**Relazione Laboratorio ingegneria dell’informazione**

Introduzione con obbiettivi e motivazioni del progetto

Il progetto Lewe2.0 nasce dalle ceneri di un precedente progetto denominato Lewe (<https://github.com/alessandro1105/Lewe)> con l’obbiettivo di evolverne codice e hardware, mantenendone, invece, inalterati gli scopi e gli obiettivi.

Lewe è stato concepito per raccogliere e rielaborare varie tecnologie di telecomunicazione nel mondo dell’IOT (Internet Of Things) dove vari dispositivi devono comunicare in tempo reale per la realizzazione di un obiettivo comune, che spazia dalla lettura di sensori al telecontrollo di elettrodomestici.

In questo momento nell’ambito dell’IOT non esiste un protocollo di comunicazione comunemente accettato che sia al contempo abbastanza leggero, flessibile, efficiente e sicuro da integrarsi agli svariati contesti dove l’IOT vede la sua applicazione.

Lewe (e Lewe2.0) nasce dunque con l’idea di realizzare un protocollo di comunicazione che rispecchi quanto più possibile le caratteristiche sopra citate partendo da un contesto applicativo semplificato, ma non banale, come quello della rilevazione di alcuni dati biometrici attraverso un bracciale dotato di sensori e l’invio degli stessi ad una applicazione per smartphone Android.

Il contesto non risulta in alcun modo modo banale per il fatto che gran parte delle applicazioni quotidiane dell’IOT prevedono proprio il monitoraggio da remoto di sensori sulle cui letture vengono intraprese delle azioni di telecontrollo di qualsiasi genere e tipologia.

Ricapitolando, il progetto Lewe2.0 prevede la realizzazione di un bracciale (prototipo di fitness tracker) dotato di alcuni sensori le cui letture devono essere inviate, attraverso il protocollo di comunicazione denominato Jack, ad una applicazione Android.

Il protocollo stesso è in realtà un prototipo e lo scopo principale di Lewe2.0 è quello di comprendere i requisiti che lo stesso dovrà avere, partendo da una struttura simile al JSON (JavaScript Object Notation) che risulta molto più semplice da comprendere per un essere umano rispetto ad un protocollo basato su codice binario tipo CoAP (Constrained Application Protocol).

Architettura e progettazione dell’applicazione

Il progetto si suddivide in tre blocchi principali: il bracciale e l’applicazione per smartphone Android e il protocollo di comunicazione Jack.

IL BRACCIALE

***(Foto bracciale)***

Il bracciale è composto da Fishino UNO (n.d.r. un clone di Arduino che integra RTC e WiFi) e da una basetta autocostruita contenente tutto l’hardware per la gestione dei sensori di temperatura e GSR (Galvanic Skin Response).

