Alma Mater Studiorum
Università di Bologna
Campus di cesena

dipartimento di ingegneria dell'energia elettrica e dell'informazione
"Guglielmo marconi"

Corso di laurea in Ingegneria elettronica per l'energia e l'informazione

# Nodi sensori LoRaWAN a micropotenze: progettazione hardware, firmware e a livello di applicazione

*Elaborato in*

Elettronica dei sistemi digitali

*Relatore:*

Prof. Aldo Romani

*Presentata da:*

Mattia Chiappalone

*Correlatore:*

Simone Sindaco

Anno Accademico 2021/2022

SESSIONE I

# Indice

# Introduzione

Una rete wireless di sensori consiste in un grande numero di nodi-sensore, definiti end-node, che collezionano informazioni dall'ambiente circostante, le processano e le comunicano a nodi appartenenti alla rete nelle vicinanze. Questi end-node sono di norma alimentati a batteria e dato che non sempre è possibile ricaricarle e il loro rimpiazzo può essere un problema in alcuni tipi di applicazione, il consumo di potenza dell'end-node diventa un parametro molto importante di progetto.

All'interno di questo elaborato si parlerà del protocollo di comunicazione LoRaWAN, che permette lo scambio di informazione a bit-rate limitata, coprendo distanze elevate e sfruttando poca energia, rendendosi una scelta ottima per applicazioni in ambito IoT.

L'obiettivo di questo elaborato è il progetto di un nodo-sensore LoRaWAN a micro-potenze, caratterizzato da una corrente di sleep nell'ordine dei nano-Ampere in modo tale che sia possibile raggiungere tempi di vita di molteplici anni prima che la batteria si scarichi o si degradi naturalmente.

Nel primo capitolo si discute la modulazione LoRa, i suoi parametri principali, vantaggi e svantaggi. Si parlerà del protocollo LoRaWAN spiegando i meccanismi di comunicazione tra i vari nodi.

Il secondo capitolo intende descrivere l'architettura di riferimento dalla quale si è partiti per realizzare il progetto, ovvero l'end-node utilizzato inizialmente, una scheda di sviluppo Microchip, il gateway Dragino e l'Application Server Chirpstack. Si collauda il corretto funzionamento dell'applicazione, ossia dalla lettura di dati dall'end-node fino alla decodifica e visualizzazione a lato server degli stessi.

Il terzo capitolo elenca e spiega i comandi seriali principali del Modem Microchip RN2483, utilizzato come parte chiave dell'end-node nel capitolo precedente.

Nel capitolo quattro è stata trattata l'ottimizzazione energetica del nodo a livello Firmware.

Il capitolo cinque si occupa di elencare tutti gli accorgimenti necessari attraverso vari livelli di astrazione per ottenere consumi nell'ordine delle centinaia di nano-Ampere in stato di sleep. Verranno eseguite stime di vita del sensore in funzione del tempo di trasmissione dei dati e di potenza di trasmissione del segnale.

Verrà infine mostrato lo schematico e layout del PCB pronto per il processo di fabbricazione, concludendo con gli sviluppi futuri e conclusioni.

# Capitolo 1

# 1. Il protocollo LoRaWAN

Con il termine LoRaWAN, si intende il protocollo per la comunicazione e tutte le regole per instaurare una comunicazione bidirezionale end-to-end che sfrutta la tecnologia LoRa per trasmettere i dati a grande distanza con un basso dispendio di energia, rendendosi quindi promettente per applicazioni di energy harvesting o alimentate a batteria.

## 1.1 La modulazione LoRa

LoRa, abbreviativo di Long Range, è una tecnologia proprietaria di Semtech che si basa sulla modulazione Chirp Spread Spectrum (CSS). Una modulazione sviluppata intorno al 1940 per le comunicazioni militari, diventata poi comune a scopi civili grazie al suo basso consumo di trasmissione, robustezza ai disturbi causati dai cammini multipli, effetto doppler e jamming. Lo strato fisico del modello ISO-OSI della modulazione CSS è entrato a fare parte dello standard IEEE per le Low-Rate Wireless Personal Area Networks (LR-WPANs) 802.15.4. Ulteriori dettagli si possono trovare nel documento [1].

### 1.1.1 Chirp Spread Spectrum CSS

La modulazione è caratterizzata dai cosiddetti chirp, che rappresentano la variazione lineare in frequenza dal segnale portante e possono essere di due tipi:

- up-chirp (se la frequenza della portante aumenta)
- down-chirp (se la frequenza diminuisce)

Come si vede nella figura seguente, nella modulazione LoRa il chirp varia nell'intera banda assegnata al canale (tipicamente 125 kHz, 250 kHz per l'Europa e fino a 500 kHz in altri paesi).

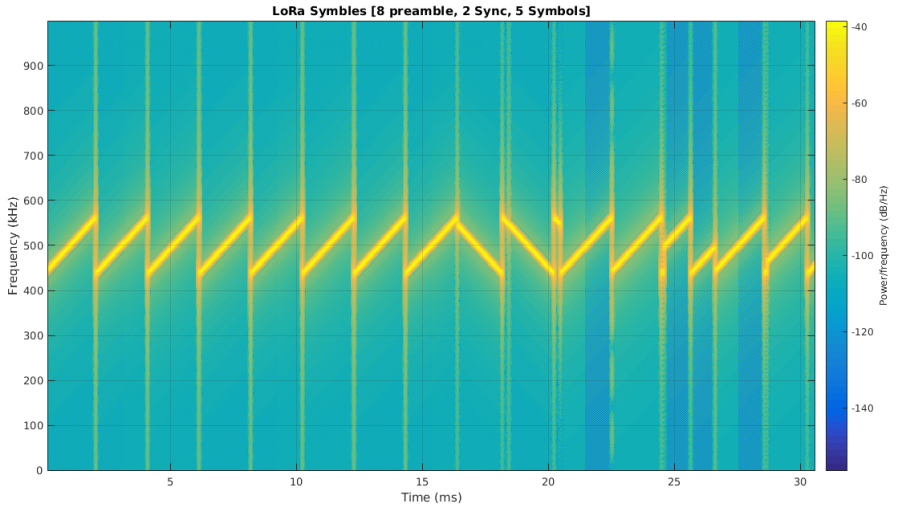



*Figura 1.1 Esempio di pacchetto LoRa frequency vs time, fonte All About LoRa and LoRaWAN S. Ghoslya [2]*

### 1.1.2 Lo spreading factor

Un altro parametro importante previsto nella modulazione LoRa è lo spreading factor (SF), il quale può variare da un valore di 7 fino a 12.

Questo parametro incide sulla pendenza della rampa del chirp. Infatti, ogni incremento unitario di SF corrisponde al raddoppio del tempo necessario a spedire il simbolo e di conseguenza un dimezzamento della bit rate.

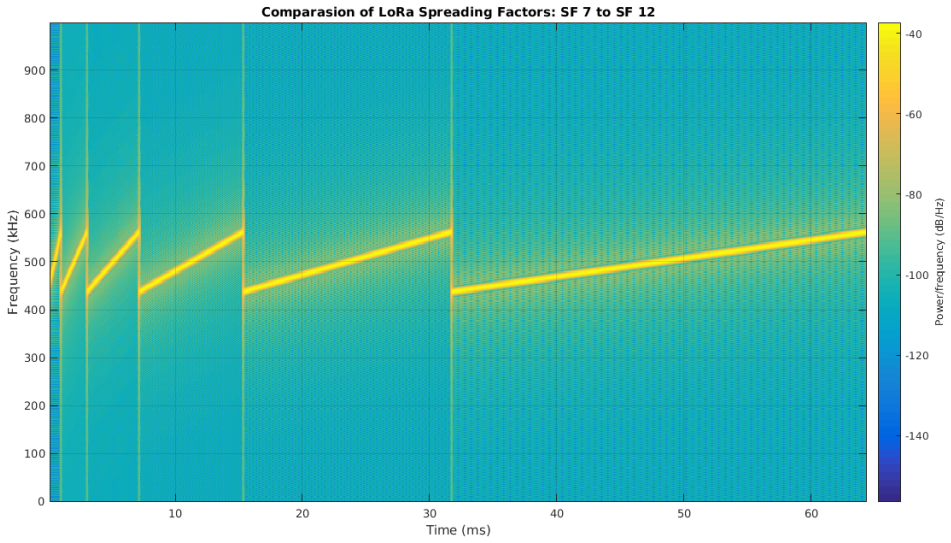



*Figura 1.2 Confronto di diversi spreading factor, fonte All About LoRa and LoRaWAN S. Ghoslya [2]*

Uno SF maggiore, implica quindi un tempo di trasmissione maggiore, aumentando i consumi dell'end-device e riducendo la vita della batteria. Tuttavia, a lato ricevitore, è semplificata la demodulazione del segnale, permettendo così di aumentare la distanza tra trasmettitore e ricevitore.

Per l'ottimizzazione del tempo di vita della batteria è necessario quindi utilizzare lo SF opportuno, tale da permettere la demodulazione corretta dal ricevitore, impiegando il minimo tempo di trasmissione possibile.

### 1.1.3 Data Bit Rate

La modulazione LoRa presenta un trade-off in quanto permette di raggiungere distanze elevate impiegando piccole potenze, al prezzo di sacrificare la bit rate.

Fatte le precedenti considerazioni sullo SF, si può definire il periodo di simbolo $T_s$

$$T_s = \frac{2^{SF}}{BW}[s] \tag{1.1}$$

e di conseguenza il suo reciproco: Symbol rate $R_s$

$$R_s = \frac{1}{T_s} = \frac{BW}{2^{SF}}[simboli/s] \tag{1.2}$$

Gli up-chirps e down-chirps sono elementi base della modulazione. La banda disponibile viene suddivisa in $2^{SF}$ intervalli di uguale larghezza e l'informazione è codificata nella frequenza iniziale con la quale l'up-chirp viene trasmesso. Per esempio, con SF = 7, sono ammessi i valori tra 0 e 127. Per maggiore chiarezza facciamo riferimento alla seguente figura, dove si ipotizza bandwidth BW = 125 kHz e SF = 7.

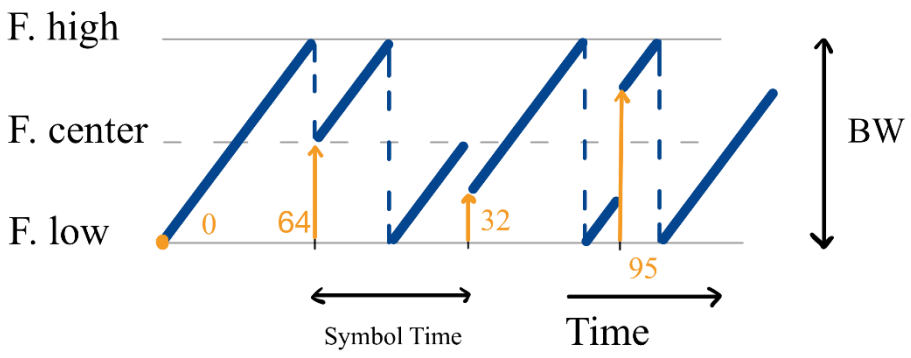



*Figura 1.3 Codifica Simboli Modulazione LoRa*

In Figura 1.3 sono stati inviati 4 simboli, aventi corrispettivamente valore 0, 64, 32, 95. Si può quindi vedere che nel caso SF = 7, ogni simbolo porta 7 bit di informazione.

In maniera del tutto generale è possibile affermare che ogni simbolo è in grado di portare $SF$ bit di informazione, è immediato ricavare la bit rate come $R_b$

$$R_b = SF \cdot \frac{BW}{2^{SF}} [bits/sec] \tag{1.3}$$

Tuttavia, la modulazione LoRa prevede uno schema di correzione di errori che aggiunge dei bit di ridondanza che non portano informazione. La bit rate effettiva, influenzata sia dallo SF, che banda utilizzata e code rate CR diventa quindi

$$R_b = SF \cdot \frac{\frac{4}{4+CR}}{\frac{2^{SF}}{BW}} [bits/sec] \tag{1.4}$$

Dove:

$R_b$ = bit rate
SF = spreading factor ∈ [7,12]
CR = code rate: numero di bit di ridondanza ∈ [1,4]
BW = bandwidth ∈ (125, 250, 500 kHz)

Nella seguente tabella si possono osservare alcune bit rate al variare di SF, per dare un'idea dei valori che si possono raggiungere con questo tipo di tecnologia (più bassa la sensitivity e maggiore la distanza coperta dal segnale). Per ulteriori informazioni riguardo la modulazione LoRa, visitare il documento [1].

| *Modalità* | *Bit rate equivalente [kb/s]* | *Sensitivity [dBm]* |
|---|---|---|
| SF = 12 | 0.293 | -137 |
| SF = 11 | 0.537 | -134.5 |
| SF = 10 | 0.980 | -132 |
| SF = 9 | 1.76 | -129 |
| SF = 8 | 3.125 | -126 |
| SF = 7 | 5.469 | -123 |

*Tabella 1.1 Bit rate al variare dello spreading factor*

### 1.1.4 La struttura dei pacchetti LoRa

Un frame LoRa è formato da un preambolo, un header (opzionale), il payload e il CRC (cyclic redundancy code, anch'esso opzionale). Si faccia riferimento alla Figura 1.4.

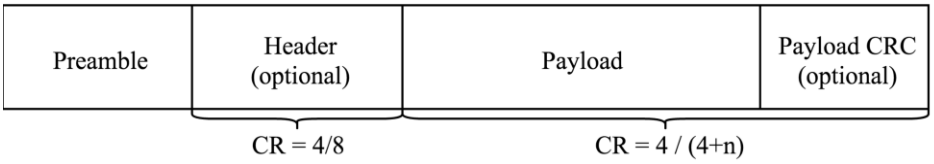



*Figura 1.4 Struttura di un frame LoRa*

Il preambolo è costituito da una sequenza costante di up-chirp che ricoprono l'intera banda di frequenza utilizzata (di default sono otto). Gli ultimi due up-chirp codificano la sync word: utilizzata per differenziare diverse LoRa networks all'interno della stessa banda. Un end-node configurato con una certa sync word, smetterà di ascoltare il pacchetto se la decodifica di quest'ultima è diversa da quella impostata dal nodo. Dopo la sync word seguono due down-chirp.

L'header, quando è presente, viene trasmesso con un code rate di 4/8 (dove il primo numero indica i bit che portano informazione, mentre il secondo la somma dei bit di ridondanza e informazione). Questa sezione del frame, indica il numero di bytes del payload, la code rate utilizzata per la fine della trasmissione e se è presente o meno un CRC a 16 bit per il payload.

L'equazione (1.5), derivata dal documento [3], mostra il numero di simboli necessari da trasmettere per un determinato payload:

$$n_s = 8 + max\left(ceil\left[\frac{8PL - 4SF + 28 + 16CRC - 20H}{4(SF - 2DE)}\right](CR + 4), 0\right) \qquad (1.5)$$

Dove:

PL = dimensione in byte del payload (1,255)
SF = spreading factor (7,12)
CRC = 1 se il CRC è abilitato, 0 altrimenti
H = 0 quando l'header è abilitato, 1 altrimenti
DE = 1 quando low data rate optimization è abilitato, 0 altrimenti
CR = coding rate (1 se fissato a 4/5, fino a 4 se fissato a 4/8)

Possiamo calcolare ora la durata di trasmissione del payload moltiplicando la (1.5) per il tempo di trasmissione di un simbolo ricavato nella (1.1)

$$T_{payload} = T_s \cdot n_s = \frac{2^{SF}}{BW} \cdot n_s \tag{1.6}$$

Dal documento [3] è riportata inoltre la durata del preambolo, calcolata come segue

$$T_{preamble} = (n_{preamble} + 4.25) \cdot T_s \tag{1.7}$$

dove

$n_{preamble}$ = numero di simboli per il preambolo (di default 8)

Possiamo ora calcolare il Time on Air (TOA), ovvero il tempo di trasmissione per un generico pacchetto moltiplicando la (1.5) per la (1.1)

$$TOA = T_{preamble} + T_{payload} \tag{1.8}$$

Il time on air è un parametro importante, in quanto incide sui consumi energetici che un determinato nodo assumerà e incide sul tempo nel quale una sotto banda di frequenza viene occupata. Come si vedrà nel paragrafo 1.2.4, sarà necessario cercare di minimizzare il più possibile tale valore.

### 1.1.5 Le bande di frequenza utilizzate

Le bande di frequenza utilizzate appartengono tutte alla banda ISM (Industrial Medical Scientific) le quali non prevedono uso di licenza sotto certe potenze. In Europa le frequenze sono 433 MHz e 868 MHz mentre in USA 915 MHz. Facciamo riferimento a [4] per le sotto bande disponibili con relative potenze e duty cycle di trasmissione massima consentita.

| *Sub-band* | *Frequenza [MHz]* | *Duty Cycle* |
|---|---|---|
| G | (863.0 – 868.0) | 1% |
| G1 | (868.0 – 868.6) | 1% |
| G2 | (868.7 – 869.2) | 0.1% |
| G3 | (869.4 – 869.65) | 10% |
| G4 | (869.7 – 870.0) | 1% |

*Tabella 1.2 Sotto bande e frequenze in Europa*

Su tutte queste bande si ha una potenza massima di trasmissione di 25 mW ad eccezione della G3 che permette 500 mW.

