POLITECNICO DI MILANO

SCUOLA DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

SOCIAL WIRELESS - STM32F4

Progetto di real time operating systems

Sebastiano Mariani, Fabio Gritti

Contents

1 Introduzione

Lo scopo principale del progetto è quello di sviluppare un driver che permetta di utilizzare il transceiver wi-fi NRF24L01P dalla board STM32F4_Discovery. Una volta sviluppato tale modulo è prevista l'integrazione con i lavori svolti dai colleghi per permettere l'utilizzo nel contesto voluto dalla traccia. (comunicazione bidirezionale dei passi fatti e feedback audio tra due dispostivi che si trovano nelle vicinanze)

I moduli integrati all'interno del nostro lavoro sono:

- **Wifi:**
  - Thread che permette la comunicazione wifi tra due dispostivi vicini
- **Podometro:**
  - Thread che si occupa di rilevare e contare i passi fatti
- **Suono:**
  - Libreria che permette di avere feedback audio circa la vittoria o la sconfitta in base al numero di passi effettuati ed una indicazione del numero di passi fatti fino a quel momento

2 Note e Strumenti utilizzati

- **Note:**
  - Nel contesto dell'applicazione vengono trasmessi solo il numero di passi effettuati, ragion per cui si è scelto di fissare il payload in modo statico a 4 byte.
  - Il thread di valutazione statistiche del modulo podometro non è stato attivato dato che per il livello di integrazione fatto risultava superfluo (è possibile comunque attivarlo senza problemi)
- **Strumenti Utilizzati:**
  - STM32F4 Discovery: board con architettura ARM a 32 bit, fornitaci da STMicroelectronics©
  - Software per il controllo di versione git con repository remota hostata sul portale Github
  - Modulo wifi NRF24L01P prodotto da Nordic Semiconductor©
  - Sistema operativo open source per sistemi embedded Miosix sviluppato dall'Ing. Federico Terraneo
  - QSTLink2: programma sviluppato da STMicroelectronics© che permette di flashare eseguibili sulla board
  - Cavo seriale

3 Funzionamento NRF24L01P

Mostriamo in questa sezione l'automa a stati finiti che riassume le varie fasi di lavoro del transceiver wifi, descrivendo successivamente come il nostro codice si comporta rispetto a questo.