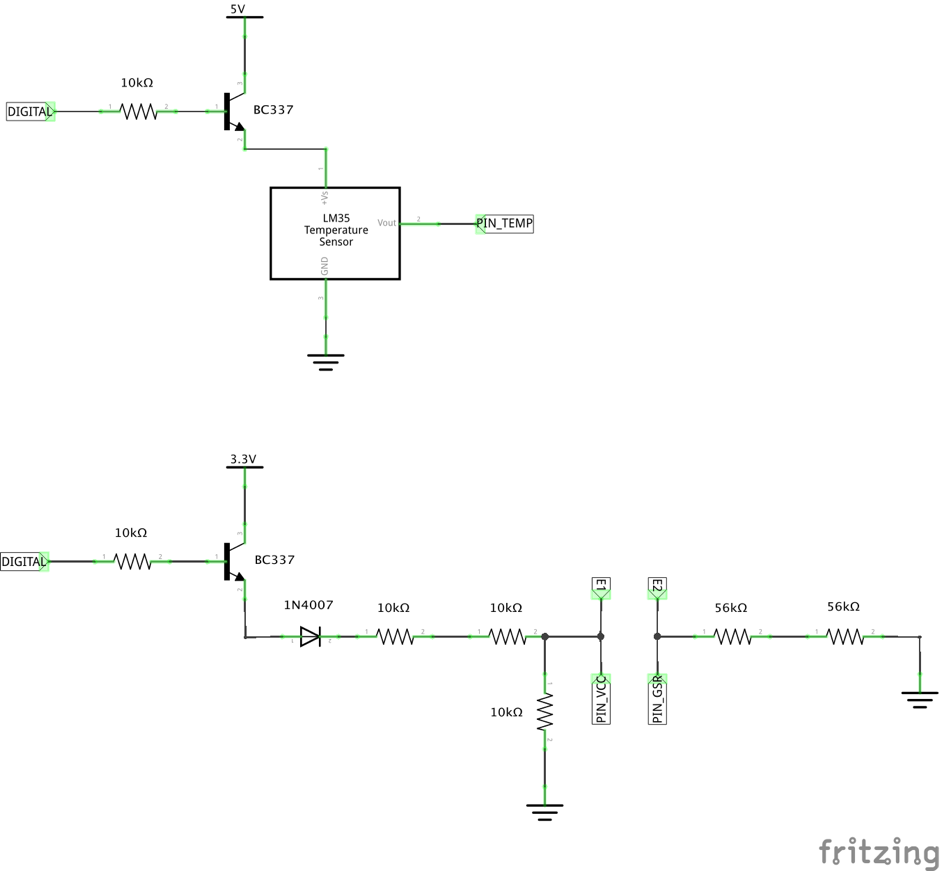


Fig. 1: Schema completo del circuito di gestione dei sensori.

Il sensore di temperatura è un LM35DZ e fornisce come output un valore di tensione proporzionale alla temperatura (10mV/°C).

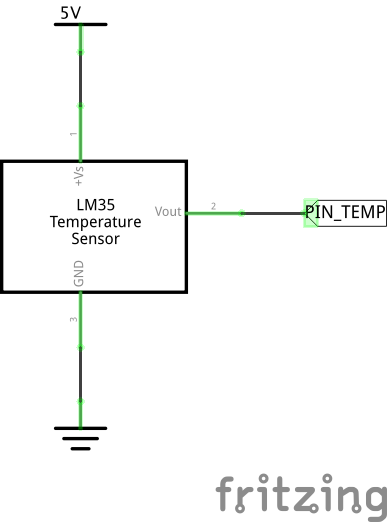


Fig. 2: Schema di connessione del sensore di temperatura LM35DZ.

La temperatura è ricavata attraverso la formula:

Dove è la temperatura di riferimento per l’ADC (Analog/Digital Converter) di Fishino UNO, che per questa applicazione è stato impostato a 1.1V.

Il sensore GSR (Galvanic Skin Response) è formato da due elettrodi a contatto diretto con la pelle, ai quali viene applicata una piccolissima tensione, impercettibile dal soggetto indossatore del bracciale, attraverso la quale è possibile misurare la resistenza della pelle che varia in relazione a stimoli emotivi che agiscono sulle ghiandole sudoripare.

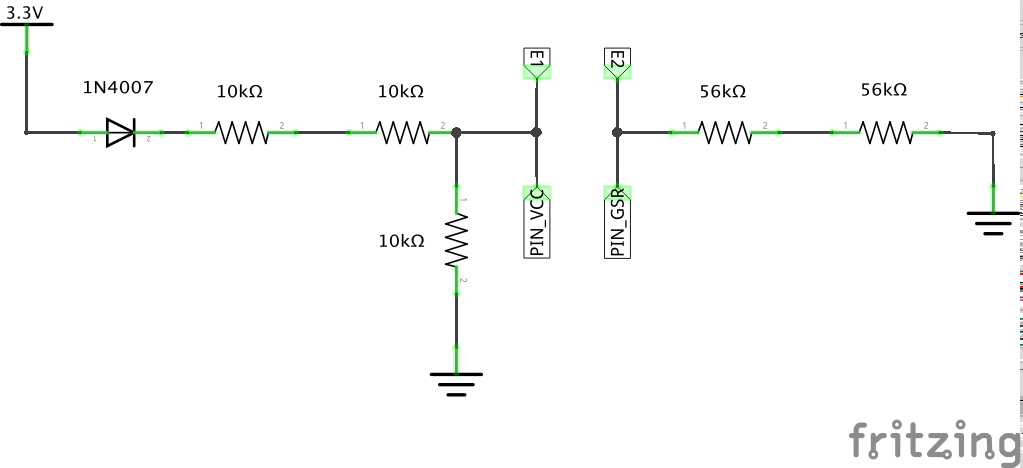
****

Fig. 3: Schema del sensore GSR.