## 1.2 LoRaWAN

LoRaWAN, abbreviativo di Long Range Wide Area Network è un protocollo che prevede un'architettura a stella di stelle: come si vede in Figura 1.5, prevede una comunicazione tra end-nodes e gateway tramite lo strato fisico LoRa. I gateways sono a loro volta connessi ad internet tramite lo standard TCP/IP e si comportano come dei bridge trasparenti, ovvero convertono i pacchetti RF in IP e viceversa.

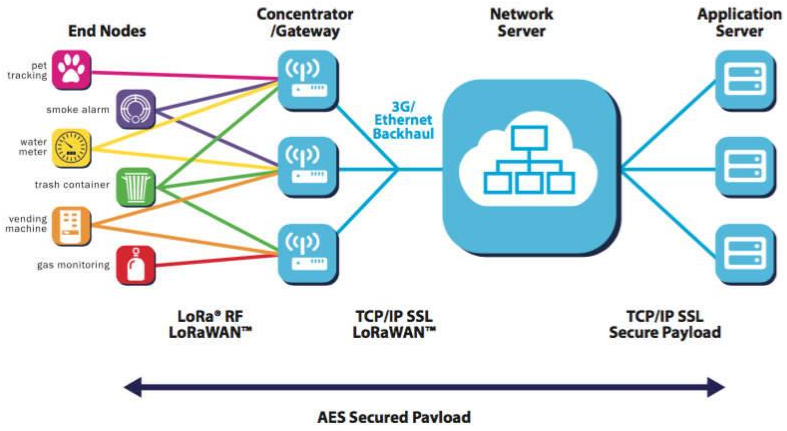



*Figura 1.5 Architettura LoRaWAN, fonte The Things Network [18]*

I pacchetti trasmessi da un singolo end-node possono essere ricevuti da uno o più gateways. Essi saranno semplicemente inoltrati tramite protocollo IP all'utente finale, indicato con il termine "Application Server". Il vantaggio di questa struttura è di spingere la complessità al Network Server, che si occuperà di filtrare pacchetti ridondanti, eseguire controlli di sicurezza, gestire l'adaptive data rate, indirizzarli al corretto Application Server (ovvero l'utente finale), etc.

Gli end-nodes in una rete LoRaWAN sono asincroni e possono comunicare in un qualsiasi istante, senza bisogno di sincronizzarsi con i gateway (Aloha method). Questa fase di sincronizzazione, necessaria in altre tecnologie, necessiterebbe di significanti quantità di energia, riducendo drasticamente la vita della batteria.

La comunicazione tra end-node e gateway può essere bidirezionale, in particolare viene definito:

- Uplink: trasmissione da end-node verso gateway
- Downlink: trasmissione da Application Server verso end-node

### 1.2.1 Device Classes

Le specifiche LoRaWAN prevedono tre differenti classi di appartenenza per gli end-devices, ognuna per un differente tipo di applicazione:

- Classe A – Lowest power, bi-directional end-devices

  È la classe di default. Prevede che ogni singola comunicazione sia iniziata dall'end-device in maniera asincrona. Dopo aver trasmesso i dati, l'end-node si mette in ascolto per due brevi intervalli di tempo, per dare opportunità ad un eventuale gateway di effettuare un downlink e quindi assicurare un tipo di comunicazione bidirezionale.

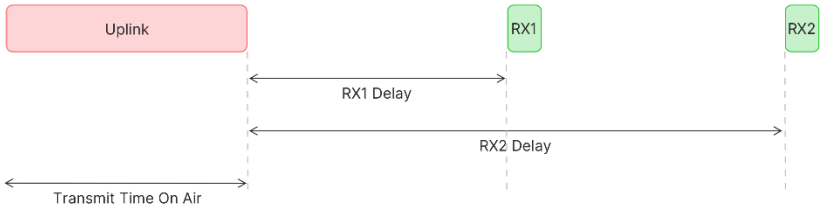



*Figura 1.6 Esempio funzionamento nodo classe A LoRaWAN*

- Classe B – Bi-directional end-devices with deterministic downlink latency

  In aggiunta alla classe A, questo tipo di classe è in grado di aprire delle finestre di ricezione programmate ad intervalli prestabiliti fino ad ogni 128 secondi. In questo modo, un gateway è in grado di effettuare un downlink senza dover aspettare che l'end-node inizi la comunicazione.
  Per rendere questo possibile, un beacon viene periodicamente inviato dal gateway in modo tale che l'end-device in ascolto possa sincronizzarsi con il network. Grazie al riferimento del beacon, l'end-device può aprire periodicamente delle finestre di ricezione chiamate ping slots. Ognuno dei ping slot, può essere utilizzato dal gateway per iniziare un downlink.
  Bisogna tuttavia considerare un consumo di potenza maggiore, richiesto dalle finestre di ricezione aperte dal nodo.

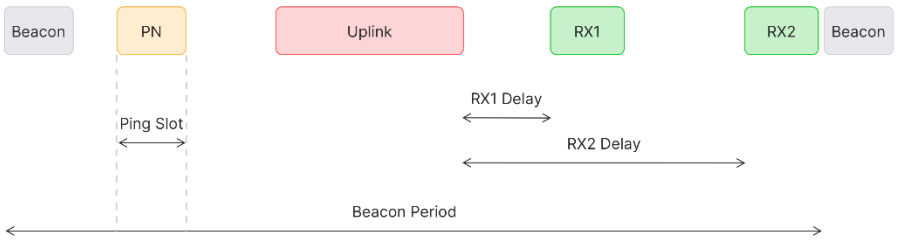



*Figura 1.7 Esempio funzionamento nodo classe B LoRaWAN*

- Classe C – Lowest Latency, bi-directional end-devices

  In aggiunta alla classe A, questo tipo di classe minimizza la latenza del downlink tenendo sempre aperta una finestra in ricezione. In questo modo il gateway può iniziare la comunicazione in qualsiasi istante in quanto assume che il dispositivo sia sempre in ascolto.
  Essendo questa la classe più energivora delle tre, si ricorda che un end-node è sempre in grado di cambiare la propria classe e quindi passare da una A ad una C in caso di necessità.

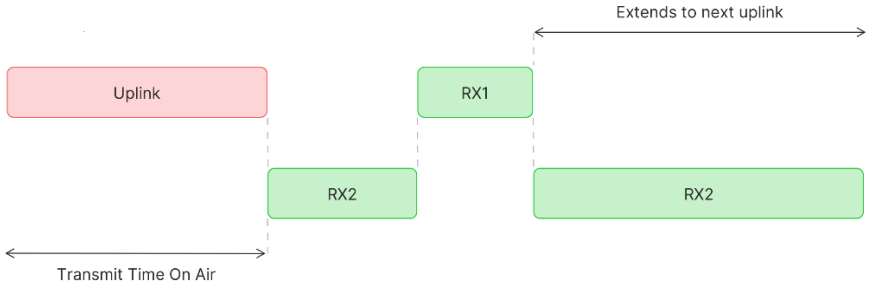



*Figura 1.8 Esempio funzionamento nodo classe C LoRaWAN*

Fatte queste considerazioni, all'interno di questo elaborato verrà utilizzata la classe A, in quanto si vuole privilegiare il risparmio energetico.

### 1.2.2 Indirizzamento e metodi di attivazione

Per poter partecipare ad una rete LoRaWAN un end-device deve prima essere attivato. Da protocollo, esistono due diversi metodi di attivazione: Over-The-Air Activation (OTAA) e Activation By Personalization (ABP).

Per la modalità ABP bisogna fornire all'end-device le seguenti informazioni:

- **DevAddr** : End-device address. Un identificativo a 32 bit, dove 7 sono utilizzati come network identifier e i restanti 25 come indirizzo di rete dell'end-device.
- **NwkSkey** : (Network Session Key). Una chiave a 128 bit utilizzata dal Network Server e dall'end-device per calcolare il MIC (Message Integrity Code) e verificare l'integrità dei dati ricevuti.
- **AppSkey** : (Application Session Key). Una chiave a 128 bit utilizzata dall'end-node e dall'Application Server per criptare/decriptare il frame payload.

Per la modalità OTAA, una procedura di join comprendente una join-request e join-accept è utilizzata per scambiare ogni volta la NwkSkey e AppSkey. Questa procedura si rivela più sicura della precedente in quanto i nodi devono seguire una procedura di associazione dove si scambiano le chiavi con il Network Server.

In questo elaborato sarà utilizzata la procedura ABP e le chiavi sopra citate verranno programmate all'interno dell'end-node in quanto l'obiettivo è limitare la complessità assieme ad una riduzione dei consumi di potenze.

### 1.2.3 Sicurezza nelle reti LoRaWAN

Le reti LoRaWAN soddisfano i tre pilastri della sicurezza: autenticità, integrità e confidenzialità. Viene utilizzata una codifica AES che cifra i dati inviati dall'end-node in modo che solamente l'utente finale possa leggere i dati in chiaro. L'integrità viene invece offerta dal MIC (Message Integrity Code).

### 1.2.4 Duty Cycle e Data Rate

Nella Tabella 1.3 sono indicati il duty cycle per ogni sotto banda disponibile nel range di frequenze di LoRa. Ogni banda è suddivisa in canali: per esempio la banda G (863.0 – 868.0 MHz) e G1 (868.0 – 868.6) contiene i seguenti canali dedicati all'uplink:

| *Canale* | *Frequenza [MHz]* | *SF minimo* | *SF massimo* | *Banda [kHz]* |
|---|---|---|---|---|
| 1 | 868.1 | 7 | 12 | 125 |
| 2 | 868.3 | 7 | 12 | 250 |
| 3 | 868.5 | 7 | 12 | 125 |
| 4 | 867.1 | 7 | 12 | 125 |
| 5 | 867.3 | 7 | 12 | 125 |
| 6 | 867.5 | 7 | 12 | 125 |
| 7 | 867.9 | 7 | 12 | 125 |

*Tabella 1.3 Canali nella banda G e G1 EU*

All'interno di tutti questi canali, il duty cycle è limitato all'1%: questo parametro limita il tempo massimo per il quale un end-device può occupare un singolo canale.

Per esempio, se il tempo di trasmissione di un pacchetto è noto e pari a TOA, ricavato nell'equazione (1.8), allora un end-device dovrà aspettare un tempo uguale a $99 \cdot TOA$ prima di poter trasmettere nuovamente in questo canale.

Quando un end-node deve trasmettere un pacchetto, il protocollo LoRaWAN prevede che il canale nel quale trasmette, cambi ogni volta, in modo tale da minimizzare il più possibile il duty cycle a livello di singolo canale.

Altri limiti possono essere imposti, per esempio la community The Things Network (una piattaforma online che offre gratuitamente e open-source l'infrastruttura basata sul protocollo LoRaWAN [19]) impone una *Fair Use Policy* che limita l'uplink TOA a 30 secondi per nodo per giorno.

Per motivi pratici, si utilizza la denominazione di data rate per indicare la combinazione di SF e BW utilizzati dall'end-node. Da ora in poi sarà comodo fare riferimento ai valori della seguente tabella.

| *Data Rate* | *Configurazione* | *Bit rate indicativa [bit/s]* | *TOA indicative [ms]* |
|---|---|---|---|
| 0 | SF12 – 125 kHz | 250 | 991 |
| 1 | SF11 – 125 kHz | 440 | 557 |
| 2 | SF10 – 125 kHz | 980 | 288 |
| 3 | SF9 – 125 kHz | 1760 | 144 |
| 4 | SF8 – 125 kHz | 3125 | 72 |
| 5 | SF7 – 125 kHz | 5470 | 41 |

*Tabella 1.4 Legame tra Data Rate, SF, bit rate e TOA*

### 1.2.5 Adaptive Data Rate ADR

L'ADR è un meccanismo che fornisce LoRaWAN per aumentare la capacità della rete e migliorare le prestazioni energetiche dei singoli nodi.

Come si vede dalla Tabella 1.1, all'aumentare dello SF, diminuisce anche la sensitivity, che in maniera pratica si riflette sulla massima distanza che si può porre tra un end-node e un gateway. Tuttavia, come si è appena osservato dalla Tabella 1.4, aumenta sensibilmente anche il TOA del singolo pacchetto.

L'obiettivo dell'ADR è quindi minimizzare il data rate in modo tale da rendere possibile lo scambio di informazioni, diminuire il TOA e quindi anche l'energia impiegata per trasmettere un pacchetto.

Quando un gateway riceve un messaggio da un end-node e lo converte in un pacchetto IP, aggiunge anche le informazioni relative alla potenza del segnale. In base a quest'ultima, la rete può determinare quale sia la data rate ottima da assegnare al nodo. Se quest'ultimo ha abilitato l'ADR, allora viene eseguito un downlink verso il nodo per l'assegnamento della data rate.

# Capitolo 2

# 2. Architettura di riferimento

In questo elaborato, l'end-node utilizzato è basato sulla scheda di sviluppo della Microchip LoRa (TM) Technology Mote. Essa è responsabile di acquisire dati ambientali quali temperatura e luminosità relativa ed inviarli tramite protocollo LoRaWAN ai gateway nelle vicinanze.
Il gateway utilizzato è il Dragino LPS8, posto all'interno del Campus di Cesena, per assicurare la ricezione dei dati, in quanto in data odierna, l'area di Cesena non assicura una buona copertura a livello di reti LoRaWAN pubbliche. [5].

ChirpStack è un Network Server open-source che dispone di una interfaccia web per la gestione del gateway ed end-devices. Qui avviene la registrazione dei dispositivi, la gestione dell'Application, programmazione dei downlink e dettagli riguardo pacchetti ricevuti e inviati. Inoltre, è compreso il forwarding dei pacchetti ricevuti, tramite protocollo MQTT ad un server esterno, dove è possibile utilizzare grafici per mostrare la telemetria. I due precedenti server sono collocati presso il Centro di Ricerca sui Sistemi Elettronici ARCES dell'Università di Bologna. Facciamo riferimento alla Figura 2.1 per ulteriore chiarezza.

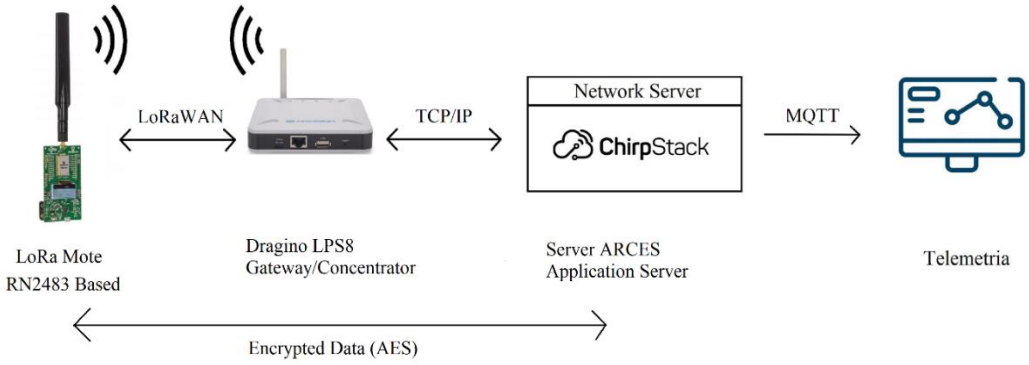



*Figura 2.1 Architettura del sistema*

Durante lo svolgimento della tesi, l'architettura rimarrà costante, fatta eccezione dell'end-node. Saranno fatte considerazioni a vari livelli di astrazione mirate al progetto di un nodo operante a micropotenza.

## 2.1 LoRa Mote

Il LoRa ® MoTe è una discovery board della Microchip configurata come dispositivo di Classe A, funzionante a 868 MHz, basato sul LoRa Modem RN2483.

Il Mote include un sensore di luminosità e temperatura per generare telemetria da trasmettere in tempi prefissati o tramite la pressione di un pulsante.

Il MoTe supporta un bridge USB-to-UART per il collegamento al computer e l'invio di comandi seriali per il modem RN2483. Si possono trovare ulteriori informazioni all'interno della guida Microchip [6]


*Figura 2.2 LoRa Mote e zoom su modem RN2483*

### 2.1.1 Descrizione Hardware

È possibile alimentare la discovery board tramite connettore USB o in alternativa due batterie AAA da 1.5 V. Entrambe le sorgenti di alimentazione vengono poi convertite internamente ad una tensione di 3.3 V (tensione di lavoro del modem RN2483).