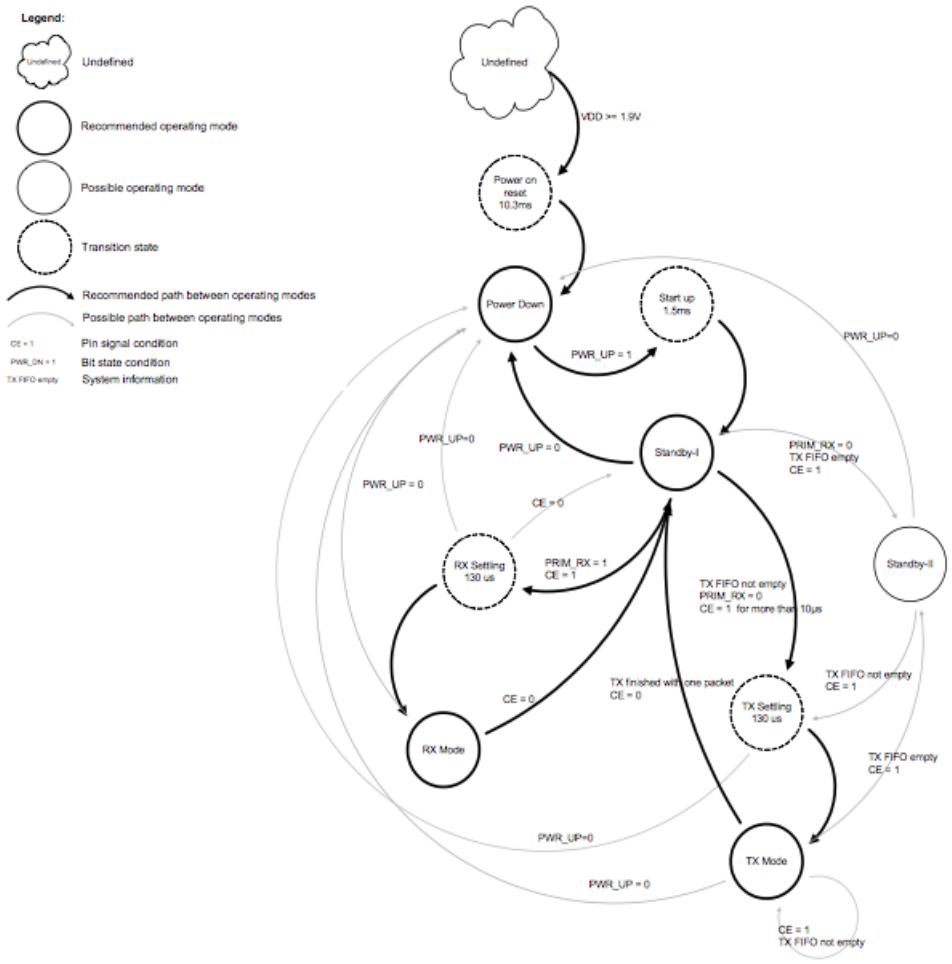

Figure 1: Schema di funzionamento NRF24L01P

Il codice è stato strutturato in questo modo:

- Passaggio da stato Power down a stato Standby I tramite funzione *powerUp()*
- Passaggio da Stabdby I a RX mode tramite funzione *setReceiveMode()*
- Passaggio da RX mode a TX mode passando da Standby I tramite funzione *TrasmitData()* che al suo interno invoca la funzione *setTransmitMode()*

- Il passaggio da TX Mode a RX mode viene affidato sempre alla funzione *TrasmitData()*, la quale quando conclude la trasmissione si preoccupa di riportare lo stato del modulo in RX mode passando da StandbyI

Per rendere più semplice il codice questo è stato pensato affinchè si evitasse il passaggio allo stato Standby II che, come riportato nel reference manual di NRF24L01P, risulta essere uno stato ibrido (Non si ha niente da trasmettere ma sono in trasmissione) .

4 Descrizione Classi

In questa sezione viene data una breve overview delle classi create, successivamente verrà spiegato come i thread si sincronizzano per arrivare al risultato descitto nell'introduzione. Le varie classi sono state suddivise in cartelle, una per modulo, per rendere più comprensibile la struttura del progetto e di conseguenza renderla più navigabile. La suddivisione è la seguente:

- Wifi-Module
- Pedometer
- Speaker + suoni (cartella contenente i campioni dei suoni utilizzati)

4.1 Wifi-Module

Contiene tutte le classi C++ per il funzionamento del transceiver NRF24L01P, più alcuni file utili a scopi di debugging. Nello sviluppo di questo modulo si è cercato di mantenere una linea molto semplice ed intuitiva in modo da permettere ad eventuali sviluppatori futuri di usare la classe con facilità. (Sono stati inseriti numerosi commenti nei passi più critici)

- **NRF24L01P:**
  - classe contenente le funzioni strettamente legate al setup e alle operazioni per permettere la trasmissione dei dati wireless. Nel file header sono state definite tutte le macro che permettono un utilizzo user-friendly dei registri e delle istruzioni di configurazione.
- **Spi:** implementa un driver minimale per trasmettere i dati al modulo wifi tramite l'interfaccia spi della board.
  - costruttore: esegue il setup del bus spi in modalità full-duplex
  - send(int data): permette di inviare dati tramite bus spi
  - receive(): permette di ricevere dati tramite bus spi
- **exorcizo.cpp:**
  - binario che compie un reset di tutti i registri della board. Molto utile in fase di debug
- **mainRX.cpp - mainTX.cpp:**
  - files C++ derivanti da una prima versione del progetto in cui una board si occupava solamente della trasmissione e l'altra solo della ricezione. Molto utile per capire a fondo i meccanismi del modulo Wifi

