di leggere il sensore, costruire il buffer e inviarlo, quindi inviare i dati acquisiti in tempo reale. Nulla vieta di inserire all'interno dello stesso buffer di trasmissione dati provenienti da più sensori, l'importante è saperli distinguerli in modo da interpretarli di conseguenza. In tale procedimento va tenuta in considerazione la massima dimensione del buffer che il NUS riesce ad inviare e ricevere.
Analizzando innanzitutto il comportamento di base del NUS si vede che è definita la funzione `ble_nus_data_send`, in particolare:

```
1. uint32_t ble_nus_data_send(ble_nus_t * p_nus,
2.                            uint8_t   * p_data,
3.                            uint16_t  * p_length,
4.                            uint16_t    conn_handle);
```

Tale funzione invia una stringa, che nel linguaggio C corrisponde ad un array di caratteri (`char`), quindi nell'usare tale funzione quello che bisognerà fare è andare a creare una stringa, la stringa che verrà inviata. Il secondo argomento della funzione sopra (`uint8_t *p_data`) è proprio quello che punta alla stringa da passare.
Andando a vedere nelle varie define si capisce che il massimo numero di caratteri è 244, ovvero 244 `char` (o `uint8_t` che è la stessa cosa). Si ottiene questo valore perché è definito come:

BLE_NUS_MAX_DATA_LEN =	NRF_SDH_BLE_GATT_MAX_MTU_SIZE -	247
	OPCODE_LENGTH -	1
	HANDLE_LENGTH =	2
		244

In `bluetooth.c` è stato dunque definito il buffer (riga 1 sotto) in cui scrivere i dati dei sensori e da passare alla funzione `ble_nus_data_send`:

```
1. char buf[BLE_NUS_MAX_DATA_LEN+1] = {0};
2. uint16_t data_length;
```

Esso non è altro che un array di tipo `char` di dimensione `BLE_NUS_MAX_DATA_LEN+1`. Bisogna infatti ricordare che nel linguaggio C ogni stringa ha il carattere terminatore '`\0`', questo sta a significare che i caratteri che possono essere inviati, cioè i caratteri di cui è composta la stringa e che il NUS riesce a inviare, sono pari a `BLE_NUS_MAX_DATA_LEN`, ma occorre considerare il fatto che ogni stringa ha bisogno del terminatore. In questo modo si evita di generare una stringa di 245 caratteri e salvarla in `buf`; per farlo `buf` dovrebbe essere lungo 246 "caselle". Al tempo stesso si fa in modo di rispettare la massima lunghezza della stringa che il NUS può inviare.
Questa soluzione, seppur vantaggiosa dal punto di vista della praticità, ha anche i suoi svantaggi. Definire infatti un array in questo modo, significa infatti allocare staticamente la memoria che lo deve contenere e, sicuramente, il più delle volte non sarà necessario adoperare un buffer così lungo; in altre parole, difficilmente viene mandata una stringa così lunga. Sarebbe bene minimizzare l'uso della memoria e lasciarla libera laddove servirebbe e quindi definire un buffer di dimensione ragionevolmente più corta o effettuare un'allocazione dinamica. Tuttavia, di memoria ce n'è abbastanza e ci si può permettere di lasciare `buf` di queste dimensioni, anche per sviluppi futuri nel caso in cui si voglia trasmettere con una sola stringa molteplici dati (ad esempio sensori tutti attivi).
Nel codice sopra si vede come `buf` viene inizializzato come un array vuoto, e alla seconda riga si nota che viene definita anche la variabile `data_length` di tipo `uint16_t` (cioè un *unsigned short*) la cui utilità si vedrà tra poco.
Come detto in precedenza, i dati vengono inviati sull'UART e all'interno dell'interrupt dell'UART (`uart_event_handle`) vengono inviati via bluetooth tramite la funzione `ble_nus_data_send`. Tale comportamento è stato modificato, vale a dire che non è necessario inviare prima i dati acquisiti sull'UART e poi da lì inviarli via Bluetooth. Un simile comportamento è stato definito dal NUS per emulare l'SPP che fa uno delle interfacce seriali da ambo le parti della comunicazione.
Per questo motivo, anziché usare le istruzioni interne a `uart_event_handle` per inviare i dati, è stata definita la funzione `Send_Bluetooth` come segue:

```
1. void Send_Bluetooth(char *buf)
2. {
3.  data_length = strlen(buf);
4.  uint32_t  err_code;
5.  do
6.  {
7.   err_code = ble_nus_data_send(&m_nus, buf, &data_length, m_conn_handle);
8.   if ((err_code != NRF_ERROR_INVALID_STATE) &&
9.       (err_code != NRF_ERROR_RESOURCES) &&
10.      (err_code != NRF_ERROR_NOT_FOUND))
11.     {
12.        APP_ERROR_CHECK(err_code);
13.     }
14.  } while (err_code == NRF_ERROR_RESOURCES);
15. }
```

La funzione prende in ingresso il buffer, ne calcola la lunghezza (terza riga) con la funzione `strlen(buf)` salvandola in `data_length` e passa entrambi alla funzione

`ble_nus_data_send`. `Strlen` infatti calcola il valore della lunghezza della stringa, escluso il terminatore '`\0`', ovvero il numero di caratteri presenti (compresi gli spazi) prima del terminatore.
In questo modo, `ble_nus_data_send` non invierà tutto il buffer di 244 caratteri (molti dei quali neanche presenti). Si noti che era anche possibile definire `data_length` all'interno della funzione `Send_Bluetooth` e non fuori come invece è stato fatto. È stata fatta questa scelta per sperimentare parti di codice laddove poteva tornare utile tale variabile.
Resta da costruire il buffer `buf` da inviare. A titolo di esempio, si riportano di seguito le istruzioni relative al convertitore analogico digitale per i sensori di forza e di flessione, in particolare l'ISR dell'ADC.

```
1.  static void saadc_callback(nrf_drv_saadc_evt_t const * p_event)
2.  {
3.      if (p_event->type == NRF_DRV_SAADC_EVT_DONE)
4.      {
5.          ret_code_t err_code;
6.
7.          err_code =
8.              nrf_drv_saadc_buffer_convert(p_event->data.done.p_buffer,
9.                                           SAMPLES_IN_BUFFER);
10.         APP_ERROR_CHECK(err_code);
11.
12.         sprintf(buf,
13.                 "%d %d %d %d %d %d %d %d %d\r",
14.                 ADC_ADDR,
15.                 p_event->data.done.p_buffer[0],
16.                 p_event->data.done.p_buffer[1],
17.                 p_event->data.done.p_buffer[2],
18.                 p_event->data.done.p_buffer[3],
19.                 p_event->data.done.p_buffer[4],
20.                 p_event->data.done.p_buffer[5],
21.                 p_event->data.done.p_buffer[6],
22.                 p_event->data.done.p_buffer[7]);
23.
24.         Send_Bluetooth(buf);
25.     }
26. }
```

Si nota come all'interno dell'ISR dell'ADC è presente un `if` che controlla il verificarsi dell'evento `NRF_DRV_SAADC_EVT_DONE`. Tale evento si verifica ogni qualvolta il buffer dell'ADC viene riempito, cioè quando 8 campioni sono stati acquisiti. La conversione viene effettuata alla riga 8 dall'apposita funzione e successivamente, dalla riga 12 in poi, viene creato il buffer per la trasmissione. La funzione utile allo scopo è `sprintf`. Essa converte, nell'ordine riportato nel codice, i valori di `ADC_ADDR`, `p_event->data.done.p_buffer[0], …, p_event->data.done.p_buffer[7]` da valori interi (`%d`) a stringa (cioè sequenza di caratteri) e li dispone all'interno di `buf` sempre nello stesso ordine. In questo modo, il buffer `buf` che si invierà successivamente con la funzione `Send_Bluetooth(buf)` alla riga 24 sarà una stringa, come d'altronde richiesto dalla funzione `ble_nus_data_send`. Inoltre, quando si invia una stringa contenente dei dati, affinché essa venga interpretata correttamente in ricezione, è necessario specificare di che dati si tratta, cioè da quale sensore provengono; motivo per il quale la prima informazione contenuta in `buf` è sempre quella relativa al tipo di sensore. Nel caso

dell'ADC viene inserito `ADC_ADDR`, cioè un identificativo (ID) che indica, in ricezione, che i dati presenti in `buf` sono quelli acquisiti dal convertitore. In ricezione, si andrà a leggere la parte di stringa di `buf` prima del primo spazio e si interpreteranno i caratteri successivi (al primo spazio) di conseguenza.
Un comportamento simile è stato realizzato anche per i per i sensori digitali. I sensori digitali sono stati programmati per restituire il dato che essi acquisiscono solo quando vengono interrogati dal microcontrollore (in sviluppi futuri ciò non toglie la possibilità di implementare una conversione automatica e abilitare un interrupt per prelevare i dati disponibili). In questa sede, è stato scritto dunque un codice (`scan_digital_sensor.c` e `scan_digital_sensor.h`) che effettua una scansione dei sensori digitali, uno per uno. È stato abilitato il `TIMER2` e definito uno `SCAN_RATE`; più avanti si riporterà e verrà opportunamente commentata l'ISR del `TIMER2`.
I sensori da abilitare e di cui fare uno scanning dipendono dal tipo di training che si intende effettuare, motivo per cui è importante definire come interpretare i dati che si ricevono sulla scheda, ovvero la configurazione iniziale definita dall'utente. Si riporta di seguito la funzione che gestisce i dati in ricezione (`nus_data_handler`):

```
1.  void nus_data_handler(ble_nus_evt_t * p_evt)
2.  {
3.      if (p_evt->type == BLE_NUS_EVT_RX_DATA)
4.      {
5.          NRF_LOG_HEXDUMP_DEBUG(p_evt->params.rx_data.p_data,
6.                                p_evt->params.rx_data.length);
7.
8.          if(strstr(p_evt->params.rx_data.p_data, "pinching") != NULL)
9.          {
10.             adc_init();
11.         }
12.         else if(strstr(p_evt->params.rx_data.p_data, "grasping") != NULL)
13.         {
14.             adc_init();
15.         }
16.         else if(strstr(p_evt->params.rx_data.p_data, "wrist") != NULL)
17.         {
18.             twi_init();
19.             LSM6DS0_configuration();  //imu init
20.             LIS3MDL_configuration();  //magnetometer init
21.             TimerScanDigitalSensors_Init();
22.         }
23.         else if(strstr(p_evt->params.rx_data.p_data, "reaching") != NULL)
24.         {
25.             adc_init();
26.         }
27.         else if(strstr(p_evt->params.rx_data.p_data,"following") != NULL)
28.         {
29.             twi_init();
30.             LSM6DS0_configuration();
31.             TimerScanDigitalSensors_Init();
32.         }
33.         else if(strstr(p_evt->params.rx_data.p_data,
34.                                "physiological") != NULL)
35.         {
36.             twi_init();
37.             MAX30105_power_up();    //pulse-oximeter init
38.             gsr_init();
39.         }
```

```
40.         else if(strstr(p_evt->params.rx_data.p_data, "deinit") != NULL)
41.         {}
42.     }
43. }
```