Un PIC18LF45K50 si occupa della comunicazione USB-UART: infatti inoltra i comandi ricevuti dalla porta seriale al modem RN2483 e gestisce lo schermo LCD. Le impostazioni di default della porta UART sono quelle in Tabella 2.1 [7]

| *Specifica* | *Descrizione* |
|---|---|
| Baud Rate | 57600 bps |
| Packet Length | 8 bit |
| Parity bit | No |
| Stop bit | 1 bit |
| Hardware Flow Control | No |
| Carriage Return <CR> | Sì |
| Line feed <LF> | Sì |

*Tabella 2.1 Specifiche UART del LoRa Mote*

È presente su scheda il sensore di temperatura MCP9700 e quello di luminosità ALS-PT19-315C/L177/TR8.

Il sensore di temperatura prevede una accuratezza di ±2°C fino a +70°C e un'uscita analogica in tensione di 10.0 mV/°C. La sua tensione in uscita media a 0°C può essere ricavata dalla seguente figura: per semplicità verrà utilizzato 0.5V [8].

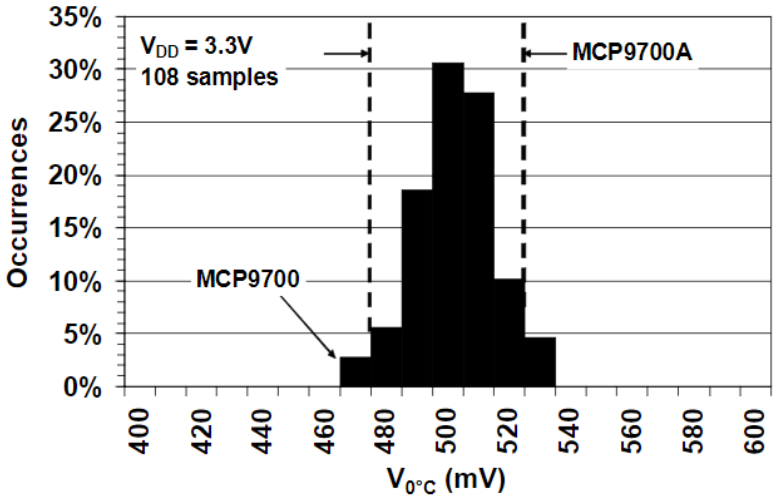



*Figura 2.3 Tensione di uscita a 0°C del sensore di temperatura*

Come si vede in Figura 2.4 ed illustrato in [9] il modem LoRa RN2483 contiene al suo interno un PIC18LF46K22, il transceiver LoRaWAN SX1276, uno switch RF e due cristalli.

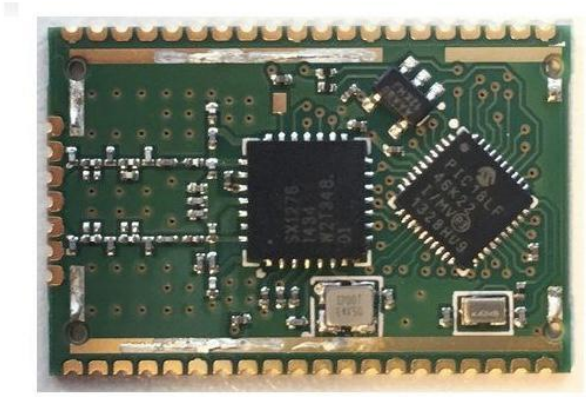


*Figura 2.4 Interno del modem RN2483 [9]*

In un primo momento il progetto consisterà nel testare il corretto funzionamento del sistema, quindi inviare i comandi da calcolatore tramite porta seriale per instaurare una comunicazione LoRaWAN e ricevere correttamente i pacchetti nel server ChirpStack.

In un secondo momento, verrà riprogrammato il PIC18LF46K22, interno all'RN2483, per renderlo autonomo dal resto della discovery board: invece che gestire i comandi seriali, verrà creato un semplice sketch che periodicamente acquisisce temperatura e luminosità, invia i pacchetti LoRaWAN ed entra in sleep fino al prossimo ciclo.

Questo permette di non dover utilizzare un secondo microcontrollore esterno (in questo caso era il PIC18LF45K50), per inviare i comandi seriali per la gestione del modem, riducendo sensibilmente i consumi e costi di progetto.

Inoltre, è eliminata la dipendenza dal modem RN2483: infatti è possibile creare un PCB che contenga il PIC18LF46K22 e il transceiver SX1276 e rendersi ulteriormente autonomi, mantenendo lo stesso firmware.

## 2.2 Gateway Dragino LPS8

È un open source LoRaWAN Gateway adatto ad uso interno che permette di fare da bridge tra la rete LoRa e IP supportando WiFi o Ethernet. Una volta connesso ad internet è possibile collegarsi tramite browser ad una interfaccia grafica dove si può impostare l'indirizzo IP del Network Server che si vuole utilizzare: in questo caso ChirpStack. Sempre dall'interfaccia grafica è possibile verificare che le connettività LoRa/LoRaWAN e WiFi siano presenti.

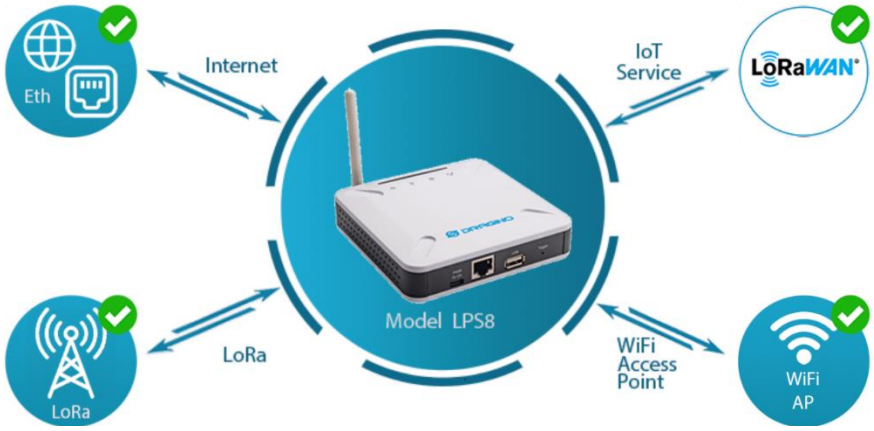



*Figura 2.5 Configurazione Dragino LPS8 alla porta 8000*

## 2.3 ChirpStack

All'interno del server ChirpStack ci sono tre importanti sezioni:

- Device Profile:
- Gateways
- Application

### 2.3.1 Gateways

All'interno della voce Gateways è possibile registrare i propri Gateway tramite il loro ID unico: in questo modo si ha accesso ai dettagli riguardanti le statistiche sui pacchetti ricevuti e trasmessi, sulle data rate dei pacchetti e tutti i frame ricevuti ed inviati[1].

Facciamo riferimento alle seguenti figure per alcuni esempi esplicativi:

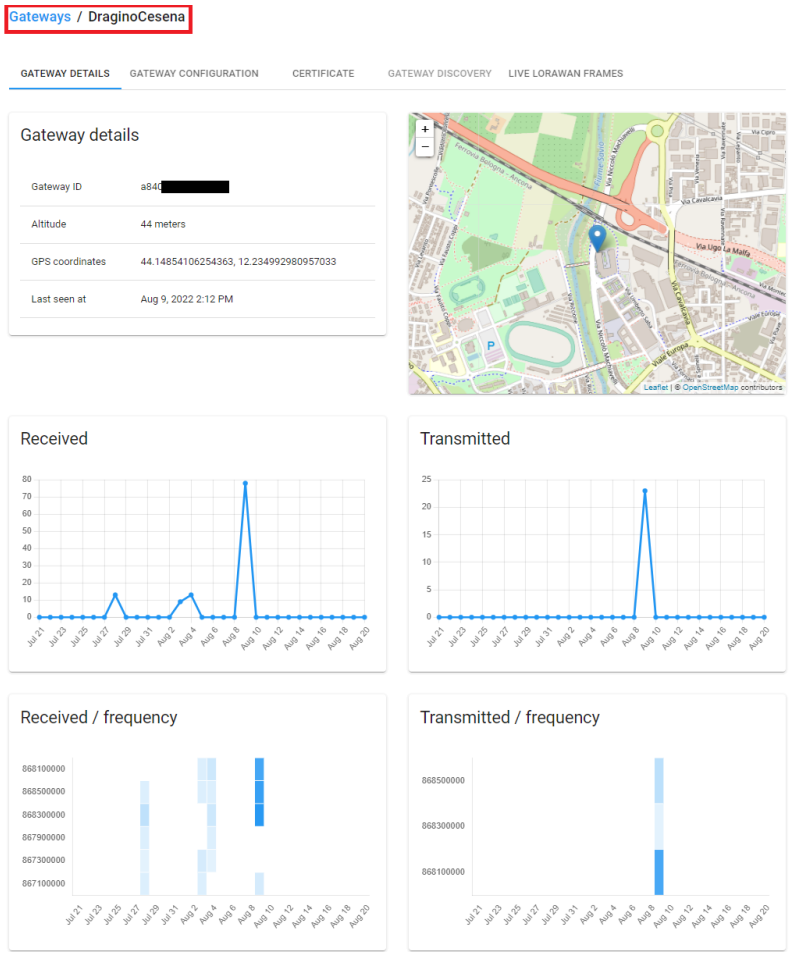



*Figura 2.6 Statistiche del Gateway di Cesena - ChirpStack*

[1] A livello del gateway, il payload dei frame ricevuti dagli end-node circostanti sono tutti criptati secondo la codifica AES. Solamente all'interno della propria Application, i frame sono decodificati.

Nella Figura 2.7 sono presenti le statistiche del pacchetto ricevuto dal Gateway LPS8: nella voce “frmPayload->bytes” è riportato il frame inviato dall’end-node, codificato secondo crittografia AES e formato in base64.

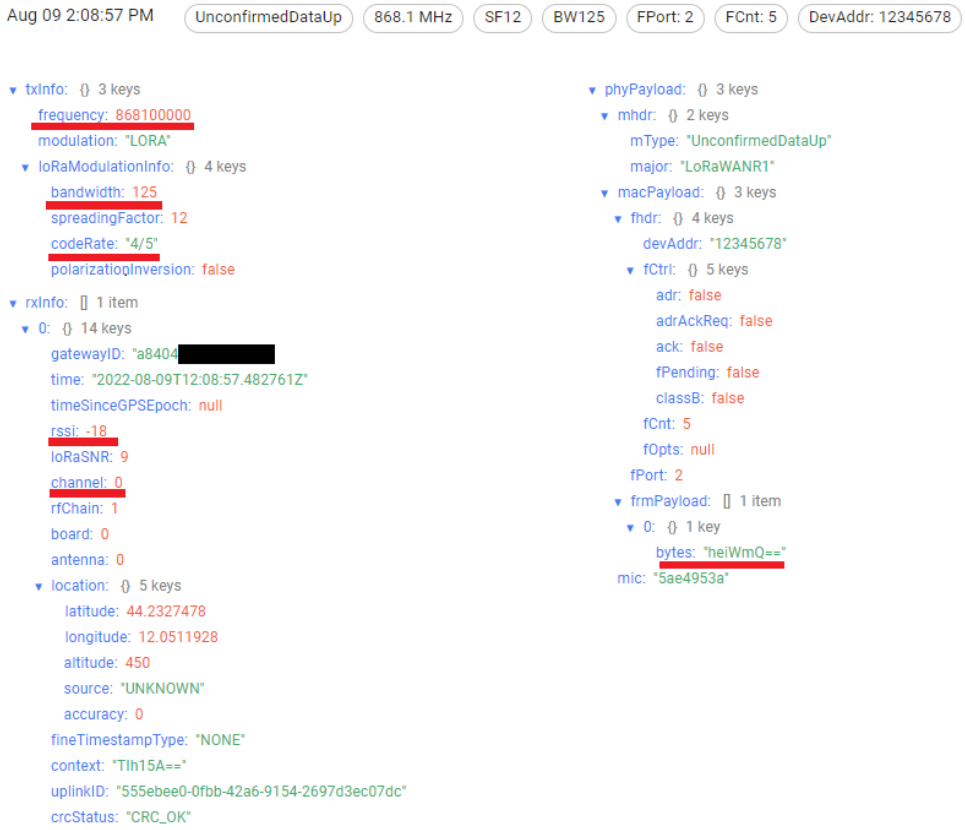



*Figura 2.7 Statistiche pacchetto ricevuto dal Gateway*

### 2.3.2 Application

In questa sezione è possibile creare la propria Application, nella quale si possono poi registrare diversi end-devices, ognuno caratterizzato da un diverso nome e metodo di attivazione (vedi paragrafo 1.2.2).

In questo elaborato è stata creata l’Application “LoRa Cesena_Microchip” e registrato la discovery board come “RN2483-LoRaWAN”. Una volta impostate le chiavi di attivazione secondo la procedura ABP, che ricordiamo dovranno essere le stesse programmate all’interno del LoRa Mote, tutti i pacchetti del Mote, ricevuti da un qualsiasi Gateway, verranno inoltrati nella nostra Application e il payload del frame sarà automaticamente decodificato.

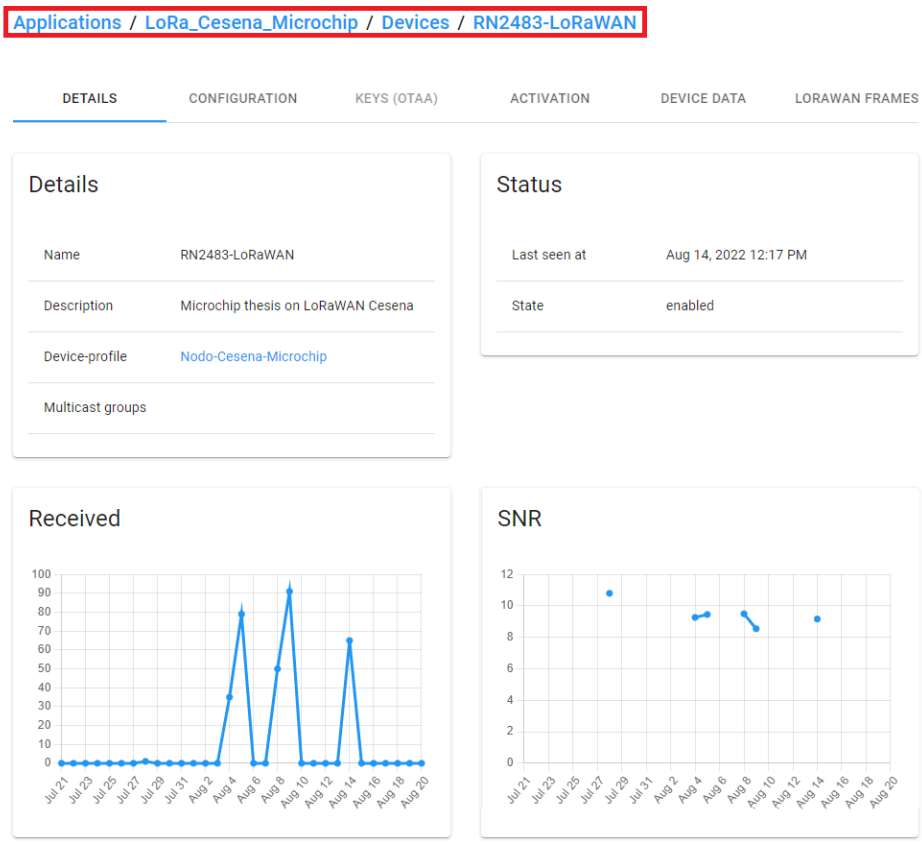



*Figura 2.8 Statistiche della Application RN2483 - Chirpstack*

Sempre all'interno di questa sezione, è possibile pianificare un downlink specificando la porta e i dati che si vogliono inviare all'end-device: essendo il LoRa Mote configurato come dispositivo di Classe A, il downlink sarà previsto non appena il Mote invierà un dato, all'interno di una delle due finestre di ricezione (Rx1 o Rx2).

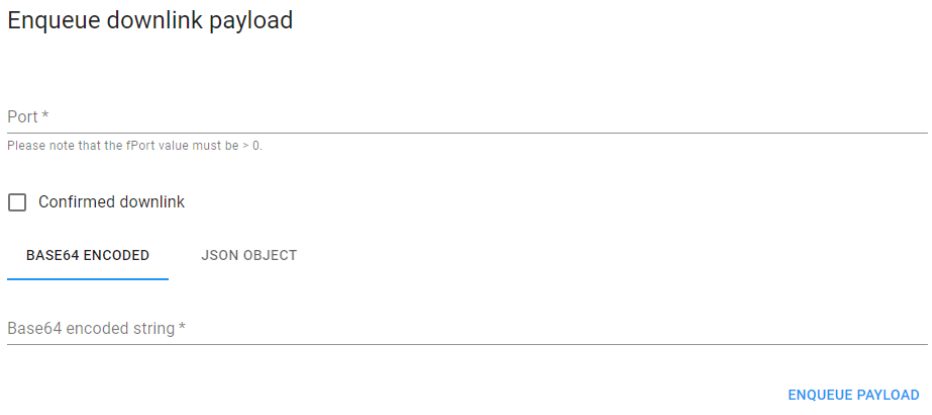



*Figura 2.9 Programmazione di un downlink - Chirpstack*

All'interno della sezione Application, nella voce "Device Data" è possibile leggere le statistiche dei pacchetti ricevuti/inviati dall'end-node. L'interfaccia è simile a quella vista in precedenza nella sezione del Gateway, tuttavia, il frame payload è decifrato e la telemetria è in chiaro.