La lettura del sensore in Ohm non è di facile manipolazione e per questo motivo si è preferito convertire questo valore in percentuale per rendere più agevole la sua lettura da parte di chiunque. La formula usata per la conversione è la seguente:

Il rapporto è usato per autocalibrare il sensore, rendendolo immune ad eventuali cali di tensione dovuti all’esaurimento progressivo dell’alimentazione.

Poiché il bracciale è alimentato interamente a batteria è stato anche previsto un meccanismo per il risparmio energetico che consente di attivare i sensori solo per la lettura e di disattivarli al termine della stessa riducendone quindi il consumo. Il circuito si avvale di un transistor BC337 in modalità ON/OFF, ovvero è usato come interruttore pilotato da un segnale digitale proveniente dal MCU (MicroController Unit).

a) b)

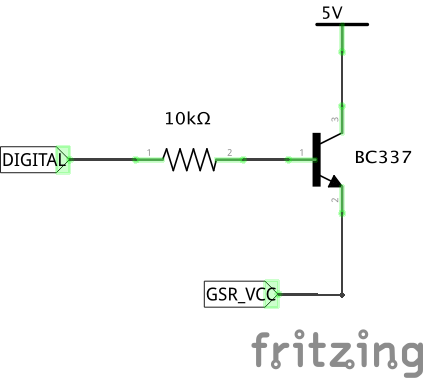
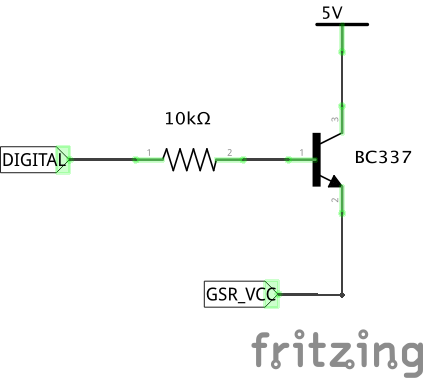
** **

Fig. 4a: Schema di risparmio energetico per il sensore di temperatura LM35DZ.

Fig. 4b: Schema di risparmio energetico per il sensore GSR.

Il modulo usato per la connessione bluetooth con l’applicazione per smartphone Android è un HM-10. E’ un modulo BLE (Bluetooth Low Energy) che simula una connessione seriale attraverso un servizio e una caratteristica condivisa tra RX e TX. Esso è dotato di un’interfaccia UART che consente, oltre che l’invio di dati attraverso la connessione seriale simulata, l’esecuzione di comandi (AT), alcuni di essi sono stati usati per la configurazione iniziale (si rimanda al datasheet del modulo per una completa trattazione dei comandi e delle possibili configurazioni).

*Il firmware del bracciale è stato sviluppato in C/C++, insieme alla realizzazione della libreria Hashmap (basata sui sorgenti di Alexander Brevig per il progetto Wiring) in quanto l’IDE di Arduino ne risultava sprovvisto ed era di essenziale importanza per la libreria Jack che implementa il protocollo di comunicazione omonimo.*

APPLICAZIONE PER SMARTPHONE ANDROID

L’applicazione per smartphone Android permette la visualizzazione delle letture dei sensori posti nel bracciale.

**(foto main activity temperarura/gsr)**

La main activity consiste di due sezioni: la prima visualizza l’ultima lettura ricevuta dei sensori (insieme alla data e ora di rilevazione), mentre la seconda mostra graficamente le ultime 5 letture. Cliccando sul grafico è possibile accedere ad un’altra activity che permette di visionare, sempre attraverso un grafico, tutte le letture ricevute dal bracciale.