Come si nota dalla riga 3, il NUS controlla quando un nuovo dato è stato ricevuto tramite l'evento `BLE_NUS_EVT_RX_DATA`. Così come quando si inviano dati dalla scheda, essi devono essere ben interpretati dal peer che li riceve, analogamente i parametri che l'utente manda devono essere ben definiti in modo che la scheda li comprenda. Motivo per il quale sono state definite delle stringhe ben precise. Ad esempio, se il training che si vuole fare è il *pinching*, l'interfaccia utente dovrà inviare la stringa "`pinching`". Era possibile definire ad esempio anche un numero e associarlo al particolare tipo di training, ma si è preferito indicare esattamente il nome del training per semplicità di lettura del codice.
Tramite una serie di `if else` annidati e tramite la funzione `strstr`, si determina il training da andare ad eseguire. Infatti, considerando ad esempio il training relativo al pinching, la parola "*pinching*" viene cercata all'interno del buffer ricevuto, o meglio della stringa ricevuta, cioè `p_evt->params.rx_data.p_data`. Qualora l'esito sia positivo, ovvero se la parola "*pinching*" è presente all'interno del buffer ricevuto, la funzione `strstr` restituisce il puntatore alla prima occorrenza trovata; mentre se l'esito è negativo, cioè se non è presente la parola, allora restituisce `NULL`. Quindi la relativa abilitazione viene effettuata quando il risultato è diverso da `NULL`.
Il motivo per cui si è preferito usare `strstr` e non `strcmp` (che fa un confronto tra le stringhe) deriva dal fatto che l'esito di `strcmp` si ottiene se le due stringhe sono esattamente le stesse, compresi quindi i caratteri terminatori ('`\n`' e '`\r`'). In altre parole, affinché si abbia esito positivo nel confronto delle stringhe (e quindi inizializzazione delle periferiche) bisogna inviare esattamente la stringa per come è stata preimpostata. Questo comportamento si è verificato poco comodo in fase di debug dato che alcuni terminali seriali (come ad esempio *Termite* o *RealTerm*) inviano i caratteri terminatori, anche se l'impostazione può essere modificata. Al tempo stesso, dato che in una prima fase è stata usata l'app "nRF UART v2.0", messa a disposizione dalla Nordic Semiconductor, l'invio dei caratteri terminatori risultava scomodo.
Per far fronte a entrambe le situazioni e per favorire futuri sviluppi si è pensato che `strstr` facesse più al caso.
In realtà la situazione è un po' più complessa perché dipende anche dal codice implementato per inviare i dati verso la scheda del guanto, come si vedrà più avanti. Ad ogni modo conviene sempre inviare i caratteri terminatori.
In caso di successo nel confronto, vengono abilitate le periferiche per il training in questione. Ad esempio nel caso delle misure fisiologiche (`"physiological"`) viene inizializzato l'I2C per far funzionare il pulse-oximeter, inizializzato il pulse-oximeter e infine l'ADC per acquisire il GSR.
Altre stringhe non predefinite giungono alla scheda, vengono lette ma non hanno alcun effetto sul suo comportamento. È comunque giusto ricordare che l'interfaccia grafica utente provvederà ad inviare solo stringhe predefinite; in altre parole l'utente non può inviare dati a piacimento verso il guanto (non avrebbe senso).
Il comportamento di `nus_data_handler`, per come ora è definito, presenta però alcuni problemi legati alla inizializzazione delle periferiche. Infatti, nel caso in cui si voglia passare da un training a un altro bisogna stare particolarmente attenti. Se ad esempio l'I2C è stato già inizializzato nel training precedente eseguendo la funzione `twi_init()`,

non è possibile rieseguire la stessa funzione per il nuovo training, pena errore nel microcontrollore perché si trova a dover inizializzare una periferica già inizializzata. Il comportamento è del tutto analogo a quanto già detto sulle funzioni `adc_init()` e `gsr_init()` nel paragrafo 3.4.1. Ecco dunque che tra le possibili stringhe predefinite è stata prevista, alla riga 39, una relativa alla de-inizializzazione delle periferiche, che dovrà tenere conto di quali sono le periferiche già inizializzate. Anche qui, questo non è l'unico modo di risolvere tale problema; un'altra possibile soluzione potrebbe essere quella di lasciare inizializzate le periferiche, quindi senza eseguire una de-inizializzazione (anche se c'è il caso particolare del convertitore analogico-digitale in comune tra sensori di forza-flessione e GSR), ma semplicemente cambiare i sensori che vengono scansionati.
Essenzialmente, il motivo per cui conviene puntare a inizializzare e de-inizializzare le periferiche è il risparmio energetico, cioè usarle soltanto quando servono.

4.3.1 Risparmio energetico

Per comprendere le funzionalità del risparmio energetico e come si è fatto in modo di ridurre il più possibile il le operazioni svolte dalla CPU, si riporta sotto il codice presente nel file `main.c`.

```
1. #include "include.h"
2.
3. int main(void)
4. {
5.     bluetooth_init();
6.
7.     // Enter main loop.
8.     for(;;)
9.     {
10.
11.         NRF_LOG_FLUSH();
12.         idle_state_handle();
13.
14.     }//for (main loop)
15. }//main
```

Come di nota, quando la scheda viene accesa la prima cosa che viene eseguita è l'inizializzazione della connessione Bluetooth tramite la funzione `bluetooth_init()`. Successivamente, il microcontrollore entra nel main loop, cioè il ciclo infinito privo di qualsiasi condizione, e quindi sempre verificato, che mantiene il microcontrollore acceso. All'interno del `for(;;)` sono presenti due funzioni. La prima `NRF_LOG_FLUSH()` serve esclusivamente al modulo di log, ovvero la parte del microcontrollore che si occupa di emettere tutti i vari segnali utili allo sviluppatore in fase di debug e che pertanto non gioca alcun ruolo nel funzionamento finale del microcontrollore. Tale funzione potrà essere eliminata con la versione finale del firmware, come pure la disabilitazione di tutti i log che ora sono presenti nel codice.
La funzione che invece è di fondamentale importanza è la seconda: `idle_state_handle()`. Come dice il nome stesso, il microcontrollore entra in uno stato di *idle*, ovvero di inattività. Questa funzione è definita in `bluetooth.c` nel seguente modo:

```
1. void idle_state_handle(void)
2. {
3.     UNUSED_RETURN_VALUE(NRF_LOG_PROCESS());
4.     nrf_pwr_mgmt_run();
5. }
```

Lasciando perdere la riga 3, che implementa anch'essa funzionalità relative al log, alla riga 4 si trova la funzione `nrf_pwr_mgmt_run()` che gestisce il consumo di energia del microcontrollore, ovvero fa in modo che esso rimanga in uno stato di *idle,* riducendo quindi il consumo energetico e "sveglia" il microcontrollore soltanto quando necessario, ovvero quando ci sono operazioni da eseguire.
Infine, ma non meno importante, è la seguente considerazione: tutto il firmware del microcontrollore è stato concepito per eseguire istruzioni solo quando richiesto, ovvero quando si verifica qualche interrupt. Questo è il motivo per cui il *main.c* è costituito da poche righe di codice, ovvero si è fatto in modo che il microcontrollore non effettui alcun polling sui sensori e su alcuna variabile.
Un main così "alleggerito" unito all'impiego della funzione `idle_state_handle()` al suo interno, consentono di far rimanere il microcontrollore nello stato di idle il più tempo possibile (bisogna infatti intercalarsi nelle velocità dei dispositivi elettronici in questione, sebbene dal punto di vista umano sembrino tempi molto piccoli) e svegliare il microcontrollore solo quando si verifica un interrupt che richiede il suo intervento. A tal proposito, si noti infatti che per come è stato programmato il funzionamento del convertitore analogico-digitale, cioè utilizzando il PPI, l'intervento del microcontrollore è ulteriormente ridotto.

Capitolo 5

Prove di trasmissione, ricezione e analisi dati

Una volta implementata la connessione e il codice relativo alla creazione del buffer di trasmissione e ricezione e relativo invio, in questo capitolo vengono affrontate le varie strategie messe in atto e i vari tool software di cui si è fatto uso per testare la correttezza della comunicazione, simulare i vari training previsti e analizzare i dati ricevuti.
Il primo strumento che è stato usato per avere un rapido riscontro sulla connessione, nonché l'invio e la ricezione dei dati, è stata l'app nRF UART v2.0, messa a disposizione dalla Nordic Semiconductor per usare il proprio smartphone come terminale (Figura 5.1).
Nella figura di sinistra si vede come l'app trova varie periferiche Bluetooth nelle vicinanze, tra cui anche la scheda prototipo del guanto che è stata chiamata *"Glv EvlBrd"*. Cliccando sul nome viene instaurata la connessione (figura di sinistra); il codice che è stato scritto in questa fase è stato fatto in modo tale da far partire l'acquisizione dei sensori subito dopo la connessione.
I dati ricevuti provengono da vari sensori; il primo numero infatti corrisponde all'indirizzo del sensore, anche se sarebbe bene associare a ciascuno una lettera sola o un numero solo in modo da mandare meno byte con il vantaggio di ridurre la lunghezza del buffer o avere più spazio a disposizione per mandare dati di più sensori (con un solo invio). Per ora, come si nota sempre dalla Figura 5.1 a destra, ad ogni acquisizione segue un invio, eccezion fatta per il convertitore analogico digitale per il quale vengono inviati i dati dei 4 sensori di forza e i 4 di flessione (anche se si tratta comunque di un solo buffer ADC).
Di seguito si riporta il codice che è stato implementato per effettuare uno scanning dei sensori digitali:

```
1. static void timer_handler(nrf_timer_event_t event_type, void * p_context)
2. {
3.     if(event_type == NRF_TIMER_EVENT_COMPARE0)
4.     {
5.         static uint8_t num_sensor;
6.         switch(num_sensor)
7.         {
8.             case 0:
9.               accelerazione = LSM6DS0_read_accelerometer_data();
```

```
10.               sprintf(buf, "%d %c %d %d %d\r", IMU_ADDR, 'a',
11.                    accelerazione[0], accelerazione[1], accelerazione[2]);
12.             break;
13.
14.             case 1:
15.               giroscopio = LSM6DS0_read_gyroscope_data();
16.               sprintf(buf, "%d %c %d %d %d\r", IMU_ADDR, 'g',
17.                           giroscopio[0], giroscopio[1], giroscopio[2]);
18.             break;
19.
20.             case 2:
21.               temp_pulse_ox = MAX30105_read_temp();
22.               sprintf(buf, "%d %d %d\r", PULSE_OX_ADDR,
23.                                     temp_pulse_ox[0], temp_pulse_ox[1]);
24.             break;
25.
26.             case 3:
27.               temperatura = MAX30205_read_temp();
28.               sprintf(buf, "%d %d\r", TEMP_SENS_ADDR, temperatura);
29.             break;
30.         }
31.         num_sensor = (num_sensor+1)%4;
32.         Send_Bluetooth(buf);
33.     }
34. }
```