*Figura 2.10 Statistiche pacchetto Application - ChirpStack*

All'interno del riquadro rosso, la voce "data", rappresenta il frame payload in base64. L'oggetto JSON contenente luminosità_percentuale e temperatura_celsius invece è stato ricavato dal payload e calcolato tramite una funzione che fa parte del Device-Profile.

### 2.3.3 Device-Profile

All'interno di questa sezione, nella voce "Codec", è possibile creare una funzione in javascript che si occupa della decodifica del payload di un end-device.

La funzione contiene un array di bytes, che rappresentano il frame payload dell'end-node, che per scelte progettuali è sempre fisso e riporta la seguente struttura.

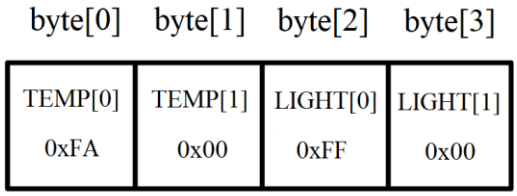



*Figura 2.11 Struttura del frame payload su ChirpStack*

Si ricorda che il PIC18LF46K22 che esegue la misura dei sensori, ha un ADC a 10 bit, mentre la dimensione delle celle della RAM è di 8 bit. Per questo motivo, i dati letti dal convertitore AD vengono salvati all'interno di una variabile di tipo *uint16_t* (intero senza segno a 16 bit) la quale occupa due celle di memoria.

Inoltre, i bytes sono ordinati secondo lo standard "Little Endian", ovvero la memorizzazione inizia dal byte meno significativo, finendo al più significativo.

Per quanto appena detto, la corretta decodifica di temperatura e luminosità riferita alla Figura 2.11 è la seguente:

TEMP = 0x00FA  LIGHT = 0x00FF;

È riportato in seguito il codice javascript che si occupa della conversione dei dati grezzi, ad un oggetto JSON

```
function Decode(fPort, bytes, variables) {

// From the LoRaWAN mote are sent 2 bytes for the temperature and 2 bytes for
the light sensor.
// Temperature sensor: MCP9700, 10 mV/°Celsius, Vout = 500 mV @ 0°C
// Light sensor:       ALS_PT19-315C/L177/TR8
out = {};

// Concateno due byte secondo lo standard Little Endian: il primo byte è il
meno significativo:
// Per esempio, b0 = 0xFA e b1 = 0x7C -> Il risultato è temp = 0x7CFA
var temp = (bytes[1]<<8 | bytes[0]);
var lum = (bytes[3]<<8 | bytes[2]);

// Passo da valore ADC a 10 bit (temp) a volt. La temperatura è data dalla
//formula ( Vtemp - 0.5 [V] ) * 100
var tempSensor = (temp*3.3/1023 -0.5)*100;

// Metodo toFixed() per arrotondare
out["temperatura_celsius"] = tempSensor.toFixed(2);
out["luminosità_percentuale"] = ((lum/1023)*100).toFixed(2);
out["bytes"] = bytes;
 return out;
}
```

# Capitolo 3

# 3. I comandi seriali Microchip

In questo capitolo è trattata la procedura per testare la comunicazione LoRaWAN all'interno della struttura citata nella parte precedente.

Prima di riprogrammare il modem RN2483 è necessario verificare che i dati vengano letti, inviati e ricevuti correttamente.

## 3.1 RN2483 LoRa Module Command Reference

All'interno del documento [10] sono riportati i comandi seriali che si possono eseguire collegando il LoRa Mote alla porta seriale del computer (o in un utilizzo del tutto generale, una qualsiasi periferica come un microcontrollore che dispone di porta UART). Le specifiche per il corretto funzionamento della porta seriale si trovano nel paragrafo 2.1.1.

Ci sono tre categorie generali di comando:

- <sys> System
- <mac> LoRaWAN Class A and Class C Protocols
- <radio> Transceiver commands

In questo elaborato saranno trattate solamente le prime due, in quanto i comandi di tipologia <radio> fanno uso dello strato fisico LoRa PHY e non integrano il protocollo LoRaWAN.

### 3.1.1 I comandi <sys>

Sono riportati nella

Tabella 3.1 i comandi <sys> più importanti, insieme ad una breve descrizione:

| *Parametro* | *Descrizione* |
|---|---|
| `reset` | Resetta e riavvia il modulo RN2483. |
| `factoryRESET` | Resetta il modulo RN2483 e riporta il contenuto della EEPROM ai valori di fabbrica. |
| `set` | Imposta la periferica selezionata ad un valore scelto |
| `get` | Riceve il valore di una certa periferica |

*Tabella 3.1 Comandi <sys> modem RN2483*

Esempio di utilizzo:

| *Comando* | *Possibile Risposta* | *Descrizione* |
|---|---|---|
| `sys set pindig GPIO10 1` | `ok` | Porta il pin collegato al led arancione a 3V3, accendendolo |
| `sys get pinana GPIO 12` | `235` | Esegue una lettura tramite l'ADC del sensore di temperatura e riporta il risultato in decimale |

### 3.1.2 I comandi <mac>

Ogni comando per il protocollo LoRaWAN inizia con la parola chiave mac e include una delle seguenti categorie:

| *Parametro* | *Descrizione* |
|---|---|
| `reset` | Resetta il modem RN2483 in una specifica banda di frequenza |
| `tx` | Trasmette una stringa di dati esadecimali in una specifica porta |
| `join` | Esegue l'attivazione alla rete LoRaWAN |
| `save` | Salva i parametri LoRaWAN Class A all'interno della EEPROM |
| `set` | Permette di impostare vari parametri MAC |
| `get` | Permette di leggere valori dei parametri MAC |

*Tabella 3.2 Comandi <mac> modem RN2483*

Il comando `tx` deve essere formulato nella maniera seguente:

```
mac tx <type> <portno> <data>

<type>              Può essere cnf o uncnf (rispettivamente confermato o non
                    confermato)

<portno>:           Numero decimale tra 1 e 223 che rappresenta il numero di porta al
                    quale inviare il pacchetto

<data>:             Valore esadecimale che rappresenta il payload


Esempio: Invia un messaggio non confermato tramite la porta 10 con un

payload "ABCD"

      mac tx uncnf 10 ABCD

      Possibile Risposta:

      ok

      mac_tx_ok
```

Esempio di utilizzo di comandi <mac>

| *Comando* | *Possibile Risposta* | *Descrizione* |
|---|---|---|
| `mac set devaddr 01234567` | `ok` | Imposta il device address a 4-byte esadecimale |
| `mac set appskey xxxxxxxx` | `ok` | Imposta il numero 16-byte esadecimale rappresentante l'application session key |
| `mac set nwkskey xxxxxxxx` | `ok` | Imposta il numero 16-byte esadecimale rappresentante la network session key |
| `mac save` | `ok` | Salva nella EEPROM le chiavi sovra citate |
| `mac join abp` | `ok accepted` | Metodo di attivazione dell'end-device |

Tramite questi comandi, è possibile configurare l'end-device e iniziare a trasmettere pacchetti ad un gateway nelle vicinanze.

## 3.2 Test del LoRa MoTe

Dopo aver collegato il LoRa Mote ed aver aperto una connessione seriale[2] ad esso, mostriamo il funzionamento del sistema digitando i seguenti comandi:

```
mac set devaddr 12345678^M^J
ok
mac set appskey dd3254f08d55d652e7730bbc263e0ada^M^J
ok
mac set nwkskey 75e0661aa0bc21e44b5721b0d630f7b2^M^J
ok
sys set pinmode GPIO13 ana^M^J
ok
sys set pinmode GPIO12 ana^M^J
ok
sys get pinana GPIO13^M^J
67
sys get pinana GPIO12^M^J
255
mac save^M^J
ok
mac join abp^M^J
ok
accepted
mac tx uncnf 2 FF004300^M^J
ok
mac_tx_ok
```

Come visto in precedenza, il comando `tx` richiede che il payload sia in formato esadecimale, per questo motivo, il risultato della conversione del GPIO13 (sensore di luminosità) e del GPIO12 (sensore di temperatura) è stato convertito di conseguenza. Inoltre, è stata replicata la struttura del pacchetto citata in Figura 2.11, ovvero secondo il Little Endian.

[2] È stato utilizzato il programma Putty, inizializzato con le impostazioni trovate nel paragrafo 2.1.1 La sintassi ^M^J rappresenta l'inserimento del carriage return <CR> e line feed <LF>

Possiamo verificare ora all'interno della sezione Application, del server ChirpStack, che il pacchetto è stato ricevuto e decodificato correttamente:


All'interno del riquadro rosso troviamo lo stesso Device Address che è stato impostato all'interno del MoTe, la porta numero 2, specificata nel momento della trasmissione ed infine il payload in base64, che una volta riportato in esadecimale è esattamente identico a quello scritto in precedenza:

$$(/wBDAA ==)_{Base64} = (FF004300)_{16}$$

Anche il CODEC funziona come desiderato e crea un oggetto JSON contenente temperatura e luminosità percentuale.

Ora che il sistema funziona come aspettato e i dati vengono ricevuti, possiamo passare alla riprogrammazione del PIC interno all'RN2483 e realizzare un nodo più efficiente energeticamente.

# Capitolo 4

# 4. Firmware modem RN2483

La casa produttrice Microchip, mette a disposizione una libreria LoRaWAN compatibile con il microcontrollore utilizzato all'interno del modem RN2483. In particolare, il documento [11] descrive come creare i file necessari al corretto funzionamento del programma, partendo da MPLAB Code Configurator (MCC), un'interfaccia grafica per velocizzare la scrittura del software. Nel paragrafo 4.5 si trova un elenco del software, libreria e licenza utilizzati in questo progetto.

Facciamo riferimento alla seguente figura per una descrizione ad alto livello della software application:

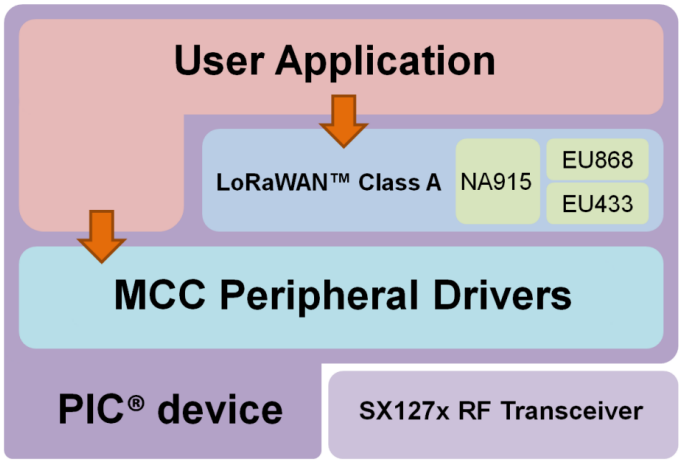



*Figura 4.1 Software Application High Level Design [11]*

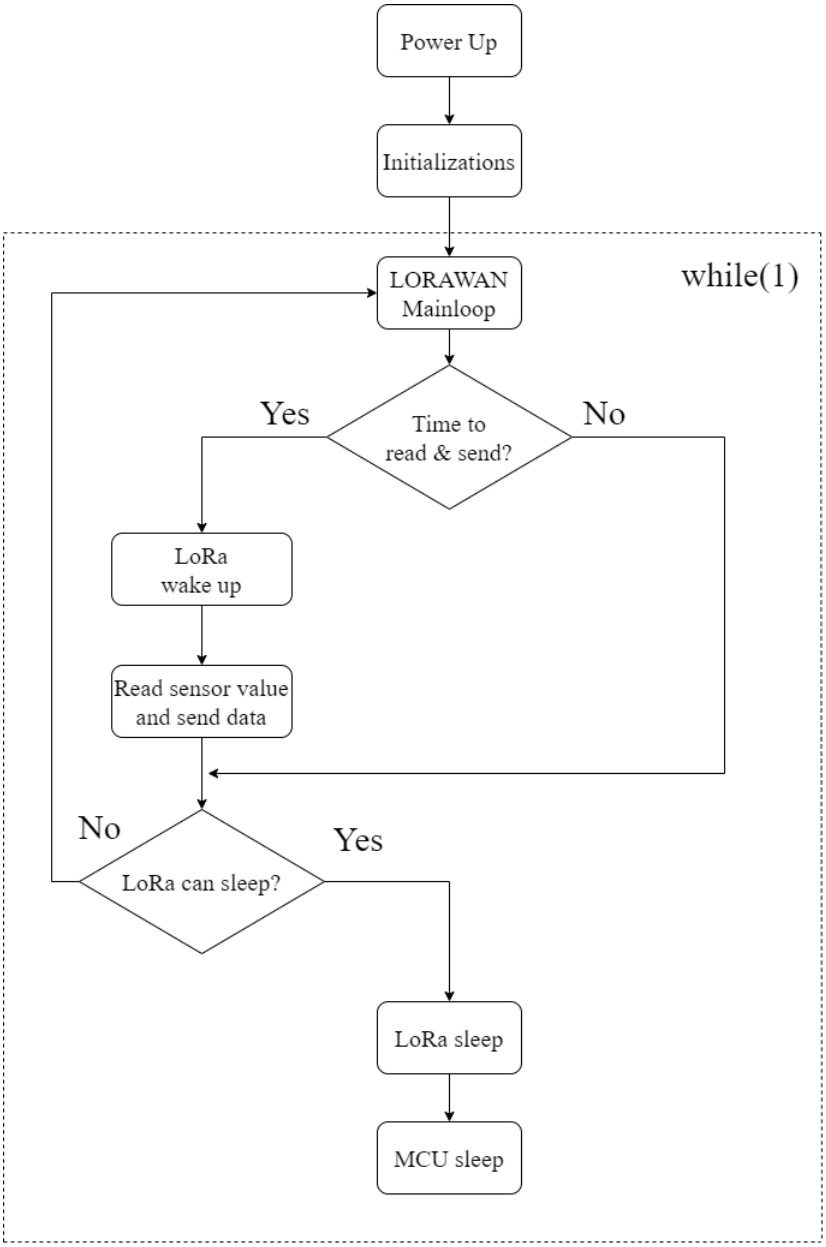
L'applicazione comprende i seguenti strati:

1. **User Application**, gestisce le interfacce per leggere i sensori e inviare i dati al server.
2. **LoRaWAN Class A Layer**, generato da MCC secondo le specifiche dell'utente usando la libreria visiva MCC LoRaWAN Library Plug-In.
3. **MCC Peripheral Drivers**, generati automaticamente da MCC secondo le esigenze dell'utente (interrupt, timer, ADC, ecc).
4. **Hardware Layer**: un dispositivo a 8-bit PIC e il radio transceiver Semtech.

## 4.1 Applicazione ad alto livello

Dal momento in cui l'applicazione lavora all'interno di un end-device alimentato a batteria, lo scopo principale sarà rimanere in uno stato di sleep a basso consumo di energia, svegliarsi una volta all'ora, leggere la telemetria, inviarla al server e tornare nello stato di sleep.

Viene ora mostrato il flow-chart relativo all'applicazione appena descritta.

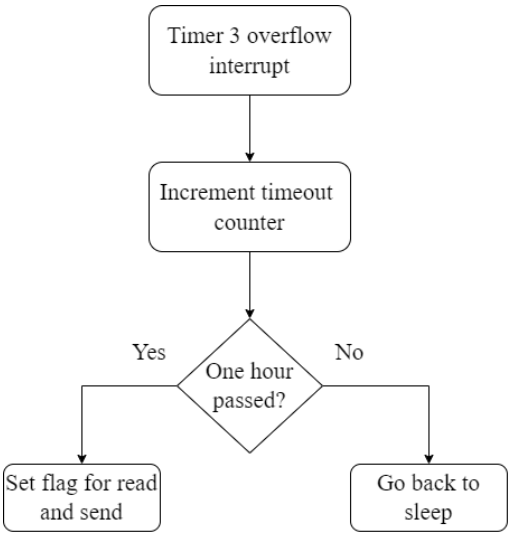


All'interno della fase di inizializzazione, sono abilitati gli interrupt, timer e viene inizializzato lo stack LoRaWAN (impostate le chiavi necessarie alla comunicazione ed eseguita la procedura di join alla rete LoRaWAN).

All'interno del ciclo while, è necessario che venga eseguita la funzione *LoRaWAN Mainloop()*, la quale si occupa della gestione dello stack LoRaWAN per poi restituire il controllo. In seguito, il sistema controlla se sia passata un'ora; in caso affermativo legge i dati dai sensori, inizializza il transceiver e invia un pacchetto al server.