**(foto chart temperatura/gsr)**

L’associazione tra applicazione e bracciale avviene attraverso una schermata specifica, dove l’applicazione si connette al primo device bluetooth chiamato “LW2.0” e ne ricorda l’indirizzo MAC per potersi connettere nuovamente alla successiva apertura.

a) b)

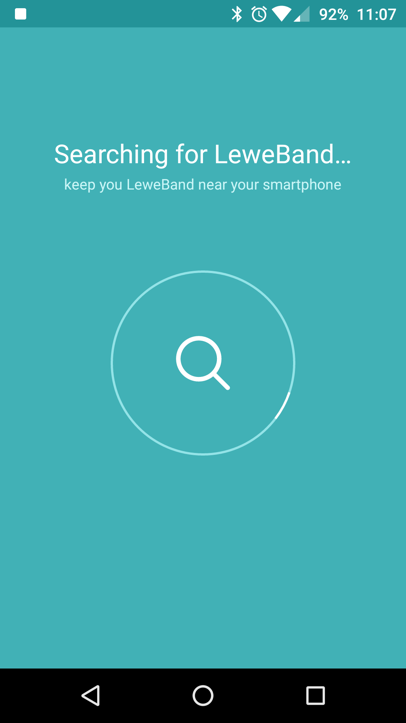
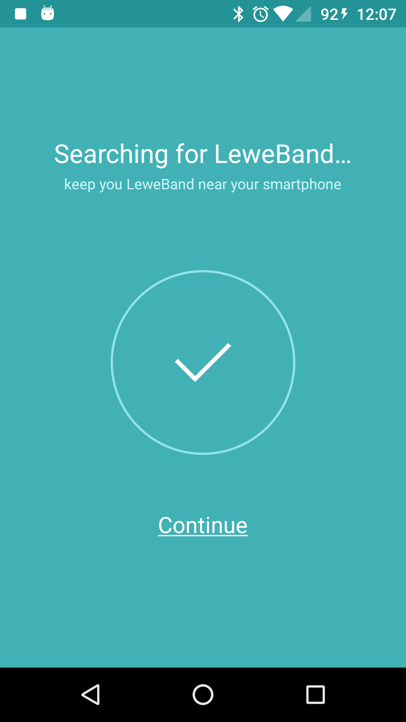
** **

Fig. 8a: Screenshot schermata di ricerca di LeweBand.

Fig. 8b: Screenshot di associazione completata.

IL PROTOCOLLO DI COMUNICAZIONE JACK

Il protocollo di comunicazione Jack (JSON ACK) è stato basato sul formato JSON data la semplicità e la facilità di lettura tipiche di questo formato, al quale sono state però applicate delle regole per la composizione dei messaggi.

Il protocollo garantisce una connessione affidabile tra mittente e destinatario implementando un sistema di conferme e mettendo a disposizione due tipologie di messaggi: *DATA* e *ACK*.

IL MESSAGGIO DATA

Questa è la tipologia di messaggio principale, ovvero quella che contiene i dati che l’host mittente desidera inviare all’host destinatario.

Data l’importanza del contenuto di questo messaggio, esso necessità di una conferma che ha il duplice significato che il messaggio sia stato ricevuto e che esso sia nel formato valido, ovvero che sia della tipologia DATA.

La struttura del messaggio è riportata di seguito:

{

“id”: ID,

“type”: “data”,

“val”:

{

“key1”: val1,

“key2: “val2,

}

}

Dove ID indica l’identificativo univoco del messaggio.

L’oggetto “val” contiene tutti i dati che il messaggio deve trasportare nella forma chiave: valore, esso non è obbligatorio e può anche essere omesso.

IL MESSAGGIO ACK

Il messaggio ACK è il messaggio di conferma che il destinatario invia al mittente per confermare la ricezione e validità di un messaggio DATA ricevuto.

Il destinatario in mancanza della conferma, allo scadere di un determinato timer, rinvierà il messaggio contenente i dati fintanto che esso non verrà confermato.

La struttura del messaggio ACK è riportata di seguito:

{

“id”: ID,

“type”: “ack”

}

Dove ID è l’identificativo del messaggio da confermare.