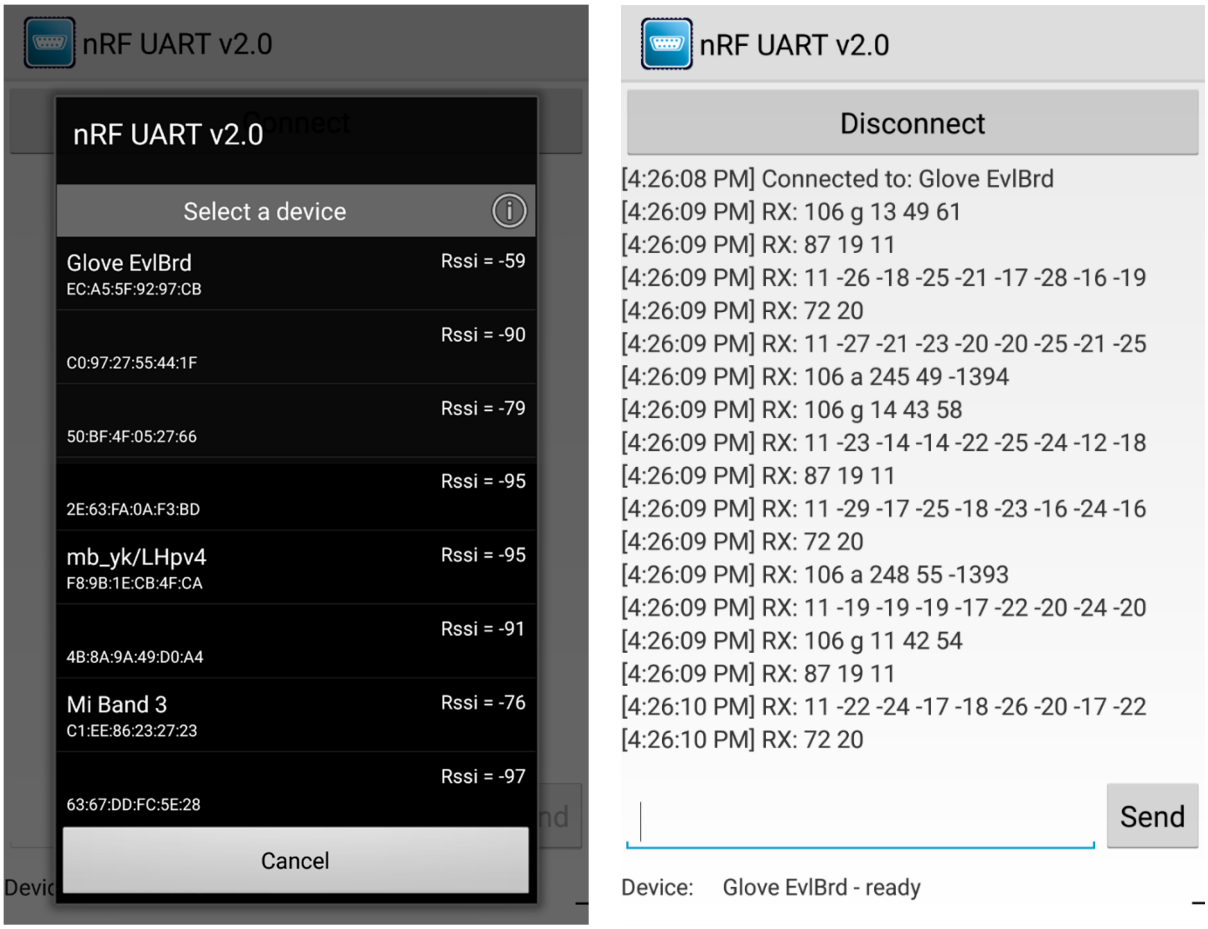

Figura 5.1 Ricezione dati su nRF UART v2.0

Il principio di funzionamento del codice è il medesimo di quello per l'ADC, dove viene letto il sensore digitale e i dati che restituisce (le funzioni sono state scritte in modo tale da restituire dati sotto forma di array) vengono inserite all'interno di `buf` per poi essere inviate con la funzione `Send_Bluetooth(buf)`. In tal caso però è stato abilitato il `TIMER2` il cui interrupt scatta sul *compare register 0* ogni `SCAN_RATE` millisecondi. `SCAN_RATE` è modificabile a piacere; per ora è stato impostato a 100 ms. Da notare che ogni 100 ms l'interrupt scatta, viene scansionato un sensore ogni volta differente e successivamente si procede all'invio. È chiaro che, per ora, in questa implementazione manca l'abilitazione dei sensori che si vuole scansionare in base al tipo di training e `buf` viene ogni volta riempito con dati di sensori diversi e poi inviato. L'implementazione migliore consiste nell'effettuare uno scanning di tutti i sensori abilitati per il training, inserire tutti i dati nello stesso buffer e procedere all'invio. Questa implementazione è più critica però dal punto di vista delle tempistiche, perché deve essere rispettato lo scan rate del timer per effettuare tutte queste operazioni. Vale la pena inoltre precisare che la funzione `sprintf` ogni volta che viene richiamata azzera `buf`, cioè se ad esempio nel frattempo scatta l'interrupt dell'ADC, contenente anch'esso la funzione `sprintf` su `buf`, non si hanno problemi di incongruenza dei dati perché `sprintf` azzera `buf` evitando che ci siano dati provenienti da diverse parti di codice.

5.1 Firmware per la ricezione

Lo scopo è chiaramente quello di inviare i dati al PC dove verranno memorizzati e analizzati. Come soluzione provvisoria per testare la connessione e il corretto invio e ricezione dei dati dei sensori (verso il PC) è stata adoperato il kit di sviluppo PCA10040, ovvero la scheda nella Nordic Semiconductor che viene usata per programmare la scheda del guanto.
Dopo aver resettato tutta la memoria interna con nRFgo Studio, l'esempio che è stato adoperato è ble_app_uart_c all'interno della cartella *examples→ble_central* del Software Development Kit.
L'esempio è già pronto per poter funzionare ma si riporta di seguito qualche informazione utile a comprenderne il funzionamento, in particolare si riporta parte della routine `uart_event_handle` per l'invio dei dati verso la scheda del guanto:

```
1.  void uart_event_handle(app_uart_evt_t * p_event)
2.  {
3.      static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
4.      static uint16_t index = 0;
5.      uint32_t ret_val;
6.
7.      switch (p_event->evt_type)
8.      {
9.          /**@snippet [Handling data from UART] */
10.         case APP_UART_DATA_READY:
11.             UNUSED_VARIABLE(app_uart_get(&data_array[index]));
12.             index++;
13.
14.             if ((data_array[index - 1] == '\n') ||
15.                 (index >= (m_ble_nus_max_data_len)))
16.             {
17.                 NRF_LOG_INFO("Ready to send data over BLE NUS");
```

```
18.                 NRF_LOG_HEXDUMP_DEBUG(data_array, index);
19.
20.             //PARTE DI INVIO CON IL BLUETOOTH
```

L'interrupt dell'UART scatta quando qualcosa viene immesso sull'UART, ovvero quando viene passata qualche stringa manualmente dal terminale; ma la parte fondamentale è alla riga 14 e 15. Come già accennato prima parlando dei terminatori, l'invio della stringa avviene soltanto quando:

- l'ultimo carattere del `data_array`, cioè l'ultimo carattere della stringa che si vuole inviare, è il terminatore di *new line* '`\n`'
- oppure quando la stringa è uguale o più lunga di `m_ble_nus_max_data_len` (corrispondente a 20 byte).

In tutti gli altri casi la stringa arriva sull'UART ma non viene inviata. Ecco dunque che, utilizzando (provvisoriamente) questo codice per il kit di sviluppo, bisogna assicurarsi di mandare il carattere terminatore, cioè abilitare il terminale per l'invio, oppure mandare una stringa molto più lunga contenente il nome del training (ma sarebbe poco ragionevole).
Infine, che il carattere terminatore arrivi sulla scheda del guanto o meno poco importa, poiché la funzione `strstr` effettua una ricerca della stringa racchiusa tra virgolette (nella quale non compare il carattere terminatore) all'interno del buffer ricevuto.

5.1.1 Prove di trasmissione e definizione dei training

In questa fase ci si avvicina al funzionamento del firmware finale, ovvero viene inizializzato soltanto il Bluetooth e nient'altro. L'inizializzazione dei sensori e la loro acquisizione viene effettuata soltanto a comando, cioè inviando il nome del training. Sono state infatti definite alcune *define*, in `bluetooth.h`, in modo da riconoscere il training più facilmente nel codice stesso:

```
1. #define PINCHING        1
2. #define GRASPING        2
3. #define WRIST           3
4. #define REACHING        4
5. #define FOLLOWING       5
6. #define PHYSIOLOGICAL   6
```

Sono state successivamente ridefinite le routine `nus_data_handler` e `timer_handler` (all'interno di `scan_digital_sensors.c`). Si riporta di seguito soltanto una parte della funzione `nus_data_handler` in modo da poter commentare le migliorie apportate.

```
1.  void nus_data_handler(ble_nus_evt_t * p_evt)
2.  {
3.      if (p_evt->type == BLE_NUS_EVT_RX_DATA)
4.      {
5.          […]
6.          else if(strstr(p_evt->params.rx_data.p_data, "wrist") != NULL)
7.          {
8.              training = WRIST;
9.              twi_init();
10.             LSM6DS0_configuration();  //imu init
```

```
            LIS3MDL_configuration();  //magnetometer init
            TimerScanDigitalSensors_Init();
        }
        […]
```

Quando si riceve la parola identificativa del training che si vuole eseguire, alla variabile `training` (definita come `uint8_t` in `bluetooth.c`) viene assegnato il corrispondente valore numerico. Tale variabile viene poi richiamata in `scan_digital_sensor.c` come `extern` poiché servirà a stabilire quali sensori devono essere sottoposti a scanning:

```
static void timer_handler(nrf_timer_event_t event_type, void * p_context)
{
    if(event_type == NRF_TIMER_EVENT_COMPARE0)
    {
        switch(training)
        {
            case PINCHING:
            break;

            case GRASPING:
            break;

            case WRIST:
                accelerazione = LSM6DS0_read_accelerometer_data();
                giroscopio = LSM6DS0_read_gyroscope_data();
                //aggiungere codice lettura magnetometro
                sprintf(buf, "%d %d %d %d %d %d\r",
                    accelerazione[0], accelerazione[1], accelerazione[2],
                    giroscopio[0], giroscopio[1], giroscopio[2]);
            break;

                […]
        }
        Send_Bluetooth(buf);
    }
}
```

Si è riportata soltanto una parte dell'handler del timer per comprenderne il funzionamento. Quindi, ad esempio, se viene inviata la parola *"wrist"*, in `nus_data_handler` viene innanzitutto compreso quale training eseguire, vengono abilitate le periferiche e i sensori opportuni, successivamente il `timer_handler` in `scan_digital_sensor.c` effettua lo scanning dei sensori in base al training richiesto. Il `timer_handler` è stato dunque completamente cambiato rispetto al codice precedentemente riportato. Si noti inoltre che ha poco senso includere in `buf` l'indirizzo identificativo del sensore in quanto, nel momento in cui viene richiesto un particolare tipo di training, l'interfaccia dovrà essere già a conoscenza della struttura del buffer che riceve e separare i vari dati andando semplicemente a leggere gli spazi tra i valori numerici.
Di seguito si riportano varie figure che illustrano l'invio dei dati dalla scheda prototipo alimentata in maniera indipendente con un power bank (cavo bianco) e la ricezione su terminale, mediante la scheda di sviluppo collegata via USB (Figura 5.2).
In Figura 5.3 si vede come, una volta connessa la scheda di sviluppo al PC e una volta accesa la scheda prototipo, la connessione Bluetooth tra le due va a buon fine e avviene in maniera immediata. Infatti il firmware della scheda di sviluppo esegue un continuo

scanning finché non trova un altro dispositivo che usa come servizio il Nordic UART Service.