Prima di tornare in stato di sleep è necessario controllare che lo stato del transceiver LoRaWAN sia in "IDLE" e non stia né trasmettendo né ricevendo.

L'unico modo per tenere traccia di quanto tempo sia passato è utilizzare un Timer. Lo stack LoRaWAN utilizza già il Timer 1, quindi è stato scelto il Timer 3 in quanto permette di raggiungere un periodo massimo di 16 secondi (grazie ad un oscillatore esterno a 32.768 kHz all'interno del modulo RN2483). Incrementando una variabile ad ogni interrupt del Timer 3 è possibile contare un periodo di un'ora dopo 225 overflows. Nel seguente flow-chart è approfondita la fase "*Time to Read & Send?*".




La fase di wake-up dallo stato di sleep consuma energia. Pertanto, sarebbe preferibile svegliare il microcontrollore solamente quando si intende trasmettere un pacchetto. I timer del PIC sono solitamente limitati a qualche secondo ed accorerebbero risvegli multipli. Per evitare questo problema può quindi convenire in fase di progetto includere un timer esterno a bassissimo consumo di potenza, per svegliare un'unica volta il microcontrollore. Più avanti in questo documento verrà approfondita la metodologia appena descritta.

## 4.2 Applicazione a basso livello

Viene ora allegato il codice relativo alla fase di inizializzazione.

```
// Funzione creata da MCC per inizializzare Timer 1, oscillatori, SPI
SYSTEM_Initialize();
// Abilitazione Interrupt
INTERRUPT_GlobalInterruptEnable();
INTERRUPT_PeripheralInterruptEnable();
// La funzione "handle16sInterrupt" è impostata come callback
all'overflow del Timer 3
TMR3_SetInterruptHandler(handle16sInterrupt);
// Spengo tutte le periferiche che il sistema non utilizza
SysConfigSleep();
// Inizializzo ADC
ADC_Init();
// Configuro i GPIO pins come ingresso/uscita analogico/digitale
IO_pins_init();
// Il led arancione sarà spento in seguito quando l'end-node joinerà
// il server
LED_ORANGE = 1;
// Gestione stack LoRaWAN
LORAWAN_Init(RxDataDone, RxJoinResponse);
LORAWAN_SetNetworkSessionKey(nwkSKey);
LORAWAN_SetApplicationSessionKey(appSKey);
LORAWAN_SetDeviceAddress(devAddr);
LORAWAN_Join(ABP);
// Variabile booleana per l'invio dei dati
TimeToSend = 1;
```

In seguito, verranno approfondite le funzioni appartenenti alla libreria LoRaWAN di Microchip. Viene allegato ora il codice all'interno del while loop.

```
while (1){
      // Has to be called once per loop (runs system timers and check
      // DIO pins)
      LORAWAN_Mainloop();

         if(TimeToSend){    // a hour has passed
             LoRaWakeUp();
             readAndSend();
             TimeToSend = 0;
         }

      // it's not transmitting nor receiving
      if(LORAWAN_GetState() == IDLE){
            LoRaSleep();
            SLEEP();
       }
    } //end of while(1)
} // end of main()

void readAndSend(void){
    // payload is a uint16_t array
    payload[0] = ADC_Read(TEMP);
    payload[1] = ADC_Read(LIGHT);
    LORAWAN_Send(UNCNF, portNumber, &payload, sizeof(payload));
}
```

Le funzioni *LoRaWakeUp()* e *LoRaSleep()* si occupano di abilitare e disabilitare il modulo MSSP, il quale gestisce il protocollo SPI per la comunicazione tra il PIC e il transceiver Semtech.

Viene in fine approfondito il codice relativo alla funzione *handle16sInterrupt()*, responsabile di svegliare il PIC dallo stato di sleep ad ogni overflow del Timer 3.

```
void handle16sInterrupt() {
    static volatile uint8_t counterSleepTimeout = 0;

    // ONE_HOUR_TIMEOUT_COUNTS is set to 225 (3600s in a hour/ 16s =
    // 225)
    if( ++counterSleepTimeout == ONE_HOUR_TIMEOUT_COUNTS )
    {
        // a hour passed
        TimeToSend = 1;
        counterSleepTimeout = 0;
    }
    else
    {
        SLEEP();
    }
}
```

## 4.3 Gestione dello Stack LoRaWAN

All'interno del documento [12] si trovano le funzioni della libreria LoRaWAN di Microchip ed è inoltre descritto come impostare tramite la GUI di MCC i moduli hardware necessari allo Stack LoRaWAN.

Tra i moduli hardware che devono essere configurati si trova il modulo MSSP2 per la comunicazione SPI tra PIC e transceiver SX1276, il Timer1 e sei pin GPIO.

Per il protocollo LoRaWAN vengono invece scelte la banda ISM, nel nostro caso *Europe 868*, la classe dell'end-node e i canali nei quali si vuole trasmettere.

È possibile raggruppare per vari livelli i file generati da MCC: facendo riferimento alla Figura 4.1 Software Application High Level Design

1. User application source files

| |
|---|
| main.c |

2. LoRaWAN Stack

**Header Files**
interrupt_manager_lora_addons.h
lorawan.h
lorawan_aes.h
lorawan_aes_cmac.h
lorawan_defs.h
lorawan_init.h
lorawan_na.h
lorawan_eu.h
lorawan_private.h
lorawan_radio.h
AES.h
AESdef.h

**Source Files**
AES.c
interrupt_manager_lora_addons.c
lorawan.c
lorawan_aes.c
lorawan_aes_cmac.c
lorawan_init.c
lorawan_na.c
lorawan_eu.c

**Radio Driver header files**
radio_driver_SX1276.h
radio_driver_SX1272.h
radio_interface.h
radio_registers_SX1276.h
radio_registers_SX1272.h

**Radio driver source files**
radio_driver_SX1276.h
radio_driver_SX1272.h

**SW Timer System header files and Source files**
sw_timer.h
tmr_lora_addons.h
sw_timer.c
tmr_lora_addons.c

3. MCC Pheripheral Drivers (HAL files: Hardware Abstraction Layer)

**HAL header files**
mcc_lora_config.h
tmr_lora_addons.h
pin_manager_lora_addons.h
radio_driver_hal.h
spi2.h
tmr1.h

**HAL source files**
radio_driver_hal.c
spi2.c
tmr1.c

## 4.4 Funzioni Stack LoRaWAN

### 4.4.1 LORAWAN_Init

| | |
|---|---|
| Nome | `LORAWAN_Init` |
| Prototipo | `LORAWAN_Init(RxAppDataCb_t RxPayload, RxJoinResponseCb_t RxJoinResponse)` |
| Descrizione | Inizializza lo stack LoRaWAN e il modulo sx1276 |
| Parametri | • `RxPayload` – puntatore a funzione che viene chiamata alla fine della comunicazione bidirezionale (tx – rx1 – rx2)<br>• `RxJoinResponse` – puntatore a funzione che viene chiamata dopo la procedura di attivazione |
| Esempio | `LORAWAN_Init(RxDataDone, RxJoinResponse)` |

La funzione `RxPayload` permette di recuperare il payload di un eventuale downlink che al fine di ridurre i consumi può intervenire sul modificare la potenza di trasmissione oppure ogni quanto tempo il PIC deve inviare un pacchetto.

La funzione `RxJoinResponse` invece può essere utilizzata per controllare che l'attivazione sia avvenuta con successo.

Viene riportato come esempio parte del codice preso dal `main.c`

```
// Must match with the port from chirpstack
#define DOWNLINK_PORT   30
void RxDataDone(uint8_t* pData, uint8_t dataLength, OpStatus_t
status);
void RxJoinResponse(bool status);
LORAWAN_Init(RxDataDone, RxJoinResponse);

void RxDataDone(uint8_t* pData, uint8_t dataLength, OpStatus_t status)
{
    // Any received data is stored in a buffer pointed by *pData with
    // a lenght of dataLength bytes
    // pData[0] is the fport used from the gateway in downlink

    if(pData[0]==DOWNLINK_PORT){
        LORAWAN_SetCurrentDataRate(pData[1]);
        LED_ORANGE = pData[2];
        LED_GREEN = pData[3];
        ONE_HOUR_TIMEOUT_COUNTS = pData[4];
    }
}

void RxJoinResponse(bool status)
{
    //When network is joined orange led is turned off
    LED_ORANGE = 0;
}
```

### 4.4.2 LORAWAN_SetNetworkSessionKey

| Nome | `LORAWAN_SetNetworkSessionKey` |
|---|---|
| Descrizione | Imposta la Network Session Key: deve essere la stessa impostata nel server Chirpstack |
| Esempio | `uint8_t nwkSKey[16] = {0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C};`<br>`LORAWAN_SetNetworkSessionKey(nwkSKey);` |

### 4.4.3 LORAWAN_SetApplicationSessionKey

| | |
|---|---|
| Nome | `LORAWAN_SetApplicationSessionKey` |
| Descrizione | Imposta l'Application Session Key: deve essere la stessa impostata nel server Chirpstack |
| Esempio | `uint8_t mcastAppSKey[16] = {0x3C, 0x8F, 0x26, 0x27, 0x39, 0xBF, 0xE3, 0xB7, 0xBC, 0x08, 0x26, 0x99, 0x1A, 0xD0,0x50, 0x4D};`<br>`LORAWAN_SetApplicationSessionKey(mcastAppSKey);` |

### 4.4.4 LORAWAN_SetDeviceAddress

| | |
|---|---|
| Nome | `LORAWAN_SetDeviceAddress` |
| Descrizione | Imposta il device Address a 32 bit: deve essere lo stesso impostato nel server Chirpstack |
| Esempio | `uint32_t devAddr = 0x12345678;`<br>`LORAWAN_SetDeviceAddress(devAddr);` |

### 4.4.5 LORAWAN_Join

| | |
|---|---|
| Nome | `LORAWAN_Join` |
| Descrizione | Questa funzione comincia la procedura LoRaWAN di attivazione, la quale finisce con la funzione di callback `RxJoinResponse`. Può avere come argomento `OTAA` o `ABP` |
| Esempio | `LORAWAN_Join(ABP);` |

### 4.4.6 LORAWAN_Mainloop

| | |
|---|---|
| Nome | `LORAWAN_Mainloop` |
| Descrizione | Questa funzione è utilizzata per far funzionare il Timer 1 e controllare periodicamente i pin DIO del transceiver SX1276. Deve essere chiamata una volta per loop |
| Esempio | `LORAWAN_Mainloop();` |

### 4.4.7 LORAWAN_Send

| Nome | `LORAWAN_Send` |
|---|---|
| Prototipo | `LorawanError_t LORAWAN_Send (TransmissionType_t confirmed, uint8_t port, void *buffer, uint8_t bufferLength)` |
| Descrizione | Questa funziona comincia un processo di comunicazione bilaterale. La procedura è conclusa quando viene chiamata la funzione di callback `RxPayload`. |
| Parametri | • `confirmed` – rappresenta il tipo di trasmissione, può essere `UNCNF` o `CNF`<br>• `port` – rappresenta il numero di porta per la trasmissione, è un numero compreso tra 0 e 255<br>• `buffer` – un data buffer per memorizzare i dati da inviare<br>• `bufferLength` – la lunghezza in bytes del data buffer |
| Esempio | `LORAWAN_Send (UNCNF, 4, "Hello Word", 12)` |

Analizzando il file sorgente della libreria LoRaWAN è possibile vedere che questa funzione esegue una serie di controlli per assicurarsi che sia possibile trasmettere un pacchetto (che il duty cycle massimo sia rispettato, che sia già stata eseguita la procedura di join, che il numero di porta sia coerente, che la dimensione del payload sia minore della massima consentita, che il transceiver sia in stato di IDLE). In seguito, richiama la funzione `AssemblePacket` per criptare secondo la codifica AES il payload e aggiungere i frame dell'overhead (frame counter, MIC, ack, ecc..).

Dopodiché la funzione `RADIO_Transmit` riceve il pacchetto assemblato e lo mette in coda per la trasmissione, cambiando lo stato dello stack in `TRANSMISSION_OCCURRING`.

### 4.4.8 LORAWAN_GetState

| Nome | `LORAWAN_GetState` |
|---|---|
| Descrizione | Questa funzione restituisce lo stato dello Stack LoRaWAN. I possibili stati appartenenti alla classe A e C sono i seguenti:<br>• IDLE<br>• TRANSMISSION_OCCURRING<br>• BEFORE_RX1<br>• RX1_OPEN<br>• BETWEEN_RX1_RX2<br>• RX2_OPEN<br>• RETRANSMISSION_DELAY<br>• ABP_DELAY<br>• CLASS_C_RX2_1_OPEN<br>• CLASS_C_RX2_2_OPEN |
| Esempio | `uint8_t state;`<br>`state = LORAWAN_GetState(void);` |

Questa funzione viene utilizzata per mandare in sleep in microcontrollore. Infatti, è necessario che lo stack LoRaWAN non sia in IDLE prima di spegnere le periferiche hardware ed entrare nello stato di basso consumo energetico.

## 4.5 Programmi e versioni utilizzate

I programmi utilizzati in questo progetto sono stati i seguenti:

- MPLAB X IDE v3.40

  Ambiente di sviluppo del firmware. Il progetto è basato sul PIC18LF46K22 utilizzando il compilatore XC8 PRO (v1.38). Altri compilatori e/o microcontrollori potrebbero essere incompatibili con la libreria LoRaWAN di Microchip.

- MCC v3.35

  Microchip Code Configurator è un plug-in del software MPLAB X IDE ed è utilizzato per configurare le impostazioni della libreria LoRaWAN tramite un'interfaccia grafica. È necessario utilizzare la versione specificata in quanto è l'unica compatibile con la libreria *LoRaWAN Library Plug-in v01.10.00 beta* fornita da Microchip.

- MPLAB X IPE v3.40

  Programma utilizzato per la programmazione del firmware all'interno del microcontrollore. È necessario utilizzare la modalità *Low Voltage Program* accessibile nella sezione *Power* per evitare di danneggiare il modem RN2483.

# Capitolo 5

# 5.Progettazione end-node a micropotenza

Lo scopo principale di un end-node è di trasmettere ad intervalli di tempo prestabiliti dei dati di una qualsiasi natura, che siano ambientali, riguardanti lo stato di un pulsante, coordinate spaziali, ecc.

Dal momento in cui il tempo di lettura dei sensori e trasmissione del segnale è molto minore rispetto al tempo nel quale l'end-node si trova in stato di sleep, è necessario che il consumo del dispositivo sia estremamente ridotto. In questa maniera, l'end-node alimentato a batteria, è in grado di resistere per molti anni, fino al tempo di deterioramento della batteria stessa.

## 5.1 Livelli di astrazione

In questo capitolo verranno quindi analizzate varie scelte progettuali attraverso diversi livelli di astrazione per raggiungere consumi nell'ordine delle micropotenze. Facciamo riferimento alla Figura 5.1.

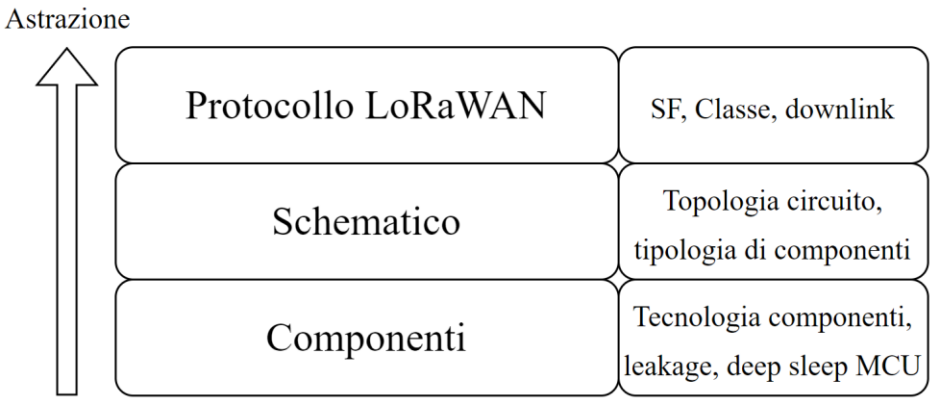



*Figura 5.1 Livelli di astrazione, progettazione end-node*

### 5.1.1 Layer Protocollo LoRaWAN

In cima al livello di astrazione abbiamo le scelte riguardanti il protocollo LoRaWAN. Da quanto detto nel paragrafo 1.1.4 è necessario scegliere lo spreading factor più basso che permetta la ricezione dei dati. Questo parametro è fortemente influenzato dalla distanza dal gateway e dall'ambiente che circonda l'end-node. Si ricorda che un minore SF riduce sia il tempo di trasmissione che il dispendio energetico: è possibile utilizzare quindi l'algoritmo Adaptive Data Rate per trovare il

valore opportuno oppure eseguire dei test partendo dal valore massimo scendendo fino a quando non si ricevono più i dati.