*Sviluppo e implementazione*

Valutazione e collaudo

La natura del progetto si è prestata ad uno sviluppo modulare in cui ogni componente è stata testata autonomamente prima di essere integrata nel sistema completo, che a sua volta è stato collaudato testando ogni sua funzionalità; particolare attenzione è stata posta all’elaborazione dei messaggi scambiati attraverso il protocollo di comunicazione.

L’hardware di gestione dei sensori è stato implementato utilizzando, come supporto di protipazione, una basetta sperimentale (breadboard) e successivamente, ottenuto il circuito definitivo adeguatamente testato, è stato costruito il PCB finale utilizzando il software Fritzing per generare lo schema del circuito.

Il firmware del bracciale è stato testato grazie all’ausilio di stringhe di log deditamente inserite all’interno dello stesso e stampate utilizzando la seriale hardware presente in Fishino UNO e visionate attraverso lo strumento “Monitor seriale” fornito all’interno dell’IDE di Arduino.

I log sono stati stampati utilizzando il seguente snippet di codice:

#ifdef DEBUG

Serial.print(F("log"));

#endif

Per disabilitare tutto il codice di collaudo, per la versione definitiva del firmware, è stata semplicemente eliminata (commentata) la direttiva al preprocessore chiamata DEBUG.

Tutte le librerie sviluppate per fornire le funzionalità richieste dal firmware del bracciale sono state sviluppate e collaudate con la stessa metodologia.

*PROTOCOLLO DI COMUNICAZIONE ARDUINO LIB => SERIALE => HM-10 connesso con app su android*

*APP => sviluppo modulare: grafica, bluetooth, database, servizi e jack onguno sviluppato e testato singolarmente con LOG*

Conclusioni e lavoro futuro

Il progetto Lewe2.0 ha visto la realizzazione di un primo prototipo di un protocollo di comunicazione nell’ambito IOT che cerca di rispettare alcuni requisiti fondamentali che sono leggerezza, flessibilità, efficienza e sicurezza.

Il prototipo ha evidenziato che il formato JSON non è adeguato a questo tipo di applicazioni in quanto la sua intrinseca lunghezza dei messaggi degrada notevolmente le prestazioni e risorse di device che, per risparmiare energia, sono dotati del minino necessario per l’esecuzione del processo a loro assegnato.

Dall’altra parte un formato interamente binario risulta sicuramente una scelta migliore per questi contesti applicativi, limitando però la comprensione umana dei messaggi scambiati tra i vari host e rendendo di conseguenza difficoltoso il processo di debugging.

Probabilmente un formato per questo protocollo dovrà essere intermedio tra il binario e il JSON ricucendo, in questo modo, le risorse necessarie all’elaborazione senza degradare troppo la leggibilità umana degli stessi.

Un altro aspetto che è stato evidenziato da questo progetto è la necessità di ampliare le funzionalità del protocollo di comunicazione inserendo alcune tipologia di messaggi per poter fornire supporto a diversi contenti applicativi differenti da quello preso in esame.

Alcune di queste integrazioni sono l’inserimento di messaggi la cui bassa rilevanza non necessità conferma da parte dell’host destinatario e una speciale tipologia di messaggi che dia la possibilità ai device di indicare alla rete la loro presenza e le funzionalità messe a disposizione della stessa.

Tutto il materiale presentato in questo documento è stato posto sotto licenza Apache 2.0 (dove non diversamente segnalato) ed è fruibile gratuitamente ai seguenti indirizzi:

Applicazione per smartphone Android

(<https://github.com/alessandro1105/Lewe2.0App)>

Firmware e schemi per il bracciale

(<https://github.com/alessandro1105/Lewe2.0)>

Libreria HashMap per Arduino IDE (<https://github.com/alessandro1105/HashMap_Arduino_Library)>

Implementazione del protocollo Jack per Arduino IDE (<https://github.com/alessandro1105/Jack_Arduino_Library)>

Libraria SoftwareSerialJack per il protocollo Jack

(<https://github.com/alessandro1105/SoftwareSerialJack_Arduino_Library)>

Tutte le componenti del progetto sviluppate da terzi sono di loro proprietà ed è quindi necessario fare riferimento ai relativi sviluppatori per qualsiasi concessione sull’utilizzo di tale materiale.