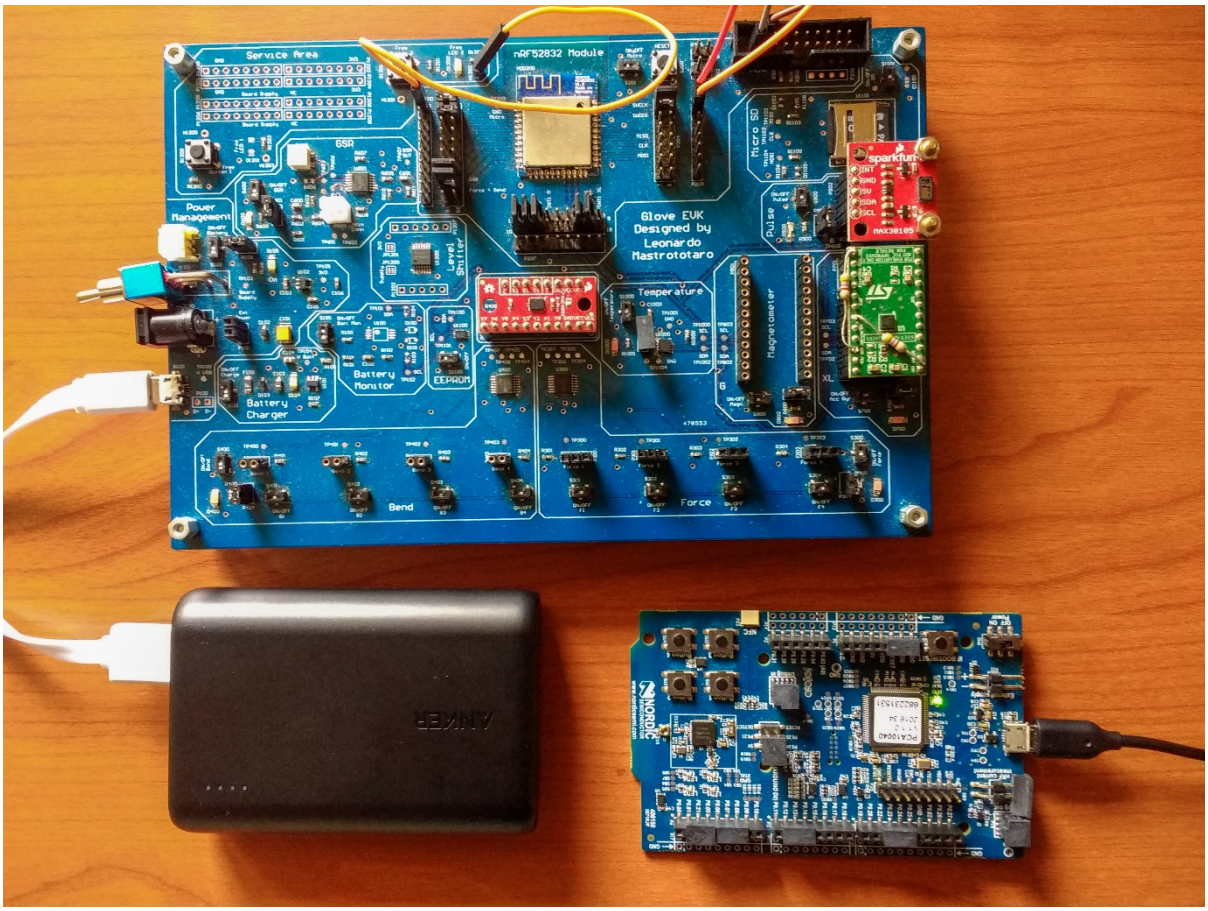

Figura 5.2 Funzionamento indipendente della scheda prototipo

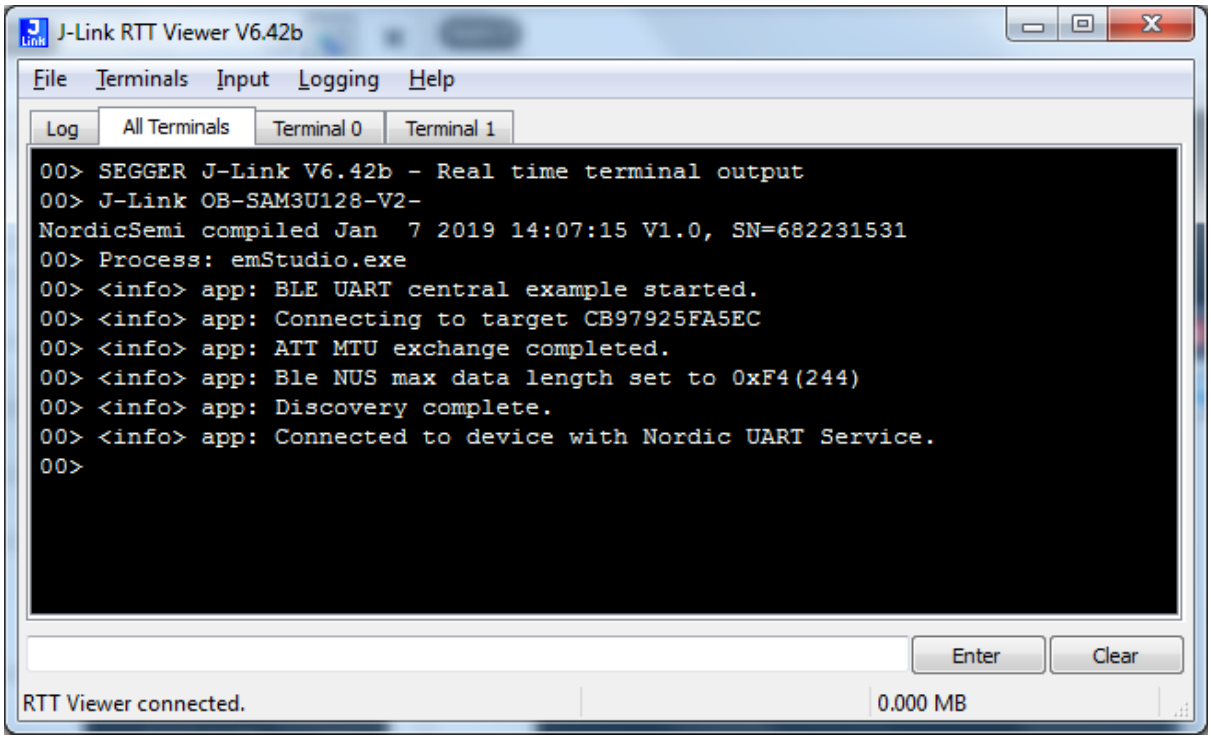

Figura 5.3 J-Link RTT Viewer

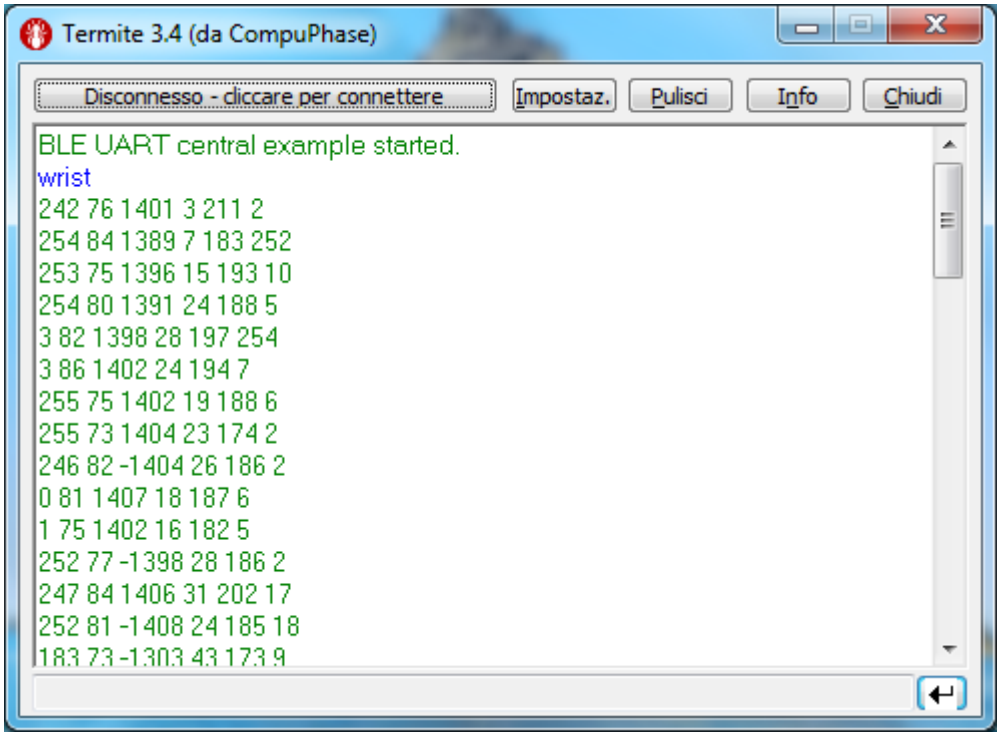

Figura 5.4 Ricezione dati su terminale

La Figura 5.4 invece mostra come a seguito dell'invio della stringa *"wrist"* in blu, si iniziano a ricevere tutti i dati che la scheda prototipo invia effettuando la scansione dei sensori relativi al training.

5.2 Misure con sensori analogici

Per analizzare i dati di entrambi i tipi di sensori (forza e flessione) è stato leggermente cambiato il codice in modo da acquisire soltanto da un ingresso del multiplexer e focalizzare l'attenzione sui valori di tensione provenienti da un solo sensore per volta. In altre parole è stato settato in `mux_init()` l'ingresso desiderato e commentato il codice relativo alla commutazione del MUX nell'ISR del timer.

5.2.1 Calibrazione del convertitore

Nel fare varie prove, si notava come, talvolta, il buffer dell'ADC contenesse valori leggermente negativi o leggermente positivi (cioè attorno allo zero). Considerando ad esempio i sensori di forza e il relativo circuito di condizionamento in Figura 2.5, la tensione che va a finire in ingresso al convertitore è quella ai capi della resistenza R. La resistenza del sensore, invece, decresce al crescere della forza esercitata, motivo per cui quando non vi è alcuna forza applicata la sua resistenza è molto più grande di R e quindi ci si aspetterebbe un valore di tensione nullo (o circa) in ingresso all'ADC (sarebbe idealmente nullo qualora la resistenza del sensore fosse infinitamente grande). Ciò tuttavia non accade per vari motivi, primo fra tutti il fatto che è sempre presente rumore che tende a far oscillare i valori di tensione letti. Un altro valido motivo è che la caratteristica tensione-valore numerico corrispondente del convertitore analogico-digitale non è sempre la stessa qualunque essa sia la condizione di misura, ma subisce

variazioni di offset che dipendono, ad esempio, da variazioni della temperatura. Una possibile situazione è quella rappresentata in Figura 5.5.

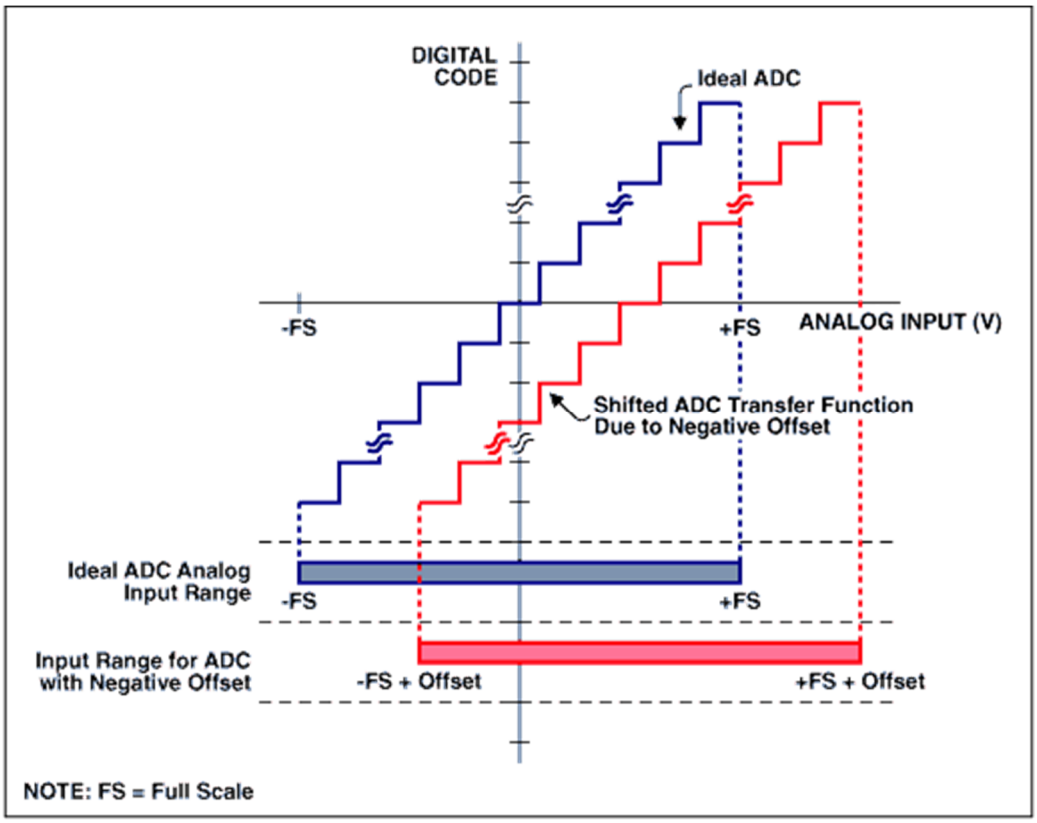

Figura 5.5 Possibile offset negativo nella caratteristica dell'ADC [**49**]