Riportiamo nella Tabella 5.1 i dati indicativi riguardanti una modulazione LoRa con potenza di trasmissione di 14 dBm (la massima consentita in Europa nelle bande sotto riportate).

| *Data Rate* | *Configurazione* | *TOA indicative [ms]* | *Link Budget [dB]* |
|---|---|---|---|
| 0 | SF12 – 125 kHz | 991 | 151 |
| 1 | SF11 – 125 kHz | 557 | 148.5 |
| 2 | SF10 – 125 kHz | 288 | 146 |
| 3 | SF9 – 125 kHz | 144 | 143 |
| 4 | SF8 – 125 kHz | 72 | 140 |
| 5 | SF7 – 125 kHz | 41 | 137 |

*Tabella 5.1 Legame tra Data Rate, TOA e Link Budget*

La corrente che necessita il transceiver è legata alla potenza di trasmissione e in tutti i casi della tabella riportata è di 44 mA [13].

Per fare una stima del tempo di vita di un end-node prendiamo come riferimento una batteria AAA da 1000 mAh di capacità. La durata in anni è data dalla seguente formula:

$$T_{years} = \frac{c}{i_{avg}} \cdot \frac{1}{24 \cdot 365}$$

Dove:

$c$ = capacità batteria in mAh
$i_{avg}$= corrente media consumata dall'end-node in mA

La corrente media si può stimare una volta che si conoscono la corrente di sleep $i_s$, quella di trasmissione $i_{tx}$ e quella di consumo nominale del PIC $i_{idle}$.

$$i_{avg} = \frac{i_s \cdot t_s + i_{tx} \cdot t_{tx} + i_{idle} \cdot t_{idle}}{t_s + t_{tx} + t_{idle}}$$

Dove:

$t_s$ = tempo di sleep
$t_{tx}$ = tempo di trasmissione, si può ricavare dalla Tabella 5.1
$t_{idle}$ = tempo di wake-up del microcontrollore, lettura sensori, ecc.

Un'importante scelta progettuale che impatta sulla durata della batteria è con quale frequenza inviare i pacchetti. Avendo preso in considerazione

le formula sopra citate è possibile fare una stima, facciamo quindi riferimento alla Figura 5.2.

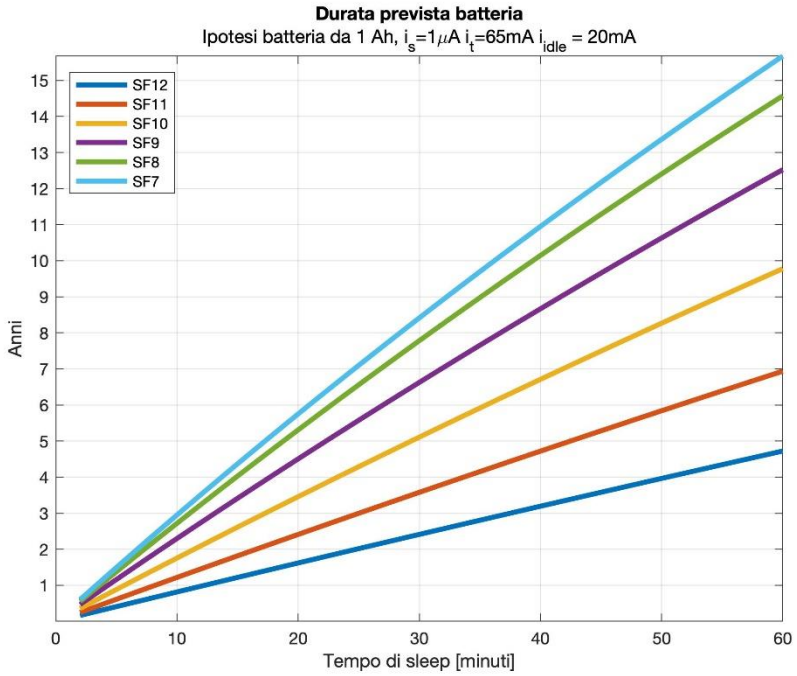



*Figura 5.2 Durata batteria vs tempo di sleep*

È ora evidente quanto lo SF incida sulla durata complessiva della batteria. Nel caso l'end-node sia posto a una distanza tale che sia necessario utilizzare valori alti di SF è consigliato ridurre la frequenza di invio dei pacchetti: in fase di progetto può essere una buona norma permettere la riprogrammazione sia SF che della frequenza di trasmissione dei pacchetti tramite un messaggio in downlink.

Questo elaborato mira a raggiungere una corrente di sleep nell'ordine dei nA, quindi il valore utilizzato nel grafico $i_s = 1\mu A$ è da intendersi come ampiamente peggiorativo.

Può essere ora utile, in sede di progetto, considerare quanto la corrente di sleep incida sulla vita del nodo. Dato che la fase di trasmissione è quella più energivora, si può infatti immaginare che un dispositivo che trasmette molto frequentemente dei pacchetti, abbia un tempo di vita relativamente breve sia nel caso in cui la corrente di sleep sia nell'ordine dei micro-Ampere, che nano-Ampere.

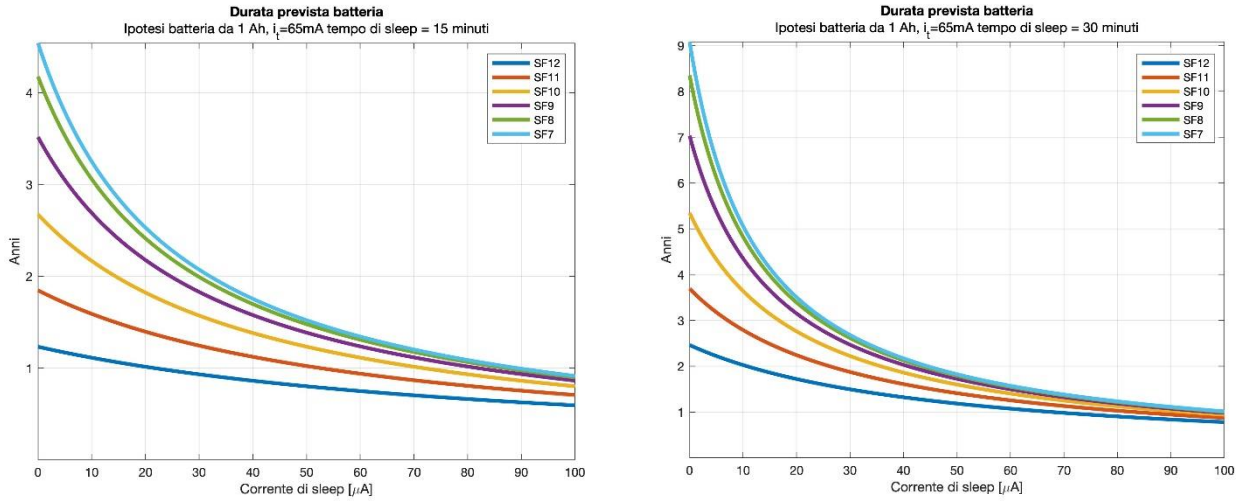


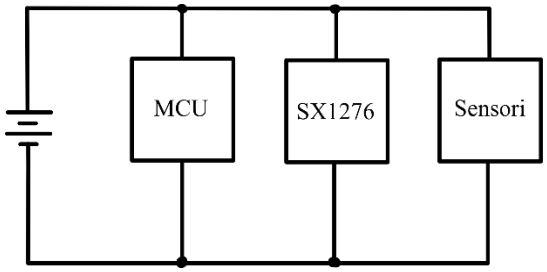

*Figura 5.3 Durata batteria vs corrente di sleep*

Questi due grafici mostrano la durata di batteria in funzione della corrente di sleep nell'ipotesi di trasmettere pacchetti ogni 15 o 30 minuti. Notiamo che per consumi maggiori di circa 80-90 µA la durata quasi non dipende dallo SF. Notiamo invece che per correnti minori a 1µA il parametro SF incide notevolmente.

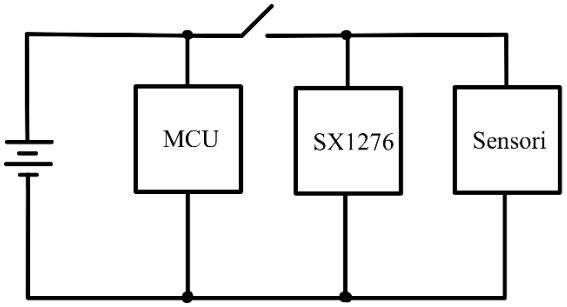
### 5.1.2 Layer Schematico

Scendendo di un livello di astrazione si decide come collegare i vari componenti tra di loro, in quanto ogni topologia di circuito riporta vantaggi o svantaggi in termini di consumi.

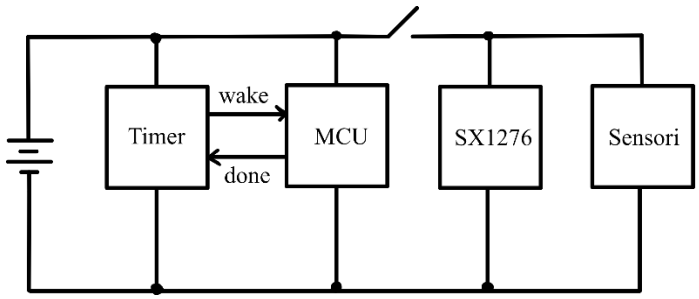
Questo primo schema è il più semplice in termini di hardware; tuttavia, riporta una corrente di sleep $i_s$ alta in quanto sia il transceiver che i sensori rimangono alimentati durante il loro non utilizzo.




Si può pensare di introdurre uno switch per scollegare elettricamente il transceiver e i sensori quando essi non devono essere utilizzati. Facendo così la corrente di sleep è dettata solo dal consumo del microcontrollore.




Il microcontrollore rimane in stato di sleep fino a che uno dei suoi timer interni, arrivando all'overflow provoca il suo risveglio. Generalmente i timer interni alla MCU possono arrivare a conteggiare periodi di pochi secondi, mentre per quanto detto poco prima, noi vorremmo rimanere in sleep per qualche minuto o ora. Inoltre, i timer interni hanno consumi più elevati rispetto al target del progetto. Per ovviare questo problema possiamo inserire un timer esterno che riesca a gestire conteggi superiori.




Grazie alla topologia di schematico appena presentata, la corrente di sleep è data solo dal consumo del Timer esterno, dall'MCU e dal leakage dello switch. Questo schema risulta particolarmente vantaggioso in quanto l'MCU è dotato di modalità di sleep avanzata con consumi dell'ordine dei 10 nA.

Per motivi di flessibilità è possibile impostare a livello hardware un periodo di sleep del Timer di 15 minuti, ad esempio, e poi a livello software decidere ogni quanti risvegli trasmettere i pacchetti. Così facendo è possibile raggiungere i range di vita della batteria presentati nel grafico in Figura 5.2.

### 5.1.3 Layer Componenti

Come ultimo livello di astrazione c'è la scelta del singolo componente, in quanto le sue caratteristiche e tecnologia influiscono sulla corrente di sleep o di leakage del componente stesso.

Per la scelta della batteria sono state valutate le seguenti tipologie:

| | Ioni di litio | AAA x2 in serie | Batteria a bottone |
|---|---|---|---|
| Tensione di lavoro [V] | 4.2 ÷ 2.9 | 3 ÷ 1.6 | 3 ÷ 2 |
| Resistenza interna [Ω] | 0.3 ÷ 0.4 | 0.3 ÷ 0.6 | 3 ÷ 9 |
| Capacità [mAh] | 250 ÷ 1500 | 800 ÷ 1200 | 200 ÷ 500 |

La batteria a ioni di litio è stata scartata in quanto avrebbe richiesto un regolatore DC-DC per abbassare la tensione, mentre quella a bottone a causa della resistenza interna troppo alta, che causerebbe durante i picchi di corrente in trasmissione delle cadute di tensione non accettabili, che rischierebbero di mandare in reset il PIC o SX1276 oltre a degradare la batteria stessa. Si è deciso quindi di utilizzare due batterie AAA in serie senza regolatore di tensione. (Sia il microcontrollore che transceiver sono in grado di lavorare fino a 1.8V).

Il microcontrollore scelto è il PIC18LF46K22, con una tensione di lavoro da 5 V fino a 1.2V e una corrente di sleep di 20 nA.

Nel Layer Schematico si è parlato di switch. Quest'ultimo può essere implementato tramite un transistor in modo che interrompa l'alimentazione positiva o quella negativa. La scelta di quest'ultimo non è banale in quanto bisogna assicurarsi che la corrente di leakage all'interno del range di temperatura del dispositivo sia accettabile (inferiore al $\mu A$), che la $R_{DS}$ sia abbastanza piccola per evitare che in fase di trasmissione, dei picchi di corrente portino a una caduta di potenziale sullo switch che possa mandare in reset il microcontrollore o disturbare il transceiver.

Si è quindi scelto di utilizzare il SIP32431, un ultra-low leakage load switch integrato, che presenta una corrente di leakage di 10pA @ 3.3V e una resistenza $R_{DS} = 0.135\,\Omega$ @ 3V. Per ulteriori informazioni consultare il documento [14].

Una tipica applicazione è la seguente. Il pin di ON/OFF è pilotato dal PIC quando desidera leggere i sensori o utilizzare il transceiver.

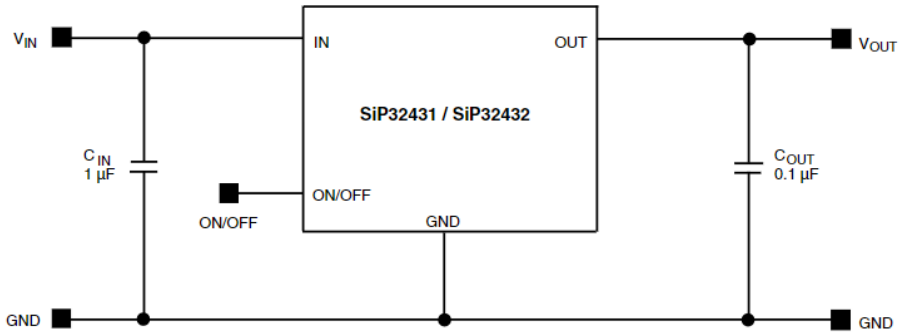



*Figura 5.4 Collegamento tipico SIP32431 [14]*

Come regola generale per i condensatori con valore superiore al µF è di utilizzare un dielettrico C0G laddove disponibile, in quanto hanno le prestazioni migliori in termini di corrente di leakage. Essi sono seguiti da quelli con dielettrico X7R e X5R.

Per la scelta del Timer è stato impiegato il TPL5010, appartenente alla categoria ultra-low power (35nA @ 2.5V) e con funzione di watchdog incorporata.

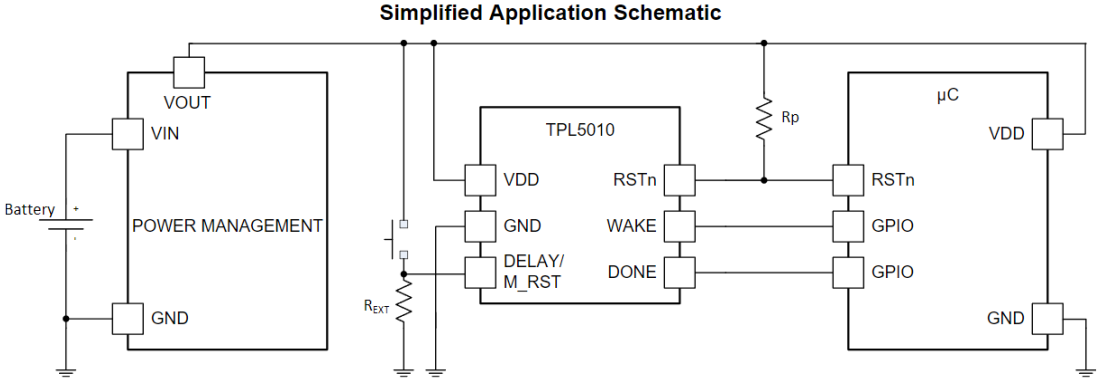



*Figura 5.5 Collegamento tipico TPL5010 [15]*

Per impostare il periodo di sleep $t_{ip}$ basta collegare una resistenza al pin DELAY/M_RST del valore opportuno (visitare il documento [15]). Allo scadere del conteggio il pin WAKE, collegato ad un pin di interrupt del PIC si porta ad un valore logico alto.

Si osserva che sarebbe possibile mettere in parallelo varie resistenze $R_{ext}$ ognuna in serie a un transistor MOS pilotato dal microcontrollore. In questa maniera, tenendo uno solo dei transistor attivati sarebbe possibile scegliere un valore diverso della resistenza e di conseguenza del tempo $t_{ip}$. Questa soluzione non è stata presa in considerazione all'interno dell'elaborato in quanto avrebbe aggiunto costi e complessità a livello hardware.

Dopo che il microcontrollore ha acquisito temperatura e trasmesso il pacchetto, prima di ritornare in sleep manda un impulso digitale sul pin DONE del Timer, altrimenti il watchdog forzerà un reset del PIC allo scadere del tempo $t_{ip}$. Facciamo riferimento alla seguente figura per un esempio di funzionamento normale e con reset.