4.2 Pedometer

- **Pedometer:**
  - classe contenente le funzioni che permettono il setup iniziale del podometro e la lettura dei dati rilevati.
- **lis302dl:**
  - Si occupa del setup dell'accellerometro per la rivelazione dei movimenti sui 3 assi cartesiani
- **stats:**
  - classe in grado di ricavare statistiche, come ad esempio il calcolo delle calorie consumate

4.3 Speaker

- **Player:**
  - classe contenente le funzioni che permettono lo stream corretto dell'audio
- **slice-and-play:**
  - Mette a disposizione le funzioni che permettono di avere feedback audio di vittoria, sconfitta e di speaking del numero di passi fatti.
- **convert:**
  - classe in grado di convertire i file campionati da formato .wav a formato .h per un eventuale aggiunta di nuovi suoni.

5 Threads e Sincronizzazione

In questa ultima parte del documento spieghiamo come sono stati integrati tra di loro i moduli per raggiungere la funzionalità richiesta dal dispositivo in sviluppo. Per implementare lo stato di ricezione/trasmissione pseudo-contemporanea sono stati utilizzati due thread che contendono una mutex ('modality'). Tale mutex si occupa di arbitrare l'accesso al transceiver garantendo la configurazione in mutua esclusione tra ricezione e trasmissione. I due thread utilizzati dal nostro modulo sono:

- *send_handler(void* arg)*: Si occupa di passare in stato di trasmissione e di trasmettere ogni secondo
- *irq_handler(void* arg)*: rimane in attesa di un interrupt di ricezione; quando questo si verifica si occupa di effettuare il confronto tra i dati ricevuti ed il numero di passi corrente rilevati dal podometro e successivamente richiama la corrispondente funzione dello speaker per informare l'utente della sua vittoria/sconfitta.

Sono stati mascherati gli interrupt di trasmissione (TX_DS) e di massimo numero di ritrasmissioni raggiunte (MAX_RT) perchè poco utili al fine del progetto. Seguno ora i flow-chart che descrivono dettagliatamente come i thread di trasmissione e ricezione cooperano all'interno del sistema.