Per questo motivo, è stato scritto anche un codice per la calibrazione iniziale della caratteristica dell'ADC, che viene eseguito nella fase iniziale, prima di acquisire il sensore. Si riporta di seguito il codice relativo alla calibrazione:

```
1.  void calibrate_saadc(void)
2.  {
3.      nrfx_err_t nrfx_err_code = NRFX_SUCCESS;
4.
5.      // Stop ADC
6.      nrf_saadc_int_disable(NRF_SAADC_INT_ALL);
7.      NRFX_IRQ_DISABLE(SAADC_IRQn);
8.      nrf_saadc_task_trigger(NRF_SAADC_TASK_STOP);
9.
10.     // Wait for ADC being stopped.
11.     bool result;
12.     NRFX_WAIT_FOR(nrf_saadc_event_check(NRF_SAADC_EVENT_STOPPED),
13.                                     CALIBRATION_TIMEOUT, 0, result);
14.     NRFX_ASSERT(result);
15.
16.     // Start calibration
17.     NRFX_IRQ_ENABLE(SAADC_IRQn);
18.     nrfx_err_code = nrfx_saadc_calibrate_offset();
19.     APP_ERROR_CHECK(nrfx_err_code);
20.     while(nrfx_saadc_is_busy()){};
21.
22.     // Stop ADC
23.     nrf_saadc_int_disable(NRF_SAADC_INT_ALL);
```

```
24.     NRFX_IRQ_DISABLE(SAADC_IRQn);
25.     nrf_saadc_task_trigger(NRF_SAADC_TASK_STOP);
26.
27.     // Wait for ADC being stopped.
28.     NRFX_WAIT_FOR(nrf_saadc_event_check(NRF_SAADC_EVENT_STOPPED),
29.                                     CALIBRATION_TIMEOUT, 0, result);
30.     NRFX_ASSERT(result);
31.
32.     // Enable IRQ
33.     NRFX_IRQ_ENABLE(SAADC_IRQn);
34. }
```

Come si vede, esso funziona come una sorta di macchina a stati, ovvero vengono definite varie fasi per portare a termine la calibrazione iniziale. Innanzitutto, viene interrotta qualsiasi operazione che sta correntemente effettuando il convertitore tramite la disabilitazione dei relativi interrupt; per portare a termine ed essere certi che l'ADC sia stato momentaneamente interrotto, bisogna assicurarsene passando per uno stato di *"wait"*. Solo successivamente è possibile effettuare la calibrazione con la funzione `nrfx_saadc_calibrate_offset()`. Si riaspetta che l'operazione venga portata a termine, si disabilita nuovamente l'ADC, si aspetta che sia stato effettivamente bloccato e infine si riabilitano gli interrupt iniziali per prepararlo al funzionamento.
La calibrazione è in genere un'operazione da eseguire periodicamente durante il funzionamento del convertitore in quanto possono presentarsi variazioni di temperatura che possono compromettere il valore digitale ottenuto dalla conversione. Tuttavia, per il tipo di applicazione qui in esame difficilmente si hanno grosse variazioni di temperatura e solitamente i training non durano troppo a lungo, motivo per cui la calibrazione viene eseguita soltanto una volta in fase iniziale, cioè nell'inizializzazione del convertitore, e non viene richiamata durante l'acquisizione dei sensori.
Stabilire il momento esatto in cui eseguire la calibrazione, cioè il momento in cui lanciare la funzione `calibrate_saadc()` è di fondamentale importanza per evitare errori nell'inizializzazione dell'ADC e quindi evitare il blocco del microcontrollore. Dopo varie prove, è stato verificato un corretto funzionamento inserendo la fase di calibrazione subito dopo l'impostazione del canale di acquisizione, quindi all'interno della funzione `saadc_init()`, dalla riga 18 alla 21.

```
1.  static void saadc_init(void)
2.  {
3.   ret_code_t err_code;
4.
5.   nrf_drv_saadc_config_t saadc_config = NRF_DRV_SAADC_DEFAULT_CONFIG;
6.   saadc_config.resolution = NRF_SAADC_RESOLUTION_12BIT;
7.
8.   nrf_saadc_channel_config_t channel_config =
9.         NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN1);
10. channel_config.gain = NRF_SAADC_GAIN1;
11. channel_config.reference = NRF_SAADC_REFERENCE_VDD4;
12. err_code=nrf_drv_saadc_init(&saadc_config, saadc_callback);
13. APP_ERROR_CHECK(err_code);
14.
15. if(calibration_enabled)
16. {
17.   calibrate_saadc();
18. }
```

È stata definita una variabile `calibration_enabled` in modo da poter scegliere se eseguire o meno la calibrazione iniziale.

5.2.2 Misure con sensore di forza

Per testare i sensori di forza è stato adoperato uno presente sul guanto prototipo, collegandolo in uno degli ingressi per i sensori di forza presenti sulla scheda prototipo e settando il multiplexer di conseguenza. È stato scelto, casualmente, l'ingresso FORCE_3.
Successivamente è stato adagiato il sensore in maniera piatta in modo da poter applicare delle masse note. Per ogni massa applicata sono stati acquisiti 500 campioni, memorizzati al PC effettuando un log del terminale e successivamente analizzati in MATLAB. In Tabella 5.1 si riportano i valori delle masse note, il valor medio (intero) ottenuto dai 500 campioni e il relativo valore di tensione.

Tabella 5.1 Misure con masse note

Massa [g]	Valor medio ADC	Tensione corrispondente [V]
200	43	0,0346
400	467	0,3762
600	679	0,5470
800	1000	0,8057
1002	1359	1,0949
1199	1702	1,3712
1400	1988	1,6017
1600	2439	1,9650
1804	2988	2,4073
2014	3662	2,9503

Per comprendere meglio l'andamento che lega la massa applicata e la tensione che produce è stato eseguito un fitting dei vari punti (Figura 5.6), utilizzando *Curve Fitting toolbox.* Il metodo applicato per descrivere il modello matematico e fare previsioni a partire da dati sperimentali è il metodo dei minimi quadrati, ovvero una tecnica di ottimizzazione che consente di trovare una funzione che descrive al meglio l'andamento di un insieme di dati (tipicamente punti nel piano) [50]. Se si può ipotizzare una dipendenza di tipo lineare tra due variabili (in tal caso tensione e massa) allora quello che si trova è la cosiddetta "retta di regressione" che descrive l'andamento, ovvero lega le due variabili linearmente a meno di un errore (residuo). L'andamento dei punti è pressoché lineare e la retta di regressione che è stata ottenuta può essere descritta dalla seguente equazione:

$$V(m) = p_1 \cdot m + p_2 \quad [V] \tag{5.1}$$

dove *V(m)* rappresenta la tensione letta dal convertitore analogico-digitale in funzione della massa m applicata, mentre p_1 e p_2 sono i 2 coefficienti ottenuti dal fitting con un intervallo di confidenza del 95%, i cui valori si riportano in Tabella 5.2.

Tabella 5.2 Coefficienti ottenuti dalla regressione

Coeff.	Valore	u.d.m
p_1	0,001516	Volt/grammo
p_2	-0,3548	Volt

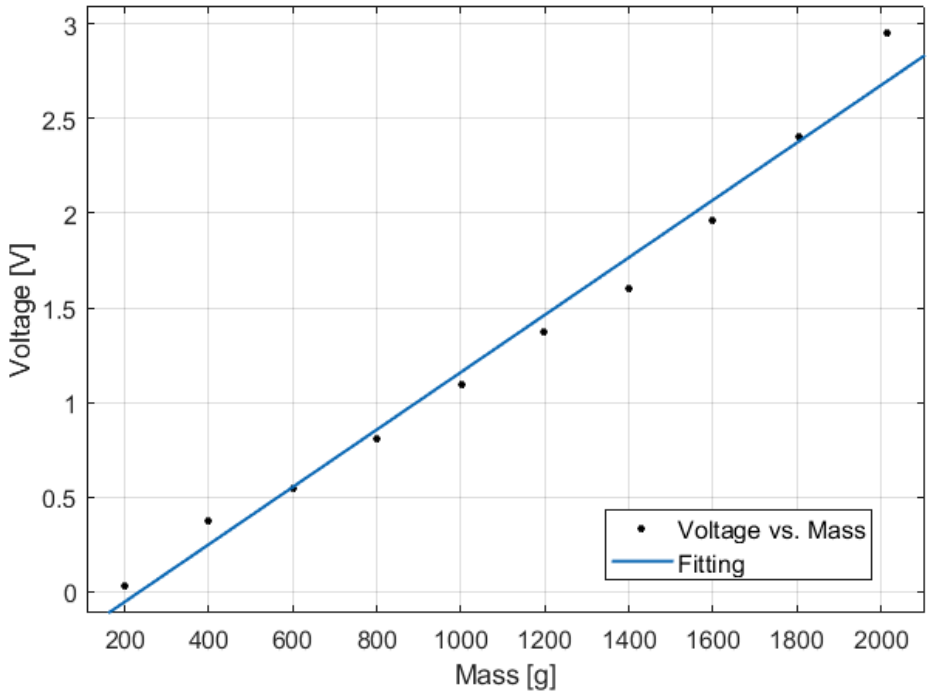

Figura 5.6 Fitting massa-tensione

È chiaro che, ogni qualvolta si ottiene un modello matematico a partire da dati sperimentali, è bene quantificarne la bontà, ovvero quanto esso riesce a descrivere in maniera ottimale l'andamento dei dati ottenuti. A tal proposito esistono vari indici, come ad esempio il grafico dei residui o altri parametri di tipo numerico. Sebbene tali aspetti esulano dallo scopo di questa tesi, si riporta comunque qualche valore ottenuto dal fitting (Tabella 5.3):

Tabella 5.3 Indicatori di bontà del fitting

SSE	0,1419
R-Square	0,9818
RMSE	0,1332

SSE *(Sum of Squares for Error)* è la somma dei quadrati dei residui. Dato che il grafico dei residui deve essere casuale, cioè distribuito in maniera random (e a media nulla), più il valore di SSE è prossimo a zero e più il contributo dell'errore casuale sul modello ottenuto è piccolo.
R-Square, chiamato anche "coefficiente di determinazione", è una proporzione tra la variabilità dei dati e la correttezza del modello. Indici di variabilità dei dati sono ad esempio la varianza o lo scarto quadratico medio. Pertanto, esso misura la frazione della varianza della variabile dipendente (tensione) espressa dalla regressione. I valori che assume vanno da 0 a 1; valori prossimi a 0 indicano che il modello ottenuto non

spiega quasi per niente i dati, viceversa con valori prossimi a 1 la maggior parte dei dati sono spiegati correttamente. Facendo riferimento al valore riportato in Tabella 5.3, il modello spiega bene il 98,18% della variabile dipendente, cioè della tensione, quindi dei risultati ottenuti.
Infine, l'RMSE (*Root-Mean-Square Error*), essendo la radice quadrata dell'errore quadratico medio, non è una grandezza adimensionale ma ha come unità di misura quella della variabile dipendente, ovvero Volt e in particolare è un parametro che indica la discrepanza media fra i valori dei dati osservati e i valori dei dati stimati.