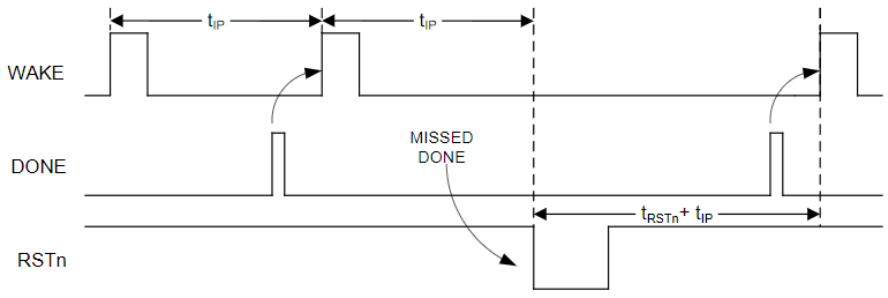



*Figura 5.6 Esempio di funzionamento TPL5010 [15]*

## 5.2 Schema elettrico

Possiamo ora creare uno schema elettrico che tenga conto di quanto detto nel paragrafo precedente, mantenendo la topologia seguente:

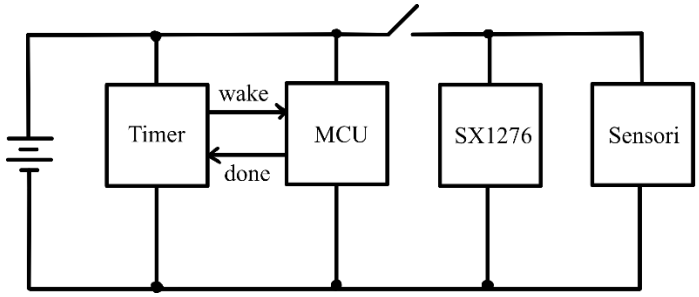



*Figura 5.7 Topologia schematico end-node progettato*

Mostriamo la sezione riguardante il Power Management, comprensiva di Timer esterno e load switch.

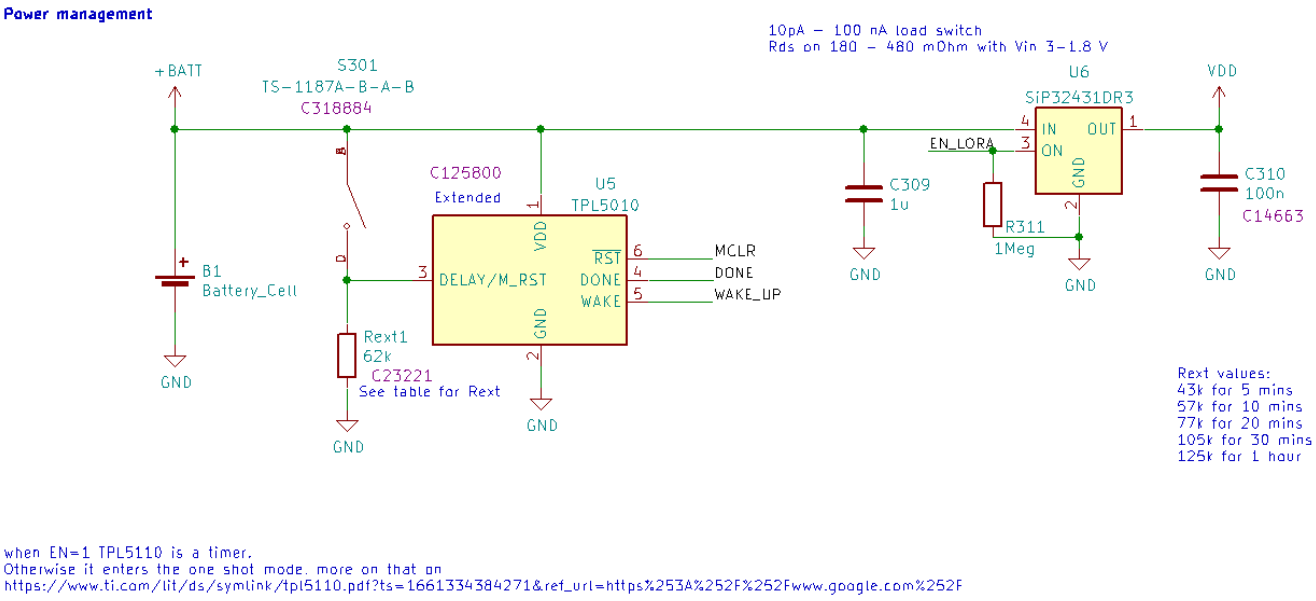



*Figura 5.8 Sezione power-management, Schematico*

Il pulsante permette di bypassare il tempo $t_{ip}$ per il wake up del PIC, mentre la resistenza R311 è in pulldown per evitare che il load switch si chiuda a causa di rumore nelle fasi di avvio in cui il microcontrollore ha i pin in alta impedenza.

Viene ora mostrata la parte relativa al microcontrollore, sensore di temperatura e porta di programmazione.

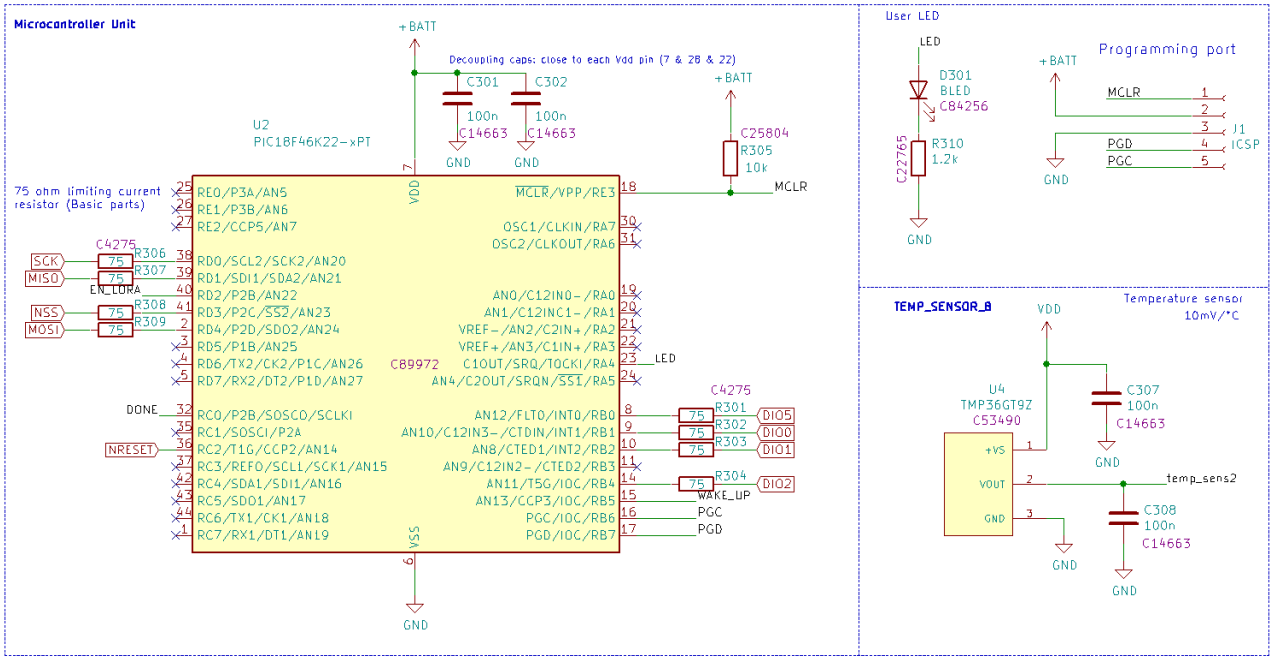



*Figura 5.9 Sezione MCU e sensore, Schematico*

I pin digitali DIO0, DIO1, DIO2, DIO5 sono utilizzati per la gestione dello stack LoRaWAN e sono collegati al transceiver SX1276. I due integrati comunicano tramite il protocollo SPI, che impiega i segnali MISO, MOSI, NSS e NRESET. Ogni linea digitale vede una resistenza di 75Ω in serie per limitare la corrente in caso di conflitti sui livelli logici.

Il sensore di temperatura è simile a quello visto in precedenza: presenta un gradiente di tensione di 10mV per grado centigrado e una tensione a 0°C di 500mV.

Per quanto riguarda la parte a radiofrequenza, la casa produttrice Semtech mette a disposizione vari reference design per l'SX1276. Per un maggiore approfondimento visitare il documento [16].

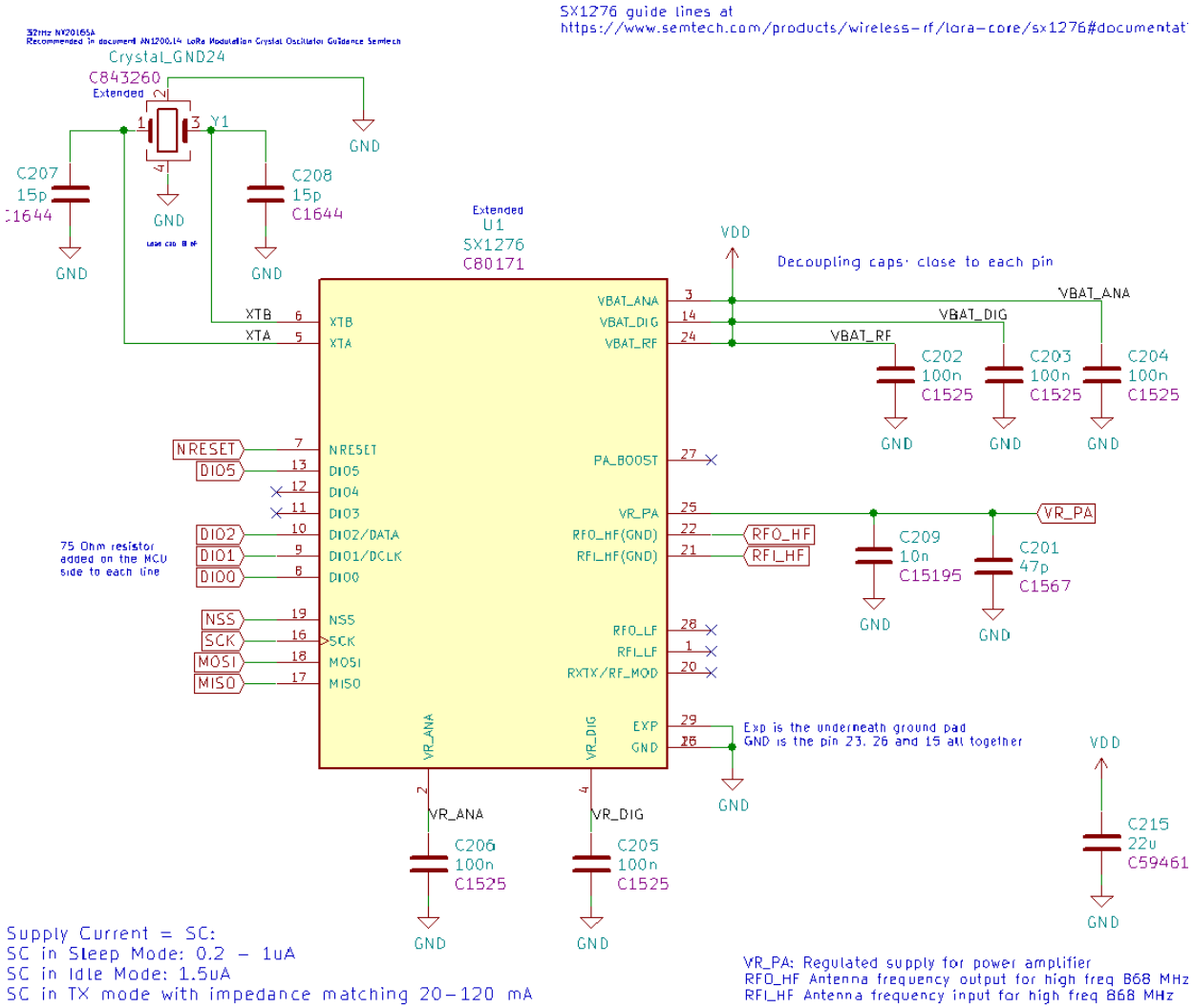



*Figura 5.10 Sezione RF, Schematico*

La parte più critica è rappresentata dal percorso che parte dall'IC e termina al connettore per l'antenna. Per la sua realizzazione è stato deciso di basarsi sul design del progetto open-source LoRa868 Olimex [17] , che implementa un percorso condiviso tra via di trasmissione e via di ricezione. Si aggiunge anche una doppia fila di connettori header 2.54 mm per dare la possibilità di saldare sulla scheda il modulo in caso di mancanza di componenti o strumenti per la saldatura dei componenti (componenti passivi con footprint 0402).

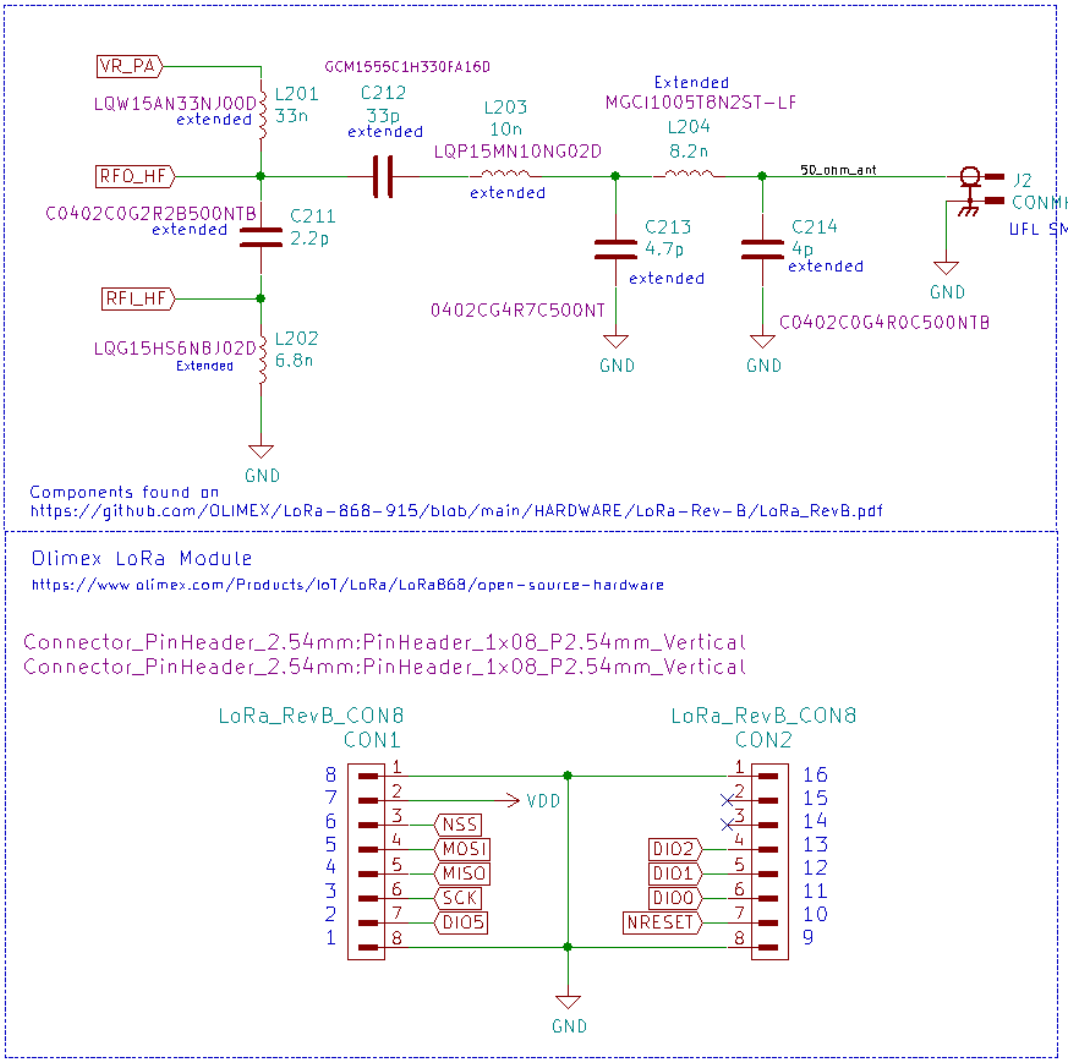



*Figura 5.11 Sezione RF-antenna, Schematico*

### 5.2.1 Stima dei consumi complessivi

Per quanto riguarda la corrente di sleep, si somma il consumo del Timer TPL5010, del PIC18LF46K22 in stato di sleep e del leakage nel load switch:

$$i_{sleep_{typ}} = i_{MCU_{typ}} + i_{TPL5010_{typ}} + i_{leak_{typ}} = 10 + 35 + 0.03 = 45nA$$

$$i_{sleep_{max}} = i_{MCU_{max}} + i_{TPL5010_{max}} + i_{leak_{max}} = 60 + 50 + 300 = 410nA$$

Dove è stato fatto riferimento ai datasheet dei vari componenti per i valori tipici e massimi. [15] [14] [18].