5.1 Thread Send_handler

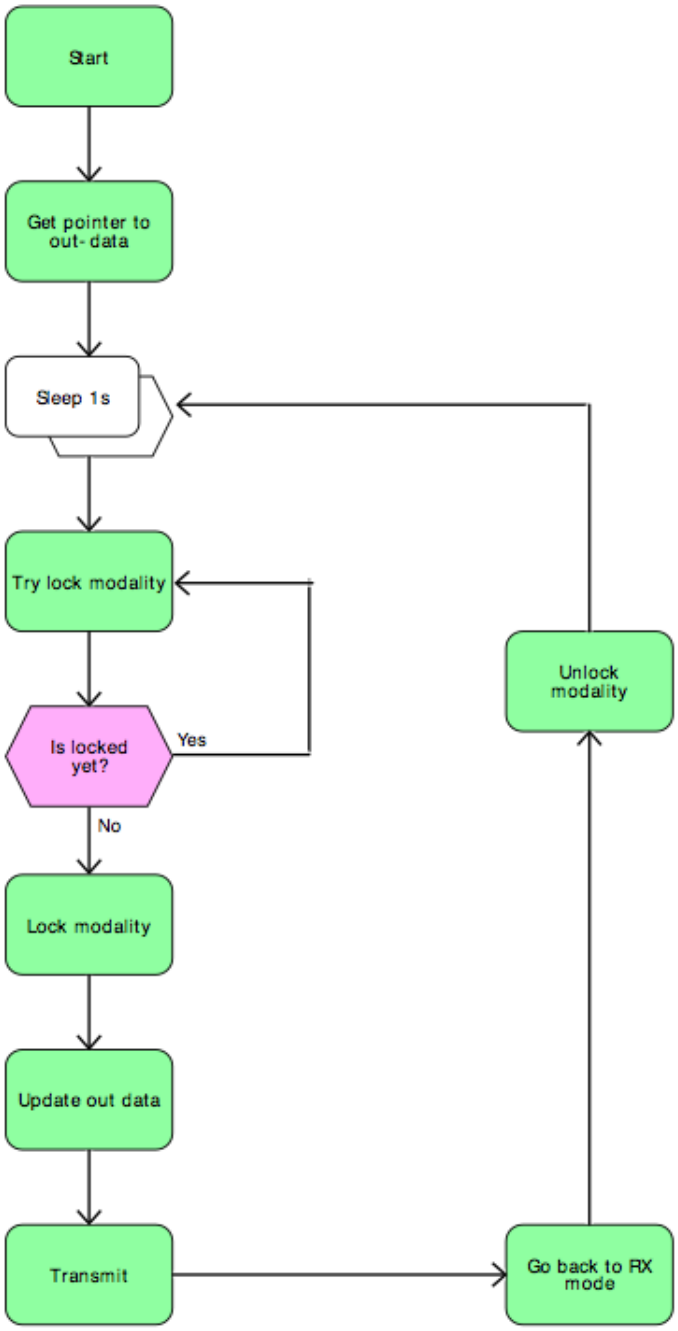

Figure 2: Flow chart thread trasmissione

5.2 Thread Irq_handler

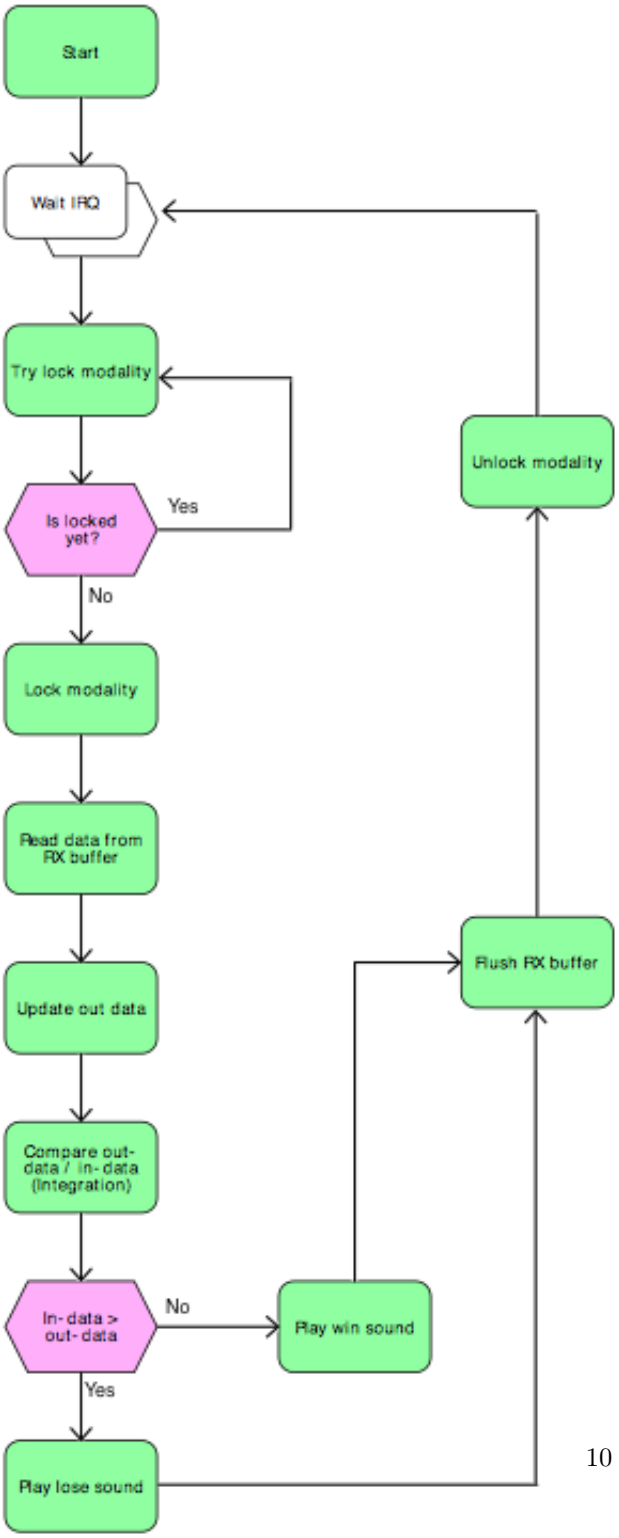

Figure 3: Flow chart thread ricezione

5.3 Bottone

Come ultima feature del sistema è stato aggiunto un thread in attesa di un interrupt da parte del bottone (blu sulla board). Quando questo si verifica viene richiamata la funzione dello speaker che riproduce il numero di passi fatti fino a quel momento.

5.4 Overview generale

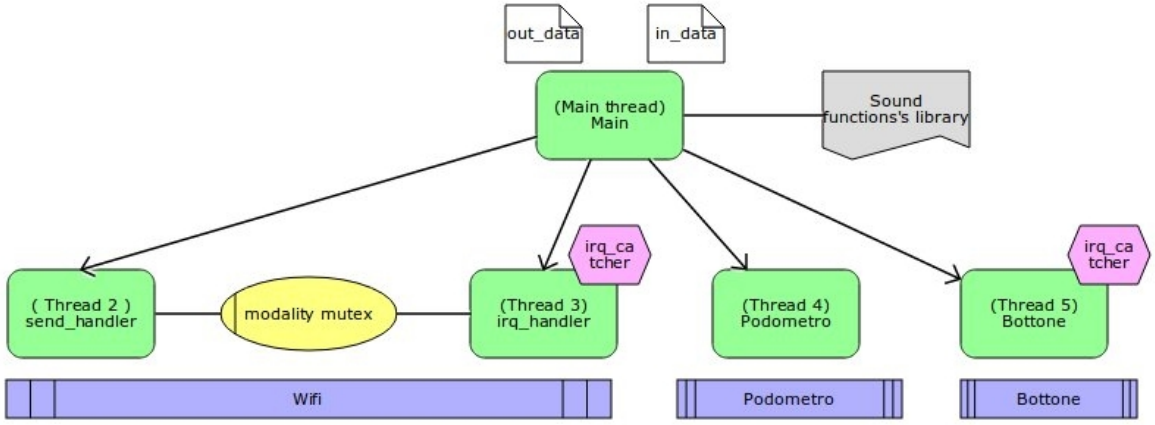

Figure 4: Schema generale

6 Conclusioni

Come già ripetuto più volte si è cercato di mantenere il codice più semplice e pulito possibile rispetto al contesto dell'applicazione sviluppata; l'implementazione fatta risulta essere abbastanza riduttiva rispetto alle potenzialità del modulino wifi, sono infatti rimaste inutilizzate feature come : multiceiver, payload dinamico, ack e ritrasmissioni automatiche; Non si riteneva tuttavia necessario l'utilizzo di queste per il progetto in questione. Il codice prodotto risulta essere comunque un buon punto di partenza per implementare una generica applicazione che richieda una trasmissione wifi di dati, con delle piccole modifiche alla classe è possibile implementare tutte le funzionalità mancanti senza troppi problemi.

7 Ringraziamenti

Si ringraziano le seguenti persone che hanno sviluppato il codice di speaker e podometro e che hanno reso possibile la creazione del dispositivo descritto in questo documento:

- Andrea Piscitello - Giada Tacconelli (Podometro)
- Marco Mezzanotte - Filippo Garetti - Tommaso Calcina (Speaker)