5.2.3 Misure con sensore di flessione

Per effettuare le prove di flessione è stato innanzitutto misurata la resistenza nominale senza nessuna flessione applicata ed è risultata di circa 30 kΩ (come riportato in 2.1.4 si aggira infatti tra i 25 e i 30 kΩ).
Osservando il circuito di Figura 2.7, si intuisce che, come per i sensori di forza, la dinamica della tensione di ingresso al convertitore analogico-digitale dipende dal valore di resistenza R scelto. Chiamando R_{SENSE} il valore di resistenza del sensore, la tensione in ingresso all'ADC è espressa dalla (2.2).
Si è anche interessati, come già espresso a misurare un angolo che va da 0° (dita completamente dritte) a 90° sull'articolazione PIP. È chiaro pertanto che si vuole massimizzare la dinamica su tale intervallo di interesse. Scegliere una resistenza R troppo grande, cioè troppo grande rispetto a R_{SENSE} (vari ordini di grandezza) pregiudica la misura poiché gran parte della tensione cadrà su R e non si riveleranno apprezzabili variazioni di tensione su R_{SENSE} . D'altra parte, adoperare una resistenza R troppo piccola farà sì che la maggior parte della tensione cadrà sempre su R_{SENSE} (per via della legge del partitore) con una conseguente riduzione della dinamica, oltre al fatto che un aumento della resistenza a fronte di una flessione potrebbe portare il convertitore quasi a saturare.
È ragionevole quindi avere una stima della resistenza massima ottenibile a fronte dell'angolo di flessione massimo previsto. Effettuando varie misure si è notato che il valore $R_{SENSE,MAX}$ si aggira attorno ai 120 kΩ, mentre $R_{SENSE,MIN} = 30$ kΩ. Riscrivendo la (2.2) come segue:

$$V_{ADC} = 3{,}3\frac{1}{1 + R/R_{SENSE}} \quad [V] \tag{5.2}$$

e riformulando quanto appena detto, deve essere:

$$V_{ADC,MAX} = 3{,}3\frac{1}{1 + R/R_{SENSE,MAX}} \;\; V \to 3{,}3\,V \Rightarrow R/R_{SENSE,MAX} \to 0$$

ovvero $R \ll R_{SENSE,MAX}$. Eseguendo gli stessi ragionamenti si trova che $R \gg R_{SENSE,MIN}$ e quindi tenendo conto di entrambe le relazioni è chiaro che conviene scegliere R esattamente a metà tra i due, cioè (30+120)/2 kΩ = 75 kΩ. Così facendo si massimizza la dinamica della tensione in ingresso all'ADC, ma chiaramente non si avrà mai 0 come valore convertito a fronte di flessione nulla e il valore massimo (in base alla risoluzione) a fronte della massima flessione. Motivo per cui, prima di effettuare le misure è ragionevole eseguire una calibrazione facendo una misura in assenza di flessione e una con

la massima flessione. Bisogna ricordare, oltretutto, che le misure che si ottengono dipendono da persona a persona perché ognuno ha la propria grandezza e forma della mano e delle dita. Dunque, tenendo conto di questi aspetti, le due misure iniziali per la calibrazione vengono eseguite una con mano totalmente aperta e l'altra con il pugno chiuso che garantisce approssimativamente un angolo di 90° dell'articolazione interfalangea prossimale (PIP).
La tensione misurata con le dita completamente distese è di 0,2465 V, mentre quella con le dita piegate è di 0,4850 V. Di conseguenza, tenendo conto che il comportamento del sensore di flessione è lineare (dichiarato da datasheet), per trovare l'andamento che descrive gli angoli di flessione basterà trovare la retta passante per due punti:

$$V(x) = 2{,}6497 \cdot 10^{-3} \cdot x + 0{,}2465 \quad [V] \tag{5.3}$$

dove x è l'angolo di flessione espresso in gradi sessagesimali e $V(x)$ è il corrispondente valore di tensione misurato a fronte della flessione. L'andamento della retta è rappresentato in Figura 5.7.

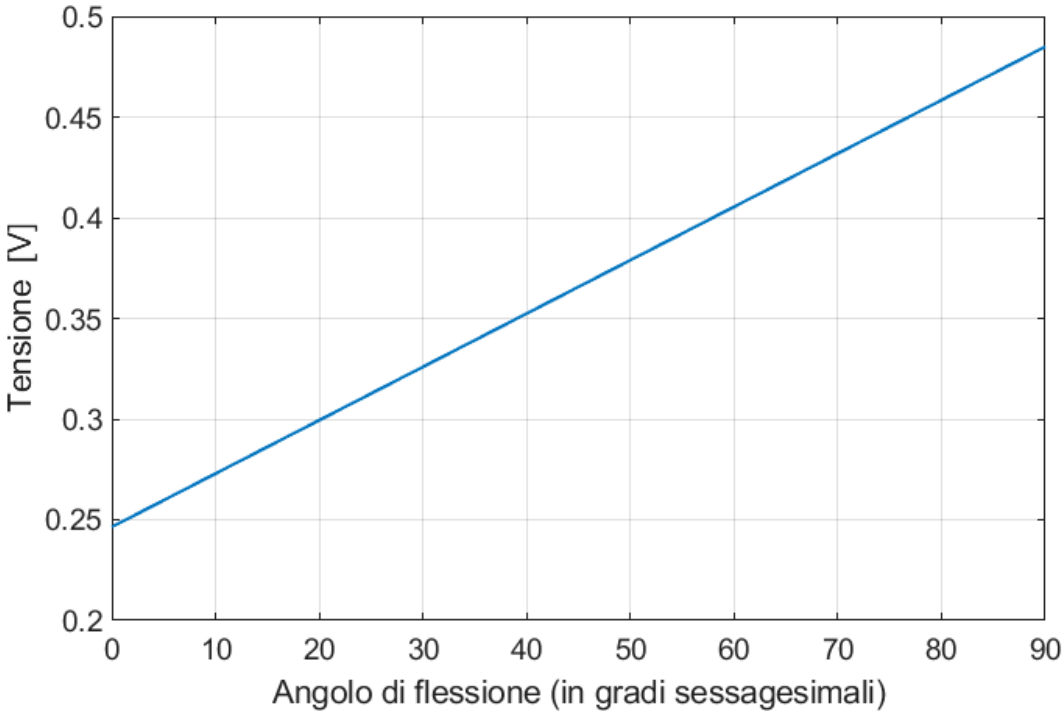

Figura 5.7 Tensione in funzione dell'angolo di flessione

Come verifica del modello trovato, è stata fatta una misura a 45° al quale sono corrisposti 0,3625 V, mentre utilizzando la (5.3) il valore fornito dal modello è di 0,3657 V e, pertanto, può ritenersi valido per il tipo di misure richieste dai training.

5.3 Prototipo dimostrativo

L'ultima fase del lavoro svolto ha previsto la realizzazione del guanto dimostrativo, focalizzandosi soprattutto sullo studio dei movimenti delle dita e, quindi, adoperando i sensori di forza e di flessione. Sono stati pertanto attaccati al guanto i sensori di forza sui polpastrelli e i sensori di flessione sul dorso delle dita (Figura 5.8). Da notare che,

anche se i sensori di forza hanno i loro prolungamenti per il collegamento sul dorso delle dita, la parte attiva, ovvero quella sensibile all'applicazione della forza, è posizionata sui polpastrelli.

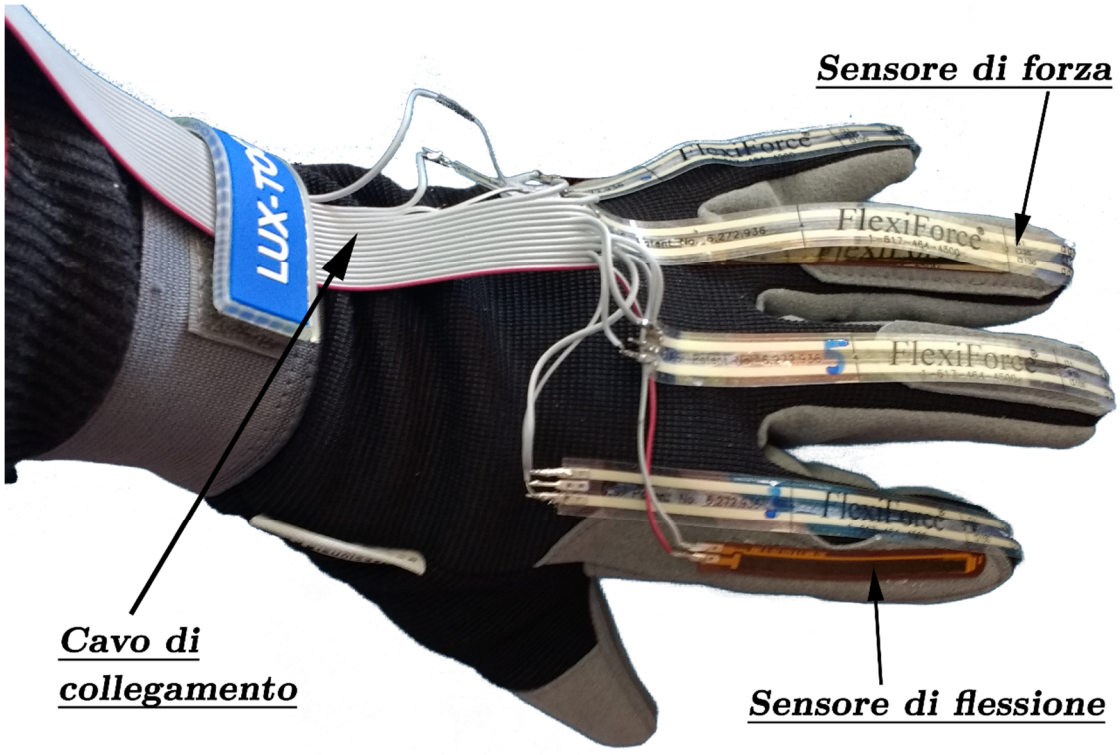

Figura 5.8 Prototipo dimostrativo

Una volta attaccati opportunamente i sensori e assicurandosi di aver scelto la posizione corretta per ciascuno in modo da poter misurare le grandezze in gioco, si è passati alla saldatura dei 16 contatti (2 per ciascun sensore). L'impiego di un cavo piatto flessibile a 16 fili si è rilevato molto utile allo scopo in quanto ha consentito di effettuare i vari collegamenti in maniera pulita e ordinata evitando possibili intrecci qualora si fossero adoperati più fili tutti separati. Da un'estremità sono stati realizzate le connessioni con i sensori, dall'altra i vari pin da inserire nei contatti previsti sull'Evaluation Board.

5.3.1 Visualizzazione grafica in tempo reale

Per poter visualizzare in tempo reale i dati acquisiti è stato realizzato uno script Matlab, di cui si riporta il codice in Appendice A.1.
Lo script usa *Instrument Control Toolbox* per eseguire una scansione delle varie porte COM disponibili. Successivamente, trovata la porta COM (alla quale è connessa il kit di sviluppo della Nordic Semiconductor, qui la COM4) apre una comunicazione di tipo seriale impostando la corretta velocità (Baud Rate definita nel firmware a 115200) e il carattere terminatore per il buffer di trasmissione e di ricezione.
Si precisa che la comunicazione Bluetooth, per come è stata programmata, avviene autonomamente e istantaneamente; di conseguenza l'ordine da seguire è: alimentare a parte l'Evaluation Board (con guanto connesso), connettere e accendere il kit di sviluppo al PC, a questo punto il kit rileva l'Evaluation Board e si connette da solo, lanciare lo script Matlab.

Lo script Matlab esegue le operazioni appena riportate e, instaurata la comunicazione seriale, invia la stringa "`pinching`" per abilitare il convertitore analogico-digitale (sull'Evaluation Board) ad acquisire.
I dati arrivano sul kit di sviluppo che li immette sull'UART; dunque per poter essere letti al PC si esegue una continua scansione della porta COM mediante la funzione `fscanf`. Questa memorizza la stringa ricevuta in una variabile che successivamente viene convertita in un vettore riga di 8 elementi: i dati degli 8 sensori.
Infine, i dati vengono mappati su due grafici in tempo reale, distinguendo un grafico per i sensori di forza (a sinistra in Figura 5.9) e uno per i sensori di flessione (a destra in Figura 5.9).