La corrente di trasmissione rimane la stessa utilizzata nel paragrafo 5.1.1, $i_{tx} = 65mA$ e $i_{idle} = 20mA$.

Facciamo riferimento alla figura seguente per la stima della durata complessiva dell'end-node, utilizzando due batterie AAA in serie aventi capacità 1000 mAh.

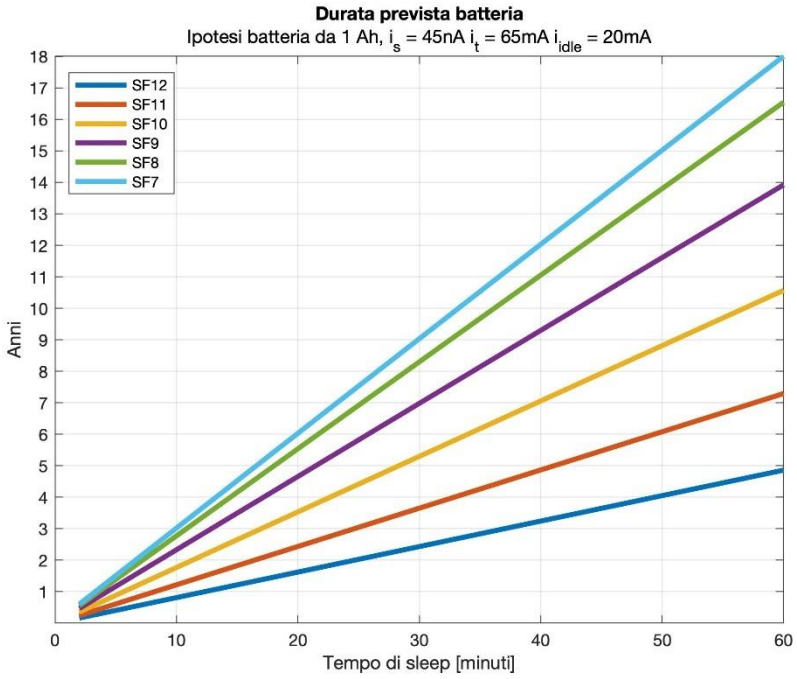



*Figura 5.12 Durata prevista batteria dell'end-node progettato*

Il grafico mostra stime di vita dell'end-node molto promettenti. Inoltre, si può avere il pieno controllo della durata con la possibilità di cambiare sia il tempo di sleep che lo SF tramite downlink.

## 5.3 Printed Circuit Board PCB

L'outline della scheda è definito dalla dimensione del porta-batterie AAA posto nel retro. La scheda è pertanto delle dimensioni 25.6x54mm. Sono stati utilizzati quattro layer, i due interni rispettivamente per il piano di massa e di alimentazione mentre i due esterni per il routing delle piste. Vengono ora allegate le visualizzazioni 3D e dei vari layer per la fabbricazione.

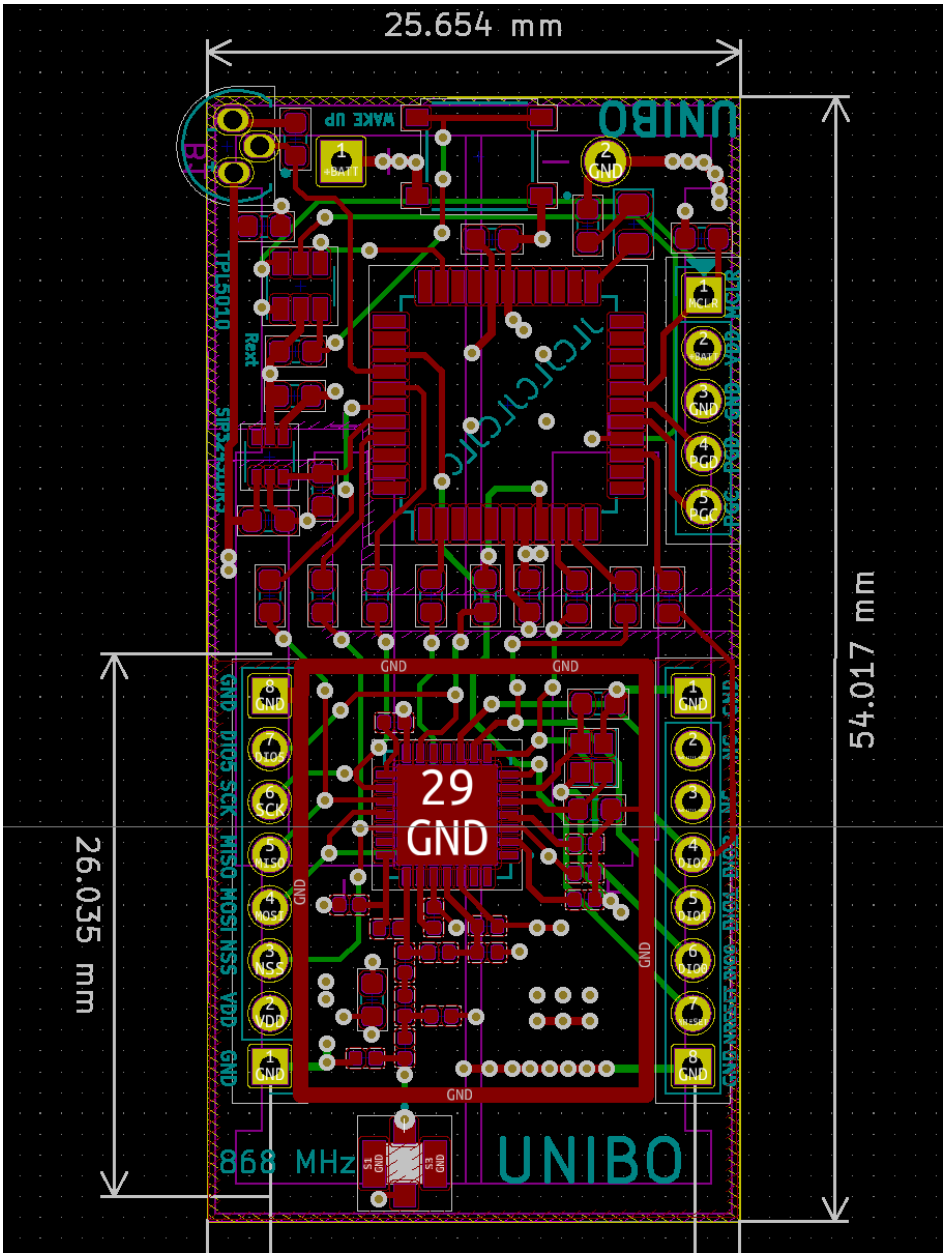



*Figura 5.13 Layout PCB*

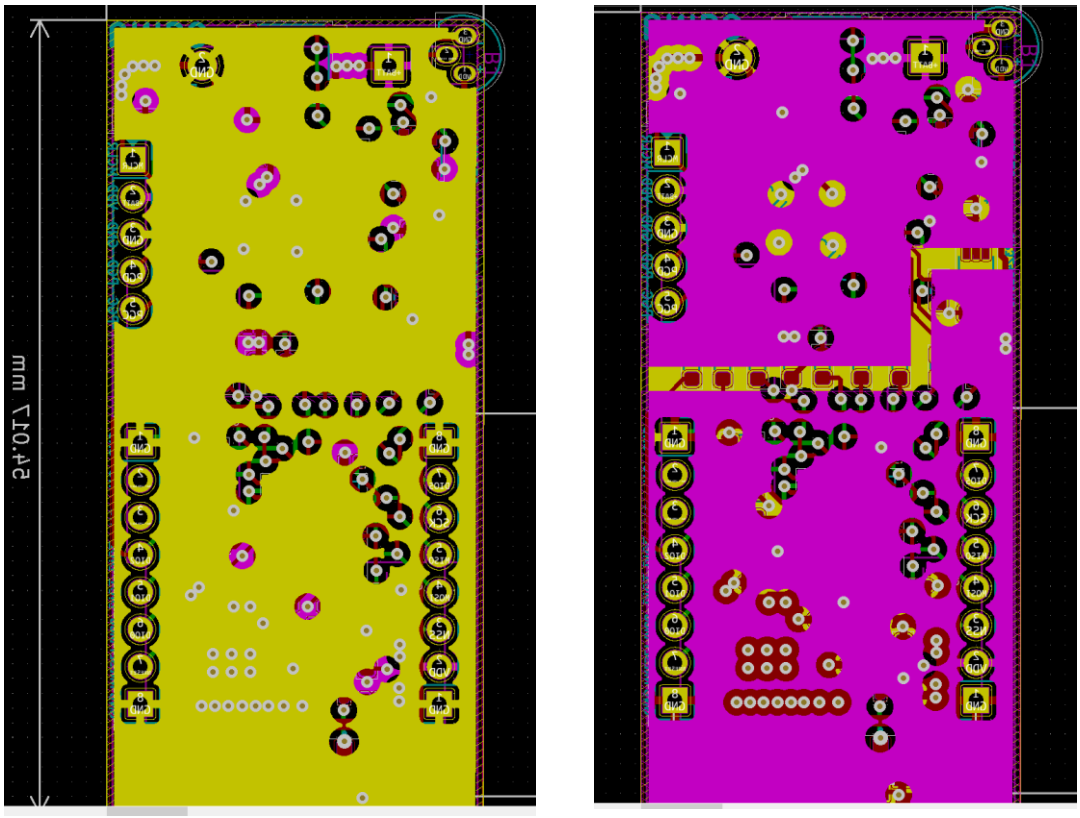



*Figura 5.14 Power planes PCB*

A sinistra il ground plane, a destra il plane di Vbat e Vdd.

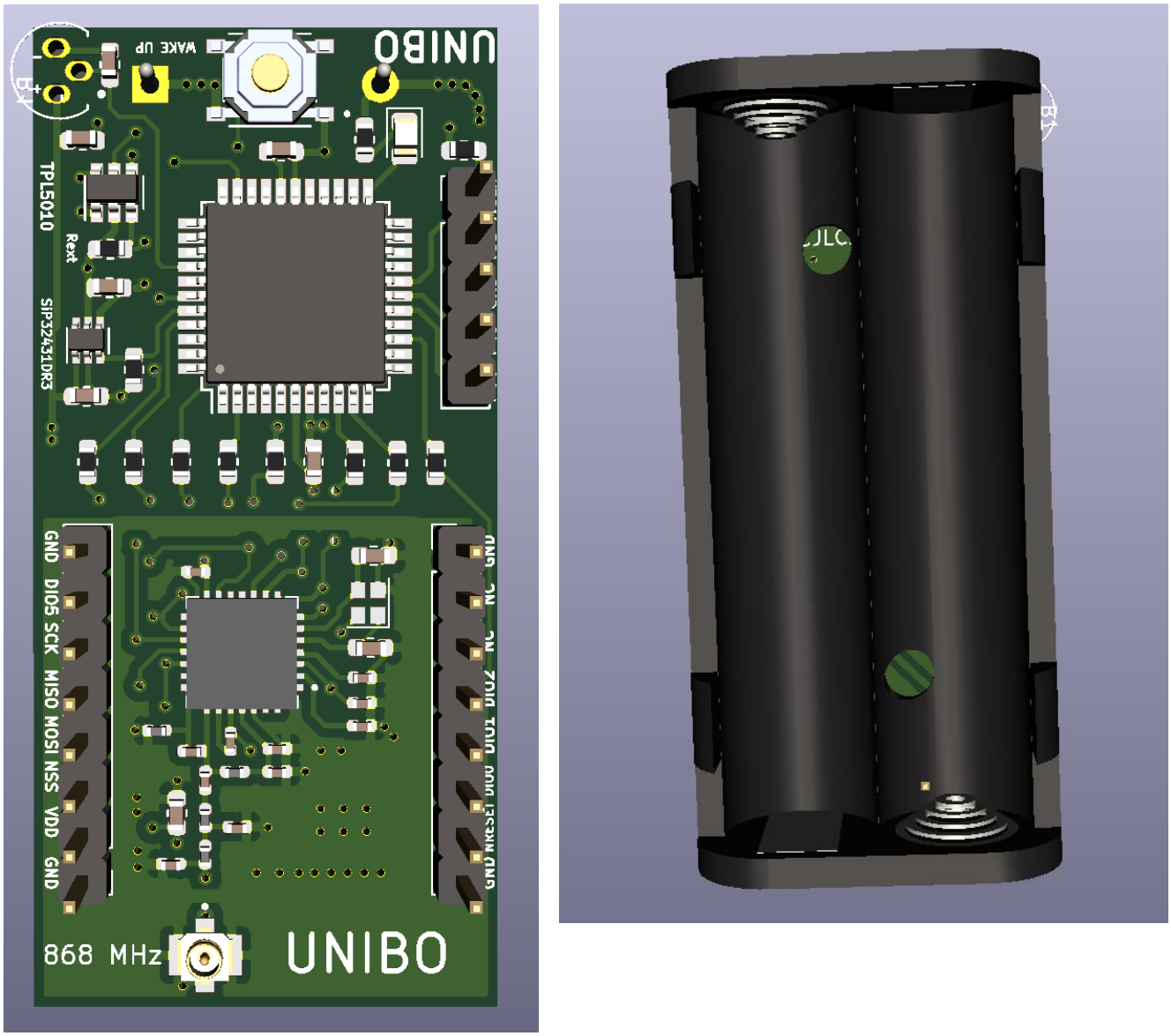



*Figura 5.15 Visualizzazione 3D PCB*

# Conclusioni

In questo elaborato è stata trattata la progettazione di un nodo sensore LoRaWAN a micro-potenza.

Dopo essere partiti da una scheda di sviluppo Microchip per la sperimentazione dell'applicazione si è proceduto a progettare un nodo sensore, avendo cura di ottimizzare i consumi energetici in vari livelli di astrazione, in modo tale da estendere al massimo la durata della batteria.

Grazie ai vantaggi della modulazione LoRaWAN in termini di potenza e distanza di copertura, è possibile trovare applicazioni in ambienti agricoli, urbani ed industriali.

La fase di progetto si è conclusa con il design e sviluppo del PCB, il quale è pronto per il processo di fabbricazione.

Come sviluppi futuri si prevede il montaggio della scheda stessa e la verifica dei consumi stimati all'interno di questo elaborato, per la conferma del tempo di vita.

# Indice delle figure

# Indice delle tabelle

# Bibliografia


[1] «LoRa™ Modulation Basics,» *AN1200.22,* Revision 2, May 2015.

[2] S. Ghoslya, «All about LoRa and LoRaWAN,» [Online]. Available: http://www.sghoslya.com/p/lora-is-chirp-spread-spectrum.html.

[3] Semtech, SX1276/77/78/79 Datasheet.

[4] ERC RECOMMENDATION 70-03 (Tromsø 1997 and subsequent amendments).

[5] «The Things Network, Gateway Mapper,» [Online]. Available: https://ttnmapper.org/heatmap/.

[6] Microchip, «LoRa Mote User's Guide,» pp. 11-33, 2015.

[7] Microchip, «Datasheet RN2483 LoW-Power Long Range LoRa Technology Transceiver Module,» p. 9.

[8] Microchip, «Datasheet MCP9700/9700A MCP9701/9701A».

[9] Microchip, «RN2xx3_LORAWAN_FIRMWARE,» [Online]. Available: https://github.com/search?q=topic:rn2483+org:MicrochipTech.

[10] Microchip, «RN2483 LoRa® TechnologyModule Command ReferenceUser's Guide,» 2015-2018.

[11] Marius Nicolae Microchip Technology. Inc., «AN22337 Low-Power LoRaWAN Class A Application».

[12] Microchip, «LoRaWAN Library Plug-in for MPLAB Code Configurator User's Guide».

[13] Semtech, «SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver,» p. 14.

[14] VISHAY, «SiP32431, SiP32432 10 pA, Ultra Low Leakage and Quiescent Current,Load Switch with Reverse Blocking,» [Online].

[15] Texas Instruments, «TPL5010Nano-PowerSystemTimer With WatchdogFunction,» [Online].

[16] Semtech, «AN1200.19 SX127x Reference Design Overview,» [Online].

[17] Olimex, «LoRa868 open source hardware,» [Online]. Available: https://www.olimex.com/Products/IoT/LoRa/LoRa868/open-source-

hardware.

[18] Microchip, «Datasheet PIC18(L)F2X/4XK22,» *27.2DC Characteristics: Power-Down Current, PIC18(L)F2X/4XK22,* p. 418.

[19] The Things Network. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/.

[20] «RF Wireless Word,» [Online]. Available: https://www.rfwireless-world.com/Terminology/What-is-difference-between-Chip-and-Chirp-in-LoRaWAN.html.