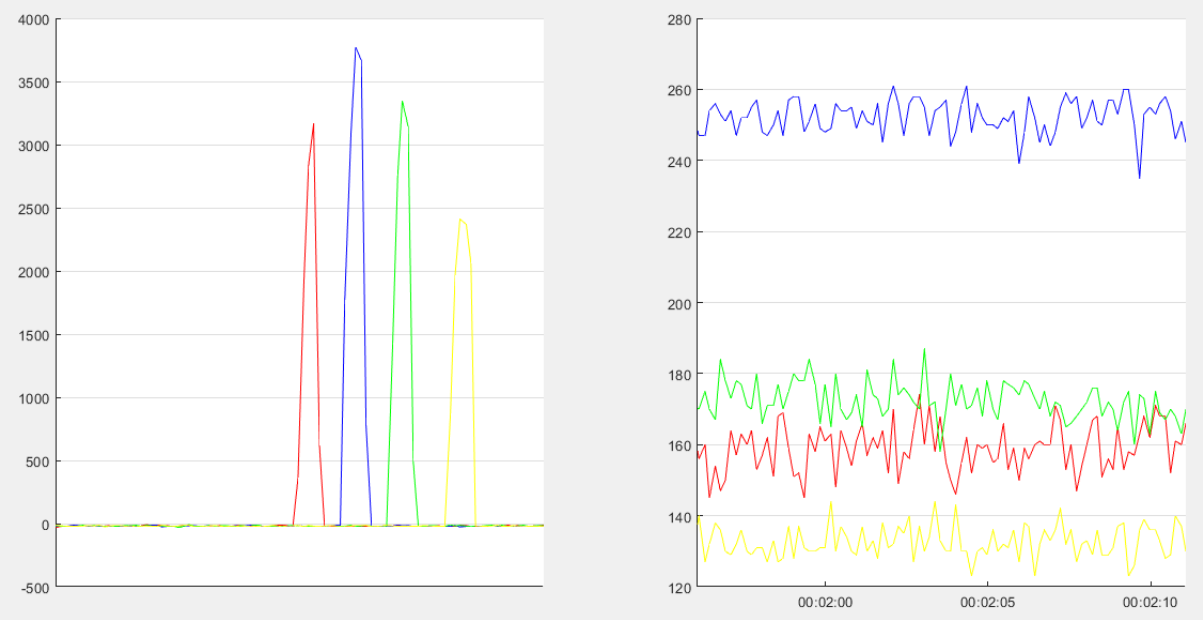

Figura 5.9 Sensori di forza e flessione in tempo reale

I valori dell'asse delle ordinate sono i valori direttamente restituiti dal convertitore analogico-digitale. Non è stata effettuata una conversione nella corrispondente grandezza fisica in quanto qui ci si limita ad effettuare altri tipi di osservazioni.
Ogni dito viene identificato con uno specifico colore, in particolare rosso per l'indice, blu il medio, verde l'anulare, giallo il mignolo.
Nel grafico di sinistra in Figura 5.9 si nota come è stato eseguito il training di *pinching*, che consiste nel toccare la punta del pollice con quella delle altre dita seguendo una particolare sequenza predefinita. Qui, a scopo dimostrativo, è stata eseguita la sequenza, indice, medio, anulare, mignolo.
Nel grafico di destra in Figura 5.9, corrispondente agli stessi istanti di tempo di quello di sinistra e ottenuto facendo in modo di non flettere le dita durante il pinching, corrisponde all'andamento dei sensori di flessione a riposo, cioè dita non flesse.
Oltre al rumore presente, si nota come i vari andamenti assumono valori ben diversi tra loro anche in assenza di flessione; in altre parole, sono posizionati ad altezze differenti. Un andamento del genere è tuttavia spiegato dal fatto che sia la resistenza adoperata per il condizionamento di Figura 2.7, sia il sensore di flessione, che a riposo assume una resistenza finita (a differenza di quello di forza che nelle medesime condizioni ha una resistenza infinitamente grande), sono caratterizzati da tolleranze di fabbricazione e di conseguenza, essendo il condizionamento un partitore resistivo, la tensione (a riposo) acquisita dal convertitore sarà diversa per ciascun sensore.

Un simile comportamento avvalora quanto già detto, ovvero è necessario fare una calibrazione iniziale per ciascun sensore di flessione, in modo da rimuovere gli offset iniziali nella conversione in grandezza fisica.
Di seguito, in Figura 5.10, si riporta l'andamento dei sensori di flessione piegando in ordine l'indice, il medio, l'anulare e il mignolo.

Figura 5.10 Andamento a fronte di flessione delle dita

5.3.2 Problemi riscontrati e correzioni

Sebbene un'attenta ispezione dei valori numerici contenuti all'interno del buffer del convertitore analogico-digitale avrebbe già messo in luce il problema fin dall'inizio, la visualizzazione grafica delle acquisizioni dei sensori si è rivelata fondamentale per la correzione di un bug: l'ordine dei sensori all'interno del buffer dell'ADC.
Andando a mappare su grafico i dati, ci si è accorti che l'ordine prestabilito non veniva sempre rispettato. Questo è essenzialmente dovuto a una ragione principale, ovvero il fatto che in fase di inizializzazione dell'ADC, per poter acquisire sempre con lo stesso ordine, occorre ben sincronizzare i tempi dell'interrupt del timer che trigghera la conversione dell'ADC e effettua la commutazione del multiplexer.
Se infatti il timer viene abilitato e il suo interrupt scatta quando ancora tutte le istruzioni per la corretta inizializzazione della conversione non sono state portate a termine, basta soltanto una volta che venga eseguita l'Interrupt Service Routine del timer affinché la variabile `buffer_index` cambi il suo valore e, di conseguenza, anche l'ordine dei campioni all'interno del buffer. Si notava che l'ordine della sequenza stabilita con il costrutto `switch-case` veniva rispettata; ciò che non veniva rispettato era il primo sensore da acquisire nella sequenza e di conseguenza tutti gli altri a seguire, provocando uno "slittamento in avanti" di tutto il buffer.
Non solo, talvolta in fase di debug il problema non si presentava in quanto l'intervento dell'utente sullo stabilire le tempistiche delle istruzioni da eseguire faceva sì che non si verificassero problemi di questo tipo.

Il codice di inizializzazione del convertitore è stato pertanto cambiato: la funzione `saadc_sampling_event_enable()` non viene più eseguita all'interno di `adc_init()`, ma all'interno di `timer_handler()`. Per l'esattezza è stata definita una variabile `adc_started` che viene settata a `true` dopo che viene effettuata l'inizializzazione del multiplexer con `mux_init()`. La variabile `adc_started` funge da condizione di un `if` all'interno dell'ISR del timer, che viene eseguito soltanto una volta; si riporta il codice di seguito.

```
1. if(event_type == NRF_TIMER_EVENT_COMPARE0)
2.     {
3.         if(adc_started)
4.         {
5.             adc_started = false;
6.             buffer_index = 0;
7.             saadc_sampling_event_enable();
8.         }
9.
10.        switch(buffer_index)
11.        {
12.            case 0:   //FORCE_1 => 000
13.            nrf_gpio_pin_clear(ACONNO_MODULE_S1);
14.            break;
15.
16.            [...]
```

All'interno dell'`if`, qualora fosse già stata eseguita una o più volte l'ISR del timer, ci si assicura di inizializzare nuovamente a 0 `buffer_index` e, successivamente, viene abilitata la conversione dell'ADC con la funzione `saadc_sampling_event_enable()`.
Altra osservazione da fare è che in `mux_init()` i pin GPIO relativi al MUX vengono tutti inizializzati a 0, ovvero $S_2 = 0$, $S_1 = 0$, $S_0 = 0$ e non a $S_2 = 0$, $S_1 = 1$, $S_0 = 0$.
Infine, un problema noto fin dall'inizio, passato poi in secondo piano, ma che si è riusciti comunque a risolvere sia in parte grazie all'ispezione visiva dei dati dei sensori, sia effettuando misure di tensione e svariate prove sul codice del microcontrollore, è il valore di tensione (e quindi il valore logico) che assume erroneamente il pin GPIO relativo all'ingresso di selezione S_1 del multiplexer. Si è notato, infatti, che il suo stato era sempre al valore logico alto, indipendentemente da istruzioni che ne invertissero lo stato. Il motivo di tale problema è stato riscontrato sull'UART, che usa come `TX_PIN_NUMBER` proprio il pin relativo a S_1, cioè P0.6.
Sebbene l'UART venga inizializzata per il Nordic UART Service, ma non se ne fa realmente uso come accuratamente descritto in precedenza, il problema è stato risolto settando a `NULL` i vari pin relativi all'UART.

Conclusioni e sviluppi futuri

Come visto nei cinque capitoli di cui si articola questo lavoro di tesi, la programmazione di un sistema di *sensing* indossabile, come quello di un guanto sensorizzato, prevede varie fasi di sviluppo.
Dopo aver fornito nel primo capitolo le informazioni di base sulla progettazione di un *data glove*, i suoi svariati impieghi, le grandezze fisiche che si intende misurare, i sensori comunemente impiegati, nonché i materiali adoperati nella sua realizzazione, si è passati ad inquadrare, nel capitolo successivo, il caso di studio qui trattato: un guanto sensorizzato utile per eseguire dei cosiddetti *training* di riabilitazione clinici.
Tali training consistono nel compiere dei particolari movimenti con la mano, pertanto, ciascuno di essi necessita di acquisire soltanto determinate grandezze, come ad esempio forza e flessione delle dita, o rotazione del polso.
Sempre nel secondo capitolo è stata introdotta l'elettronica di cui si è fatto uso, ovvero l'*Evaluation Board* e la realizzazione della sua equivalente miniaturizzata.
Successivamente, nel terzo capitolo sono state descritte le fasi dello sviluppo del firmware a partire dalle specifiche di partenza. Si è visto come la configurazione dell'ambiente di sviluppo sia il primo passo da eseguire per poter far fronte a tutto lo sviluppo successivo e, quindi, comprendere bene come è strutturato il *Software Development Kit* è di fondamentale importanza.
Si è passati poi a rilevare e programmare l'acquisizione dei sensori digitali e analogici, con particolare enfasi alla modularità del firmware; in primis, poiché il codice deve essere sempre di facile lettura: così come uno scrittore suddivide la propria opera in capitoli, paragrafi e argomenti, allo stesso modo un programmatore dovrà fare con il proprio codice, cioè suddividere il progetto in più file in modo che sia facilmente interpretabile da egli stesso e da chi un domani proseguirà il lavoro.
Inoltre, la modularità è dettata anche dal fatto che la scheda presenta una particolare configurazione di funzionamento in base al tipo di training che è stato impostato.
Nel quarto capitolo è stata affrontata la tematica della connettività wireless, di fondamentale importanza per la selezione del training da eseguire e il successivo invio dei dati provenienti dai sensori. Talvolta, è stato possibile notare come l'introduzione di nuove funzionalità può far sorgere problematiche e malfunzionamenti sulla parte di codice già testata e che, pertanto, va necessariamente rianalizzata in modo da integrarla con il codice nuovo. Nello stesso capitolo è stato accuratamente studiato anche il modo in cui trasmettere le informazioni e, quindi, come organizzare i dati di più sensori all'interno del buffer da trasmettere.

Infine, nell'ultimo capitolo, dopo aver eseguito varie prove di trasmissione e ricezione, è stata fatta la calibrazione per i sensori analogici ottenendo due modelli che legano il valore letto dal convertitore analogico-digitale alla grandezza fisica.
La realizzazione del guanto prototipo ha messo ulteriormente in evidenza, tramite un tracciamento grafico dei dati in tempo reale, come ciascun sensore necessita di una calibrazione iniziale, sia per rimuovere inevitabili offset, sia perché le misure sono dipendenti dalle caratteristiche fisiche, quali forma e spessore, della mano del paziente.

Tra gli sviluppi futuri, il primo che sicuramente merita di essere elencato è la realizzazione di un opportuno software con interfaccia grafica capace di connettersi direttamente alla scheda per impostare il training desiderato, ricevere i dati e analizzarli con chiarezza. Bisogna infatti ricordare che chi utilizzerà lo strumento ultimato in tutte le sue parti è interessato a sapere soltanto l'esito finale del training e quindi disporre di un ambiente *user-friendly* per la valutazione della motilità del paziente.
Non meno importante, vi è l'integrazione di tutta l'elettronica nel guanto, e quindi la realizzazione di una seconda scheda miniaturizzata contenente il pulsimetro e il termometro che andrà installata, assieme alla principale, nel guanto con tutti i sensori.

In definitiva, le migliorie che si possono apportare e gli sviluppi che si possono intraprendere sono infiniti: una volta comprese le problematiche progettuali e che si è a conoscenza delle potenzialità che lo strumento offre, l'ultimo limite, nonché compito finale dell'ingegnere, è quello di dare forma alla propria fantasia e creatività.

La fantasia è come la marmellata,
bisogna che sia spalmata su una solida fetta di pane.
(Italo Calvino)

Appendice

A.1 Codice MATLAB per acquisizione Real-Time

Si riporta di seguito il codice relativo all'acquisizione e tracciamento dell'andamento dei sensori di forza e flessione utilizzato per i risultati del paragrafo 5.3.1.

```
close all;
clear all;
clc;

figure;
subplot(2,1,1);
fig1 = gcf;
a1 = fig1.CurrentAxes;
a1.YGrid = 'on';
datetick('x','keeplimits');
force1 = animatedline('Color','r');
force2 = animatedline('Color','b');
force3 = animatedline('Color','g');
force4 = animatedline('Color','y');

subplot(2,1,2);
datetick('x','keeplimits');
fig2 = gcf;
a2 = fig2.CurrentAxes;
a2.YGrid = 'on';
bend1 = animatedline('Color','r');
bend2 = animatedline('Color','b');
bend3 = animatedline('Color','g');
bend4 = animatedline('Color','y');

%% SERIAL COMMUNICATION

% Cerca un serial port object
obj1 = instrfind('Type', 'serial', 'Port', 'COM4', 'Tag', '');

% Crea il serial port object se non esiste
% altrimenti usa quello che è stato trovato
if isempty(obj1)
    obj1 = serial('COM4');
else
    fclose(obj1);
    obj1 = obj1(1);
end
```

```
39.
40.% Configura l'instrument object, obj1
41.set(obj1, 'Terminator', {'CR','LF'});
42.set(obj1,'BaudRate',115200);
43.
44.%% SERIAL CONNECTION
45.
46.% Connessione a instrument object, obj1
47.fopen(obj1);
48.
49.%% Serial Configuration and Control
50.
51.% Comunicazione con instrument object, obj1
52.fprintf(obj1, 'pinching');
53.pause(0.5);
54.
55.startTime = datetime('now');
56.while 1
57.
58.    % Acquisizione buffer
59.    data = fscanf(obj1);
60.    adc = str2num(data);
61.
62.    t =  datetime('now') - startTime;
63.
64.    addpoints(force1, datenum(t), adc(1));
65.    addpoints(force2, datenum(t), adc(4));
66.    addpoints(force3, datenum(t), adc(8));
67.    addpoints(force4, datenum(t), adc(5));
68.    a1.XLim = datenum([t-seconds(15) t]);
69.    datetick('x','keeplimits');
70.
71.    addpoints(bend1, datenum(t), adc(2));
72.    addpoints(bend2, datenum(t), adc(3));
73.    addpoints(bend3, datenum(t), adc(7));
74.    addpoints(bend4, datenum(t), adc(6));
75.    a2.XLim = datenum([t-seconds(15) t]);
76.    datetick('x','keeplimits');
77.
78.    drawnow;
79.
80.end
```

Bibliografia

[1] Wikipedia, «Wired Glove,» [Online]. Available: https://en.wikipedia.org/wiki/Wired_glove.

[2] T. G. Zimmerman, J. Lanier, C. Blanchard, S. Bryson e Y. Harvill, «A Hand Gesture Interface Device,» VPL Research, Inc., Redwood City, CA, 1987.

[3] American Society For Surgery of the Hand , «Finger Laceration,» [Online]. Available: http://www.assh.org/handconsult/Cases-A-Z/Condition-Information/ArticleID/50826/Finger-Laceration#prettyPhoto.

[4] Humanware srl, «Humanglove Developers Manual,» Pisa, 1998.

[5] Williams, Penrose, Caddy, Barnes, Hose e Harley, «A goniometric glove for clinical hand assessment. Construction, calibration and validation.,» Sheffield, UK.

[6] N.W.Williams, «The virtual hand: The Pulvertaft Prize Essay for 1996,» Sheffield , 1997.

[7] L. Simone, N. Sundarrajan, X. Luo, Y. Jia e D. G. Kamper, «A low cost instrumented glove for extended monitoring and functional hand assessment,» 2007.

[8] L. Swallow e E. Siores, «Tremor Suppression Using Smart Textile Fibre Systems,» *Journal of Fiber Bioengineering and Informatics,* vol. 1, n. 4, pp. 261-266, 2009.

[9] J. J. LaViola, «A Survey of Hand Posture and Gesture Recognition Techniques and Technology,» 1999.

[10] N. Oess, J. Wanek e A. Curt, «Design and evaluation of a low-cost instrumented glove for hand function assessment,» 2012.

[11] Wikipedia, «Elastam,» [Online]. Available: https://it.wikipedia.org/wiki/Elastam.

[12] M. Stoppa e A. Chiolerio, «Wearable Electronics and Smart Textiles: A Critical Review,» 2014.

[13] B. O'Flynn, J. T. Sanchez, P. Angove, J. Connolly, J. Condell, K. Curran e P. Gardiner, «Novel smart sensor glove for arthritis rehabiliation,» 2013.

[14] G. H. Büscher, R. Kõiva, C. Schürmann, R. Haschke e H. J. Ritter, «Flexible and stretchable fabric-based tactile sensor,» in *Robotics and Autonomous Systems*, vol. 63, Elsevier, 2015, pp. 244-252.

[15] Wikipedia, «Filtro di Kalman,» [Online]. Available: https://it.wikipedia.org/wiki/Filtro_di_Kalman.

[16] H. Kortier, V. Sluiter, D. Roetenberg e P. Veltink, «Assessment of hand kinematics using inertial and magnetic sensors,» *Journal of NeuroEngineering and Rehabilitation,* 2014.

[17] «Development and evaluation of a sensor glove for hand function assessment and preliminary attempts at assessing hand coordination,» in *Measurement*, vol. 93, Elsevier, 2016, pp. 1-12.

[18] Dipartimento di Psicologia - Università di Torino, «SAMBA (SpAtial, Motor & Bodily Awareness),» [Online]. Available: https://www.dippsicologia.unito.it/do/gruppi.pl/Show?_id=hhuv.

[19] Wikipedia, «Propriocezione,» [Online]. Available: https://it.wikipedia.org/wiki/Propriocezione.

[20] L. Mastrototaro, *Design and prototyping of a wearable sensing system for clinical rehabilitation trainings,* Tesi di Laurea, 2017.

[21] BrainSigns - Spin-off dell'Università la Sapienza di Roma, «Risposta galvanica della pelle (GSR),» [Online]. Available: https://www.brainsigns.com/it/science/s2/technologies/gsr.

[22] STMicroelectronics, «LSM6DSO - iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope,» 2019.

[23] Wikipedia, «Sensibilità di un sistema di misura,» [Online]. Available: https://it.wikipedia.org/wiki/Sensibilit%C3%A0_di_un_sistema_di_misura.

[24] Wikipedia, «LIS3MDL - Digital output magnetic sensor: ultra-low-power, high-performance 3-axis magnetometer,» 2017.

[25] Maxim Integrated, «MAX30105 - High-Sensitivity Optical Sensor for Smoke Detection Applications».

[26] M. Integrated, «MAX30205 Datasheet - Human Body Temperature Sensor». 2016.

[27] aconno, *ACN52832 Datasheet v1.2,* 2016.

[28] F. Sbarra, *Progettazione e prototipazione di sistemi di sensing indossabili per training di riabilitazione clinici,* Tesi di Laurea Magistrale, 2018.

[29] TAIYO YUDEN CO., LTD., *Bluetooth Low Energy Module EYSHSNZWZ - Data Report,* 2017.

[30] TED KRITSONIS, «Review: Athos wearable tech shirt and shorts,» [Online]. Available: http://whatsyourtech.ca/2015/12/29/review-athos-wearable-tech-shirt-and-shorts/.

[31] Truly Gadgets, «Peregrine Gaming Glove: Giving Your Gameplay A Hand,» [Online]. Available: http://www.trulygadgets.com/4244/gadgets/peregrine-gaming-glove-giving-gameplay-hand/.

[32] Nordic Semiconductor, «nRF52832 Product Brief Verion 2.0,» [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF52832_PB_v2.0.pdf.

[33] Wikipedia, «Header file,» [Online]. Available: https://it.wikipedia.org/wiki/Header_file.

[34] Wikipedia, «Codice oggetto,» [Online]. Available: https://it.wikipedia.org/wiki/Codice_oggetto.

[35] Wikipedia, «Variabili,operatori e costanti,» [Online]. Available: https://it.wikibooks.org/wiki/C/Variabili,_operatori_e_costanti/Variabili#volatile.

[36] P. Prinz e T. Crawford, «Type Qualifier,» in *C in a Nutshell - A desktop quick reference*, O'Reilly, p. 175.

[37] Nordic Semiconductor, «nRF52832 Product Specification v1.4,» 10 Ottobre 2017. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf.

[38] Nordic Semiconductor, *nRF5 SDK v15.2.0 Documentation.*

[39] Wikipedia, «Bluetooth,» [Online]. Available: https://it.wikipedia.org/wiki/Bluetooth.

[40] Nordic Semiconductor, «Bluetooth 5 Ready and Able,» [Online]. Available: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/Bluetooth-5.

[41] Nordic Semiconductor, «nRF52 Series Product Brief v3.0,» [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF52_Series_PB_v3.0.pdf.

[42] K. Townsend, C. Cufi, A. Davidson e R. Davidson, Getting started with Bluetooth Low Energy - Tools and Techniques for Low-Power Network, O'Reilly.

[43] Dr. Chatschik Bisdikian - IBM Research, «Bluetooth Architecture Overview,» [Online]. Available: https://slideplayer.com/slide/702889/.

[44] Nordic Semiconductor, «Creating Bluetooth® Low Energy Applications - Application Note v1.1,» [Online]. Available: http://infocenter.nordicsemi.com/pdf/nan_36.pdf.

[45] Adafruit, «Introduction to Bluetooth Low Energy,» [Online]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

[46] Sparkfun, «Bluetooth Basics,» [Online]. Available: https://learn.sparkfun.com/tutorials/bluetooth-basics/bluetooth-profiles.

[47] Philips, «BGB203 BT 2.0 Serial Port Profile Module User's Guide,» [Online]. Available: https://www.sparkfun.com/datasheets/Wireless/Bluetooth/BGB203_SPP_UserGuide.pdf.

[48] Nordic Semiconductor, «Webcast: Interfacing with Nordic SoftDevices,» [Online]. Available: https://www.youtube.com/watch?v=tZjlixQPO-Q.

[49] Maxim Integrated, «The ABCs of ADCs: Understanding How ADC Errors Affect System Performance,» [Online]. Available: https://www.maximintegrated.com/en/app-notes/index.mvp/id/748.

[50] Wikipedia, «Metodo dei minimi quadrati,» [Online]. Available: https://it.wikipedia.org/wiki/Metodo_dei_minimi_quadrati.